ВИКТОР ЗИБОРОВ

Visual C# 2012

НА ПРИМЕРАХ

БЫСТРАЯ И ЭФФЕКТИВНАЯ РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ ОБРАБОТКИ ТЕКСТОВЫХ, ТАБЛИЧНЫХ И ГРАФИЧЕСКИХ ДАННЫХ

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ MS WORD, MS EXCEL, AUTOCAD И MATLAB, А ТАКЖЕ СОЗДАНИЕ PDF-ФАЙЛОВ

ОБРАБОТКА БАЗ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ ADO.NET И LINQ

СОЗДАНИЕ ИНТЕРАКТИВНЫХ ВЕБ-ПРИЛОЖЕНИЙ

СОЗДАНИЕ ВЕБ-СЛУЖБ И ИХ КЛИЕНТОВ С ПОМОЩЬЮ ТЕХНОЛОГИЙ WEB SERVICE И WCF SERVICE

РАЗРАБОТКА WPF-ПРИЛОЖЕНИЙ

БОЛЕЕ 140 ТИПИЧНЫХ ПРИМЕРОВ

Материалы на www.bhv.ru

Виктор Зиборов

Visual C# 2012

НА ПРИМЕРАХ

Санкт-Петербург
«БХВ-Петербург»
2013

УДК 004.4
ББК 32.973.26-018.2
З-59

Зиборов В. В.

З-59 Visual C# 2012 на примерах. — СПб.: БХВ-Петербург, 2013. — 480 с.: ил.

ISBN 978-5-9775-0888-9

Рассмотрено более 140 типичных примеров, встречающихся в практике реального программирования для платформы .NET Framework в среде Microsoft Visual C# 2012: 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, а также создание PDF-файлов, использование технологий LINQ и ADO.NET при работе с базами данных, разработка интерактивных веб-приложений, создание веб-служб с помощью технологий Web Service и WCF Service, 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. На сайте издательства находятся примеры из книги.

Для программистов

УДК 004.4
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*

Подписано в печать 30.11.12.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 38,7.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-0888-9

© Зиборов В. В., 2013
© Оформление, издательство "БХВ-Петербург", 2013

Оглавление

Предисловие **9**

Введение. Что такое "хороший стиль программирования"? **13**

Глава 1. Простейшие программы с экранной формой и элементами управления **15**

Пример 1. Форма, кнопка, метка и диалоговое окно 15
Пример 2. Событие *MouseHover* 20
Пример 3. Выбор нужной даты 24
Пример 4. Ввод данных через текстовое поле *TextBox* с проверкой типа методом *TryParse* 27
Пример 5. Ввод пароля в текстовое поле и изменение шрифта 30
Пример 6. Управление стилем шрифта с помощью элемента управления *CheckBox* 32
Пример 7. Побитовый оператор "исключающее ИЛИ" 34
Пример 8. Вкладки *TabControl* и переключатели *RadioButton* 36
Пример 9. Свойство *Visible* и всплывающая подсказка *ToolTip* в стиле *Balloon* 39
Пример 10. Калькулятор на основе комбинированного списка *ComboBox* 41
Пример 11. Вывод греческих букв, символов математических операторов. Кодовая таблица *Unicode* 45

Глава 2. Программирование консольных приложений **49**

Пример 12. Ввод и вывод в консольном приложении 49
Пример 13. Вывод на консоль таблицы чисел с помощью форматирования строк 52
Пример 14. Вызов метода *MessageBox.Show* в консольном приложении. Формат даты и времени 53
Пример 15. Вызов функций Visual Basic из программы C# 55
Пример 16. Программирование интервалов с помощью оператора *else if* 57
Пример 17. Замечательной структурой данных является словарь *Dictionary* 60

Глава 3. Инициирование и обработка событий мыши и клавиатуры **63**

Пример 18. Координаты курсора мыши относительно экрана и элемента управления 63
Пример 19. Создание элемента управления *Button* "программным" способом и подключение события для него 65

Пример 20. Обработка нескольких событий одной процедурой ... 68
Пример 21. Калькулятор ... 69
Пример 22. Ссылка на другие ресурсы *LinkLabel* ... 74
Пример 23. Обработка событий клавиатуры ... 76
Пример 24. Разрешаем вводить в текстовое поле только цифры ... 79
Пример 25. Разрешаем вводить в текстовое поле цифры, а также разделитель целой и дробной части числа ... 80
Пример 26. Программно вызываем событие "щелчок на кнопке" ... 83

Глава 4. Чтение, запись текстовых и бинарных файлов, текстовый редактор ... 85
Пример 27. Чтение/запись текстового файла в кодировке *Unicode*. Обработка исключений *try...catch* ... 85
Пример 28. Чтение/запись текстового файла в кодировке Windows 1251 ... 89
Пример 29. Простой текстовый редактор. Открытие и сохранение файла. Событие формы *Closing* ... 91
Пример 30. Программа тестирования знаний студента по какому-либо предмету ... 96
Пример 31. Простой RTF-редактор ... 102
Пример 32. Программа ввода каталога координат (числовых данных) из текстового файла ... 106
Пример 33. Печать текстового документа ... 110
Пример 34. Чтение/запись бинарных файлов с использованием потока данных ... 114

Глава 5. Редактирование графических данных ... 119
Пример 35. Простейший вывод отображения графического файла в форму ... 119
Пример 36. Использование элемента *PictureBox* для отображения растрового файла с возможностью прокрутки ... 122
Пример 37. Рисование в форме графических примитивов (фигур) ... 124
Пример 38. Выбор цвета с использованием *ListBox* ... 127
Пример 39. Экранная форма с треугольником прозрачности ... 130
Пример 40. Печать графических примитивов ... 132
Пример 41. Печать BMP-файла ... 133
Пример 42. Создание JPG-файла "на лету" и вывод его отображения в форму ... 134
Пример 43. Смена выведенного изображения с помощью обновления формы ... 136
Пример 44. Рисование в форме указателем мыши ... 139
Пример 45. Управление сплайном Безье ... 141
Пример 46. Построение графика методами класса *Graphics* ... 145

Глава 6. Управление буфером обмена с данными в текстовом и графическом форматах ... 151
Пример 47. Буфер обмена с данными в текстовом формате ... 151
Пример 48. Элемент управления *PictureBox*. Буфер обмена с растровыми данными ... 153
Пример 49. Имитация нажатия комбинации клавиш <Alt>+<PrintScreen> ... 156
Пример 50. Запись содержимого буфера обмена в BMP-файл ... 157
Пример 51. Использование таймера *Timer* ... 159
Пример 52. Запись в файлы текущих состояний экрана каждые пять секунд ... 161

Глава 7. Ввод и вывод табличных данных. Решение системы уравнений 163
Пример 53. Формирование таблицы. Функция *String.Format*..... 163
Пример 54. Форматирование *Double*-переменных в виде таблицы.
Вывод таблицы на печать. Поток *StringReader*..... 166
Пример 55. Вывод таблицы в Internet Explorer 169
Пример 56. Формирование таблицы с помощью элемента управления *DataGridView* 172
Пример 57. Отображение хэш-таблицы с помощью элемента *DataGridView* 174
Пример 58. Табличный ввод данных. *DataGridView*. *DataTable*. *DataSet*.
Инструмент для создания файла *XML*..... 177
Пример 59. Решение системы линейных уравнений. Ввод коэффициентов
через *DataGridView*..... 181
Пример 60. Организация связанных таблиц..... 186
Пример 61. Построение графика по табличным данным с использованием
элемента *Chart*..... 190

Глава 8. Элемент управления *WebBrowser* 195
Пример 62. Отображение HTML-таблиц в элементе *WebBrowser* 195
Пример 63. Отображение Flash-файлов 197
Пример 64. Отображение веб-страницы и ее HTML-кода 198
Пример 65. Программное заполнение веб-формы..... 201
Пример 66. Синтаксический разбор веб-страницы без использования
элемента *WebBrowser* 205

**Глава 9. Использование функций MS Word, MS Excel, AutoCAD
и MATLAB, а также создание PDF-файла..... 209**
Пример 67. Проверка правописания в текстовом поле с помощью обращения
к MS Word 209
Пример 68. Вывод таблицы средствами MS Word 213
Пример 69. Обращение к функциям MS Excel из программы на Visual C# 12 216
Пример 70. Использование финансовой функции MS Excel 218
Пример 71. Решение системы уравнений с помощью функций MS Excel..... 221
Пример 72. Построение диаграммы средствами MS Excel..... 224
Пример 73. Управление функциями AutoCAD из программы на Visual C# 12..... 227
Пример 74. Вызов MATLAB из вашей программы на Visual C# 12 230
Пример 75. Решение системы уравнений путем обращения к MATLAB 232
Пример 76. Создание PDF-файла "на лету" с возможностью вывода кириллицы 234
Пример 77. Вывод таблицы в PDF-документ..... 238
Пример 78. Вывод графических данных в PDF-документ 243

Глава 10. Обработка баз данных с использованием технологии ADO.NET ... 249
Пример 79. Создание базы данных SQL Server..... 249
Пример 80. Отображение таблицы базы данных SQL Server на консоли..... 251
Пример 81. Редактирование таблицы базы данных MS Access в среде Visual Studio
без написания программного кода 253
Создание базы данных в среде MS Access..... 253
Открытие базы данных Access в среде Visual Studio 254

Пример 82. Чтение всех записей из таблицы БД MS Access на консоль с помощью объектов классов *Command* и *DataReader* ... 256
Пример 83. Создание базы данных MS Access в программном коде ... 258
Пример 84. Запись структуры таблицы в пустую базу данных MS Access. Программная реализация подключения к БД ... 261
Пример 85. Добавление записей в таблицу базы данных MS Access ... 263
Пример 86. Чтение всех записей из таблицы базы данных с помощью объектов классов *Command*, *DataReader* и элемента управления *DataGridView* ... 265
Пример 87. Чтение данных из БД в сетку данных *DataGridView* с использованием объектов классов *Command*, *Adapter* и *DataSet* ... 267
Пример 88. Обновление записей в таблице базы данных MS Access ... 269
Пример 89. Удаление записей из таблицы базы данных с использованием SQL-запроса и объекта класса *Command* ... 273

Глава 11. Использование технологии LINQ ... 275
Пример 90. LINQ-запрос к массиву данных ... 275
Пример 91. Запрос к коллекции (списку) данных методами LINQ ... 278
Пример 92. Группировка элементов списка с помощью LINQ-запроса ... 283
Пример 93. Группировка словаря данных *Dictionary* с помощью LINQ-запроса ... 287
Пример 94. Создание XML-документа методами классов пространства имен *System.Xml.Linq* ... 290
Пример 95. Извлечение значения элемента из XML-документа посредством LINQ-запроса ... 292
Пример 96. Поиск строк (записей) в XML-данных с помощью LINQ-запроса ... 297
Пример 97. Получение производных XML-данных от XML-источника ... 300
Пример 98. LINQ-запрос к набору данных *DataSet* ... 303

Глава 12. Другие задачи, решаемые с помощью *Windows Application* ... 307
Пример 99. Проверка вводимых данных с помощью регулярных выражений ... 307
Пример 100. Управление прозрачностью формы ... 310
Пример 101. Время по Гринвичу в полупрозрачной форме ... 312
Пример 102. Ссылка на процесс, работающий в фоновом режиме, в форме значка в области уведомлений ... 315
Пример 103. Нестандартная форма. Перемещение формы мышью ... 317
Пример 104. Воспроизведение звуков операционной системы ... 319
Пример 105. Проигрыватель Windows Media Player 12 ... 321
Пример 106. Воспроизведение только звуковых файлов ... 325
Пример 107. Программирование контекстной справки. Стандартные кнопки в форме ... 327

Глава 13. Программирование простейших веб-ориентированных приложений на Visual C# 12 ... 331
Создание веб-страницы на языке HTML. Интернет-технологии ... 331
Веб-хостинг на платформах UNIX и Windows ... 332
Клиент-серверное взаимодействие на основе технологии ASP.NET ... 333
Отладка активного веб-приложения ... 334
Пример 108. Создание простейшей активной веб-страницы на Visual C# 12 ... 334

Пример 109. Проверка введенных пользователем числовых данных с помощью валидаторов ... 337
Пример 110. Проверка достоверности ввода имени, адреса e-mail, URL-адреса и пароля с помощью валидаторов ... 340
Пример 111. Регистрация и аутентификация пользователя с помощью базы данных Access ... 345
Пример 112. Таблица с переменным числом ячеек, управляемая двумя раскрывающимися списками ... 354
Пример 113. Организация раскрывающегося меню гиперссылок с помощью *DropDownList* ... 356
Пример 114. Передача данных между веб-страницами через параметры гиперссылки ... 359
Пример 115. Передача данных HTML-формы на ASPX-страницу методами класса *Request* ... 362
Пример 116. Передача значений элементов управления на другую веб-страницу с помощью объекта *PreviousPage* ... 366
Пример 117. Отображение табличных данных в веб-форме с помощью элемента управления *GridView* ... 369
Пример 118. Отображение хэш-таблицы в веб-форме ... 370

Глава 14. Типичные веб-ориентированные приложения ASP.NET на Visual C# 12 ... 375
Пример 119. Чтение/запись текстового файла веб-приложением ... 375
Пример 120. Программирование счетчика посещений сайта с использованием базы данных и объекта *Session* ... 379
Пример 121. Чтение/запись cookie-файлов ... 384
Пример 122. Вывод изображения в веб-форму ... 388
Пример 123. Формирование изображения методами класса *Graphics* и вывод его в веб-форму ... 391
Пример 124. Гостевая книга ... 394
Пример 125. Отображение времени в веб-форме с использованием технологии AJAX ... 398

Глава 15. Создание веб-служб и их клиентов ... 401
О веб-службах ... 401
Пример 126. Клиентское веб-приложение, потребляющее сервис веб-службы "Прогноз погоды" ... 402
Пример 127. Клиентское Windows-приложение, использующее ту же веб-службу "Прогноз погоды" ... 409
Пример 128. Создание простейшей веб-службы ... 411
Пример 129. Создание Windows-приложения, потребителя сервиса веб-службы ... 414
Пример 130. Создание веб-службы "Торговая рекомендация на рынке Forex" ... 417
Пример 131. Клиентское приложение, потребляющее сервис веб-службы "Торговая рекомендация на рынке Forex" ... 420
Пример 132. Клиентское веб-приложение, потребляющее сервис веб-службы "Морфер" ... 421
Пример 133. Получение веб-приложением данных от веб-службы Центрального банка РФ ... 424

Пример 134. Получение Windows-приложением данных от веб-службы Национального банка Республики Беларусь 426
Пример 135. Создание веб-службы на основе WCF (WCF Service) 428
Пример 136. Создание Windows-приложения, потребителя сервиса WCF-службы 430

Глава 16. Использование технологии WPF 433
Что может нам дать WPF? 433
Пример 137. Создание простейшего WPF-приложения. Компоновка элементов управления с помощью сетки *Grid* 434
Пример 138. Использование одного из эффектов анимации 439
Пример 139. Эффект постепенной замены (прорисовки) одного изображения другим 442
Пример 140. Закрашивание области текста горизонтальным линейным градиентом 445
Пример 141. Программирование WPF-проигрывателя. Компоновка элементов управления с помощью панели *StackPanel* 446
Пример 142. Наложение текста на видео 450
Пример 143. Переходы в WPF-приложениях 453

Приложение. Содержание электронного архива с примерами из книги 457

Предметный указатель 471

Предисловие

Система разработки программного обеспечения Microsoft Visual Studio 12 является мировым лидером на рынке программного обеспечения. Используя эту систему, можно "малой кровью" и очень быстро написать, почти сконструировать, *как в детском конструкторе*, довольно-таки функционально сложные как настольные приложения (в виде EXE-файлов), так и приложения, исполняемые в браузере. В центре системы Visual Studio 12 находится среда программирования (платформа) .NET Framework — это встроенный компонент Windows, который поддерживает создание и выполнение приложений нового поколения и веб-служб. Основными компонентами .NET Framework являются общеязыковая среда выполнения (CLR) и *библиотека классов* .NET Framework, включающая ADO.NET, ASP.NET, Windows Forms и Windows Presentation Foundation (WPF). Платформа .NET Framework предоставляет среду управляемого выполнения, возможности упрощения разработки и развертывания, а также возможности интеграции со многими языками программирования.

Среда разработки программного обеспечения Visual Studio 12 включает в себя языки программирования Visual Basic, Visual C#, Visual C++ и Visual F#. Используя эти языки программирования, можно подключаться к библиотекам классов и тем самым *иметь все преимущества* ускоренной разработки приложений. В этой среде программный код пишется проще, легче читается, а конечный продукт получается очень быстро.

Существенный положительный эффект достигается при групповой разработке какого-либо проекта. Используя Visual Studio, над одним проектом могут работать программисты на C#, на Visual Basic и на C++, при этом среда .NET обеспечит совместимость программных частей, написанных на разных языках.

Цель книги — популяризация программирования. 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. Приведено несколько подходов

к выводу диаграмм (графиков). Рассмотрены манипуляции табличными данными, в том числе организация связанных таблиц. Показан принцип использования элемента управления `WebBrowser` для отображения различных данных, а также для программного заполнения веб-форм. Обсуждены примеры программирования с применением функций (методов) объектных библиотек систем MS Excel, MS Word, AutoCAD и MATLAB. Представлено несколько выразительных примеров создания PDF-файла. 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. Представлено много различных авторских оригинальных решений задач программирования, которых читатель не сможет найти в Интернете. Приведены примеры программирования *веб-ориентированных приложений,* а также использования и разработки *веб-служб* (как Web Service, так и WCF Service). Новейшая технология WPF представлена несколькими показательными 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 ранее 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 140 примеров) и содержит эта книга. Автор пытался выделить *наиболее типичные*, актуальные задачи и решить их, с одной стороны, максимально эффективно, а с другой стороны, *кратко* и *выразительно*.

> ***Внимание!***
>
> Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Очень удобно читать книгу и пытаться запускать, трассировать отладчиком системы программирования, изменять, исследовать те или иные примеры из скачанного архива.

Самая серьезная проблема в программировании больших, сложных проектов — это *запутанность текстов*. Из-за нее в программах появляются ошибки, нестыковки и проч. Как следствие — страдает производительность процесса создания программ *и их сопровождение*. Решение этой проблемы состоит в *структуризации* программ. Переход к объектно-ориентированному программированию связан 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*, отказ от использования операторов перехода `goto` и проч. Кроме того, современные системы программирования разрешают в названиях переменных, методов, свойств, событий, классов, объектов *применять русские буквы (символы кириллицы)*. Между тем современные программисты, как правило, *не используют* такую 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).

Эта 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 *MS C#*. Как пользоваться книгой? Для начинающих эффективнее всего — это последовательно решать примеры в порядке их представления в книге, поскольку примеры расположены *от простых к более сложным*. И тем самым постепенно совершенствовать свои навыки разработки на изучаемом языке программирования. Программистам среднего уровня можно посоветовать искать *выборочно* именно те примеры, которые соответствуют вопросам, возникшим у них при программировании текущих задач. Если вы программируете на Visual Basic или C++/CLI, то также можете получить положительный эффект от чтения этой книги, поскольку и C#, и C++/CLI, и Visual Basic "питаются" *все той же средой* .NET, следовательно, названия соответствующих классов, методов, свойств и событий являются одинаковыми, а программные коды различных языков отличаются всего лишь синтаксисом.

Надеюсь, что изучая эту книгу, читатель получит одновременно интеллектуальное *удовольствие* и *пользу* от примененения полученных знаний в своей работе и творчестве. Свои впечатления о книге присылайте по адресу **ziborov@ukr.net**, я с удовольствием с ними ознакомлюсь.

ВВЕДЕНИЕ

Что такое "хороший стиль программирования"?

Отвечая на этот вопрос одной строчкой, можно сказать, что хороший стиль программирования подразумевает наличие *структуры* у написанной программы. Соответственно, если программа содержит много строк программного кода в одной процедуре, не разделена на смысловые блоки, напоминает "спагетти", т. е. имеет множество переходов управления, содержит мало комментариев и, как следствие, становится запутанной, громоздкой, *трудно модифицируемой и управляемой*, то это и есть "плохой стиль программирования".

Отсюда появилось понятие *структурное программирование*. Появилось оно еще в 60-е годы прошлого века, когда весь мир переживал кризис разработки программного обеспечения. То есть создание больших автоматизированных систем затягивалось, программисты не укладывались в сроки, а в готовых программах обнаруживалось множество ошибок. С этими двумя проблемами (низкая производительность разработки программ и ошибки в них) программисты борются во все времена. В 1968 году голландец Дийкстра назвал (предположил) причину — в больших программах часто нет четкой математической структуры, они *неоправданно сложны*. Причем эту "неоправданную" сложность вносит команда `goto`. Программу с множеством `goto` называют "спагетти". Для сравнения, представьте, что вы читаете роман: сначала две страницы назад, потом четыре страницы вперед и т. д. Если программа содержит много операторов `goto`, то человеку проследить передачу управления (последовательность управления) весьма затруднительно (это только компьютеру все равно). Вместо операторов `goto` Дийкстра предложил использовать всего три типа управляющих структур. Первый тип — это простая последовательность *(следование)*, когда операторы выполняются друг за другом слева направо и сверху вниз. Второй тип — это альтернатива *(ветвление)*, выбор по условию (`if – else`), множественный выбор (`switch – case`). И третий тип управляющей структуры — это *цикл*, т. е. повторение одного или нескольких операторов до выполнения условия выхода из цикла.

Так вот, основная идея Дийкстры заключается в том, что, используя эти три структуры (отсюда и название "структурное программирование"), можно обходиться без операторов `goto`. Он утверждал, что отсутствие наглядности — это и есть основной источник ошибок. Сначала программисты лишь посмеивались над идеями Дийкстры, тем более, что он был всего лишь теоретиком программирования. Однако смеяться над чем-то новым — весьма опрометчиво. Один русский прославленный генерал, когда ему впервые показали только что изобретенный пулемет, снисходительно похлопал изобретателя по

плечу и сказал, что если бы в реальном бою нужно было убить одного человека несколько раз, то его изобретение было бы очень кстати. Так что не следует спешить высмеивать новое, непонятное и непривычное.

Так случилось и с идеями Дийкстры — фирма IBM весьма успешно применила принципы структурного программирования для создания базы данных газеты "Нью-Йорк таймс". И это стало весомым аргументом в пользу такого подхода. Со временем оператор `goto` программисты стали называть "позорным" и использовать его лишь в очень крайних случаях. И с тех пор концепция структурного программирования оказывает заметное влияние на теорию и практику программного обеспечения всех рангов.

Сегодня структурное программирование, как альтернатива "интуитивному", "рефлекторному" подходу, — это методология разработки программного обеспечения, в основе которой лежит не только три типа управляющих структур, но и многое другое, что помогает представлять программу в виде иерархической структуры и тем самым "спрятать сложность". Все тело программы стараются делить на логически целостные вычислительные блоки, которые оформляются в виде подпрограмм, функций и классов. Эти вычислительные блоки и формируют собой иерархическую структуру, причем наименее значимые "подробности" программы стараются спрятать в самые дальние ветви иерархии, чтобы они не мешали пониманию основной логики программы. Пряча те или иные смысловые блоки подальше от глаз, мы тем самым скрываем сложность программы для того, чтобы ее (программы) логика была максимально понятой. Причем "понятность", читабельность нужно обеспечить не только автору программы, но и другим разработчикам. Вполне справедливо утверждение, что любую, самую сложную, программу можно сделать доступной для понимания "широким слоям разработчиков", скрывая сложности в смысловых блоках.

Каждый смысловой вычислительный блок должен содержать как можно больше комментариев, а в его начале необходима преамбула, в которой указано, что этот блок делает, и какие технологии и приемы при этом используются. Концепция объектно-ориентированного программирования — это одна из ступеней к большей структуризации программ. Современные системы программирования стараются обеспечить максимальную читабельность наших программ — обратите, кстати, внимание на отступы и разноцветность программного кода в окнах системы, это придает программам бóльшую выразительность и понятность.

Следует всегда учитывать простой факт — суть программы, которую вы писали неделю назад, очень быстро забывается. А если вы посмотрите на программу, написанную вами же месяц назад? Вы удивитесь, но у вас создастся впечатление, что это чужая программа, написанная никак не вами. Не стоит удивляться, это особенность человеческого организма — отбрасывать все на его (организма) взгляд ненужное!

В ходе написания программного кода следует большое внимание уделять как можно более точным названиям объектов, переменных, процедур. Тогда смысл написанных операторов будет более "прозрачен". Автор в своих примерах старался присваивать соответствующие названия на русском языке, что обеспечило еще большую структурированность написанных программ. Вообще, следует помнить, что антонимом слова "структура" будет — "хаос".

Таким образом, бóльшая структуризация программы — это путь для повышения производительности разработки, уменьшения количества ошибок, и это есть "хороший стиль программирования".

ГЛАВА 1

Простейшие программы с экранной формой и элементами управления

Пример 1. Форма, кнопка, метка и диалоговое окно

После инсталляции системы программирования Visual Studio 2012, или кратко VS 12, включающей в себя Visual C# 12, загрузочный модуль системы devenv.exe будет, скорее всего, расположен в папке: C:\Program Files\Microsoft Visual Studio 12.0\Common7\IDE.

После запуска системы мы увидим начальный пользовательский интерфейс, показанный на рис. 1.1.

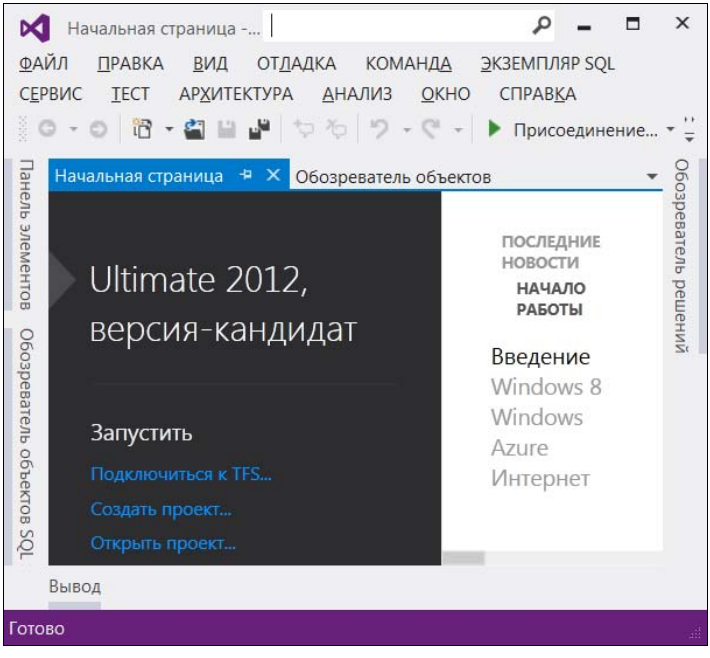


Рис. 1.1. Фрагмент стартовой страницы системы Visual Studio 12

Чтобы запрограммировать какую-либо задачу, необходимо в пункте меню **File** выполнить команду **New Project**. В левой колонке открывшегося окна **New Project** находится список установленных 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. Выберем язык Visual C#, а затем в области шаблонов (в средней колонке) щелкнем на шаблоне (**Templates**) **Windows Forms Application**. Теперь введем имя проекта (**Name**): `First` и щелкнем на кнопке **OK**. В результате увидим окно, представленное на рис. 1.2.

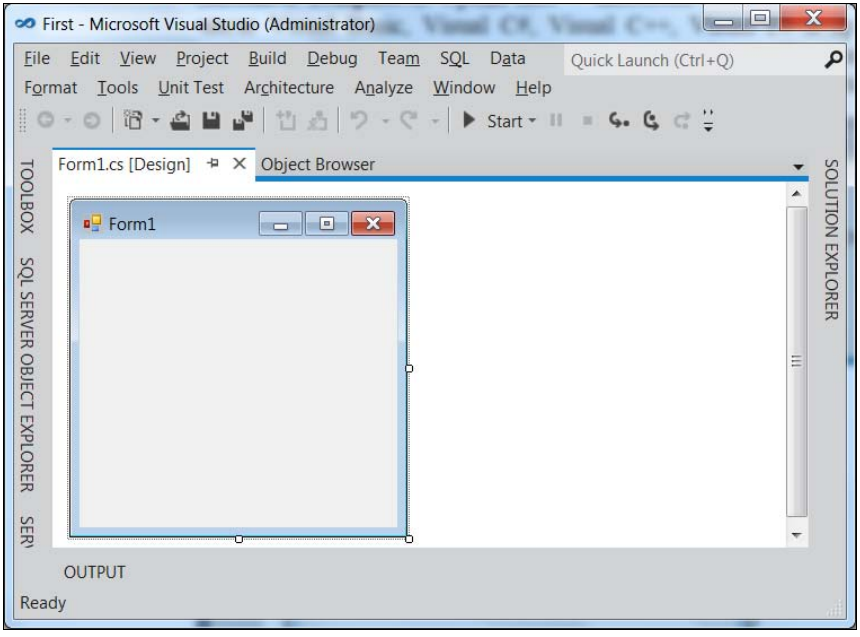


Рис. 1.2. Окно для проектирования пользовательского интерфейса

В этом окне изображена *экранная форма* — **Form1**, в которой программисты располагают различные компоненты графического интерфейса пользователя или, как их иначе называют, *элементы управления*: поля для ввода текста **TextBox**, командные кнопки **Button**, строчки текста в форме (метки **Label**, которые не могут быть отредактированы пользователем) и проч.

О визуальном программировании

Обратите внимание — здесь используется самое современное так называемое *визуальное программирование*, предполагающее простое перетаскивание мышью в форму тех или иных элементов управления из панели **Toolbox**, где все они и расположены. Таким образом, сводится к минимуму непосредственное написание программного кода.

В вашей первой программе мы создадим экранную форму с надписью, например, "Microsoft Visual C#". Кроме нее в форме будет расположена командная кнопка с надписью "Нажми меня". При нажатии кнопки откроется диалоговое окно с сообщением "Всем привет!"

Пояснение

Написать такую программку — вопрос 2–3 минут. Но сначала я хотел бы буквально парой слов пояснить основной принцип современного объектно-ориентированного подхода к программированию. А заключается он в том, что в программе все, что может быть определено как имя существительное, называют *объектом*. Так в нашей программе имеется четыре объекта: форма **Form**, надпись на форме **Label**, кнопка **Button** и диалоговое окно **MessageBox** с текстом "Всем привет!" — окно с приветом☺.

Теперь давайте добавим в форму названные элементы управления. Для этого нам понадобится панель элементов управления **Toolbox**. Если в данный момент вы панель элементов не видите, то ее можно добавить, например, с помощью комбинации клавиш <Ctrl>+<Alt>+<x> или выполнив команду меню **View | Toolbox**. Итак, добавьте метку **Label** и кнопку **Button** в форму, щелкнув двойным щелчком на этих элементах в панели **Toolbox**. А затем расположите их примерно так, как показано на рис. 1.3.

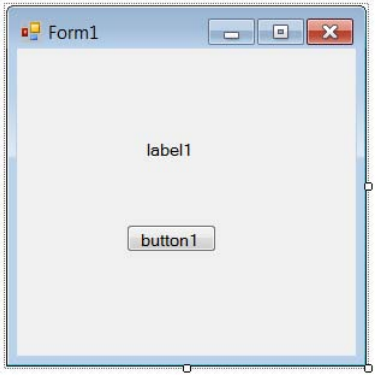


Рис. 1.3. Форма первого проекта

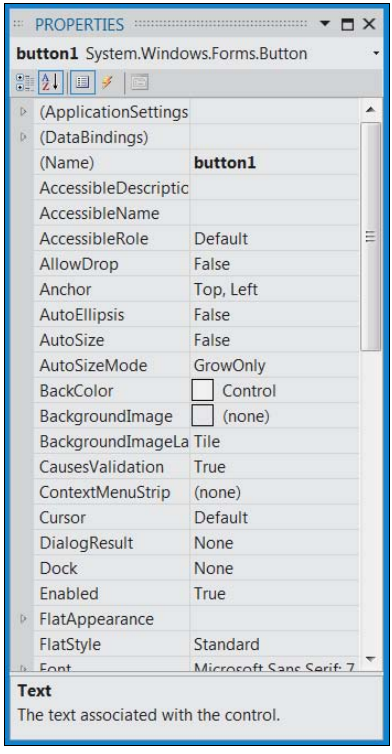


Рис. 1.4. Свойства кнопки `button1`

Любой такой объект можно создать самостоятельно, а можно воспользоваться готовым. В данной задаче мы пользуемся готовыми визуальными объектами, которые можно также просто перетаскивать мышью из панели элементов управления **Toolbox** в окно формы.

При этом нам нужно знать, что каждый объект имеет *свойства* (**Properties**). Например, свойствами кнопки являются (рис. 1.4): имя кнопки (**Name**) — в нашем случае **Button1**, надпись на кнопке (**Text**), расположение кнопки (**Location**) в системе координат формы **X**, **Y**, размер кнопки (**Size**) и т. д. Свойств много, все их можно увидеть в панели

свойств **Properties** (см. рис. 1.4), которая откроется, если щелкнуть правой кнопкой мыши в пределах формы и выбрать в контекстном меню команду **Properties**.

Указывая мышью на все другие элементы управления в форме, можно просмотреть и их свойства: формы **Form1** и надписи в форме — метки **Label1**.

Вернемся к нашей задаче. Для объекта **Label1** выберем свойство **Text** и напишем у этого поля: `Microsoft Visual C# 12` (вместо текста `label1`). Для объекта `button1` также в свойстве `Text` напишем `Нажми меня`.

Итак, что объекты имеют свойства, мы уже поняли. Следует также 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. Управляют тем или иным событием посредством написания процедуры обработки события в программном коде. Для этого вначале нужно получить *"пустой" обработчик события* (т. е. заготовку записи программного кода). 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 перед нами откроется вкладка программного кода **Form1.cs** (рис. 1.5).

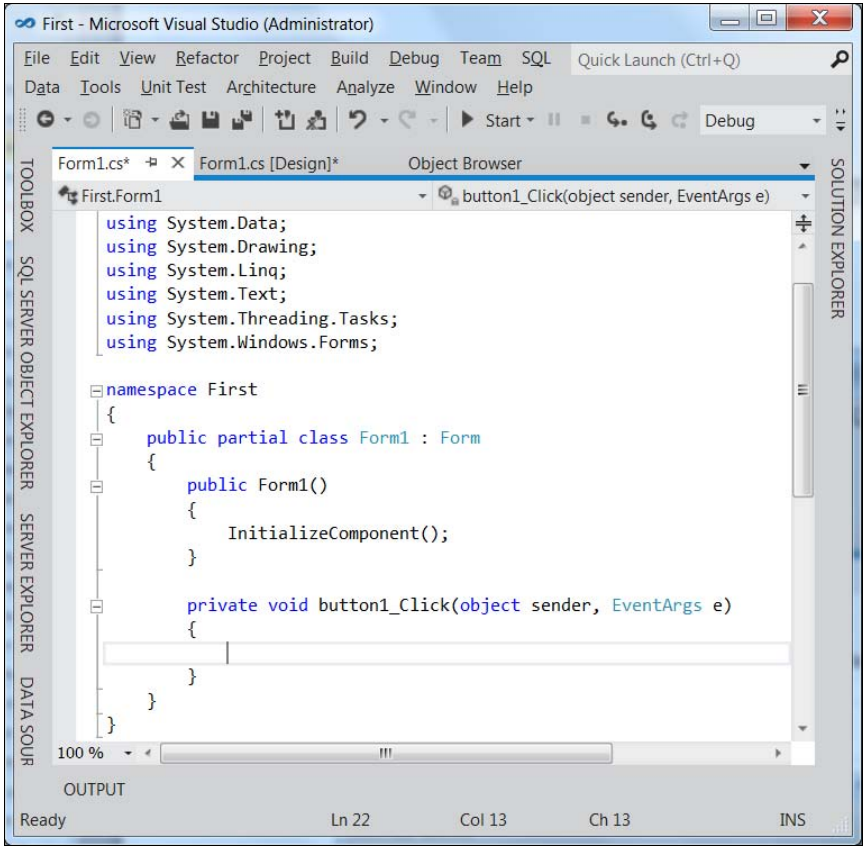


Рис. 1.5. Вкладка программного кода

На вкладке **Form1.cs** мы увидим, что управляющая среда Visual C# 12 сгенерировала четыре строки программного кода. Читателю не стоит ужасаться, глядя на этот непонятный код — постепенно многое прояснится.

Итак, на рис. 1.5 показан программный код, описывающий упомянутый нами "пустой" обработчик события `Button1_Click`. Здесь между фигурными скобками, следующими за оператором `private void`, мы можем написать команды, подлежащие выполнению после щелчка пользователем на командной кнопке **button1**.

Как можно видеть, у нас теперь две вкладки: вкладка программного кода **Form1.cs** и вкладка визуального проекта программы **Form1.cs [Design]** (другое название этой вкладки — *дизайнер формы*). Переключаться между ними можно мышью или нажатием комбинации клавиш <Ctrl>+<Tab>, как это принято для переключения между вкладками в Windows, а также функциональной клавишей <F7>.

Напомню, что по условию задачи после щелчка на кнопке **Button1** должно открыться диалоговое окно с надписью: "Всем привет!" Для этого в фигурных скобках обработчика события напишем:

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

Этой записью вызывается *метод* (программа) `Show` объекта `MessageBox` с текстом "Всем привет!" Оператор точка (`.`) указывает системе найти метод `Show` среди методов объекта `MessageBox`. Ну вот, я здесь "нечаянно" проговорился, что объекты, кроме свойств, имеют также и *методы*, т. е. программы, которые обрабатывают объекты.

> ***Совет***
>
> Заметьте, что в языках с С-подобным синтаксисом в конце каждого оператора ставят точку с запятой (`;`). И чтобы не задумываться, в каком режиме в данный момент пребывает ваша клавиатура: в режиме ввода русских букв или английских — *очень удобно*, не переключаясь в другой режим, ввести точку с запятой набором на боковой клавиатуре цифр `59`, удерживая при этом нажатой клавишу <Alt>.

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 MS Visual C# 12!

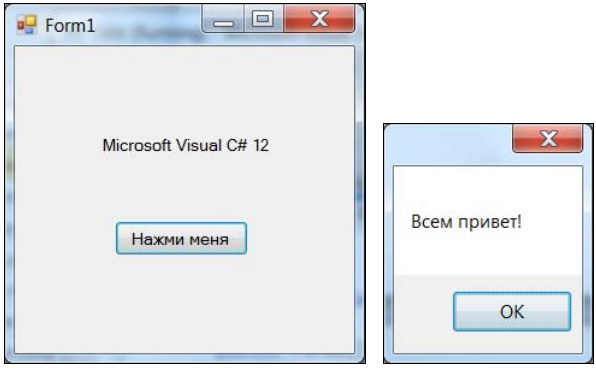


Рис. 1.6. Пример работы программы

Убедиться в работоспособности программы можно, открыв решение First.sln в папке First сопровождающего книгу электронного архива.

> ***Напоминание***
>
> Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 2. Событие *MouseHover*

Немного усложним задачу предыдущего примера. Добавим для объекта **Label1** обработку еще одного события — **MouseHover**. Событие **MouseHover** наступает, когда пользователь указателем мыши "зависает" над каким-либо объектом, причем именно "зависает" (от англ. *hover* — реять, парить), а не просто проводит мышью над объектом. Можно сказать также, что событие **MouseHover** происходит, когда указатель мыши *наведен* на элемент.

> ***Примечание***
>
> Существует еще событие **MouseEnter** (Войти), когда указатель мыши *входит в пределы* области элемента управления (в данном случае метки **Label1**), но сейчас мы воспользуемся именно событием **MouseHover**.

Таким образом, программа в нашем примере должна содержать на экранной форме текстовую метку **Label** и кнопку **Button**. Метка будет отображать текст "Microsoft Visual C# 12", а при щелчке по командной кнопке, на которой по-прежнему написано "Нажми меня", откроется диалоговое окно с сообщением "Всем привет!" Кроме того, когда указатель мыши окажется наведен на текстовую метку (то самое событие **MouseHover**), должно открыться диалоговое окно с текстом "Событие Hover".

Для решения этой задачи запустим Visual Studio 12, щелкнем на пункте меню **New Project**. В открывшемся окне **New Project** в левой колонке найдем пункт **Visual C#**, а затем в области **Templates** (Шаблоны) — в средней колонке — выберем шаблон **Windows Forms Application**. В качестве имени проекта введем имя `Hover` и щелкнем на кнопке **OK**.

В дизайнере формы из панели **Toolbox** перетащим на форму метку **Label** и кнопку **Button**, а затем немного уменьшим размеры формы на свой вкус. Теперь добавим три обработчика событий в программный код. Для этого в панели **Properties** следует щелкнуть на значке молнии (**Events**) и двойными щелчками последовательно выбрать:

- событие загрузки формы **Load**;
- событие щелчок на кнопке button1 — **Click**;
- событие "зависания" указателя мыши на текстовой метке label1 — **MouseHover**.

При этом осуществится переход на вкладку программного кода **Form1.cs** и среда Visual Studio 12 сгенерирует три пустых обработчика события. Например, обработчик последнего события будет иметь вид:

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

Между этими двумя 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!

Уточнение позиции

А сейчас я слегка войду в противоречие с самим себой. Ранее я говорил про визуальную технику программирования, направленную на минимизацию "ручного" 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, 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. Как уже отмечалось во *введении*, написанная только что программа имеет свойство быстро забываться. Если вы посмотрите на строчки кода, которые писали неделю назад, то почувствуете, что многое забыли, по прошествии же месяца 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 те или иные средства. Кроме того, сам программист должен придерживаться 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.

Создадим простой обработчик этого события. Для этого, как и в предыдущих случаях, можно выбрать нужное событие в панели свойств объекта, а можно еще проще — щелкнуть двойным щелчком в пределах проектируемой формы на вкладке **Form1.cs [Design]**. В любом случае на вкладке программного кода будет сформирован "пустой" обработчик события.

Заметим, что для формы умалчиваемым событием, для которого можно двойным щелчком получить пустой обработчик, является событие загрузки формы **Form1_Load**, а для командной кнопки **Button** и метки **Label** таким событием является одиночный щелчок мышью на этих элементах управления. То есть, если щелкнуть двойным щелчком в дизайнере формы по кнопке, то получим в программном коде пустой обработчик одиночного щелчка **Button1_Click**, аналогично получаем программный код и для обработки одиночного щелчка по метке **Label**.

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

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

Здесь, между двумя этими фигурными скобками, обычно вставляют свойства различных объектов и даже часто пишут много строчек программного кода. Назначим свойству **Text** объекта **label1** значение `Microsoft Visual C# 12`:

```
label1.Text = "Microsoft Visual C# 12";
```

Аналогично для объекта **button1**:

```
button1.Text = "Нажми меня";
```

О средстве IntelliSense

Совершенно необязательно писать каждую букву приведенных команд. Например, для первой строчки достаточно написать `la` — появится раскрывающееся меню, откуда вы сможете выбрать нужные для данного контекста ключевые слова. Это работает IntelliSense — очень мощное и полезное современное средство редактирования программного кода (его иногда называют *суфлером*). И если вы от Visual Studio 12 перешли в другую систему программирования, где подобный сервис отсутствует, то станете ощущать сильный дискомфорт.

Пользуясь функцией IntelliSense, очень удобно после ввода оператора разрешения области действия или оператора-точки — в обоих случаях вводим символ точки (.) — получать список допустимых вариантов дальнейшего ввода. Можно выделить элемент и нажать клавишу <Tab> или <Enter> или щелкнуть двойным щелчком по элементу, чтобы вставить его в код. Так что 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 весьма длинные имена ключевых слов, имена переменных и проч.

Я призываю вас, уважаемые читатели, также использовать в своих программах для названий переменных, объектов наиболее ясные, полные имена, причем предпочтительно 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 или через ссылку `this`, или используя ключевое слово `base`:

```
base.Text = "Приветствие";
```

Написав последнюю строчку кода, мы должны увидеть на экране программный код, представленный в листинге 1.1.

Листинг 1.1. Фрагмент файла Form1.cs, содержащего программный код с тремя обработчиками событий

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace Hover
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void Form1_Load(object sender, EventArgs e)
        {
            // Процедура обработки события загрузки формы
            this.Text = "Приветствие";
            // или base.Text = "Приветствие";
            label1.Text = "Microsoft Visual C# 12";
            button1.Text = "Нажми меня";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Процедура обработки события "щелчок на кнопке"
            MessageBox.Show("Всем привет!");
        }
        private void label1_MouseHover(object sender, EventArgs e)
        {
            // Обработка события, когда указатель мыши "завис" над меткой:
            MessageBox.Show("Событие Hover!");
        }
    }
}
```

Комментарии, поясняющие работу программы, в окне редактора кода выделены зеленым цветом, чтобы в тексте он выразительно отделялся от прочих элементов программы. В языках с C-подобным синтаксисом (C#, C++, PHP, Java, JavaScript) комментарий пишут после двух прямых слэшей (`//`) или внутри пар `/* */`.

> ***Совет***
>
> Уважаемые читатели, даже если вам кажется весьма очевидным то, что вы пишете в программном коде, *добавьте комментарий*. Как показывает опыт, даже весьма очевидный замысел программиста забывается удивительно быстро. Человеческая память отметает все, что по оценкам организма считается ненужным. В любом случае, даже если текст программы вполне ясен, в начале программы обязательно опишите ее назначение и способ использования, т. е. как бы "преамбулу" программы. Далее в примерах мы будем следовать этому правилу.

На рис. 1.7 приведен пример работы программы.

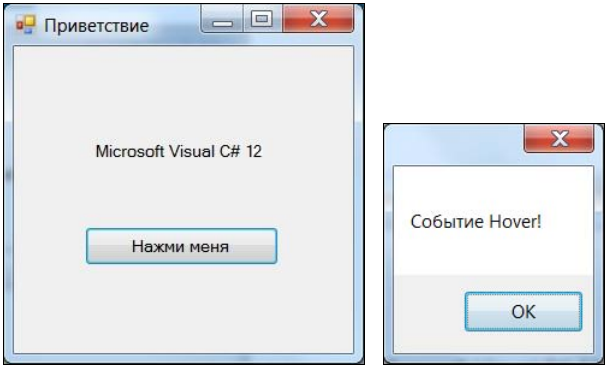


Рис. 1.7. Пример работы программы

Обычно в редакторах программного кода используется моноширинный шрифт, в котором все символы имеют одинаковую ширину — от точки до прописной русской буквы "Ш". По умолчанию в редакторе программного кода Visual C# 12 задан шрифт Consolas. Однако если пользователь привык к шрифту Courier New, то шрифт по умолчанию можно изменить, выбрав меню **Tools | Options | Environment | Fonts and Colors**.

Теперь закроем проект: **File | Close Project**. Система предложит нам сохранить проект, сохраним его под именем Hover. Теперь программный код этой программы можно посмотреть, открыв решение Hover.sln в папке Hover сопровождающего книгу электронного архива.

Пример 3. Выбор нужной даты

Условие следующей нашей задачи состоит том, чтобы, например, при заказе железнодорожных билетов или при регистрации в качестве отдыхающего в санатории (ситуаций может быть много) при щелчке на соответствующем элементе управления появлялся бы календарь, из которого можно было бы выбрать необходимую дату. Это может быть дата отправления поезда, или дата окончания отдыха в санатории, или другие ситуации, связанные с выбором нужной даты.

Для решения этой задачи запускаем систему программирования Visual Studio 12 и щелкаем на пункте **New Project**. В открывшемся окне **New Project** выбираем пункт **Visual Basic**, а затем в области шаблонов (в средней колонке) — шаблон (**Templates**) **Windows Forms Application**. В качестве имени проекта введем имя `ВыборДаты` и щелкнем на кнопке **OK**.

В дизайнере формы из панели **Toolbox** перетащим на форму командную кнопку **Button**, метку **Label** и элемент **DateTimePicker**. Этот элемент непосредственно выполняет функцию выбора даты. Если щелкнуть на стрелке элемента **DateTimePicker**, то раскроется календарь для выбора даты (рис. 1.8). Также мы хотим, чтобы календарь элемента **DateTimePicker** раскрывался при нажатии кнопки **Button**. В итоге выбранная дата должна отображаться в текстовой метке **Label**.

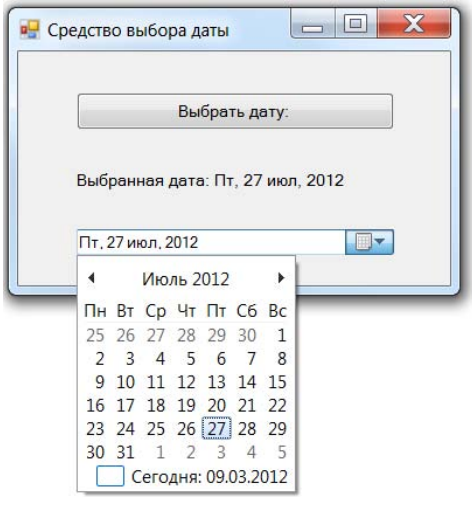


Рис. 1.8. Выбор нужной даты из раскрывающегося календаря

Немного изменим указателем мыши размеры экранной формы и расположим элементы управления, как показано на рис. 1.8. Двойной щелчок в пределах формы вызовет переход на вкладку программного кода **Form1.cs**. При этом среда Visual Studio сгенерирует пустой обработчик события загрузки формы **Form_Load** (см. листинг 1.2). Таким же образом, т. е. двойным щелчком на командной кнопке в дизайнере формы, мы получим пустой обработчик события "щелчок на кнопке". Нам еще понадобится обработчик события **ValueChanged**. Чтобы его получить, перейдем на вкладку конструктора формы **Form1.cs[Design]** и в панели свойств, щелкнув на значке молнии (**Events**), для элемента **DateTimePicker1** двойным щелчком выберем событие **ValueChanged** (можно также на вкладке конструктора формы просто щелкнув двойным щелчком на элементе **dateTimePicker1**). Это приведет к созданию пустого обработчика этого события на вкладке **Form1.vb**. Добавим на вкладке программного кода недостающие команды (листинг 1.2).

Листинг 1.2. Файл Form1.cs, содержащий программный код выбора необходимой даты

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
// Программа выбора нужной даты
namespace ВыборДаты
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Обработка события загрузки формы:
            this.Text = "Средство выбора даты";
            dateTimePicker1.Format = DateTimePickerFormat.Custom;
            dateTimePicker1.CustomFormat = "ddd, dd MMM, yyyy";
            button1.Text = "Выбрать дату:";
            label1.Text = String.Format(
                            "Сегодня: {0}", dateTimePicker1.Text);
        }
        private void dateTimePicker1_ValueChanged(
                                    object sender, EventArgs e)
        {
```

```
            // Обработка события изменения даты:
            label1.Text = String.Format(
                "Выбранная дата: {0}", dateTimePicker1.Text);
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Обработка события "щелчок на кнопке":
            // Передаем фокус на элемент управления dateTimePicker1:
            dateTimePicker1.Focus();
            // Имитируем нажатие клавиши <F4>:
            SendKeys.Send("{F4}");
        }
    }
}
```

Как видно из программного кода, при обработке события загрузки формы задается нужный формат отображения даты:

- вначале (первые три буквы "ddd") — вывод дня недели в краткой форме;
- затем ("dd MMM") — числа и названия месяца, также в краткой форме;
- и, наконец, вывод года ("yyyy").

При обработке события изменения даты ValueChanged текстовой метке label1 присваиваем выбранное значение даты. При этом пользуемся методом String.Format. Использованный формат "Выбранная дата: {0}" означает — взять нулевой выводимый элемент, т. е. свойство Text объекта DataTimePicker1, и записать его вместо фигурных скобок.

При обработке события "щелчок на кнопке" вначале передаем фокус на элемент управления DateTimePicker1. Теперь для раскрытия календаря можно использовать нажатие клавиши <F4>.

> ***Пояснение***
>
> Стандартно функциональная клавиша <F4> обеспечивает раскрытие списка в приложениях Windows. Например, если в браузере Internet Explorer передать фокус щелчком мыши на адресную строку, то после нажатия клавиши <F4> раскроется список веб-страниц, которые вы уже посещали, и вам останется лишь выбрать в этом списке нужный ресурс. В данном случае мы имитируем нажатие клавиши <F4> — метод Send посылает активному приложению сообщение о нажатии соответствующей клавиши.

Пример работы программы показан на рис. 1.8.

Заметим, что подобный пример использования раскрывающегося календаря приведен в документации Visual Studio на сайте Microsoft **www.msdn.com**. Автором внесены в него лишь некоторые изменения.

Убедиться в работоспособности программы можно, открыв решение ВыборДаты.sln в папке ВыборДаты сопровождающего книгу электронного архива.

Пример 4. Ввод данных через текстовое поле *TextBox* с проверкой типа методом *TryParse*

При работе с формой очень часто ввод данных организуют через элемент управления **TextBox** (Текстовое поле). 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, если введено не число, сообщает об этом пользователю.

Решая сформулированную задачу, запускаем Visual Studio 12 и выбираем пункт меню **File | New | Project**. В открывшемся окне **New Project** выбираем пункт **Visual C#**, а затем в области шаблонов (в средней колонке) — шаблон (**Templates**) **Windows Forms Application**. В качестве имени проекта введем имя `Корень` и щелкнем 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**. Поместить названные элементы на проектируемую экранную форму можно также двойным щелчком мыши на каждом элементе в панели **Toolbox**. Таким образом, в форме будут находиться три элемента управления. Расположим их так, как показано на рис. 1.9.

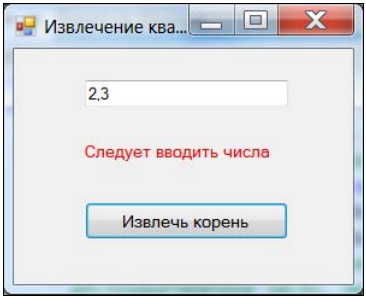


Рис. 1.9. Пример работы программы, если введено не число

Теперь нам следует изменить некоторые свойства элементов управления. Это удобно сделать при обработке загрузки формы. Чтобы получить пустой обработчик загрузки формы, двойным щелчком щелкнем на проектируемой экранной форме. Сразу после этого мы попадаем на вкладку программного кода **Form1.cs**. Здесь задаем свойствам формы (к форме обращаемся посредством ссылки `base`), командной кнопки **Button1**, текстового поля **TextBox1** и метке **Label1** следующие значения:

```
base.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, двойным щелчком щелкнув

по этой кнопке на вкладке дизайнера формы. Между двумя появившимися строчками программируем диагностику правильности вводимых данных, конвертирование строковой переменной в переменную типа `Single` и непосредственное извлечение корня (листинг 1.3).

Листинг 1.3. Извлечение корня с проверкой типа методом `TryParse`

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
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()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            button1.Text = "Извлечь корень";
            label1.Text = String.Empty;
            // или label1.Text = null;
            base.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)
        {
        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,
        // шестнадцатеричное число, экспоненциальный вид числа и проч.).
        // Третий параметр форматирует значения на основе текущего языка
        // и региональных параметров из Панели управления — Язык и
        // региональные стандарты числа 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; // — цвет текста на метке
            return; // — выход из процедуры или Return
        }
        // Извлечение корня с преобразованием в тип Single:
        var Y = (Single)Math.Sqrt(X);
        // или var Y = Convert.ToSingle(Math.Sqrt(X));
        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, число ли введено 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.

Примечание

По умолчанию при инсталляции русифицированных версий Windows XP и Windows 7 разделителем целой и дробной части числа принимается запятая. Однако эту установку можно изменить, если в Панели управления выбрать опцию **Язык и региональные стандарты**, затем на вкладке **Форматы** щелкнуть на кнопке **Дополнительные параметры** и в открывшейся вкладке указать в качестве разделителя целой и дробной частей вместо запятой — точку. В обоих случаях (и для запятой, и для точки) метод `TryParse` будет работать так, как указано на вкладке **Форматы**.

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, какие-либо 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. 1.9 показан пример работы программы как раз в этом случае. Из рисунка видно, что функция `TryParse` не восприняла введенные символы `2,3` как число, поскольку автор специально для демонстрации данного примера указал на вкладке **Форматы** 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 12 являются методами класса `Math`. Их можно увидеть, набрав `Math` и введя так называемый оператор разрешения области действия (`.`). В раскрывающемся списке вы увидите множество математических функций: `Abs`, `Sin`, `Cos`, `Min` и т. д. и два свойства — две кон-

станты `E = 2,71...` (основание натуральных логарифмов) и `PI = 3,14...` (число диаметров, уложенных по длине окружности). Функция `Math.Sqrt(X)` возвращает значение типа `double` (двойной точности с плавающей запятой), которое *приводим* с помощью преобразования `CType` или `Convert.ToSingle` к переменной одинарной точности типа `Single`.

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 ее работы представлен на рис. 1.10.

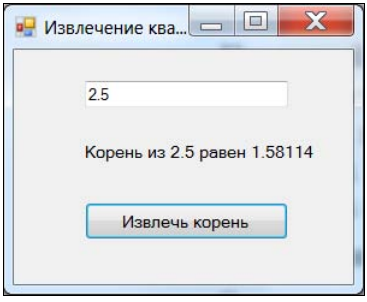


Рис. 1.10. Извлечение квадратного корня

Если выявились ошибки, то работу программы следует проверить отладчиком — клавишами <F10> 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 сопровождающего книгу электронного архива.

Пример 5. 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 экранной формы, текстового поля **TextBox** и метки **Label**, куда для демонстрации возможностей мы будем "раскрывать" пароль ("паспорт", т. е. введенные в качестве пароля секретные слова), а также командной кнопки **Button — Покажи паспорт**.

После запуска Visual Studio 12 и выбора приложения шаблона **Windows Forms Application Visual C#** перемещаем в проектируемую форму все названные элементы управления. Текст программы приведен в листинге 1.4.

Листинг 1.4. Программа, организующая 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 Паспорт
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Обработка события загрузки формы:
            this.Text = "Введи пароль";
            textBox1.Text = null; textBox1.TabIndex = 0;
            textBox1.PasswordChar = '*';
            textBox1.Font = new Font("Courier New", 10.0F);
            // или textBox1.Font = new Font(
            //        FontFamily.GenericMonospace, 10.0F);
            label1.Text = String.Empty;
            label1.Font = new Font("Courier New", 10.0F);
            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, при обработке события загрузки формы мы 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.

Пояснение

Все символы шрифта Courier New имеют одинаковую ширину, поэтому его и называют моноширинным шрифтом. Кстати, с использованием именно этого шрифта, благодаря одинаковой ширине его букв, удобно программировать таблицы. Еще одним широко используемым моноширинным шрифтом является шрифт Consolas.

Задаем шрифт, используя свойство `Font` обоих объектов: `TextBox1` и `Label1`. Число `10.0` задает 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. 1.11 приведен вариант работы данной программы.

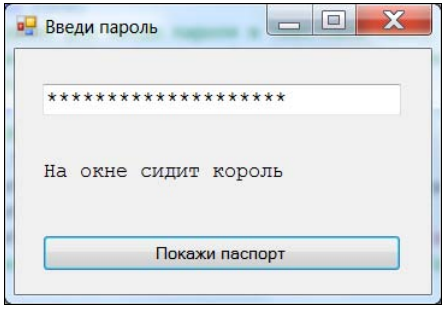


Рис. 1.11. Вариант работы программы

При необходимости используйте отладчик (клавиши <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 Паспорт.sln в папке Паспорт сопровождающего книгу электронного архива.

Пример 6. 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.**

Запустим Visual Studio 12 и выберем шаблон **Windows Forms Application Visual C#**. Используя панель элементов **Toolbox**, в форму поместим метку **Label** и флажок **CheckBox**. В листинге 1.5 приведен текст программы управления этими объектами.

Листинг 1.5. Программный код, управляющий 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 Флажок1
{
    public partial class Form1 : Form
```

```
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            base.Text = "Флажок CheckBox";
            checkBox1.Text = "Полужирный"; checkBox1.Focus();
            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)
        {
            // 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);
        }
    }
}
```

При обработке события загрузки формы задаем начальные значения некоторых свойств объектов `Form1` (посредством ссылки `base`), `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 двойным щелчком 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 будучи записано, как здесь, — выражение выглядит более ясным и понятным.

Программа написана, нажмите клавишу <F5> — в "исправном" состоянии она должна работать примерно так, как показано на рис. 1.12.

Убедиться в работоспособности программы можно, открыв решение Флажок1.sln в папке Флажок1 сопровождающего книгу электронного архива.

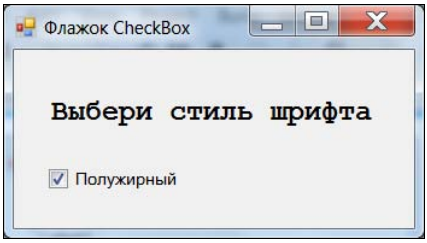


Рис. 1.12. Пример работы программы управления стилем шрифта

Пример 7. 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 `Xor` всегда будет назначать противоположное тому, что записано в `Label1.Font.Style`.

Как можно видеть, применение побитового оператора привело к существенному уменьшению количества программного кода.

> ***Совет***
>
> 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`. Фрагмент новой программы приведен в листинге 1.6.

Листинг 1.6. Усовершенствованный программный код, управляющий стилем шрифта

```
// Программа управляет стилем шрифта текста, выведенного на метку Label,
// посредством двух флажков CheckBox. Программа использует побитовый
// оператор (^) Xor — исключающее ИЛИ
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace Флажок2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.Text = "Флажок CheckBox";
            checkBox1.Text = "Полужирный"; checkBox1.Focus();
            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 Font("Courier New", 14.0F,
                                   label1.Font.Style ^ FontStyle.Bold);
        }
        private void checkBox2_CheckedChanged(object sender, EventArgs e)
        {
            label1.Font = new Font("Courier New", 14.0F,
                                   label1.Font.Style ^ FontStyle.Italic);
        }
    }
}
```

Как можно видеть, здесь принципиально ничего нового нет, только лишь добавлена обработка события изменения состояния флажка **CheckedChanged** для **CheckBox2**. Пример работы программы приведен на рис. 1.13.

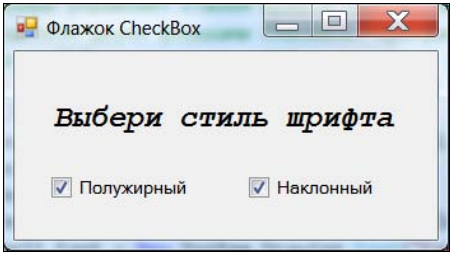


Рис. 1.13. Пример работы усовершенствованной программы

Убедиться в работоспособности программы можно, открыв решение Флажок2.sln в папке Флажок2 сопровождающего книгу электронного архива.

Пример 8. Вкладки *TabControl* и переключатели *RadioButton*

Вкладки программируют для организации управления и оптимального использования экранного пространства. Выразительным примером применения вкладок является диалоговое окно **Свойства обозревателя** Internet Explorer. То есть, если требуется отобразить большое количество управляемой информации, весьма уместно обратить внимание на 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. 1.14.

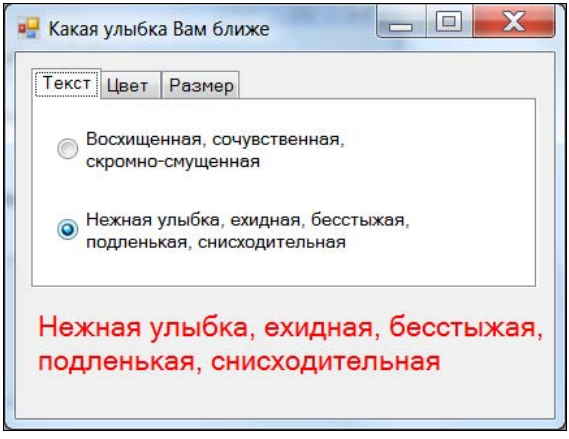


Рис. 1.14. Программа с переключателями и вкладками

Программируя поставленную задачу, запустим Visual Studio 12 и выберем шаблон **Windows Forms Application Visual C#**. Назовем этот проект `Вкладки`. Используя панель элементов **Toolbox**, перетащим в форму мышью элемент управления **TabControl**. Как можно видеть, по умолчанию предусмотрены две вкладки, а по условию задачи (см. рис. 1.14) нам нужны три. Добавить третью вкладку можно в конструкторе формы, а можно и программно.

Вначале покажем, как добавить третью вкладку в конструкторе. Для этого в свойствах (окно **Properties**) элемента управления **TabControl1** выбираем свойство **TabPages** и 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 определяется 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 при обра-

ботке события загрузки формы (листинг 1.7). 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 щелчком правой кнопкой мыши в пределах формы переключаемся на редактирование программного кода.

Листинг 1.7. Программный код, управляющий вкладками и переключателями

```
// 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 Вкладки
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        // Создание третьей вкладки "программно":
        var tabPage3 = new TabPage();
        tabPage3.UseVisualStyleBackColor = true;
        // Добавление третьей вкладки в существующий набор
        // вкладок tabControl1:
        this.tabControl1.Controls.Add(tabPage3);
        // 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 Point(20, 15);
        this.radioButton6.Location = new 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, " + "\n" + "скромно-смущенная";
            radioButton2.Text = "Нежная улыбка, ехидная, бес" +
                "стыжая, " + "\n" + "подленькая, 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 размер шрифта:
        radioButton5.Text = "11 пунктов";
        radioButton6.Text = "13 пунктов";
        label1.Text = radioButton1.Text;
        }
        // Ниже обрабатываем события 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 метода `InitializeComponent()` "программно" создается третья вкладка. Заметьте, что мы ее объявили посредством ключевого слова `var`, причем тип переменной `tabPage3` назначается из выражения инициализации (как объявление `dim` в Visual Basic или `auto` в C++). Далее добавляем новую вкладку `tabPage3` 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 элементе управления (в данном случае на различных вкладках), "отрицают" друг друга, т. е. если пользователь выбрал один переключатель, то другой переходит в противоположное состояние. Отслеживать изменения состояния переключателей удобно с помощью обработки событий переключателей **CheckChanged** (см. листинг 1.7). Чтобы получить пустой об-

работчик этого события в конструкторе формы, следует двойным щелчком щелкнуть на соответствующем переключателе и таким образом запрограммировать изменения состояния переключателей.

Убедиться в работоспособности программы можно, открыв решение Вкладки.sln в папке Вкладки сопровождающего книгу электронного архива.

Пример 9. 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 всплывающая подсказка "Нажми меня" в экзотическом стиле **Balloon** (стиль воздушного шара, стиль комиксов).

Для программирования этой задачи запускаем Visual Studio 12, выполняем команду меню **File | New | Project** и выбираем шаблон **Windows Forms Application Visual C#**. Назовем этот проект `ВоздШар`. 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** у свойства **ToolTip on Tip** мы могли бы написать `Нажми меня`. Однако я предлагаю написать это непосредственно в программном коде. В таком случае программисту не придется долго искать соответствующее свойство, когда он станет применять этот фрагмент в своей следующей программе!

Перейдем на вкладку программного кода — щелкнем правой кнопкой мыши в пределах формы и выберем команду **View Code**. Окончательный текст программы представлен в листинге 1.8.

Листинг 1.8. 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" в стиле Balloon
using System;
//using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ВоздШар
{
    public partial class Form1 : Form
    {
        public Form1()
```

```
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Обработка события "загрузка формы":
            this.Text = "Житейская мудрость";
            label1.Text =
                   "Сколько ребенка ни учи хорошим манерам," + "\n" +
                   "он будет поступать так, как папа с мамой";
           // label1.TextAlign = ContentAlignment.MiddleCenter;
            button1.Text = "Кнопка";
            toolTip1.SetToolTip(
                     button1, "Переключатель" + "\n" + "видимости");
            // 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 исчезновение/появление текста на
            // метке так:
            // if (label1.Visible == true) label1.Visible = false;
            // else label1.Visible = true;
            // или так:
            // label1.Visible = label1.Visible ^ true;
            // здесь ^ (Xor) — логическое исключающее ИЛИ,
            // или совсем просто:
            label1.Visible = ! label1.Visible;
        }
    }
}
```

При обработке события загрузки формы свойству `Text` метки присваиваем некоторый текст, "склеивая" его с помощью знака "плюс" (+) из отдельных фрагментов. Использование в текстовой строке символов `"\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`). Выражение, содержащее `ToolTip1`, устанавливает (`Set`) текст всплывающей подсказки для кнопки `button1` при "зависании" над ней указателя мыши (рис. 1.15). 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 (стиль Balloon).

Чтобы в программном коде получить пустой обработчик события "щелчок мышью на кнопке", следует в дизайнере (конструкторе) формы (т. е. на вкладке

Form1.cs[Designer]) двойным щелчком щелкнуть на кнопке **Button1**. При обработке этого события, как можно видеть, 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.

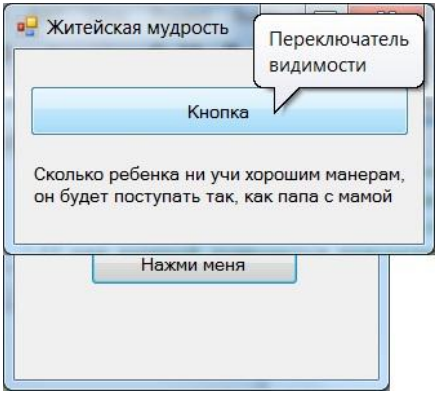


Рис. 1.15. Пример работы программы

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. Теперь вы убедились, что можно поразному программировать подобные ситуации?

> ***Совет***
>
> Как можно видеть, мы использовали много закомментированных строчек программного кода. Очень удобно комментировать строки программного кода в редакторе Visual Studio 12, вначале выделяя их, а затем применяя комбинацию клавиш <Ctrl>+<K> | <C> (Comment). Чтобы убрать с нескольких строк знак комментария, можно, аналогично, — вначале выделить их, а затем воспользоваться уже другой комбинацией клавиш: <Ctrl>+<K> | <U> (Uncomment).

Текст программы можно посмотреть, открыв решение ВоздШар.sln в папке ВоздШар сопровождающего книгу электронного архива.

Пример 10. Калькулятор на основе комбинированного списка *ComboBox*

Элемент управления **ComboBox** служит для отображения вариантов выбора в раскрывающемся списке. Продемонстрируем работу этого элемента управления на примере

программы, реализующей функции калькулятора. Здесь для отображения вариантов выбора арифметических операций используется комбинированный список **ComboBox**.

Запустим Visual Studio 12, выберем шаблон **Windows Forms Application Visual C#**. Назовем этот проект Комби. Из панели **Toolbox** перетащим в форму два текстовых поля **TextBox**, кнопку **Button**, метку **Label** и комбинированный список **ComboBox**.

Текст программы представлен в листинге 1.9.

Листинг 1.9. Суперкалькулятор

```
// Программа, реализующая функции калькулятора. Здесь для отображения
// вариантов выбора арифметических действий используется комбинированный
// список ComboBox
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace Комби
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Суперкалькулятор"; label1.Text = "Равно: ";
            button1.Text = "Выбери операцию";
            comboBox1.Text = "Выбери операцию";
            // Заполнение (инициализация) раскрывающегося списка:
            String[] Операции = {"Прибавить", "Отнять",
                          "Умножить", "Разделить", "Очистить"};
            comboBox1.Items.AddRange(Операции);
            comboBox1.TabIndex = 2;
            textBox1.Clear(); textBox1.TabIndex = 0;
            textBox2.Clear(); textBox2.TabIndex = 1;
        }

        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; Z = 0;
```

```
            var Число_ли1 = Single.TryParse(textBox1.Text,
                      System.Globalization.NumberStyles.Number,
                      System.Globalization.NumberFormatInfo.CurrentInfo,
                      out X);
            var 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; // — если ошибка, то выход из процедуры
            }
            // 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);
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Обработка события "щелчок на кнопке":
            // "Принудительно" раскрываем список:
            comboBox1.DroppedDown = true;
        }
    }
}
```

В данной программе сразу при обработке события загрузки формы 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 назначили `textBox1.TabIndex = 0`. Далее при нажатии пользователем клавиши <Tab> будет происходить переход от элемента к элементу соответственно табличным индексам (рис. 1.16).

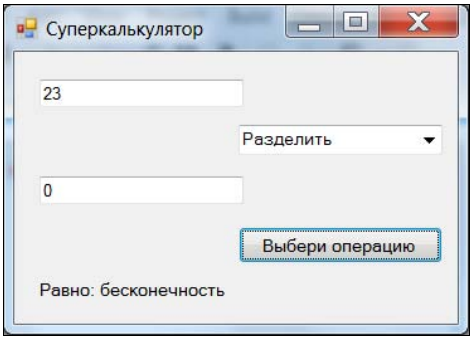


Рис. 1.16. Переход от одного текстового поля к другому

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.

Примечание

По умолчанию при инсталляции русифицированных версий Windows XP и Windows 7 разделителем целой и дробной части числа принимается запятая. Однако эту установку можно изменить, если в Панели управления выбрать опцию **Язык и региональные стандарты**, затем на вкладке **Форматы** щелкнуть на кнопке **Дополнительные параметры** и в открывшейся вкладке указать в качестве разделителя целой и дробной частей вместо запятой — точку. В обоих случаях (и для запятой, и для точки) метод `TryParse` будет работать так, как указано на вкладке **Форматы**.

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 case` передает управление той или иной метке `case`.

Внимание!

Оператор множественного выбора в C# и C++, как говорят программисты, "проваливается" в отличие от Visual Basic. То есть управление, зайдя на какую-либо метку `case`, переходит на следующую метку, поэтому приходится использовать оператор `break`.

Последний оператор в процедуре обработки события изменения индекса выбранного элемента 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. 1.16).

Последняя процедура обработки события "щелчок на кнопке" обеспечивает раскрытие комбинированного списка через нажатие на кнопку, а не на стрелку в элементе **ComboBox**. Эта возможность реализована с помощью метода `DroppedDown` объекта `ComboBox1`.

Убедиться в работоспособности программы можно, открыв решение Комби.sln в папке Комби сопровождающего книгу электронного архива.

Пример 11. 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 таких таблицах каждой ячейке соответствуют номер в таблице и символ. Мы упомянем 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 символы от 0 до 127 (базовая таблица) кодируют английские буквы, цифры, знаки препинания, управляющие символы. Символы от 128 до 255 (расширенная таблица) кодируют русские буквы и символы псевдографики. Некоторые из этих символов соответствуют клавишам IBM-совместимых компьютеров. Эту таблицу еще называют "ДОСовской" по имени операционной системы MS-DOS, где она применяется. Такая кодировка встречается также иногда у некоторых веб-страниц.

В операционной системе Windows используется преимущественно таблица ANSI (Windows 1251). Базовые символы с кодами от 0 до 127 в таблицах ANSI и ASCII совпадают, а расширенные — нет. То есть русские буквы в этих таблицах находятся в разных местах. Из-за этого возникают 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 — $2^{16} = 65\,536$ символов. 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 файлах Блокнота будут присутствовать, например, греческие буквы, математические операторы Π, Δ, Σ и проч. Кстати, греческая буква Σ и математический оператор Σ имеют в Unicode различные коды. Размер файла при сохранении в Блокноте текстов, содержащих символы Unicode, окажется 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 $\pi = 3{,}14$. Результат вычислений выведем в диалоговое окно **MessageBox** также с греческой буквой.

После традиционного запуска Visual Studio 12 и выбора шаблона **Windows Forms Application Visual C#** перетащим в форму две метки **Label**, текстовое поле **TextBox** и командную кнопку **Button**. Посмотрите на рис. 1.17 — так должна выглядеть форма после программирования этой задачи.

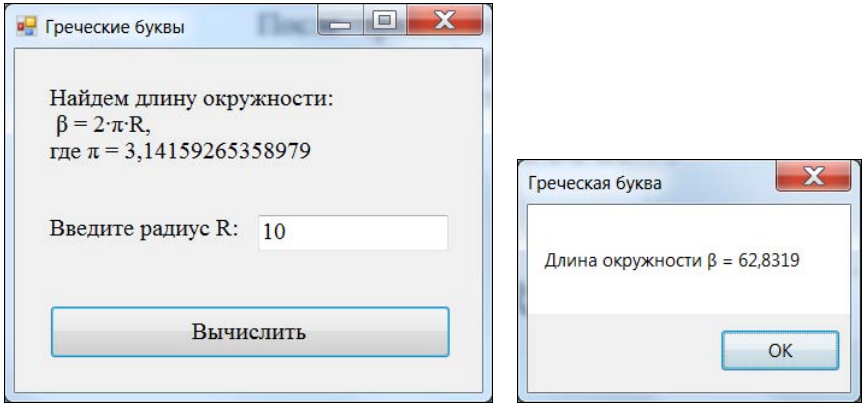


Рис. 1.17. Пример работы программы, использующей символы Unicode

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 12 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 — вставлять подобные символы с помощью функции `Convert.ToChar`, а на вход этой функции подавать номер символа в таблице Unicode. В этом случае, даже если CS-файл будет сохранен в традиционной для Блокнота кодировке ANSI, программа отработает корректно.

Совет

Номер символа в таблице Unicode легко выяснить, выбрав в редакторе MS Word меню **Вставка | Символ**. В открывшейся таблице следует найти этот символ и соответствующий ему код в шестнадцатеричном представлении.

Чтобы перевести шестнадцатеричное представление в десятичное, следует перед шестнадцатеричным числом поставить `0x`. Например, после выполнения оператора `n = 0x3B2` в переменную `n` будет записано десятичное число 946. На этом месте в таблице Unicode расположена греческая буква β.

Именно таким образом запрограммирована данная задача (листинг 1.10).

Листинг 1.10. 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.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace Unico
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
        this.Font = new Font("Times New Roman", 12.0F);
        this.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, \n" + "где {2} = {3}",
           Convert.ToChar(0x3B2), Convert.ToChar(0x2219),
           Convert.ToChar(0x3C0), Math.PI);
        // Здесь: 0=бета, 1 — точка, 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
        var 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; // — если ошибка, то выход из процедуры
        }
        var beta = 2 * 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, при обработке события загрузки формы мы задали шрифт Times New Roman, 12 пунктов — *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 сопровождающего книгу электронного архива.

ГЛАВА 2

Программирование консольных приложений

Пример 12. 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. Такая же ситуация возникает и 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.

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

Для этого запускаем Visual Studio 12, создаем новый проект (**New Project**), в пункте **Visual C#** выбираем шаблон **Console Application Visual C#**, 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 12 приготовила несколько строк программного кода. Эта вполне работоспособная программа, по ее операторам можно "пройтись" отладчиком, последовательно нажимая клавишу <F10>. При запуске консольного или Windows-приложения Visual C# метод `Main()` является первым вызываемым методом. Между фигурными скобками после оператора `void Main()` мы вставим собственный программный код (листинг 2.1).

Листинг 2.1. Ввод и вывод данных в консольном приложении

```
using System;
// Программа организует ввод двух чисел, их сложение и вывод суммы на консоль
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();
            // Ввод первого слагаемого:
            Console.WriteLine("Введите первое слагаемое:");
            var Строка = Console.ReadLine();
            // Преобразование строковой переменной в число:
            var X = Single.Parse(Строка);
            // Ввод второго слагаемого:
            Console.WriteLine("Введите второе слагаемое:");
            Строка = Console.ReadLine();
            var Y = Single.Parse(Строка);
            var 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();
        }
    }
}
```

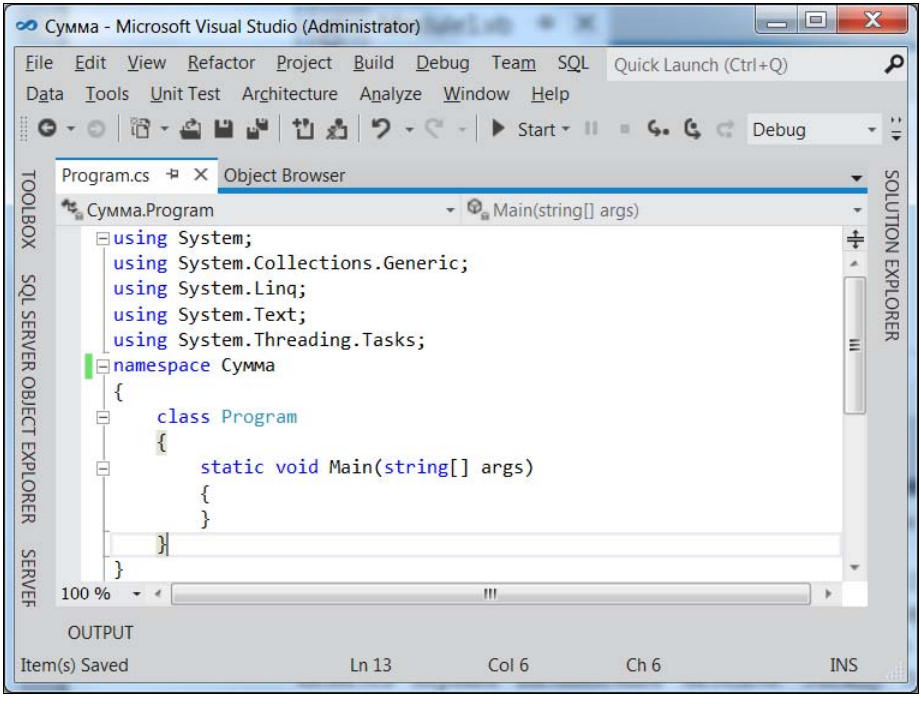


Рис. 2.1. Вкладка программного кода

Итак, в данной программе `void 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`. После ключевого слова `var`, названия переменной и оператора присваивания мы сознательно не указали тип переменной, поскольку транслятор сам назначит ей тип соответственно возвращаемому значению функции `ReadLine()`. Для преобразования строки символов, введенных пользователем в числовое значение типа `Single`, используем метод `Parse`. Тип данных `Single` применяется тогда, когда число, записанное в переменную, может иметь целую и дробную части. Переменная типа `Single` занимает 4 байта.

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. Пример работы данной программы представлен на рис. 2.2.

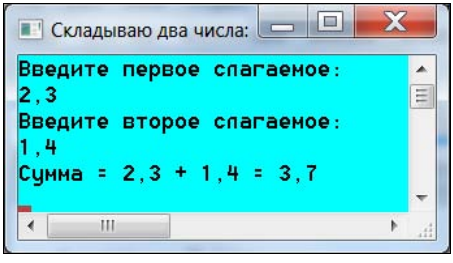


Рис. 2.2. Пример работы консольного приложения

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.

Убедиться в работоспособности программы можно, открыв решение Сумма.sln в папке Сумма сопровождающего книгу электронного архива.

Напоминание

Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 13. Вывод на консоль таблицы чисел с помощью форматирования строк

Воспользуемся тем, что шрифт вывода на консоль является моноширинным (т. е. каждый выводимый символ — и точка, и заглавная буква "Ш" — имеют одинаковую ширину), и покажем, как путем форматирования строк можно в окно консоли вывести таблицу. Для этого выведем в таблицу извлеченный квадратный корень в цикле от нуля до десяти. Итак, запускаем Visual Studio 12, заказываем новый проект (**New Project**), в пункте **Visual C#** выбираем шаблон **Console Application**, задаем имя решения — `ТаблКорней` и на вкладке программного кода вводим программный код, представленный в листинге 2.2.

Листинг 2.2. Вывод таблицы извлечения квадратного корня на консоль

```
using System;
// Консольное приложение задает цвета и заголовок консоли, а затем выводит
// таблицу извлечения квадратного корня от нуля до десяти
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).

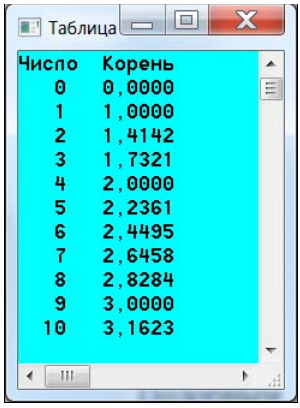


Рис. 2.3. Вывод на консоль таблицы

Пользуясь методом `String.Format`, удобно форматировать различные переменные, поэтому подобным образом мы будем часто обрабатывать строки в последующих примерах данной книги.

Убедиться в работоспособности программы можно, открыв решение ТаблКорней.sln в папке ТаблКорней сопровождающего книгу электронного архива.

Пример 14. Вызов метода *MessageBox.Show* в консольном приложении. Формат даты и времени

Программирование консольного приложения возвращает тех, кто помнит, в период конца 1980-х годов, когда появились первые персональные компьютеры с очень слабой производительностью и небольшой памятью. Для вывода данных Visual C# имеет удобное средство `MessageBox.Show`, однако его используют, когда работают с экранными формами, а не с консолью. Но в консольном приложении его вызвать все-таки можно. Покажем на следующем примере, как практически это сделать.

Итак, в этом 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 12, создадим новый проект, в пункте **Visual C#** выберем шаблон **Console Application**, зададим имя решения: `ConsoleMessageBox`.

Далее необходимо в проект *добавить ссылку на библиотеку*, содержащую объект `MessageBox`. Для этого в пункте меню **Project** выберем команду **Add References** и в новом окне в колонке **Name** двойным щелчком щелкнем на ссылке **System.Windows.Forms**. Теперь убедимся, что эта ссылка добавлена нами в текущий проект, для чего в пункте меню **Project** выберем команду **ConsoleMessageBox Properties**. Далее, щелкнув указателем мыши на пункте **References** (Ссылки), увидим добавленную в наш проект выбранную ссылку. Это ссылка на библиотеку (файл) System.Windows.Forms.dll, которая расположена в папке Program Files, куда инсталлирована среда .NET Framework.

Чтобы обеспечить более компактные выражения 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;
// Добавляем пространство имен для более коротких выражений:
using System.Windows.Forms;
// Добавим ссылку на библиотеку System.Windows.Forms.dll, содержащую
// класс MessageBox:
// Project | Add Reference | Framework | System.Windows.Forms
namespace ConsoleMessageBox
{
    class Program
    {
        static void Main(string[] args)
        {
            var 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) — здесь — \"полная\" дата и \"короткое\" время: {0:f}\n" +
      "(F) — \"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 date/time — время по Гринвичу. . {0:U}\n" +
      "(Y) — по этому формату выводится только год. . . . . . {0:Y}",
                                                        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).

Чтобы во время выполнения программы *не появлялось* черное окно DOS, выполним следующее. В меню **Project** выберем пункт **ConsoleMessageBox Properties**, далее в новом окне, щелкнув указателем мыши на пункте **Application**, выберем в раскрывающемся меню **Application type** пункт **Windows Forms Application**. Теперь наша программа будет работать без вывода черного окна консоли.

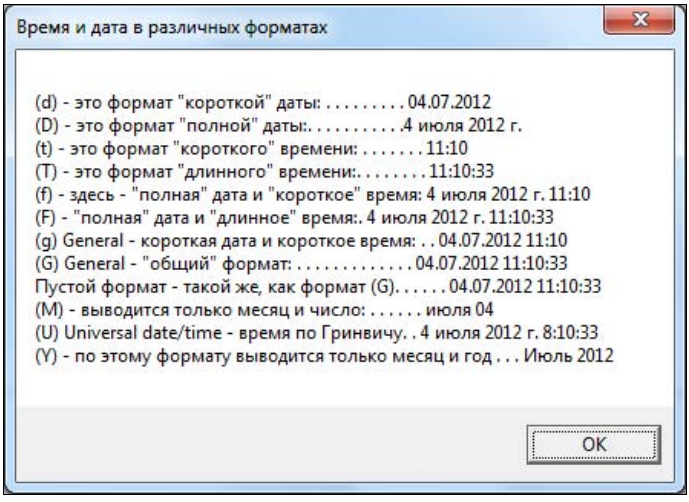


Рис. 2.4. Вывод текущего времени и даты в различных форматах

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ConsoleMessageBox сопровождающего книгу электронного архива.

Пример 15. Вызов функций Visual Basic из программы C#

Покажем, как можно воспользоваться в 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 шаблон 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 **Framework** щелкнем двойным щелчком на ссылке **Microsoft.VisualBasic**. Теперь в окне **Solution Explorer**, раскрыв пункт **References**, увидим добавленную в наш проект 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 три 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.
// ~ ~ ~ ~ ~ ~ ~ ~
// Добавим ссылку на библиотеку Microsoft.VisualBasic.dll:
// Project | Add Reference | Framework | Microsoft.VisualBasic
namespace ССылкаНаVisualBasic
{
    class Program
    {
        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).

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`.

Примечание

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.

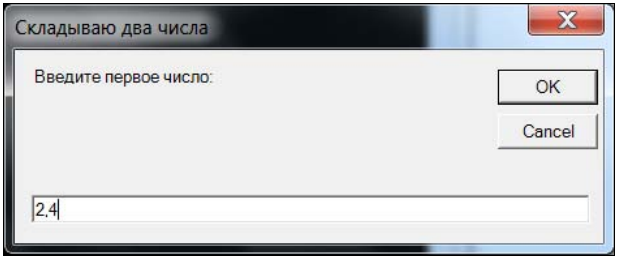


Рис. 2.5. Диалоговое окно ввода данных

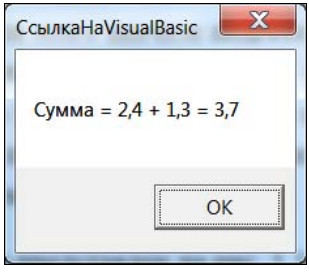


Рис. 2.6. Диалоговое окно вывода данных

Совет

Каждый раз, запуская это приложение, нам приходится искать диалоговые окна под черным окном консоли. Чтобы избавиться от этого окна, в свойствах проекта **Project | Properties** на вкладке **Application** в раскрывающемся меню **Output type** щелкните на пункте **Windows Application**.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ССылкаНаVisualBasic сопровождающего книгу электронного архива.

Пример 16. Программирование интервалов с помощью оператора *else if*

В этой задаче мы намереваемся сыграть роль как бы кассира в супермаркете, и если наш очередной покупатель купил товаров на сумму до 10 условных денежных единиц, то ему назначаем скидку 10 %, для суммы от 10 до 20 единиц полагается скидка 20 %. А при покупке на сумму свыше 20 единиц покупателю положена скидка 30 %. Таким образом, в этой задаче мы должны программировать следующие четыре интервала:

- сумма ≤ 0 → выход;
- $0 <$ сумма ≤ 10 → скидка = 10 %;

- 10 < сумма ≤ 20 → скидка = 20 %;
- сумма >20 → скидка = 30 %.

Как оптимальным образом решать подобные задачи? Можно воспользоваться традиционным условием `If – Else`, но при этом программа будет выглядеть очень запутанно (такой запутанный клубок условий называют "спагетти"). Мы потратим много усилий на отладку этого "спагетти", а когда потребуется модернизировать данный участок программы (например, добавить еще один интервал), нам опять придется решать очередной ребус. Хорошим выходом из этой ситуации является использование оператора множественного выбора `switch case`. Такое решение вполне обеспечит наглядность, понятность программы. Заметим, кстати, что понятность написанной программы имеет сегодня *гораздо* больший приоритет, чем скорость работы написанной программы.

Ну что, уважаемый читатель, я убедил вас использовать в решении этой задачи `switch case`? Между тем, не все так однозначно, — применение оператора `Select Case` считается весьма банальным, есть более красивое решение задачи программирования интервалов. Примерно с 2002 года, т. е. с момента появления Visual Studio .NET, мы можем решать эту задачу с помощью операторов `else if`, которые встраивают в конструкции `If – Else`. Оператор `else if` обеспечивает наглядность, выразительность и *эффективность* программирования!

Итак, запустим Visual Studio 12, далее создаем новый проект, в пункте **Visual C#** выберем шаблон **Console Application**, зададим имя решения: `Интервалы`. На вкладке программного кода введем программный код, представленный в листинге 2.5.

Листинг 2.5. Использование *else if* для программирования интервалов

```
// Программирование интервалов
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе.
// ~ ~ ~ ~ ~ ~ ~ ~
namespace Интервалы
{
    class Program
    {
        static void Main(string[] args)
        {
            // Здесь четыре интервала.
            // Задаем строку заголовка консоли:
            Console.Title = "Программирование интервалов";
            Console.BackgroundColor = ConsoleColor.Cyan; // — цвет фона
            Console.ForegroundColor = ConsoleColor.Black; // — цвет текста
            Console.Clear();
            Console.WriteLine("Введите сумму за приобретенные товары:");
            Single Сумма, Скидка;
            do // или for (; ; ) — бесконечный цикл
            {
                // Получаем значение очередной суммы от пользователя:
                var Строка = Console.ReadLine();
```

```
                // Преобразование строковой переменной в число:
                Сумма = Single.Parse(Строка);
                if (0 < Сумма && Сумма <= 10)
                {
                    // Второй интервал 0 < Сумма <= 10
                    Скидка = 10; // 10%
                }
                else if (10 < Сумма && Сумма <= 20)
                {
                    // Третий интервал 10 < Сумма <= 20
                    Скидка = 20; // 20%
                }
                else if (Сумма > 20)
                {
                    // Четвертый интервал Сумма > 20
                    Скидка = 30; // 30%
                }
                else
                {
                    // Первый интервал Сумма <= 0
                    // Цен <= 0 не существует, поэтому выход из цикла:
                    Console.WriteLine(
                      "Сумма = {0}, поэтому — выход из программы", Сумма);
                    Console.ReadKey();
                    break;
                }
                Console.WriteLine(
                     "Сумма = {0}, поэтому скидка = {1}%", Сумма, Скидка);
            } while (true); // Конец тела вечного цикла
        }
    }
}
```

И вовсе не для "красного словца" мы только что сказали, что операторы `else if` обеспечивают эффективный программный код. Дело в том, что программирование множе-

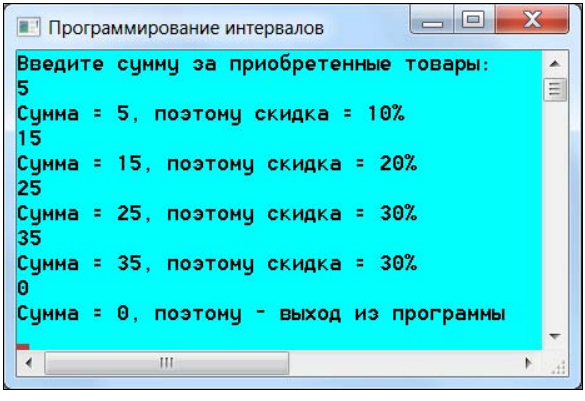


Рис. 2.7. Выбор интервала с помощью `else if`

ственного выбора, или, как его еще называют *мультиветвления*, обеспечивает наибольшую скорость выполнения написанного программного кода именно при использовании операторов `else if`.

На рис. 2.7 приведен пример работы обсуждаемой программы.

Убедиться в работоспособности программы можно, открыв решение Интервалы.sln в папке Интервалы сопровождающего книгу электронного архива.

Пример 17. Замечательной структурой данных является словарь *Dictionary*

Словарь данных `Dictionary` 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 удобство и технологичность манипулирования месяцами года для какого-либо анализа с помощью словаря данных `Dictionary`. Здесь мы помещаем в словарь данных названия месяцев — они будут являться ключом словаря, а также количество дней в данном месяце, которое будет представлять значение. В таком случае очень удобно манипулировать парами "ключ — значение". Для примера выведем из этого словаря на консоль названия месяцев, в которых количество дней равно 30.

Для решения этой задачи запускаем Visual Studio 12, далее создаем новый проект, в пункте **Visual C#** выбираем шаблон **Console Application**, указываем имя (**Name**): `Месяцы`. В листинге 2.6 приведен программный код данного консольного приложения.

Листинг 2.6. Использование словаря данных для манипулирования месяцами года

```
// Программа создает словарь данных типа Dictionary и записывает в этот
// словарь названия месяцев и количество дней в каждом месяце. Ключом
// словаря является название месяца, а значением — количество дней.
// Используя цикл foreach, программа выводит на консоль только те месяцы,
// количество дней в которых равно 30
using System;
using System.Collections.Generic;
// Другие директивы using удалены, поскольку они не используются в данной
// программе.
namespace Месяцы
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            // Задаем цвет текста на консоли для большей выразительности:
            Console.ForegroundColor = ConsoleColor.White;
            Console.Title = "Использование словаря данных";
            // Создаем словарь данных с полями типа String и int:
            var Месяцы = new Dictionary<String, int>();
            // Инициализировать словарь месяцев можно так:
            //  Месяцы["Январь"] = 31;    Месяцы["Июль"] = 31;
            //  Месяцы["Февраль"] = 29;   Месяцы["Август"] = 31;
            //  Месяцы["Март"] = 31;      Месяцы["Сентябрь"] = 30;
            //  Месяцы["Апрель"] = 30;    Месяцы["Октябрь"] = 31;
            //  Месяцы["Май"] = 31;       Месяцы["Ноябрь"] = 30;
            // можно так:
            //  Месяцы.Add("Июнь", 30);   Месяцы.Add("Декабрь", 31);
            // А МОЖНО БОЛЕЕ ИНТЕРЕСНО:
            //  Инициализация словаря месяцев
            for (int i = 0; i <= 11; i++)
                Месяцы.Add(
                    System.Globalization.CultureInfo.CurrentUICulture.
                    DateTimeFormat.MonthNames[i],
                    DateTime.DaysInMonth(2012, i+1));

            Console.WriteLine("Месяцы с 30 днями: ");
            Console.WriteLine();
            // Поиск в словаре месяцев, содержащих 30 дней:
            foreach (KeyValuePair<String, int> Месяц in Месяцы)
                if (Месяц.Value == 30)
                    Console.WriteLine("{0} — {1} дней",
                                      Месяц.Key, Месяц.Value);

            // Ждем от пользователя нажатия какой-либо клавиши:
            Console.ReadKey();
        }
    }
}
```

В программном коде вначале создаем объект класса `Dictionary`, т. е. словарь данных с полями типа `String` — для названий месяцев и типа `Int32` — для количества дней. Инициализация словаря, т. е. присвоение начальных значений здесь очень удобная, название месяца выступает в роли "индекса", если использовать терминологию массивов. Поиск в словаре месяцев, содержащий 30 дней, мы организовали с помощью цикла `foreach`. Здесь использована системная структура `KeyValuePair` (пара "ключ — значение"), предназначенная именно для манипуляций со словарем, хотя ее можно использовать и в других случаях.

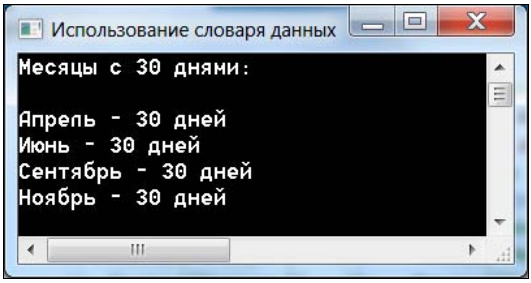


Рис. 2.8. Вывод на консоль месяцев, содержащих 30 дней

Результат работы программы показан на рис. 2.8.

Убедиться в работоспособности программы можно, открыв решение Месяцы.sln в папке Месяцы сопровождающего книгу электронного архива.

ГЛАВА 3

Инициирование и обработка событий мыши и клавиатуры

Пример 18. Координаты курсора мыши относительно экрана и элемента управления

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

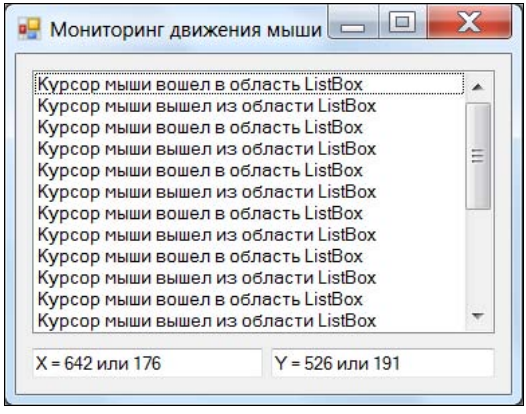


Рис. 3.1. Пример работы программы определения координат курсора мыши

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 12 и выбора шаблона **Windows Forms Application Visual C#** из панели элементов управления **Toolbox** перетащим в форму элемент управления **ListBox** и два текстовых поля.

В этой программе нам понадобится обработать событие загрузки формы, а также три события, относящиеся к объекту **ListBox1**. Получить пустой обработчик события загрузки формы весьма просто — надо только двойным щелчком щелкнуть на проекти-

руемой экранной форме. А чтобы получить три соответствующих пустых обработчика для объекта **ListBox1**, следует в конструкторе формы в панели свойств (**Properties**) щелкнуть на пиктограмме молнии и в открывшемся списке возможных событий для объекта **ListBox1** выбрать следующие три события: **MouseEnter**, **MouseLeave** и **MouseMove**. Соответственно, получим четыре пустых обработчика событий.

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 Мониторинг
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.Text = "Мониторинг движения мыши";
        }
        private void listBox1_MouseEnter(object sender, EventArgs e)
        {
            // Процедура обработки события, когда указатель мыши входит
            // в видимую часть элемента управления ListBox:
            // ~ ~ ~ ~ ~ ~ ~ ~
            // Добавляем в список элементов новую запись
            listBox1.Items.Add("Курсор мыши вошел в область ListBox");
        }
        private void listBox1_MouseLeave(object sender, EventArgs e)
        {
            // Процедура обработки события, когда указатель мыши покидает
            // элемент управления ListBox:
            listBox1.Items.Add("Курсор мыши вышел из области ListBox");
        }
        private void listBox1_MouseMove(object sender, MouseEventArgs e)
        {
            // Процедура обработки события, происходящего при перемещении
            // указателя мыши на элементе управления ListBox:
            // ~ ~ ~ ~ ~ ~ ~ ~
            // Свойство объекта Control MousePosition возвращает точку,
            // соответствующую текущему положению мыши относительно
            // левого верхнего угла монитора
```

```
            textBox1.Text = String.Format("X = {0} или {1}",
                                          Control.MousePosition.X, e.X);
            textBox2.Text = String.Format("Y = {0} или {1}",
                                          Control.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 Мониторинг.sln папки Мониторинг сопровождающего книгу электронного архива.

> ***Напоминание***
>
> Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 19. Создание элемента управления *Button* "программным" способом и подключение события для него

Мы уже знаем, как перенести мышью в форму нужный элемент, используя панель элементов управления **Toolbox**. Чтобы сделать разработку программы более управляемой, в этом примере мы научимся создавать элементы управления в форме "программным" способом, т. е. с помощью написания непосредственно программного кода, не прибегая при этом к панели 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, наша 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 12, в окне **New Project** выбираем шаблон **Windows Forms Application Visual C#**. Чтобы к программному коду добавить пустой обработчик события загрузки формы, двойным щелчком щелкнем на проектируемой экранной форме. 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 НоваяКнопка
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
        // Создание кнопки без панели элементов управления:
        var Кнопка = new Button();
        // Задаем свойства кнопки:
        Кнопка.Visible = true;
        // Ширина и высота кнопки:
        Кнопка.Size = new Size(150, 30);
        // Расположение кнопки в системе координат формы:
        Кнопка.Location = new Point(70, 80);
        Кнопка.Text = "Новая кнопка";
        // Добавление кнопки в коллекцию элементов управления
        this.Controls.Add(Кнопка);
        // Подписку на событие Click для кнопки можно делать "вручную".
        // Связываем событие Click с процедурой обработки этого события:
        Кнопка.Click += new EventHandler(КЛИК);
        }
        // Создаем обработчик события Click для кнопки:
        private void КЛИК(object sender, EventArgs e)
```

```
        {
            MessageBox.Show("Нажата новая кнопка");
        }
    }
}
```

Как можно видеть, при обработке события загрузки формы создается новый объект `Кнопка` 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 двойным щелчком 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 из открывшегося списка 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, как показано 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. Эту процедуру мы назвали "`Кнопка_Click`". Создание обработчика события "щелчок на кнопке" также показано в листинге 3.2. В этой процедуре предусмотрен вывод сообщения "Нажата новая кнопка". На рис. 3.2 приведен пример работы программы.

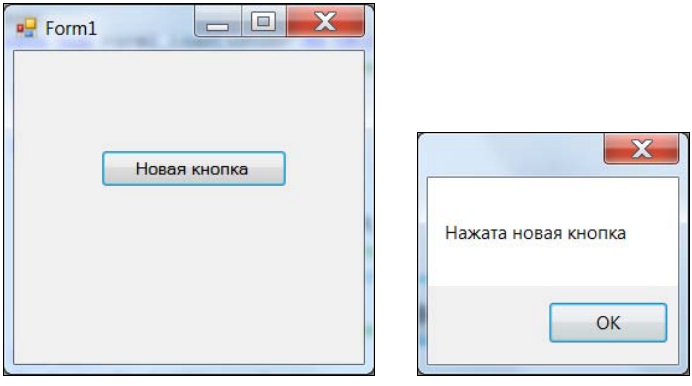


Рис. 3.2. Создание кнопки программным способом

Убедиться в работоспособности программы можно, открыв решение НоваяКнопка.sln папки НоваяКнопка сопровождающего книгу электронного архива.

Пример 20. 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 (C++, C#, F# и Visual Basic) 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 мы убедились в *главе 1*, синтаксис языка позволяет назвать процедуру обработки события как угодно, даже по-русски.

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 обработки события `sender`, мы определим, на какой из двух кнопок щелкнули мышью.

Итак, запустим Visual Studio 12, в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ДваСобытияОднаПроц`. Затем из панели элементов перенесем в форму две командные кнопки и текстовую метку. Далее двойным щелчком мыши в пределах проектируемой формы создадим пустой обработчик загрузки формы и перейдем к вкладке программного кода, куда и впишем необходимые строки 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.Drawing;
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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            base.Text = "Щелкните на кнопке"; label1.Text = null;
            // Связываем события Click от обеих кнопок с одной
            // процедурой Кнопка_Click:
            button1.Click += new EventHandler(КЛИК);
            button2.Click += new EventHandler(КЛИК);
```

```
            // Подпиской на событие называют связывание названия события,
            // например, Click, с названием процедуры обработки события,
            // например, КЛИК посредством EventHandler
        }
        // Создаем обработчик события Click для кнопки:
        private void КЛИК(object sender, EventArgs e)
        {
            // Получить текст, отображаемый на кнопке, можно таким образом:
            // label1.Text = Convert.ToString(sender);
            // или
            // label1.Text = ((Button)sender).Text;
            var Кнопка = (Button)sender;
            label1.Text = "Нажата кнопка " + Кнопка.Text; // или Кнопка.Name
        }
    }
}
```

Как видно из текста программы, при обработке события загрузки формы осуществляется так называемая "подписка на событие", т. е. происходит связывание названия события с названием процедуры обработки события `Кнопка_Click` посредством метода (делегата) `EventHandler`. Этот метод делегирует (передает полномочия) обработку события "щелчок на кнопке" процедуре, именуемой `КЛИК`. Заметим, что события `Click` от обеих кнопок мы связали с одной и той же процедурой `КЛИК`.

Далее создаем процедуру обработки события `КЛИК`. Параметр этой процедуры `sender` содержит *ссылку на объект-источник события*, т. е. кнопку, нажатую пользователем. С помощью 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.

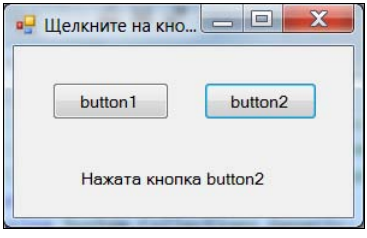


Рис. 3.3. Пример работы программы, определяющей нажатую кнопку

В этом примере мы убедились, что сведения об объекте, который создал событие, находятся в объектной переменной `sender`. Работу этой программы можно исследовать, открыв соответствующее решение в папке ДваСобытияОднаПроц сопровождающего книгу электронного архива.

Пример 21. Калькулятор

Обработка *нескольких событий от разных объектов одной процедурой* оказывается весьма полезной при программировании Калькулятора. Напишем программу Калькуля-

тор с кнопками-цифрами, выполняющую лишь арифметические операции, причем разрешим управление Калькулятором только мышью.

Запустим Visual Studio 12, в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `Калькулятор`. Из панели **Toolbox** перетащим 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.

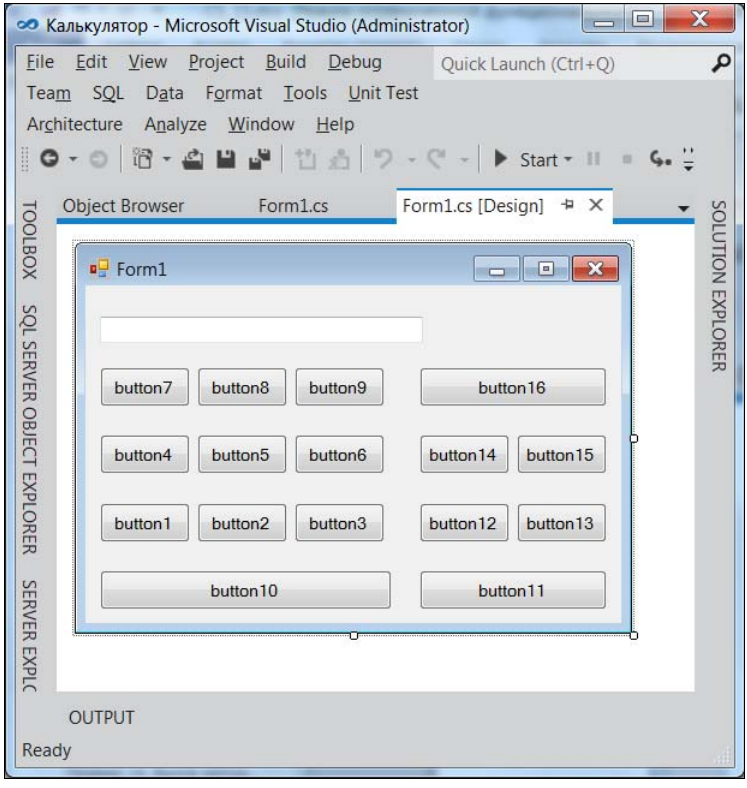


Рис. 3.4. Вкладка конструктора формы

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 Калькулятор
{
    public partial class Form1 : Form
```

```
{
    // Объявляем внешние переменные, видимые из всех процедур класса Form1:
    String Znak = String.Empty; // — знак арифметической операции:
    // Признак того, что пользователь вводит новое число:
    Boolean Начало_Ввода = true;
    // Первое и второе числа, вводимые пользователем:
    Double Число1, Число2;
    public Form1()
    {
        InitializeComponent();
    }
    private void Form1_Load(object sender, EventArgs e)
    {
        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;
            }
            // "Сцепляем" 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 события 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 события 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 в процедуре обработки события загрузки формы `Form1_Load`. Именно здесь заданы надписи на кнопках, нуль в текстовом поле, причем этот нуль 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.

Теперь необходимо значение `Digit` присвоить свойству `TextBox1.Text`, но здесь изначально записан нуль. 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. она *"видна"* в процедурах обработки всех событий.

Это означает, что мы различаем начало ввода числа `Начало_Ввода = True`, когда нуль 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 "сцепляем" 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 `Число1` 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 и известны также оба числа, мы можем выполнить непосредственно арифметическую операцию. После того как пользователь получит результат, например `Результат = Число1 + Число2`, возможно, он захочет с этим результатом выполнить еще какое-либо действие, поэтому этот результат записываем в первую переменную `Число1`.

Заметьте, что в этой программе мы сознательно не предусмотрели обработку исключительной ситуации *деления на ноль*, поскольку среда Visual Studio 12 (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 Калькулятор.sln в папке Калькулятор сопровождающего книгу электронного архива.

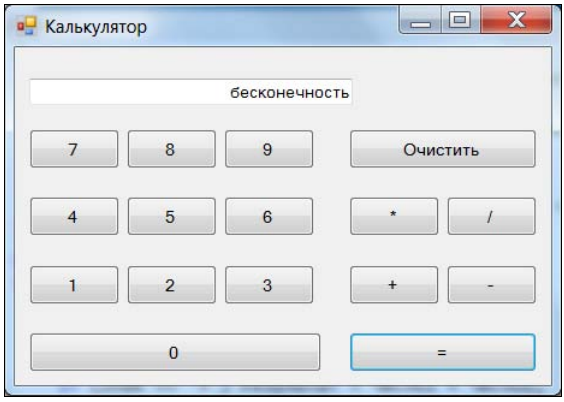


Рис. 3.5. Пример работы калькулятора

Пример 22. Ссылка на другие ресурсы *LinkLabel*

Элемент управления **LinkLabel** позволяет *создавать в форме ссылки* на веб-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 запустим Visual Studio 12, в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `СсылкиLinkLabel`. Затем из панели **Toolbox** перетащим в форму три элемента управления **LinkLabel** и 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 при обработке процедуры загрузки формы (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.Drawing;
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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Обработка процедуры загрузки формы:
            this.Text = "Щелкните на ссылке:";
            linkLabel1.Text = "www.mail.ru";
            linkLabel2.Text = @"Папка C:\Windows\";
            linkLabel3.Text = "Вызвать \"Блокнот\"";
            this.Font = new 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)
        {
            // Обработка щелчка на любой из ссылок:
            var ссылка = (LinkLabel)sender;
            // Выбор ссылки:
            switch (ссылка.Name)
            {
                case "linkLabel1": // Интернет-ресурс
                     System.Diagnostics.Process.Start(
                                 "IExplore.exe", "http://www.mail.ru");
                     break;
                case "linkLabel2": // Папка файловой системы
                     System.Diagnostics.Process.Start(
                                  "C:\\Windows\\");
                     break;
                case "linkLabel3": // Редактор Блокнот
                     System.Diagnostics.Process.Start(
                                  "Notepad", "text.txt");
                     break;
            }
        }
    }
}
```

Как видно из программного кода, в свойстве `Text` каждой из ссылки **LinkLabel** задаем текст, из которого пользователь поймет назначение этих ссылок. В задании свойства

`Text` ссылки `LinkLabel3` для того, чтобы слово "Блокнот" было в двойных кавычках, кавычки используем дважды. Для большей выразительности задаем шрифт Consolas, 12 пунктов. Это шрифт моноширинный (кстати, по умолчанию редактор в Visual Studio 12 также предлагает шрифт Consolas). Поскольку свойство `LinkVisited = True`, то соответствующая ссылка отображается как уже посещавшаяся (изменяется цвет).

Так же как и в предыдущих примерах, 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 обычного присваивания 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 веб-страницы, или имя текстового файла.

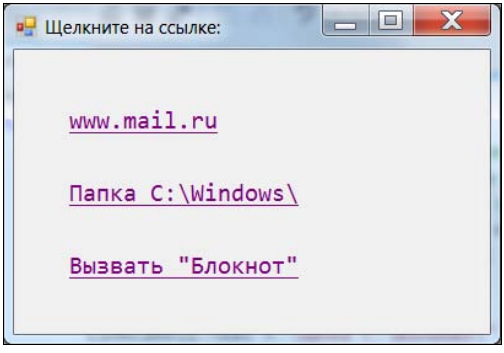


Рис. 3.6. Ссылки на ресурсы

Пример работы обсуждаемой программы приведен на рис. 3.6. Убедиться в ее работоспособности можно, открыв соответствующее решение в папке СсылкиLinkLabel сопровождающего книгу электронного архива.

Пример 23. Обработка событий клавиатуры

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

Напишем программу, информирующую пользователя о тех клавишах и комбинациях клавиш, которые тот нажал. Запустим Visual Studio 12, в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `СобытияКлавиатуры`. Затем из панели **Toolbox** перетащим в форму две текстовые 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. Комплектный программный код приложения приведен в листинге 3.6.

Листинг 3.6. Обработка событий клавиатуры

```
// Программа, информирующая пользователя о тех клавишах и
// комбинациях клавиш, которые он нажал
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace СобытияКлавиатуры
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Устанавливаем шрифт с фиксированной шириной (моноширинный):
            this.Font = new Font(FontFamily.GenericMonospace, 14.0F);
            // Поскольку мы задали этот шрифт увеличенным (от 8 по умолчанию
            // до 14), форма окажется пропорционально увеличенной
            this.Text = "Какие клавиши нажаты сейчас:";
            label1.Text = String.Empty; label2.Text = String.Empty;
        }
        private void Form1_KeyPress(object sender, KeyPressEventArgs e)
        {
            // Здесь обрабатываем событие нажатия клавиши. При удержании
            // клавиши событие 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 += String.Format(
                "Код клавиши: {0} \nKeyData: {1} \nKeyValue: {2}",
                                        e.KeyCode, e.KeyData, 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 текстовую метку 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 пример работы программы.

Убедиться в работоспособности программы можно, открыв соответствующий SLN-файл в папке СобытияКлавиатуры сопровождающего книгу электронного архива.

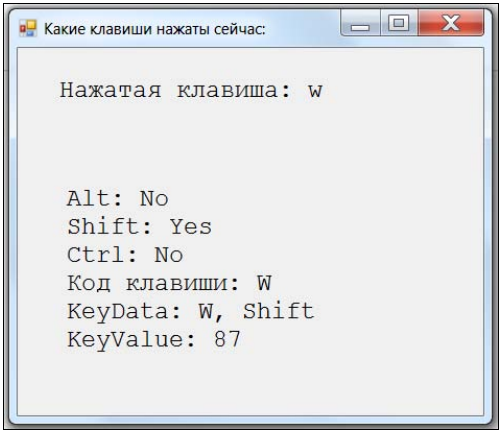


Рис. 3.7. Пример работы программы, определяющей нажатую клавишу

Пример 24. 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 подается 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.

Наша 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* возможности пользователю ввода нечисловых данных.

Запустим систему Visual Studio 12, в окне **New Project** выберем приложение шаблона **Windows Forms Application Visual C#**. Новую программу назовем `ТолькоЦифры`. 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 ТолькоЦифры
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            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 == Convert.ToChar(Keys.Back)) return;
```

```
            // Запрет на ввод других вводимых символов:
            e.Handled = true;
        }
    }
}
```

Как видно из программного кода, при обработке события загрузки формы задаем текст строки заголовка "Введите число" и очищаем текстовое поле методом `Clear()`. Самое интересное начинается при обработке события нажатия клавиши в текстовом поле `textBox1_KeyPress`. Пустой обработчик этого события мы получаем, как и в 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 12 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.

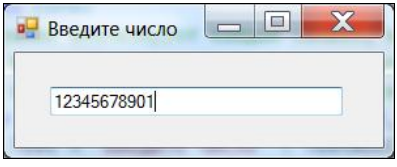


Рис. 3.8. Контроль введенных данных

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ТолькоЦифры сопровождающего книгу электронного архива.

Пример 25. 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 в качестве разделителя целой и дробной частей числа: точка или

запятая — можно, например, запустив Калькулятор Windows. Здесь среди экранных кнопок мы увидим кнопку либо с десятичной точкой, либо с десятичной запятой.

> ***Совет***
>
> Очень легко поменять эту установку системы (обычно по умолчанию в русифицированной версии Windows — запятая). Для этого следует в Панели управления выбрать опцию **Язык и региональные стандарты**, на вкладке **Региональные параметры** щелкнуть на кнопке **Настройка** и на открывшейся 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, который указан на вкладке **Региональные параметры**.

Для решения данной задачи запустим систему Visual Studio 12, в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ТолькоЦифры+ТчкОгЗпт`. 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 разделителя целой и дробной частей числа (т. е. точки или запятой)
// ~ ~ ~ ~ ~ ~ ~ ~
// 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
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ТолькоЦифры_ТчкОгЗпт
{
    public partial class Form1 : Form
    {
        String ТчкИлиЗпт;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Введите число";
            // Выясняем, что установлено на данном ПК в качестве
            // разделителя целой и дробной частей: точка или запятая
            ТчкИлиЗпт = System.Globalization.
                NumberFormatInfo.CurrentInfo.NumberDecimalSeparator;
        }
        private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            var ТчкИлиЗптНАЙДЕНА = false;
```

```
            // Разрешаю ввод десятичных цифр:
            if (Char.IsDigit(e.KeyChar) == true) return;
            // Разрешаю ввод клавиши <Backspace>:
            if (e.KeyChar == Convert.ToChar(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 и 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 оператора `Return` последнего оператора процедуры `e.Handled = True`, запрещающего ввод символа в текстовое поле.

В рассмотренной задаче присутствует определенная проблема с разрешением ввода разделителя целой и дробной частей, поскольку разрешить его ввод мы можем только один раз, но при этом надо иметь в виду, что пользователь может этот разделитель удалить и ввести его в другом месте числовой строки. Эту проблему мы решили следующим образом — каждый раз при очередном нажатии клавиши, разрешив ввод десятичных цифр, ищем в текстовом поле 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 пример работы программы.

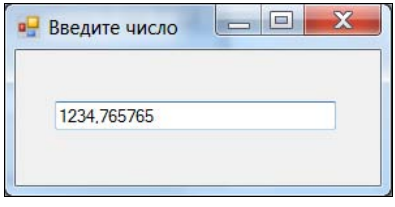


Рис. 3.9. Кроме цифр программа разрешает ввод десятичной запятой

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ТолькоЦифры+ТчкОrЗпт сопровождающего книгу электронного архива.

Пример 26. Программно вызываем событие "щелчок на кнопке"

Щелкая указателем мыши на кнопке, мы вызываем это событие и можем обрабатывать его в соответствующей процедуре. Здесь мы хотим показать, как можно программно вызвать это событие *без щелчка* на самой кнопке. Пусть у нас будут две командные кнопки на экранной форме. Щелкая на первой кнопке, мы программируем появление окна с сообщением о произошедшем событии — нажатии первой кнопки. Однако, щелкая на второй кнопке, мы имитируем нажатие первой кнопки путем программного вызова события нажатия первой кнопки.

Для решения этой задачи запустим систему Visual Studio 12, в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `Пгм-ноВызватьКликКнопки`. На панели элементов **Toolbox** нам понадобятся две командные кнопки. Текст программы представлен в листинге 3.9.

Листинг 3.9. Программный вызов события нажатия кнопки

```
// На экранной форме имеем две кнопки. Щелчок на первой кнопке вызывает
// появление окна с сообщением о произошедшем событии нажатия первой
// кнопки. Щелкая на второй кнопке, мы имитируем нажатие первой кнопки
// путем программного вызова события нажатия первой кнопки
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace Пгм_ноВызватьКликКнопки
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show(
                "Произошло событие \"щелчок на первой кнопке\"");
        }
        private void button2_Click(object sender, EventArgs e)
        {
            // Здесь программно вызываем событие "щелчок на первой
            // кнопке", хотя щелкнули на второй кнопке:
            button1.PerformClick();
            // То же самое можно сделать так:
            // button1_Click(null, null);
            // или так:
            // button1_Click(button1, EventArgs.Empty);
        }
    }
}
```

При обработке события "щелчок на второй кнопке" мы вызвали событие "щелчок на первой кнопке". В комментарии указали варианты вызова этого события другими способами. Это открывает возможность использования приведенной методики и для других элементов управления, не имеющих прямого метода вызова соответствующего события наподобие `PerformClick`.

На рис. 3.10 приведен пример работы программы.

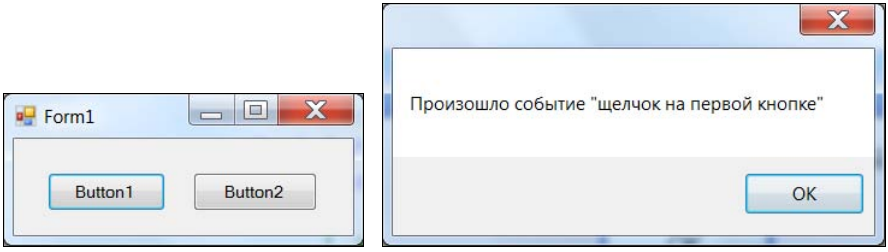


Рис. 3.10. Щелкая на второй кнопке, вызываем событие нажатия первой кнопки

Убедиться в работоспособности программы можно, открыв соответствующее решение (SLN-файл) в папке Пгм-ноВызватьКликКнопки сопровождающего книгу электронного архива.

ГЛАВА 4

Чтение, запись текстовых и бинарных файлов, текстовый редактор

Пример 27. Чтение/запись текстового файла в кодировке *Unicode*. Обработка исключений *try...catch*

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

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

```
// Создание экземпляра StreamReader для чтения из файла:
var Читатель = new System.IO.StreamReader(@"D:\Text1.txt");
// Непосредственное чтение всего файла в текстовое поле:
textBox1.Text = Читатель.ReadToEnd();
Читатель.Close(); // закрытие файла
```

Однако есть некоторые серьезные нюансы. Напишем программу, содержащую на экранной форме текстовое поле и две командные кнопки. При щелчке мышью на первой кнопке происходит чтение текстового файла в текстовое поле в кодировке Unicode. При щелчке на второй кнопке отредактированный пользователем текст в текстовом поле сохраняется в файл на диске.

Запустим систему Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `TxtUnicode`. Далее в форму из панели **Toolbox** перенесем текстовое поле и две командные кнопки. Для текстового поля в окне **Properties** сразу укажем для свойства `Multiline` значение `True`, чтобы текстовое поле имело не одну строку, а столько, сколько поместится в растянутом указате-

лем мыши поле. Одна кнопка предназначена для открытия файла, а другая — для сохранения файла на машинном носителе. В листинге 4.1 приведен текст этой программы.

Листинг 4.1. Чтение/запись текстового файла в кодировке Unicode

```
// Программа для чтения/записи текстового файла в кодировке Unicode
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace TxtUnicode
{
    public partial class Form1 : Form
    {
        // Объявляем ИмяФайла вне всех подпрограмм, чтобы эта переменная
        // была "видна" во всех процедурах обработок событий:
        String ИмяФайла;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Установка начальных значений:
            textBox1.Multiline = true; textBox1.Clear();
            textBox1.Size = new Size(268, 112);
            button1.Text = "Открыть"; button1.TabIndex = 0;
            button2.Text = "Сохранить";
            this.Text = "Здесь кодировка Unicode";
            ИмяФайла = @"D:\Text1.txt";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Щелчок на кнопке Открыть.
            // Русские буквы будут корректно читаться,
            // если открыть файл в кодировке UNICODE:
            try
            {   // Создание объекта StreamReader для чтения из файла:
                var Читатель = new System.IO.StreamReader(ИмяФайла);
                // 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):
                // var МассивСтрок = System.IO.File.
                //                   ReadAllLines(@"D:\Text1.txt");
            }
```

```
            catch (System.IO.FileNotFoundException Ситуация)
            {
                // Обработка исключительной ситуации:
                MessageBox.Show(Ситуация.Message + "\n" +
                                "Нет 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
            {   // Создание объекта StreamWriter для записи в файл:
                var Писатель = new System.IO.
                                        StreamWriter(ИмяФайла, 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(
                //                         @"D:\tmp.tmp", textBox1.Text);
            }
            catch (Exception Ситуация)
            {
                // Отчет обо всех возможных ошибках:
                MessageBox.Show(Ситуация.Message, "Ошибка",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
        }
    }
}
```

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

При обработке события "щелчок на кнопке **Открыть**" организован ввод файла D:\Text1.txt. Обычно в такой ситуации для выбора файла применяют элемент управления **OpenFileDialog**. Мы не стали использовать этот элемент управления для того, чтобы *не "заговорить"* проблему, а также свести к минимуму программный код.

Далее создаем объект (поток) `Читатель` для чтения из файла. Для большей выразительности операций с данным объектом мы написали его название русскими буквами. Затем следует чтение файла `ИмяФайла` методом `ReadToEnd()` в текстовое поле `TextBox1.Text` и закрытие файла методом `Close()`.

При обработке события "щелчок на кнопке **Сохранить**" аналогично организована запись файла на диск через объект `Писатель`. При создании объекта `Писатель` первым аргументом является `ИмяФайла`, а второй аргумент `false` указывает, что данные следует *не добавить* (`append`) к содержимому файла (если он уже существует), а *перезаписать* (`overwrite`). Запись на диск производится с помощью метода `Write()` из свойства `Text` элемента управления `TextBox1`. На рис. 4.1 приведен пример работы программы.

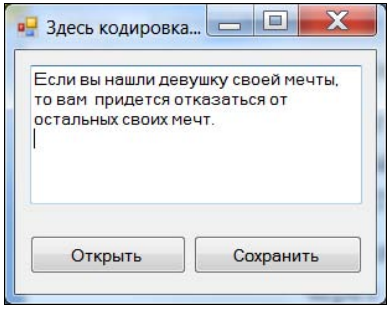


Рис. 4.1. Чтение/запись текстового файла в кодировке Unicode

Внимание!

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

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

Еще об отображении кириллицы

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 примененная в тексте программы технология описана в учебных пособиях по C#, в MSDN, на сайтах, подобных **http://msdn.microsoft.com/library/rus/**. Чаще всего — это 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, с русскими же буквами всегда возникают недоразумения. Об этом говорят программисты на различных форумах в Интернете.

Программистам такие недоразумения следует уметь учитывать. И разрешению *данного недоразумения* посвящена следующая программа.

А убедиться в работоспособности рассмотренной в этом примере программы можно, открыв решение TxtUnicode.sln в папке TxtUnicode сопровождающего книгу электронного архива.

Напоминание

Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 28. Чтение/запись текстового файла в кодировке Windows 1251

В данном примере также на экранной форме разместим текстовое поле и две командные кнопки (назначение этих элементов управления аналогично предыдущему примеру). Однако чтение и запись текстового файла здесь происходит в кодировке Windows 1251 (ANSI). Поскольку структура этой программы аналогична предыдущей, сразу обратимся к ее коду в листинге 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.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace TXT_1251
{
    public partial class Form1 : Form
    {
        String ИмяФайла;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            textBox1.Multiline = true; textBox1.Clear();
            textBox1.Size = new Size(268, 112);
            button1.Text = "Открыть"; button1.TabIndex = 0;
            button2.Text = "Сохранить";
            this.Text = "Здесь кодировка Windows 1251";
            ИмяФайла = @"D:\Text2.txt";
        }
```

```
private void button1_Click(object sender, EventArgs e)
{
    // Щелчок на кнопке Открыть
    try
    {
        // Чтобы русские буквы читались корректно, объявляем
        // объект Кодировка:
        var Кодировка = System.Text.Encoding.GetEncoding(1251);
        // Создание экземпляра StreamReader для чтения из файла
        var Читатель = new System.IO.
                            StreamReader(ИмяФайла, 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):
        var МассивСтрок = System.IO.File.
                      ReadAllLines(@"D:\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(
                                   ИмяФайла, 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(@"D:\tmp.tmp",
        //                                textBox1.Text, Кодировка);
    }
```

```
            catch (Exception Ситуация)
            {
                // Отчет обо всех возможных ошибках:
                MessageBox.Show(Ситуация.Message, "Ошибка",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
        }
    }
}
```

Текст этой программы отличается от предыдущей лишь тем, что здесь введен новый объект — `Кодировка`. Метод `GetEncoding(1251)` устанавливает кодовую страницу Windows 1251 для объекта `Кодировка`. Можно убедиться в этом, если распечатать свойство `Кодировка.HeaderName`.

При создании объекта `Читатель` используются уже два аргумента: имя файла и объект `Кодировка`, указывающий, в какой кодировке (для какой кодовой страницы) читать данные из текстового файла. А при создании объекта `Писатель` используются три аргумента: имя файла, установка `false` (для случая, если файл уже существует, нужно будет не добавлять новые данные, а создавать новый файл) и объект `Кодировка`, указывающий, в какой кодировке писать данные в файл.

Заметьте, что при обработке события "щелчок на второй кнопке" мы объявили переменные `Кодировка` и `Писатель` просто как `var`, т. е. типы этих переменных выводятся из выражения инициализации (как объявление dim в Visual Basic и auto в C++) — это сравнительно новая возможность в Visual Studio, начиная с версии 2010.

Убедиться в работоспособности программы можно, открыв решение TXT_1251.sln в папке TXT_1251 сопровождающего книгу электронного архива.

Пример 29. 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.

Запустим систему Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ТекстовыйРедактор`. Из панели элементов управления **Toolbox** перенесем 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).

Из панели **Toolbox** нам понадобятся еще элементы управления **OpenFileDialog** и **SaveFileDialog**, также перенесем их в форму, хотя их очень легко объявить и использовать непосредственно в программном коде. Чтобы растянуть текстовое поле на всю форму, в свойстве `Multiline` укажем `true` (разрешим введение множества строк).

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! Кстати, чтобы увеличить количество строчек программного кода, которое вы хотели бы одновременно видеть на экране, удобно пользоваться комбинацией клавиш <Shift>+Alt>+<Enter> (**Full Screen**).

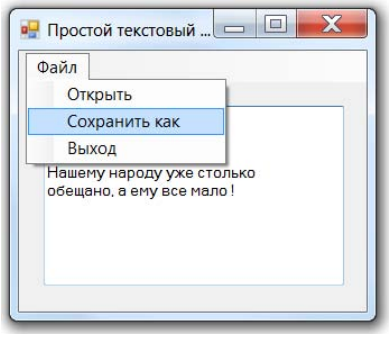


Рис. 4.2. Простой текстовый редактор

Листинг 4.3. Простой текстовый редактор

```
// Простой текстовый редактор
using System;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ТекстовыйРедактор
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            textBox1.Multiline = true; textBox1.Clear();
            textBox1.Size = new Size(268, 160);
            this.Text = "Простой текстовый редактор";
            openFileDialog1.FileName = @"D:\Text2.txt";
            openFileDialog1.Filter =
                "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
            saveFileDialog1.Filter =
                "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
        }
        private void открытьToolStripMenuItem_Click(
                                        object sender, EventArgs e)
        {
            // Вывести диалог открытия файла
            openFileDialog1.ShowDialog();
```

```
        if (openFileDialog1.FileName == String.Empty) return;
        // Чтение текстового файла:
        try
        {
            // Создание экземпляра StreamReader для чтения из файла
            var Читатель = new System.IO.StreamReader(
                                openFileDialog1.FileName,
                                  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.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 сохранитьКакToolStripMenuItem_Click(
                                          object sender, EventArgs e)
    {
        // Пункт меню Сохранить как
        saveFileDialog1.FileName = openFileDialog1.FileName;
        if (saveFileDialog1.ShowDialog() ==
                                     DialogResult.OK) Запись();
    }
    private 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 (Exception Ситуация)
        {
            // Отчет обо всех возможных ошибках
            MessageBox.Show(Ситуация.Message, "Ошибка",
                MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
    }
```

```
        private void выходToolStripMenuItem_Click(
                                        object sender, EventArgs e)
        {
            this.Close();
        }
        private void Form1_FormClosing(
                              object sender, FormClosingEventArgs e)
        {
            // Обработка момента закрытия формы:
            if (textBox1.Modified == false) return;
            // Если текст модифицирован, то спросить, записывать ли файл?
            var MBox = MessageBox.Show(
                "Текст был изменен." + " Сохранить 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 из текста программы, при обработке события загрузки формы 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 стандартное диалоговое окно открытия файлов `OpenFileDialog`, и если полученное в результате диалога имя файла не пусто (`Empty`), 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 стандартное диалоговое окно `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 примера. Как можно видеть, 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** (см. процедуру обработки

события, начинающуюся со слова "`выход`" в листинге 4.3), 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 надо выйти из программы. Изменения в текстовом 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).

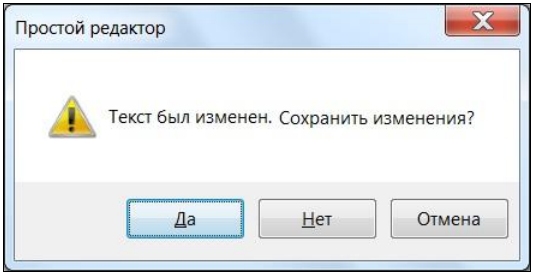


Рис. 4.3. Диалоговое окно при выходе из программы

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

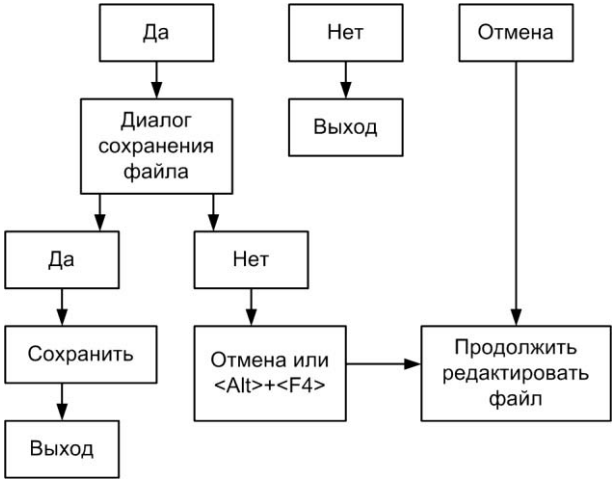


Рис. 4.4. Алгоритм обработки ответа пользователя программы

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 стандартное диалоговое окно сохранения файла, и если при этом он передумал (диалоговое окно сохранения закрыто кнопкой **Отмена** или комбинацией клавиш <Alt>+<F4>), то следует предложить пользователю продолжить редактирование файла: `e.Cancel = True`. Как можно видеть, в процедуре `Form1_FormClosing`, к сожалению, не удается избежать вложенных операторов условия.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ТекстовыйРедактор сопровождающего книгу электронного архива.

Пример 30. 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 этом примере мы создадим инструмент для тестирования студентов — напишем программу, которая читает заранее подготовленный преподавателем текстовый файл с вопросами по какому-либо предмету и 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 представление 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. В архиве с примерами из этой книги представлен более полный файл test_полный.txt, с помощью 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 систему Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `Тестирование`. Затем из панели элементов перенесем в форму две командные кнопки, текстовую метку и три переключателя **RadioButton**. Далее щелчком правой кнопкой мыши перейдем к вкладке программного кода, куда и впишем необходимые строки 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
    {
        // Внешние переменные:
        int СчетВопросов; // Счет вопросов
        int ПравилОтветов;  // Количество правильных ответов
        int НеПравилОтветов; // Количество неправильных ответов
        // Массив вопросов, на которые даны неправильные ответы:
        String[] НеПравилОтветы; // Размерность этого массива зададим позже
        int НомерПравОтвета;  // Номер правильного ответа
        int ВыбранОтвет;  // Номер ответа, выбранный студентом
        System.IO.StreamReader Читатель;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            button1.Text = "Следующий вопрос";
            button2.Text = "Выход";
            // Подписка на событие "изменение состояния
            // переключателей" RadioButton:
            radioButton1.CheckedChanged += new EventHandler(ИзмСостПерекл);
            radioButton2.CheckedChanged += new EventHandler(ИзмСостПерекл);
            radioButton3.CheckedChanged += new EventHandler(ИзмСостПерекл);
            НачалоТеста();
        }
        void НачалоТеста()
        {
            var Кодировка = System.Text.Encoding.GetEncoding(1251);
            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;
                // Задаем размер массива для НеПравилОтветы:
                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;
    // Первую кнопку задаем неактивной, пока студент не выберет
    // вариант ответа
    button1.Enabled = false;
    СчетВопросов = СчетВопросов + 1;
    // Проверка, конец ли файла:
    if (Читатель.EndOfStream == true) button1.Text = "Завершить";
}
void ИзмСостПерекл(Object sender, EventArgs e)
{
    // Кнопка "Следующий вопрос" становится активной, и ей
    // передаем фокус:
    button1.Enabled = true; button1.Focus();
    RadioButton Переключатель = (RadioButton)sender;
    var 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")
            {
                // Закрываем текстовый файл:
                Читатель.Close();
                // 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 * 5.0F) / СчетВопросов);
                // 5F — это максимальная оценка
                button1.Text = "Начать тестирование сначала";
                // Вывод вопросов, на которые "Вы дали неправильный ответ":
                var 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");
            } // Конец условия if (button1.Text == "Завершить")
            if (button1.Text == "Следующий вопрос") ЧитатьСледВопрос();
        }
        private void button2_Click(object sender, EventArgs e)
        {
            // Щелчок на кнопке "Выход"
            this.Close();
        }
    }
}
```

В программе есть несколько переменных, которые объявлены в начале вне всех процедур, чтобы они были "видны" из всех процедур класса `Form1`. В процедуре обработки загрузки формы организуем подписку на событие "изменение состояния переключателей" **RadioButton** одной процедурой `ИзмСостПерекл`. В этой 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, подлежа-

щей 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 нашей 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 зафиксирует выбранный вариант только после 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.

Пример работы тестирующей программы представлен на рис. 4.5.

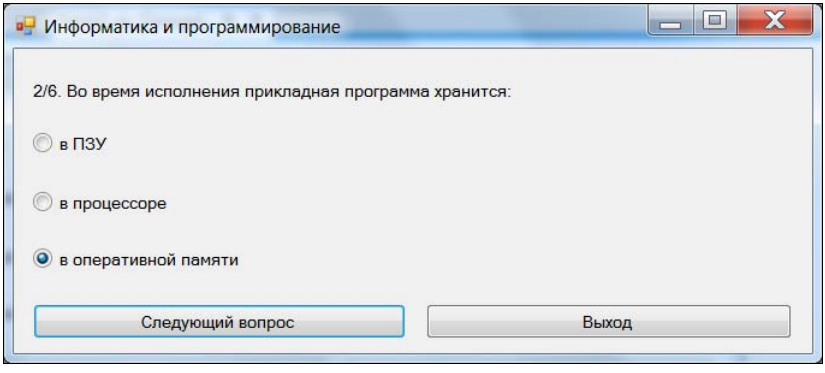


Рис. 4.5. Интерфейс тестирующей программы

На рис. 4.6 показано финальное окно тестирующей программы, где выведено обоснование оценки тестирования со списком вопросов, на которые студент ответил неправильно.

Вы можете совершенствовать эту программу. Например, добавить элемент управления **Timer**, чтобы ограничить время сдачи теста. Или в качестве исходных данных для программы использовать не текстовый файл, который может обеспечить только текстовое представление, а файл HTML. Такой файл будет содержать не только текстовые вопросы, но и графические, включающие изображения.

Убедиться в работоспособности программы можно, открыв решение Тестирование.sln в папке Тестирование сопровождающего книгу электронного архива.

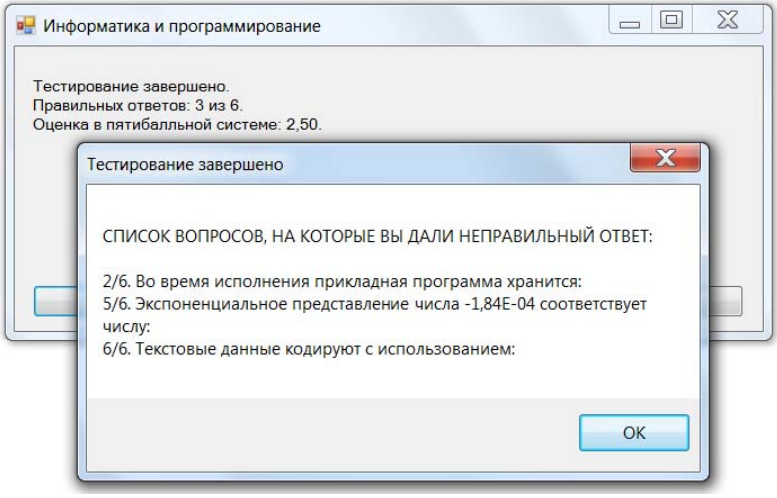


Рис. 4.6. Финальное окно тестирующей программы

Пример 31. Простой RTF-редактор

Читателю, вероятно, известно, что Visual C# 12 имеет элемент управления **RichTextBox** (форматированное текстовое поле). 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.

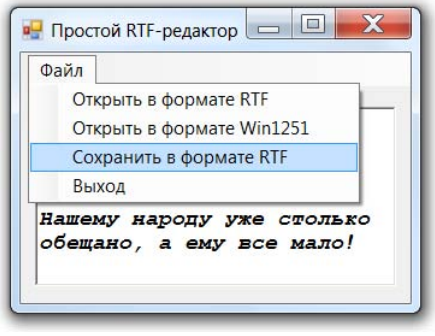


Рис. 4.7. Простой RTF-редактор

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**, **OpenFileDialog** и **SaveFileDialog**. В раскрывающемся меню **Файл** предусмотрим следующие пункты: **Открыть в формате RTF**, **Открыть в формате Win1251**, **Сохранить в формате RTF** и **Выход** (рис. 4.7). Новую программу назовем `RtfРедактор`. 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 RtfРедактор
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Простой RTF-редактор"; richTextBox1.Clear();
            openFileDialog1.FileName = @"D:\Text2.txt";
            saveFileDialog1.Filter = "Файлы RTF (*.RTF)|*.RTF";
            // Оба события щелчка указателем мыши на пунктах меню "Открыть
            // в формате RTF" и "Открыть в формате Win1251" будем
            // обрабатывать одной процедурой ОТКРЫТЬ:
            открытьВФорматеRTFToolStripMenuItem.Click +=
                                        new EventHandler(ОТКРЫТЬ);
            открытьВФорматеWin1251ToolStripMenuItem.Click +=
                                        new EventHandler(ОТКРЫТЬ);
        }
        void ОТКРЫТЬ(Object sender, EventArgs e)
        {
            // Процедура обработки событий открытия
            // файла в двух разных форматах.
            // Выясняем, в каком формате открыть файл:
            var t = (ToolStripMenuItem)sender;
            // Читаем надпись на пункте меню:
            var Формат = t.Text;
            try
            {   // Открыть в каком-либо формате:
                if (Формат == "Открыть в формате RTF")
                {
                    openFileDialog1.Filter =
                                    "Файлы RTF (*.RTF)|*.RTF";
                    openFileDialog1.ShowDialog();
                    if (openFileDialog1.FileName == null) return;
                    richTextBox1.LoadFile(openFileDialog1.FileName);
                }
                if (Формат == "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 Ситуация)
        {
            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 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 Ситуация)
        {
            // Отчет обо всех возможных ошибках:
            MessageBox.Show(Ситуация.Message, "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
    }
    private void выходToolStripMenuItem_Click(
                                        object sender, EventArgs e)
    {
        this.Close();
    }
    private void Form1_FormClosing(
                                object sender, FormClosingEventArgs e)
    {
        // Обработка момента закрытия формы:
        if (richTextBox1.Modified == false) return;
```

```
            // Если текст модифицирован, то выясняем, записывать ли файл?
            var MBox = MessageBox.Show(
                "Текст был изменен.\n" + "Сохранить 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
            } // Конец условия DialogResult.Yes
        }
    }
}
```

Структура программы аналогична программе простого текстового редактора, рассмотренного в *примере 29*. В процедуре обработки события загрузки формы 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 18, 19* и др. *(см. главу 3)*. Из этих примеров читатели знают, что сведения об объекте, создавшем событие, находятся в объектной переменной `sender`. Какой пункт меню выбрал пользователь, можно узнать, конвертировав переменную `sender` в объект `t` класса `ToolStripMenuItem`. При этом мы можем прочитать в свойстве `Text` название выбранного пользователем пункта меню. Таким образом, в строковую переменную `Формат` 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 выбрал пункт меню **Открыть в формате RTF** для открытия текстового файла, он получает сообщение "Недопустимый формат файла".

Сохранение файла (см. 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`.

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 *примера 30*. Вдобавок обработаны два события: пункт меню **Выход** и всевозможные варианты за-

крытия программы традиционными способами Windows. Предусмотрен диалог с пользователем в случае, если имеются несохраненные данные.

> ***Замечание***
>
> 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 RtfРедактор.sln в папке RtfРедактор сопровождающего книгу электронного архива.

Пример 32. Программа ввода каталога координат (числовых данных) из текстового файла

Вообще говоря, хороший тон программирования предполагает ввод подобного рода данных через удобную для пользователя таблицу с интуитивно понятным интерфейсом. Для этой цели, кстати, очень удобно использовать элемент управления **DataGridView** (просмотр сетки данных) из панели элементов **Toolbox**. Однако, если надо поскорее получить конечный результат, и на сервис для пользователя можно обращать не очень много внимания, вполне допустима ситуация, которая легла в основу постановки следующей задачи.

Требуется написать программу ввода каталога координат (числовых данных), где каждая точка представлена четырьмя координатами: *X, Y, U* и *V*. Предполагается, что пользователь в Блокноте заполняет этими числовыми данными текстовый файл. Четыре указанных числа могут располагаться как угодно: или по четыре координаты на одной строчке, или по две координаты в строке, или "1 + 3", возможны и любые другие комбинации, однако количество введенных координат должно быть кратно четырем. При этом в качестве разделителя целой и дробной частей пользователь может использовать одновременно и точку, и запятую (в одном числе — точка, в следующем — запятая, потом наоборот и т. д.). Между числами может быть сколько угодно пробельных символов и/или знаков табуляции (`'\t'`). Пользователь может оставлять пустые строки в середине каталога, а также пустые строки в самом конце файла. Программа должна "понять" намерения пользователя и вывести распознанный каталог координат в текстовое поле экранной формы.

В листинге 4.7 приведен пример такого текстового файла.

Листинг 4.7. Содержание текстового файла с каталогом координат (файл convert1.txt)

```
167.5 437,5 13.8  120,0
217.5 437.5 62,5  131.2

182.5 413.8 35.0  100
210   410   62.5  101.2
240   415   90    113.8
162,5 387.5 21.2  68.8
191.2 372.5 52.5  61.2
211.2 390   67.5  82.5
240   388.8 96.5  88.8
160   357,5 26.2  40
225   367.5 86.2  65
161.2 341.2 31.2  25
187.5 340   56.2  28.8
212.5 337.5 81.2  32.5
241.2 341.2 108.8 42.5
```

Как можно видеть, файл содержит пустые строки в середине и в конце каталога, имеют место и точка, и запятая в качестве разделителя целой и дробной частей числа. Между числами имеется разное количество пробелов и невидимых символов табуляции (следствие нажатия клавиши <Tab>).

Для решения этой задачи запустим систему Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ВводКаталогаКоординат`. В панели **Toolbox** двойным щелчком щелкнем на элементе **TextBox**, в его свойствах зададим свойству **Multiline** значение **True** и растянем текстовое поле на всю экранную форму.

Текст программы приведен в листинге 4.8.

Листинг 4.8. Программа ввода каталога координат

```
// Чтение текстового файла, содержащего каталог координат.
// В данном каталоге каждая точка представлена четырьмя координатами:
// X, Y, U и V. Эти четыре числа могут располагаться как угодно: или
// по четыре координаты на одной строчке, или по две координаты в строке,
// или "1 + 3", или в любых других комбинациях. Однако количество введенных
// координат должно быть кратно четырем. При этом в качестве разделителя
// целой и дробной частей пользователь может использовать точку и/или
// запятую. Между числами может быть сколько угодно пробельных символов
// и/или знаков табуляции ('\t'). Пользователь может пропускать пустые
// строки в середине каталога и/или в конце файла. Программа должна
// "понять" введенный текст и вывести распознанный каталог координат в
// текстовое поле экранной формы
using System;
using System.Windows.Forms;
```

```
// Другие директивы using удалены, поскольку они не используются в данной программе
namespace ВводКаталогаКоординат
{
    public partial class Form1 : Form
    {
        // Объявляем массивы координат вне всех процедур, чтобы
        // они были видны из любой из процедур:
        Double[] X, Y, U, V;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            var ВесьТекст = String.Empty;
            try
            {
                var ТекущийКаталог = System.IO.Directory.
                                                GetCurrentDirectory();
                // Читаем файл в строковую переменную:
                ВесьТекст = System.IO.File.ReadAllText(
                                        ТекущийКаталог + @"\convert1.txt");
            }
            catch (System.IO.FileNotFoundException Ситуация)
            {
                MessageBox.Show(
                    "Нет такого файла" + "\n" + Ситуация.Message,
                    "Ошибка", MessageBoxButtons.OK,
                              MessageBoxIcon.Exclamation);
            }
            catch (Exception Ситуация)
            {
                // Отчет о других ошибках:
                MessageBox.Show(Ситуация.Message, "Ошибка",
                   MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }

            // Во всем тексте заменяем точки на запятые, т. е.
            // можно вводить числа с точкой, а можно с запятой:
            ВесьТекст = ВесьТекст.Replace(".", ",");
            // В качестве сепаратора задаем: знаки табуляции '\t'),
            // перевод каретки '\r', '\n' и пробелы ' ':
            Char[] Сепаратор = { '\t', '\r', '\n', ' ' };
            var Координаты = ВесьТекст.Split(Сепаратор,
                           StringSplitOptions.RemoveEmptyEntries);
            // Теперь в каждом элементе массива Координаты() содержится
            // одна координата
            var n = Координаты.Length; // — количество элементов массива
            var z = n % 4;
```

```
            // % вычисляет остаток от деления целых чисел, в VB: z = n Mod 4
            if (z != 0) MessageBox.Show(
                     "Количество исходных данных не кратно четырем");
            // Здесь уже можно определиться с размерностью массивов:
            X = new Double[n / 4]; U = new Double[n / 4];
            Y = new Double[n / 4]; V = new Double[n / 4];
            // Признаки успешного преобразования из строки в число:
            Boolean A, B, C, D;
            var j = 0; // — счет чисел (элементов) в массиве Координаты()
            for (var i = 0; i <= n / 4 — 1; i++)
            {   // Преобразование из строки в число:
                A = Double.TryParse(Координаты[j], out X[i]); j = j + 1;
                B = Double.TryParse(Координаты[j], out Y[i]); j = j + 1;
                C = Double.TryParse(Координаты[j], out U[i]); j = j + 1;
                D = Double.TryParse(Координаты[j], out V[i]); j = j + 1;
                // Если хотя бы одно из преобразований
                // метода TryParse ложно, то:
                if ((A && B && C && D) == false)
                          MessageBox.Show(String.Format(
                          "В строке {0} — не числовой ввод !", i + 1),
                          "Ошибка", MessageBoxButtons.OK,
                          MessageBoxIcon.Exclamation);
            }
            this.Text = String.Format("Кол-во точек = {0};", n / 4);
            textBox1.Multiline = true; textBox1.Clear();
            // В цикле добавляем строку координат в текстовое поле:
            for (var i = 0; i <= n / 4 — 1; i++)
            {
                textBox1.AppendText(String.Format("X = {0,5:F1};   " +
                "Y = {1,5:F1};   U = {2,5:F1};   V = {3,5:F1};" + "\n",
                                             X[i], Y[i], U[i], V[i]));
                // Использованный формат "{0,5:F1}" означает: взять нулевой
                // выводимый элемент и записать его с выравниванием вправо в
                // следующих пяти позициях в формате с фиксированной точкой
                // и одним десятичным знаком после запятой.
            }
        }
    }
}
```

Как видно из текста программы, процесс распознавания каталога координат происходит при обработке события загрузки формы. Вначале из текущей папки считываем в строковую переменную `ВесьТекст` содержимое файла convert1.txt и во всем тексте делаем контекстную замену точек на запятые, используя метод `Replace`. Далее используем очень эффективную и красивую функцию `Split` для копирования всего текста в строковый массив. Для этого зададимся в качестве сепаратора знаками табуляции `'\t'` (следствие нажатия клавиши <Tab>), невидимыми символами перевода каретки `'\r'`

и `'\n'`, а также пробельными символами `' '`. Благодаря параметру `RemoveEmptyEntries` пустые строки в тексте игнорируются, и функция `Split` возвращает нам массив, в каждом элементе которого записана ровно одна координата (число). Теперь мы можем определенно выяснить количество вводимых данных и назначить, наконец, количество элементов в массивах координат. В следующем цикле преобразуем считанные из файла подстроки в числа с помощью функции `TryParse`. Эта функция возвращает `true` при успешном преобразовании. Поскольку мы имеем четыре числа в каждой строке (X, Y, U и V), получим четыре возврата: `A`, `B`, `C` и `D`. Логическое умножение (конъюнкция) этих булевых переменных ответит на вопрос, было ли хотя бы одно из четырех преобразований метода `TryParse` ложно. В этом случае организуем вывод соответствующего сообщения.

Пример работы данной программы можно увидеть на рис. 4.8.

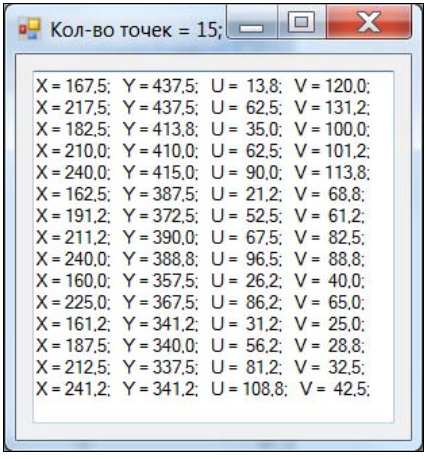


Рис. 4.8. Вывод распознанного каталога координат в текстовое поле

Убедиться в работоспособности этой программы можно, открыв соответствующий SLN-файл в папке ВводКаталогаКоординат сопровождающего книгу электронного архива.

Пример 33. Печать текстового документа

Любой текстовый редактор (и не только текстовый) должен иметь возможность печати на принтере. Мы сознательно не добавляли такую возможность в текстовые редакторы, приведенные в предыдущих разделах, чтобы не запутать читателя. Понятно, что чем больше функциональности имеет программа, тем сложнее ее программный код и тем труднее текст программы для понимания. А наша задача — выразительно и ярко демонстрировать технологии в максимально простой форме.

Программа, представленная в этом примере, имеет следующие скромные возможности: открыть в стандартном диалоговом окне Windows текстовый файл, просмотреть его в окне программы (в текстовом поле) без возможности изменения текста (`ReadOnly`) и при желании пользователя вывести этот текст на принтер. Эту программу назовем `TxtPrint`.

Запустив Visual Studio 12 и выбрав шаблон **Windows Forms Application C#**, в дизайнере формы из панели элементов перенесем следующие элементы управления: текстовое поле **TextBox** со свойством **MultiLine = true**, меню **MenuStrip** с пунктами меню: **Открыть**, **Печать** и **Выход** (рис. 4.9), а также элементы управления **OpenFileDialog** и **PrintDocument**. Текст программы представлен в листинге 4.9.

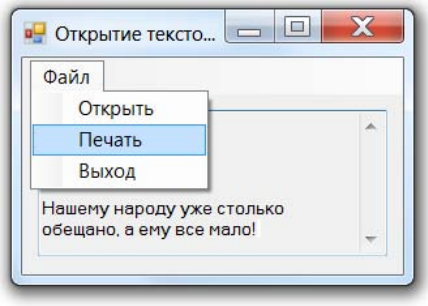


Рис. 4.9. Пример работы программы печати текстового файла

Листинг 4.9. Печать текстового документа

```
// Программа позволяет открыть в стандартном диалоговом окне 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 TxtPrint
{
    public partial class Form1 : Form
    {
        System.IO.StreamReader Читатель;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Открытие текстового файла и его печать";
            textBox1.Multiline = true; textBox1.Clear();
            textBox1.Size = new 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 Ситуация)
    {
        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 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 Ситуация)
            {
                MessageBox.Show(Ситуация.Message);
            }
        }
        private void printDocument1_PrintPage(object sender,
                              System.Drawing.Printing.PrintPageEventArgs e)
        {
            // Событие вывода на печать страницы (PrintPage)
            // Чтобы получить пустой обработчик этого события, можно дважды
            // щелкнуть на значке printDocument1 в дизайнере формы
            Single СтрокНаСтранице = 0.0F;
            Single Y = 0;
            var ЛевыйКрай = e.MarginBounds.Left;
            var ВерхнийКрай = e.MarginBounds.Top;
            var Строка = String.Empty;
            var Шрифт = new Font("Times New Roman", 12.0F);
            // Вычисляем количество строк на одной странице
            СтрокНаСтранице = e.MarginBounds.Height /
                                    Шрифт.GetHeight(e.Graphics);
            // Печатаем каждую строку файла
            var i = 0; // — счет строк
            while (i < СтрокНаСтранице)
            {
                Строка = Читатель.ReadLine();
                if (Строка == null) break; // выход из цикла
                // Для VB: If Строка Is Nothing Then Exit While
                Y = ВерхнийКрай + i *
                       Шрифт.GetHeight(e.Graphics);
                // Печать строки
                e.Graphics.DrawString(Строка, Шрифт, Brushes.Black,
                                  ЛевыйКрай, Y, new StringFormat());
                i = i + 1; // или i += 1 — счет строк
            }
            // Печать следующей страницы, если есть еще строки файла
            if (Строка != null) e.HasMorePages = true;
            // Для VB: If Строка <> Null Then ...
            else e.HasMorePages = false;
        }
        private void выходToolStripMenuItem_Click(
                                          object sender, EventArgs e)
        {
            // Выход из программы
            this.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 текстовых редакторах, поэтому оставим их без комментария. Замечу только, что после чтения файла в текстовое поле назначаем видимость пункту меню **Печать**: `печатьToolStripMenuItem.Visible = true`, поскольку уже есть, что печатать на принтере (файл открыт).

Представляет интерес обработка события "щелчок на пункте меню **Печать**". Здесь во вложенных блоках `try...finally...catch` программа еще раз создает поток `StreamReader`, а затем запускает процесс печати документа `printDocument1.Print`. Если ничего более не программировать, только метод `printDocument1.Print`, то принтер распечатает лишь пустую страницу. Чтобы принтер распечатал текст, необходимо обработать событие `PrintPage` (см. текст программы), которое создает объект `PrintDocument`. То есть роль метода `Print` — это создать событие `PrintPage`.

Обратите внимание на обработку события `printDocument1_PrintPage`. Пример обработки этого события приведен в MSDN. Вначале перечислены объявления переменных, значения некоторых из них получаем из аргументов события `e`, например, `ЛевыйКрай` — значение отступа от левого края, и т. д. Назначаем шрифт печати — Times New Roman, 12 пунктов.

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

В переменной `i` происходит счет строк. Если количество строк оказывается большим, чем число строк на странице, то происходит выход из цикла, поскольку страница распечатана. Если есть еще страницы, а программа выясняет это, анализируя содержимое переменной `Строка`, — если ее содержимое отличается от значения `null` (`Строка != null`), то аргументной переменной `e.HasMorePages` назначаем `true`, что инициирует опять событие `PrintPage`, и процедура `printDocument1_PrintPage` начинает свою работу вновь. И так, пока не закончатся все страницы `e.HasMorePages = false` для печати на принтере.

Убедиться в работоспособности программы можно, открыв решение TxtPrint.sln в папке TxtPrint сопровождающего книгу электронного архива.

Пример 34. Чтение/запись бинарных файлов с использованием потока данных

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

Операции с двоичным файлом гораздо более скоростные, и такой файл занимает существенно меньшее пространство на диске. Зато текстовый файл можно читать и редактировать любым текстовым редактором. Обычно программист выбирает компромисс между этими двумя преимуществами.

Приведем пример самого простейшего случая записи на диск бинарного файла с данными об успеваемости одного студента. Понятно, что эта программа может быть маленькой частью большой программы по обработке успеваемости студентов в вузе. Данная программа принимает от пользователя сведения только об одном студенте в текстовые поля формы. При нажатии кнопки **Сохранить** программа записывает введенные сведения в двоичный файл, а при нажатии кнопки **Читать** читает эти сведения из двоичного файла в текстовые поля формы.

Итак, в форме мы должны иметь три текстовых поля, куда пользователь может записать соответственно номер студента по порядку, фамилию студента и его средний балл успеваемости. Поэтому в форму из панели **Toolbox** перенесем три текстовых поля **TextBox**, три метки **Label** и две командные кнопки: **Читать** и **Сохранить**. Таким образом, получим пользовательский интерфейс, показанный на рис. 4.10. Назовем программу `ReadWriteBin`. Текст ее приведен в листинге 4.10.

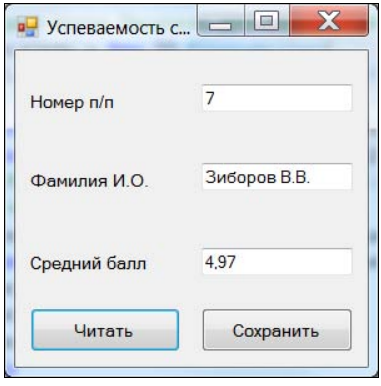


Рис. 4.10. Фрагмент работы программы чтения/записи бинарных файлов

Листинг 4.10. Чтение/запись бинарных файлов

```
// Программа для чтения/записи бинарных файлов с использованием потока данных
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ReadWriteBin
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void Form1_Load(object sender, EventArgs e)
        {
            this.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 (System.IO.File.Exists(@"D:\student.usp") == false) return;
            // Создание потока Читатель
            var Читатель = new System.IO.BinaryReader(
                               System.IO.File.OpenRead(@"D:\student.usp"));
            try
            {
                var Номер_пп = Читатель.ReadInt32();
                var ФИО = Читатель.ReadString();
                var 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
            var Писатель = new System.IO.BinaryWriter(
                               System.IO.File.Open(@"D:\student.usp",
                               System.IO.FileMode.Create));
            try
            {
                var Номер_пп = Convert.ToInt32(textBox1.Text);
                var ФИО = Convert.ToString(textBox2.Text);
                // Разрешаем в качестве разделителя целой и дробной
                // части как запятую, так и точку:
                textBox3.Text = textBox3.Text.Replace(".", ",");
                var СредБалл = Convert.ToSingle(textBox3.Text);
                Писатель.Write(Номер_пп);
                Писатель.Write(ФИО);
                Писатель.Write(СредБалл);
            }
            finally { Писатель.Close(); }
        }
    }
}
```

Как можно видеть, в процедуре обработки события загрузки формы организована инициализация (присвоение начальных значений) элементов формы: текстовых полей, меток и кнопок.

Запись файла на диск происходит при обработке события `button2.Click`, т. е. щелчка мышью на кнопке **Сохранить** (рис. 4.10). С этой целью создаем поток байтов `Писатель` для открытия файла student.usp. Если такого файла не существует, то он создается (`Create`), а если файл уже есть, то он перезаписывается. Как видно из текста программы, для ее упрощения мы не использовали элемент управления `OpenFileDialog` открытия файла в диалоге.

Далее преобразуем записанное в текстовых полях в более естественные типы данных. Номер по порядку `Номер_пп` — это тип целых чисел, преобразование в целый тип может быть реализовано операцией `Convert.ToInt32` (можно использовать другие функции преобразования), для переменной `СредБалл` (средний балл) больше всего подходит тип с плавающей точкой `Single`, при этом преобразование осуществляется операцией `Convert.ToSingle`. Преобразование для строковой переменной `ФИО` является необязательным и приведено для симметрии записей. Операторы `Писатель.Write` записывают эти данные в файл. После блока `finally` происходит обязательное закрытие (`Close`) файла.

Чтение файла выполняется при обработке события "щелчок мышью на кнопке **Читать**". Как уже упоминалось, для максимального упрощения в данной программе не предусмотрено открытие файла через стандартное диалоговое окно, поэтому в начале процедуры выясняем, существует ли такой файл. Если файла D:\student.usp нет, то программируем выход (`return`) из обработчика данного события. Заметьте, чтобы программисту было максимально легко отслеживать ветви оператора `if`, мы написали: "Если файла нет, то `return`". При этом длинная ветвь логики "если файл есть" не включена непосредственно в оператор `if`. Поэтому такой фрагмент программного кода читается (воспринимается) программистом легко — это типичный пример хорошо написанного программного кода.

Далее создается поток байтов `Читатель` из файла student.usp, открытого для чтения. Чтение из потока в каждую переменную реализовано с помощью функции `ReadInt32` — читать из потока `Читатель` в переменную целого типа, аналогично функциям `ReadString` и `ReadSingle`. Затем осуществлено конвертирование этих переменных в строковый тип `Convert.ToString`. Как видно из программы, можно было изначально все текстовые поля записывать в файл без конвертирования, но при дальнейшем развитии этой программы значения полей все равно пришлось бы преобразовывать в соответствующий тип. После блока `finally` происходит закрытие (`Close`) файла. Блок `finally` выполнится всегда, даже если перед ним была команда `return`.

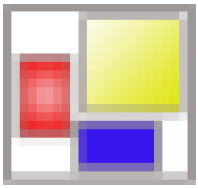
Дальнейшее развитие этой программы может быть осуществлено путем добавления в файл сведений о других студентах. В таком случае при чтении файла количество студентов будет неизвестно. Тогда следует обработать ситуацию достижения конца файла:

```
catch (EndOfStreamException e)
```

а затем закрыть файл.

Убедиться в работоспособности программы можно, открыв решение ReadWriteBin.sln в папке ReadWriteBin сопровождающего книгу электронного архива.

ГЛАВА 5



Редактирование графических данных

Пример 35. 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 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `SimpleImage1`.

Двойной щелчок на проекте формы приведет нас на вкладку программного кода. Работать с графикой в форме можно по-разному. В первом варианте покажем работу с графикой через переопределение метода `OnPaint`.

Метод `OnPaint` является членом класса `Form`. Этот метод можно увидеть в раскрывающемся списке *методов и свойств* объекта `Form1`, набрав `"this. "` внутри какой-нибудь процедуры. Во всплывающей подсказке приводится синтаксис *переопределения* метода `OnPaint`. Переопределить метод означает, что можно добавить к уже существующим функциям собственные. Для этого в окне программного кода вместо `private void Form1_Load...{}` напишем:

```
protected override void OnPaint(PaintEventArgs e){}
```

В этом случае транслятор в окне **ERROR LIST** выдаст сообщение об ошибке, что наша программа: **does not contain a definition for Form1_Load**. Щелкнув двойным щелчком на этом сообщении, мы попадем на вкладку программного кода дизайнера формы **Form1.Designer.cs**. Здесь красной волнистой линией будет подчеркнута подписка на событие `Form1_Load` — нам следует просто удалить эту строку.

Теперь вернемся на вкладку программного кода **Form1.cs** и дополним имеющийся программный код 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;
using System.Drawing;
```

```
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace SimpleImage1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        protected override void OnPaint(PaintEventArgs e)
        {
            this.Text = "Рисунок";
            // Размеры формы
            this.Width = 240; this.Height = 240;
            // Создаем объект для работы с изображением
            Image Рисунок = (Image) new Bitmap(@"D:\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. `Bitmap` (дословный перевод с английского — карта битов) — метод представления изображения в форме прямоугольной матрицы, каждое значение которой соответствует цвету одной из точек изображения.

Теперь обращаемся непосредственно к методу рисования изображения в форме `DrawImage`, извлекая графический объект `Graphics` из аргумента `e` процедуры `OnPaint`.

Пример работы программы показан на рис. 5.1.

Рис. 5.1. Вывод растрового изображения в форму

Заметим, что это не единственный способ работы с графикой. Другой способ — это вызов того же метода `OnPaint` косвенно, через событие формы `OnPaint`. Такой способ работы с графикой представлен в следующей нашей программе — назовем ее `SimpleImage2` (листинг 5.2). Создадим процедуру обработки данного события обычным образом, т. е. на вкладке конструктора формы в панели свойств **Properties** щелкнем на значке 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;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace SimpleImage2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            // В свойствах формы щелкнем на значке молнии и в появившемся
            // списке всех событий для объекта Form1 выберем событие Paint.
            // Событие Paint — это событие рисования формы:
            this.Text = "Рисунок";
            // Создаем объект для работы с изображением:
            var Рисунок = Image.FromFile(@"D:\poryv.png");
            // или var Рисунок = new Bitmap(@"D:\poryv.png");
            // Вывод изображения в форму:
            e.Graphics.DrawImage(Рисунок, 5, 5);
        }
    }
}
```

Покажем *еще один способ* вывода графики в форму — соответствующую программу назовем `SimpleImage3` (листинг 5.3). В этом способе при щелчке на командной кнопке происходит непосредственное создание объекта класса `Graphics`.

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

```
// Простейший вывод изображения в форму
using System;
using System.Drawing;
using System.Windows.Forms;
```

```
// Другие директивы using удалены, поскольку они не используются в данной программе
namespace SimpleImage3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Событие загрузки формы:
            this.Text = "Рисунок";
            button1.Text = "Показать рисунок";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Событие "щелчок на кнопке"
            var Рисунок = new Bitmap(@"D:\poryv.png");
            // Создание графического объекта:
            var Графика = this.CreateGraphics();
            // или var Графика = CreateGraphics();
            Графика.DrawImage(Рисунок, 5, 5);
        }
    }
}
```

Убедиться в работоспособности приведенных программ можно, открыв соответствующие решения в папках SimpleImage1, SimpleImage2 и SimpleImage3 сопровождающего книгу электронного архива.

> ***Напоминание***
>
> Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

В заключение замечу, что с рассмотренными в этом примере методами можно работать *не только* для вывода изображений графических файлов в форму, но и решать многие *другие задачи, связанные с графикой*.

Пример 36. Использование элемента *PictureBox* для отображения растрового файла с возможностью прокрутки

Обычно для отображения точечных рисунков, рисунков из метафайлов, значков, рисунков из файлов в формате BMP, JPEG, GIF, PNG и проч. используется объект класса **PictureBox** (графическое поле). Часто рисунок оказывается слишком большим и не по-

мещается 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 изображения (`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 указав при этом для **PictureBox** `SizeMode = AutoSize`, имеем шанс решить задачу прокрутки изображения.

Запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `СкроллингБольшогоРисунка`. 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 в 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;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace СкроллингБольшогоРисунка
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Скроллинг";
            // Назначаем размеры панели:
            // panel1.Size = new Size(200, 151);
            // Назначаем имя файла рисунка:
            pictureBox1.Image = Image.FromFile(@"D:\Ris.JPG");
            // или pictureBox1.Image = new Bitmap(@"D:\Ris.JPG");
            // Размеры PictureBox в точности соответствуют изображению:
            pictureBox1.SizeMode = PictureBoxSizeMode.AutoSize;
            // Разрешаем прокрутку изображения:
            panel1.AutoScroll = true;
        }
    }
}
```

Как можно видеть, весь наш пользовательский программный код написан для обработки события загрузки формы. Напомним, чтобы получить пустой обработчик этого события, достаточно на вкладке **Form1.cs[Design]** двойным щелчком щелкнуть в пределах проектируемой формы. Текст программы сопровождают подробные комментарии.

Пример работы программы приведен на рис. 5.2.

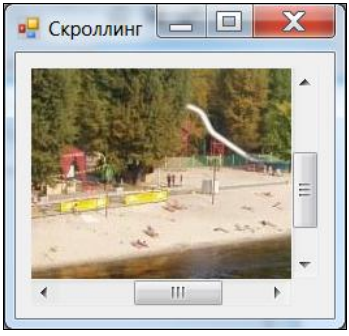


Рис. 5.2. Вывод изображения с возможностью прокрутки

Убедиться в работоспособности данной программы можно, открыв соответствующее решение в папке СкроллингБольшогоРисунка сопровождающего книгу электронного архива.

Пример 37. Рисование в форме графических примитивов (фигур)

В векторных чертежах *графическим примитивом* называют элементарные составляющие чертежа: отрезок, дугу, 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 осуществляется через элемент управления **ListBox** (Список). Причем при рисовании очередного графического примитива нужно "стереть" предыдущий рисунок.

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `РисФигур`. Перетащим в форму из окна **Toolbox** элемент управления **ListBox**. Далее — двойной щелчок в пределах формы, и мы попадаем в пустой обработчик события загрузки формы, где *создадим список* графических примитивов, заполняя коллекцию (`Items`) элементов списка `listBox1` (листинг 5.5).

Листинг 5.5. 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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Выбери графический примитив";
            var Коллекция = new Object[] {
                    "Окружность", "Отрезок", "Прямоугольник",
                    "Сектор", "Текст", "Эллипс", "Закрашенный сектор"
                                        };
            listBox1.Items.AddRange(Коллекция);
            Font = new Font("Times New Roman", 12);
        }
        private void listBox1_SelectedIndexChanged(
                                            object sender, EventArgs e)
        {
            // Здесь вместо этого события можно было бы обработать
            // также событие listBox1_Click.
            // Создание графического объекта
            var Графика = CreateGraphics();
            // Создание пера для рисования им фигур
            var Перо = new Pen(Color.Red);
            // Создание кисти для "закрашивания" фигур
            var 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("#F0F0F0"));
            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:
                    Графика.DrawString(
                        "Каждый во что-то верит, но" + "\n" +
                        "жизнь преподносит сюрпризы",
                        Font, Кисть, 10, 100); break;
                case 5:   // — выбран эллипс:
                    Графика.DrawEllipse(Перо, 30, 30, 150, 200); break;
                case 6:   // — выбран закрашенный сектор:
                    Графика.FillPie(Кисть, 20, 50, 150, 150, 0, 45); break;
            }
            Графика.Dispose();
        }
    }
}
```

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 `"#F0F0F0"` (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— так же окрашены 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. Далее оператор `switch` осуществляет множественный выбор графического примитива в зависимости от индекса выбранного элемента списка `SelectedIndex`. Оператор `switch case` передает управление той или иной метке `case`.

Внимание!

Оператор множественного выбора в C# и C++, как говорят программисты, "проваливается" в отличие от Visual Basic, — т. е. управление, зайдя на какую-либо метку `case`, переходит на следующую метку, поэтому приходится использовать оператор `break`.

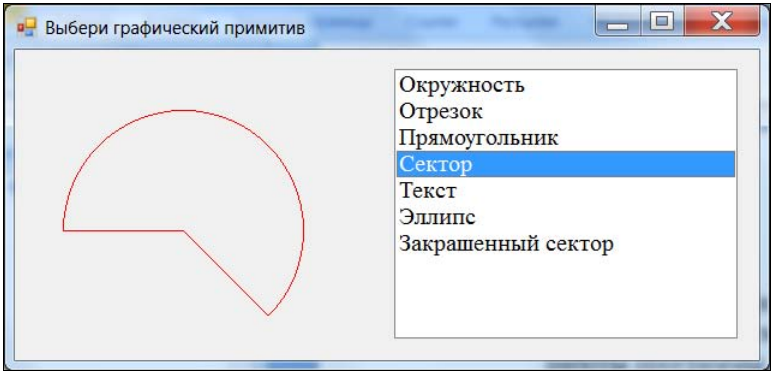


Рис. 5.3. Рисование графического примитива на форме

На рис. 5.3 представлен пример работы программы.

Убедиться в работоспособности программы можно, открыв решение РисФигур.sln в папке РисФигур сопровождающего книгу электронного архива.

Пример 38. Выбор цвета с использованием *ListBox*

В этом примере мы ставим задачу написать программу, которая *меняет* цвет фона формы `BackColor`, перебирая константы цвета, предусмотренные в Visual Studio 12, с помощью элемента управления **ListBox**.

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ВыборЦвета1`. На вкладке дизайнера формы из панели элементов **Toolbox** перетащим на форму список **ListBox**. На вкладке программного кода **Form1.cs** введем текст, представленный в листинге 5.6.

Листинг 5.6. Выбор цвета с помощью элемента управления *ListBox* (вариант 1)

```
// Программа меняет цвет фона формы BackColor, перебирая константы
// цвета, предусмотренные в Visual Studio 12, 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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Получаем массив строк имен цветов из перечисления KnownColor
            var ВсеЦвета = Enum.GetNames(typeof(KnownColor));
            // В VB: Dim ВсеЦвета = [Enum].GetNames(GetType(KnownColor))
            listBox1.Items.Clear();
            // В список элементов добавляем имена всех цветов:
            listBox1.Items.AddRange(ВсеЦвета);
            // Сортировать записи в алфавитном порядке
            listBox1.Sorted = true;
        }
        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
```

```
        // Цвет Transparent является "прозрачным":
        if (listBox1.Text == "Transparent")return;
        this.BackColor = Color.FromName(listBox1.Text);
        this.Text = "Цвет: " + listBox1.Text;
        }
    }
}
```

Как видно из программного кода, при обработке события загрузки формы, 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 (клавишей <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 12 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. 5.4 приведен пример работы программы. Как можно видеть, пользователь выбрал цвет **Control**, который соответствует цвету формы по умолчанию.

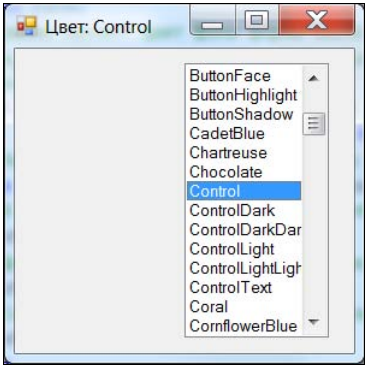


Рис. 5.4. Закраска формы выбранным цветом

Эту программу можно написать более элегантно с использованием цикла `foreach` при заполнении списка формы именами всех цветов. Такую новую программу назовем `ВыборЦвета2`, ее программный код представлен в листинге 5.7.

Листинг 5.7. Выбор цвета с помощью элемента управления *ListBox* (вариант 2)

```
// Программа меняет цвет фона формы
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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Получаем массив строк имен цветов из перечисления KnownColor.
            // Enum.GetNames возвращает массив имен констант
            // в указанном перечислении.
            // Удаление всех элементов из коллекции:
            listBox1.Items.Clear();
            // Добавляем имена всех цветов в список listBox1:
            foreach (String Цвет in Enum.GetNames(typeof(KnownColor)))
                if (Цвет != "Transparent") listBox1.Items.Add(Цвет);
            // Цвет Transparent является "прозрачным"
            // Сортируем все цвета в списке в алфавитном порядке:
            listBox1.Sorted = true;
        }
        private void listBox1_SelectedIndexChanged(
                                        object sender, EventArgs e)
        {
            // Обработка события изменения выбранного
            // индекса в списке listBox1:
            this.BackColor = Color.FromName(listBox1.Text);
            // Надпись в строке заголовка формы:
            this.Text = "Цвет: " + listBox1.Text;
        }
    }
}
```

Как видно из программного кода, во втором варианте цикл `foreach` обеспечивает заполнение списка элементов именами цветов в строковом представлении — кроме цвета `Transparent`, поэтому теперь его даже не надо "отсеивать" в процедуре обработки события изменения выбранного индекса.

Пояснение

В структуре `Color` содержится 167 констант (146 цветов). Вообще говоря, в Visual Studio 12 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 эле-

ментарной 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, `#F0F0F0` (этот цвет соответствует цвету `Control`). Этот цвет мы и подаем 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 сопровождающего книгу электронного архива.

Пример 39. Экранная форма с треугольником прозрачности

Используя графические возможности системы программирования Visual Studio 12, можно добиться самых невероятных, экзотических эффектов. В этом примере мы продемонстрируем возможность программирования экранной формы, в которой будет размещен прозрачный треугольник. Такой треугольник выглядит, как отверстие в форме, через которое видны другие приложения, запущенные в данный момент на вашем компьютере.

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Назовем программу `ПрозрачныйТреугольник`. Из контекстного меню проектируемой формы перейдем на панель свойств **Properties** объекта **Form1**. Щелкнув на значке молнии, символизирующем события формы, найдем событие перерисовки элемента управления **Form1_Paint** и щелкнем по нему двойным щелчком. После чего система откроет для нас вкладку программного кода **Form1.cs** со сгенерированным пустым обработчиком события **Form1_Paint**. Далее введем текст программы, представленный в листинге 5.8.

Листинг 5.8. Программирование экранной формы с треугольником прозрачности

```
// Программирование экранной формы, в которой размещен
// прозрачный треугольник
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ПрозрачныйТреугольник
```

```
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            // Событие перерисовки экранной формы:
            this.ClientSize = new Size(240, 200);
            // Устанавливаем вершины треугольника:
            var p1 = new Point(20, 20);
            var p2 = new Point(225, 66);
            var p3 = new Point(80, 185);
            // Инициализируем массив точек:
            Point[] Точки = { p1, p2, p3 };
            // Закрашиваем этот треугольник цветом ControlDark:
            e.Graphics.FillPolygon(new SolidBrush(
                                SystemColors.ControlDark), Точки);
            // Цвет ControlDark задаем прозрачным:
            this.TransparencyKey = SystemColors.ControlDark;
        }
    }
}
```

При обработке события перерисовки экранной формы назначим три вершины треугольника, инициализируя массив точек. Объект `Graphics` получим из аргумента события `e`. Воспользуемся методом `FillPolygon` для рисования закрашенного многоугольника. Цвет закрашивания может быть любым, однако этот же цвет необходимо назначить в качестве прозрачного в свойстве формы `TransparencyKey`.

Результат работы программы продемонстрирован на рис. 5.5. Здесь форма с треугольником прозрачности показана на фоне текста данной книги, отображаемой в MS Word.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ПрозрачныйТреугольник сопровождающего книгу электронного архива.

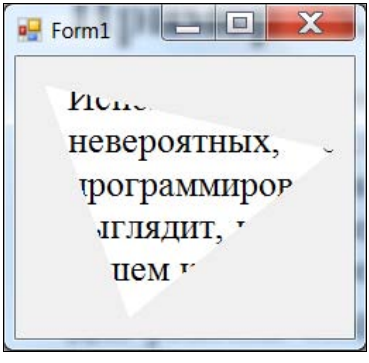


Рис. 5.5. Экранная форма с прозрачным треугольником внутри

Пример 40. Печать графических примитивов

Здесь 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 программы запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ПечатьЭллипса`. Далее из панели элементов **Toolbox** в форму перенесем элемент управления **PrintDocument**. Из контекстного меню проектируемой формы перейдем на панель свойств **Properties** объекта **printDocument1**. Щелкнув на значке молнии, символизирующем события формы, найдем событие **PrintPage** и щелкнем по нему двойным щелчком. После чего система откроет для нас вкладку программного кода **Form1.cs** со сгенерированным пустым обработчиком события **PrintPage**. 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;
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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            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
        var Перо = new Pen(Color.Red);
        e.Graphics.DrawEllipse(Перо, new Rectangle(200, 250, 300, 200));
        // или e.Graphics.DrawEllipse(Перо, 50, 50, 150, 150);
        }
    }
}
```

Как видно из программного кода, с целью максимального упрощения программы для генерации события `PrintPage` при обработке события загрузки формы 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 сопровождающего книгу электронного архива.

Пример 41. Печать BMP-файла

В этом примере программа выводит на печать графический файл формата BMP. На логическом диске D: заранее подготовим графический файл формата BMP и назовем его D:\pic.bmp. Этот файл будет распечатываться на принтере программой, которую мы напишем.

Итак, запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ПечатьBMPфайла`. Теперь добавим в форму из панели элементов **Toolbox** командную кнопку **Button** и объект **PrintDocument**. Программный код представлен в листинге 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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = @"Печать файла d:\pic.bmp";
            button1.Text = "Печать";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Пользователь щелкнул на кнопке
            try
            {
                printDocument1.Print();
            }
```

```
            catch (Exception Ситуация)
            {
                MessageBox.Show("Ошибка печати на принтере\n",
                                                    Ситуация.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(@"D:\pic.bmp"),
                                    e.Graphics.VisibleClipBounds);
            // Следует ли распечатывать следующую страницу?
            e.HasMorePages = false;
        }
    }
}
```

Как видно из программного кода, 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 сопровождающего книгу электронного архива.

Пример 42. Создание JPG-файла "на лету" и вывод его отображения в форму

Часто бывает удобно сначала создать изображение, что-либо на нем нарисовать, например, отобразить сегодняшний курс рубля или график какой-нибудь зависимости, актуальный именно на сегодня, затем это изображение записать на диск и вывести его в форму для удобного использования в ходе текущей работы. Таким графиком может быть, скажем, график продаж продукции по месяцам, по неделям, по дням.

Задача, решаемая в этом примере, состоит в следующем. Имеем экранную форму и кнопку на ней, при щелчке указателем мыши на кнопке нужно создать изображение и методами класса `Graphics` вывести на это изображение текстовую строку, представляющую текущую дату. С целью демонстрации возможностей методов `Graphics` следует развернуть данную строку на некоторый угол относительно горизонта. Далее — сохранить рисунок в текущий каталог и вывести его отображение в форму.

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Дадим проекту имя: `Создать_JPG`. Из панели **Toolbox** перетащим на форму элементы управления **Button и PictureBox**. Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 5.11.

Листинг 5.11. Создание изображения, запись его на диск в формате JPG и вывод отображения файла в форму

```
// Программа формирует изображение методами класса Graphics, записывает
// его на диск в формате JPG-файла и выводит его отображение в
// экранную форму
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace Создать_JPG
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            button1.Text = "Показать дату";
        }
        private void button1_Click(object sender, EventArgs e)
        {
                    pictureBox1.Size = new Size(250, 40);
        // Создаем точечное изображение размером 250x40 точек
        // с глубиной цвета 24
        var Рисунок = new Bitmap(250, 40, System.Drawing.
                          Imaging.PixelFormat.Format24bppRgb);
        // Создаем новый объект класса Graphics из изображения РАСТР
        var Графика = Graphics.FromImage(Рисунок);
        // Теперь становятся доступными методы класса Graphics!
        // Заливка поверхности цветом формы:
        Графика.Clear(Color.FromName("Control"));
        // Вывод в строку полной даты:
        var 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);
        // Сохраняем изображение в файле risunok.jpg:
        Рисунок.Save("risunok.jpg",
                System.Drawing.Imaging.ImageFormat.Jpeg);
        // Задаем стиль границ рисунка:
        pictureBox1.BorderStyle = BorderStyle.None; // FixedSingle
        // Загружаем рисунок из файла:
        pictureBox1.Image = Image.FromFile("risunok.jpg");
```

```
        // Освобождение ресурсов:
        Рисунок.Dispose(); Графика.Dispose();
        // Эквивалент C++: delete Графика
        }
    }
}
```

Как видно из программного кода, при обработке события "щелчок на кнопке" создаем точечное изображение заданного 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. Методом `Save` сохраняем файл изображения в текущей папке в формате JPEG. Далее элементу управления `pictureBox1` указываем путь к файлу изображения.

На рис. 5.6 показан пример работы программы.

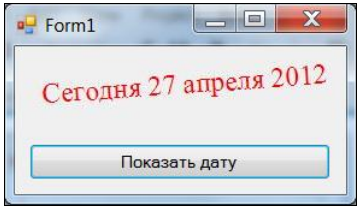


Рис. 5.6. Вывод в форму изображения, сформированного программно

Убедиться в работоспособности программы, рассмотренной в данном разделе, можно, открыв соответствующее решение в папке Создать_JPG сопровождающего книгу электронного архива.

Пример 43. Смена выведенного изображения с помощью обновления формы

При проектировании сценария диалога с пользователем часто встречается необходимость дать возможность пользователю что-либо изменять в выведенном на экранную форму изображении. Программная реализация такой возможности осуществляется с помощью перерисовки области элемента управления, здесь принимает участие метод **Invalidate**.

Конкретная постановка задачи состоит в следующем. В экранную форму выводим текстовую строку. Нажатие на командную кнопку заставляет текстовую строку разворачиваться на некоторый угол.

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Дадим проекту имя: `СменаИзображения`. Из панели **Toolbox** перетащим на форму элементы управления **Button и Panel**. Вообще говоря, элемент **Panel** предназначен для группировки на нем коллекций элементов управления. Однако элемент **Panel** часто используют также для вывода на нем графических изображений.

Нам понадобятся три пустых обработчика событий. Для их создания на вкладке дизайнера формы **Form.cs[Конструктор]** в контекстном меню проектируемой формы выберем команду **Свойства**. В окне свойств щелкнем на пиктограмме молнии (События) и, выбирая в раскрывающемся меню объекты **Form1**, **Panel1** и **Button1**, щелкнем двойным щелчком, соответственно, на событиях **Load**, **Paint** и **Click**. В результате получим три пустых обработчика событий. Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 5.12.

Листинг 5.12. Программа меняет выведенное в форму изображение, используя метод *Invalidate*

```
// Программа выводит в панель рисования текстовую строку. При щелчке
// на командной кнопке происходит перерисовка изображения с разворотом
// текстовой строки
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace СменаИзображения
{
    public partial class Form1 : Form
    {
        // Две внешние переменные:
        Graphics Графика; Single Угол;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Смена изображения";
            button1.Text = "Развернуть";
            Угол = 0.0F;
            panel1.Location = new Point(12, 12);
            panel1.Size = new System.Drawing.Size(240, 200);
        }
        private void panel1_Paint(object sender, PaintEventArgs e)
        {
            // Событие Paint происходит при необходимости
            // перерисовки изображения:
            panel1.BackColor = Color.AliceBlue;
            panel1.Cursor = Cursors.Cross;
            // Задаем поверхность для рисования из аргумента события e:
            Графика = e.Graphics;
            // С помощью смещения задаем центр вращения:
            Графика.TranslateTransform(120.0F, 100.0F);
            // Поворот:
            Графика.RotateTransform(Угол);
```

```
            // Вывод текстовой строки полужирным и одновременно
            // наклонным шрифтом:
            Графика.DrawString("Весело живём!",
                    new Font("Comic Sans MS", 12,
                    FontStyle.Bold ^ FontStyle.Italic),
                    Brushes.Red, -70, -15);
            // Здесь -70, -15 — координаты левого верхнего угла
            // создаваемого текста
            Графика.Dispose();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Разворот на угол в градусах:
            Угол = Угол + 30.0F;
            // Обновить панель рисования:
            panel1.Invalidate();
        }
    }
}
```

Как видно из программного кода, непосредственное рисование изображения происходит при обработке события `panel1_Paint`. Именно здесь мы выводим текстовую строку, задаем центр вращения и разворачиваем ее на некоторый угол. Значение угла регулируем при обработке события "щелчок на командной кнопке". Можно отследить отладчиком и убедиться, что процесс перерисовки изображения происходит довольно часто. Мы программно также можем инициировать перерисовку, используя метод `Invalidate`, что мы и запрограммировали при обработке события `button1_Click`.

Графическая система координат является левой, т. е. ось X направлена вправо, а ось Y — вниз. Поскольку углы и в левой, и в правой системах координат измеряются от направления оси X в сторону оси Y, при каждом щелчке указателем мыши на кнопке текстовая строка разворачивается по часовой стрелке.

Пример работы программы представлен на рис. 5.7.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке СменаИзображения сопровождающего книгу электронного архива.

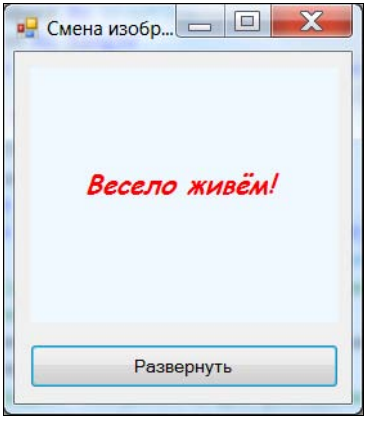


Рис. 5.7. Перерисовка изображения при каждом щелчке мыши

Пример 44. Рисование в форме указателем мыши

В этом 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**, предназначенную для очистки формы.

Вначале надо создать форму с командной кнопкой, как мы это делали прежде. Для этого запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Дадим проекту имя: РисМышью. Из панели **Toolbox** перетащим на форму элемент управления **Button**. Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 5.13.

Листинг 5.13. Рисование в 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
    {
        // Булева переменная Рисовать_ли дает разрешение на рисование:
        Boolean Рисовать_ли;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Рисую мышью в форме";
            button1.Text = "Стереть";
            Рисовать_ли = false; // в начале — не рисовать
        }
        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)
                var 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
                Графика.Dispose(); // Эквивалент C++: delete Графика
            }
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Методы очистки формы:
            var Графика = CreateGraphics();
            Графика.Clear(this.BackColor);
            // Графика.Clear(SystemColors.Control);
            // Графика.Clear(Color.FromName("Control"));
            // Графика.Clear(Color.FromKnownColor(KnownColor.Control));
            // Графика.Clear(ColorTranslator.FromHtml("#F0F0F0"));
            // this.Refresh(); // Этот метод также перерисовывает форму
        }
    }
}
```

Здесь в начале программы объявлена переменная `Рисовать_ли` логического типа (`Boolean`) 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 = frue`). При перемещении мыши с нажатой кнопкой программа создает графический объект `Graphics` пространства имен `System.Drawing`, используя метод `CreateGraphics()`, и рисует прямоугольник `FillRectangle()` размером 10×10 пикселов, заполненный красным цветом. `e.X`, `e.Y` — координаты указателя мыши, которые так же являются координатами левого верхнего угла прямоугольника.

На рис. 5.8 приведен пример рисования в форме. Чтобы стереть все нарисованное в форме, следует нажать кнопку **Стереть**. При этом вызывается метод `Refresh()`, предназначенный для *перерисовывания* формы.

В комментариях 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`.

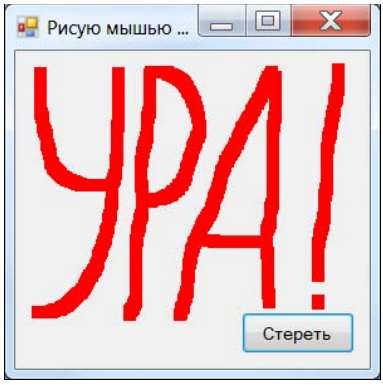


Рис. 5.8. Рисование указателем мыши в форме

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 оператора "точка" (`.`) 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 также представлены в комментарии. Цвет `#F0F0F0` 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 сопровождающего книгу электронного архива.

Пример 45. Управление сплайном Безье

Система Visual Studio 12 предоставляет возможность построения фундаментального сплайна и сплайна Безье. В принципе, программирование сплайна Безье сводится к подаче на вход соответствующей функции `DrawBezier` узловых точек, через которые проходит сплайн, а также дополнительных контрольных точек. В сплайне Безье *с помощью контрольных точек управляют* его кривизной, т. е. *его формой*. На рис. 5.9 точки P_0

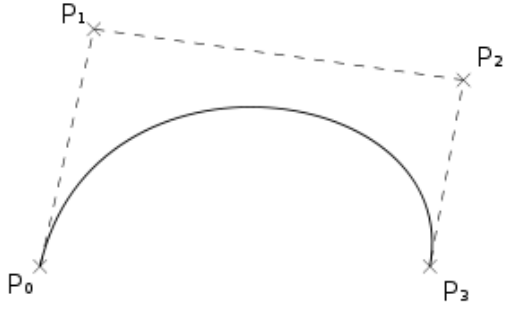


Рис. 5.9. Контрольные точки P_1 и P_2 определяют форму кривой

и P_3 являются узловыми, а через контрольные точки P_1 и P_2 проходят касательные к кривой в узловых точках.

Перемещая контрольные точки P_1 и P_2 в ту или другую сторону, мы можем управлять формой (кривизной) сплайна. Глядя на рисунок, можно себе представить, как при некотором положении контрольных точек добиться того, что сплайн будет изгибаться в обе стороны, т. е. между двумя точками появятся положительная кривизна и отрицательная, а между ними — точка перегиба.

> ***Примечание***
>
> В русскоязычной документации на сайте **www.msdn.com** эти точки называют "контрольными", а в англоязычной — "control points". Но "control" переводится еще и как "управлять". Таким образом, эти точки лучше называть "управляющими", поскольку они управляют формой кривой.

Наша задача в этом примере состоит в следующем. На экранной форме расположим две узловые точки, а обе управляющие точки соединим в одну. Научимся перемещать мышью эту единую управляющую точку и тем самым изменять форму сплайна.

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Назовем этот проект `Spline`. Нам понадобятся пять пустых обработчиков разных событий. Их легко получить, если на вкладке конструктора формы **Form1.cs[Design]** через контекстное меню перейти на вкладку свойств, щелкнуть на значке молнии (**Events**), а затем последовательно выбирать события, указанные в листинге 5.14.

Листинг 5.14. Управление формой кривой с помощью одной контрольной точки

```
// Программа строит сплайн Безье по двум узловым точкам, а две
// контрольные (управляющие) точки совмещены в одну. Эта одна
// управляющая точка отображается в форме в виде красного
// прямоугольника. Перемещая указателем мыши управляющую точку,
// мы регулируем форму сплайна (кривой)
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace Spline
{
    public partial class Form1 : Form
    {
        PointF[] МассивТочек;
        // Запрещение управлять формой кривой:
        Boolean Управлять;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
```

```
{
    Управлять = false;
    this.Text = "Управление сплайном Безье";
    // Начальная узловая точка:
    var p0 = new PointF(50, 50);
    // Две контрольные (управляющие) точки, мы их совместили в одну:
    var p1 = new PointF(125, 125);
    var p2 = new PointF(125, 125);
    // Конечная узловая точка:
    var p3 = new PointF(200.0F, 200.0F);
    МассивТочек = new PointF[] { p0, p1, p2, p3 };
}
private void Form1_Paint(object sender, PaintEventArgs e)
{
    // Задаем поверхность для рисования из аргумента события e:
    var Графика = e.Graphics;
    var Перо = new Pen(Color.Blue, 3);
    // Рисуем начальную и конечную узловые точки диаметром 4 пиксела:
    Графика.DrawEllipse(Перо,
        МассивТочек[0].X — 2, МассивТочек[0].Y — 2, 4.0F, 4.0F);
    Графика.DrawEllipse(Перо,
        МассивТочек[3].X — 2, МассивТочек[3].Y — 2, 4.0F, 4.0F);
    // Одна управляющая точка в виде прямоугольника красного цвета:
    Перо.Color = Color.Red;
    Графика.DrawRectangle(Перо, МассивТочек[1].X — 2,
                МассивТочек[1].Y — 2, 4.0F, 4.0F);
    Перо.Color = Color.Blue;
    // Рисуем сплайн Безье:
    Графика.DrawBeziers(Перо, МассивТочек);
    // Освобождение ресурсов:
    Графика.Dispose();
}
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    // Событие перемещения указателя мыши в области экранной формы.
    // Если указатель мыши расположен над управляющей точкой
    if (Math.Abs(e.X — МассивТочек[1].X) < 4.0F &&
        Math.Abs(e.Y — МассивТочек[1].Y) < 4.0F &&
                Управлять == true)
    { // и при этом нажата кнопка мыши,
        // то меняем координаты управляющей точки:
        МассивТочек[1].X = e.X;
        МассивТочек[1].Y = e.Y;
        МассивТочек[2].X = e.X;
        МассивТочек[2].Y = e.Y;
        // и обновляем (перерисовываем) форму:
        this.Invalidate();
    }
}
```

```
        private void Form1_MouseUp(object sender, MouseEventArgs e)
        {
            // Если кнопку мыши отпустили, то запрещаем
            // управлять формой кривой:
            Управлять = false;
        }
        private void Form1_MouseDown(object sender, MouseEventArgs e)
        {
            // Если нажата кнопка мыши, то разрешаем
            // управлять формой кривой:
            Управлять = true;
        }
    }
}
```

В программном коде массив из четырех точек (две узловые и две контрольные) объявляем вне процедур класса `Form1`, чтобы этот массив был "виден" из всех этих процедур. Начальные значения элементов массива задаем при обработке события загрузки формы.

При обработке события перерисовки формы рисуем начальную и конечную узловые точки, одну управляющую точку в виде прямоугольника красного цвета и, наконец, сплайн Безье по текущим значениям массива четырех точек. Булева переменная `Управлять`, объявленная также вне всех процедур, принимает значение `true` при нажатой кнопке мыши, а если кнопку мыши отпустили, то этой переменной присваиваем значение `false`. При обработке события перемещения указателя мыши в пределах экранной формы происходит проверка — если указатель мыши расположен вблизи управляющей точки, и при этом нажата кнопка мыши, то координаты управляющей точки назначаем равными текущим координатам положения мыши (их берем из аргументной переменной `e`) и перерисовываем (т. е. обновляем) форму.

Пример работы программы показан на рис. 5.10.

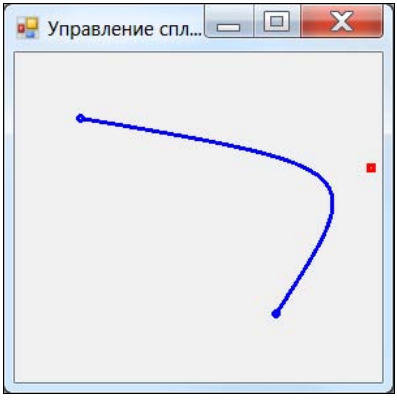


Рис. 5.10. Меняем форму кривой, перемещая мышью красный прямоугольник

Убедиться в работоспособности программы можно, открыв решение Spline.sln в папке Spline сопровождающего книгу электронного архива.

Пример 46. Построение графика методами класса *Graphics*

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 способом можно построить *любой график для других прикладных целей*.

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем График. Перенесем из панели элементов **Toolbox** в проектируемую форму элемент управления **PictureBox** (графическое поле) и командную кнопку **Button**. Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 5.15.

Листинг 5.15. Программа для вывода графика в форму

```
// Программа рисует график объемов 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
    {
        String[] Months;
        int[] Sales;
        Graphics Графика;
        // Далее, создаем объект Bitmap, который имеет
        // тот же размер и разрешение, что и PictureBox
        Bitmap Растр;
        int ОтступСлева, ОтступСправа, ОтступСнизу, ОтступСверху;
        int ДлинаВертОси, ДлинаГоризОси, YГоризОси, Xmax, XНачЭпюры;
        // Шаг градуировки по горизонтальной и вертикальной осям:
        Double ГоризШаг;
        int ВертШаг;
        int i;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Исходные данные для построения графика (т. е. исходные
            // точки):
            Months = new String[] {"Янв", "Фев", "Март", "Апр", "Май",
                                   "Июнь", "Июль", "Авг", "Сент",
                                   "Окт", "Нояб", "Дек"};
```

```
        Sales = new int[] {335, 414, 572, 629, 750, 931,
                           753, 599, 422, 301, 245, 155};
        ОтступСлева = 35; ОтступСправа = 15;
        ОтступСнизу = 20; ОтступСверху = 10;
        this.Text = " Построение графика ";
        button1.Text = "Нарисовать график";
        this.ClientSize = new Size(593, 319);
        pictureBox1.Size = new Size(569, 242);
        Растр = new Bitmap(pictureBox1.Width,
                           pictureBox1.Height,
                           pictureBox1.CreateGraphics());
        // Нарисовать границу pictureBox1 для отладки:
        // pictureBox1.BorderStyle = BorderStyle.FixedSingle;
        YГоризОси = pictureBox1.Height — ОтступСнизу;
        Xmax = pictureBox1.Width — ОтступСправа;
        ДлинаГоризОси = pictureBox1.Width -
                                    (ОтступСлева + ОтступСправа);
        ДлинаВертОси = YГоризОси — ОтступСверху;
        ГоризШаг = Convert.ToDouble(ДлинаГоризОси / Sales.Length);
        ВертШаг = Convert.ToInt32(ДлинаВертОси / 10);
        XНачЭпюры = ОтступСлева + 30;
    }
    private void button1_Click(object sender, EventArgs e)
    {
        // Последовательно вызываем следующие процедуры:
        Графика = Graphics.FromImage(Растр);
        РисуемОси();
        РисуемГоризЛинии();
        РисуемВертЛинии();
        РисованиеЭпюры();
        pictureBox1.Image = Растр;
        // Освобождаем ресурсы, используемые объектом класса Graphics:
        Графика.Dispose();
    }
    void РисуемОси()
    {
        var 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);
        var Шрифт = new Font("Arial", 8);
        for (this.i = 1; i <= 10; i++)
        {
            // Рисуем "усики" на вертикальной координатной оси:
            int Y = YГоризОси — i * ВертШаг;
            Графика.DrawLine(Перо, ОтступСлева — 4, Y, ОтступСлева, Y);
```

```
            // Подписываем значения продаж через каждые 100 единиц:
            Графика.DrawString((i * 100).ToString(), Шрифт,
                                                Brushes.Black, 0, Y — 5);
        } // конец цикла по i
        // Подписываем месяцы на горизонтальной оси:
        for (this.i = 0; i <= Months.Length — 1; i++)
        {
            Графика.DrawString(Months[i], Шрифт, Brushes.Black,
               (int)(ОтступСлева + 18 + i * ГоризШаг), (YГоризОси + 4));
        }
    }
    void РисуемГоризЛинии()
    {
        var ТонкоеПеро = new Pen(Color.LightGray, 1);
        for (this.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();
    }
    void РисуемВертЛинии()
    {
        // Рисуем вертикальные почти "прозрачные" линии
        var ТонкоеПеро = new Pen(Color.Bisque, 1);
        for (this.i = 0; i <= Months.Length — 1; i++)
        {
            int X = XНачЭпюры + Convert.ToInt32(ГоризШаг * i);
            Графика.DrawLine(ТонкоеПеро, X, ОтступСверху, X, YГоризОси — 4);
        }
        ТонкоеПеро.Dispose();
    }
    void РисованиеЭпюры()
    {
        var ВертМасштаб = Convert.ToDouble(ДлинаВертОси / 1000.0);
        // или  var  ВертМасштаб = (Double)ДлинаВертОси / 1000.0;
        // Значения ординат на экране:
        var Y = new int[Sales.Length];
        // Значения абсцисс на экране:
        var X = new int[Sales.Length];
        for (this.i = 0; i <= Sales.Length — 1; i++)
        {   // Вычисляем графические координаты точек:
            Y[i] = YГоризОси — Convert.ToInt32(Sales[i] * ВертМасштаб);
            // Отнимаем значения продаж, поскольку ось Y экрана
            // направлена вниз
            X[i] = XНачЭпюры + Convert.ToInt32(ГоризШаг * i);
        }
```

```
            // Рисуем первый кружок:
            var Перо = new Pen(Color.Blue, 3);
            Графика.DrawEllipse(Перо, X[0] — 2, Y[0] — 2, 4, 4);
            for (this.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** (графическое поле), а затем, вызывая соответствующие процедуры, поэтапно рисуем координатные оси, сетку из горизонтальных и вертикальных линий и непосредственно эпюру (рис. 5.11).

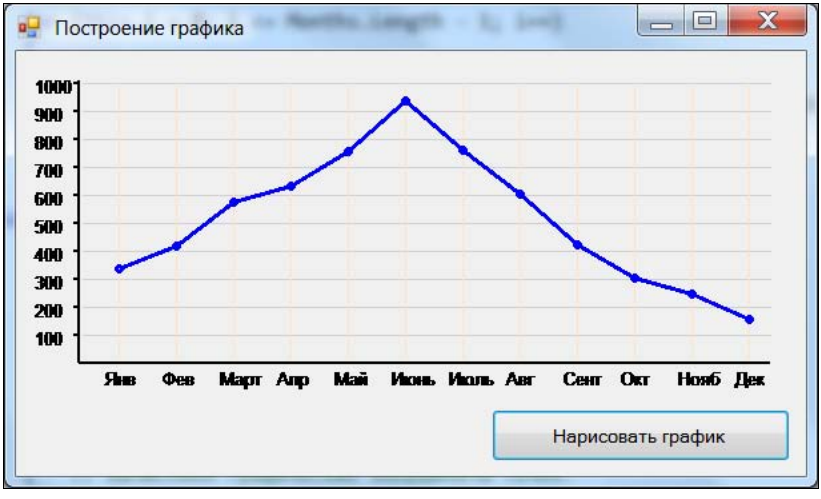


Рис. 5.11. График объемов продаж по месяцам

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 назва-

ний 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 значение максимальной абсциссы (см. рис. 5.11), 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.

Убедиться в работоспособности программы можно, открыв решение График.sln в папке График сопровождающего книгу электронного архива.

ГЛАВА 6

Управление буфером обмена с данными в текстовом и графическом форматах

Пример 47. Буфер обмена с данными в текстовом формате

Напишем программу для управления буфером обмена с данными в текстовом формате. Эта программа будет записывать какой-либо текст в буфер обмена (БО), а затем извлекать этот текст из БО.

Запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `БуферОбменаТХТ`. В форме создадим два текстовых поля, а также две командные кнопки под этими полями. Одну кнопку назовем **Записать в БО**, а другую — **Извлечь из БО**.

Чтобы записать в БО какой-либо текст, нужно набрать его в верхнем поле, выделить (с помощью клавиш управления курсором при нажатой клавише <Shift>), а затем нажать кнопку **Записать в БО**. Нажимая эту кнопку, мы как бы моделируем комбинацию клавиш <Ctrl>+<C>.

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

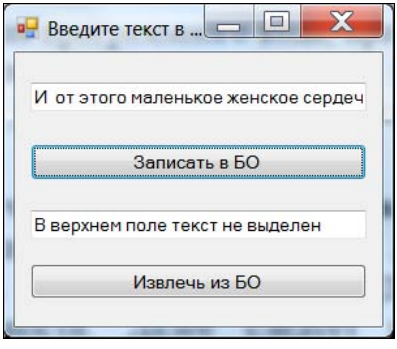


Рис. 6.1. Запись текстовых данных в буфер обмена и их чтение

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 БуферОбменаТХТ
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            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
            var 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 // иначе
                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, нажав кнопку **Извлечь из БО**, пользователь нашей программы в нижнем поле должен увидеть содержимое буфера обмена. Для этого объявляем объектную переменную `Получатель`. Это объект-получатель из буфера обмена. Эта переменная сознательно названа по-русски для большей выразительности. Далее следует проверка — в текстовом ли формате (`DataFormat.Text`) данные, представленные в буфере обмена. Если формат текстовый, то в текстовое поле `TextBox2` записываем содержимое буфера обмена, используя функцию `ToString`. Эта функция конвертирует строковую часть объекта в строковую переменную.

Убедиться в работоспособности программы можно, открыв решение БуферОбменаTXT.sln в папке БуферОбменаTXT сопровождающего книгу электронного архива.

Напоминание

Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 48. Элемент управления *PictureBox*. Буфер обмена с растровыми данными

Напишем программу, которая оперирует буфером обмена, когда тот содержит изображение. Здесь мы воспользуемся элементом управления **PictureBox** (графическое поле), способным 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 в экранной форме программы *анимацию*, т. е. движущееся изображение. Пример отображения такой GIF-анимации вы можете посмотреть, открыв решение PictureBoxGif.sln в папке PictureBoxGif сопровождающего книгу электронного архива.

Примечание

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 редактора, например того же Paint, MS Office Picture Manager, ACDSee или др. Затем, нажав кнопку **Извлечь из БО** нашей программы, получить в форме содержимое БО.

Для создания данной программы запустим Visual Studio 12 и в окне **New Project** выберем приложение шаблона **Windows Forms Application Visual C#**. Новую программу

назовем `БуферОбменаBitmap`. Затем из панели **Toolbox** в проектируемую форму перетащим элементы управления **PictureBox** (графическое поле) и кнопку **Button**. Текст программы приведен в листинге 6.2.

Листинг 6.2. Обмен графических данных 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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Содержимое БО:";
            button1.Text = "Извлечь из БО";
            // Размеры графического окна:
            pictureBox1.Size = new Size(184, 142);
            // Записать в PictureBox изображение из файла:
            try
            {
                pictureBox1.Image = Image.FromFile(
                      System.IO.Directory.
                      GetCurrentDirectory() + @"\poryv.png");
            }
            catch (System.IO.FileNotFoundException Ситуация)
            {  // Обработка исключительной ситуации:
                MessageBox.Show(
                    "Нет такого файла\n" + Ситуация.Message, "Ошибка",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                button1.Enabled = false;
                return;
            }
            // 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
            var 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 обработке события загрузки формы в графическое поле `pictureBox1` записываем какой-нибудь файл, например poryv.png. Функция `GetCurrentDirectory()` возвращает полный путь текущей папки. Далее по команде

```
Clipboard.SetDataObject(PictureBox1.Image)
```

происходит запись содержимого графического поля в буфер обмена.

Теперь пользователь может проверить содержимое БО с помощью какого-либо графического редактора, например Paint. Далее он 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. Пример работы программы представлен на рис. 6.2.

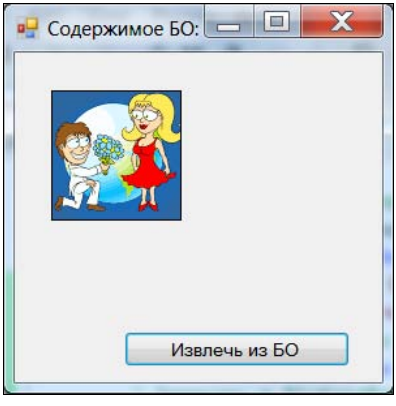


Рис. 6.2. Извлечение изображения из буфера обмена

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 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 сопровождающего книгу электронного архива.

Пример 49. Имитация нажатия комбинации клавиш <Alt>+<PrintScreen>

Как известно, при нажатии клавиши <PrintScreen> происходит копирование изображения экрана в буфер обмена, т. е. получаем так называемый *screen shot* — *моментальный снимок экрана* (другое название — *screen capture*). После извлечения из БО графической копии экрана в любом графическом редакторе (например, Paint) эту графическую копию можно редактировать.

При нажатии комбинации клавиш <Alt>+<PrintScreen> в буфер обмена копируется не весь экран, а *только активное окно*. Для имитации нажатия сочетания клавиш <Alt>+<PrintScreen> современная система программирования Visual Studio в пространстве имен `System.Windows.Forms` имеет класс `SendKeys`, который предоставляет методы для отправки приложению сообщений о нажатиях клавиш, в том числе и нашей комбинации клавиш <Alt>+<PrintScreen>.

Сейчас мы напишем программу, которая будет имитировать нажатие искомых клавиш. Для этого запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `AltPrintScreen`. Программа сознательно максимально упрощена, чтобы более выразительно показать ее суть. Здесь в форме имеется только командная кнопка. При обработке события "щелчок на кнопке" будем имитировать нажатие комбинации клавиш <Alt>+<PrintScreen> методом `Send` класса `SendKeys`. При этом форму можно как угодно менять — растягивать по ширине и по высоте, но после нажатия кнопки в форме изображение формы запишется в буфер обмена. Следует убедиться в этом — запустить MS Paint и извлечь графическую копию формы из буфера обмена. Текст программы представлен в листинге 6.3.

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

```
// Программная имитация нажатия клавиш <Alt>+<PrintScreen>
// методом Send класса SendKeys
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace AltPrintScreen
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Имитируем нажатие <Alt>+<PrintScreen>";
            button1.Text = "методом Send класса SendKeys";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Метод SendKeys.Send посылает сообщение активному приложению
            // о нажатии клавиш <Alt>+<PrintScreen>
            SendKeys.Send("%{PRTSC}");
            // Так можно получить символьное представление клавиши:
            // var S = Keys.PrintScreen.ToString();
        }
    }
}
```

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

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

Убедиться в работоспособности программы можно, открыв решение AltPrintScreen.sln в папке AltPrintScreen сопровождающего книгу электронного архива.

Пример 50. 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. Для этой цели запустим Visual Studio 12 и в окне **New Project** выберем приложение шаблона **Windows Forms Application Visual C#**. Новую программу назовем `БуферОбменаSaveBMP`. Перенесем из панели элементов **Toolbox** в проектируемую форму командную кнопку **Button**. 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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            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
            var 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
                var Объект = Получатель.GetData(DataFormats.Bitmap);
                Растр = (Bitmap)Объект;
                // Сохранить изображение в файле Clip.bmp
                Растр.Save(@"D:\Clip.BMP");
                // this.Text = "Сохранено в файле D:\Clip.BMP":
                // button1.Text = "Еще записать?";
                MessageBox.Show(
                @"Изображение из БО записано в файл D:\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");
        }
    }
}
```

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 излишне запутать читателя и упростить программу, автор не стал организовывать запись файла в диалоге. При необходимости читатель сделает это самостоятельно, воспользовавшись элемен-

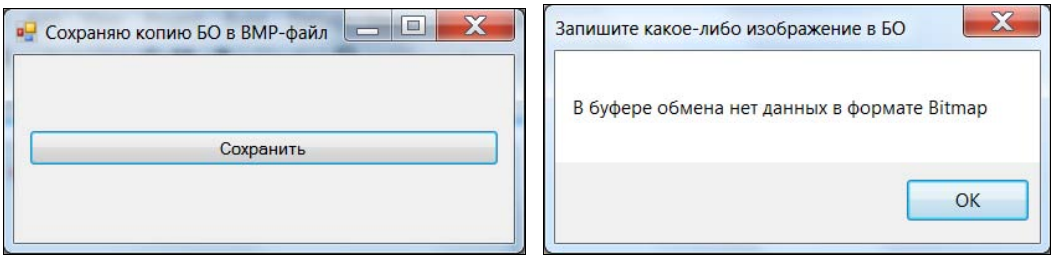


Рис. 6.3. Запись изображения из буфера обмена в файл

том управления **SaveFileDialog**. Пример работы данной программы представлен на рис. 6.3.

Убедиться в работоспособности программы можно, открыв решение БуферОбменаSaveBMP.sln в папке БуферОбменаSaveBMP сопровождающего книгу электронного архива.

Пример 51. 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 офшорного 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.

Наш пример является частью упомянутой задачи, сформулируем его 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 запустим Visual Studio 12 и в окне **New Project** выберем приложение шаблона **Windows Forms Application Visual C#**. Новую программу назовем `ПростоTimer`. Затем в проектируемую 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 ПростоTimer
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            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 пример работы программы.

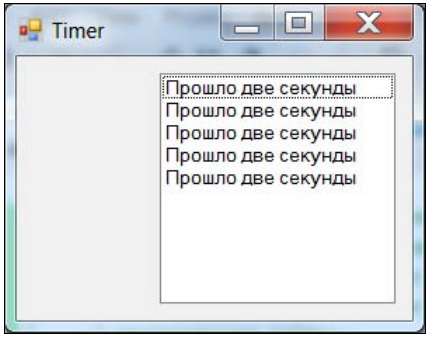


Рис. 6.4. Вывод сообщения

Убедиться в работоспособности программы можно, открыв решение ПростоTimer.sln в папке ПростоTimer сопровождающего книгу электронного архива.

Пример 52. Запись в файлы текущих состояний экрана каждые пять секунд

Как уже говорилось в предыдущем примере, работа с таймером — довольно распространенная задача. Напишем очередную программу в следующей постановке: после запуска программы через каждые пять секунд снимается текущее состояние экрана и записывается в файлы Pic1.BMP, ..., Pic5.BMP.

Для решения этой задачи мы имеем уже весь инструментарий, он был рассмотрен в предыдущих примерах. А именно задание интервала времени в две секунды с 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 запустим Visual Studio 12 и в окне **New Project** выберем приложение шаблона **Windows Forms Application Visual C#**. Новую программу назовем `SaveСкриншотКаждые5сек`. Затем из панели элементов перетаскиваем в проектируемую форму элементы управления **Timer** и **Button**. Текст программы приведен в листинге 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; // счет секунд
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            i = 0; // счет секунд
            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);
```

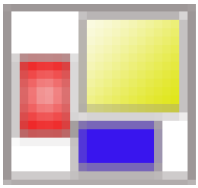
```
            if (i >= 28) { timer1.Enabled = false; this.Close(); }
            Single Остаток = i % 5;
            // В VB целочисленный остаток от деления вычисляет оператор Mod
            if (Остаток != 0) return;
            // Имитируем нажатие клавиш <Alt>+<PrintScreen>
            SendKeys.Send("%{PRTSC}");
            // Объявление объекта-получателя из буфера обмена
            var Получатель = Clipboard.GetDataObject();
            // 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
                var Объект = Получатель.GetData(DataFormats.Bitmap);
                var Растр = (Bitmap)Объект;
                // Сохранить изображение из переменной Растр
                // в файл D:\Pic1, D:\Pic2, D:\Pic3, ...
                var ИмяФайла = String.Format(@"D:\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 из программного кода, структура программы включает в себя обработку события "щелчок на кнопке" `button1_Click` и обработку события `timer1_Tick`. В начале программы задаем переменную `i`, которая считает прошедшие секунды. При щелчке на кнопке **Пуск** задаем интервал времени `Interval`, равный 1000 миллисекунд, т. е. одной секунде. Далее даем команду таймеру начать отсчет времени `timer1.Enabled = True`, и через каждую секунду наступает событие `timer1_Tick()`, т. е. управление переходит этой процедуре.

При обработке события `timer1_Tick` наращиваем значение переменной `i`, которая ведет счет секунд после старта таймера. Выражение `i % 5` вычисляет целочисленный остаток после деления первого числового выражения на второе. Понятно, что если число `i` будет кратно пяти, то этот остаток будет равен нулю, и только в этом случае будет происходить имитация нажатия клавиш <Alt>+<PrintScreen> и запись содержимого буфера обмена в файл.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке SaveСкриншотКаждые5сек сопровождающего книгу электронного архива.

ГЛАВА 7



Ввод и вывод табличных данных. Решение системы уравнений

Пример 53. Формирование таблицы. Функция *String.Format*

При создании инженерных, экономических и других приложений часто задача сводится к вводу данных, расчету (обработке введенных данных), а затем — к 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 `Имена` и массив телефонов `Тлф`. 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.

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ТаблТхт`. Затем из панели элементов перетащим в проектируемую форму элементы управления текстовое поле **TextBox** со свойством `Multiline = true` и меню **MenuStrip**. С пунктами меню **Файл**, **Показать таблицу в Блокноте** и **Выход**. 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
// раскрывающегося меню, с помощью которого пользователь выводит
// сформированную таблицу в Блокнот с целью последующего редактирования
// и вывода на печать
```

```
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ТаблТхт
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Escape-последовательность для перехода на новую строку:
            const String НС = "\r\n"; // Новая строка
            this.textBox1.Multiline = true;
            this.ClientSize = new Size(438, 272);
            this.textBox1.Size = new Size(415, 229);
            this.Text = "Формирование таблицы";
            String[] Имена = {
                      "Андрей — раб", "Света-Х", "ЖЭК",
                      "Справка по тел", "Александр Степанович",
                      "Мама — дом", "Карапузова Таня",
                      "Погода сегодня", "Театр Браво"};
            String[] Тлф = {
                      "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 Font("Courier New", 9.0F);
            textBox1.Text = "ТАБЛИЦА ТЕЛЕФОНОВ" + НС + НС;
            var i = 0;
            // Считают, что цикл foreach/Next — смотрится читабельнее,
            // чем обычный цикл:
            foreach (var Имя in Имена)
            {
                textBox1.Text += String.Format(
                              "{0, -21} {1, -21}" + НС, Имя, Тлф[i]);
                i = i + 1;
            }
            // Или формирование таблицы с помощью обычного пошагового цикла:
            // for (i = 0; i <= 8; i++)
            //    textBox1.Text += String.Format(
            //            "{0, -21} {1, -21}", Имена[i], Тлф[i]) + НС;
            textBox1.Text += НС + "ПРИМЕЧАНИЕ:" +
               НС + "для корректного отображения таблицы" +
               НС + "в Блокноте укажите шрифт Courier New";
```

```
            // Запись таблицы в текстовый файл D:\Table.txt.
            // Создание экземпляра StreamWriter для записи в файл
            var Писатель = new System.IO.
            StreamWriter(@"D:\Table.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", @"D:\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 пишущей 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 `Имена[i]`, а единица во вторых фигурных скобках — вставить вместо единицы строку `Тлф[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. Использование константы `НС` означает, что следует начать текст с новой строки. Внешний вид таблицы в текстовом поле формы показан на рис. 7.1.

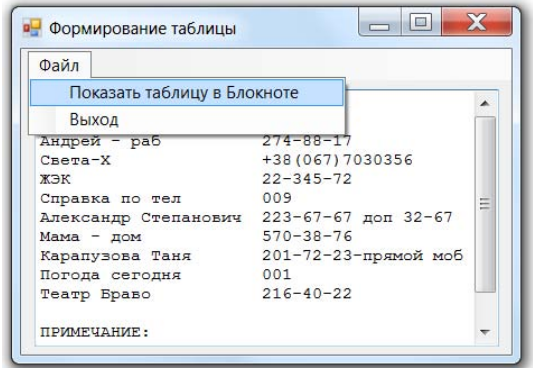


Рис. 7.1. Таблица из двух массивов в текстовом поле

После формирования всех строк `textBox1.Text` записываем их в текстовый файл D:\Table.txt через `StreamWriter` с кодовой таблицей Windows 1251. Подробное обсуждение этого фрагмента программы читатель может посмотреть в *примере 26 (см. главу 4)*.

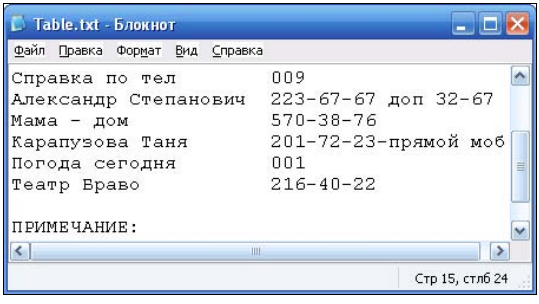


Рис. 7.2. Просмотр созданной таблицы в Блокноте

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 D:\Table.txt (см. рис. 7.2).

Убедиться в работоспособности программы можно, открыв решение ТаблTxt.sln в папке ТаблTxt сопровождающего книгу электронного архива.

> ***Напоминание***
>
> Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 54. Форматирование *Double*-переменных в виде таблицы. Вывод таблицы на печать. Поток *StringReader*

Эта 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. Таблицу следует назвать "Каталог координат". Координаты в таблице должны быть округлены до двух 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, так же как и в предыдущем примере, запустим Visual Studio 12 и в окне **New Project** выберем приложение шаблона **Windows Forms Application Visual C#**. Новую программу назовем `ТаблТхtPrint`. Затем в проектируемой форме с помощью панели элементов управления **Toolbox** создадим текстовое поле **TextBox**, выберем элементы управления **MenuStip** и **PrintDocument**. На вкладке **Design** подготовим пункты меню **Файл**, **Печать** и **Выход** так, как показано на рис. 7.3. Чтобы растянуть текстовое поле на всю форму, в свойстве `Multiline` укажем `true` (разрешим введение множества строк).

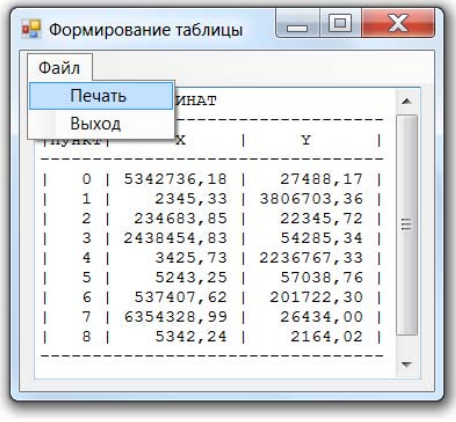


Рис. 7.3. Вывод таблицы в текстовое поле

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.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ТаблТхtPrint
{
    public partial class Form1 : Form
    {
        System.IO.StringReader Читатель; // — внешняя переменная
```

```
public Form1()
{
    InitializeComponent();
}
private void Form1_Load(object sender, EventArgs e)
{
    // Escape-последовательность для перехода на новую строку:
    const String НС = "\r\n"; // Новая строка
    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 Font("Courier New", 9.0F);
    textBox1.Text = "КАТАЛОГ КООРДИНАТ" + НС;
    textBox1.Text += "---------------------------------" + НС;
    textBox1.Text += "|Пункт|      X     |     Y      |" + НС;
    textBox1.Text += "---------------------------------" + НС;
    // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
    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]) + НС;
    textBox1.Text += "---------------------------------" + НС;
}
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 Ситуация)
            {
                MessageBox.Show(Ситуация.Message);
            }
        }
        private void выходToolStripMenuItem_Click(
                                    Object sender, EventArgs e)
        {
                    // Выход из программы
            this.Close();
        }
    }
}
```

Как видно из программного кода, 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 33 (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.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ТаблTxtPrint сопровождающего книгу электронного архива.

Пример 55. Вывод таблицы в Internet Explorer

Рассмотрим 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) веб-страниц Internet Explorer.

Пояснение

HTML (HyperText Markup Language, язык гипертекстовой разметки) — специальные инструкции браузеру, с помощью которых создаются веб-страницы. То есть веб-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 веб-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-X<td>+38(067)7030356
<tr><td>ЖЭК<td>22-345-72
<tr><td>Справка по тел<td>009
</table>
```

Примечание

Вообще говоря, если строго придерживаться правил языка HTML, то приведенный фрагмент следует оформить тегами `<HTML></HTML>` и `<BODY></BODY>`, заключить теги `<title></title>` в теги `<HEAD></HEAD>` и т. д. Однако современные браузеры понимают и такую разметку, которая приведена в листинге 7.3. В нашем примере даже не указаны закрывающие теги 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, сохранить с расширением htm и открыть в браузере Internet Explorer, Mozilla Firefox или каком-либо другом. Также поучительно открыть этот файл в редакторе HTML Microsoft Office SharePoint Designer, Microsoft Office FrontPage или другом и проследить, как меняется HTML-форматирование при тех или иных изменениях, сделанных в таблице.

Итак, запустим Visual Studio 12 и в окне **New Project** выберем приложение шаблона **Windows Forms Application Visual C#**. Новый проект назовем `Табл_HTM`. Наша программа имеет данные, уже знакомые читателю из *примера 54*. Эти данные находятся в двух массивах: `Имена[]` и `Тлф[]`. На основании этих двух массивов программа формирует таблицу в формате HTML, т. е. создает текстовый файл (в нашей программе он называется D:\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 веб-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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Таблица в формате HTML";
            String[] Имена = {
                     "Андрей — раб", "Света-Х", "ЖЭК",
                     "Справка по тел", "Александр Степанович", "Мама — дом",
                     "Карапузова Таня", "Погода сегодня", "Театр Браво"};
            String[] Тлф = {
                     "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 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}",
                                            Имена[i], Тлф[i]) + "\r\n";
            text = text + "</table>";
            // Запись таблицы в текстовый файл D:\Tabl_tel.htm.
            // Создание экземпляра StreamWriter для записи в файл
            var Писатель = new
                  System.IO.StreamWriter(@"D:\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", @"D:\Tabl_tel.htm");
                // Файл HTM можно открывать также с пом MS_WORD:
                // System.Diagnostics.Process.Start(
                //                      "WinWord", @"D:\Tabl_tel.htm");
            }
            catch (Exception Ситуация)
            {
                // Отчет об ошибках
                MessageBox.Show(Ситуация.Message, "Ошибка",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
        }
    }
}
```

Как видно из программного кода, HTML-файл формируется с помощью строковой переменной `text`. Между тегами `<title>` и `</title>` указано название страницы. Это текст, который браузер покажет в строке заголовка 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 хорошо известный читателю `String.Format`, т. е. вместо фигурной скобки `{0}` подставляется элемент массива `Имена[i]`, а вместо `{1}` — `Тлф[i]`.

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

Результат работы программы приведен на рис. 7.4.

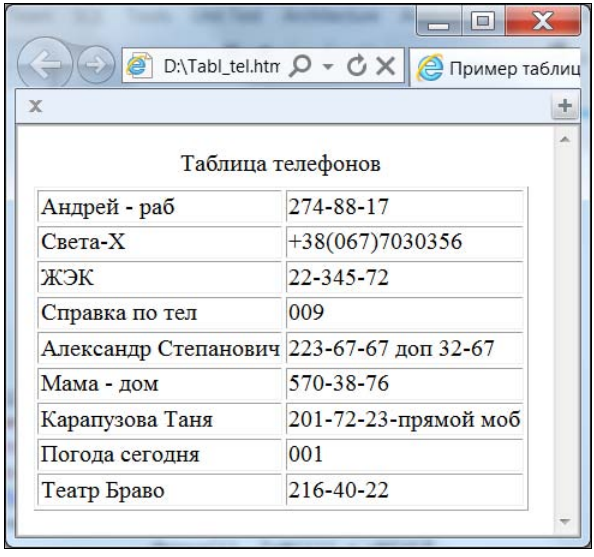


Рис. 7.4. Вывод таблицы в браузер

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

```
Diagnostics.Process.Start("WinWord", "D:\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 сопровождающего книгу электронного архива.

Пример 56. 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.

Запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новый проект назовем `ТаблGrid`. В дизайнере формы из панели

элементов перетащим элемент управления **DataGridView** (Сетка данных). В данной программе два уже знакомых читателю массива `Имена[]` и `Тлф[]` выводятся на сетку данных **DataGridView**. Для максимального упрощения программы формируем таблицу при обработке события загрузки формы. 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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Формирование таблицы";
            String[] Имена = {
                     "Андрей — раб", "Света-Х", "ЖЭК",
                     "Справка по тел", "Ломачинская Светлана", "Мама — дом",
                     "Карапузова Таня", "Погода сегодня", "Театр Браво"};
            String[] Тлф = {
                     "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"
            var 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(Имена[i], Тлф[i]);
            // Для сетки данных указываем источник данных
            dataGridView1.DataSource = Таблица;
        }
    }
}
```

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

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

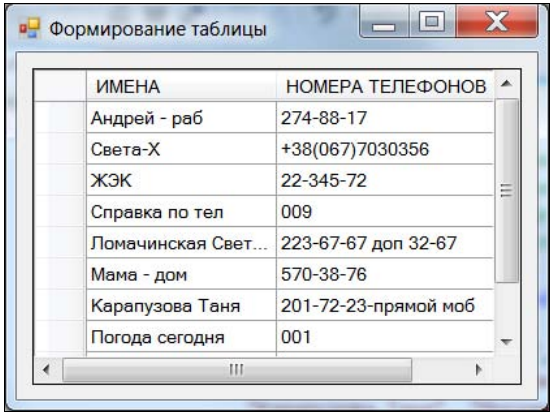


Рис. 7.5. Формирование таблицы в элементе **DataGridView**

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 сопровождающего книгу электронного архива.

Пример 57. Отображение хэш-таблицы с помощью элемента *DataGridView*

Структура данных, называемая *хэш-таблицей*, представляет собой таблицу из двух столбцов, один столбец содержит ключи, а второй — значения. То есть каждая строка в этой таблице образует пару "ключ — значение". Имея ключ, в хэш-таблице легко и быстро найти значение. Хэш-таблицу можно назвать таблицей соответствий. Простейшим примером хэш-таблицы является таблица телефонов, которая участвовала в предыдущих примерах, однако там мы программировали ее просто как два массива. Если эти два массива поместить в хэш-таблицу, то ключами в данном случае были бы имена, а значениями — номера телефонов. При этом программирование поиска значения по ключу оказалось бы тривиальной задачей, операция добавления и удаления пары также упростилась бы, поскольку хэш-таблица — это объект, который содержит соответствующие эти и другие методы.

В реальной жизни много разнообразных примеров представления данных в виде хэш-таблиц. 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). Примерами хэш-таблиц могут служить англо-русский и другие словари, а также база доменных имен,

которая доменному имени сопоставляет IP-адрес. По принципу хэш-таблицы организованы объекты `ViewState` и `Session` технологии ASP.NET.

Поставим следующую задачу — сопоставить в хэш-таблице государства в качестве ключей, а их столицы — в качестве значений. Далее, используя элемент управления **DataGridView**, вывести эту хэш-таблицу в форму.

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**, в поле **Name** укажем имя `ХэшGridView`. Перенесем в форму элемент управления **DataGridView** (Сетка данных) из панели **Toolbox**. Далее на вкладке **Form1.cs** введем программный код, приведенный в листинге 7.6.

Листинг 7.6. Вывод хэш-таблицы в экранную форму

```
// В данной программе используется структура данных, называемая
// хэш-таблицей. С ее помощью программа ставит в соответствие
// государствам их столицы. При этом в качестве ключей указываем
// названия государств, а в качестве значений — их столицы. Далее,
// используя элемент управления DataGridView, программа выводит
// эту хэш-таблицу в форму
using System;
using System.Collections;
using System.Data;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ХэшGridView
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Пример хэш-таблицы";
            // Создаем новую хэш-таблицу:
            var Хэш = new Hashtable();
            // Заполнение хэш-таблицы.
            // Можно добавлять записи "ключ — значение" таким образом:
            Хэш["Украина"] = "Киев";
            // А можно добавлять так:
            Хэш.Add("Россия", "Москва");
            // Здесь государство — это ключ, а столица — это значение
            Хэш.Add("Беларусь", "Минск");
            // Создаем обычную таблицу (не хэш):
            var Таблица = new DataTable();
```

```
            // Задаем схему таблицы, заказывая две колонки:
            Таблица.Columns.Add("ГОСУДАРСТВА");
            Таблица.Columns.Add("СТОЛИЦЫ");
            // В цикле заполняем обычную таблицу парами
            // из хэш-таблицы по рядам:
            foreach (DictionaryEntry ОднаПара in Хэш)
                // Здесь структура DictionaryEntry
                // определяет пару "ключ — значение"
                Таблица.Rows.Add(ОднаПара.Key, ОднаПара.Value);
            // Указываем источник данных для DataGridView:
            dataGridView1.DataSource = Таблица;
        }
    }
}
```

Здесь при обработке события загрузки формы 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, так же как и в предыдущем примере, для `dataGridView1` указываем в качестве источника данных заполненный объект `DataTable`. Пример работы данной программы показан на рис. 7.6.

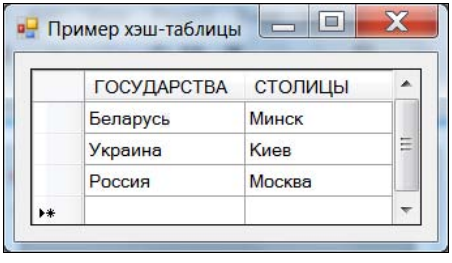


Рис. 7.6. Отображение в форме хэш-таблицы

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 данной программы можно, открыв решение ХэшGridView.sln в папке ХэшGridView сопровождающего книгу электронного архива.

Пример 58. 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. Элемент управления **DataGridView** поддерживает выделение, изменение, удаление, разбиение на страницы и сортировку.

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

Для написания программы запустим Visual Studio 12 и в окне **New Project** выберем приложение шаблона **Windows Forms Application Visual C#**. Новый проект назовем `ТаблВвод`. Далее из панели элементов **Toolbox** перенесем указателем мыши сетку данных **DataGridView** и экранную кнопку **Button**. Текст программы приведен в листинге 7.7.

Листинг 7.7. 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 D:\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;  // Объявление объекта "таблица данных"
        DataSet НаборДанных;  // Объявление объекта "набор данных"
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Почти табличный редактор";
            button1.Text = "Запись";
            Таблица = new DataTable();
            НаборДанных = new DataSet();
            if (System.IO.File.Exists(@"D:\tabl.xml") == true)
            {
                // Если XML-файл ЕСТЬ:
                НаборДанных.ReadXml(@"D:\tabl.xml");
                // Содержимое DataSet в виде строки XML для отладки:
                var СтрокаXML = НаборДанных.GetXml();
                dataGridView1.DataMember = "Название таблицы";
                dataGridView1.DataSource = НаборДанных;
            }
            else
            {
                // 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);
            }
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Сохранить файл tabl.xml:
            Таблица.TableName = "Название таблицы";
            НаборДанных.WriteXml(@"D:\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, 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, 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. 7.7) — событие `button1.Click` — происходит запись XML-файла на диск (`WriteXml`).

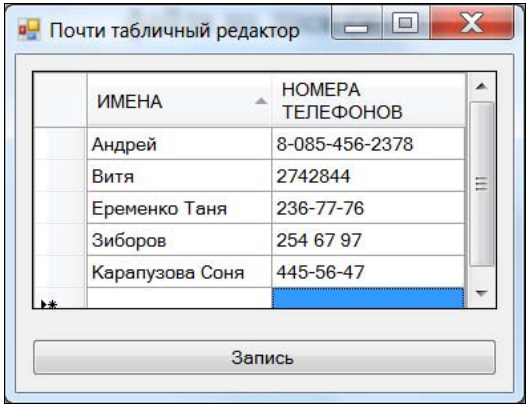


Рис. 7.7. Почти табличный редактор

Пояснение: XML-файлы

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 файл текстовый. 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-документы корректно открываются веб-браузером, программами XML-editor (входит в состав Visual Studio 12), MS Office SharePoint Designer, MS Front Page и др. При этом отступы и разные цвета позволяют выразительно увидеть структуру файла. XML-файлы открываются и табличным редактором MS Excel, причем файл в нем может отобразиться в виде таблицы. На рис. 7.8 приведен образец представления XML-файла в MS Word.

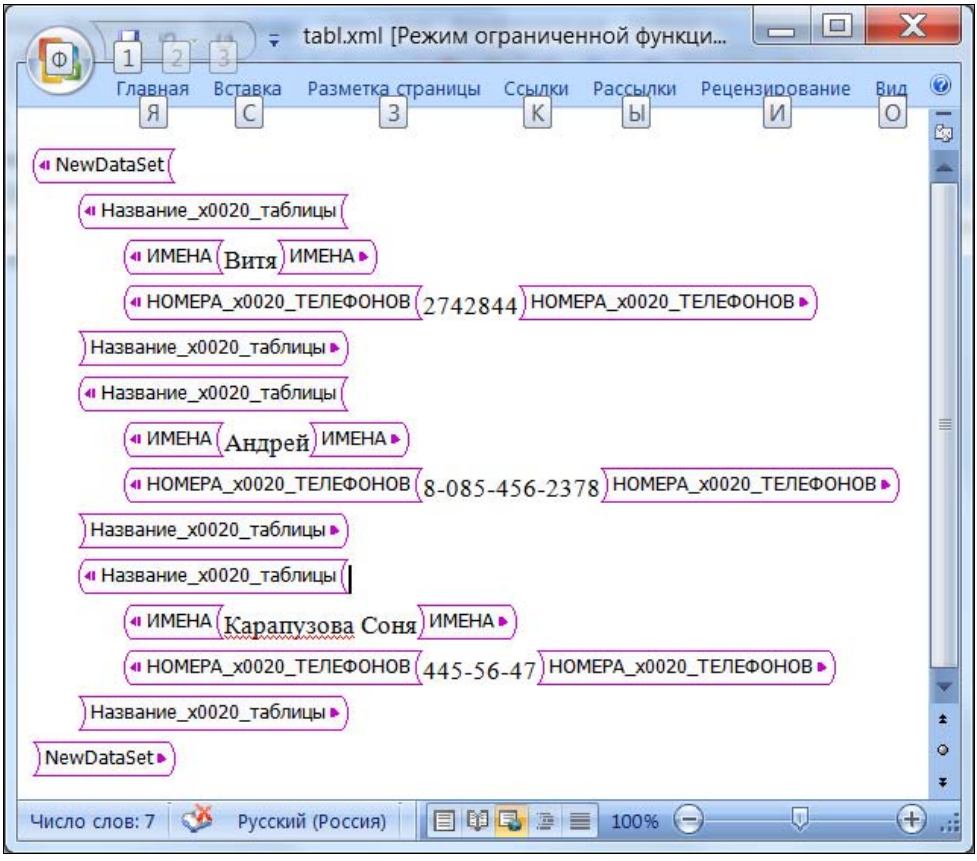


Рис. 7.8. Образец представления XML-файла в MS Word

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

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`.

Убедиться в работоспособности программы можно, открыв решение ТаблВвод.sln в папке ТаблВвод сопровождающего книгу электронного архива.

Пример 59. 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 такой задачи в последнее время редко приводится в книгах по языкам программирования. Предлагаемая 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 **DataGridView** (Сетка данных), 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 коэффициентов должно быть равно количеству неизвестных). Далее происходит непосредственное решение введенной системы методом Гаусса и вывод результатов вычислений с помощью `MessageBox`.

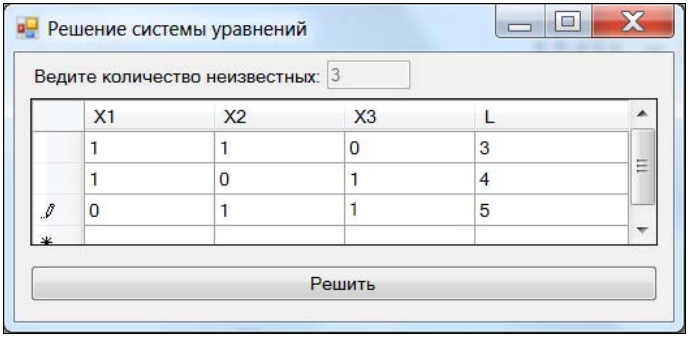


Рис. 7.9. Пример табличного ввода данных

Итак, запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ГауссGrid`. Поскольку пользовательский интерфейс (рис. 7.9) будет включать в себя метку **Label**, текстовое поле **TextBox**, элемент управления **DataGridView** и кнопку **Button**, перенесем названные элементы из панели элементов управления в проектируемую форму. В листинге 7.8 приведен программный код решения задачи.

Листинг 7.8. Решение системы линейных уравнений

```
// Программа для решения системы линейных уравнений. Ввод коэффициентов
// предусмотрен через DataGridView
using System;
using System.Data; // — для DataGridView
using System.Windows.Forms;
// Добавим директиву Globalization:
using System.Globalization;
```

```
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ГауссGrid
{
    public partial class Form1 : Form
    {
        // Данные переменные объявляем вне всех процедур, чтобы
        // они были видны из любой из процедур:
        int n; // — размерность СЛАУ
        DataTable Таблица;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.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 = "Введите количество неизвестных:";
            button1.Text = "Ввести"; // — первоначальная надпись на кнопке
            Таблица = new DataTable();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            int i, j;
            // Матрица коэффициентов линейных уравнений:
            Double[,] A;
            // Вектор свободных членов:
            Double[] L;
            // Признак ввода числовых данных:
            var Число_ли = false;
            var tmp = "временная рабочая переменная";
            if (button1.Text == "Ввести")
                for (; ; )
                {
                    // Бесконечный цикл, пока пользователь не введет
                    // именно число:
                    Число_ли = int.TryParse(textBox1.Text,
                                            NumberStyles.Integer,
                                            NumberFormatInfo.CurrentInfo,
                                            out n);
                    if (Число_ли == false) return;
                    // Задаем другую надпись на кнопке:
                    button1.Text = "Решить";
                    // Теперь уже текстовое поле недоступно:
                    textBox1.Enabled = false;
```

```
        // Теперь уже сетку данных видно:
        dataGridView1.Visible = true;
        dataGridView1.DataSource = Таблица;
        // Создаем "шапку" таблицы
        for (i = 1; i <= n; i++)
        {
            tmp = "X" + Convert.ToString(i);
            Таблица.Columns.Add(new DataColumn(tmp));
        }
        // Колонка правой части системы:
        Таблица.Columns.Add(new DataColumn("L"));
        return;
    } // — конец тела вечного цикла
else  // — button1.Text = "Решить"
{   // Нажали кнопку "Решить"
    // Таблица.Rows.Count — количество рядов
    if (Таблица.Rows.Count != n)
    {
        MessageBox.Show(
            "Количество строк не равно количеству колонок");
        return;
    }
    // Теперь можем определиться с размерностью массивов:
    // Матрица коэффициентов линейных уравнений:
    A = new Double[n, n];
    // Вектор свободных членов:
    L = new Double[n];
    // 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;
        } // — конец тела внутреннего цикла по i
        // Правая часть системы B[j, 0]
        L[j] = ВернутьЧисло(j, i, ref Число_ли);
        if (Число_ли == false) return;
    } // — конец тела внешнего цикла по j
} // — button1.Text = "Решить"
// ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
// Решение системы A*x = L методом Гаусса:
gauss(n, A, ref L);
// L — вектор свободных членов системы, сюда
// же возвращается решение x
var s = "Неизвестные равны:\n";
for (j = 1; j <= n; j++)
{
    tmp = L[j — 1].ToString();
```

```
            s = s + "X" + j.ToString() + " = " + tmp + ";\n";
        } // — конец цикла по j
        MessageBox.Show(s);
    }
    Double ВернутьЧисло(int j, int i,
                            ref Boolean Число_ли)
    {
        // j — номер строки, i — номер столбца
        // Передаем аргумент Число_ли по ссылке
        Double rab; // — рабочая переменная
        var tmp = Таблица.Rows[j][i].ToString();
        Число_ли = Double.TryParse(tmp,
                   NumberStyles.Number,
                   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;
    }
    void gauss(int n, double[,] A, ref double[] LL)
    {
        // n  — размер матрицы
        // A  — матрица коэффициентов линейных уравнений
        // LL — правая часть, сюда же 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
        }
    }
}
```

Как следует из 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 визуализации значений неизвестных. Переменная `s` выводится посредством диалогового окна `MessageBox` (рис. 7.10).

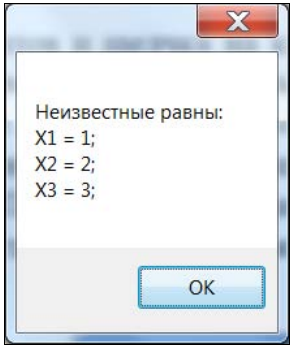


Рис. 7.10. Вывод неизвестных значений в диалоговое окно

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 про-

грамму на решение систем большой размерности. Если количество неизвестных равно одному, то программа также нормально функционирует.

Убедиться в работоспособности программы можно, открыв решение GaussGrid.sln в папке GaussGrid сопровождающего книгу электронного архива.

Пример 60. Организация связанных таблиц

Представляем вашему вниманию пример программирования двух связанных таблиц. Одна таблица содержит сведения о клиентах-заказчиках. В нашем случае это название организации и контактная информация (рис. 7.11), хотя в реальной жизни в эту таблицу включают гораздо больше колонок данных. Вторая таблица представляет данные о заказах, она может содержать номер (идентификацию) заказа, его объем, организацию-заказчика, сроки выполнения, адрес доставки и прочие необходимые сведения. В нашем случае с целью компактного изложения эта таблица будет содержать только номер заказа, его объем и название организации-заказчика (рис. 7.12).

Обе таблицы связываем таким образом, чтобы из родительской таблицы можно было уточнять сведения дочерней таблицы. Связь осуществляем с помощью одинаковой ко-

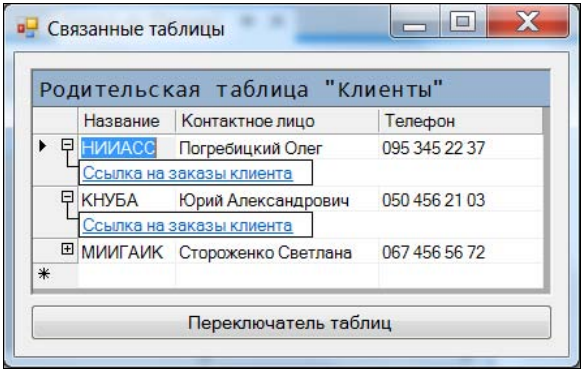


Рис. 7.11. Родительская таблица "Клиенты" содержит ссылки на строки дочерней таблицы

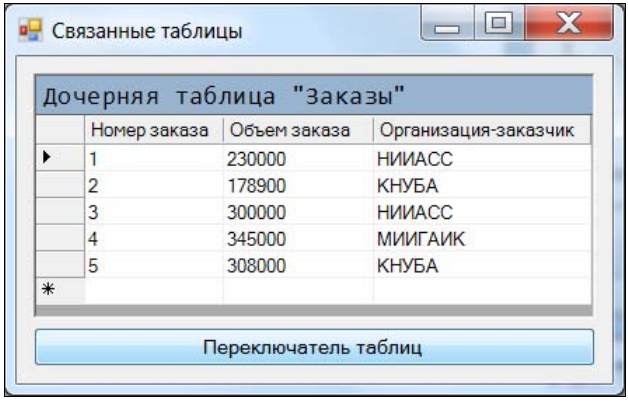


Рис. 7.12. Содержание дочерней таблицы "Заказы"

лонки (столбца), содержащей название организации. При этом родительская таблица "Клиенты" (см. рис. 7.11) в заголовке каждой строки будет отображать знак +, раскрывая который щелчком мыши можно уточнять сведения из дочерней таблицы "Заказы" (см. рис. 7.12).

Для решения задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Программу назовем `СвязанныеТаблицы`.

> ***Пояснение: элемент DataGrid***
>
> Обычно для отображения таблиц пользуются элементом управления **DataGridView**. Это сравнительно новый элемент управления, он появился в Visual Studio примерно с 2007 года. До 2007 года применялся элемент управления **DataGrid**. Элемент управления **DataGridView** поддерживает ряд простых и сложных функций, отсутствующих в элементе управления **DataGrid**. Кроме того, архитектура элемента управления **DataGridView** упрощает его расширение и настройку по сравнению с **DataGrid**. Почти для всех целей следует использовать элемент управления **DataGridView**. Единственная функция, доступная в элементе управления **DataGrid** и недоступная в **DataGridView**, — иерархическое отображение данных из двух связанных таблиц в едином элементе управления. Для отображения данных из двух связанных таблиц требуется два элемента управления **DataGridView**. Поэтому для отображения двух связанных таблиц *более технологичным вариантом будет использовать* именно элемент **DataGrid**.

Итак, в панели **ToolBox** в узле **Данные** нам необходим элемент **DataGrid**. Однако поскольку он устарел, элемент **DataGrid** в панели **ToolBox** отсутствует, и нам нужно его самостоятельно добавить в панель элементов. Для этого щелкаем правой кнопкой мыши в пределах панели элементов управления и в открывшемся контекстном меню выбираем пункт **Choose Items**. Далее на вкладке **Компоненты .NET Framework** устанавливаем флажок возле элемента **DataGrid** пространства имен **System.Windows.Forms**. После щелчка на кнопке **OK** указанный элемент появится в узле **General** панели **ToolBox**. Теперь мы можем перенести элемент **DataGrid** в проектируемую форму. Кроме этого нам понадобится элемент **Button** (как же без него) — его также переносим в нашу форму.

В листинге 7.9 приведен программный код решения задачи.

Листинг 7.9. Программирование двух связанных таблиц

```
// Программа создает таблицу с данными о клиентах (названия организаций,
// контакты) и таблицу с данными о заказах (номер заказа, его объем,
// организация-заказчик). Между таблицами устанавливается связь
// посредством одинаковых столбцов (названий организаций). При этом
// таблица с данными о клиентах будет родительской, а таблица с данными
// о заказах — дочерней. Каждая строка родительской таблицы отображается
// со знаком "плюс". При щелчке на знаке "плюс" открывается узел,
// содержащий ссылку на дочернюю таблицу
using System;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace СвязанныеТаблицы
```

```
{
    public partial class Form1 : Form
    {
        // Внешние переменные:
        Boolean ПоказатьКлиентов;
        DataSet НаборДанных;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            button1.Text = "Переключатель таблиц"; button1.TabIndex = 0;
            this.Text = "Связанные таблицы"; ПоказатьКлиентов = true;
            // Р О Д И Т Е Л Ь С К А Я  Т А Б Л И Ц А  "К Л И Е Н Т Ы":
            var Таблица = new DataTable("Клиенты");
            // Создаем первый столбец в таблице:
            var Столбец = new DataColumn("Название организации");
            Столбец.ReadOnly = true; // — его нельзя модифицировать
            // Значения в каждой строке столбца должны быть уникальными:
            Столбец.Unique = true;
            // Добавляем этот столбец в таблицу:
            Таблица.Columns.Add(Столбец);
            // Добавляем второй и третий столбец в таблицу:
            Таблица.Columns.Add("Контактное лицо");
            Таблица.Columns.Add("Телефон");
            // Создаем набор данных:
            НаборДанных = new DataSet();
            // Добавляем таблицу "Клиенты" в набор данных:
            НаборДанных.Tables.Add(Таблица);
            // Заполняем строки (ряды) в таблице:
            Таблица.Rows.Add(
                "НИИАСС", "Погребицкий Олег", "095 345 22 37");
            Таблица.Rows.Add(
                "КНУБА", "Юрий Александрович", "050 456 21 03");
            Таблица.Rows.Add(
                "МИИГАИК", "Стороженко Светлана", "067 456 56 72");
            // Д О Ч Е Р Н Я Я  Т А Б Л И Ц А  "З А К А З Ы":
            Таблица = new DataTable("Заказы");
            // Создаем первый столбец в таблице "Заказы":
            var Столбец1 = new DataColumn("Номер заказа");
            Столбец1.DataType = Type.GetType("System.Int32");
            Столбец1.ReadOnly = true; Столбец1.Unique = true;
            Таблица.Columns.Add(Столбец1);
            // Добавляем второй и третий столбец в таблицу:
            Таблица.Columns.Add("Объем заказа");
            Таблица.Columns.Add("Организация-заказчик");
            НаборДанных.Tables.Add(Таблица);
```

```
            // Заполняем строки (ряды) в таблице:
            Таблица.Rows.Add(1, "230000", "НИИАСС");
            Таблица.Rows.Add(2, "178900", "КНУБА");
            Таблица.Rows.Add(3, "300000", "НИИАСС");
            Таблица.Rows.Add(4, "345000", "МИИГАИК");
            Таблица.Rows.Add(5, "308000", "КНУБА");
            // С О З Д А Н И Е  С В Я З Е Й  М Е Ж Д У  Т А Б Л И Ц А М И:
            // Связываем одинаковые столбцы в двух таблицах:
            var Родитель = НаборДанных.
             Tables["Клиенты"].Columns["Название организации"];
            var Дочь = НаборДанных.
                        Tables["Заказы"].Columns["Организация-заказчик"];
            var Связь1 = new DataRelation(
                        "Ссылка на заказы клиента", Родитель, Дочь);
            // В родительской таблице значения в связываемом столбце
            // должны быть уникальными, а в дочерней — нет.
            НаборДанных.Tables["Заказы"].ParentRelations.Add(Связь1);
            // И С Т О Ч Н И К  Д А Н Н Ы Х  Д Л Я  D A T A G R I D:
            dataGrid1.SetDataBinding(НаборДанных, "Клиенты");
            dataGrid1.CaptionText = "Родительская таблица \"Клиенты\"";
            dataGrid1.CaptionFont = new Font("Consolas", 11);
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Показываем либо "таблицу Клиенты", либо таблицу "Заказы":
            ПоказатьКлиентов = !ПоказатьКлиентов;
            if (ПоказатьКлиентов == true)
            {
                dataGrid1.SetDataBinding(НаборДанных, "Клиенты");
                dataGrid1.CaptionText =
                                "Родительская таблица \"Клиенты\"";
            }
            else
            {
                dataGrid1.SetDataBinding(НаборДанных, "Заказы");
                dataGrid1.CaptionText =
                                   "Дочерняя таблица \"Заказы\"";
            }
        }
    }
}
```

При обработке события загрузки формы создаем родительскую таблицу "Клиенты", а также дочернюю таблицу "Заказы". В этой главе мы уже не раз создавали таблицу класса `DataTable`, поэтому не будем повторяться в комментариях. Укажем только, что создавая первый столбец "Название организации", мы установили уникальность каждого значения в этом столбце — таково требование к родительской таблице. В родительской таблице значения в связываемом столбце не должны повторяться. В то время как к связываемому столбцу дочерней таблицы такого ограничения нет.

Создавая связь между таблицами (объект класса `DataRelation`), указываем одинаковые столбцы в двух таблицах. В набор данных класса `DataSet` мы добавили две таблицы класса `DataTable` и объект связи между таблицами класса `DataRelation`. При подключении набора `DataSet` к какой-либо базе данных (SQL Server, MS Access, Oracle и пр.) можно добавлять записи (строки) в обе таблицы, их модифицировать (изменять), организовывать различные SQL-запросы к данным и более эффективно их использовать.

Обрабатывая событие "щелчок на командной кнопке **Button**", мы организовали переключение между родительской и дочерней таблицами. Примеры работы программы показаны на рис. 7.11–7.13.

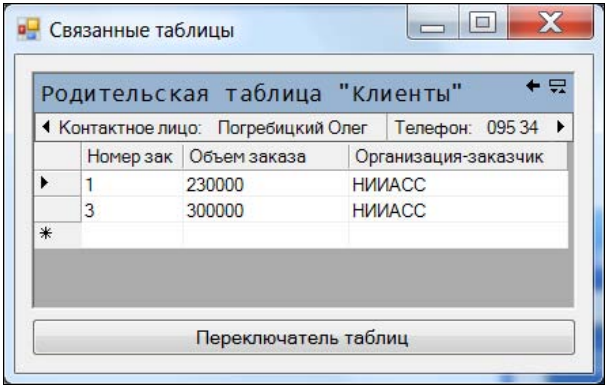


Рис. 7.13. Результат ссылки на заказы организации НИИАСС

Убедиться в работоспособности программы можно, открыв решение СвязанныеТаблицы.sln в папке СвязанныеТаблицы сопровождающего книгу электронного архива.

Пример 61. Построение графика по табличным данным с использованием элемента *Chart*

Уважаемый читатель, давайте посмотрим на панель элементов управления **Toolbox**. Среди множества элементов найдем элемент **Chart**. Он предназначен для вывода в экранную форму графика (диаграммы). Очень удобно строить этот график по табличным данным, представленным в виде объекта класса `DataTable`.

В этом примере решим следующую задачу. Даны сведения об объемах продаж за пять месяцев. Требуется наглядно представить эти данные на графике в виде гистограммы.

Для решения задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Программу назовем `ГрафикChart`. Из панели **ToolBox** (узел **Data**) перенесем в проектируемую экранную форму элементы: диаграмму **Chart** и сетку данных **DataGridView**. Для наглядности мы хотим, чтобы при щелчке мышью на графике он менял свой внешний вид, а именно, чтобы столбики гистограммы отображались в виде цилиндров. Для этого нам понадобится пустой обработчик события "щелчок на диаграмме **Chart**". Для получения этого пустого обработчика, используя контекстное меню элемента **Chart** в конструкторе проектируемой формы, перейдем к свойствам этого элемента. Здесь в окне **Properties** щелкнем на значке молнии

(События) и в списке возможных событий выберем событие **Click**. Двойным щелчком щелкнув на отображении этого события, мы попадаем на вкладку программного кода с готовым пустым обработчиком для него. В листинге 7.10 приведен программный код решения задачи.

Листинг 7.10. Вывод в форму графика посредством элемента управления *Chart*

```
// Программа, используя элементы управления Chart и DataGridView, выводит
// график (диаграмму) зависимости объемов продаж от времени по месяцам.
// При этом в качестве источника данных указываем объект класса DataTable
using System;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ГрафикChart
{
    public partial class Form1 : Form
    {
        // Отображать ли столбики гистограммы в виде цилиндров:
        Boolean Цилиндр;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            Цилиндр = false;
            this.Text = "Щелкните на графике";
            var Таблица = new DataTable();
            // В этой таблице заказываем две колонки: "Месяц" и
            // "Объем продаж":
            Таблица.Columns.Add("Месяц", typeof(String));
            // В VB: Таблица.Columns.Add("Месяц", GetType(String))
            // Значения во второй колонке назначаем типа Long:
            Таблица.Columns.Add("Объем продаж", typeof(long));
            // В VB: Таблица.Columns.Add("Объем продаж", GetType(Long))
            // Заполняем первую строку (ряд) в таблице:
            var Ряд = Таблица.NewRow();
            Ряд["Месяц"] = "Май"; Ряд["Объем продаж"] = 15;
            Таблица.Rows.Add(Ряд);
            // Добавляем вторую строку в таблице:
            Ряд = Таблица.NewRow();
            Ряд["Месяц"] = "Июнь"; Ряд["Объем продаж"] = 35;
            Таблица.Rows.Add(Ряд);
            // Добавляем третью строку:
            Ряд = Таблица.NewRow();
            Ряд["Месяц"] = "Июль"; Ряд["Объем продаж"] = 65;
            Таблица.Rows.Add(Ряд);
```

```
            // Добавляем четвертую строку:
            Ряд = Таблица.NewRow();
            Ряд["Месяц"] = "Авг"; Ряд["Объем продаж"] = 85;
            Таблица.Rows.Add(Ряд);
            // Добавляем пятую строку:
            Ряд = Таблица.NewRow();
            Ряд["Месяц"] = "Сент"; Ряд["Объем продаж"] = 71;
            Таблица.Rows.Add(Ряд);
            // Составленную таблицу указываем в качестве источника данных:
            chart1.DataSource = Таблица;
            // На одном графике можно изобразить несколько зависимостей.
            // Например, первая зависимость — объемы продаж по указанным
            // месяцам в 2009 году, а вторая зависимость — продажи по
            // тем же месяцам в 2010 году.
            // В данном графике мы покажем только одну зависимость, данные
            // для отображения этой зависимости назовем "Series1"
            // По горизонтальной оси откладываем названия месяцев:
            chart1.Series["Series1"].XValueMember = "Месяц";
            // А по вертикальной оси откладываем объемы продаж:
            chart1.Series["Series1"].YValueMembers = "Объем продаж";
            // Название графика (диаграммы):
            chart1.Titles.Add("Объемы продаж по месяцам");
            // Задаем тип диаграммы — столбиковая гистограмма:
            chart1.Series["Series1"].ChartType =
                System.Windows.Forms.
                    DataVisualization.Charting.SeriesChartType.Column;
            // Тип диаграммы может быть другим, например: Pie, Line и др.
            chart1.Series["Series1"].Color = Color.Aqua; // то же, что Cyan
            // Легенду на графике не отображаем:
            chart1.Series["Series1"].IsVisibleInLegend = false;
            // Привязка графика к источнику данных:
            chart1.DataBind();
            // Для сетки данных указываем источник данных
            dataGridView1.DataSource = Таблица;
        }
        private void chart1_Click(object sender, EventArgs e)
        {
            // Обработка события "щелчок на графике"
            Цилиндр = !Цилиндр;
            // Изображение столбиков гистограммы в виде цилиндров:
            if (Цилиндр == true)
                chart1.Series["Series1"]["DrawingStyle"] = "Cylinder";
            else
                chart1.Series["Series1"]["DrawingStyle"] = "Default";
        }
    }
}
```

В программном коде при обработке события загрузки формы объявляем объект `Таблица` класса `DataTable`. Этот объект представляет одну таблицу данных в опера-

тивной памяти. Для визуализации этой таблицы на экране служит элемент **DataGridView** (Сетка данных). Объект класса `DataTable` используют в качестве исходных данных и для сетки данных **DataGridView**, и для диаграммы **Chart**.

В таблице `DataTable` определяем ее схему, заказывая две колонки: "Месяц" и "Объем продаж". А далее заполняем таблицу по ее строкам (рядам), используя метод `Add`. Заполненную пятью строками таблицу указываем в качестве источника данных для элементов **Chart** и **DataGridView**. Далее оформляем внешний вид диаграммы, что подробно представлено в комментариях к программному коду.

Пример работы программы показан на рис. 7.14.

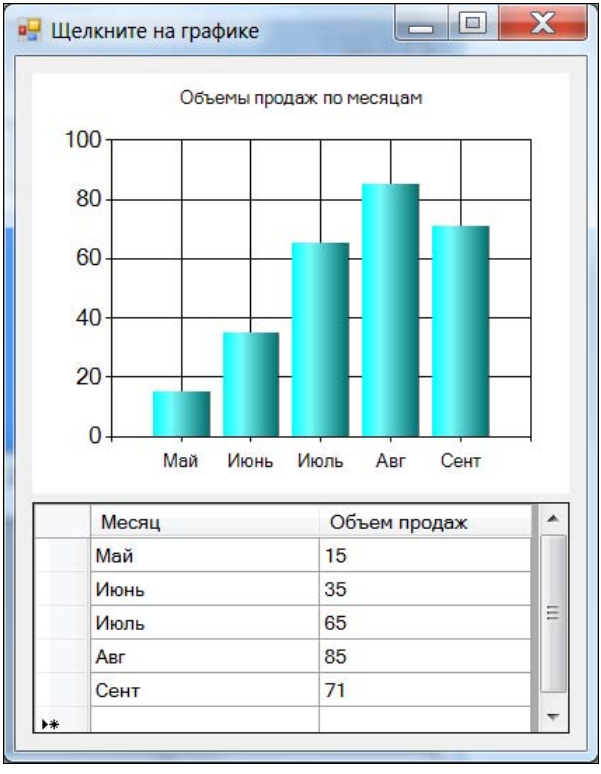


Рис. 7.14. Вывод диаграммы и соответствующей таблицы в экранную форму

Убедиться в работоспособности программы можно, открыв решение ГрафикChart.sln в папке ГрафикChart сопровождающего книгу электронного архива.

ГЛАВА 8

Элемент управления *WebBrowser*

Пример 62. Отображение HTML-таблиц в элементе *WebBrowser*

Мы знаем, что существуют настольные приложения (или Windows-приложения), которые пользователи запускают непосредственно (например, EXE-файлы), и существуют веб-приложения, выполняющиеся внутри браузера. Однако в панели элементов **Toolbox** имеется элемент управления **WebBrowser**, который позволяет просматривать веб-приложения внутри экранной формы Windows-приложения.

В этом примере мы 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 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, вывести эту таблицу в поле элемента управления **WebBrowser**, при этом не записывая сам HTML-файл на винчестер. Для этой цели воспользуемся простейшей таблицей из примера о формировании таблицы.

Приступим к программированию поставленной задачи. Запустим Visual Studio 12 и в окне **New Project** выберем приложение шаблона **Windows Forms Application Visual C#**. Укажем имя решения `ТаблWebHTM`. В панели элементов **Toolbox** находим 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;
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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Никакие края (из четырех) не привязаны к экранной форме:
            webBrowser1.Dock = DockStyle.None;
            // или webBrowser1.Navigate(@"D:\table.htm");
            var СтрокаHTML = "Какой-либо текст до таблицы" +
                        "<table border> " +
                        "<caption>Таблица телефонов</caption> " +
                        "<tr><td>Андрей — раб<td>274-88-17 " +
                        "<tr><td>Света-X<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:" + СтрокаHTML);
        }
    }
}
```

Как видно из текста программы, здесь реализована обработка события загрузки формы. В процедуре этой обработки в строковой переменной `СтрокаHTML` 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.

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 сопровождающего книгу электронного архива.

Напоминание

Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

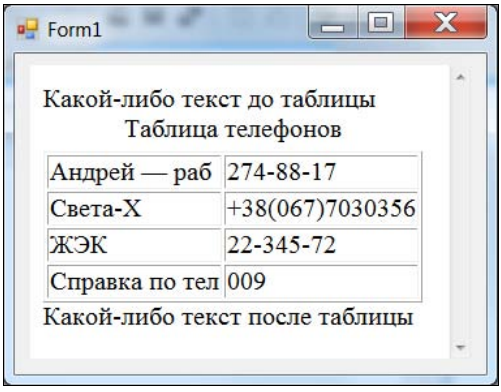


Рис. 8.1. Отображение таблицы в поле элемента управления **WebBrowser**

Пример 63. Отображение Flash-файлов

Ранее мы уже обсуждали элемент управления **PictureBox** (графическое поле), способный отображать растровые файлы различных форматов. Однако элемент **PictureBox** не умеет отображать Flash-файлы формата SWF. А 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 этом примере мы создадим программу, которая будет отображать в поле элемента управления **WebBrowser** файл SWF. Запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Укажем имя решения `FlashWeb`. Перетащим с панели **Toolbox** элемент управления **WebBrowser** в форму. Теперь, двойным щелчком щелкнув в пределах проектируемой формы, мы попадаем на вкладку программного кода, где система автоматически сгенерировала пустой обработчик загрузки формы. Введем туда текст из листинга 8.2.

Листинг 8.2. Отображение Flash-файлов

```
// Программа использует элемент управления WebBrowser для отображения
// Flash-файлов
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace FlashWeb
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void Form1_Load(object sender, EventArgs e)
        {
            // Никакие края (из четырех) не привязаны к экранной форме:
            webBrowser1.Dock = DockStyle.None;
            // Так можно вывести веб-страницу в поле элемента WebBrowser:
            // webBrowser1.Navigate("www.mail.ru");
            // return;
            var ИмяФайла = System.IO.Directory.
                           GetCurrentDirectory() + @"\Shar.swf";
            // Если такого файла нет
            if (System.IO.File.Exists(ИмяФайла) == false)
            {
                MessageBox.Show(
                    "Файл " + ИмяФайла + " не найден", "Ошибка");
                return;
            }
            webBrowser1.Navigate(ИмяФайла);
        }
    }
}
```

Здесь в тексте программы вызываем метод `Navigate` объекта класса `WebBrowser`. На вход метода подаем полный путь к Flash-файлу Shar.swf. Полный путь к текущей папке получаем с помощью функции `GetCurrentDirectory()`. В комментарии показано, как можно вывести веб-страницу из Интернета в поле элемента управления **WebBrowser**.

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

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

К сожалению, на книжной иллюстрации не видно, что шар вращается. Убедиться в работоспособности программы можно, открыв решение FlashWeb.sln в папке FlashWeb сопровождающего книгу электронного архива.

Пример 64. Отображение веб-страницы и ее HTML-кода

Напишем программу, способную в одном поле отображать веб-страницу, а в другом поле, ниже, показывать HTML-код загружаемой в браузер страницы. Ядром этой про-

граммы будет все тот же элемент управления **WebBrowser**. Таким образом, экранная форма будет содержать элемент управления **WebBrowser**, два текстовых поля и кнопку.

Запустим Visual Studio 12 и в окне **New Project** выберем приложение шаблона **Windows Forms Application Visual C#**. Укажем имя решения `Split`. Используя панель элементов, в проектируемую экранную форму добавим элемент управления **WebBrowser**, два текстовых поля и кнопку. Для элемента **WebBrowser** укажем "Undock in Parent Container" (не заполнять родительский контейнер) — в этом случае мы сможем регулировать границы веб-браузера в проектируемой форме. Первое текстовое поле предназначено для ввода URL-адреса желаемой веб-страницы (например, **www.latino.ho.ua**), а второе поле 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 веб-страницы и ее HTML-кода

```
// Эта программа использует элемент управления WebBrowser
// для отображения веб-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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Веб-страница и ее HTML-код";
            textBox1.Text = String.Empty;
            textBox2.Text = String.Empty;
            textBox2.Multiline = true;
            textBox2.ScrollBars = ScrollBars.Vertical;
            button1.Text = "ПУСК";
            webBrowser1.Dock = DockStyle.None;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Обработка события "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)
        {
            // Обработка события "Веб-документ полностью загружен"
            // Получаем HTML-код из элемента WebBrowser:
            textBox2.Text = webBrowser1.Document.Body.InnerHtml;
        }
    }
}
```

Как видно из программного кода, при обработке события загрузки формы 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" метод `Navigate` объекта класса `WebBrowser` вызывает в поле браузера веб-страницу с адресом, указанным пользователем данной программы в первом текстовом поле. В комментарии приведены варианты дальнейшего совершенствования данной программы. Так метод `GoBack` возвращает пользователя на предыдущую веб-страницу, метод `GoForward` — на последующую, а метод `GoHome` загружает в поле браузера домашнюю веб-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.

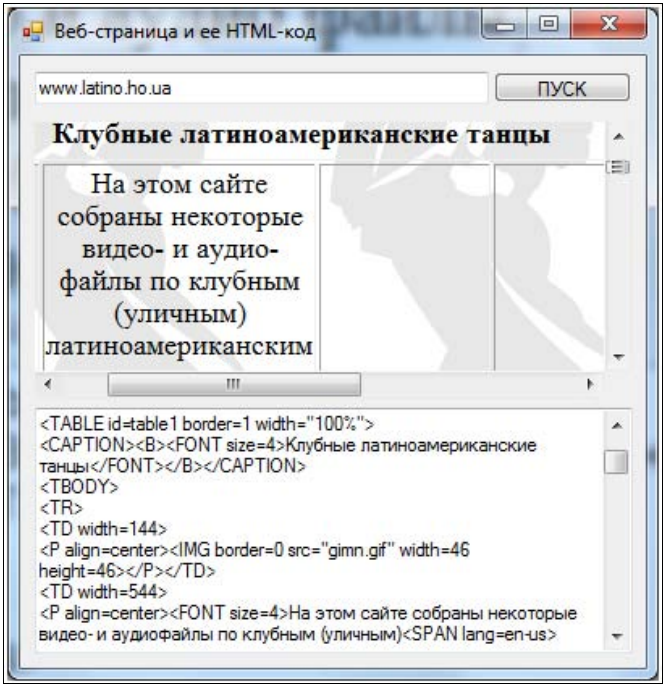


Рис. 8.3. Загрузка веб-страницы и ее HTML-кода

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` второго текстового поля HTML-код, соответствующий загруженной в браузер веб-страницы. На рис. 8.3 показан пример работы данной программы.

Убедиться в работоспособности программы можно, открыв решение Split.sln в папке Split сопровождающего книгу электронного архива.

Пример 65. Программное заполнение веб-формы

Задача состоит в том, чтобы, воспользовавшись элементом управления **WebBrowser**, отобразить в этом элементе какую-либо веб-страницу, содержащую веб-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 по завершении заполнения программно щелкнуть на кнопке **Submit** (она может называться поразному) для отправки заполненной веб-формы для обработки на сервер. В итоге в элементе управления **WebBrowser** будет отображен уже результат, полученный с сервера после обработки.

> ***Примечание***
>
> Подобным средством можно воспользоваться и для подбора паролей, однако мы не хотели бы популяризировать эту возможность из соображений морали.

В этом примере мы будем заполнять строку поиска в наиболее используемых поисковых системах, таких как Yandex, Rambler, Google и др.

Для решения задачи запустим Visual Studio 12 и в окне **New Project** выберем приложение шаблона **Windows Forms Application Visual C#**. Укажем имя решения: `ЗаполнениеВебФормы`. Из панели элементов **Toolbox** в экранную форму перенесем элемент управления **WebBrowser**. Чтобы получить удобный доступ к конкретным дескрипторам (тегам) HTML, составляющим веб-страницу, к текущему проекту добавим ссылку на так называемую "неуправляемую" библиотеку Microsoft.mshtml.dll.

> ***Пояснение***
>
> Не стоит пугаться слова "неуправляемая" — так компания Microsoft называет библиотеки, содержащие "неуправляемый" код, который выполняется не общеязыковой средой, а непосредственно операционной системой. Как можно видеть, имя файла библиотеки содержит не совсем привычные две точки. Этот файл, скорее всего, расположен на вашем компьютере в папке: C:\Program Files\VS2012\Visual Studio Tools for Office\PIA\Common.

Для добавления ссылки на эту библиотеку в пункте меню **Project | Add Reference**, в левой колонке, раскроем узел **Assemblies** и выберем пункт **Extensions**, а затем установим флажок ("птичку") напротив ссылки **Microsoft.mshtml** и щелкнем на кнопке **OK**. Теперь в окне **Sulution Explorer** в папке **References** (Ссылки) увидим добавленную в наш проект выбранную ссылку (рис. 8.4).

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

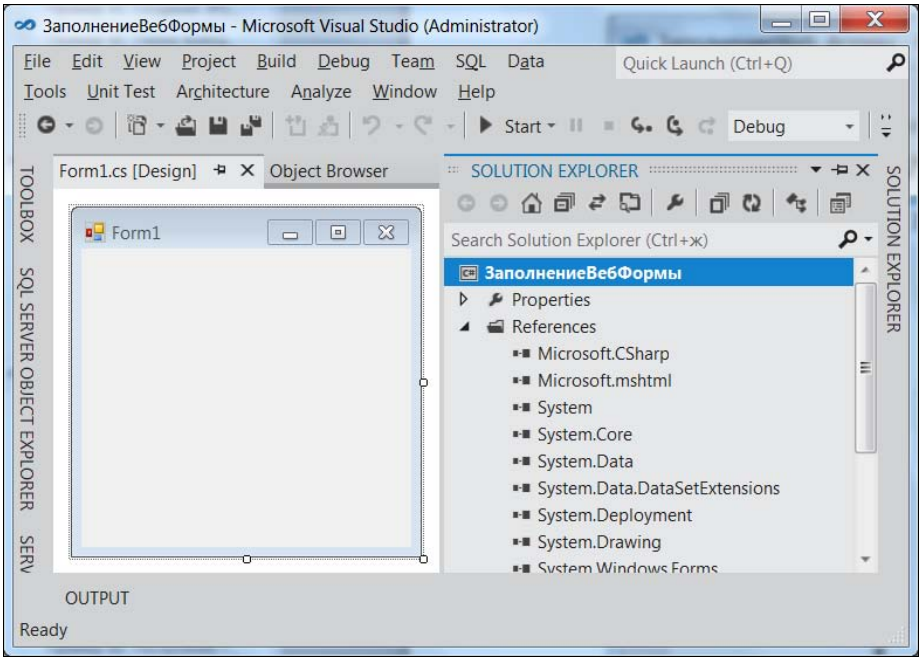


Рис. 8.4. Добавленная в текущий проект ссылка на Microsoft.mshtml

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 веб-страницы по их именам. Заполняем поле
// поиска ключевыми словами для нахождения соответствующих веб-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
namespace ЗаполнениеВебФормы
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
private void Form1_Load(object sender, EventArgs e)
{
    this.Text = "Программное заполнение формы";
    // *** Для сайта "http://google.com":
    // var АдресСайта = "http://google.com";
    // var ИмяФормы = "f";
    // var ИмяПоляФормы = "q";
    // Для сайта "http://meta.ua":
    // var АдресСайта = "http://meta.ua";
    // var ИмяФормы = "sForm";
    // var ИмяПоляФормы = "q";
    // *** Для сайта "http://yandex.ru":
    // var АдресСайта = "http://yandex.ru";
    // var ИмяФормы = "form";
    // var ИмяПоляФормы = "text";
    // *** Для сайта "http://rambler.ru":
    // var АдресСайта = "http://rambler.ru";
    // var ИмяФормы = "rSearch";
    // var ИмяПоляФормы = "query";
    // *** Для сайта "http://aport.ru":
    // var АдресСайта = "http://aport.ru";
    // var ИмяФормы = "aport_search";
    // var ИмяПоляФормы = "r";
    // *** Для сайта "http://bing.com":
    // var АдресСайта = "http://bing.com";
    // var ИмяФормы = "sb_form";
    // — в HTML-коде нет name формы, но есть id = "sb_form"
    // var ИмяПоляФормы = "q";
    // *** Для сайта "http://yahoo.com";
    var АдресСайта = "http://yahoo.com";
    var ИмяФормы = "sf1"; // или "p_13838465-searchform"
    var ИмяПоляФормы = "p"; // или "p_13838465-p"
    // Загружаем веб-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(
            "Возможно, вы не подключены к Интернету", "Ошибка");
        return;
    }
    // Свойство DomDocument приводим к указателю IHTMLDocument2:
    var Док = (mshtml.IHTMLDocument2)
                             webBrowser1.Document.DomDocument;
```

```
            // В этом случае мы получаем доступ к формам веб-страницы
            // по их именам:
            var Форма = (mshtml.HTMLFormElement)
                                        Док.forms.item(ИмяФормы, null);
            if (Форма == null)
            // В VB: If Форма Is Nothing Then...
            {
                MessageBox.Show(String.Format("Форма с " +
                    "именем \"{0}\" не найдена", ИмяФормы), "Ошибка");
                return;
            }
            // В форме находим нужное поле по его (полю) имени:
            var ТекстовоеПоле = (mshtml.IHTMLInputElement)
                                         Форма.namedItem(ИмяПоляФормы);
            if (ТекстовоеПоле == null)
            // В VB: If ТекстовоеПоле Is Nothing Then...
            {
                MessageBox.Show(String.Format("Поле формы с " +
                    "именем \"{0}\" не найдено ", ИмяПоляФормы), "Ошибка");
                return;
            }
            // Заполняем текстовое поле:
            var ТекстЗапроса = "Зиборов Visual";
            ТекстовоеПоле.value = ТекстЗапроса;
            // "Программно" нажимаем кнопку "Submit":
            Форма.submit();
        }
    }
}
```

Как видно из текста программы, весь наш пользовательский программный код написан для обработки события загрузки формы. Здесь мы подготовили для поисковой системы Yahoo ее веб-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 открывающим тегом `<form>` и закрывающим 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 веб-страницах поисковых систем, которые представлены в комментариях.

После первичной загрузки веб-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 веб-страницы по их именам. Далее заполняем поле поиска ключевыми словами для нахождения ссылок на соответствующие веб-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. 8.5).

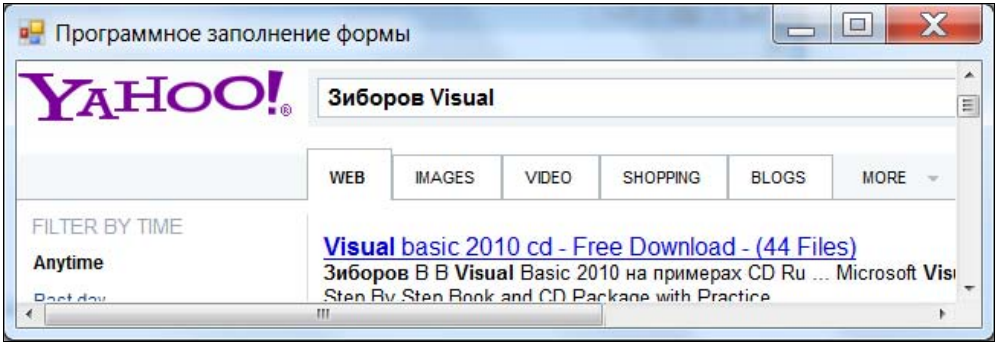


Рис. 8.5. Результат работы поисковой системы

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ЗаполнениеВебФормы сопровождающего книгу электронного архива.

Пример 66. Синтаксический разбор веб-страницы без использования элемента *WebBrowser*

Чтобы иметь доступ к данным какой-либо веб-страницы (к ее гипертекстовой разметке), вовсе не обязательно применять элемент управления **WebBrowser**. В этом примере покажем, как можно воспользоваться какими-либо актуальными сведениями, представленными на веб-странице, скопировав их на экранную форму нашей программы.

Ставя задачу более конкретно, представим, что нам необходимо по роду наших занятий знать официальный курс доллара США на текущий момент. Где нам взять этот официальный курс? Конечно же, на сайте Центрального банка Российской Федерации **www.cbr.ru**. Причем курс доллара нам необходимо видеть постоянно, и чтобы мы не обращались к этому сайту каждый раз, когда он нам понадобится, информация о текущем курсе доллара должна быть выведена на наш компьютерный рабочий стол. Отсюда появляется задача *воспользоваться классом* **WebClient**, с его помощью прочитать гипертекстовую разметку в строковую переменную, найти в ней нужные сведения и скопировать их в текстовую метку нашей экранной формы. Заметьте, что здесь участие элемента управления **WebBrowser** вовсе не обязательно.

Для решения задачи запустим Visual Studio 12, щелкнем на пункте меню **New Project** и в окне **New Project** в области шаблоны (в средней колонке) выберем шаблон

(Templates) **Windows Forms Application Visual C#**. В качестве имени проекта введем имя `РазборВебСтраницы` и щелкнем на кнопке **OK**. А с панели элементов **Toolbox** нам понадобится текстовая метка **Label** и графическое поле **PictureBox** (для отображения логотипа Банка России с его сайта).

Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 8.5.

Листинг 8.5. Синтаксический разбор HTML-разметки веб-страницы Центрального банка РФ

```
// Программа, используя класс WebClient, читает веб-страницу Центрального
// банка РФ www.cbr.ru, ищет в ее гипертекстовой разметке курс доллара
// США и копирует его в текстовую метку Label. Кроме того, элемент
// управления графическое поле PictureBox отображает логотип банка,
// используя URL-адрес этого изображения
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace РазборВебСтраницы
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Сведения от банка России";
            // Создаем объект для чтения веб-страницы:
            var КЛИЕНТ = new System.Net.WebClient();
            // Если веб-страница записана в кодировке Win1251, то чтобы
            // русские буквы читались корректно, следует объявлявить
            // объект Кодировка:
            // var Кодировка = System.Text.Encoding.GetEncoding(1251);
            System.IO.Stream ПОТОК;
            String СТРОКА;
            try
            {
                // Попытка открытия веб-ресурса:
                ПОТОК = КЛИЕНТ.OpenRead("http://www.cbr.ru/");
            }
            catch (Exception Ситуация)
            {
                СТРОКА = String.Format(
                    "www.cbr.ru" + "\n{0}", Ситуация.Message);
```

```
                label1.Text = СТРОКА;
                return;
            }
            // Чтение HTML-разметки веб-страницы в кодировке
            // Unicode (по умолчанию):
            var Читатель = new System.IO.StreamReader(ПОТОК); //, Кодировка);
            // Копируем HTML-разметку в строковую переменную:
            СТРОКА = Читатель.ReadToEnd();
            // Ищем в HTML-разметке страницы курс доллара США:
            var i = СТРОКА.IndexOf("Доллар США");
            СТРОКА = СТРОКА.Substring(i, 300);
            i = СТРОКА.IndexOf("nowrap");
            СТРОКА = СТРОКА.Substring(i + 7);
            i = СТРОКА.IndexOf(" ");
            СТРОКА = СТРОКА.Remove(i);
            // Вставляем текущую дату:
            СТРОКА = String.Format(
                            "Курс доллара США на {0:D}:\n" +
                            "{1} руб за $1 USD", DateTime.Now, СТРОКА);
            ПОТОК.Close();
            // Копируем в текстовую метку найденный курс доллара:
            label1.Text = СТРОКА;
            // В графическом поле отображаем логотип Центрального банка:
            pictureBox1.ImageLocation =
                        "http://www.cbr.ru/images/main_logo.gif";
        }
    }
}
```

Как видно из текста программы, при обработке события загрузки формы создаем объект класса **WebClient**, с помощью которого читаем веб-страницу. Данная веб-страница записана в кодировке Unicode.

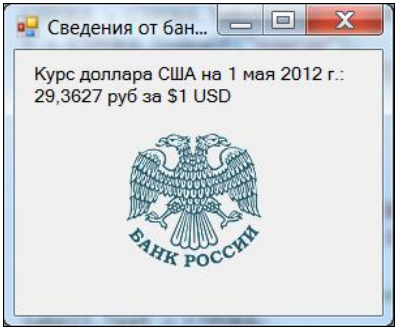


Рис. 8.6. Поиск нужных сведений на веб-ресурсе и их отображение в форме

Пояснение

Чтобы определить, в какой кодировке создана та или иная веб-страница, следует в любом браузере отобразить эту веб-страницу, затем в контекстном меню (нажав правую кнопку мыши) выбрать пункт **Кодировка**. Если бы страница была бы написана в другой кодировке,

например, очень распространенной Windows1251, то для корректного отображения русских букв нам пришлось бы создать объект класса **Encoding**, как и показано в комментарии.

Прочитав в строковую переменную HTML-разметку нужного нам веб-ресурса, найдем с помощью строковых операций курс доллара США на текущий момент. Результат работы программы представлен на рис. 8.6.

Убедиться в работоспособности программы можно, открыв решение РазборВебСтраницы.sln в папке РазборВебСтраницы сопровождающего книгу электронного архива.

ГЛАВА 9

Использование функций MS Word, MS Excel, AutoCAD и MATLAB, а также создание PDF-файла

Пример 67. Проверка правописания в текстовом поле с помощью обращения к MS Word

Пакет приложений Microsoft Office является сервером OLE-объектов, и его функции могут использоваться другими приложениями. Продемонстрируем это на примере, для чего создадим программу, которая предлагает 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.

Для решения задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. В качестве имени проекта введем имя `Орфография1`. 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.

Далее к текущему проекту добавим объектную библиотеку MS Word (библиотеку компонентов). Для этого в пункте меню **Project** укажем команду **Add Reference**. Затем

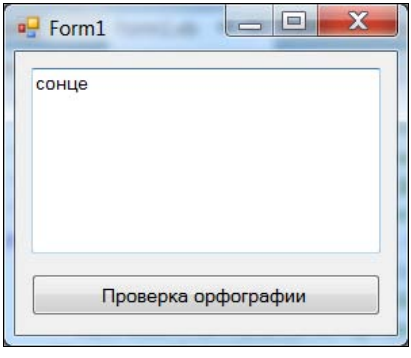


Рис. 9.1. Фрагмент работы программы проверки орфографии

в узле **Assemblies** выберем пункт **Extensions**, в появившемся списке ссылок установим флажок напротив ссылки **Microsoft.Office.Interop.Word** и щелкнем на кнопке **OK**. Таким образом мы подключили библиотеку объектов MS Word. Далее введем программный код, представленный в листинге 9.1.

Листинг 9.1. Проверка орфографии (вариант 1)

```
// 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
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace Орфография1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // В данный проект следует добавить ссылку
            // Microsoft.Office.Interop.Word
            textBox1.Clear(); button1.Text = "Проверка орфографии";
            textBox1.TabIndex = 0; button1.TabIndex = 1;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Создаем новый экземпляр класса Word.Application:
            var Ворд1 = new Microsoft.Office.
                                    Interop.Word.Application();
            // Ворд1.Visible = false;
            // Открываем новый документ MS Word:
            var Документ = Ворд1.Documents.Add();
            // Вводим в документ MS Word текст из текстового поля:
            Документ.Words.First.InsertBefore(
                                        textBox1.Text);
            // Непосредственная проверка орфографии:
            Документ.CheckSpelling();
            // Получаем исправленный текст:
            var ИсправленныйТекст = Документ.Content.Text;
            // Возвращаем в текстовое поле исправленный текст:
            textBox1.Text = ИсправленныйТекст;
            Ворд1.Documents.Close(Microsoft.Office.Interop.Word.
                                WdSaveOptions.wdDoNotSaveChanges);
```

```
            // Закрыть документ Word без сохранения:
            Ворд1.Quit();
            Ворд1 = null;
        }
    }
}
```

Как видно из текста программы, при обработке события загрузки формы очищается текстовое поле и инициализируется название кнопки **Проверка орфографии**. При обработке события "щелчок по кнопке" создается новый объект класса `Word.Application` и командой `Documents.Add` открывается новый документ. Далее весь введенный пользователем текст копируется в этот документ (команда `InsertBefore`). Затем происходит непосредственная проверка орфографии методом `CheckSpelling`. Если текст проверен, то правильный вариант написания слова находится теперь в одном из свойств объекта класса `Word.Document`, а именно в свойстве `Content.Text`. Возвращаем исправленный текст назад в текстовое поле. Промежуточный документ MS Word закрывается без сохранения изменений `wdDoNotSaveChanges`, поэтому при работе программы мы этот документ даже не замечаем.

Теперь проверим, как работает написанная нами программа. Запустим ее, нажав на функциональную клавишу <F5>, и в текстовое поле введем какое-либо слово с заведомой ошибкой. После щелчка на кнопке **Проверка орфографии** получим диалоговое окно, подобное представленному на рис. 9.2.

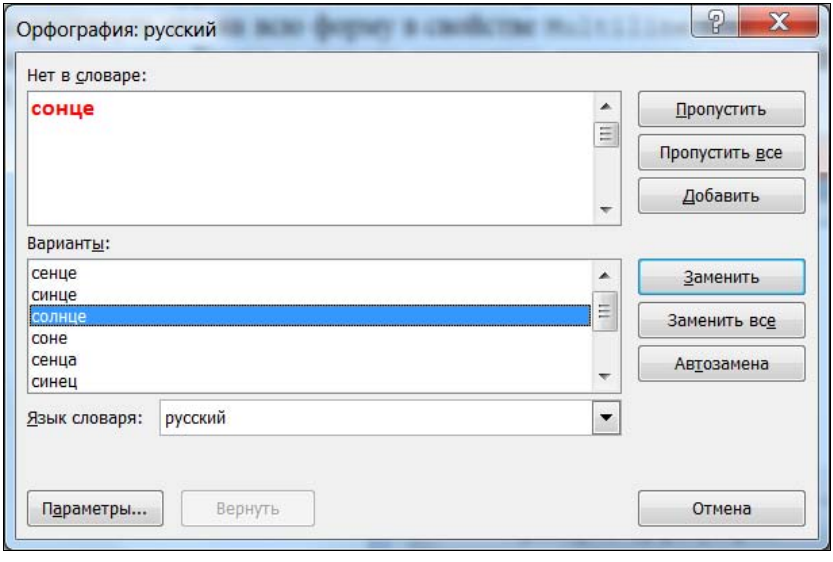


Рис. 9.2. Проверка правописания в текстовом поле

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.

Ниже, в листинге 9.2, приведем другой, более краткий вариант решения этой задачи с использованием объекта класса **Selection**. Имя нового проекта — `Орфография2`.

Листинг 9.2. Проверка орфографии (вариант 2)

```
// Программа позволяет пользователю проверить
// орфографию введенного текста
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace Орфография2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            textBox1.Clear(); button1.Text = "Проверка орфографии";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            var Ворд1 = new Microsoft.Office.Interop.Word.Application();
            Ворд1.Visible = false;
            // Открываем новый документ MS Word:
            Ворд1.Documents.Add();
            // Копируем содержимое текстового окна в документ
            Ворд1.Selection.Text = textBox1.Text;
            // Непосредственная проверка орфографии:
            Ворд1.ActiveDocument.CheckSpelling();
            // Копируем результат назад в текстовое поле
            textBox1.Text = Ворд1.Selection.Text;
            Ворд1.Documents.Close(false);
            // Закрыть документ Word без сохранения:
            Ворд1.Quit();
            Ворд1 = null;
        }
    }
}
```

Здесь аналогично создаем новый объект класса `Word.Application` и, используя метод `Add`, открываем документ MS Word. Далее воспользуемся объектом `Ворд1.Selection`, он оперирует с текущей позицией "невидимого" курсора, например, может подать команду ввода текста в документ MS Word (метод `TypeText`). Мы используем свойство `Text` этого объекта для копирования содержимого текстового окна в документ MS Word. Соответственно, доступ к этому свойству должен быть реализован с помощью предложения: `Ворд1.Selection.Text`. Далее происходит непосредственная проверка орфографии аналогично предыдущему варианту решения этой задачи.

Убедиться в работоспособности приведенных программ можно, открыв решения в папках, соответственно, Орфография1 и Орфография2 сопровождающего книгу электронного архива.

Напоминание

Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 68. Вывод таблицы средствами MS Word

В этой 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# 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 имеются программные объекты, названия и назначения которых во многом схожи с объектами, используемыми в Visual C# 12. 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# 12, которая, используя функции MS Word, строит таблицу. Итак, запускаем Visual Studio 12 и в окне **New Project** выбираем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ТаблицаWord`. Далее к текущему проекту добавляем объектную библиотеку MS Word. Для этого в меню **Project** укажем команду **Add Reference**. Затем в узле **Assemblies** выберем пункт **Extensions**, в появившемся списке ссылок установим флажок ("птичку") напротив ссылки **Microsoft.Office.Interop.Word** и щелкнем на кнопке **OK**. 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 9.3.

Листинг 9.3. 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;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ТаблицаWord
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // В меню Project укажем команду Add Reference и в узле
            // Assemblies выберем пункт Extensions, в появившемся списке
            // отметим ссылку Microsoft.Office.Interop.Word.
            button1.Text = "Пуск"; base.Text = "Построение таблицы";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Инициализируем два строковых массива:
            String[] Имена = {
                         "Андрей — раб", "Света-Х", "ЖЭК",
                         "Справка по тел", "Александр Степанович",
                         "Мама — дом", "Карапузова Таня",
                         "Погода сегодня", "Театр Браво"};
            String[] Тлф = {
                       "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"};
            // Создаем новый экземпляр класса Word.Application:
            var Ворд1 = new Microsoft.Office.Interop.
                                   Word.Application();
            Ворд1.Visible = true;
            // Открываем новый документ MS Word:
            var Документ = Ворд1.Documents.Add();
            // Вводим текст в документ MS WORD с текущей позиции:
            Ворд1.Selection.TypeText("ТАБЛИЦА ТЕЛЕФОНОВ");
            // Параметр, указывающий, показывать ли границы ячеек:
            var t1 = Microsoft.Office.Interop.
                         Word.WdDefaultTableBehavior.wdWord9TableBehavior;
            // Параметр, указывающий, будет ли приложение Word автоматически
            // изменять размер ячеек в таблице для подгонки содержимого:
            var t2 = Microsoft.Office.Interop.
                         Word.WdAutoFitBehavior.wdAutoFitContent;
            // Создаем таблицу из 9 строк и 2 столбцов:
            Ворд1.ActiveDocument.Tables.Add(Ворд1.Selection.Range,
                                                     9, 2, t1, t2);
```

```
            // Заполнять ячейки таблицы можно так:
            for (int i = 1; i <= 9; i++)
            {
                Ворд1.ActiveDocument.Tables[1].Cell(i, 1).
                                    Range.InsertAfter(Имена[i — 1]);
                Ворд1.ActiveDocument.Tables[1].Cell(i, 2).
                                      Range.InsertAfter(Тлф[i — 1]);
                // Программируя на VB, мы написали бы:
                // Ворд1.ActiveDocument.Tables(1).Cell(i, 2).
                //                      Range.InsertAfter(Тлф(i — 1))
            }
            // Назначаем единицы измерения в документе приложения MS Word:
            var t3 = Microsoft.Office.Interop.Word.WdUnits.wdLine;
            // Параметр, указывающий на девятую строку в документе MS Word:
            var Строка9 = 9;
            // Перевести текущую позицию (Selection) за пределы таблицы,
            // (в девятую строку), чтобы здесь вывести какой-либо текст:
            Ворд1.Selection.MoveDown(t3, Строка9, null);
            // И здесь печатаем следующий текст:
            Ворд1.Selection.TypeText("Какой-либо текст после таблицы");
            // Сохранять документ нет смысла, но это решит пользователь:
            // var ИмяФайла = @"D:\a.doc";
            // Ворд1.ActiveDocument.SaveAs(ИмяФайла);
        }
    }
}
```

Заметим, что содержимое текстовой таблицы такое же, как и в *примере 53 (см. главу 7)*. То есть наш сюжет меняется, а действующие персонажи остаются прежними.

Данные находятся в двух массивах: `Имена[]` и `Тлф[]`. Далее создаем экземпляр объекта `Word.Application` и открываем новый документ `Document.Add`. Затем демонстрируем, как можно добавлять какие-либо тексты в новый документ из написанной нами программы. Например, мы вводим в активный документ текст "ТАБЛИЦА ТЕЛЕФОНОВ", используя объект `Selection`, определяющий текущую позицию "невидимого" курсора и содержащего методы для ввода в документ MS Word.

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.

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 сопровождающего книгу электронного архива.

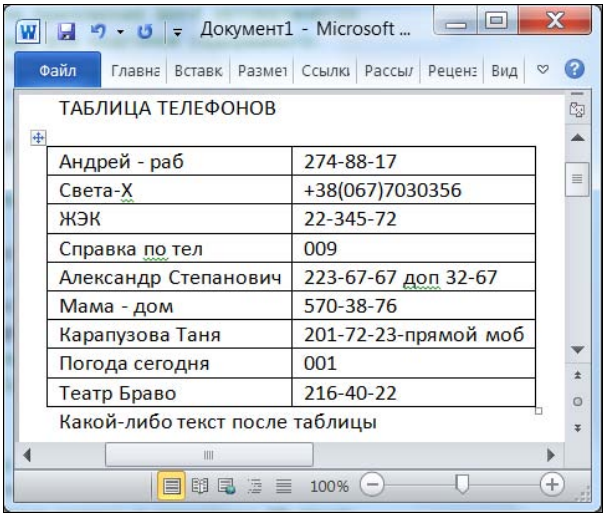


Рис. 9.3. Программно сформированная таблица в редакторе MS Word

Пример 69. Обращение к функциям MS Excel из программы на Visual C# 12

Очень заманчиво обратиться из какой-нибудь вашей программы, написанной на Visual C# 12, 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, причем доступ к этим средствам, как мы убедимся, очень простой и удобный.

В этой программе мы продемонстрируем буквально в трех строчках программного кода обращение к одной простой функции MS Excel, а именно получению значения числа $\pi = 3{,}14$. Число π представлено в классе `Math` языка Visual C# 12. Но цель данной программы — показать легкость доступа к функциям MS Excel. Если у нас получится обращение к этой элементарной функции, то мы открываем для себя целый пласт возможностей управления огромной библиотекой объектов MS Excel.

Как обычно, запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ExcelПи`. 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 в узле **Assemblies** выберем пункт **Extensions**, в появившемся списке ссылок установим флажок ("птичку") напротив ссылки **Microsoft.Office.Interop.Excel** и щелкнем на кнопке **OK**.

Примечание

Обратите внимание, процедура добавления новой библиотеки объектов такая же, как и в примерах с использованием 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**. Если в данный момент вы не увидели это окно в своем текущем проекте, воспользуйтесь комбинацией клавиш <Ctrl>+<Alt>+<L>.

Итак, мы подключили библиотеку объектов MS Excel. На содержимое этой библиотеки мы можем посмотреть в окне **Object Browser** (Обозреватель объектов). Чтобы его открыть, удобно воспользоваться комбинацией клавиш <Ctrl>+<Alt>+<J>. В этом окне найдем объект `WorksheetFunction`, при этом в области **Members of 'WorksheetFunction'** увидим доступные нам функции MS Excel для объекта `WorksheetFunction`. Теперь в программном коде обратимся к одной из этих функций, а именно к функции `Pi()`.

Для этого перейдем на вкладку **Form1.cs[Design]** и щелкнем двойным щелчком в пределах проектируемой формы. При этом произойдет автоматический переход на вкладку программного кода **Form1.cs** и будет создан пустой обработчик события загрузки формы. Здесь напишем программный код, приведенный в листинге 9.4.

Листинг 9.4. Обращение к одной из функций MS Excel

```
// Программа обращается к одной простой функции объектной библиотеки
// MS Excel для получения значения числа Пи = 3,14...
// ~ ~ ~ ~ ~ ~ ~ ~
// Чтобы добавить ссылку на объектную библиотеку Excel, в пункте меню
// Project укажем команду Add Reference и в узле Assemblies выберем
// пункт Extensions, в появившемся списке отметим ссылку
// Microsoft.Office.Interop.Excel.
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ExcelПи
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Создание экземпляра класса Excel.Application:
            var XL = new Microsoft.Office.Interop.
                                        Excel.Application();
            var PI = XL.WorksheetFunction.Pi();
            // Выводим значение ПИ в строку заголовка формы
            this.Text = "PI = " + PI.ToString();
            XL.Quit();
        }
    }
}
```

Как видно, весь наш пользовательский программный код написан для обработки события загрузки формы. Мы создаем объект `Excel.Application`, с помощью которого имеем доступ для одной из функций MS Excel, возвращающей число π = 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 сопровождающего книгу электронного архива.

Управлять функциями MS Excel в программе на Visual C# 12 очень перспективно, т. к. появляются большие возожности. Например, оцените возможность решать сложнейшие в математическом смысле оптимизационные задачи (т. е. задачи нахождения максимума/минимума с набором ограничений), доступные в MS Excel через меню **Сервис | Поиск решения**.

Пример 70. Использование финансовой функции MS Excel

Рассмотрим еще один пример обращения к функциям MS Excel из программы на Visual C# 12. Допустим, вы взяли в кредит 100 тыс. долларов на покупку квартиры 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 программу на Visual C# 12 с обращением к функции `Pmt()` и проверить результат в русскоязычной версии MS Excel.

> ***Примечание***
>
> 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 12, важно найти соответствие русскоязычных функций MS Excel и их аналогов в объекте `WorksheetFunction` для отладки на тестовых примерах.

Запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ExcelПлт`. В проектируемую экранную форму из панели **Toolbox** перенесем три метки, три текстовых поля (для ввода трех указанных ранее аргументов функции `Pmt()`) и кнопку. В текущий проект подключаем библиотеку объектов MS Excel. Для этого в меню **Project** укажем команду **Add Reference**, затем в узле **Assemblies** выберем пункт **Extensions**, в появившемся списке ссылок установим флажок напротив ссылки **Microsoft.Office.Interop.Excel** и щелкнем

на кнопке **OK**. Теперь можно перейти к программному коду, приведенному в листинге 9.5.

Листинг 9.5. 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
// ~ ~ ~ ~ ~ ~ ~ ~
// Для подключения библиотеки объектов MS Excel в пункте меню Project
// выберем команду Add Reference. Затем в узле Assemblies выберем
// пункт Extensions, в появившемся списке отметим ссылку
// Microsoft.Office.Interop.Excel.
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ExcelПлт
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Расчет ежемесячных платежей";
            label1.Text = "Год. ставка в %";
            label2.Text = "Срок в месяцах";
            label3.Text = "Размер кредита";
            textBox1.Clear(); textBox2.Clear(); textBox3.Clear();
            button1.Text = "Расчет";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                var XL = new Microsoft.Office.Interop.
                                        Excel.Application();
                // Переменная с "пустым" значением:
                var t = Type.Missing;
                // Получаем размер месячного платежа:
                var pay = XL.WorksheetFunction.Pmt(
                             (Convert.ToDouble(textBox1.Text)) / 1200,
                              Convert.ToDouble(textBox2.Text),
                              Convert.ToDouble(textBox3.Text), t, t);
                // ИЛИ, если использовать функцию Pmt()
                // из Microsoft.VisualBasic:
                // var FV = 0.0;
```

```
                // var dt =
                //      Microsoft.VisualBasic.DueDate.EndOfPeriod;
                // var pay = Microsoft.VisualBasic.Financial.Pmt(
                //     (Convert.ToDouble(textBox1.Text)) / 1200,
                //      Convert.ToDouble(textBox2.Text),
                //      Convert.ToDouble(textBox3.Text), FV, dt);
                var Строка = String.Format(
                    "Каждый месяц следует платить {0:$#.##} долларов",
                                                    Math.Abs(pay));
                MessageBox.Show(Строка);
                XL.Quit();
            }
            catch (Exception Ситуация)
            {
                MessageBox.Show(Ситуация.Message, "Ошибка",
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
        }
    }
}
```

Как видно из текста программы, при обработке события загрузки формы 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 вход функции также приходится подать еще два параметра, которые являются необязательными. Здесь мы использовали поле `Type.Missing` для получения значения параметра метода по умолчанию. 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.

В данной программе мы несколько переменных объявили как `var` — это означает, что тип соответствующей переменной выводится из выражения инициализации (как объявление Dim в Visual Basic или auto в C++). Ранее мы уже отмечали, что это новая возможность в Visual C# 12.

Интерфейс программы показан на рис. 9.5.

Убедиться в работоспособности программы можно, открыв решение ExcelПлт.sln в папке ExcelПлт сопровождающего книгу электронного архива.

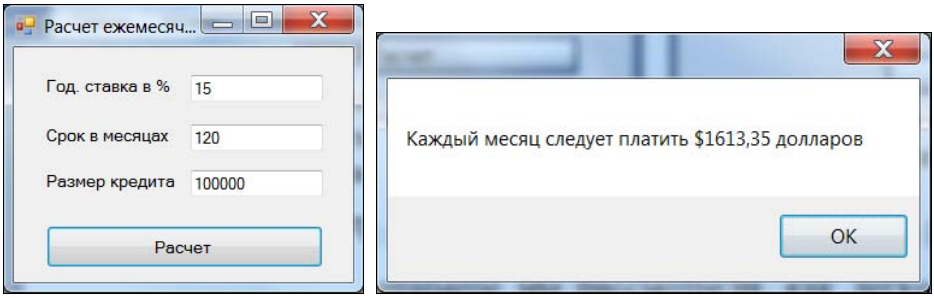


Рис. 9.5. Расчет ежемесячных платежей

Уточнение

Здесь 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 12 в пространстве имен `Microsoft.VisualBasic.Financial` точно с такими же параметрами. (Более того, эта функция была еще в Visual Basic 6.) В комментариях мы показали, как обращаться к функции `Pmt()`, если в проект *добавить ссылку на библиотеку* Microsoft.VisualBasic.dll. Для этого в пункте меню **Project** надо выбрать команду **Add Reference**, а на вкладке **.NET** щелкнуть двойным щелчком 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 программы на Visual C# 12.

Пример 71. Решение системы уравнений с помощью функций MS Excel

Используя функции MS Excel, в своей программе, созданной на Visual C# 12, 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 функ-

цию `MInverse()`. Для умножения обратной матрицы на вектор свободных членов есть соответственно функции `МПРОИЗ()` и `MMult()`. Таких функций нет в Visual Studio 12, и в данном случае мы получаем реальный положительный эффект от подключения к функциям MS Excel.

Для программной реализации решения поставленной задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новый проект назовем `ExcelСЛАУ`. В проектируемую форму из панели **Toolbox** добавим текстовую метку **Label**. На ней будем формировать ответ задачи. Кроме того, добавим библиотеку объектов MS Excel. Для этого в пункте меню **Project** укажем команду **Add Reference**. Затем, в узле **Assemblies** выберем пункт **Extensions**, в появившемся списке ссылок установим флажок напротив ссылки **Microsoft.Office.Interop.Excel** и щелкнем на кнопке **OK**.

Программу построим следующим образом (листинг 9.6): при обработке события загрузки формы прямо в тексте программы зададим (инициализируем) прямую матрицу в виде двумерного массива и строку свободных членов в виде одномерного массива. Затем после решения системы выведем ответ на текстовую метку.

Листинг 9.6. Решение системы линейных уравнений с использованием объектной библиотеки MS Excel

```
// Программа решает систему уравнений с помощью функций объектной
// библиотеки MS Excel
// ~ ~ ~ ~ ~ ~ ~ ~
// Для подключения библиотеки объектов MS Excel в пункте меню Project
// выберем команду Add Reference. Затем в узле Assemblies выберем
// пункт Extensions, в появившемся списке отметим ссылку
// Microsoft.Office.Interop.Excel.
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ExcelСЛАУ
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Escape-последовательность для перехода на новую строку:
            const String НС = "\r\n"; // Новая строка
            // 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:
            // Массив 3 X 3 типа Double:
            Double[,] A = {{1.0, 1,   1},
                           {1,   1,   0},
                           {0,   1,   1}};
            // Свободные члены:
            Double[] L = { 6.0, 3, 5 }; // — одномерный массив типа Double
            // Создание экземпляра класса Excel.Application:
            var XL1 = new Microsoft.Office.Interop.
                                      Excel.Application();
            // Вычисление детерминанта матрицы A
            var det_A = XL1.Application.WorksheetFunction.MDeterm(A);
            // Если det_A близок к 0, то выход из процедуры:
            if (Math.Abs(det_A) < 0.01)
            {
                label1.Text = "Система не имеет решения, поскольку" +
                                 HC + "определитель равен нулю";
                return;
            }
            // Получение обратной матрицы oA:
            var oA = XL1.Application.WorksheetFunction.MInverse(A);
            // Функция Transpose преобразует строку свободных
            // членов в вектор:
            var ВекторL = XL1.Application.WorksheetFunction.Transpose(L);
            // Умножение обратной матрицы на вектор свободных членов:
            var X = XL1.Application.WorksheetFunction.MMult(oA, ВекторL);
            // Чтобы ответ приобрел индексированные свойства,
            // преобразуем его в массив:
            var Xd = new Object[3, 1];
            Xd = X;
            // Получаем двумерный массив, индексы которого
            // начинаются с единицы:
            label1.Text = String.Format(
                         "Неизвестные равны:" + HC +
                         "X1 = {0};  X2 = {1};  X3 = {2}.",
                           Xd[1, 1], Xd[2, 1], Xd[3, 1]);
        }
    }
}
```

Как видно из текста программы, задавая прямую матрицу, значения коэффициентов присваиваем сразу при объявлении двумерного массива. Аналогично поступаем со строкой (одномерным массивом) свободных членов. Согласно требованию объекта `WorksheetFunction` возвращаемые обратная матрица и вектор неизвестных должны быть объявлены как объектные переменные. Вначале вычисляем детерминант (определи-

тель) 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 текстовой метке, что система не имеет решения.

Если определитель матрицы по абсолютному значению больше `0.01`, то с помощью функции MS Excel `MInverse()` находим обратную матрицу. Теперь строку (одномерный массив) свободных членов следует преобразовать в вектор, для этой цели MS Excel имеет функцию `Transpose` (т. е. транспонировать массив `L`). Далее обратную матрицу с помощью функции MS Excel `MMult()` умножаем на вектор свободных членов. В результате функция `MMult()` возвращает двумерный массив, индексы которого начинаются с единицы (но не с нуля). Следующим оператором форматируем ответ в текстовой метке.

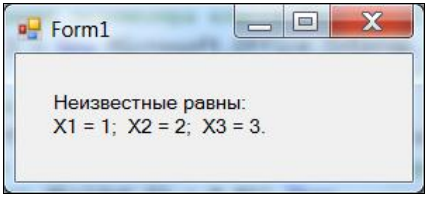


Рис. 9.6. Решение системы линейных алгебраических уравнений

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

Как видим, довольно сложные задачи благодаря обращению к функциям MS Excel можно решать в коротенькой программе. Причем на компьютере, где будет работать данная программа, вовсе не обязательно должен быть инсталлирован MS Excel. Однако соответствующая DLL-библиотека в вашей системе присутствовать должна.

Пример 72. Построение диаграммы средствами MS Excel

Очень часто необходимо изобразить на графике (диаграмме) те или иные экономические показатели или технические измерения (геодезические, метрологические, астрономические), например, для принятия каких-либо решений. И сделать это надо очень оперативно. Традиционно измеренные данные вносятся в ячейки рабочего листа MS Excel, а чтобы получить график, построенный по этим данным, нужно выполнить команду меню **Вставка | Диаграмма**. В этом примере я покажу, как можно очень "малой кровью" получить график (диаграмму) Visual C# 12, используя средства (объекты компонентной библиотеки) MS Excel.

Запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Console Application**, поскольку экранная форма нам не нужна. Новую программу назовем `ExcelГрафик`. Теперь к текущему проекту добавим библиотеку объектов MS Excel. Для этого в пункте меню **Project** выберем команду **Add Reference**, затем в узле **Assemblies** выберем пункт **Extensions**, в появившемся списке ссылок установим флажок напротив ссылки

Microsoft.Office.Interop.Excel и щелкнем на кнопке **OK**. А на вкладке программного кода введем текст, приведенный в листинге 9.7.

Листинг 9.7. Построение диаграммы средствами MS Excel

```
// Программа строит диаграмму (график), используя объекты
// компонентной библиотеки MS Excel
// ~ ~ ~ ~ ~ ~ ~ ~
// Для подключения библиотеки объектов MS Excel в пункте меню Project
// выберем команду Add Reference. Затем в узле Assemblies выберем
// пункт Extensions, в появившемся списке отметим ссылку
// Microsoft.Office.Interop.Excel.
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Чтобы обеспечить более компактные выражения в программном
// коде, вставим следующую директиву:
using Microsoft.Office.Interop.Excel;
namespace ExcelГрафик
{
    class Program
    {
        static void Main(string[] args)
        {
            // Создаем экземпляр класса Excel.Application:
            var XL1 = new Application();
            XL1.Visible = true;
            // Задаем параметр "по умолчанию" для его дальнейшего
            // использования в соответствующих методах:
            var t = Type.Missing;
            // Создаем новую книгу MS Excel:
            var Книга = XL1.Workbooks.Add(t);
            // Объявляем листы в книге:
            var Листы = Книга.Worksheets;
            // Выбираем первый лист:
            Worksheet Лист = Листы.Item[1];
            // Записываем данные по продажам в соответствующих ячейках:
            Лист.Range["A1", t].Value2 = "Месяцы";
            Лист.Range["A2", t].Value2 = "Март";
            Лист.Range["A3", t].Value2 = "Апр";
            Лист.Range["A4", t].Value2 = "Май";
            Лист.Range["A5", t].Value2 = "Июнь";
            Лист.Range["A6", t].Value2 = "Июль";
            Лист.Range["B1", t].Value2 = "Продажи";
            Лист.Range["B2", t].Value2 = 138;
            Лист.Range["B3", t].Value2 = 85;
            Лист.Range["B4", t].Value2 = 107;
            Лист.Range["B5", t].Value2 = 56;
            Лист.Range["B6", t].Value2 = 34;
```

```
            // Заказываем построение диаграммы (графика) с
            // умалчиваемыми параметрами
            Chart График = XL1.Charts.Add(t, t, t, t);
            // Задаем диапазон значений для построения графика:
            График.SetSourceData(Лист.Range["A2", "B6"],
                                 XlRowCol.xlColumns);
            // Задаем тип графика "столбиковая диаграмма" (гистограмма):
            График.ChartType = XlChartType.xlColumnClustered;
            // Отключаем легенду графика:
            График.HasLegend = false;
            // График имеет заголовок:
            График.HasTitle = true;
            График.ChartTitle.Caption = "ПРОДАЖИ ЗА ПЯТЬ МЕСЯЦЕВ";
            // Подпись оси X:
            Axis ГоризонтальнаяОсь = График.Axes(
                        XlAxisType.xlCategory, XlAxisGroup.xlPrimary);
            ГоризонтальнаяОсь.HasTitle = true;
            ГоризонтальнаяОсь.AxisTitle.Text = "Месяцы";
            // Подпись оси Y:
            Axis ВертикальнаяОсь = График.Axes(
                           XlAxisType.xlValue, XlAxisGroup.xlPrimary);
            ВертикальнаяОсь.HasTitle = true;
            ВертикальнаяОсь.AxisTitle.Text = "Уровни продаж";
            // Сохранение графика в растровом файле:
            // XL1.ActiveChart.Export(@"D:\ExcelГрафик.jpg", t, t);
        }
    }
}
```

Вначале программного кода создаем объект класса `Excel.Application` и рабочую книгу. Далее заполняем ячейки первого листа: вначале в ячейки `Ai` пишем подписи абсцисс гистограммы, а в ячейки `Bi` — значения ординат.

Диаграмму строим, используя объект класса `Chart`. Напомним читателю, что мы в предыдущих примерах уже строили диаграмму, используя элемент управления экранной формы `Chart`. Однако, несмотря на похожесть названий, эти классы расположены в разных пространствах имен, соответственно разные у них названия свойств, методов и событий.

Заказывая построение диаграммы, задаем необходимый диапазон значений. Затем задаем тип диаграммы — `xlColumnClustered`, что соответствует гистограмме (столбиковой диаграмме). Далее указываем название гистограммы и подписываем горизонтальную и вертикальную оси. Затем в комментарии даем возможность сохранять полученную диаграмму на диске в виде JPG-файла с помощью функции `Export()`. Графическое отображение этого файла можно посмотреть на рис. 9.7, а соответствующие исходные данные для построения диаграммы на листе Excel на рис. 9.8.

Убедиться в работоспособности программы можно, открыв решение ExcelГрафик.sln в папке ExcelГрафик сопровождающего книгу электронного архива.

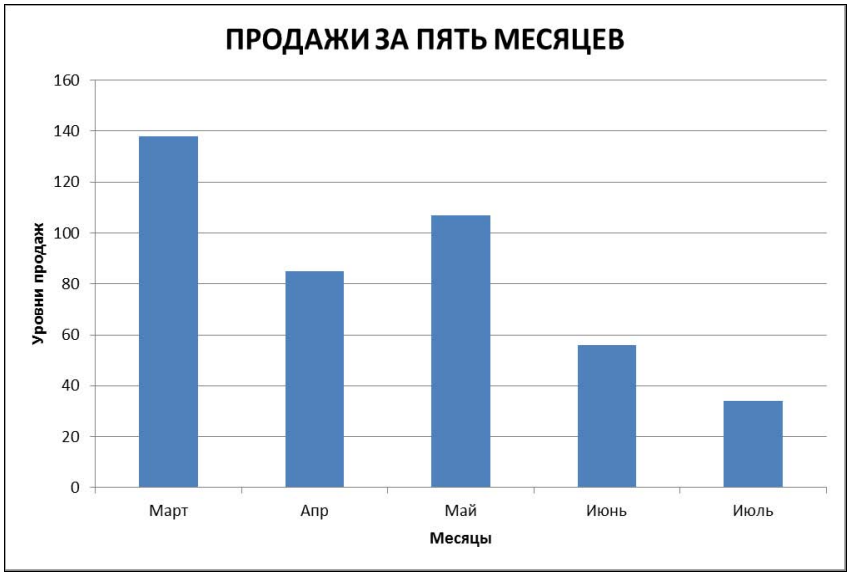


Рис. 9.7. Графическое отображение полученного JPG-файла

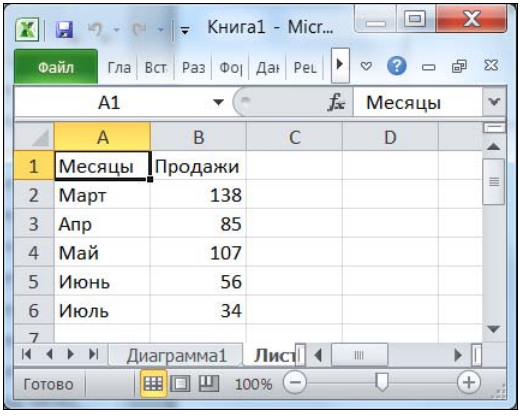


Рис. 9.8. Исходные данные для построения диаграммы на листе MS Excel

Пример 73. Управление функциями AutoCAD из программы на Visual C# 12

Если результатом работы вашей программы должно быть какое-либо векторное изображение (техническая документация, строительный чертеж, географическая карта и проч.), то самый быстрый путь создания такого приложения — это обращение из вашей программы к функциям AutoCAD. AutoCAD (Computer-Aided Design) — представляет собой 2- и 3-мерную систему автоматизированного проектирования и черчения. Эта система, так же как и пакет приложений Microsoft Office, является сервером OLE-объектов, и ее 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 этих элементов. Программа на Visual C# 12 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. Поскольку в данной задаче нас не интересует пользовательский интерфейс, будем программировать консольное приложение. Запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Console Application**. Новый проект назовем `ACADЭлементарныйЧертеж`. Далее следует подключить библиотеку объектов AutoCAD, для этого в пункте меню **Project** выбираем команду **Add Reference**, затем на вкладке **COM** выберем две ссылки: одну — с названием **AutoCAD 2008 Type Library**, а вторую — **AutoCAD/ObjectDBX Common 17.0 Type Library**.

Таким образом, мы подключили две библиотеки объектов AutoCAD версии 2008. Если на вашем компьютере инсталлирована другая версия AutoCAD, то действуйте аналогично. Программа, выводящая в DWG-файл два отрезка, дугу, а также две горизонтально и вертикально ориентированных строки текста, представлена в листинге 9.8.

Листинг 9.8. 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 2008, рисует
// два отрезка, одну дугу и два текстовых объекта, сохраняет чертеж в
// файле D:\Чертеж.dwg и завершает работу AutoCAD
// ~ ~ ~ ~ ~ ~ ~ ~
// Следует подключить библиотеку объектов AutoCAD. Для этого надо выбрать
// Project | Add Reference — вкладка COM — AutoCAD 2008 Type Library,
// а также AutoCAD/ObjectDBX Common 17.0 Type Library
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ACADЭлементарныйЧертеж
{
    class Program
    {
        static void Main(string[] args)
        {
            // Создаем экземпляр класса AcadApplication:
            var ACAD1 = new Autodesk.AutoCAD.Interop.AcadApplication();
            // Задаем параметр "по умолчанию" для его дальнейшего
            // использования в соответствующих методах:
            var t = Type.Missing;
            var Документ = ACAD1.Documents.Add(t);
```

```
            // Видимость:
            ACAD1.Visible = true;
            // Узловые точки чертежа:
            Double[] T1 = { 10.0, 10, 0 };
            Double[] T2 = { 200.0, 200, 0 };
            Double[] T3 = { 200.0, 10, 0 };
            Double[] T4 = { 15.0, 200, 0 };
            Double[] T5 = { 100.0, 100, 0 };
            // Нарисовать отрезок от точки T1 до точки T2:
            Документ.ModelSpace.AddLine(T1, T2);
            // Вертикальный отрезок от точки T2 до точки T3:
            var ВертикальныйОтрезок =
                            Документ.ModelSpace.AddLine(T2, T3);
            // Построить дугу с центром в точке T5, радиусом 80,
            // от линии, параллельной оси X длиной 180 градусов:
            var Дуга = Документ.ModelSpace.AddArc(T5, 80, 0, Math.PI);
            // или просто Документ.ModelSpace.AddArc(T5, 80, 0, Math.PI)
            // Горизонтальный текст (с разворотом 0 градусов)
            Документ.ModelSpace.AddText("Горизонтальный", T4, 22);
            // Вертикальный текст с разворотом на 90 градусов = ПИ/2:
            var ВертикТекст = Документ.ModelSpace.
                                    AddText("Вертикальный", T1, 22);
            // Задаем разворот текста на 90 гдадусов:
            ВертикТекст.Rotation = Math.PI / 2;
            // Сохраняем чертеж на диске:
            Документ.SaveAs(@"D:\Чертеж.dwg", t, t);
            ACAD1.Quit();
        }
    }
}
```

Как видно из текста программы, вначале мы создаем объект класса `AcadApplication` и принадлежащий ему документ класса `AcadDocument`. 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 пять точек, 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 о том, что AutoCAD способен строить трехмерные чертежи.

Затем рисуем два отрезка `AddLine` через точки `T1`, `T2` и `T2`, `T3`. 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 D:.

В результате работы этой программы получаем чертеж в системе AutoCAD, подобный представленному на рис. 9.9.

Убедиться в работоспособности программы можно, открыв SLN-файл в папке АСАДЭлементарныйЧертеж сопровождающего книгу электронного архива.

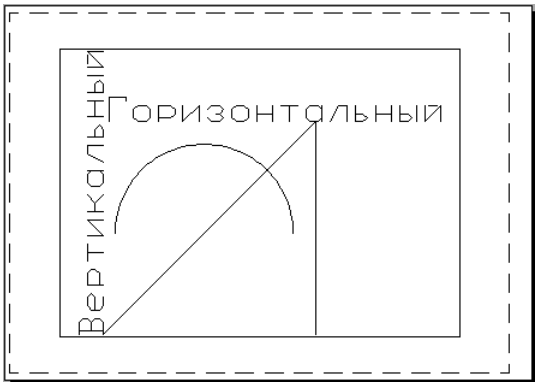


Рис. 9.9. Отображение полученного DWG-файла в системе AutoCAD

Пример 74. Вызов MATLAB из вашей программы на Visual C# 12

Из вашей программы можно вызывать MATLAB! Среда MATLAB является стандартным мощным инструментом для работы в различных отраслях математики. При подготовке этого примера автор пользовался версией MATLAB 7.12.0.635 (R2011a). Продемонстрируем здесь 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 задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новый проект назовем `MatlabВызов`. 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 9.9.

Листинг 9.9. Использование возможностей среды MATLAB

```
// Программа вызывает простейшую функцию MATLAB
// ~ ~ ~ ~ ~ ~ ~ ~
// Для успешной работы программы нет необходимости добавлять ссылку на
// объектную библиотеку через Project | Reference. Однако на компьютере
// MATLAB должен быть установлен
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace MatlabВызов
{
    public partial class Form1 : Form
    {
        public Form1()
```

```
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            button1.Text = "Вызвать MATLAB";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Получить тип приложения MATLAB:
            var ТипМатЛаб = Type.GetTypeFromProgID("Matlab.Application");
            // Создать экземпляр объекта типа MATLAB:
            var МатЛаб = Activator.CreateInstance(ТипМатЛаб);
            // Подготавливаем команды для MATLAB:
            Object[] Команды =
               { "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" };
            // { "surf(peaks)" };
            // Вызываем MATLAB, подавая ему на вход подготовленные команды:
            var Результат =
                      ТипМатЛаб.InvokeMember("Execute", System.Reflection.
                      BindingFlags.InvokeMethod, null, МатЛаб, Команды);
            // MessageBox.Show(Результат.ToString());
        }
    }
}
```

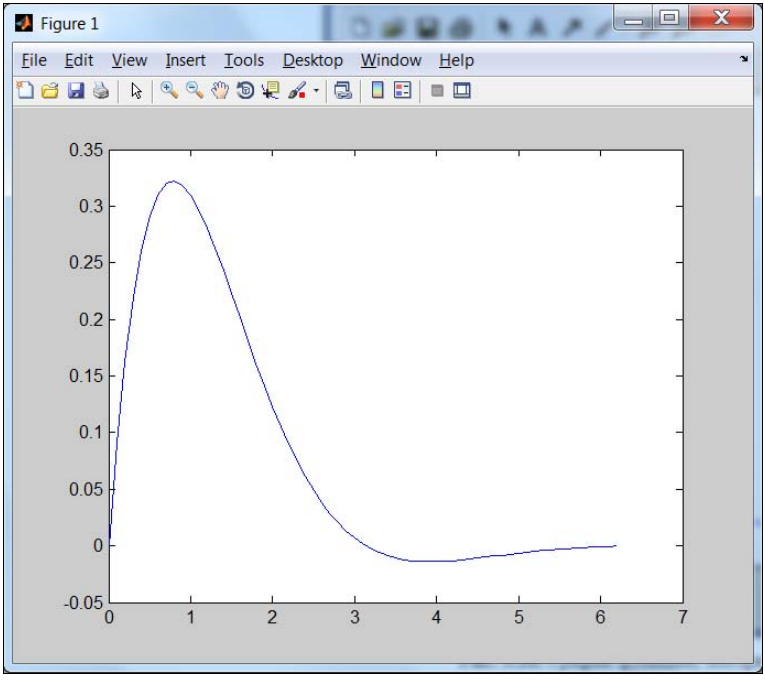


Рис. 9.10. График функции, построенный в среде MATLAB

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. 9.10).

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

Таким образом, мы показали принципиальную возможность взаимодействия с MATLAB. Убедиться в работоспособности приведенной программы можно, открыв решение MatlabВызов.sln в папке MatlabВызов сопровождающего книгу электронного архива.

Пример 75. 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 эволюционизировала с задач матричной алгебры, отсюда и слово MATLAB, которое 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 12 и в окне **New Project** выберем приложение шаблона **Windows Forms Application Visual C#**. Новую программу назовем `MatlabСлау`. Из панели элементов **Toolbox** перенесем командную кнопку **Button**. Далее на вкладке программного кода введем текст из листинга 9.10.

Листинг 9.10. 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 на экран с помощью MsgBox
// ~ ~ ~ ~ ~ ~ ~ ~
// Для успешной работы программы нет необходимости добавлять ссылку на
// объектную библиотеку через Project | Reference. Однако MATLAB должен
// быть установлен на вашем компьютере
using System;
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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            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:
            var ТипМатЛаб = Type.GetTypeFromProgID("Matlab.Application");
            // Создать экземпляр объекта типа MATLAB:
            var МатЛаб = Activator.CreateInstance(ТипМатЛаб);
            // Подготавливаем команды для MATLAB:
            Object[] Команды =
                 {"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:
            var Результат =
                    ТипМатЛаб.InvokeMember("Execute",
                    System.Reflection.BindingFlags.InvokeMethod,
                    null, МатЛаб, Команды);
            String Стринг = Результат.ToString();
            // Таким образом мы могли бы вывести решение на экран:
            // MessageBox.Show(Стринг);
            // Однако этот результат будет внутри строки, а хотелось бы
            // получить ответ, который можно легко преобразовать в массив
            // Double для дальнейшей обработки.
            // Удаляем подстроку "X =":
            Стринг = Стринг.Replace("X =", "");
```

```
            // Задаваясь в качестве сепаратора символами перевода каретки
            // '\n' и пробельными символами ' ',
            Char[] Сепаратор = { '\n', ' ' };
            // делим весь текст на строки с удалением пустых строк:
            var МассивX = Стринг.Split(Сепаратор,
                                StringSplitOptions.RemoveEmptyEntries);
            var Строка = String.Format("Неизвестные равны:\n" +
                              "X1 = {0};  X2 = {1};  X3 = {2}.", МассивX);
            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. Поэтому выделим из строки решения каждое значение неизвестного с помощью операций со строками, используя функцию `Split`, которая возвращает строковый массив, содержащий подстроки данного экземпляра.

На рис. 9.11 приведен результат работы программы.

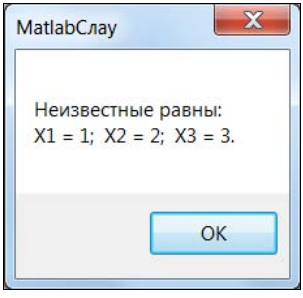


Рис. 9.11. Решение системы трех уравнений с помощью MATLAB

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 MatlabСлау.sln в папке MatlabСлау сопровождающего книгу электронного архива.

Пример 76. Создание PDF-файла "на лету" с возможностью вывода кириллицы

Представьте себе ситуацию, когда требуется вывести отчетный документ (например, результаты технических вычислений или какую-либо бухгалтерскую отчетность), но важно, чтобы пользователь вашей программы не внес никаких корректив в этот документ. Например, вы хотите, чтобы пользователь не удалял ссылку на вас, как на разработчика программы, чтобы он не вносил исправления в результаты вычислений, и проч.

Если вы пишете какое-либо веб-приложение, и согласно сценарию интерактивного взаимодействия с пользователем формируется, например, квитанция для оплаты ваших услуг, то становится чрезвычайно важным, чтобы пользователь ни случайно, ни сознательно не мог внести никакие изменения в этот финансовый документ. Подобные задачи помогает решать PDF-формат. Формат PDF гарантирует, что при просмотре файла в сети или выводе на печать вид файла останется неизменным, и данные файла не смогут быть легко изменены. Формат PDF стал международным стандартом. PDF-файл легко напечатать и использовать для совместной работы, однако трудно изменить (можно, но трудно).

Примерами таких ситуаций могут служить резюме, юридические документы, бюллетени и любые другие файлы, предназначенные главным образом для чтения и печати. Для просмотра PDF-файла можно использовать официальную бесплатную программу Adobe Reader, а также веб-браузер, например, Internet Explorer или Mozilla Firefox (они открывают PDF-документ в отдельной вкладке браузера). Последний факт оказывается очень важным при создании PDF-файла "на лету" ("on the fly") во время диалога пользователя с интерактивной веб-страницей. Приведем пример создания PDF-файла в ходе работы настольного Windows-приложения.

Решить задачу было бы совсем просто, если бы мы с вами жили в англоязычной стране. В этом случае проблем, связанных с выводом символов кириллицы, быть не может, и программа свелась бы к написанию программного кода из пяти строчек. Такую маленькую программу мы сейчас и напишем, чтобы изложить решение по принципу "от простого с сложному". Для этого запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Console Application**. Дадим проекту имя: `Создать_PDF_1`.

Поскольку Microsoft не разработала инструментов для генерации PDF, мы воспользуемся объектной библиотекой itextsharp.dll, которую бесплатно распространяет американская компания "iText Software" на сайте **www.itextsharp.com**. Эту библиотеку вы можете либо скачать с упомянутого сайта (а также ознакомиться с документацией), либо скопировать из архива с примерами к этой книге.

Теперь в текущий проект необходимо добавить ссылку на эту библиотеку. Для этого в пункте меню **Project | Properties** выберем команду **Add Reference**, затем на вкладке **Browse** (Обзор) найдем названную библиотеку. Далее на вкладке программного кода введем текст из листинга 9.11.

Листинг 9.11. Создание простейшего PDF-документа, содержащего английский текст

```
// Данная программа "на лету" генерирует PDF-документ
// Добавляем ссылку на библиотеку itextsharp.dll:
// Project | Properties | Add References и на вкладке Browse (Обзор)
// находим файл itextsharp.dll.
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace Создать_PDF_1
{
    class Program
    {
        static void Main(string[] args)
```

```
        {
            // В текущем каталоге создаем PDF-документ:
            var Документ = new iTextSharp.text.Document();
            iTextSharp.text.pdf.PdfWriter.GetInstance(
                   Документ, new System.IO.FileStream(
                "Отчет.pdf", System.IO.FileMode.Create));
            Документ.Open();
            Документ.Add(
                new iTextSharp.text.Paragraph("Hello !" + "\n" +
                                      "Do you like Visual C# 12 ?"));
            Документ.Close();
            // PDF-документ можно открыть с помощью Adobe Acrobat,
            // если он установлен:
            System.Diagnostics.Process.Start("Отчет.pdf");
            // Также созданный PDF-документ можно открыть с помощью
            // какого-либо веб-обозревателя, например, Internet Explorer:
            // var ТекущийКаталог = System.IO.Directory.
            //                              GetCurrentDirectory();
            // System.Diagnostics.Process.Start(
            //          "IExplore.exe", ТекущийКаталог + @"\Отчет.pdf");
        }
    }
}
```

Как видно из текста программы, вначале создаем PDF-документ класса `Document` из объектной библиотеки itextsharp.dll. Этот документ будет записан в файл Отчет.pdf после закрытия документа (`Close`). В документ добавлен абзац (`Paragraph`) с текстом на английском языке. Последней строчкой созданный PDF-файл вызывается с помощью Adobe Acrobat (если он установлен на вашем компьютере, если нет, то помните, что PDF-файлы также открывает интернет-браузер, например, Internet Explorer). Результат работы программы мы не приводим, поскольку он очевиден.

Задача решена, однако нет чувства удовлетворения, поскольку мы пока не научились выводить в PDF-документ русские буквы. Для этого *необходимо задать шрифт с кодировкой Windows 1251*. Как это сделать, рассмотрим в следующем примере. Запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Console Application**. Дадим проекту имя: `Создать_PDF_2`. Далее, так же как и в предыдущем случае, добавим к нашему проекту ссылку на объектную библиотеку itextsharp.dll. Затем на вкладке программного кода введем текст из листинга 9.12.

Листинг 9.12. Создание PDF-документа с текстом на русском языке

```
// Данная программа "на лету" генерирует PDF-документ
// ~ ~ ~ ~ ~ ~ ~ ~
// Добавляем ссылку на библиотеку itextsharp.dll:
// Project | Add References и на вкладке Browse (Обзор)
// находим файл itextsharp.dll.
using System;
```

```
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Добавляем пространство имен для более коротких выражений:
using iTextSharp.text.pdf;
namespace Создать_PDF_2
{
    class Program
    {
        static void Main(string[] args)
        {
            // В текущем каталоге создаем PDF-документ:
            var Документ = new iTextSharp.text.Document();
            PdfWriter.GetInstance(
                   Документ, new System.IO.FileStream(
               "Отчет.pdf", System.IO.FileMode.Create));
            Документ.Open();
            // Базовый шрифт создаем, используя один из шрифтов из папки
            // Windows:
            var БазовыйШрифт = BaseFont.CreateFont(
              @"C:\WINDOWS\Fonts\comic.ttf", "CP1251", BaseFont.EMBEDDED);
            // @"C:\WINDOWS\Fonts\times.ttf", "CP1251", BaseFont.EMBEDDED);
            // @"C:\WINDOWS\Fonts\CONSOLA.TTF", "CP1251", BaseFont.EMBEDDED);
            // Задаем шрифт размером 10 пунктов. Можно задать
            // шрифт Font.ITALIC (наклонный) или Font.BOLD (жирный):
            var Шрифт = new iTextSharp.text.Font(
                           БазовыйШрифт, 10, iTextSharp.text.Font.NORMAL);
            Документ.Add(new iTextSharp.text.Paragraph(
                            "Здравствуйте !" + "\n" +
                            "Вы увлекаетесь Visual C# 12?", Шрифт));
            Документ.Close();
            // PDF-документ можно открыть с помощью интернет-браузера:
            // System.Diagnostics.Process.Start("IExplore.exe", System.IO.
            //           Directory.GetCurrentDirectory() + @"\Отчет.pdf");
            // PDF-документ можно открыть с помощью Adobe Acrobat,
            // если он установлен:
            System.Diagnostics.Process.Start("Отчет.pdf");
        }
    }
}
```

Как видно из текста программы, в ее начале мы также создаем PDF-документ, но далее задаем базовый шрифт на основе какого-либо имеющегося на нашем компьютере шрифта в папке операционной системы C:\Windows\Fonts. Заметим, что для создания документа минимального объема следует использовать векторную графику и так называемые "безопасные" шрифты. Всего имеется 14 таких шрифтов:

- Times (обычный, курсив, полужирный и полужирный курсив);
- Courier (обычный, наклонный, полужирный и полужирный наклонный);

- Helvetica (обычный, наклонный, полужирный и полужирный наклонный);
- Symbol;
- Zapf Dingbats.

Эти шрифты можно использовать без внедрения в документ — их должны правильно отображать все программы. Любые другие шрифты, которые не были внедрены в документ и отсутствуют в системе, будут отображаться одним из имеющихся, что может стать причиной увеличения или уменьшения числа страниц, количества символов в строке, межстрочного интервала и других неприятностей, связанных с метрикой шрифта.

В результате работы данной программы программа-просмотрщик PDF-документов Adobe Reader откроет созданный файл (рис. 9.12).

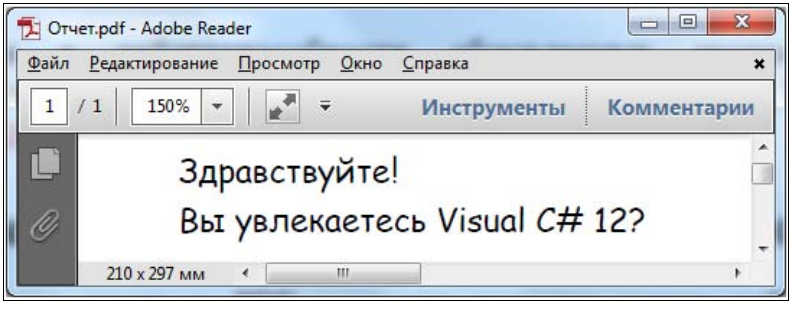


Рис. 9.12. Отображение созданного PDF-файла

Убедиться в работоспособности обсуждаемых программ можно, открыв соответствующие решения в папках Создать_PDF_1 и Создать_PDF_2 сопровождающего книгу электронного архива.

Пример 77. Вывод таблицы в PDF-документ

Очень часто всего в PDF-документ выводят какую-либо таблицу. Объектная библиотека itextsharp.dll компании "iText Software" *(см. пример 76)* предоставляет соответствующие классы для решения этой задачи. В этом примере мы рассмотрим два варианта таблицы. Первый вариант — это вариант "широкой" таблицы с многими колонками на всю страницу формата А4 (рис. 9.13). Мы условно назвали ее "широкой", поскольку при взгляде на нее не возникает ощущения пустого места вокруг таблицы. Но если бы такое ощущение возникало, то необходимо было бы компоновать все страницу так, чтобы оптимально заполнялись все подобные пустые места либо другой таблицей, либо каким-нибудь изображением, либо каким-нибудь текстом. А это как раз второй вариант, который мы условно назовем вариантом "узкой" таблицы (см. рис. 9.14).

Таким образом, первую задачу мы ставим так: вывести "широкую" таблицу в PDF-файл. Для ее решения запускаем Visual Studio 12 и в окне **New Project** выбираем шаблон **Console Application**. Дадим проекту имя: `Создать_PDF_Табл_1`. Теперь в текущий проект необходимо добавить ссылку на объектную библиотеку itextsharp.dll. Для этого в пункте меню **Project | Properties** выберем команду **Add Reference**, затем на вкладке

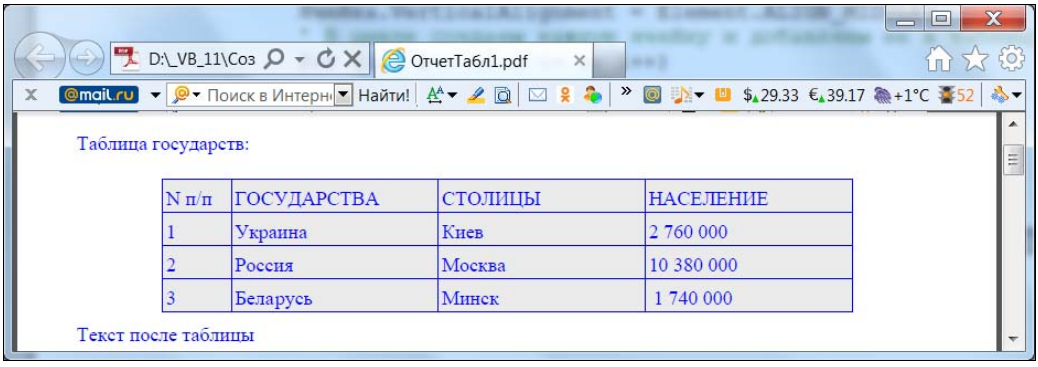


Рис. 9.13. Отображение браузером Internet Explorer созданного PDF-файла

Browse (Обзор) найдем названную библиотеку. Далее на вкладке программного кода введем текст из листинга 9.13.

Листинг 9.13. Вывод таблицы в PDF-документ (вариант 1)

```
// Программа "на лету" создает PDF-файл и записывает в этот
// файл "широкую" таблицу
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Для более компактных выражений добавим эти две директивы:
using iTextSharp.text;
using iTextSharp.text.pdf;
namespace Создать_PDF_Табл_1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Инициализируем четыре строковых массива:
            String[] Номер_п_п = { "N п/п", "1", "2", "3" };
            String[] Страны = {
                    "ГОСУДАРСТВА", "Украина", "Россия", "Беларусь" };
            String[] Столицы = { "СТОЛИЦЫ", "Киев", "Москва", "Минск" };
            String[] Население = {
                    "НАСЕЛЕНИЕ", "2 760 000", "10 380 000", " 1 740 000" };
            // В текущем каталоге создаем PDF-документ:
            var Документ = new Document();
            var Писатель = PdfWriter.GetInstance(Документ, new System.
              IO.FileStream("ОтчетТабл1.pdf", System.IO.FileMode.Create));
            // System.IO.FileMode.Create — если такой файл уже есть, то он
            // будет удален, а новый создан
            Документ.Open();
```

```
// Базовый шрифт создаем из одного из шрифтов из папки Windows:
var БазовыйШрифт = BaseFont.CreateFont(
  @"C:\WINDOWS\Fonts\times.ttf", "CP1251", BaseFont.EMBEDDED);
// @"C:\WINDOWS\Fonts\comic.ttf", "CP1251", BaseFont.EMBEDDED);
// @"C:\WINDOWS\Fonts\CONSOLA.TTF", "CP1251",
//                                          BaseFont.EMBEDDED);
// Заказываем шрифт размером 10 пунктов. Можно заказать
// шрифт Font.ITALIC (наклонный) или Font.BOLD (жирный):
var Шрифт = new Font(БазовыйШрифт, 12, Font.NORMAL);
// Цвет текста:
Шрифт.Color = BaseColor.BLUE;
// Некоторый текст до таблицы:
Документ.Add(new Paragraph(
             "Таблица государств:" + "\n\n", Шрифт));
// Заказываем таблицу с четырьмя колонками:
var Табл = new PdfPTable(4);
// Таблицу выровнять в центр:
Табл.HorizontalAlignment = Element.ALIGN_CENTER; // ALIGN_LEFT
var Ячейка = new PdfPCell(new Phrase("Ячейка", Шрифт));
// Пропорции размеров колонок в таблице:
Single[] ШиринаКолонок = { 10.0F, 30.0F, 30.0F, 30.0F };
Табл.SetTotalWidth(ШиринаКолонок);
// Степень заливки ячейки:
Ячейка.GrayFill = 0.92F;
// Высота ячейки:
Ячейка.FixedHeight = 20.0F;
// Цвет границ ячеек:
Ячейка.BorderColor = BaseColor.BLUE;
// Выравнивание содержимого ячеек:
Ячейка.HorizontalAlignment = Element.ALIGN_LEFT;// ALIGN_CENTER
Ячейка.VerticalAlignment = Element.ALIGN_MIDDLE;
// В цикле создаем каждую ячейку и добавляем ее в таблицу:
for (int i = 0; i <= 3; i++)
{
    Ячейка.Phrase = new Phrase(Номер_п_п[i], Шрифт);
    Табл.AddCell(Ячейка);
    Ячейка.Phrase = new Phrase(Страны[i], Шрифт);
    Табл.AddCell(Ячейка);
    Ячейка.Phrase = new Phrase(Столицы[i], Шрифт);
    Табл.AddCell(Ячейка);
    Ячейка.Phrase = new Phrase(Население[i], Шрифт);
    Табл.AddCell(Ячейка);
}
Документ.Add(Табл);
Документ.Add(new Paragraph("Текст после таблицы", Шрифт));
Документ.Close(); Писатель.Close();
// PDF-документ можно открыть с помощью интернет-браузера:
System.Diagnostics.Process.Start("IExplore.exe", System.IO.
         Directory.GetCurrentDirectory() + @"\ОтчетТабл1.pdf");
```

```
            // PDF-документ можно открыть с помощью Adobe Acrobat,
            // если он установлен:
            System.Diagnostics.Process.Start("ОтчетТабл1.pdf");
        }
    }
}
```

Как видно из программного кода, операторы создания PDF-документа и задания нужного шрифта аналогичны примененным в *примере 76*. Далее, программируя непосредственно таблицу, мы создаем объект класса `PdfPTable`. Этот объект содержит ряд свойств и методов для построения таблицы. Составной частью любой таблицы, ее "кирпичиком", является ячейка. Ячейку таблицы задаем, используя класс `PdfPCell`. В цикле добавляем в заданную таблицу создаваемые ячейки. Последним оператором открываем созданный PDF-документ с помощью браузера Internet Explorer (см. рис. 9.13).

К недостаткам приведенного подхода отнесем невозможность регулировать ширину всей таблицы, поскольку таблица всегда будет располагаться на всю ширину страницы. Теперь решим эту же задачу по выводу таблицы в PDF-документ несколько другим способом. И выведем так называемую "узкую" таблицу с малым числом колонок.

Для решения этой задачи, так же как и в предыдущем случае, запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Console Application**. Дадим проекту имя: `Создать_PDF_Табл_2`. Необходимым условием работы программы является добавление в текущий проект ссылки на объектную библиотеку itextsharp.dll. Для этого в пункте меню **Project | Properties** выберем команду **Add Reference**, затем на вкладке **Browse** (Обзор) найдем названную библиотеку. Далее на вкладке программного кода введем текст из листинга 9.14.

Листинг 9.14. Вывод таблицы в PDF-документ (вариант 2)

```
// Программа "на лету" создает PDF-файл и записывает в этот
// файл "узкую" таблицу с малым числом колонок
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Следует добавить эти две директивы:
using iTextSharp.text;
using iTextSharp.text.pdf;

namespace Создать_PDF_Табл_2
{
    class Program
    {
        static void Main(string[] args)
        {
            // Инициализируем два строковых массива:
            String[] Столицы = { "СТОЛИЦЫ", "Киев", "Москва", "Минск" };
            String[] Страны = {
                     "ГОСУДАРСТВА", "Украина", "Россия", "Беларусь" };
```

```
// В текущем каталоге создаем PDF-документ:
var Документ = new Document();
var Писатель = PdfWriter.GetInstance(Документ, new System.
 IO.FileStream("ОтчетТабл.pdf", System.IO.FileMode.Create));
// System.IO.FileMode.Create — если такой файл уже есть, то он
// будет удален, а новый создан
Документ.Open();
// Базовый шрифт создаем из одного из шрифтов из папки Windows:
var БазовыйШрифт = BaseFont.CreateFont(
   @"C:\WINDOWS\Fonts\times.ttf", "CP1251", BaseFont.EMBEDDED);
// @"C:\WINDOWS\Fonts\comic.ttf", "CP1251", BaseFont.EMBEDDED);
// @"C:\WINDOWS\Fonts\CONSOLA.TTF", "CP1251",
//                                          BaseFont.EMBEDDED);
// Заказываем шрифт размером 10 пунктов. Можно заказать
// шрифт Font.ITALIC (наклонный) или Font.BOLD (жирный):
var Шрифт = new Font(БазовыйШрифт, 10,
                              Font.NORMAL, BaseColor.BLUE);
// или цвет текста отдельно: Шрифт.Color = BaseColor.RED:
var Таблица = new PdfPTable(2);
var Ячейка = new PdfPCell();
Ячейка.HorizontalAlignment = Element.ALIGN_LEFT; // ALIGN_CENTER
// Две ячейки объединить в одну:
Ячейка.Colspan = 2;
// Границы ячейки не показывать:
Ячейка.Border = 0;
// Высота ячейки:
Ячейка.FixedHeight = 16.0F;
Ячейка.Phrase = new Phrase(
                    "Какой-либо текст до таблицы", Шрифт);
Таблица.AddCell(Ячейка);
Ячейка.BackgroundColor = BaseColor.LIGHT_GRAY;
// Не объединять ячейки:
Ячейка.Colspan = 1;
// Границы ячеек показывать:
Ячейка.Border = 15;
for (int i = 0; i <= 3; i++)
{
    Ячейка.Phrase = new Phrase(Страны[i], Шрифт);
    Таблица.AddCell(Ячейка);
    Ячейка.Phrase = new Phrase(Столицы[i], Шрифт);
    Таблица.AddCell(Ячейка);
}
Ячейка.Colspan = 2; // две ячейки объединить в одну
Ячейка.Border = 0;  // не показывать границ ячейки
Ячейка.Phrase = new Phrase(
         "Некоторый текст после таблицы", Шрифт);
Ячейка.BackgroundColor = BaseColor.WHITE;
Таблица.AddCell(Ячейка);
// Регулируем ширину таблицы:
Таблица.TotalWidth = Документ.PageSize.Width — 400;
```

```
            // Третий и четвертый параметры — это координаты
            // левого верхнего угла таблицы:
            Таблица.WriteSelectedRows(0, -1, 40,
                              Документ.PageSize.Height — 30,
                              Писатель.DirectContent);
            Документ.Close(); Писатель.Close();
            // PDF-документ можно открыть с помощью интернет-браузера:
            System.Diagnostics.Process.Start("IExplore.exe", System.IO.
                   Directory.GetCurrentDirectory() + @"\ОтчетТабл.pdf");
        }
    }
}
```

В программном коде появилась возможность регулировать ширину таблицы (рис. 9.14) благодаря методу `WriteSelectedRows`, который выводит таблицу по координатам левого верхнего угла. Такой подход позволяет компоновать страницу так, как это делают при создании HTML-страницы. Очень часто HTML-страницу разбивают на ячейки невидимой таблицы, при этом какие-то ячейки объединяют, при этом размещение данных в такой невидимой таблице становится оптимальным.

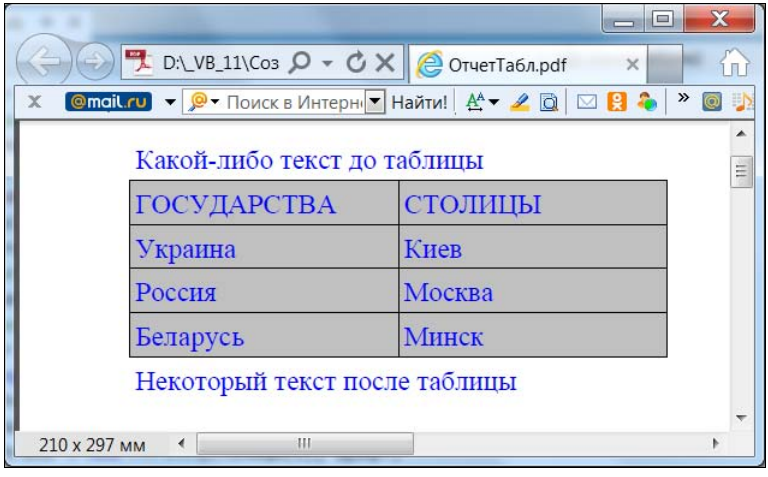


Рис. 9.14. Отображение "узкой" таблицы в браузере Internet Explorer

Убедиться в работоспособности обсуждаемых программ можно, открыв соответствующие решения в папках Создать_PDF_Табл_1 и Создать_PDF_Табл_2 сопровождающего книгу электронного архива.

Пример 78. Вывод графических данных в PDF-документ

Вначале покажем, как можно вывести файл растровой графики в PDF-документ. Для этого запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Console Application**. Дадим проекту имя: `Создать_PDF_граф_1`. Далее добавим в текущий проект

ссылку на объектную библиотеку itextsharp.dll. Для этого в пункте меню **Project | Properties** выберем команду **Add Reference**, затем на вкладке **Browse** (Обзор) найдем названную библиотеку. Далее на вкладке программного кода введем текст из листинга 9.15.

Листинг 9.15. Вывод отображений графических файлов в PDF-документ

```
// Программа выводит в PDF-документ два изображения. Чтобы оптимальным
// образом их компоновать, эти два изображения выводим в ячейки некоторой
// таблицы, состоящей из двух колонок и трех строк. Ячейки в первой
// строке объединены в одну ячейку, и в нее мы выводим текст "Как же нам
// найти гармонию между Ж и М?". В две ячейки второй строки мы выводим
// два изображения. Ячейки в третьей строке мы также объединили в одну
// ячейку и также вывели здесь текст. Границы первой строки и последней
// мы не показываем
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Следует добавить эти две директивы:
using iTextSharp.text;
using iTextSharp.text.pdf;
namespace Создать_PDF_граф_1
{
    class Program
    {
        static void Main(string[] args)
        {
                    // В текущем каталоге создаем PDF-документ:
        var Документ = new Document();
        var Писатель = PdfWriter.GetInstance(Документ, new System.
             IO.FileStream("ТаблГраф.pdf", System.IO.FileMode.Create));
        // System.IO.FileMode.Create — если такой файл уже есть, то он
        // будет удален, а новый создан
        Документ.Open();
        // Базовый шрифт создаем из одного из шрифтов из папки Windows:
        var БазовыйШрифт = BaseFont.CreateFont(
                @"C:\WINDOWS\Fonts\comic.ttf", "CP1251", BaseFont.EMBEDDED);
        // Заказываем шрифт размером 10 пунктов. Можно заказать
        // шрифт Font.ITALIC (наклонный) или Font.BOLD (жирный):
        var Шрифт = new Font(БазовыйШрифт, 10, Font.NORMAL, BaseColor.BLUE);
        // или цвет текста отдельно: Шрифт.Color = BaseColor.RED
        var Таблица = new PdfPTable(2);
        var Ячейка = new PdfPCell();
        Ячейка.HorizontalAlignment = Element.ALIGN_CENTER;
        // Две ячейки объединить в одну:
        Ячейка.Colspan = 2;
        // Границы ячейки не показывать:
        Ячейка.Border = 0;
```

```
        // Высота ячейки:
        Ячейка.FixedHeight = 16.0F;
        Ячейка.Phrase = new Phrase(
                        "Как же нам найти гармонию между Ж и М?", Шрифт);
        Таблица.AddCell(Ячейка);
        // Не объединять ячейки:
        Ячейка.Colspan = 1;
        // Границы ячеек показывать:
        Ячейка.Border = 15;
        // Увеличиваем высоту ячеек для изображений:
        Ячейка.FixedHeight = 83.0F;
        // Вставляем изображения в ячейки:
        Ячейка.Image = Image.GetInstance(@"D:\g.jpg");
        Таблица.AddCell(Ячейка);
        Ячейка.Image = Image.GetInstance(@"D:\m.jpg");
        Таблица.AddCell(Ячейка);
        Ячейка.Colspan = 2; // две ячейки объединить в одну
        Ячейка.Border = 0;  // не показывать границ ячейки
        Ячейка.Phrase = new Phrase(
               "Может, примем друг друга такими, какие мы есть?", Шрифт);
        Ячейка.BackgroundColor = BaseColor.WHITE;
        Таблица.AddCell(Ячейка);
        // Регулируем ширину таблицы:
        Таблица.TotalWidth = Документ.PageSize.Width — 380;
        // Третий и четвертый параметры — это координаты
        // левого верхнего угла таблицы:
        Таблица.WriteSelectedRows(0, -1, 40,
                 Документ.PageSize.Height — 30, Писатель.DirectContent);
        Документ.Close() ; Писатель.Close();
        // PDF-документ можно открыть с помощью интернет-браузера:
        System.Diagnostics.Process.Start("IExplore.exe", System.IO.
                     Directory.GetCurrentDirectory() + @"\ТаблГраф.pdf");
        }
    }
}
```

Эта программа очень похожа на последнюю программу из *примера 77*, только здесь мы во второй строке таблицы выводим в двух ячейках, соответственно, отображения двух графических файлов. Результат работы программы показан на рис. 9.15.

Очень часто изображение, подлежащее выводу в PDF-файл, требуется сформировать в ходе работы программы — заранее на все случаи жизни подготовить графические файлы не получается. Поэтому в примере, представленном далее, мы покажем, как можно в ходе работы программы сформировать изображение методами класса `Graphics`, а затем вывести это изображение в PDF-документ. Причем сделаем это в консольном приложении среды, которое, вообще говоря, не содержит объектной библиотеки, *поддерживающей класс Graphics*. Это естественно, поскольку в консольном приложении нет экранной формы и выводить графику некуда (разве что в растровый файл). Таким образом, изображение мы создадим "виртуальное", без "подложки", но выведем его в PDF-файл.

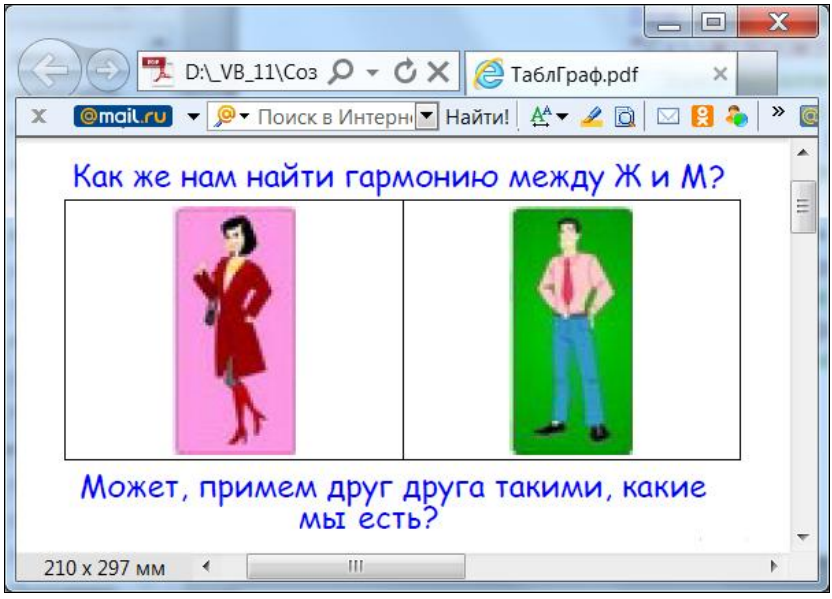


Рис. 9.15. Вывод отображений двух графических файлов

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Console Application**. Дадим проекту имя: `Создать_PDF_граф_2`. Далее, чтобы получить доступ к графическим возможностям, добавим ссылку на объектную библиотеку System.Drawing.dll. Для этого выполним команду меню **Project | Add Reference | Framework**, затем в появившемся списке установим флажок напротив ссылки **System.Drawing**. Также добавим в текущий проект ссылку на объектную библиотеку itextsharp.dll для работы с PDF-документом. Для этого, аналогично, в пункте меню **Project | Properties** выберем команду **Add Reference**, затем на вкладке **Browse** (Обзор) найдем названную библиотеку. Далее на вкладке программного кода введем текст из листинга 9.16.

Листинг 9.16. Вывод изображения, сформированного в консольном приложении, в PDF-файл

```
// Программа формирует изображение методами класса Graphics. Рисуем
// текстовую строку с использованием линейного градиента. Чтобы
// подчеркнуть, что мы создаем именно рисунок, а не просто текст,
// текстовую строку разворачиваем на некоторый угол к горизонту.
// Далее выводим сформированное изображение из оперативной памяти
// в PDF-файл
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Следует добавить эти три директивы:
using iTextSharp.text;
using iTextSharp.text.pdf;
using System.Drawing;
```

```
// Здесь добавлены ссылки на две объектные библиотеки:
// System.Drawing.dll и itextsharp.dll
namespace Создать_PDF_граф_2
{
    class Program
    {
        static void Main(string[] args)
        {
            // Создаем точечное изображение размером 415x45 точек
            // с глубиной цвета 24
            var Рисунок = new Bitmap(415, 45,
                          System.Drawing.Imaging.
                          PixelFormat.Format24bppRgb);
            // Создаем новый объект класса Graphics из изображения Рисунок:
            var Графика = Graphics.FromImage(Рисунок);
            var Точка1 = new PointF(20.0F, 20.0F);
            var Точка2 = new PointF(415.0F, 20.0F);
            var Градиент = new System.Drawing.Drawing2D.
                               LinearGradientBrush(Точка1, Точка2,
                               Color.Yellow, Color.Red);
            var Шрифт = new System.Drawing.Font("Times new Roman", 14.0F);
            Графика.Clear(Color.White);
            // Разворачиваем мир на 356 градусов по часовой стрелке:
            Графика.RotateTransform(356.0F);
            // 20.0f, 20.0f — это координаты левого нижнего угла строки:
            Графика.DrawString("Записываем графику в PDF-документ",
                                          Шрифт, Градиент, 20.0F, 20.0F);
            // Освобождение ресурсов:
            Графика.Dispose();
            // Создаем PDF-документ:
            var Документ = new Document();
            var Писатель = PdfWriter.GetInstance(Документ, new System.IO.
                           FileStream("ТаблГраф.pdf",
                           System.IO.FileMode.Create));
            Документ.Open();
            var Изо = iTextSharp.text.Image.
                                  GetInstance(Рисунок, BaseColor.WHITE);
            Документ.Add(Изо);
            Документ.Close();
            Рисунок.Dispose();
            System.Diagnostics.Process.Start("ТаблГраф.pdf");
        }
    }
}
```

Как видно из программного кода, создаем экземпляр класса `Graphics` на основе изображения класса `Image` с заданным размером и форматом. Далее создаем горизонтальный линейный градиент между двумя точками от желтого цвета к красному. Градиент обеспечивает постепенный переход от одного цвета к другому. Сформированный рисунок

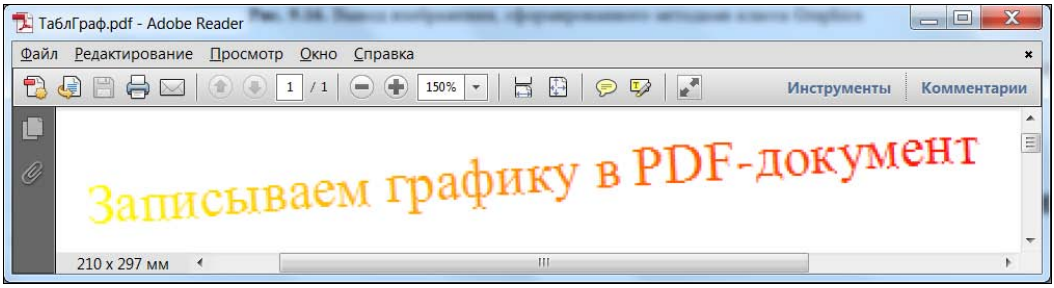


Рис. 9.16. Вывод изображения, сформированного методами класса `Graphics`

передаем с преобразованием в PDF-изображение методом `GetInstance`. Затем обычным способом выводим изображение в PDF-документ (рис. 9.16).

Убедиться в работоспособности обсуждаемых программ можно, открыв соответствующие решения в папках Создать_PDF_граф_1 и Создать_PDF_граф_2 сопровождающего книгу электронного архива.

ГЛАВА 10

Обработка баз данных с использованием технологии ADO.NET

Пример 79. Создание базы данных SQL Server

Покажем, как можно создать базу данных SQL Server в среде Visual Studio 12. 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**.

Для этого запустим Visual Studio 12, в окне **New Project** выберем шаблон **Windows Forms Application Visual C#** и дадим проекту имя: `БД_SQL_Server`. Далее непосредственно создаем новую базу данных SQL Server. Для этого в пункте меню **Project** выполним команды **Add New Item | Data | Local Database**. В качестве имени базы данных выберем имя проекта, т. е. **БД_SQL_Server.sdf**. После щелчка на кнопке **Add** откроется окно **Data Source Configuration Wizard**. Здесь мы выберем пиктограмму **Dataset** и щелкнем последовательно на кнопках **Next** и **Finish**.

Теперь в окне обозревателя решений **Solution Explorer** мы увидим название новой появившейся базы данных **БД_SQL_Server.sdf**. Убедимся, что этот файл появился в окне обозревателя решений, и теперь уже здесь двойным щелчком щелкнем на его изображении. При этом откроется окно **Server Explorer** (Обозреватель баз данных). В этом окне развернем щелчком мыши узел **БД_SQL_Server.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 (в 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 числе наших знакомых могут быть люди с одинаковыми именами или одинаковыми фамилиями, так что в создаваемой таблице в качестве первичных ключей следует использовать одновременно два столбца: столбец **Имя** и столбец **Фамилия**. Представьте себе, что в таблице уже есть сотни записей, и при попытке ввести вторую строку, содержащую уже имеющееся значение, появляется сообщение об ошибке. Это очень технологично, удобно!

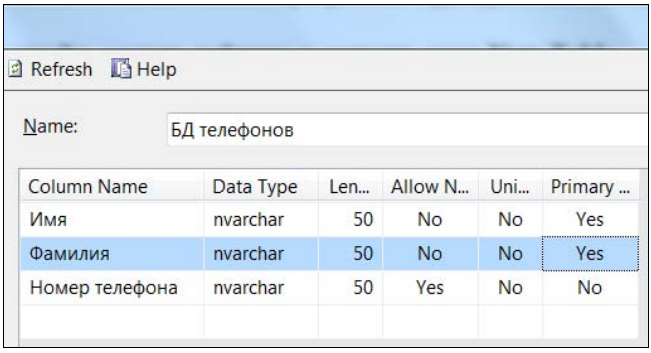


Рис. 10.1. Формирование структуры новой таблицы

Чтобы добавить в таблицу первичные ключи, 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** (разрешить нулевые значения) 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** (Показать таблицу данных). Откроется окно данных таблицы (рис. 10.2), но только пока пустое.

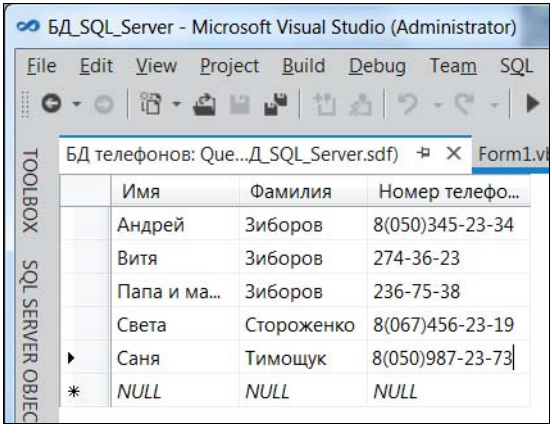


Рис. 10.2. Заполнение ячеек таблицы

Теперь заполним поля таблицы. Для нашей демонстрационной цели введем в таблицу пять строчек. После ввода в меню **File** выберите команду **Save All** для сохранения проекта и базы данных. Затем убедитесь, что в папке проекта имеется файл БД_SQL_Server.sdf.

Пример базы данных можно найти в папке БД_SQL_Server сопровождающего книгу электронного архива.

> ***Напоминание***
>
> Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 80. Отображение таблицы базы данных SQL Server на консоли

Имея базу данных, например, базу данных SQL Server в виде файла БД_SQL_Server.sdf, созданного в *примере 79*, покажем, как очень легко можно вывести таблицу из этой базы, например, на консоль.

Для упрощения строки подключения *скопируем файл* БД_SQL_Server.sdf, созданный в в *примере 79* в корневой каталог диска D: и запустим Visual Studio 12. Выберем в окне **New Project** шаблон **Console Application**. Зададим имя проекта — `БД_SQL_Server_Консоль`. Далее для манипулирования данными в базе SQL Server добавим в текущий проект объектную библиотеку System.Data.SqlServerCe.dll. Для этого выполним команды меню **Project | Add Reference | Extentions**, в появившемся списке ссылок установим флажок напротив ссылки **System.Data.SqlServerCe** и щелкнем на кнопке **OK**. В листинге 10.1 приведен программный код обсуждаемого консольного приложения.

Листинг 10.1. Чтение всех записей из таблицы БД SQL Server и вывод их на консоль

```
// Программа читает все записи из таблицы БД SQL Server (файл *.sdf) и
// выводит их на консоль с помощью объектов Command и DataReader
// ~ ~ ~ ~ ~ ~ ~ ~
// Добавим в наш проект объектную библиотеку System.Data.SqlServerCe.dll,
// для этого выберем пункты меню: Project | Add Reference и в появившемся
// списке выберем ссылку System.Data.SqlServerCe
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace БД_SQL_Server_Консоль
{
    class Program
    {
        static void Main(string[] args)
        {
            // Создаем объект класса Connection:
            var Подключение =
                            new System.Data.SqlServerCe.SqlCeConnection();
            Подключение.ConnectionString =
                                "Data Source=\"D:\\БД_SQL_Server.sdf\"";
            Подключение.Open();
```

```
            // Создаем объект класса Command: передавая ему SQL-команду
            var Команда =
                       new System.Data.SqlServerCe.SqlCeCommand();
            Команда.Connection = Подключение;
            // Передаем объекту Command SQL-команду:
            Команда.CommandText = "Select * From [БД телефонов]";
            // ДРУГИЕ КОМАНДЫ:
            // Выбрать все записи и сортировать их по колонке "Имя":
            // Команда.CommandText =
            //                "Select * From [БД телефонов] order by Имя";
            // Аналогично по колонке "Номер телефона":
            // Команда.CommandText =
            //   "Select * From [БД телефонов] ORDER BY [Номер телефона]";
            // Выполняем SQL-команду и закрываем подключение:
            var Читатель = Команда.
               ExecuteReader(System.Data.CommandBehavior.CloseConnection);
            // Задаем строку заголовка консоли:
            Console.Title = "Таблица БД:";
            Console.BackgroundColor = ConsoleColor.Cyan;  // — цвет фона
            Console.ForegroundColor = ConsoleColor.Black; // — цвет текста
            Console.Clear();
            // Выводим имена полей (колонок) таблицы:
            Console.WriteLine("{0,-11} {1,-11} {2,-15}" + "\n",
                              Читатель.GetName(0), Читатель.GetName(1),
                              Читатель.GetName(2));
            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,-11} {1,-11} {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 из текста программы, вначале создаем объект `Подключение` класса `Connection` и передаем ему строку подключения. Далее создаем объект класса `Command` и передаем ему простейшую SQL-команду:

```
Select * From [БД телефонов]
```

Она означает 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 **Имя** (`ORDER BY Имя`) и по колонке **Номер телефона** (`ORDER BY [Номер телефона]`). SQL-команда `ORDER BY` может быть записана как большими буквами, так и маленькими.

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. 10.3.

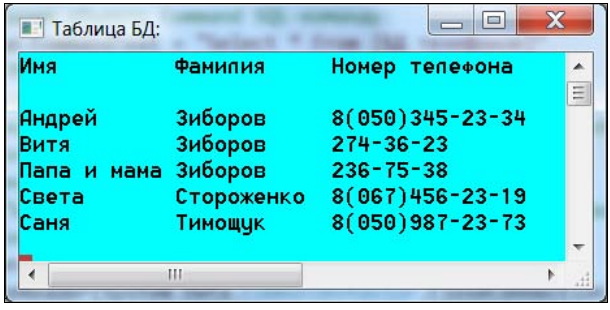


Рис. 10.3. Чтение таблицы базы данных SQL Server на консоль

Убедиться в работоспособности данной программы можно, открыв соответствующий SLN-файл решения в папке БД_SQL_Server_Консоль сопровождающего книгу электронного архива.

Пример 81. Редактирование таблицы базы данных MS Access в среде Visual Studio без написания программного кода

Создание базы данных в среде MS Access

Вначале создадим базу данных (БД) средствами Access пакета MS Office, например, версии 2010 или другой. 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 мы будем оперировать совсем маленькой БД.

Чтобы читатель этой книги смог реально повторить наши действия, создадим базу данных телефонов ваших контактов в виде таблицы следующей структуры (состава полей): **Номер п/п**, **ФИО** и **Номер телефона**. Для этого запускаем на компьютере офисное приложение Microsoft Access, в меню **Файл | Создать** выбираем команду **Новая база данных** (или нажимаем комбинацию клавиш <Ctrl>+<N>), задаем папку для размещения БД (зададим корневой каталог D:, чтобы строка подключения была более компакт-

ной) и имя файла — `vic.mdb`. Расширение файла mdb произошло от начальных букв Microsoft Data Base. Файл базы данных может иметь также расширение accdb (Access Data Base). Укажем тип файла базы данных: **Microsoft Access 2002-2003 (*.mbd)**.

Теперь щелкнем на кнопке **Создать**. В открывшемся окне **vic: база данных** выберем команду **Создание таблицы в режиме конструктора**. Далее задаем 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 для нее — введем: `БД телефонов`. В БД может быть несколько таблиц, а данную таблицу мы назвали именно так.

> ***Примечание***
>
> Заметьте, что при работе в обычных приложениях, если вы решили отказаться от внесенных в документ изменений, то его просто закрывают без сохранения. Однако при работе с БД все изменения сохраняются на диске без нашего ведома — запись на диск происходит по мере внесения изменений напрямую, минуя операционную систему.

Выполняем двойной щелчок в пределах созданной таблицы — и приступаем к ее заполнению. В нашем примере в таблице всего семь записей (рис. 10.4).

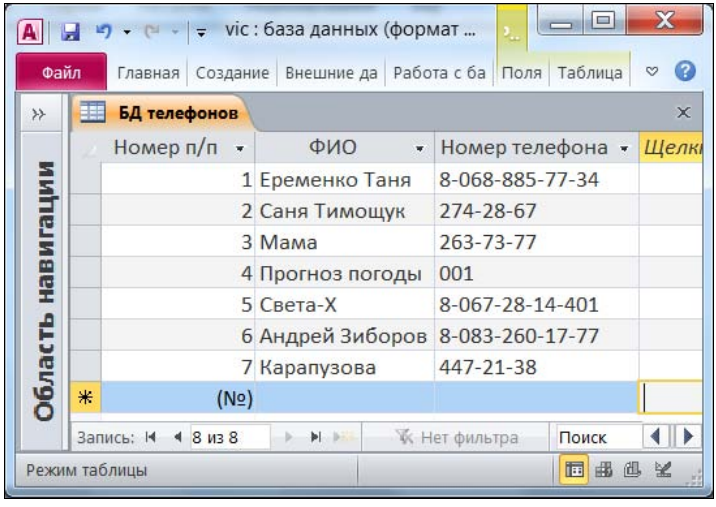


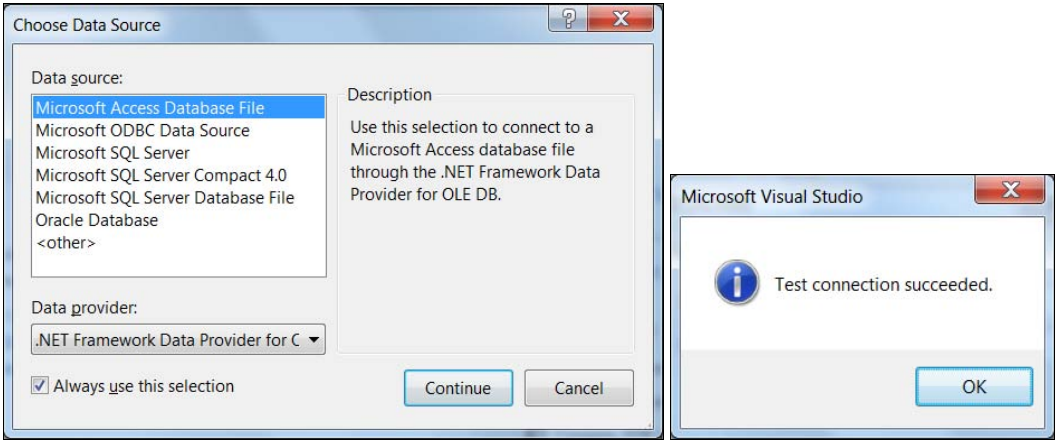
Рис. 10.4. Заполнение таблицы базы данных в среде MS Access

Теперь закроем БД Access и откроем созданную нами таблицу БД vic.mdb в среде Visual Studio 12.

Открытие базы данных Access в среде Visual Studio

Запускаем Visual Studio 12, однако сейчас мы не заказываем *новый* проект. Наша цель — открыть созданную нами базу данных в среде Visual Studio. Для этого выбираем пункт меню **View | Server Explorer** (также можно воспользоваться комбинацией клавиш <Ctrl>+<Alt>+<S>). Здесь, щелкнув правой кнопкой мыши на значке **Data Connections**, выбираем пункт **Add Connection** и в открывшемся окне **Choose Data**

Source (рис. 10.5, *а*) в области **Data source** (Источник данных) выбираем **Microsoft Access Database File** (OLE DB). Далее, щелкнув на кнопке **Continue**, в открывшемся окне с помощью кнопки **Browse** задаем путь и имя БД, например, D:\vic.mdb. Теперь проверяем подключение, нажав там же кнопку **Test Connection**. Успешное подключение выдаст сообщение, представленное на рис. 10.5, *б*.



а *б*

Рис. 10.5. Выбор источника данных (*а*) и тестирование подключения (*б*)

Проверка подключения выполнена, щелкаем на кнопке **OK**. Теперь в окне **Server Explorer** указателем мыши раскрываем узлы, символизирующие базу данных, таблицы, поля в таблице.

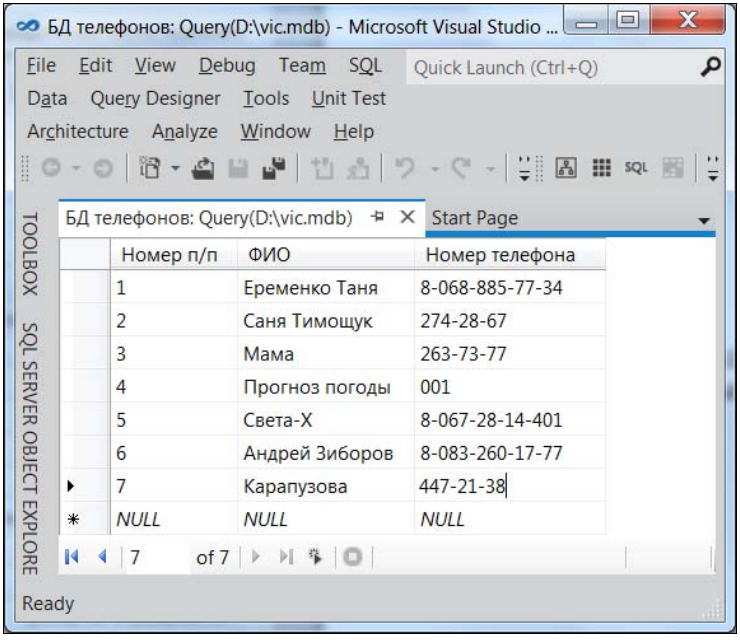


Рис. 10.6. Редактирование таблицы базы данных в среде Visual Studio

Затем щелкнем правой кнопкой мыши на узле **БД телефонов** и выберем команду **Retrieve Data**. В результате в правом окне получим содержимое этой таблицы (рис. 10.6). Здесь эту таблицу мы можем редактировать, т. е. изменять содержимое любой записи (**Update**), добавлять новые записи, т. е. новые строки в таблицу (**Insert**), 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 82. 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 оперирует преимущественно с четырьмя объектами:

- объект `Connection` обеспечивает соединение с базой данных;
- объект `Command` обеспечивает привязку SQL-выражения к соединению с базой данных;
- 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.

Запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Console Application Visual C#**. Зададим имя данного проекта: `БДDataReader`.

Мы собираемся вывести таблицу на экран простейшим способом. Если выводить ее при помощи функции `MessageBox.Show`, то ровных колонок в окне `MessageBox.Show` не получится, поскольку символы букв, 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 (на черный экран DOS), где реализован моноширинный шрифт, когда все символы (хоть буква "Ш", хоть простая точка ".") имеют одинаковую ширину. Следовательно, колонки в построенных таблицах будут ровными.

Теперь на вкладке программного кода напишем текст из листинга 10.2.

Листинг 10.2. 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;
```

```
// Другие директивы using удалены, поскольку они не используются в данной программе
// Добавляем эту директиву для краткости выражений:
using System.Data.OleDb;
namespace БДDataReader
{
    class Program
    {
        static void Main(string[] args)
        {
            // Задаем цвет текста на консоли для большей выразительности:
            Console.ForegroundColor = ConsoleColor.White;
            // Создаем объект класса Connection
            var Подключение = new OleDbConnection();
            // Передаем ему строку подключения:
            Подключение.ConnectionString =
                        "Data Source=\"D:\\vic.mdb\";User " +
                        "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";";
            Подключение.Open();
            // Создаем объект класса Command:
            var Команда = new OleDbCommand();
            Команда.Connection = Подключение;
            // Передаем ему SQL-команду:
            Команда.CommandText = "Select * From [БД телефонов]";
            // Выбрать все записи и сортировать их по колонке "ФИО":
            // Команда.CommandText =
            //                "Select * From [БД телефонов] order by ФИО";
            // Аналогично по колонке "Номер п/п":
            // Команда.CommandText =
            //        "Select * From [БД телефонов] ORDER BY [Номер п/п]";
            // Выполняем SQL-команду:
            var 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 — количество 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 из текста программы, вначале создаем объект `Подключение` класса `Connection` и передаем ему строку подключения. Далее создаем объект класса `Command` и передаем ему простейшую SQL-команду:

```
Select * From [БД телефонов]
```

Она означает 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. 10.7.

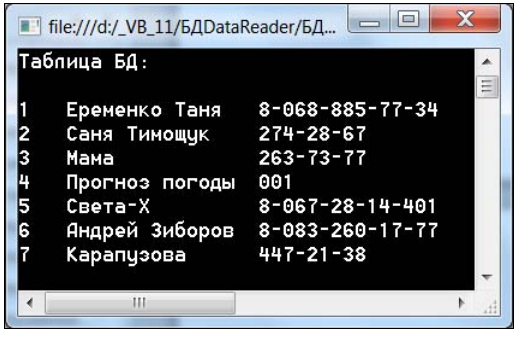


Рис. 10.7. Отображение таблицы базы данных на консоли

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 соответствующее решение в папке БДDataReader сопровождающего книгу электронного архива.

Пример 83. Создание базы данных MS Access в программном коде

Напишем программу, которая в процессе своей работы создает базу данных Access, пусть это будет файл new_BD.mdb. Полученная база данных окажется пустой, т. е. она не будет содержать ни одной таблицы. Наполнять базу данных таблицами можно впоследствии как из программного кода Visual Basic 12, так и с помощью MS Access. Заметим, что в этом примере технология ADO.NET *не использована*.

Запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Console Application**. Новую программу назовем `БДСоздание`. Для добавления в наш проект DLL-библиотеки

ADOX выполним следующие операции. В пункте меню **Project** выберем команду **Add Reference**, затем на вкладке **COM** двойным щелчком щелкнем по ссылке **Microsoft ADO Ext. 2.8 for DDL and Security** — тем самым мы добавим эту библиотеку в текущий проект. Убедиться в том, что теперь существует ссылка на эту библиотеку, можно в окне **Properties**. 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 10.3.

Листинг 10.3. 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 Visual C# 12, так и используя MS Access.
// В этом примере технология ADO.NET не использована
// ~ ~ ~ ~ ~ ~ ~ ~
// Добавим в наш проект библиотеку ADOX: Project | Add Reference, и на
// вкладке COM выбираем Microsoft ADO Ext. 2.8 for DDL and Security
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Для вызова MessageBox выберем следующие пункты меню:
// Project | Add Reference | Accemblies | Framework, затем в списке ссылок
// выберем ссылку System.Windows.Forms, а в тексте программы добавим
// директиву:
using System.Windows.Forms;
namespace БДСоздание
{
    class Program
    {
        static void Main(string[] args)
        {
            var Каталог = new ADOX.Catalog();
            try
            {
                Каталог.Create("Provider=Microsoft.Jet." +
                          "OLEDB.4.0;Data Source=D:\\new_BD.mdb");
                MessageBox.Show(
                    @"База данных D:\new_BD.mdb успешно создана");
            }
            catch (
                System.Runtime.InteropServices.COMException Ситуация)
            {
                MessageBox.Show(Ситуация.Message);
            }
            finally
            {
```

```
                Каталог = null;
            }
        }
    }
}
```

Чтобы объект `MessageBox` был доступен для вывода сообщений, добавим в проект еще одну DLL-библиотеку. Для этого, как и в предыдущем случае, выполним команду меню **Project | Properties | Accemblies | Framework**, затем двойным щелчком щелкнем на ссылке **System.Windows.Forms** (соответствует файлу 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. 10.8).

Если же тут же запустим наше приложение еще раз, то получим сообщение о том, что такая база данных уже существует (рис. 10.9), поскольку БД new_BD.mdb только что создана.

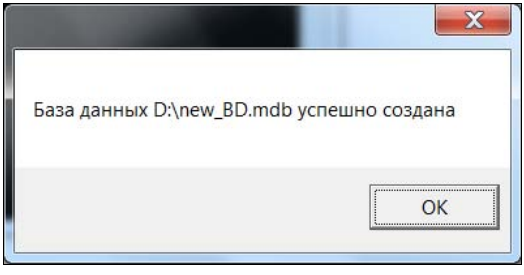


Рис. 10.8. Сообщение о создании базы данных

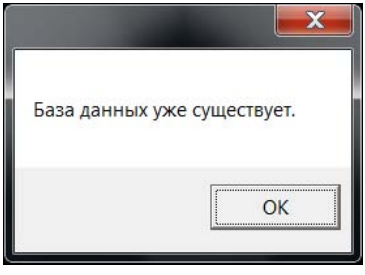


Рис. 10.9. База данных существует

Это сообщение генерировалось обработчиком исключительной ситуации. Чтобы во время выполнения программы *не появлялось* черное окно DOS, выполним следующее. В меню **Project** выберем пункт **БДСоздание Properties**, далее в новом окне, щелкнув указателем мыши на пункте **Application**, выберем в раскрывающемся меню **Application type** пункт **Windows Forms Application**. Теперь наша программа будет работать без вывода черного окна консоли.

Программный код этой программы можно посмотреть, открыв решение БДСоздание.sln в папке БДСоздание сопровождающего книгу электронного архива.

Пример 84. 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. ее "шапку", 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 D:, используя MS Access или программным путем, как это было показано в *примере 83*. Никаких таблиц в эту базу включать не станем, т. е. получим пустую БД. Теперь запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Console ApplicationVisual C#**. Новую программу назовем `БдСоздТаблицы`. Затем напишем программный код, представленный в листинге 10.4.

Листинг 10.4. 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;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Для вызова MessageBox выберем следующие пункты меню:
// Project | Add Reference | Accemblies | Framework, затем в списке ссылок
// выберем ссылку System.Windows.Forms, а в тексте программы добавим
// директиву:
using System.Windows.Forms;
// Добавляем эту директиву для более краткого обращения к классам
// обработки данных:
using System.Data.OleDb;
namespace БдСоздТаблицы
{
    class Program
    {
        static void Main(string[] args)
        {
            // ЗАПИСЬ СТРУКТУРЫ ТАБЛИЦЫ В ПУСТУЮ БД:
            // Создание экземпляра объекта Connection с указанием строки
            // подключения:
            var Подключение = new OleDbConnection(
            "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\\new_BD.mdb");
            // Открытие подключения:
            Подключение.Open();
```

```
            // Создание экземпляра объекта класса Command
            // с заданием SQL-запроса:
            var Команда = new 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 выполним команду меню **Project | Properties | Add Reference | Reference Manager | Assemblies | Framework** и в открывшемся списке установим флажок напротив ссылки **System.Windows.Forms**.

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. 10.10).

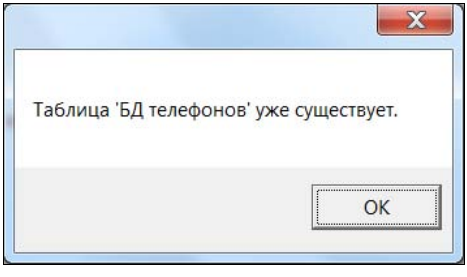


Рис. 10.10. Сообщение о существовании таблицы

Таким образом, в этой 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 мы 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 сопровождающего книгу электронного архива.

Пример 85. Добавление записей в таблицу базы данных MS Access

Совсем маленькую программу из *примера 84* можно использовать для выполнения *любого запроса* к базе данных. Так, модификация всего лишь одной строчки программного кода программы из листинга 10.4 может сделать ее пригодной *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.

> ***Примечание***
>
> Заметим, что в SQL-запросе мы сознательно обратились к таблице по имени `[бд телефонов]`, т. е. со строчной буквы, хотя надо бы 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 *примера 84*, где мы создавали новую таблицу в существующей БД.

Программный код будет выглядеть так, как представлено в листинге 10.5.

Листинг 10.5. 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;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Добавляем эту директиву для более краткого обращения к классам
// обработки данных:
using System.Data.OleDb;
// ~ ~ ~ ~ ~ ~ ~ ~
```

```
// Для вызова MessageBox выберем следующие пункты меню:
// Project | Add Reference | Accemblies | Framework, затем в списке ссылок
// выберем ссылку System.Windows.Forms, а в тексте программы добавим
// директиву:
using System.Windows.Forms;

namespace БдДобавлЗаписи
{
    class Program
    {
        static void Main(string[] args)
        {
            // ДОБАВЛЕНИЕ ЗАПИСИ В ТАБЛИЦУ БД:
            // Создание экземпляра объекта Connection
            // с указанием строки подключения:
            var Подключение = new OleDbConnection(
           "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\\new_BD.mdb");
            // Открытие подключения:
            Подключение.Open();
            // Создание экземпляра объекта Command с заданием SQL-запроса:
            var Команда = new OleDbCommand(
                "INSERT INTO [бд телефонов] (" +
                "Фио, [номер телефона]) VALUES ('Света-X', '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, при работе с БД появляется необходимость *проверки работы* программы — например, уточнить, 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 **Other Windows | Server Explorer** (комбинация клавиш <Ctrl>+<Alt>+<S>), 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 по нажатию 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 сопровождающего книгу электронного архива.

Пример 86. Чтение всех записей из таблицы базы данных с помощью объектов классов *Command*, *DataReader* и элемента управления *DataGridView*

Покажем, как легко, "очень малой кровью", вывести таблицу базы данных на элемент управления **DataGridView** (Сетка данных, в нашем случае — таблица данных) с использованием тех же объектов классов `Command` и `DataReader` (см. предыдущие программы).

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `БдReaderGridView`. 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 10.6.

Листинг 10.6. Чтение всех записей из таблицы БД

```
// Программа читает все записи из таблицы базы данных с помощью объектов
// Command, DataReader на элемент управления DataGridView (сетка данных)
using System;
using System.Data;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace БдReaderGridView
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Создаем объект Connection и передаем ему строку подключения:
            var Подключение = new System.Data.OleDb.OleDbConnection(
                        "Data Source=\"D:\\vic.mdb\";User " +
                        "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
            Подключение.Open();
            // Создаем объект Command, передавая ему SQL-команду
            var Команда = new System.Data.OleDb.
                OleDbCommand("Select * From [БД телефонов]", Подключение);
            // Выполняем SQL-команду
            var Читатель = Команда.ExecuteReader();
            // (CommandBehavior.CloseConnection)
            var 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(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 (см. листинг 10.5). 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. Пример работы программы показан на рис. 10.11.

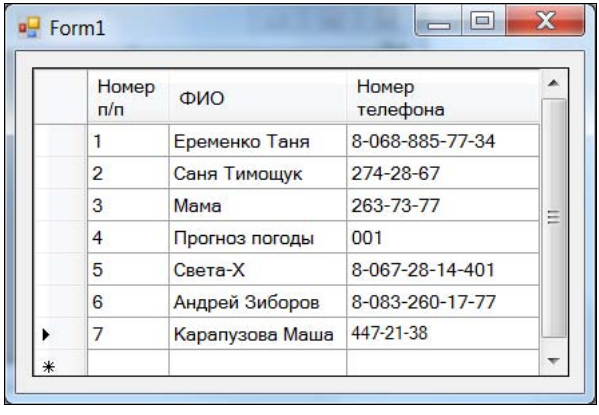


Рис. 10.11. Отображение таблицы базы данных на элементе **DataGridView**

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 сопровождающего книгу электронного архива.

Пример 87. 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, в нашем случае — 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, запустим Visual Studio 2012 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `БдАдаптерGridView`. Из панели **Toolbox** добавляем в форму элемент управления **DataGridView** и растягиваем его на всю форму, как показано на рис. 10.12.

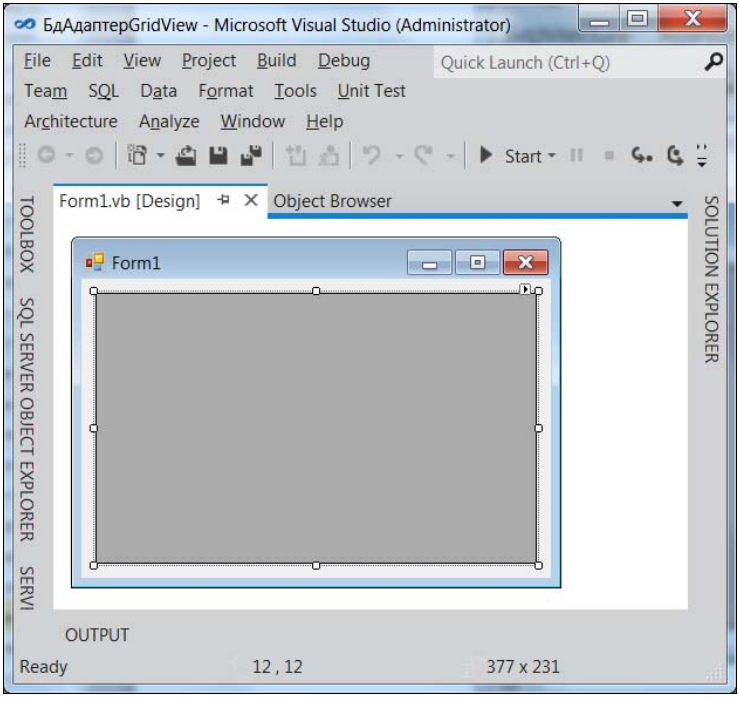


Рис. 10.12. Проектирование экранной формы

Далее пишем программный код, представленный в листинге 10.7.

Листинг 10.7. 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;
using System.Windows.Forms;
```

```
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace БдАдаптерGridView
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Чтение таблицы из БД:";
            var Подключение = new
                      System.Data.OleDb.OleDbConnection(
                     "Data Source=\"D:\\vic.mdb\";User " +
                     "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
            Подключение.Open();
            var Команда = new System.Data.OleDb.OleDbCommand(
                     "Select * From [БД телефонов]", Подключение);
            // Выбираем из таблицы только те записи, поле ФИО которых
            // начинается на букву "М":
            // var Команда =
            //    new System.Data.OleDb.OleDbCommand("SELECT * FRO" +
            //    "М [БД телефонов] WHERE (фио LIKE 'м%')", Подключение);
            // Создаем объект класса Adapter и выполняем SQL-запрос
            var Адаптер = new System.Data.OleDb.OleDbDataAdapter(Команда);
            // Создаем объект класса DataSet
            var НаборДанных = new System.Data.DataSet();
            // Заполняем DataSet результатом SQL-запроса
            Адаптер.Fill(НаборДанных, "БД телефонов");
            // Содержимое DataSet в виде строки XML для отладки:
            var 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 (метасимволами) "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 достаточным для вывода на экран результатов SQL-запроса (рис. 10.13).

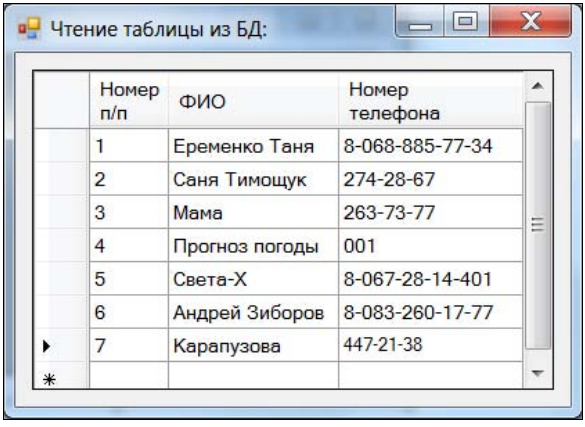


Рис. 10.13. Вывод результата SQL-запроса

Так же как и при использовании объекта класса `DataReader` в *примере 86*, 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 сопровождающего книгу электронного архива.

Пример 88. Обновление записей в таблице базы данных MS Access

Одним из основных четырех действий над данными в БД (напомним, это: `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 большим удобством (гибкостью) управления программным кодом.

Рассматриваемая в этом 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 табли-

це, после чего, нажав кнопку **Сохранить в БД**, данные в базе данных *будут модифицированы*, т. е. заменены новыми.

Для написания этой программы запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `БдUpdate`. Из панели **Toolbox** добавляем в форму элемент управления **DataGridView** и две командные кнопки. Программный код программы представлен в листинге 10.8.

Листинг 10.8. 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;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// А данную директиву добавим для краткости выражений:
using System.Data.OleDb;
namespace БдUpdate
{
    public partial class Form1 : Form
    {
        // ~ ~ ~ ~ ~ ~ ~ ~
        // Объявляем эти переменные вне всех процедур, чтобы
        // они были видны из любой из процедур:
        DataSet НаборДанных;
        OleDbDataAdapter Адаптер;
        OleDbConnection Подключение;
        OleDbCommand Команда;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            НаборДанных = new DataSet();
            Подключение = new OleDbConnection(
                "Data Source=\"D:\\vic.mdb\";User " +
                "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
            Команда = new OleDbCommand();
            button1.Text = "Читать из БД"; button1.TabIndex = 0;
            button2.Text = "Сохранить в БД";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Читать из БД:
            if (Подключение.State ==
                            ConnectionState.Closed) Подключение.Open();
```

```
            Адаптер = new OleDbDataAdapter(
                            "Select * From [БД телефонов]", Подключение);
            // Заполняем DataSet результатом SQL-запроса
            Адаптер.Fill(НаборДанных, "БД телефонов");
            // Содержимое DataSet в виде строки XML для отладки:
            var 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, 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",
                       OleDbType.VarWChar, 50, "Номер телефона");
            Команда.Parameters.Add(
                           "ФИО", OleDbType.VarWChar, 50, "ФИО");
            Команда.Parameters.Add(
                        new OleDbParameter("Original_Номер_п_п",
                         OleDbType.Integer,
                         0, 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
                var kol = Адаптер.Update(НаборДанных, "БД телефонов");
                MessageBox.Show(
                          "Обновлено " + kol.ToString() + " записей");
            }
            catch (Exception Ситуация)
            {
                MessageBox.Show(Ситуация.Message, "Недоразумение");
            }
        }
    }
}
```

Как видно текста программы, 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 двух 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`. Пример работы программы после чтения из базы данных представлен на рис. 10.14.

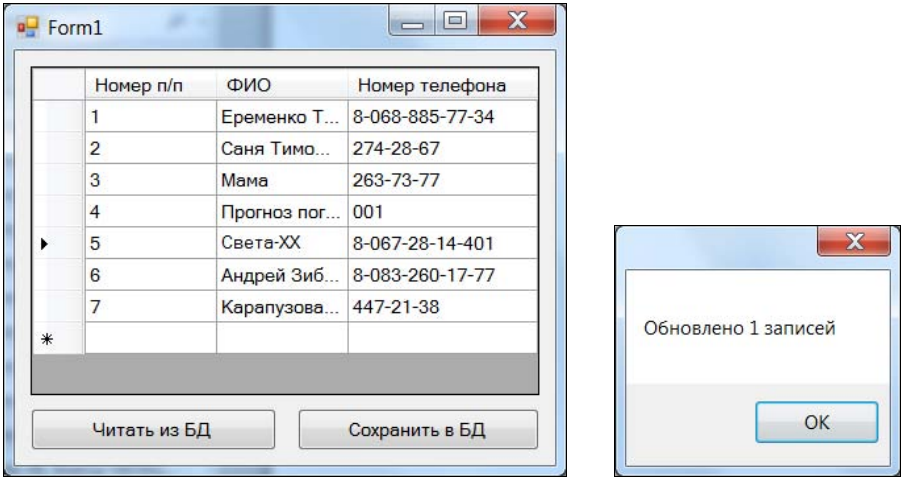


Рис. 10.14. Пример работы программы обновления данных

Представляет интерес программирование *модификации* записей базы данных. Эта возможность реализована 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 из текста программы, 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 `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 сопровождающего книгу электронного архива.

Пример 89. 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.

В данном случае экранная форма нам не нужна, поэтому выберем, как и в некоторых предыдущих примерах, шаблон консольного приложения. Запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Console Application Visual C#**. Новую программу назовем `БдУдаленЗаписи`. Чтобы иметь доступ к функции `MessageBox.Show`, добавим к проекту ссылку на динамическую библиотеку Forms.dll. Для этого щелкнем пункты соответствующие меню **Project | Add Reference | Reference Manager | Assemblies | Framework** и в появившемся списке установим флажок напротив ссылки **System.Windows.Forms**.

Далее напишем программный код из листинга 10.9.

Листинг 10.9. Удаление записей из таблицы БД

```
// Программа удаляет запись из таблицы БД с использованием SQL-запроса
// и объекта класса Command
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Для вызова MessageBox также укажем пункты меню:
// Project | Add Reference | Assemblies | Framework, щелкнем двойным щелчком
на ссылке
// System.Windows.Forms, а в тексте программы добавим директиву:
using System.Windows.Forms;
namespace БдУдаленЗаписи
{
    class Program
    {
        static void Main(string[] args)
        {
            // Создаем объект Connection и передаем ему строку подключения
            var Подключение = new System.Data.OleDb.
                      OleDbConnection(
                         "Data Source=\"D:\\vic.mdb\";User " +
                         "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
            Подключение.Open();
            // Создаем объект класса Command, передавая ему SQL-команду
            var Команда = new System.Data.OleDb.
                         OleDbCommand(
                        "Delete * From [БД телефонов] Where " +
                        "ФИО Like 'Vi%'", Подключение);
```

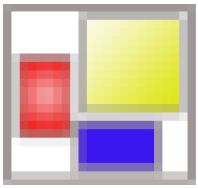
```
            // Выполнение команды SQL
            var i = Команда.ExecuteNonQuery();
            // i — количество удаленных записей
            if (i > 0) MessageBox.Show(
               "Записи, содержащие в поле ФИО фрагмент 'Vi*', удалены");
            if (i == 0) MessageBox.Show(
               "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, содержащих 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 сопровождающего книгу электронного архива.

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 этой главе рассмотрим типичные LINQ-запросы и преобразования к некоторым источникам данных.

Технология LINQ-запросов успешно работает в Visual Basic и C#, начиная с версии Visual Studio 2008. Что касается C++/CLI, то здесь LINQ-запросы пока не реализованы, можно лишь пользоваться методами класса `Linq.Enumerable`.

Пример 90. 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 этом примере рассмотрим две задачи: первая — из массива имен извлекаем имена длиной шесть символов, записывая их или в список (коллекцию) или в новый массив. Вторая задача — из массива целых чисел выбираем только те, значения которых больше четырех, также записывая результат запроса в список или массив.

Итак, запустим Visual Studio 12, закажем новый проект шаблона **Windows Forms Application Visual C#**. Новую программу назовем `LinqМассив`. Затем из панели элементов перенесем в форму текстовое поле. Далее щелчком правой кнопки мыши перейдем к вкладке программного кода (листинг 11.1).

Листинг 11.1. Извлечение данных из строкового массива

```
// 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 LinqМассив
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            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:
            var СтрокаИмен = "Витя Лариса Лариса Лена Андрей Женя " +
                             "Александр Лариса Виктор Света Оксана Наташа";
            // Из строки имен получаем массив имен, задавая в качестве
            // сепаратора деления подстрок символ пробела:
            var 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 Имя in Имена)
                textBox1.Text = textBox1.Text + Имя + " ";
            // В результате LINQ-запроса получаем список имен с количеством
            // букв равным шести:
            var Запрос = from s in Имена
                         where s.Length == 6
                         orderby s
                         select s.ToUpper(); // — перевод в верхний регистр
            // s — это переменная диапазона схожа с переменной итерации
            // в foreach;
            // where s.Length == 6 — условие выбора;
            // orderby s — сортировать в алфавитном порядке.
```

```
            // 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 s in Запрос)
                // s — переменная итерации в цикле foreach
                     textBox1.Text = textBox1.Text + s + " ";

            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 число in X)
                 textBox1.Text = textBox1.Text + число.ToString() + "  ";

            textBox1.Text = textBox1.Text + "\r\n\r\n" +
                "выбираем числа, значения " +
                "которых >= 4, и записываем их в список (коллекцию) Y, " +
                "исключая элементы-дубликаты:" + "\r\n\r\n";
            // Y — это список, куда помещаются выбранные элементы:
            var Y = from Число in X
                    where Число >= 4
                    orderby Число
                    select Число;
            // Удаляем элементы-дубликаты из списка целых чисел:
            var Z = Y.Distinct();
            // Или таким образом:
            // var Z = Y.Union(Y)
            // Вывод элементов списка Y в текстовое поле TextBox1:
            foreach (int числа in Z)
                textBox1.Text = textBox1.Text + числа.ToString() + " ";
        }
    }
}
```

Как видно из программного кода, после присвоения массиву `Имена` начальных значений создаем LINQ-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.

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 пример работы программы.

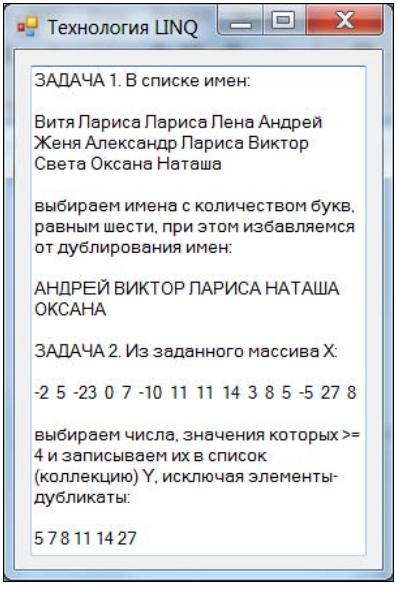


Рис. 11.1. Манипулирование массивом данных с помощью технологии LINQ

Убедиться в работоспособности программы можно, открыв решение LinqМассив.sln в папке LinqМассив сопровождающего книгу электронного архива.

Напоминание

Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 91. Запрос к коллекции (списку) данных методами LINQ

В некоторых случаях хранение данных в коллекции (например, 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 число не-

обходимых 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 12, закажем новый проект шаблона **Windows Forms Application Visual C#**. Назовем проект `LinqСписок`. Затем из панели элементов перенесем в форму текстовое поле. Далее щелчком правой кнопки мыши перейдем к вкладке программного кода (листинг 11.2).

Листинг 11.2. Извлечение данных из списков

```
// Решаем три различные 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 LinqСписок
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            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" };
            // Инициализировать список можно также таким образом:
            // 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:
            var n = Список.Count; // Количество элементов в списке
```

```
// Получение из списка его элемента с индесом 3 (как в массиве):
var A = Список.ElementAt(3);
var Ответ = Список.Remove("Лариса"); // — удаление из списка
// Преобразовать список в строковый массив:
var МассивСтрок = Список.ToArray();
foreach (String x in Список)
    textBox1.Text = textBox1.Text + x + " ";

textBox1.Text += "\r\n" +
    "выбираем имена длиной четыре символа:" + "\r\n";
// СписокВыбранныхИмен — это новый список, куда попадают
// выбранные строки в результате LINQ-запроса:
var СписокВыбранныхИмен = from Имя in Список
                          where Имя.Length == 4
                          orderby Имя
                          select Имя;
// Вывод списка выбранных имен в текстовое поле TextBox1:
foreach (String x in СписокВыбранныхИмен)
    textBox1.Text = textBox1.Text + x + " ";

// ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
// ЗАДАЧА 2.
textBox1.Text += "\r\n\r\n" +
    "ЗАДАЧА 2: Из списка сотрудников предприятия " +
              "выбираем некурящих для повышения зарплаты:" +
               "\r\n\r\n";
// Заполняем список сотрудников:
var 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}};
// LINQ-запрос:
var СписокНекурящихСотрудников =
                         from Сотрудник in Сотрудники
                         where Сотрудник.КуритЛи == false
                         orderby Сотрудник.Имя
                         select Сотрудник;
```

```
        // Select определяет, какие поля выводить в список запроса
        // Вывод списка некурящих сотрудников в текстовое поле textBox1:
        foreach (Сотрудник x in СписокНекурящихСотрудников)
            textBox1.Text = textBox1.Text + String.Format("{0} — " +
                            "возраст — {1}" + "\r\n", x.Имя, x.Возраст);

        // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
        // ЗАДАЧА 3.
        textBox1.Text += "\r\n" +
                         "ЗАДАЧА 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>();
        Студенты.Add(new Студент { Фамилия = "Зиборов",
                            Оценки = new List<int> { 5, 4, 4, 5 }});
        Студенты.Add(new Студент { Фамилия = "Стороженко",
                            Оценки = new List<int> { 3, 3, 2, 3 }});
        Студенты.Add(new Студент { Фамилия = "Ломачинская",
                            Оценки = new List<int> { 3, 4, 4, 5 }});
        Студенты.Add(new Студент { Фамилия = "Погребицкий",
                            Оценки = new List<int> { 2, 4, 3, 2 }});
        Студенты.Add(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 Студ in Студенты
                               from int Оценка in Студ.Оценки
                               where Оценка <= 2
                               orderby Студ.Фамилия
                               select new { Студ.Фамилия, Оценка };
        // Select определяет, какие поля выводить в список запроса
        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:
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:
    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 *примере 90*. 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 *примера 90*, отличие состоит лишь в применении синтаксиса манипуляций со списком типа `List`, а не с массивом. Здесь мы привели также некоторые важные способы манипуляции со 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, используя цикл `foreach`.

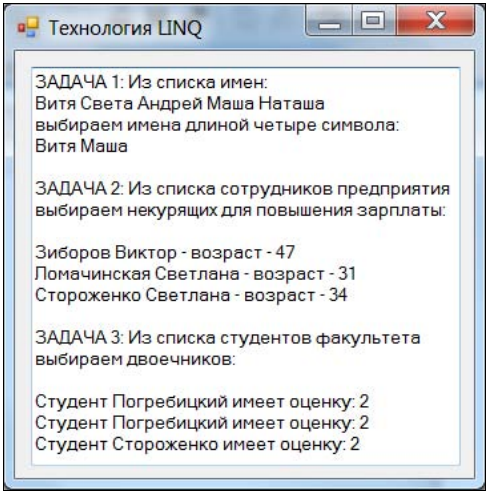


Рис. 11.2. Три LINQ-запроса к спискам данных

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 приводим 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).

> ***Примечание***
>
> При отладке LINQ-запросов следует иметь в виду, что часто транслятор указывает на "недоразумение" в одном месте, хотя на самом деле его нужно искать во всем запросе.

Убедиться в работоспособности программы можно, открыв решение LinqСписок.sln папки LinqСписок сопровождающего книгу электронного архива.

Пример 92. Группировка элементов списка с помощью LINQ-запроса

Приведем два варианта группировки данных, для этого используем стандартный шаблон LINQ-запроса для группировки элементов списка. В этих примерах будем группировать данные в списке типа `List`. Каждая запись в этом списке представляет собой два поля: название месяца в году и количество дней в этом месяце. В первом варианте задаем стандартный LINQ-запрос на группировку данных по количеству дней в месяце.

Для решения этой задачи запускаем Visual Studio 12, создаем новый проект, выбираем шаблон **Console Application Visual C#**, указываем имя **Name**: `LinqМесяцы1`. В листинге 11.3 приведен программный код этого приложения.

Листинг 11.3. Группировка элементов списка данных методом `group by` (первый вариант)

```
// Программа создает список месяцев в году с указанием их названия и
// количеством дней в месяце. Затем создает запрос на группировку этих
// данных по количеству дней. То есть запрос должен создать три группы
// месяцев, в которых содержится соответственно 28, 30 и 31 день.
// Результат этого запроса программа выводит на консоль.
using System;
using System.Collections.Generic;
using System.Linq;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace LinqМесяцы1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Задаем цвет текста на консоли для большей выразительности:
            Console.ForegroundColor = ConsoleColor.White;
```

```
            // Создаем список месяцев — их 12:
            var Месяцы = new List<Месяц>();
            // Инициализация списка:
            for (int i = 0; i <= 11; i++)
                Месяцы.Add(
                    new Месяц
                    {
                        Название = System.Globalization.
                                        CultureInfo.CurrentUICulture.
                                        DateTimeFormat.MonthNames[i],
                        Дней = DateTime.DaysInMonth(2012, i+1)
                    });
            // Запрос на группировку данных, записанных в список Месяцы:
            var Запрос = from Месяц Мес in Месяцы
                         group Мес by Мес.Дней
                         into ГруппыМесяцев
                         orderby ГруппыМесяцев.Key
                         select ГруппыМесяцев;
            // orderby — группы сортируем по количеству дней (Key)
            // select — результат запроса будет содержать количество дней
            // и перечень (группа) месяцев.
            // Выводим результаты запроса на консоль:
            // Цикл по группам:
            foreach (var Группа in Запрос)
            {
                Console.WriteLine("\r\n" +
                            "Месяцы, содержащие {0} дней:", Группа.Key);
                foreach (var M in Группа)
                            Console.WriteLine(M.Название);
            }
            // Ждем от пользователя нажатия какой-либо клавиши:
            Console.ReadKey();
        }
    }
    public class Месяц
    {
        public String Название;
        public int Дней;
    }
}
```

В программном коде вначале создаем список типа `List`. В круглых скобках указываем, что каждая запись в списке будет представлять собой экземпляр (объект) класса `Месяц`, состоящую из двух полей строкового и целого типов. После инициализации списка организуем запрос на получение трех производных групп методом `group by`. Результат запроса попадает в переменную `Запрос`. Для вывода групп на консоль используем два вложенных цикла `foreach`. Внешний цикл выполняется по группам, а второй выводит непосредственно название каждого месяца на консоль. Результат работы программы представлен на рис. 11.3.

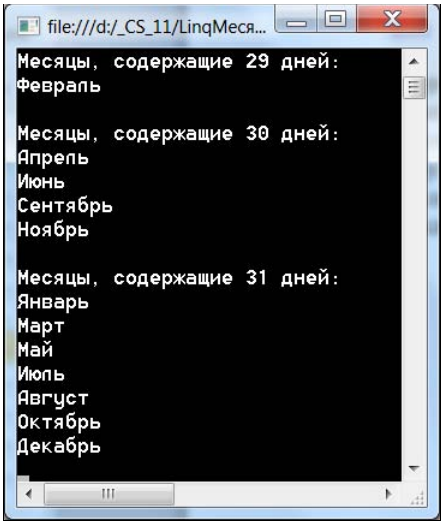


Рис. 11.3. Группировка списка по количеству дней в месяцах (первый вариант)

Во втором варианте оперируем с тем же списком месяцев, но здесь требуется получить ровно два производных списка: в первый список будут входить месяцы, количество дней в которых равно 31, а во второй — остальные месяцы.

Так же, как и в предыдущем варианте, запускаем Visual Studio 12, создаем новый проект, выбираем шаблон **Console Application Visual C#**, указываем имя **Name**: `LinqМесяцы2`. В листинге 11.4 приведен программный код этого приложения.

Листинг 11.4. Группировка элементов списка данных методом `group by` (второй вариант)

```
// Программа создает список месяцев в году с указанием их названия и
// количеством дней в месяце. Затем создает запрос на группировку этих
// данных, т. е. следует создать одну группу месяцев, в которых
// содержится 31 день, и другую группу, в которую входят прочие месяцы.
// Результат этого запроса программа выводит на консоль
using System;
using System.Collections.Generic;
using System.Linq;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace LinqМесяцы2
{
    class Program
    {
        static void Main(string[] args)
        {
            // Задаем цвет текста на консоли для большей выразительности:
            Console.ForegroundColor = ConsoleColor.White;
            // Создаем список месяцев — их 12:
            var Месяцы = new List<Месяц>();
```

```
            // Инициализация списка:
            for (var i = 0; i <= 11; i++)
                Месяцы.Add(
                    new Месяц
                    {
                        Название = System.Globalization.
                                        CultureInfo.CurrentUICulture.
                                        DateTimeFormat.MonthNames[i],
                        Дней = DateTime.DaysInMonth(2012, i + 1)
                    });
            // Запрос на группировку данных, записанных в список Месяцы.
            // Этот список делим на две групппы, одна группа включает в себя
            // месяцы, содержащие 31 день, в вторая — прочие месяцы:
            var Запрос = from Месяц Мес in Месяцы
                         group Мес by new { Критерий = (Мес.Дней == 31) }
                         into ГруппыМесяцев
                         select ГруппыМесяцев;
            // Этот запрос можно прочитать так: список месяцев разделить на
            // две группы. В одной группе содержатся месяцы, количество дней
            // в которых равно 31 дню, а во второй группе — прочие месяцы
            // Здесь критерием (True или False) является равенство дней 31.
            // В запрос выводим группу месяцев и критерий.
            // Выводим результаты запроса на консоль:
            Console.WriteLine("Две группы месяцев: ");
            foreach (var Гр in Запрос)
                if (Гр.Key.Критерий == true)
                {
                    Console.WriteLine(
                        "\r\n" + "Месяцы, содержащие 31 день: " + "\r\n");
                    foreach (var M in Гр)
                        Console.WriteLine("{0} — {1}", M.Название, M.Дней);
                }
                else
                {
                    Console.WriteLine(
                           "\r\n" + "Прочие месяцы: " + "\r\n");
                    foreach (var M in Гр)
                        Console.WriteLine("{0} — {1}", M.Название, M.Дней);
                }
            Console.ReadKey();
        }
    }
    public class Месяц
    {
        public String Название;
        public int Дней;
    }
}
```

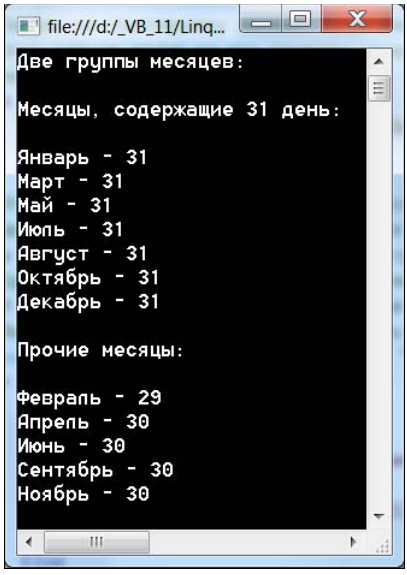


Рис. 11.4. Группировка списка по количеству дней в месяцах (второй вариант)

В комментариях к программе мы сделали подробный анализ соответствующих операторов Linq-запроса. Результат работы программы представлен на рис. 11.4.

Убедиться в работоспособности программы можно, открыв соответствующие решения в папках LinqМесяцы1 и LinqМесяцы2 сопровождающего книгу электронного архива.

Пример 93. Группировка словаря данных *Dictionary* с помощью LINQ-запроса

В задаче, решаемой в этом примере, осуществляется манипуляция со словарем данных типа `Dictionary`. В словарь данных будем записывать информацию о некоторых товарах. В реальной ситуации данные о каждом товаре займут большое количество полей, т. е. колонок в соответствующих таблицах. В нашем примере каждый товар будет иметь только название и цену. Чтобы ориентироваться в большом множестве товаров, это множество разбивают на группы. В нашей задаче мы товары, записанные в словарь данных, разделим на две ценовые группы: соответственно, дороже и дешевле 90 рублей. Это деление выполним, используя LINQ-технологию.

Как и в предыдущем случае, запускаем Visual Studio 12, создаем новый проект, выбираем шаблон **Console Application Visual C#**. Укажем имя решения **Name**: `LinqЦеныНаПродукты`. В листинге 11.5 приведен программный код этого приложения.

Листинг 11.5. Группировка элементов словаря данных `Dictionary` методом `group by`

```
// Программа создает словарь данных Dictionary продуктов питания. В словаре -
// всего два поля: наименование товара и его цена. Методом group by из данных
// словаря создаем две ценовые группы: товары дороже и дешевле 90 рублей.
// Обе группы товаров выводим на консоль
using System;
```

```
using System.Collections.Generic;
using System.Linq;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace LinqЦеныНаПродукты
{
    class Program
    {
        static void Main(string[] args)
        {
            // Задаем цвет текста на консоли для большей выразительности:
            Console.ForegroundColor = ConsoleColor.White;
            // Создаем и заполняем словарь данных:
            var Продукты = new Dictionary<String, Single>();
            Продукты.Add("Творог", 115.5F);
            Продукты.Add("Хлеб", 18.75F);
            Продукты.Add("Печенье", 93.75F);
            Продукты.Add("Чай", 76.25F);
            Продукты.Add("Мясо", 150.0F);
            Продукты.Add("Гречка", 62.5F);
            // Запрос на группировку данных, записанных в словарь Продукты:
            var Запрос = from Продукт in Продукты
                         // group Продукт new
                         group Продукт by Продукт.Value > 90 into Группа
                         select new {
                             Группа, Группа.Key,
                             СрЦенаПоГруппе = Группа.
                             Average(Продукт => Продукт.Value) };

            // Этот запрос можно прочитать так: словарь продуктов разделить
            // на две ценовые группы по критерию больше 90 руб.
            // Для отладки:
            var k0 = Запрос.ElementAt(0);
            var k1 = Запрос.ElementAt(1);
            // Выводим результаты запроса на консоль:
            Console.WriteLine("Две ценовые группы: ");
                    foreach (var Гр in Запрос)
                    {
                        if (Гр.Key == true)
                        {
                            Console.WriteLine(
                                "\r\n" + "Товары дороже 90 руб: " + "\r\n");
                            foreach (var Товар in Гр.Группа)
                                Console.WriteLine("{0} — {1}",
                                    Товар.Key, Товар.Value);

                            Console.WriteLine("Средняя цена по группе = {0}",
                                                                Гр.СрЦенаПоГруппе);
                        }
```

```
                        else
                        {
                            Console.WriteLine(
                                "\r\n" + "Товары дешевле 90 руб: " + "\r\n");
                            foreach (var Товар in Гр.Группа)
                                Console.WriteLine("{0} — {1}",
                                                    Товар.Key, Товар.Value);

                            Console.WriteLine(
                                "Средняя цена по группе = {0}",
                                                    Гр.СрЦенаПоГруппе);
                        }
                    }
            Console.ReadKey();
        }
    }
}
```

В начале программы формируем словарь продуктов питания. Далее организуем запрос на группировку данных на две группы методом `group by`. Вывод результата запроса организуем с помощью двух вложенных циклов `foreach`. Внешний цикл выполняется по ценовым группам, а внутренний — по товарам.

Пример работы программы показан на рис. 11.5.

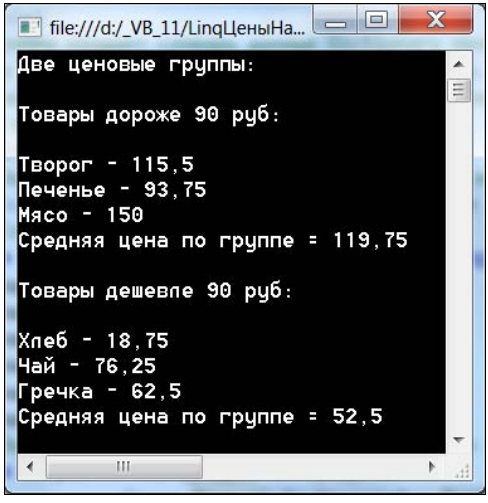


Рис. 11.5. Группировка словаря данных методом `group by`

Убедиться в работоспособности программы, рассмотренной в этом примере, можно, открыв соответствующее решение в папке LinqЦеныНаПродукты сопровождающего книгу электронного архива.

Пример 94. Создание XML-документа методами классов пространства имен *System.Xml.Linq*

Кроме пространства имен `System.Xml`, содержащего классы для обработки XML-документов, в Visual Studio 12 имеем пространство имен `System.Xml.Linq`, содержащее классы, которые позволяют легко и эффективно изменять документы XML, а также организовывать поиск данных. В этом примере представим сведения о наших повседневных телефонных контактах в виде XML-документа, используя класс `XDocument` пространства `System.Xml.Linq`. Эти данные будут 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, запускаем Visual Studio 12, создаем новый проект, выбираем шаблон **Console Application Visual C#**, укажем имя **Name**: `LinqСоздатьXML-документ`. В листинге 11.6 приведен программный код обсуждаемого консольного приложения.

Листинг 11.6. 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"),
// — имя элемента и его зачение (Value)
new XElement("Мобильный_телефон", "")),

new XComment(
    "Поскольку каждый элемент Контакт имеет атрибут и два вложенных=>"),
new XElement("СТРОКА",
// Имя атрибута — Контакт
new XAttribute("Контакт", "Борис Григорьевич"),
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",
// Значение атрибута — Света
new XAttribute("Контакт", "Света"),
new XElement("Домашний_телефон", ""),
new XElement("Мобильный_телефон", "+7(495)615-24-41")))
                                                        );
            // Сохранить XML-документ:
            XMLдокумент.Save(@"D:\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. 11.6).

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 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"`.

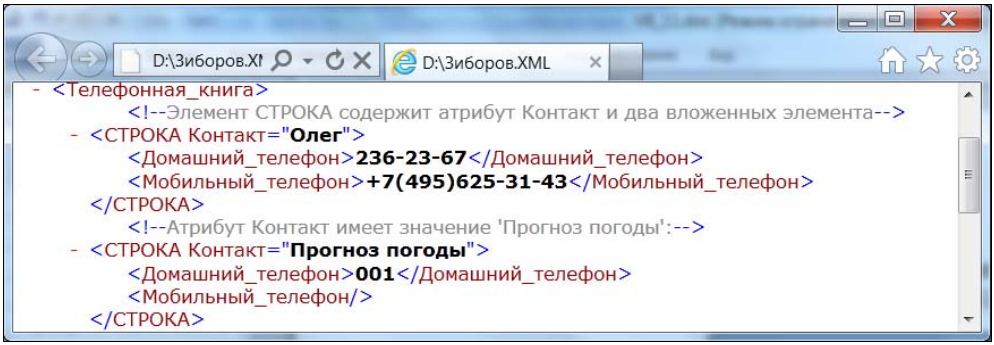


Рис. 11.6. XML-файл, открытый в Internet Explorer

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 программы будет выведено на консоль содержимое XML-документа без XML-объявления (скриншот консоли мы не приводим), а также будет создан XML-файл. Открыв этот файл с помощью MS Excel, получим таблицу телефонных контактов (рис. 11.7).

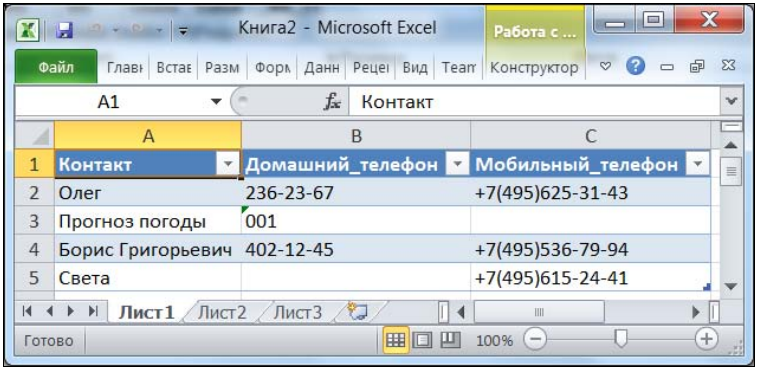


Рис. 11.7. XML-файл, открытый в MS Excel

Убедиться в работоспособности программы можно, открыв решение LinqСоздатьXML-документ.sln папки LinqСоздатьXML-документ сопровождающего книгу электронного архива.

Пример 95. Извлечение значения элемента из XML-документа посредством LINQ-запроса

В этом примере, решая задачу, мы получили строку XML-данных, например, от удаленной веб-службы, обеспечивающей прогнозом погоды. В такой текстовой XML-строке содержатся метеорологические показатели на заданный нами район для текущей даты. В рассматриваемой задаче мы *извлекаем из этих XML-данных* только значение температуры.

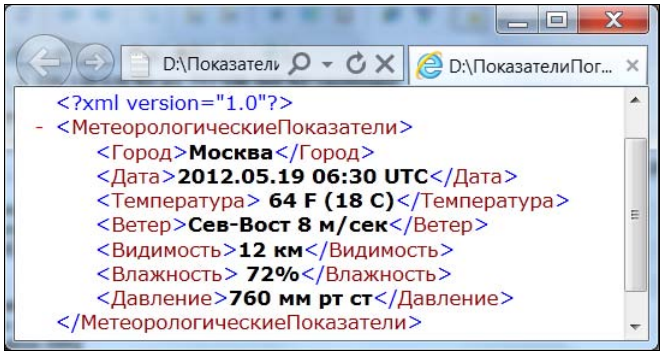


Рис. 11.8. Содержимое XML-файла с метеорологическими показателями

Несколько слов о структуре XML-документа (рис. 11.8).

Как видно из иллюстрации, 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 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 12 в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**, укажем имя решения **Name**: `ПоискXmlЭлемента1`.

Далее, попав в конструктор формы, из панели элементов **Toolbox** перетащим в нее 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 несколько строчек, в свойствах объекта `textBox1` укажем `true` напротив свойства `Multiline`. Затем на вкладке программного кода введем текст, представленный в листинге 11.7.

Листинг 11.7. Извлечение значения элемента из XML-данных (вариант 1)

```
// 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;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// Добавим эту директиву:
using System.Xml.Linq;
namespace ПоискXmlЭлемента1
```

```
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Поиск значения элемента в XML-документе";
            textBox1.Multiline = true;
            textBox1.Font = new Font("Consolas", 9.0F);
            var СтрокаXML =
                "<?xml version=\"1.0\"?>                        " + "\r\n" +
                "   <МетеорологическиеПоказатели>               " + "\r\n" +
                "      <Город>Москва</Город>                    " + "\r\n" +
                "      <Дата>2012.05.19 06:30 UTC</Дата>        " + "\r\n" +
                "      <Температура> 64 F (18 C)</Температура> " + "\r\n" +
                "      <Ветер>Сев-Вост 8 м/сек</Ветер>          " + "\r\n" +
                "      <Видимость>12 км</Видимость>             " + "\r\n" +
                "      <Влажность> 72%</Влажность>              " + "\r\n" +
                "      <Давление>760 мм рт ст</Давление>        " + "\r\n" +
                "   </МетеорологическиеПоказатели>";
            // Загрузка корневого элемента из строки, содержащей XML:
            var КорневойЭлемент = XElement.Parse(СтрокаXML);
            // Или корневой элемент XML-документа получаем через файл:
            // Записываем строку, содержащую XML в файл:
            // System.IO.File.WriteAllText(
            //        @"D:\ПоказателиПогоды.xml", СтрокаXML);
            // Загружаем корневой элемент XML:
            // var КорневойЭлемент =
            //             XElement.Load(@"D:\ПоказателиПогоды.xml");
            // Из корневого элемента извлекаем вложенный в него элемент
            // "Температура" и получаем соответствующее значение (Value)
            // этого элемента:
            var 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.

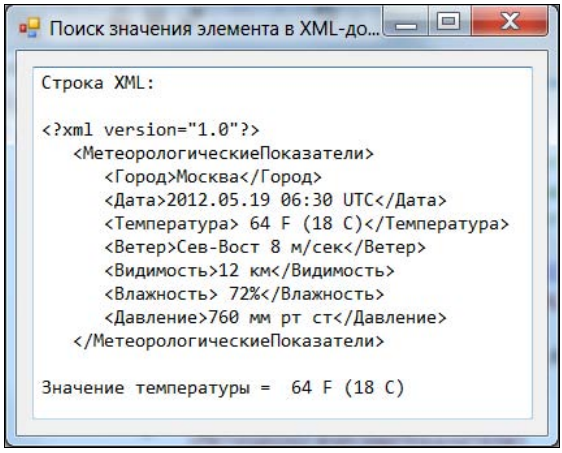


Рис. 11.9. Извлечение значения элемента из XML-документа

Пример работы программы приведен на рис. 11.9.

Убедиться в работоспособности программы можно, открыв решение ПоискXmlЭлемента1.sln папки ПоискXmlЭлемента1 сопровождающего книгу электронного архива.

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 (листинг 11.8) — ситуация из реальной жизни. 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. 11.10.

Теперь для получения значения температуры удобно воспользоваться типовым LINQ-запросом. Для этого запускаем Visual Studio 12, создаем новый проект, выбираем шаблон **Console Application Visual C#**. Даем имя решению **Name**: `ПоискXmlЭлемента2`. Далее на вкладке программного кода введем текст, представленный в листинге 11.8.

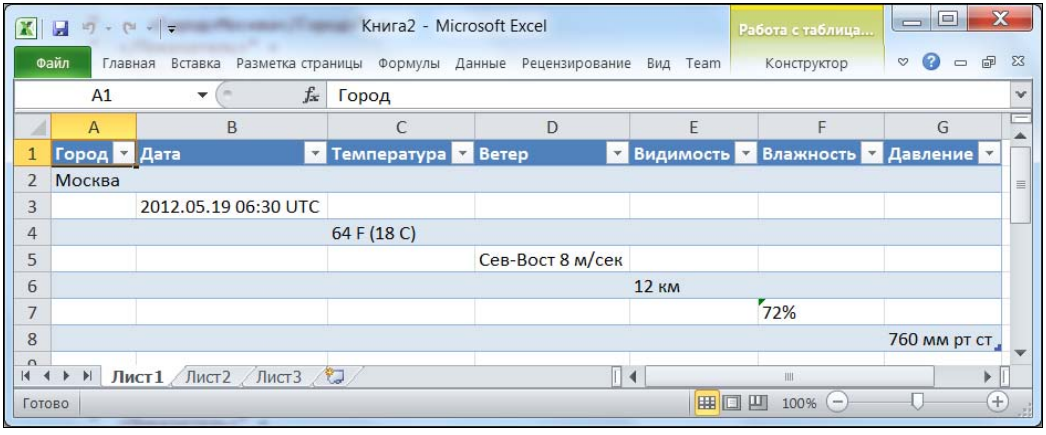


Рис. 11.10. Представление XML-данных в виде таблицы в MS Excel

Листинг 11.8. Извлечение значения элемента из XML-данных (вариант 2)

```
// 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;
using System.Linq;
// Другие директивы using удалены, поскольку они не используются в данной
// программе.
// Добавим эту директиву:
using System.Xml.Linq;
namespace ПоискXmlЭлемента2
{
    class Program
    {
        static void Main(string[] args)
        {
            // Задаем цвет текста на консоли для большей выразительности:
            Console.ForegroundColor = ConsoleColor.White;
            Console.Title = "LINQ-запрос к XML-данным";
            // Инициализация XML-строки:
            var СтрокаXML =
                    "<?xml version=\"1.0\"?>" +
                    "<МетеорологическиеПоказатели>" +
                    "  <Показатель>" +
                    "     <Город>Москва</Город>" +
                    "  </Показатель>" +
                    "  <Показатель>" +
                    "     <Дата>2012.05.19 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 = XElement.Parse(СтрокаXML);
            // Запрос — это коллекция (список) строк, куда 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 — ровно одна, однако все
            // равно ее следует извлекать из коллекции посредством цикла:
            foreach (var x in Запрос)
            {
                var xx = x.Trim();
                Console.WriteLine(
                        "\r\n" + "Значение температуры = {0}", xx);
            }
            Console.ReadKey();
        }
    }
}
```

Как видно из программного кода, вначале загружаем строку с XML-данными в XML-элемент методом `Parse`. Затем получаем коллекцию дочерних элементов с именем `Показатель`, используя метод `Elements` класса `XElement`. Теперь делаем запрос методом `Where` на выборку из этой коллекции элементов с именем `Температура`. Таких элементов в производном списке — ровно один. Далее в комментарии показано, как можно найти в данном XML-документе значение температуры без использования метода `Where`.

Пример работы программы показан на рис. 11.11.

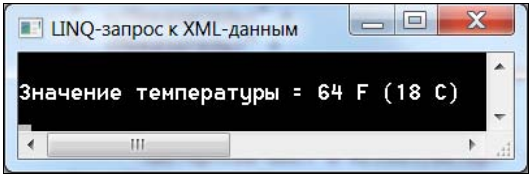


Рис. 11.11. Извлечение значения температуры из XML-документа

Убедиться в работоспособности программ, рассмотренных в этом примере, можно, открыв соответствующие решения в папках ПоискXmlЭлемента1 и ПоискXmlЭлемента2 сопровождающего книгу электронного архива.

Пример 96. Поиск строк (записей) в XML-данных с помощью LINQ-запроса

Пусть у нас есть XML-данные, в которых содержатся традиционная для нашей книги таблица с именами и телефонами, причем имена в этой телефонной таблице повторяются, — например, строка с именем "Витя" содержит мобильный телефон, а потом, по мере знакомства с этим Витей, у нас появился еще и его домашний телефон. Задача состоит в том, чтобы в данной таблице телефонов (представленной в виде XML,

см. рис. 11.6) найти все строчки с именем "Витя". Эта маленькая и несерьезная, на первый взгляд, задача моделирует вполне серьезную проблему. Имея громадную базу данных, которую мы получили на каком-то этапе их обработки в виде XML, нам требуется "отфильтровать" записи в этой базе на предмет нахождения в таком-то поле такой-то строки.

Прежде чем решать задачу, давайте посмотрим отображение обсуждаемых XML-данных в табличном редакторе MS Excel (рис. 11.12).

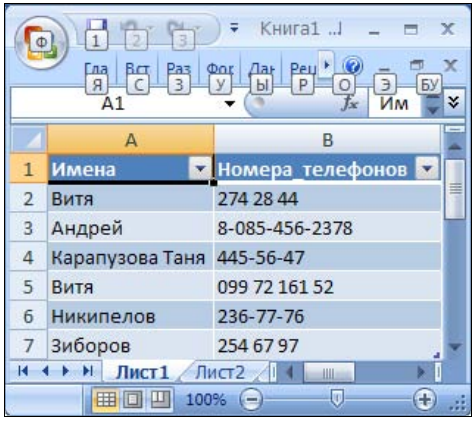


Рис. 11.12. Отображение XML-данных в MS Excel

Как видно из иллюстрации, в редакторе MS Excel наши XML-данные представлены очень наглядно. И вполне понятно, что мы хотим получить, — все номера телефонов абонента с именем "Витя".

Для решения этой задачи запустим Visual Studio 12 и выберем шаблон **Windows Forms Application Visual C#**, укажем имя **Name**: `ПоискСтрокВXml`. Далее, попав в конструктор формы, из панели элементов **Toolbox** перетащим в нее 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 11.9.

Листинг 11.9. Извлечение значения элемента из XML-данных

```
// Имеем XML-данные, в которых содержатся 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;
using System.Linq;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе.
// Добавим эту директиву:
using System.Xml.Linq;
namespace ПоискСтрокВXml
```

```
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "LINQ-запрос к XML-данным";
            textBox1.Multiline = true;
            // Инициализация XML-строки:
            var 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 = XElement.Parse(СтрокаXML);
            // Запись строки, содержащей XML в файл:
            // System.IO.File.WriteAllText(
            //                 "ТаблицаТелефонов.xml", СтрокаXML);
            // КорневойЭлемент = XElement.Load("ТаблицаТелефонов.xml");
            var Запрос =
                    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 Запрос)
                textBox1.Text = textBox1.Text + x + "\r\n";
            // Таких записей в этой коллекции — ровно одна
        }
    }
}
```

Как видно из текста программы, 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 показано, как можно получить корневой элемент в том случае, если он представлен в виде XML-файла во внешней памяти. Затем организовываем типовой стандартный LINQ-запрос. Результат запроса попадает в коллекцию записей, которую выводим в текстовое поле, используя оператор цикла. Пример работы программы показан на рис. 11.13.

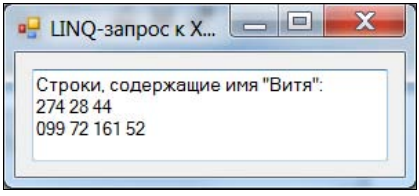


Рис. 11.13. LINQ-запрос к XML-документу

Убедиться в работоспособности программы можно, открыв решение ПоискСтрокВXml.sln из папки ПоискСтрокВXml сопровождающего книгу электронного архива.

Пример 97. Получение производных XML-данных от XML-источника

Пусть у нас есть источник данных в виде XML-файла, содержащего сведения о некоторых книгах, подготовленных в издательстве "БХВ-Петербург". Поскольку мы должны представить очень маленький и выразительный пример, то в этом файле содержатся сведения всего о пяти книгах. Фрагмент содержимого этого файла-источника в Internet Explorer приведен на рис. 11.14, а его отображение в виде таблицы в MS Excel — на рис. 11.15. Из этого XML-файла требуется выбрать книги только одного автора — Виктора Зиборова — и записать их в другой, производный XML-файл.

Для решения этой задачи запустим Visual Studio 12, выберем шаблон **Console Application Visual C#**, укажем имя **Name**: `LinqBhv`. В листинге 11.10 приведен программный код этого консольного приложения.

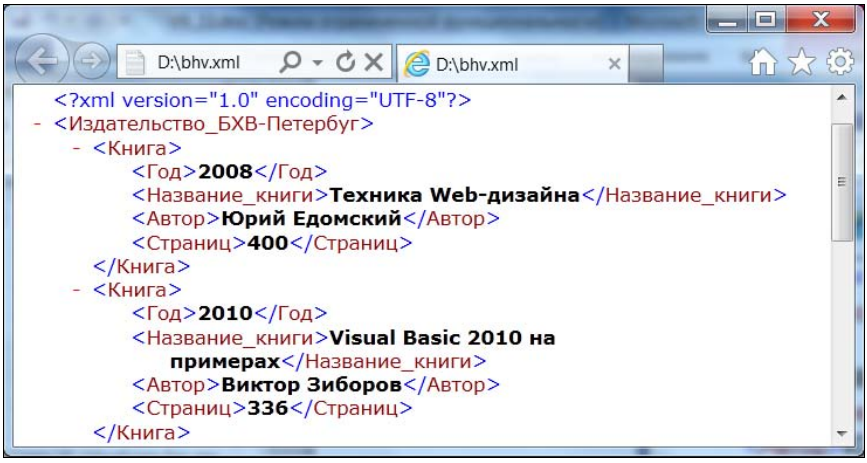


Рис. 11.14. Фрагмент XML-файла — источника данных

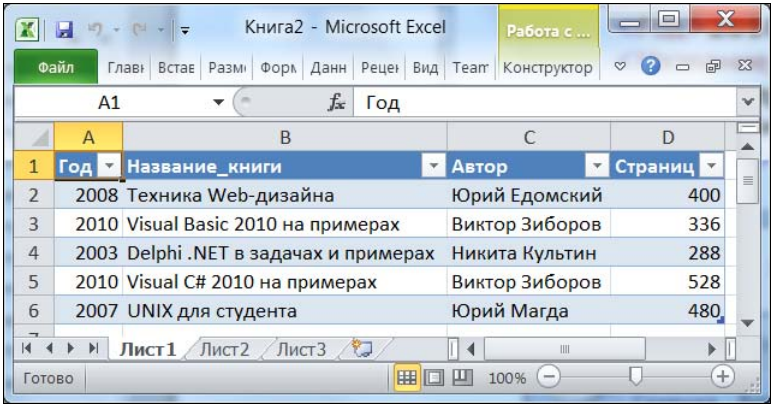


Рис. 11.15. Отображение исходного XML-файла в MS Excel

Листинг 11.10. Выборка данных из XML-файла и их запись в другой XML-файл

```
// Программа читает XML-файл (источник), содержащий сведения о книгах
// по программированию, и выбирает книги автора Зиборова. Выбранные
// книги программа записывает в другой (производный) XML-файл
using System;
using System.Linq;
// Другие директивы using удалены, поскольку они не используются в данной
// программе.
// Добавим эту директиву:
using System.Xml.Linq;
namespace LinqBhv
{
    class Program
    {
        static void Main(string[] args)
```

```
        {
            var Установки = new System.Xml.XmlWriterSettings();
            // Создавать отступы для элементов:
            Установки.Indent = true;
            // Создание производного XML-файла:
            var XmlПисатель = System.Xml.
                XmlWriter.Create(@"D:\ТолькоЗиборов.xml", Установки);
            // Добавляем начальный (корневой) элемент:
            XmlПисатель.WriteStartElement("Книги_Виктора_Зиборова");
            // Комментарий:
            XmlПисатель.WriteComment("Эти данные мы выбрали из bhv.xml");
            // Загружаем XML-документ — источник:
            var XmlДокумент = XDocument.Load(@"D:\bhv.xml");
            // Извлекаем корневой элемент:
            var КорневЭлемент =
                    XmlДокумент.Element("Издательство_БХВ-Петербуг");
            // Получаем список элементов "Книга":
            var СписокКниг = КорневЭлемент.Elements("Книга");
            // В этом списке анализируем дочерний элемент "Автор":
            foreach (var Элемент in СписокКниг)
                // Добавляем в производный XML-файл только книги Зиборова:
                if (Элемент.Element("Автор").Value == "Виктор Зиборов")
                    Элемент.WriteTo(XmlПисатель);
            // Добавляем конечный тег корневого элемента производного
            // Xml-документа:
            XmlПисатель.WriteEndElement();
            XmlПисатель.Flush();
            XmlПисатель.Close();
            System.Diagnostics.Process.Start(
                                "IExplore.exe", @"D:\ТолькоЗиборов.xml");
        }
    }
}
```

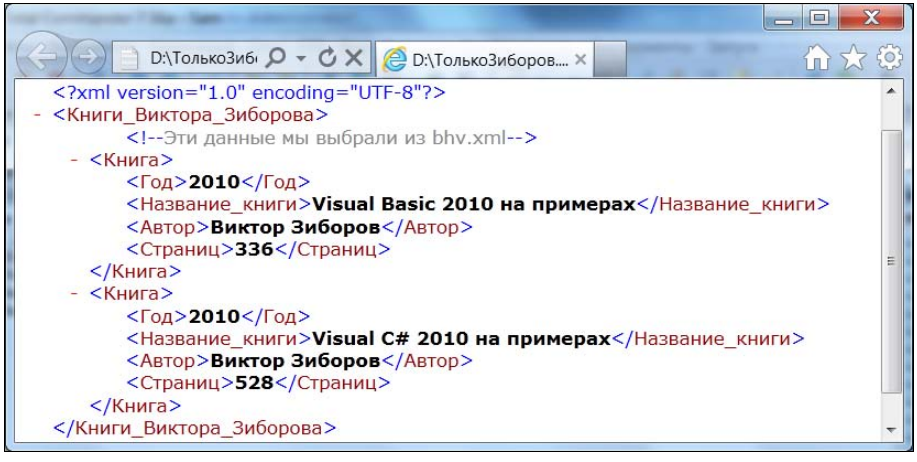


Рис. 11.16. Содержимое производного XML-файла

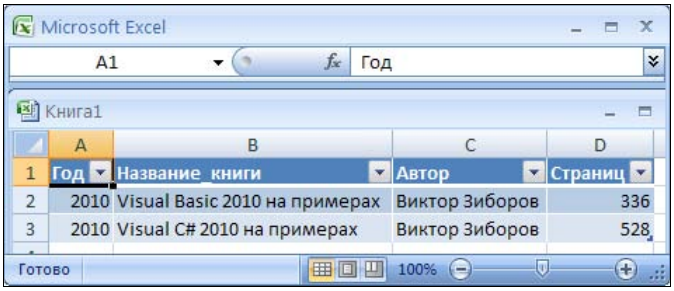


Рис. 11.17. Отображение выбранных XML-данных в MS Excel

В программном коде создаем производный XML-файл, куда будем писать выбранные сведения из файла-источника D:\bhv.xml. Из исходного XML-документа (источника) извлекаем корневой элемент, а из него получаем список элементов "Книга". В этом списке с помощью цикла `foreach` организуем поиск дочернего элемента "Автор" по содержимому элемента "`Виктор Зиборов`". В результате работы программы получим производный XML-файл с выбранными данными (рис. 11.16 и 11.17).

Убедиться в работоспособности программы можно, открыв решение LinqBhv.sln из папки LinqBhv сопровождающего книгу электронного архива.

Пример 98. 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 собой расположенный в памяти кэш данных (cache, 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 несколько объектов (таблиц) `DataTable`. Запросы LINQ к таблицам данных, кэшированным в объекте `DataSet`, упрощают и ускоряют процесс отбора.

Наша 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 — мы будем производить отбор городов, численность населения которых превышает миллион жителей.

Для решения этой задачи запустим Visual Studio 12 и выберем шаблон **Windows Forms Application Visual C#**, укажем имя **Name**: `LinqГорода`. Далее, попав в конструктор формы, из панели элементов **Toolbox** перетащим в нее 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 11.11.

Листинг 11.11. 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, то 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.Linq;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace LinqГорода
{
    public partial class Form1 : Form
    {
        // Создание объекта Таблица данных
        DataTable Таблица = new DataTable();
        // Создание объекта Набор данных
        DataSet НаборДанных = new DataSet();
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "LINQ-запрос к набору данных DataSet";
            button1.Text = "Сохранить"; button2.Text = "Найти";
            textBox1.Multiline = true;
            if (System.IO.File.Exists(@"D:\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(@"D:\Города.xml");
                // Содержимое DataSet в виде строки XML для отладки:
                // var 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(@"D:\Города.xml");
        }
        private void button2_Click(object sender, EventArgs e)
        {
            // Щелчок мышью на кнопке "Поиск" — запрос городов-миллионеров:
            textBox1.Clear(); // — очистка текстового поля
            // Linq-запрос:
            // Поскольку к таблице класса DataTable невозможно задавать
            // Linq-запросы, преобразуем ее к коллекции строк в таблице:
            var ГородаМлн =
                from Строка in Таблица.AsEnumerable()
                where Convert.ToInt32(
                Строка.Field<String>("Население")) >= 1000000
                select new
                {
                    A = Строка.Field<String>("Город"),
                    B = Строка.Field<String>("Население")
                };
            // select задает поля запроса, здесь заданы два поля.
            // ~ ~ ~ ~ ~ ~ ~ ~
            // Вывод результата запроса в текстовое поле:
            textBox1.Text = "Города-миллионеры: " + "\r\n";
            foreach (var Строка in ГородаМлн)
                textBox1.Text +=
                         Строка.A + " — " + Строка.B + "\r\n";
        }
    }
}
```

В начале программы создаем объекты классов `DataSet` и `DataTable` так, чтобы они были видимыми из всех процедур класса `Form1`. Далее при обработке события загрузки формы проверяем, существует ли файл `Города.xml`, куда мы записываем искомую таблицу. Если файла 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`.

Пример работы программы показан на рис. 11.18.

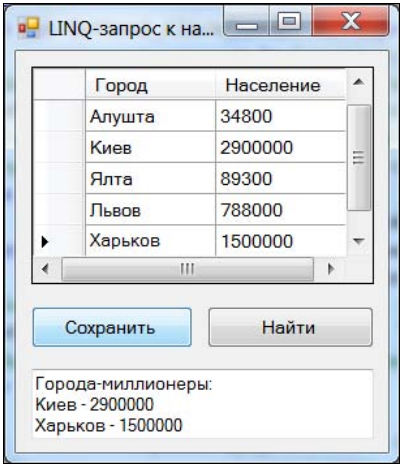
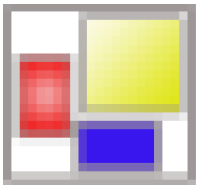


Рис. 11.18. LINQ-запрос к набору данных

Убедиться в работоспособности программы можно, открыв решение LinqГорода.sln папки LinqГорода сопровождающего книгу электронного архива.

ГЛАВА 12



Другие задачи, решаемые с помощью *Windows Application*

Пример 99. 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 правильному написанию 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 слово шаблону русской фамилии.

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `ПроверкаФамилии`. 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;
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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            label1.Text = "Введите фамилию на русском языке:";
            button1.Text = "Проверка";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            textBox1.Text = textBox1.Text.Trim();
            if (System.Text.RegularExpressions.Regex.Match(
                textBox1.Text,
                "^[А-ИК-ЩЭ-Я][а-яА-Я]*$").Success == false)
                MessageBox.Show("Неверный ввод фамилии", "Ошибка");
        }
    }
}
```

При обработке события "щелчок мыши на кнопке" текстовое поле 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. Пример работы программы представлен на рис. 12.1.

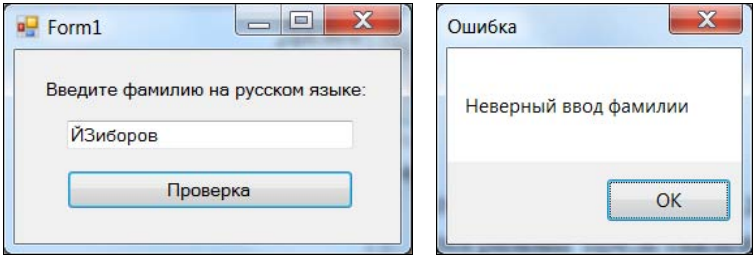


Рис. 12.1. Проверка корректности ввода фамилии на русском языке

Вторая задача, которую мы рассмотрим здесь, — 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).

Вторую программу этого примера назовем `ПроверкаЧисла`. Пользовательский интерфейс для решения 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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            label1.Text = "Введите положительное" +
                          "\r\n" + "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");
        }
    }
}
```

Поскольку эта программа построена аналогично предыдущей, в комментарии нуждается только шаблон:

```
"^(([0-9]+.[0-9]*)|([0-9]*.[0-9]+)|([0-9]+))$"
```

Здесь между символами `|`, означающими логическое ИЛИ, в круглых скобках 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 соответствующие решения в папках ПроверкаФамилии и ПроверкаЧисла сопровождающего книгу электронного архива.

> ***Напоминание***
>
> Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 100. 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, следующий щелчок процесс возобновляет и т. д.

Для написания этой программы запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем Opacity. Теперь добавим в стандартную форму из панели элементов управления **Toolbox** объект **Timer** (Таймер). Это невидимый во время работы программы элемент управления предназначен для периодического генерирования события Tick, которое происходит, когда таймер работает, и заданный интервал времени прошел. Этот интервал времени Timer1.Interval по умолчанию равен 100 миллисекундам (т. е. десятой доле секунды), его в нашей программе мы изменять не будем.

В программе мы будем обрабатывать три события. Пустой обработчик события загрузки формы можем получить, щелкнув двойным щелчком в пределах проектируемой формы. Обработчик события истечения заданного временного интервала можно получить, щелкнув двойным щелчком на изображении объекта **Timer** на вкладке дизайнера формы **Form1.cs[Design]**. И пустой обработчик события "щелчок в пределах экранной формы" можно получить в окне свойств формы, щелкнув на значке молнии и выбрав событие **Click**. 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; // — шаг изменения прозрачности
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            s = 0.1;
            this.Text = "Щелкните на форме";
            // timer1.Interval() = 400;
        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            // Если непрозрачность выходит за рамки интервала
            // 0 < Opacity < 1, то меняем знак шага на противоположный:
            if (this.Opacity <= 0 || this.Opacity >= 1) s = -s;
            // Каждую десятую долю секунды изменяем уровень прозрачности
            // формы на s = 0.1:
            this.Opacity = this.Opacity + s;
        }
        private void Form1_Click(object sender, EventArgs e)
        {
            // Включаем или выключаем таймер:
            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 возникает 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 лишь немного (косметически) его изменил.

Текст программы можно посмотреть, открыв решение Opacity.sln в папке Opacity сопровождающего книгу электронного архива.

Пример 101. 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 новости валютного рынка, или фондового рынка Форекс, или другая экономическая информация, 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. Но это не есть самое эффективное занятие для ума.

Кроме того, следует помнить, что время по Гринвичу не переводится на летние/зимние часы. Поэтому, чтобы выяснить правильное время по Гринвичу, можно на стандартном рабочем столе Windows справа внизу щелкнуть двойным щелчком на отображении текущего местного времени, в открывшемся окне 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 центрально-европейское время, Central European Time (CET)) и время по Гринвичу.

Предлагаю написать маленькую программу, которая в полупрозрачной экранной форме отображает текущее время по Гринвичу и при этом, в силу своей прозрачности, не закрывает другие приложения.

Для решения этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `Гринвич`. Теперь из панели элементов управления **Toolbox** перетащим в проектируемую экранную форму 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.

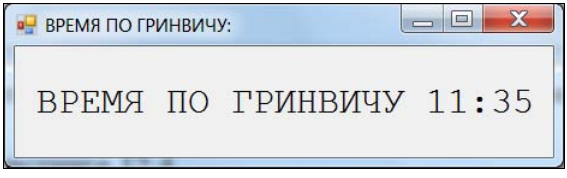


Рис. 12.2. Фрагмент работы программы определения времени по Гринвичу

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 и при этом не закрывает 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
    {
        // Булева переменная t каждую секунду меняет свое значение на
        // противоположное:
        Boolean t = false;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "ВРЕМЯ ПО ГРИНВИЧУ:";
            this.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 ";
            t = !t; // То же, что и t = true ^ t;
            String Время;
            if (t == true)
                Время = String.Format("{0:t}", DateTime.UtcNow);
            else // Через одну секунду выводим GMT-время без ":"
                Время = String.Format("{0} {1:00}", DateTime.UtcNow.Hour,
                                                    DateTime.UtcNow.Minute);
            label1.Text = label1.Text + Время;
        }
        private void label1_MouseEnter(object sender, EventArgs e)
        {
            // Указатель мыши входит в область метки
            this.Opacity = 1;
        }
        private void Form1_MouseLeave(object sender, EventArgs e)
        {
            // Указатель мыши выходит за пределы формы
            this.Opacity = 0.75;
        }
    }
}
```

Здесь при обработке события загрузки формы задаем текст "ВРЕМЯ ПО ГРИНВИЧУ:" в заголовке формы, указываем уровень непрозрачности формы `this.Opacity = 0.75`.

> ***Пояснение***
>
> При `Opacity = 0` изображение формы совсем пропадает, программа остается благополучно работать в оперативной памяти, а мы узнаем о ее работе только потому, что она будет напоминать о своем существовании в свернутом виде в панели задач. При `Opacity = 1` мы увидим 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 12 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 (каждую секунду) меняет свое значение с `true` на `false`, потом с `false` на `true` и т. д., т. е. на противоположное: `t = Not t`. Таким образом, при обработке события `Tick` (события, когда прошел заданный интервал времени `Interval`, равный 1000 миллисекунд = 1 секунда) объекта `timer1` в текстовую метку копируется новое значение времени то с двоеточием, то без него.

Как можно видеть, в тексте программы использованы переменные с русскими именами, что *добавило выразительности* программному коду. Мы придали форме способность становиться 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 сопровождающего книгу электронного архива.

Пример 102. Ссылка на процесс, работающий в фоновом режиме, в форме значка в области уведомлений

Название примера для новичка покажется чрезвычайно "заумным". На самом деле здесь нет ничего сложного, как и во многом другом в области компьютерных технологий, если разобраться в сути слов и попробовать практически испытать работу элементов этих технологий. Здесь речь идет о программах, работающих на компьютере *в фоновом режиме*, — это, например, антивирусные программы или элементы управления громкостью. Такие программы визуально 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 *примере 101*, 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 здесь программа также способна работать в фоновом режиме. Внешне она 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 остается и продолжает работать 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 этой 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 этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Новую программу назовем `Значок_в_области_уведомлений`. В конструкторе формы из панели элементов перенесем в форму текстовую 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
```

```
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Создаю значок в области уведомлений";
            // Задаем размер метки
            label1.Size = new Size(364, 107);
            // 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
            this.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; this.Show();
        }
        private void времяСтартаОСToolStripMenuItem_Click(
                                  object sender, EventArgs e)
        {
            var Время_работы_ОС_в_минутах =
                Environment.TickCount / 1000 / 60;
```

```
            // Формат "{0:F0}" округляет Double до целого значения:
            var 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(); // Синтаксис C++: delete NotifyIcon1
            Application.Exit();
        }
    }
}
```

Как видно из программного кода, при обработке события загрузки формы 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 примеры работы программы в обычном и фоновом режимах.

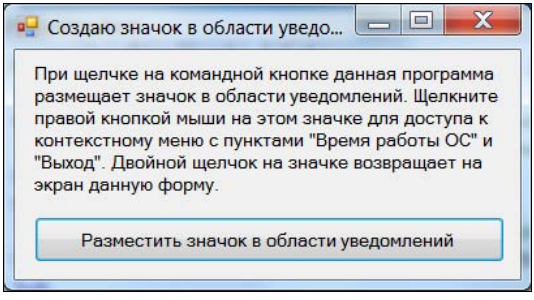


Рис. 12.3. Внешний вид формы при работе программы в обычном режиме

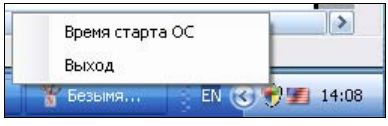


Рис. 12.4. Контекстное меню при работе программы в фоновом режиме

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке Значок_в_области_уведомлений сопровождающего книгу электронного архива.

Пример 103. 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 ра-

боты 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 приведен в книге "Visual Basic .NET. Библия пользователя"[1].

Итак, запускаем Visual Studio 12 и в окне **New Project** выбираем шаблон **Windows Forms Application Visual C#**. Новую программу назовем ПеремещениеФормы. Нам не нужны никакие элементы управления, будем оперировать только со свойствами (атрибутами) экранной формы. 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 за любое место формы
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ПеремещениеФормы
{
    public partial class Form1 : Form
    {
        Boolean Перемещение;
        // Перемещаем форму только тогда, когда Перемещение == true
        int MouseDownX;
        int MouseDownY;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            Перемещение = false;
        }
        private void Form1_MouseDown(object sender, MouseEventArgs e)
        {
            // Здесь обрабатываем событие "щелчок любой кнопкой мыши".
            // Во время щелчка левой кнопкой мыши запоминаем текущее
            // положение мыши
            if (e.Button == MouseButtons.Left)
            {
                Перемещение = 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)
                Перемещение = false;
        }
        private void Form1_MouseMove(object sender, MouseEventArgs e)
        {
            // Здесь обрабатываем событие, когда указатель
            // мыши перемещается в пределах формы.
            // Перемещаем форму только тогда, когда Перемещение = true
            if (Перемещение == true)
            {
                var Точка = new System.Drawing.Point();
                Точка.X = this.Location.X + (e.X — MouseDownX);
                Точка.Y = this.Location.Y + (e.Y — MouseDownY);
                this.Location = Точка;
            }
        }
    }
}
```

В начале программы объявлены три внешние переменные, видимые в пределах класса `Form1`. Одна из переменных — булева переменная `Перемещение` — это как бы флажок, который сигнализирует, следует ли перемещать экранную форму. Если пользователь нажал левую кнопку мыши (событие `MouseDown`, `e.Button = Left`), то `Перемещение = 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 `Перемещение = true`.

Убедиться в работоспособности программы можно, открыв решение ПеремещениеФормы.sln в папке ПеремещениеФормы сопровождающего книгу электронного архива.

Пример 104. Воспроизведение звуков операционной системы

Вы знаете, читатель, что операционная система Windows имеет в своем арсенале множество звуков, которые она использует, "обыгрывая" то или иное событие. В этом примере, покажем, как мы, разработчики программного обеспечения, можем использовать эти звуки в своих собственных программах. Для демонстрации некоторых возможностей напишем программу, содержащую несколько командных кнопок на экранной форме, при нажатии на каждую из которых будем извлекать тот или иной звук.

Запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Программу назовем `Звуки_ОС`. Из панели элементов перенесем в форму шесть командных кнопок. Получив, соответственно, шесть пустых обработчиков событий, введем текст программы, приведенный в листинге 12.7.

Листинг 12.7. Воспроизведение звуков

```
// Программа воспроизводит некоторые звуки операционной системы Windows
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace Звуки_ОС
{
    public partial class Form1 : Form
    {
        System.Media.SoundPlayer Плеер;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Звуковые сигналы ОС";
            Плеер = new System.Media.SoundPlayer();
            // Задаем названия командных кнопок:
            button1.Text = "Звук торжества \"та-да\"";
            button2.Text = "Звук завершения процесса";
            button3.Text = "Звук ошибки";
            button4.Text = "Частота звука = 1000 гц";
            button5.Text = "Загрузка Windows";
            button6.Text = "Выход из Windows";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Воспроизведение звукового WAV-файла
            Плеер.SoundLocation = @"c:\windows\media\tada.wav";
            Плеер.Play();
        }
        private void button2_Click(object sender, EventArgs e)
        {
            // Класс SystemSounds получает звуки, связанные с набором
            // типов звуковых событий операционной системы Windows:
            System.Media.SystemSounds.Asterisk.Play();
        }
        private void button3_Click(object sender, EventArgs e)
        {
            System.Media.SystemSounds.Beep.Play();
        }
```

```
        private void button4_Click(object sender, EventArgs e)
        {
            // Звуковой сигнал частотой 1000 Гц и длительностью 0.5 секунд:
            Console.Beep(1000, 500);
        }
        private void button5_Click(object sender, EventArgs e)
        {
            Плеер.SoundLocation =
                            @"c:\windows\media\Windows Logon Sound.wav";
            Плеер.Play();
        }
        private void button6_Click(object sender, EventArgs e)
        {
            Плеер.SoundLocation =
                            @"c:\windows\media\Windows Shutdown.wav";
            Плеер.Play();
        }
    }
}
```

Как можно видеть, программный код довольно простой (или "прозрачный", как говорят программисты). Заметим только, что многие звуки находятся в системной папке C:\Windows\Media, и вы, читатель, можете найти здесь что-либо подходящее для ваших целей.

Внешний вид пользовательского интерфейса программы паказан на рис. 12.5.

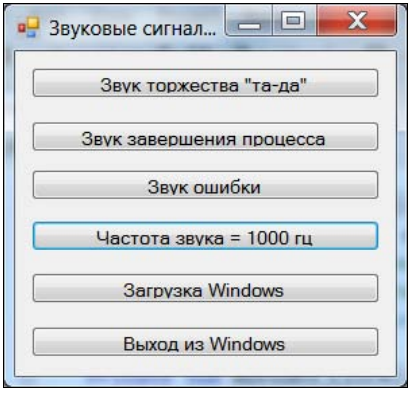


Рис. 12.5. Выбор некоторых звуков операционной системы Windows

Убедиться в работоспособности программы можно, открыв решение Звуки_ОС.sln в папке Звуки_ОС сопровождающего книгу электронного архива.

Пример 105. Проигрыватель Windows Media Player 12

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, один из самых современных плееров — Windows Media Player версии 12. 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 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Назовем этот новый проект `Player`. Теперь нам нужно добавить на проектируемую форму элемент управления **Windows Media Player**. Однако в панели **Toolbox** нет такого элемента. 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** (Выбрать элементы). Далее на вкладке **COM Components** установим флажок напротив ссылки **Windows Media Player** и щелкнем на кнопке **OK**. При этом на панели элементов появится значок **Windows Media Player**. Перетащим этот значок в форму и растянем изображения формы и плеера так, как вам это представляется наиболее привлекательным. После этого в папке проекта obj\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. 12.6.

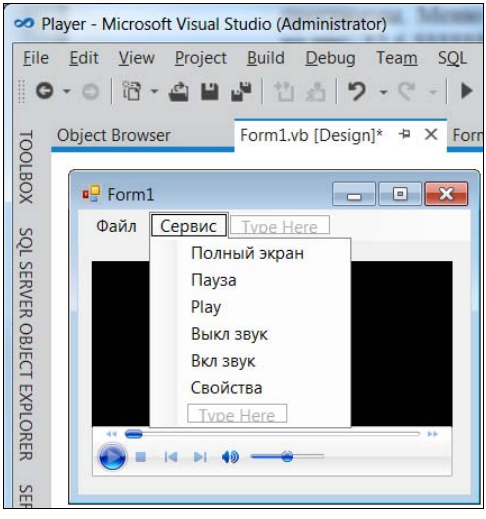


Рис. 12.6. Проектирование пунктов меню

Программный код приведен в листинге 12.8.

Листинг 12.8. Проигрыватель Windows Media Player 12

```
// Программа реализует функции проигрывателя Windows Media Player 12
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;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            openFileDialog1 = new OpenFileDialog();
            // ВЕРСИЯ ПЛЕЕРА
            this.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 28 (см. главу 4)*, а можно его создать программно, как в данном программном коде.

При обработке события загрузки формы 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 только, если плеер

пребывает в состоянии проигрывания файла `wmppsPlaying`), `Пауза`, `Play`, `ВклЗвук`, `ВыклЗвук`, вызов меню **Свойства**, выход из программы.

Пример работы программы представлен на рис. 12.7.

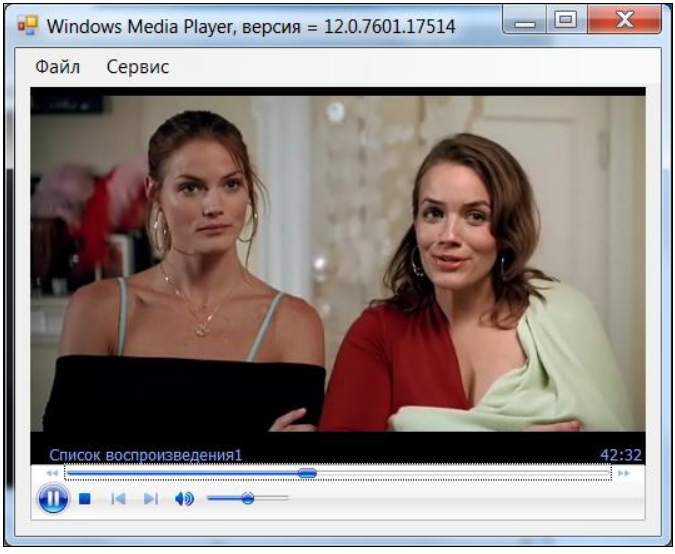


Рис. 12.7. Проигрыватель в режиме воспроизведения

Убедиться в работоспособности программы можно, открыв решение Player.sln в папке Player сопровождающего книгу электронного архива.

Пример 106. Воспроизведение только звуковых файлов

Представим ситуацию, когда нам надо воспроизводить только звуковые файлы — типа mp3, wma, mid и проч. Например, вы хотите слушать музыку в ходе работы вашей программы в фоновом режиме. Следует учесть, что такой сервис не должен быть навязчивым, и поэтому необходимо предусмотреть возможность выключения музыкального сопровождения.

Для программирования этой задачи запустим Visual Studio 12 и в окне **New Project** выберем шаблон **Windows Forms Application Visual C#**. Назовем этот проект `PlayerТолькоЗвук`. Теперь из панели элементов управления **Toolbox** перетащим в проектируемую экранную форму три командные кнопки **Button**. Первая кнопка нам нужна для выбора звукового файла, который будет подлежать воспроизведению, а остальные две — для регулировки громкости.

Далее наступает очень важный момент — следует добавить ссылку на нужную объектную библиотеку. Для этого выбираем пункты меню **Project | Add Reference** (Добавить ссылку) и, щелкнув на кнопке **Browse** (Обзор), найдем файл C:\Windows\system32\wmp.dll. Заметьте, что в этой программе мы не добавляли Windows Media Player в графический интерфейс формы. Несмотря на то, что в этой программе и в предыдущей, из

примера 105, мы добавили ссылки на разные библиотеки, на самом деле мы используем один и тот же Windows Media Player, причем одной и той же версии. Теперь перейдем на вкладку программного кода и введем текст, представленный в листинге 12.9.

Листинг 12.9. Воспроизведение звуковых файлов

```
// Программа предназначена для воспроизведения только звуковых файлов
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Добавляем ссылку на объектную библиотеку. Для этого выберем
// пункты меню Project | Add Reference и, щелкнув на кнопке
// Browse (Обзор), найдем файл C:\Windows\system32\wmp.dll
namespace PlayerТолькоЗвук
{
    public partial class Form1 : Form
    {
        WMPLib.WindowsMediaPlayer Плеер;
        OpenFileDialog ОткрытьФайл;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            Плеер = new WMPLib.WindowsMediaPlayer();
            ОткрытьФайл = new OpenFileDialog();
            // ВЕРСИЯ ПЛЕЕРА
            this.Text = "Windows Media Player " + Плеер.versionInfo;
            button1.Text = "Файл";
            button2.Text = "Увеличить громкость";
            button3.Text = "Уменьшить громкость";
            // Задаем путь к файлу:
            // Плеер.URL = @"D:\VenSalvrme.wma"
            Плеер.URL = @"C:\WINDOWS\Media\town.mid";
            // Плеер.URL = @"D:\juliana-best_of_best.mp3";
            Плеер.settings.volume = 10;
            // Команда на проигрывание файла
            Плеер.controls.play();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Кнопка Файл:
            // Пользователь выбирает файл:
            ОткрытьФайл.ShowDialog();
            // Передача плееру имени файла
            Плеер.URL = ОткрытьФайл.FileName;
```

```
            // Команда на проигрывание файла
            Плеер.controls.play();
        }
        private void button2_Click(object sender, EventArgs e)
        {
            // Увеличиваем громкость с каждым щелчком:
            Плеер.settings.volume = Плеер.settings.volume + 10;
        }
        private void button3_Click(object sender, EventArgs e)
        {
            // Уменьшаем громкость с каждым щелчком:
            Плеер.settings.volume = Плеер.settings.volume — 10;
        }
    }
}
```

В программном коде при обработке события загрузки формы создаем новый объект класса `WindowsMediaPlayer`. Задаем путь к звуковому файлу в свойстве `URL`. Причем если нам известен адрес этого файла в сети Интернет, то мы можем указать его в этом свойстве. При обработке события "щелчок мышью на первой кнопке" организуем диалог на открытие файла, и после выбора файла пользователем подаем команду на проигрывание выбранного файла.

Внешний вид графического интерфейса программы представлен на рис. 12.8.

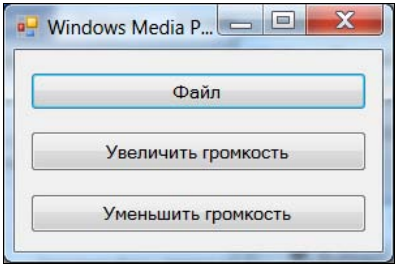


Рис. 12.8. Проигрыватель в режиме воспроизведения

Убедиться в работоспособности программы можно, открыв решение PlayerТолькоЗвук.sln в папке PlayerТолькоЗвук сопровождающего книгу электронного архива.

Пример 107. Программирование контекстной справки. Стандартные кнопки в форме

Напишем программу, демонстрирующую организацию помощи пользователю вашей программы. В этой 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 Visual C#**, получим стандартную форму. Назовем программу `Help`. Используя панель **Toolbox**, добавим в форму три текстовых поля, три метки и две кнопки, как показано на рис. 12.9.

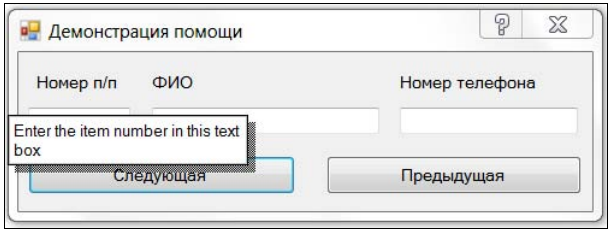


Рис. 12.9. Пример работы программы с контекстной справкой

Таким образом, мы получили пользовательский интерфейс редактирования таблицы телефонов, знакомый нам из предыдущих примеров.

На вкладке программного кода напишем следующий текст (листинг 12.10).

Листинг 12.10. 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;
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();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            MaximizeBox = false; // — отмена кнопки Развернуть
            MinimizeBox = false; // — отмена кнопки Свернуть
            // Чтобы стиль формы содержал кнопку помощи,
            // т. е. вопросительный знак
            this.HelpButton = true;
            this.Text = "Демонстрация помощи";
            // Me.Font = New Font("Courier New", 10.0F);
            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 Хелп = new HelpProvider();
            Хелп.SetHelpString(this.textBox1,
                "Enter the item number in this text box");
            // "Здесь отображаются номера записи по порядку")
            Хелп.SetHelpString(this.textBox2,
                "Поле для редактирования имени абонента");
            Хелп.SetHelpString(textBox3,
                "Поле для ввода и отображения номера телефона");
            Хелп.SetHelpString(button1,
                "Кнопка для перехода на следующую запись");
            Хелп.SetHelpString(button2,
                "Кнопка для перехода на предыдущую запись");
            // Назначаем, какой help-файл будет вызываться
            // при нажатии клавиши <F1>
            Хелп.HelpNamespace = @"C:\Windows\System32\appverif.chm";
            // "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;
```

А вместо них задать кнопку **Справка**: `HelpButton = true`. Далее объявляем объект `Хелп` класса `HelpProvider`, 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 9 в главе 1)*. Свойству `HelpNamespace` назначаем HELP-файл, который будет вызываться при нажатии функциональной клавиши <F1>. Мы указали один из файлов ОС 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 сопровождающего книгу электронного архива.

ГЛАВА 13

Программирование простейших веб-ориентированных приложений на Visual C# 12

Создание веб-страницы на языке HTML. Интернет-технологии

HTML (HyperText Markup Language) — это язык для отображения в браузере информации с веб-страницы: текста, изображений, анимации, ссылок на другие веб-страницы. Это язык специальных инструкций браузеру, но его нельзя назвать языком программирования в обычном понимании этого слова, поскольку он не позволяет использовать логику, — так, с его помощью невозможно выполнить даже элементарные арифметические операции. Мы уже ранее приводили примеры HTML-кода для отображения таблиц. В качестве примера в листинге 13.1 представлен HTML-код простейшей веб-страницы, содержащей GIF-изображение и гиперссылку для перехода на другую веб-страницу.

Листинг 13.1. HTML-код простейшей веб-страницы

```
<html>
  <head>
    <title>Простейшая веб-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 веб-страницу, выполненную в формате HTML. Тег `<head>` указывает на наличие текста, содержащего наименование страницы и дополнительные сведения о ней. В раздел `<head>` обычно вложен тег `<title>` для обозначения наименования страницы в строке заголовка. Наименования веб-страниц отображаются в строке заголовка окна браузера. Затем следует тег `<body>`, в котором размещаются: весь остальной текст, изображения, таблицы, гиперссылки и другие эле-

менты содержимого веб-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 — пункты меню **Вставка | WordArt**.

Для создания гиперссылки служит тег <a>. Синтаксис этого тега очевиден. Сам HTML-код записан с помощью Блокнота в файл primer.htm. На рис. 13.1 приведено отображение этого файла в браузере Internet Explorer.

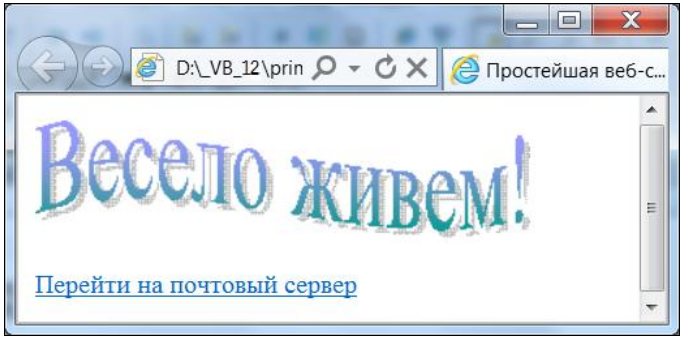


Рис. 13.1. Отображение графического файла в браузере Internet Explorer

Чтобы создавать подобные веб-страницы, вовсе не обязательно знать язык разметки HTML. Можно пользоваться каким-либо редактором веб-страниц, например Microsoft Office FrontPage или Microsoft Office SharePoint Designer. В них интерфейс для создания веб-страницы очень похож на интерфейс текстового редактора MS Word. В крайнем случае, содержание веб-страницы можно набрать непосредственно в редакторе MS Word, но сохранить файл с расширением htm и таким образом получить веб-страницу. Кстати говоря, создавать веб-страницы на языке HTML можно также с помощью Visual Studio 12, для этого следует создать новый проект шаблона **ASP.NET Empty Web Application** и к текущему проекту добавить новый элемент (**Project | Add New Item**) шаблона **HTML page**.

Однако веб-страница, написанная только на языке разметки HTML, будет *статичной*, она способна только лишь отображать какие-либо сведения. Вы знаете, что современная веб-страница является, по сути, программой, способной реагировать на действия пользователя, обеспечивать полноценный интерактивный режим, диалог с пользователем, работу с базой данных и т. д. Обеспечить такую интерактивность можно, например, используя технологию Microsoft ASP.NET, в частности язык Visual C# 12. Технология ASP (Active Server Pages, активные страницы сервера) представляет альтернативу статичным страницам, основанным на "чистом" HTML.

Веб-хостинг на платформах UNIX и Windows

Интернет можно рассматривать как совокупность включенных 24 часа в сутки и связанных между собой специализированных компьютеров, называемых *хостами* (от англ.

host — хозяин, принимающий гостей) или *серверами*. Пользователи со своих локальных (клиентских) компьютеров подключаются через *провайдеров* к Интернету и могут, используя URL-адреса, запрашивать те или иные веб-страницы, расположенные на различных серверах.

Веб-хостинг — это услуга в Интернете по предоставлению дискового пространства и программного обеспечения для веб-страниц различных пользователей. Если ваша веб-страница написана исключительно на языке HTML, то для ее размещения в Интернете подойдет любой веб-хостинг. Однако для размещения *активных веб-страниц* с интерактивным содержимым различные серверы предлагают разное программное обеспечение.

Так, серверы под управлением операционных систем семейства UNIX/Linux обеспечивают веб-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 веб-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. Эта платформа менее популярна, чем технология PHP. Между тем веб-страницы, написанные на языке Visual C# 12, выполняются по разным оценкам примерно в полтора-два раза быстрее, чем написанные на языке PHP. Кроме того, автор, имеющий навыки создания веб-страниц как на PHP, так и на основе ASP.NET, смеет утверждать, что создание веб-страниц на ASP.NET более технологично и удобно, чем на PHP. Чего стоит только отладчик на ASP.NET, снабженный суфлером IntelliSense, для редактирования программного кода!

Клиент-серверное взаимодействие на основе технологии ASP.NET

Схема работы активной веб-страницы следующая. С помощью URL-адреса веб-браузер *клиентского компьютера* запрашивает необходимую активную веб-страницу. В ответ веб-сервер посылает браузеру ASPX-документ, содержащий HTML-разметку и теги `<asp>`. ASPX-документ интерпретируется браузером клиента и представляется в виде веб-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 веб-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 веб-страницей. С помощью библиотеки DLL реализуется весь сценарий взаимодействия с пользователем. Таким образом, вся "активность" страницы выполняется на сторо-

не сервера, а веб-браузеру пользователя (клиента) отсылается лишь результат выполнения в виде HTML-потока.

Как получить библиотеку DLL? Ничего сложного — управляющая среда Visual Studio 12 сама генерирует файл разметки *.ASPX и файл программной поддержки *.ASPX.cs для веб-страницы, причем файл *.ASPX.cs является программным кодом Visual C# 12. А для того чтобы получить библиотеку DLL, следует скомпилировать разрабатываемый проект. При этом получим новую библиотеку DLL в папке bin.

Отладка активного веб-приложения

Тот факт, что веб-страница выполняет команды пользователя на стороне сервера, затрудняет отладку веб-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 12 имеет возможность использовать собственный локальный сервер для отладки активных веб-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 веб-приложения с интерактивным содержимым. Обращение к активным веб-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 12 поддерживает *обе* указанные *возможности* отладки веб-приложения. Менять установку данных возможностей можно через пункты меню **Project | *имя_проекта* Properties**, вкладка **Web**. Причем по умолчанию отладка производится именно с использованием сервера Visual Studio. Это дает возможность отлаживать интерактивные веб-приложения в любой папке винчестера.

Пример 108. Создание простейшей активной веб-страницы на Visual C# 12

Как уже отмечалось, статичные веб-страницы не способны выполнять даже элементарные арифметические действия. Создадим веб-страницу, демонстрирующую способность как раз складывать числа, введенные пользователем. И одновременно подчеркнем единый подход к разработке Windows- и веб-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 веб-приложения откроем Visual Studio 12, в окне **New Project** выполним команду **Instaled** (инсталлированные) **| Templates** (шаблоны) **| Visual Basic | Web**,

в открывшемся списке шаблонов выберем шаблон **ASP.NET Empty Web ApplicationVisual C#**. В поле **Name** укажем имя нашего проекта: `Summa`, при этом снимем флажок **Create directory for Solution**. После щелчка разработчика по кнопке **OK** управляющая среда создаст папку Summa, в которой будет находиться проект нового веб-приложения.

Теперь мы можем приступить к проектированию веб-страницы. Поскольку проект пустой (empty), добавим к текущему проекту веб-форму. Для этого в меню **Project** выберем команду **Add New Item**, в открывшемся окне укажем шаблон **Web Form** и щелкнем на кнопке **Add**. При этом в окне **Solution Explorer** будет добавлен значок файла 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 **TextBox**, метку **Label** и командную кнопку **Button**. Кроме того, в заголовке веб-страницы предусмотрим текст "Введите два числа". Для этого щелкнем правой кнопкой мыши в пределах проектируемой веб-страницы и в контекстном меню выберем пункт **Properties**. В открывшемся окне свойств **Properties** среди свойств документа (**DOCUMENT**) находим свойство **Title**, которое содержит текст заголовка веб-страницы. Этому свойству присваиваем значение: `Введите два числа`.

Уже на этом этапе посмотрим на внешний вид проекта, для чего попытаемся выполнить его, нажав клавишу <F5>. В результате Internet Explorer откроет файл WebForm1.aspx, сгенерированный управляющей средой, и мы получим веб-страницу, изображенную на рис. 13.2.

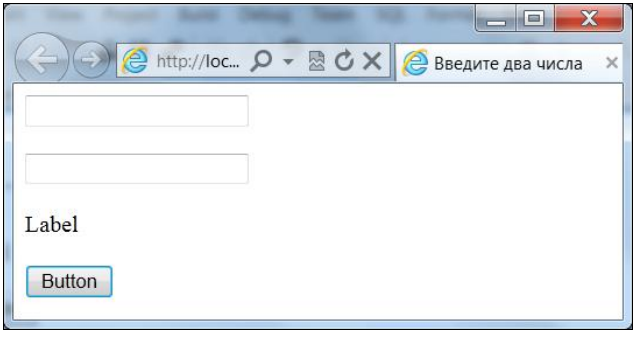


Рис. 13.2. Пример работы активной веб-страницы

Понятно, что пока эта страница является *статической*. Закроем веб-браузер и вернемся к проекту веб-страницы. По умолчанию проектируемая веб-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 веб-страницы обеспечивает файл программной поддержки *.ASPX.cs, содержащий код на Visual C# 12. Чтобы перейти на

вкладку файла программного кода **WebForm1.aspx.cs**, следует, например, воспользоваться контекстным меню проекта веб-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 веб-страницы

```
// Создание простейшей активной веб-страницы на Visual C# 12.
// Веб-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"
            var Z = Convert.ToDouble(TextBox1.Text) +
                       Convert.ToDouble(TextBox2.Text);
            Label1.Text = String.Format("Сумма = {0}", 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 заглавной 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 веб-форма будет иметь такой вид, как представлен на рис. 13.3.

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 веб-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"`.

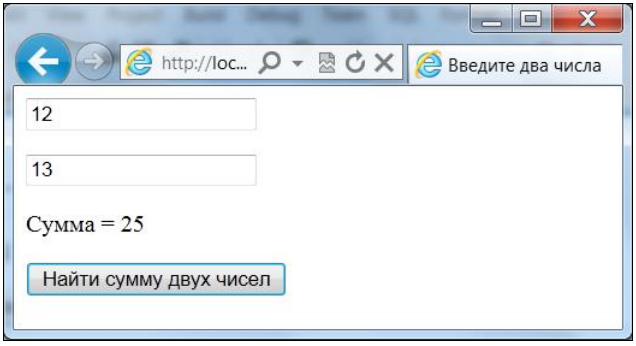


Рис. 13.3. Сумма двух введенных чисел

Убедиться в работоспособности веб-страницы можно, открыв решение Summa.sln в папке Summa сопровождающего книгу электронного архива.

> ***Напоминание***
>
> Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 109. Проверка введенных пользователем числовых данных с помощью валидаторов

Очевидно, что созданная в *примере 108* веб-страница имеет много недостатков, в частности, если пользователь не заполнит текстовые поля, то получит сообщение "Ошибка сервера...". Между тем, следовало бы подсказать пользователю 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 применяли функцию `TryParse`, однако при разработке веб-приложений среди элементов управления присутствуют 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 этой задаче мы будем совершенствовать веб-страницу, созданную нами в *примере 108*. 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 12, в окне **New Project** выберем среди шаблонов **Visual C# Web** шаблон **ASP.NET Empty Web Application**, в поле **Name** укажем имя нового решения `Valid1` и щелкнем на кнопке **OK**. Теперь добавим к текущему проекту веб-форму. Для этого в меню **Project** выберем команду **Add New Item**, в открывшемся

окне укажем шаблон **Web Form** и щелкнем на кнопке **Add**. Далее на вкладке конструктора формы перетащим из панели элементов **Toolbox** раздела **Standard** необходимые нам два текстовых поля, метку и командную кнопку.

В этой задаче мы должны контролировать два текстовых поля, поэтому перенесем мышью из панели **Toolbox** раздела **Validation** соответственно два валидатора **RequiredFieldValidator**, проверяющих факт заполнения текстовых полей, и два валидатора **CompareValidator** для проверки, ввел ли пользователь числа или другие символы. Расположим их в форме, как показано на рис. 13.4.

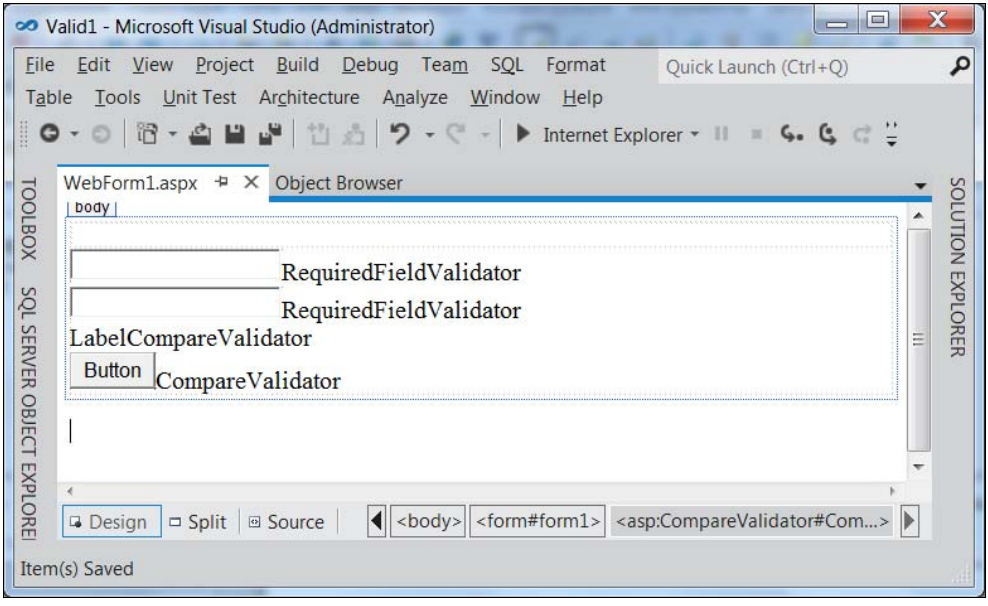


Рис. 13.4. Проектирование веб-формы программы, содержащей валидаторы

Теперь, щелкнув двойным щелчком в пределах проектируемой формы, перейдем на вкладку файла программной поддержки WebForm1.aspx.cs. Здесь введем программный код, представленный в листинге 13.3.

Листинг 13.3. Использование валидаторов на веб-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.ControlToValidate = "TextBox1";
            RequiredFieldValidator1.EnableClientScript = false;
            RequiredFieldValidator1.ErrorMessage =
                                "* Следует заполнить это текстовое поле";
            RequiredFieldValidator2.ControlToValidate = "TextBox2";
            RequiredFieldValidator2.EnableClientScript = false;
            RequiredFieldValidator2.ErrorMessage =
                                "* Следует заполнить это текстовое поле";
            // Проверяем, соответствуют ли данные, введенные в текстовые
            // поля типу Double:
            CompareValidator1.ControlToValidate = "TextBox1";
            CompareValidator1.EnableClientScript = false;
            CompareValidator1.Type = ValidationDataType.Double;
            CompareValidator1.Operator =
                                 ValidationCompareOperator.DataTypeCheck;
            CompareValidator1.ErrorMessage = "* Следует вводить числа";
            CompareValidator2.ControlToValidate = "TextBox2";
            CompareValidator2.EnableClientScript = false;
            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"
            // Если проверка текстовых полей показала, что они
            // заполнены верно:
            if (Page.IsValid == true)
            {
                var Z = Convert.ToDouble(TextBox1.Text) +
                              Convert.ToDouble(TextBox2.Text);
                Label1.Text = String.Format("Сумма = {0}", Z);
            }
        }
    }
}
```

Как видно из текста программы, при загрузке страницы в свойствах валидаторов `RequiredField` мы указали имена полей, подлежащих контролю факта их заполнения, а также привели текст сообщения, выводимого в качестве реакции на эту ошибку. В свойствах валидаторов `Compare` также указали имена полей, подлежащих проверке типа данных `DataTypeCheck`, и текст соответствующего сообщения-реакции. Заметим, что по сравнению с предыдущими версиями Visual Studio свойство валидаторов `EnableClientScript` следует устанавливать в состояние `False`, а при щелчке на кнопке контролировать свойство `Page.IsValid`.

Пример работы запрограммированных валидаторов приведен на рис. 13.5.

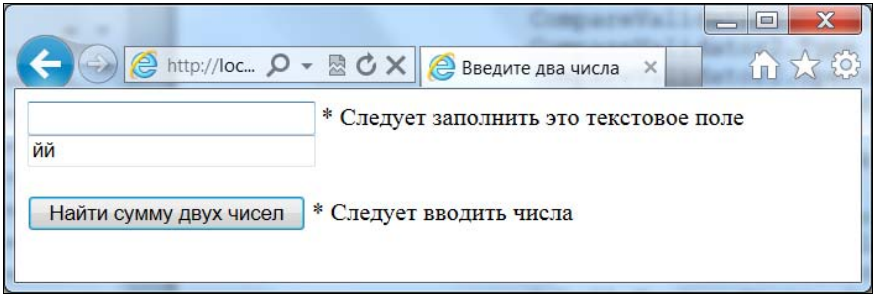


Рис. 13.5. Фрагмент работы веб-страницы с валидаторами

Среди имеющихся на панели **Toolbox** валидаторов уместно здесь отметить **ValidationSummary**. Его можно поместить на какое-либо место на веб-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 запрашивает на веб-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 веб-страницы можно, открыв решение Valid1.sln в папке Valid1 сопровождающего книгу электронного архива.

Пример 110. Проверка достоверности ввода имени, адреса e-mail, URL-адреса и пароля с помощью валидаторов

В этом примере рассмотрим типичную задачу проверки ввода пользователем имени, адреса E-mail, URL-адреса и пароля, например, при регистрации пользователя. Причем

если страница (веб-форма) успешно прошла все этапы проверки, то направляем пользователя на другую, уже разрешенную для этого пользователя, веб-страницу.

Решая поставленную задачу, запустим Visual Studio 12, закажем новый проект из шаблона **ASP.NET Empty Web Application**, укажем в поле **Name** имя `Validations` и щелкнем на кнопке **OK**. К текущему проекту добавим веб-форму. Для этого в меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Form**.

Теперь в конструктор 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 соответствуют вводу имени пользователя, адреса E-mail, URL-адреса персональной веб-страницы пользователя, пароля и подтверждения пароля.

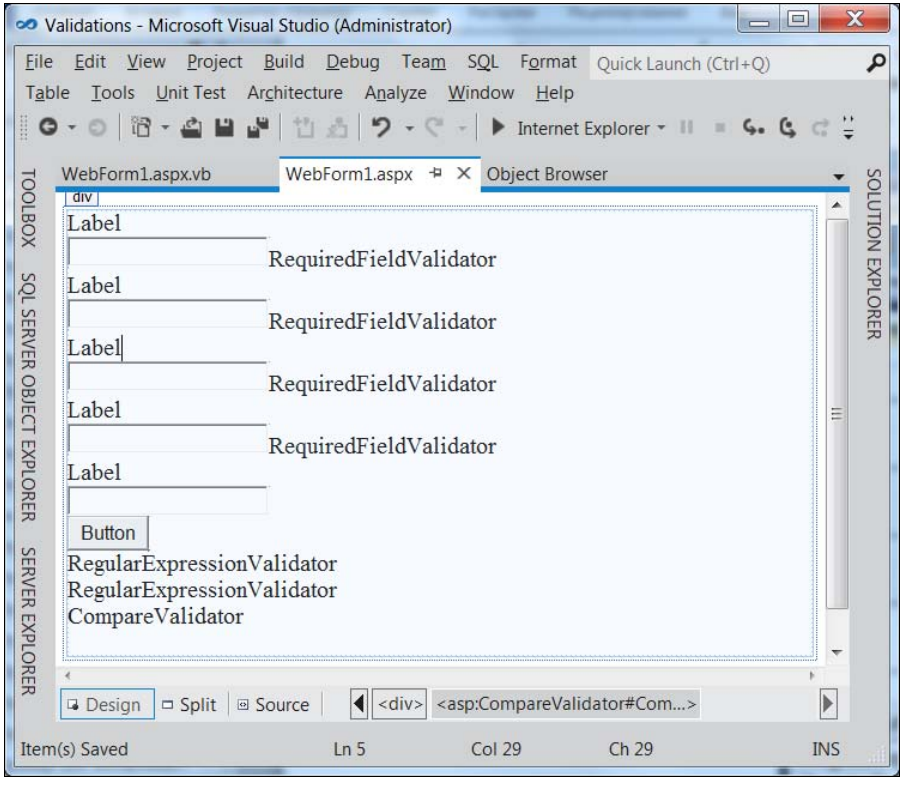


Рис. 13.6. Проектирование веб-формы проверки личных введенных данных пользователя

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 веб-

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 программировании Windows-форм визуально, непосредственно в таблице свойств каждого из элементов управления. Свойства элементов управления можно задать так же в программном коде, как это сделано в листинге 13.4.

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.
// Веб-страница проверяет ввод пользователем имени, e-mail-адреса,
// URL-адреса и пароля, например, при регистрации пользователя. Причем
// если страница (веб-форма) успешно прошла все этапы проверки, то
// направляем пользователя на другую, уже разрешенную для этого
// пользователя, веб-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 веб-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.EnableClientScript = false;
            RequiredFieldValidator1.ErrorMessage =
                        "* Следует заполнить это текстовое поле";
            RequiredFieldValidator2.ControlToValidate = "TextBox2";
            RequiredFieldValidator2.EnableClientScript = false;
            RequiredFieldValidator2.ErrorMessage =
                        "* Следует заполнить это текстовое поле";
            RequiredFieldValidator3.ControlToValidate = "TextBox3";
            RequiredFieldValidator3.EnableClientScript = false;
            RequiredFieldValidator3.ErrorMessage =
                        "* Следует заполнить это текстовое поле";
            RequiredFieldValidator4.ControlToValidate = "TextBox4";
```

```
            RequiredFieldValidator4.EnableClientScript = false;
            RequiredFieldValidator4.ErrorMessage =
                        "* Следует заполнить это текстовое поле";
            RegularExpressionValidator1.ControlToValidate = "TextBox2";
            RegularExpressionValidator1.EnableClientScript = false;
            // Контроль правильности введения e-mail-адреса и адреса
            // веб-страницы:
            RegularExpressionValidator1.ValidationExpression =
                        @"\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*";
            RegularExpressionValidator1.ErrorMessage =
                         "* Следует ввести правильный адрес E-mail";
            RegularExpressionValidator2.ControlToValidate = "TextBox3";
            RegularExpressionValidator2.EnableClientScript = false;
            RegularExpressionValidator2.ValidationExpression =
                        @"http://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?";
            RegularExpressionValidator2.ErrorMessage =
                        "* Следует ввести правильный адрес веб-узла";
            // Контроль правильности введения паспорта путем сравнения
            // содержимого двух полей:
            CompareValidator1.ControlToValidate = "TextBox4";
            CompareValidator1.ControlToCompare = "TextBox5";
            CompareValidator1.EnableClientScript = false;
            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.html");
        }
    }
}
```

Как видно из текста программы, при обработке события загрузки веб-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 эти текстовые поля вместо вводимых символов будут видны жирные точки.

Далее следуют назначения свойств валидаторам обязательности заполнения полей `RequiredField`, знакомым нам из *примера 109*. Следует подробнее остановиться на валидаторе `RegularExpression`. Регулярное выражение, используемое для проверки досто-

верности 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);
- `([-+.]\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;
- `*@\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 щелчком правой кнопки мыши открыть 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).

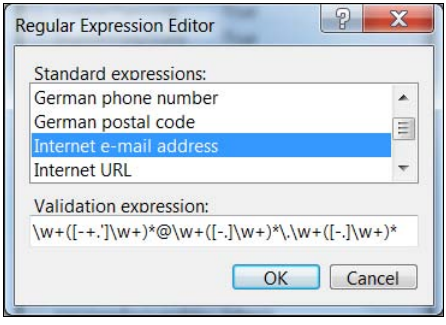


Рис. 13.7. Список типичных регулярных выражений

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. Пример работы данной веб-формы в момент ее заполнения показан на рис. 13.8.

Вернемся еще раз к программному коду. Здесь при обработке события "щелчок на кнопке **Готово**" происходит проверка, вызвана (загружена) ли данная веб-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.html, предусмотренную сценарием диалога с пользователем.

Создать в текущем проекте новую статическую веб-страницу очень просто, для этого следует в пункте меню **Project** выбрать команду **Add New Item** и двойным щелчком

щелкнуть на шаблоне **HTML page**. Далее в конструкторе страницы, используя панель элементов **Toolbox**, спроектировать необходимую веб-страницу. Заменить имя по умолчанию **HTMLPage1.html** на **Next_Page.html** удобно в окне **Solution Explorer** (Обозреватель решений).

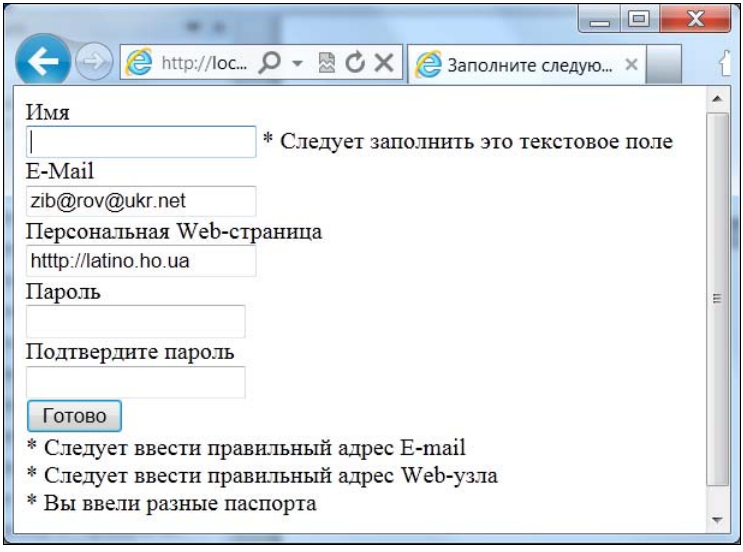


Рис. 13.8. Пример работы веб-страницы при заполнении полей

Теперь, когда вы добавили к текущему проекту HTML-файл, при правильном заполнении искомой формы, мы попадаем на уже разрешенную страницу (рис. 13.9).

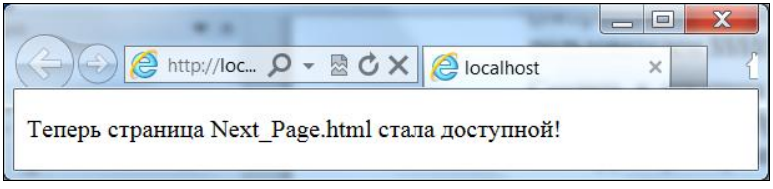


Рис. 13.9. Переход на HTML-файл при правильном заполнении формы

Убедиться в работоспособности веб-страницы можно, открыв решение Validations.sln в папке Validations сопровождающего книгу электронного архива.

Пример 111. Регистрация и аутентификация пользователя с помощью базы данных Access

Данный пример включает в себя три веб-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, например, версии 2010, далее в пункте меню **Файл** выполним команду **Создать | Новая база данных**, затем щелкнем по значку папки и в окне **Файл новой базы данных** укажем имя файла: `D:\Web.mbd` и тип файла: **Базы данных Microsoft Access 2002-2003 (*.mdb)**. Далее следует создать таблицу в режиме конструктора в соответствии с очевидным интерфейсом. Типы данных во всех проектируемых полях — текстовые, а имена полей приведены на рис. 13.10.

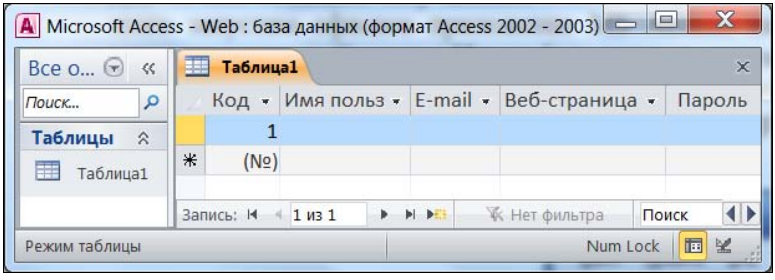


Рис. 13.10. Проектирование таблицы базы данных в режиме конструктора

В этом примере мы не собираемся заполнять поля **E-mail** и **Веб-страница**, и приводим их здесь, чтобы была преемственность с *примером 110*. Вообще говоря, следует стараться чувствовать баланс между желанием больше знать о пользователе и *не злоупотреблять терпением пользователя*, который может потерять интерес к нашему сайту и перейти на другой ресурс.

Теперь создадим регистрационную форму в системе Visual Studio 12, для этого после запуска системы закажем новый проект из шаблонов (**Templates**) **Web** шаблон **ASP.NET Empty Web Application Visual C#**, в поле **Name** укажем имя `Login.sln`. К текущему проекту добавим веб-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Form**.

Далее, попав в конструктор веб-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. 13.11.

Как можно видеть, 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,

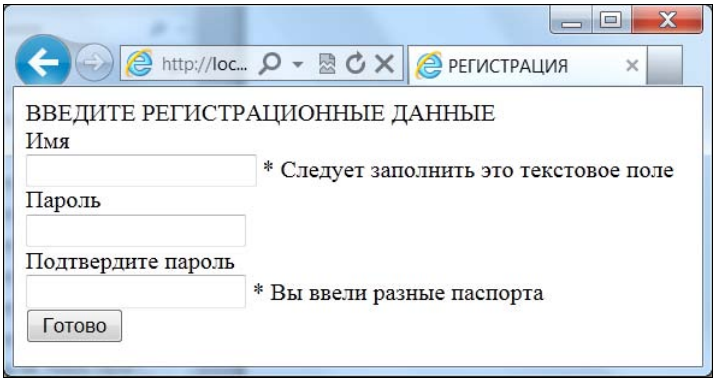


Рис. 13.11. Пример работы регистрационной формы

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 веб-формы:
// Registration.aspx и 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;
// Другие директивы using удалены, поскольку они не используются в данной
// программе.
// Добавим директиву для краткости выражений:
using System.Data.OleDb;
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.EnableClientScript = false;
        RequiredFieldValidator1.ErrorMessage =
                    "* Следует заполнить это текстовое поле";
        RequiredFieldValidator2.ControlToValidate = "TextBox2";
        RequiredFieldValidator2.EnableClientScript = false;
        RequiredFieldValidator2.ErrorMessage =
                    "* Следует заполнить это текстовое поле";
        // Контроль правильности введения паспорта путем сравнения
        // содержимого двух полей:
        CompareValidator1.EnableClientScript = false;
        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:
        var 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 Ситуация1)
        {
            Response.Write("<br><br>" + Ситуация1.Message);
            return;
        }
        var Команда = new OleDbCommand();
        // ДОБАВЛЕНИЕ ЗАПИСИ О ПОЛЬЗОВАТЕЛЕ В БД.
        // Строка SQL-запроса
        var 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 Ситуация2)
            {
                Response.Write("<br><br>" + Ситуация2.Message);
                return;
            }
            Подключение.Close();
            // Перенаправление на уже разрешенную страницу
            Response.Redirect("Secret.aspx");
        }
    }
}
```

Как видно из текста программы, в программном коде при обработке события загрузки формы происходит проверка достоверности введенных пользователем данных с помощью валидаторов так же, как и в *примере 110*. 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 веб-страницу для проверки имени пользователя и пароля с использованием таблицы **Аутентифицированные пользователи** в базе данных. Для этого добавим к текущему проекту новую веб-форму, выбрав в пункте меню **Project** команду **Add New Item | Web Form**. Назовем эту новую форму `Login.aspx`. Проект новой формы представлен на рис. 13.12. Здесь потребовалось пять меток, два текстовых поля и две командные кнопки. Заметьте, что пятую метку мы расположили между кнопками.

Внешний вид формы, запрашивающей у пользователя имя и пароль с их проверкой в базе данных, показан на рис. 13.13.

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;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// Добавим директиву для краткости выражений:
using System.Data.OleDb;
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"
            var ПОЛЬЗОВАТЕЛЬ_АУТЕНТИФИЦИРОВАН = false;
            // Строка подключения
            var 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 Ситуация1)
            {
                Label5.Text = Ситуация1.Message;
            }
            // Строка SQL-запроса для проверки имени и пароля
            var 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
                var 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 Ситуация2)
        {
            Label5.Text = Label5.Text + "<br />" + Ситуация2.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");
    }
}
}
```

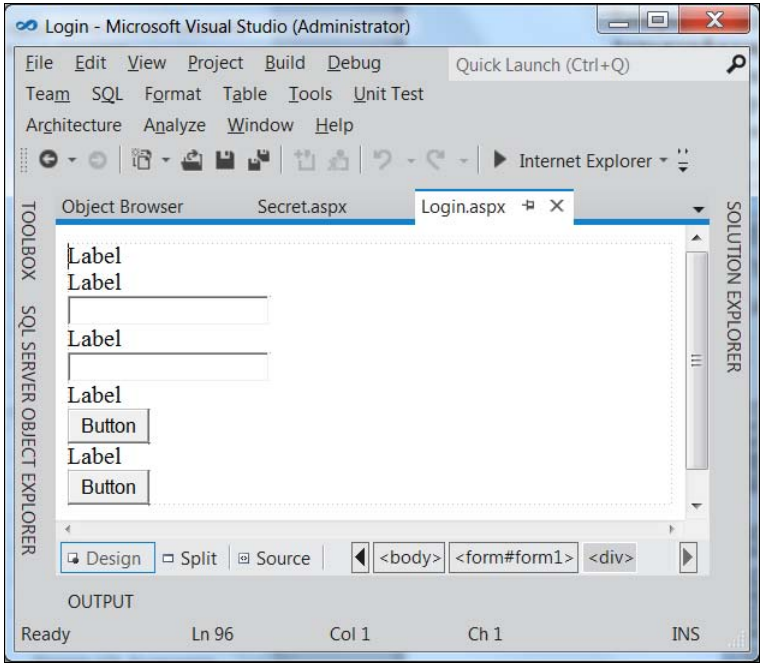


Рис. 13.12. Проект формы аутентификации пользователя

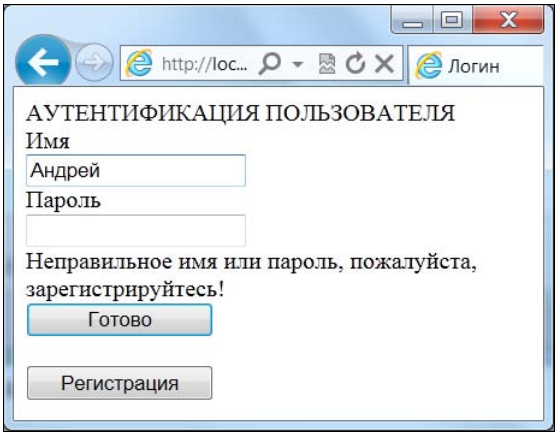


Рис. 13.13. Форма аутентификации пользователя

Здесь при нажатии пользователем кнопки **Готово** производится SQL-запрос к базе данных на наличие записи с полями имя пользователя и пароля, полученными от `TextBox1` и `TextBox2`. Если такая запись найдена, то делаем вывод, что *пользователь аутентифицирован*, и направляем его на уже разрешенную для него веб-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 веб-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 веб-форму, называем ее Secret.aspx. В эту форму из панели элементов раздела **Standard** перетаскиваем метку **Label** и командную кнопку **Button**. В листинге 13.7 приведен файл программной поддержки для файла-разметки Secret.aspx.

Листинг 13.7. Веб-форма Secret.aspx

```
// Третья форма Secret.aspx. К этой странице имеют доступ только
// зарегистрированные пользователи. Вначале пытаемся (Try) определить,
// с какой веб-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;
using System.Web.UI.WebControls;
// 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 веб-страницы вы пришли на данную
            // страницу, т. е. определение локального адреса (пути)
            try
            {
                URL_адрес = Request.UrlReferrer.LocalPath;
                // Более эффективно определять абсолютный виртуальный адрес:
                // URL_адрес = Request.UrlReferrer.AbsoluteUri
                if (URL_адрес == @"/Login.aspx" ||
                       URL_адрес == @"/Registration.aspx")
                {
                    Label1.Text = "Поскольку Вы являетесь " +
                        "зарегистрированным пользователем, то имеете " +
                        "доступ к закрытой информации. Вы пришли на эту" +
                        " страницу со страницы " + URL_адрес;
                    return;
                }
            }
            catch (Exception Ситуация)
            {
                Label1.Text = "Вы не являетесь зарегистрированным " +
                   "пользователем, поскольку пришли на эту страницу " +
                   "набрав URL-адрес в адресной строке браузера.<br />" +
                   Ситуация.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 веб-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 виртуальный адрес, как показано в комментарии. Далее разрешаем доступ к закрытой информации только пользователям, пришедшим на данную страницу с веб-страниц Registration.aspx или Login.aspx.

Убедиться в работоспособности данных веб-страниц (в данном примере их три) можно, открыв решение Login.sln в папке Login сопровождающего книгу электронного архива.

Пример 112. Таблица с переменным числом ячеек, управляемая двумя раскрывающимися списками

При веб-строительстве очень часто приходится в самых различных целях работать с таблицами. В этом примере продемонстрируем, как с помощью двух раскрывающихся списков **DropDownList** можно заказывать в таблице необходимое число рядов и столбцов.

Программируя поставленную задачу, запустим систему Visual Studio 12, создадим новый проект шаблона **ASP.NET Empty Web Application Visual C#** и назовем его `tab.sln`. К текущему проекту добавим веб-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Form**.

Далее в конструкторе **WebForm1.aspx** из панели элементов **Toolbox** раздела **Standard** 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 веб-формы, выбираем команду **View Code** и попадаем 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. Веб-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 = "- количество строк";
            Label2.Text = "- количество столбцов";
            Button1.Text = "Обновить таблицу";
        }
        protected void Button1_Click(object sender, EventArgs e)
        {
            // Один цикл по ячейкам таблицы вложен в другой по ее рядам:
            for (int i = 1; i <= int.Parse(
                                DropDownList1.SelectedItem.Value); i++)
            {
                var РЯД = new TableRow();
                // Цикл по ячейкам:
                for (int j = 1; j <= int.Parse(
                                DropDownList2.SelectedItem.Value); j++)
                {
                    var ЯЧЕЙКА = new TableCell();
                    ЯЧЕЙКА.Text = String.Format("Ряд {0}, Колон {1}", i, 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 веб-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. 13.14.

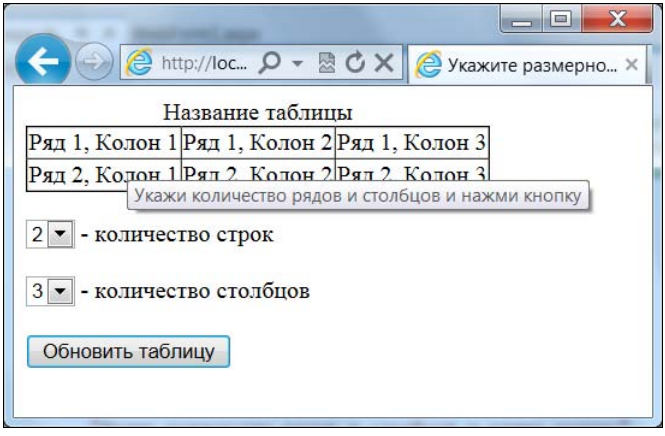


Рис. 13.14. Пример работы веб-страницы с таблицей

Убедиться в работоспособности веб-страницы можно, открыв решение tab.sln в папке tab сопровождающего книгу электронного архива.

Пример 113. Организация раскрывающегося меню гиперссылок с помощью *DropDownList*

Переходы на другие веб-страницы, так называемую *навигацию*, в статических HTML-страницах обычно организуют с помощью ряда гиперссылок. Например, сайт состоит из десяти веб-страниц. На каждой из этих страниц в левой части страницы имеется колонка из десяти строчек текстовых гиперссылок для перехода на эти страницы. В этом примере продемонстрируем, как можно организовать переход на разные страницы сайта с помощью раскрывающегося списка **DropDownList**.

Для реализации задачи запустим систему Visual Studio 12, выберем новый проект из шаблона **ASP.NET Empty Web Application Visual C#**, дадим ему имя: `Menu`. К текуще-

му проекту добавим веб-форму. Для этого в меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Form**.

Затем на проектируемую веб-форму перетащим из панели **Toolbox** раздела **Standard** текстовую метку **Label** и элемент управления **DropDownList**. Теперь перейдем к вкладке файла программной поддержки **WebForm1.aspx.cs** и здесь организуем гиперссылки от каждого пункта меню на некоторые веб-страницы, созданные в этой главе. Текст программы приведен в листинге 13.9.

Листинг 13.9. Организация навигации по веб-страницам

```
// Веб-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 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"));
            // Делать 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 веб-страницу:";
        }
        protected void DropDownList1_SelectedIndexChanged(
                                        Object sender, EventArgs e)
```

```
        {
            // Выполнить команду перейти 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 думать про постбэк.
        // Чтобы получить обработчик этого события, надо просто его
        // написать: protected void Page_PreInit...
        //}
    }
}
```

Здесь, так же как и в *примере 112*, мы заполняли пункты раскрывающегося списка при первой загрузке страницы `IsPostBack = False`. Это можно было бы делать также при инициализации веб-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 можно видеть, все выбираемые пользователем веб-страницы являются активными ASPX-страницами. Чтобы добавить их в текущий проект, следует выбрать в меню **Project** команду **Add New Item** и двойным щелчком щелкнуть на шаблоне **Web Form**. В окне **Solution Explorer** щелкнем правой кнопкой мыши на названии файла и переименуем новую веб-форму из WebForm2.aspx в `Validations.aspx`. Аналогично следует добавить вторую вызываемую веб-форму и назвать ее `tab.aspx`.

Пример работы данной веб-страницы приведен на рис. 13.15.

Упомянутая функция `Redirect` является методом объекта `Response`. Очень часто пользуются методом `Write` объекта `Response` для вывода в веб-форму каких-либо данных,

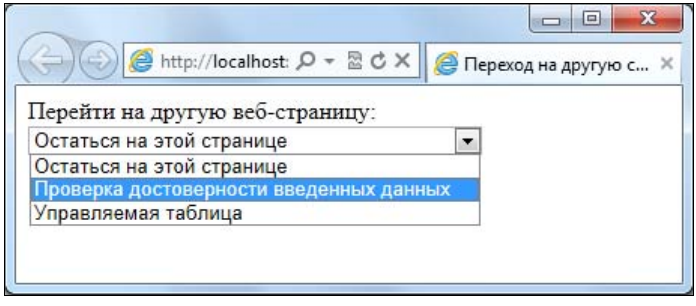


Рис. 13.15. Пример работы веб-страницы, использующей раскрывающийся список

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 веб-страницы можно, открыв решение Menu.sln в папке Menu сопровождающего книгу электронного архива.

Пример 114. Передача данных между веб-страницами через параметры гиперссылки

Передача данных между веб-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 веб-документ Target.aspx. Прочитать эти параметры в Target.aspx можно с помощью объекта `Request`.

В данном примере создадим две веб-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. На этой новой странице с помощью объекта `Request` получаем оба переданных параметра и выводим их в веб-документе.

Для реализации данной задачи запустим Visual Studio 12, далее в окне **New Project** выберем шаблон **ASP.NET Empty Web Application Visual C#**, в поле **Name** укажем имя `Ссылка`. К текущему проекту добавим веб-форму. Для этого в меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Form**.

В этом проекте будем программировать две веб-формы: одну назовем Source.aspx, а вторую — Target.aspx. Сначала запроектируем первую форму, для этого в окне **Solution Explorer** переименуем файл WebForm1.aspx в `Source.aspx`. Теперь на вкладку **Source.aspx** из панели **Toolbox** раздела **Standard** перенесем элемент управления гиперссылка **HyperLink**. Содержимое файла программной поддержки Source.aspx.cs приведено в листинге 13.10.

Листинг 13.10. Передача данных между веб-страницами через параметры гиперссылки

```
// Передача данных между веб-страницами через параметры гиперссылки.
// В данном примере имеем две веб-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 на
// 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 Ссылка
{
    public partial class WebForm1 : 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();
            var ЧИСЛО = Генератор.Next(0, 3);
            var t1 = String.Empty;
            var t2 = String.Empty;
            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 веб-страницу Target.aspx. В результате работы веб-формы Source.aspx в браузере мы получим следующее (рис. 13.16).

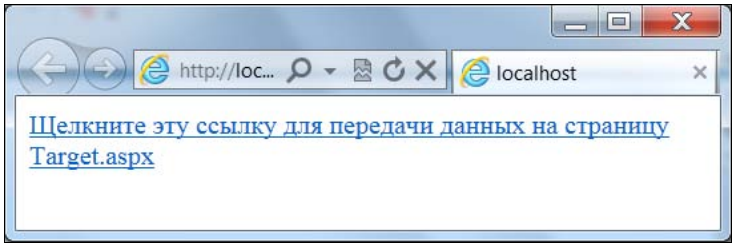


Рис. 13.16. Пример работы веб-страницы с гиперссылкой

Далее, щелкнув на этой ссылке, мы попадаем на вызываемую веб-страницу Target.aspx. Однако в нашем проекте еще нет веб-страницы Target.aspx. Для добавления новой веб-формы в меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на значке шаблона **Web Form**. В окне **Solution Explorer** переименуем (щелкнув правой кнопкой мыши) новую веб-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 веб-страницами через параметры гиперссылки.
// На этой новой странице Target.aspx.cs с помощью объекта Request
// получаем оба переданных параметра и выводим их в веб-документе
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace Ссылка
{
    public partial class Target : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            // Получаем параметры, переданные с веб-страницы, с которой была
            // вызвана данная страница
            // Request — это запрос
            var ИМЯ1 = Request.QueryString.Get("N1");
            var ИМЯ2 = Request.QueryString.Get("N2");
            try
            {
                var URL = Request.UrlReferrer.AbsoluteUri;
                Response.Write("<br />Вы попали на данную веб-" +
                                        "страницу со страницы: " + URL + "<br />");
                Response.Write("<br />Страница " + URL + " передала на " +
                            "данную страницу имя: " + ИМЯ1 + " и фамилию " + ИМЯ2);
            }
```

```
            catch (Exception Ситуация)
            {
                Response.Write("<br />Вы пытаетесь попасть на данную страницу " +
                   "не через ссылку, а набрав URL-адрес в адресной " +
                   "строке браузера.<br />" +
                Ситуация.Message);
            }
        }
    }
}
```

В этом программном коде извлекаем переданные параметры `n1` и `n2` (в данном случае большие и маленькие буквы равнозначны) методом `QueryString.Get` объекта `Request`. С помощью объекта `Request` получаем также абсолютный URL-адрес вызывающей страницы. Для вывода какого-либо текста на веб-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. Результат работы веб-страницы приведен на рис. 13.17.

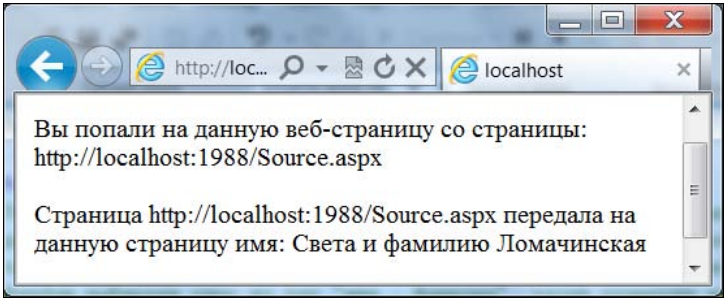


Рис. 13.17. Целевая веб-страница

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 веб-страницы можно, открыв соответствующее решение в папке Ссылка сопровождающего книгу электронного архива.

Пример 115. 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 определенный элемент из списка и т. д., т. е. выполнить те или иные манипуляции с элементами управления, а затем перейти на другую веб-страницу и отобразить на этой новой странице измененные свойства элементов управления предыдущей страницы. Замечу, что это — типичная ситуация.

Поставим следующую конкретную задачу. Имеем веб-форму, написанную на "чистом" HTML, т. е. в данной разметке нет тегов ASP, нет JavaScript-кода, нет PHP-скрипта, есть только теги HTML. Веб-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 веб-страницу для обработки и отображения принятых данных на этой новой странице.

Для решения задачи запустим Visual Studio 12, закажем новый проект шаблона **ASP.NET Empty Web Application Visual C#**. В поле **Name** укажем имя `Передача1`. Добавим веб-форму, содержащую только теги HTML, для этого в меню **Project** выберем команду **Add New Item** и в открывшемся 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 в листинге 13.12.

Листинг 13.12. HTML-разметка веб-страницы, содержащей форму

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Заполните форму</title>
</head>
<body>
    <form action="res.aspx" method="post">
    Введите ваше имя:<br />
    <input type="text" name="name" size="20"
           maxlength="20" value="Зиборов" />
    <br />
    Введите ваш E-Mail:
    <br />
    <input type="text" name="email" size="20"
           maxlength ="40" value="ziborov@ukr.net" />
    <br />
    <input type="submit" value="Отправить" />
    </form>
</body>
</html>
```

Обратите внимание на тег `<form>` — здесь атрибут `action` (действие) формы ссылается на страницу `res.aspx`, 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.

Заметим, что Visual Studio 12 содержит удобный HTML-редактор, который зеленым подчеркиванием указывает на синтаксические ошибки, допущенные при вводе HTML-разметки. Для примера, на рис. 13.16 мы не заключили значение атрибута `type` в двойные кавычки. Редактор HTML-разметки подчеркнул этот фрагмент зеленой волнистой линией, и при "зависании" указателя мыши над ним мы получили сообщение об этой ошибке: **Validation (XHTML5): Attribute values must be enclosed in quotation marks** (рис. 13.18).

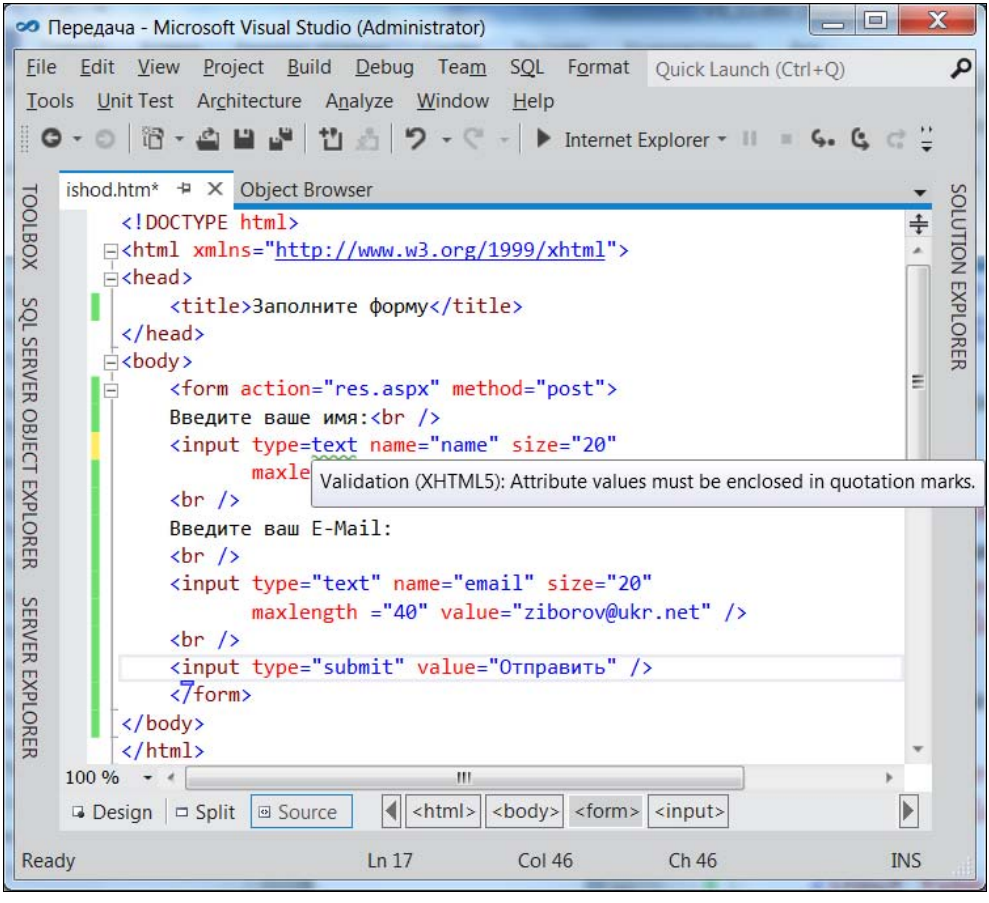


Рис. 13.18. HTML-редактор указывает синтаксическую ошибку во всплывающей подсказке

Нажмем клавишу <F5> для запуска данного проекта, в результате получим форму, поля которой уже заполнены (рис. 13.19).

Теперь создадим веб-страницу, которая будет обрабатывать данные, принятые с формы. Для этого в меню **Project** выберем команду **Add New Item**, в новом открывшемся

окне укажем шаблон **Web Form** и добавим его к нашему проекту. В окне **Solution Explorer** переименуем новый файл в `res.aspx`. В этой новой странице текущего проекта мы не будем использовать никаких элементов 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 13.13.

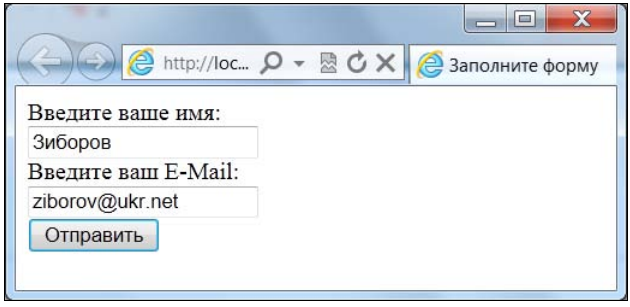


Рис. 13.19. Форма для заполнения и дальнейшей обработки

Листинг 13.13. Формирование веб-страницы, получающей данные

```
// Данная веб-страница получает имя пользователя и адрес его электронной
// почты от веб-формы ishod.htm и отображает эти данные на странице
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace Передача1
{
    public partial class res : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Page.Title = "Прием данных от страницы ishod.htm";
            var Метод = Request.HttpMethod;
            var Имя = Request.Form.Get("name");
            var Почта = Request.Form.Get("email");
            Response.Write("Передача данных произведена 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`. Пример работы веб-страницы представлен на рис. 13.20.

Убедиться в работоспособности проекта можно, открыв решение Передача1.sln в папке Передача1 сопровождающего книгу электронного архива.

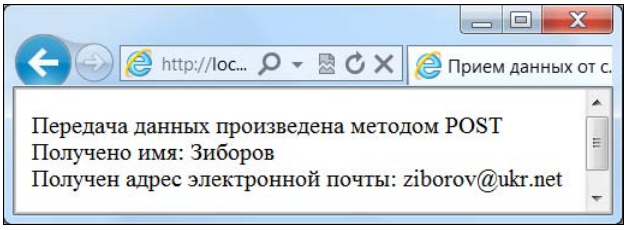


Рис. 13.20. Отображение данных формы на новой странице

Пример 116. Передача значений элементов управления на другую веб-страницу с помощью объекта *PreviousPage*

Покажем, как можно решить задачу из *примера 115* методами объекта `PreviousPage`. Итак, на начальной веб-странице имеем командную кнопку **ПЕРЕХОД** и текстовое поле, которое заполняет пользователь. После щелчка на кнопке происходит переход на другую веб-страницу. На новой странице отображается содержимое текстового поля и надпись на кнопке из предыдущей страницы.

Для решения этой задачи запустим Visual Studio 12, закажем новый проект из шаблона **ASP.NET Empty Web Application Visual C#**. В поле **Name** укажем имя `Передача2`. Далее добавим в текущий проект начальную веб-форму, для этого в меню **Project** выполним команду **Add New Item** и в открывшемся 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 панели элементов **Toolbox** раздела **Standard** перенесем командную кнопку **Button** и текстовое поле **TextBox**. На вкладке программного кода (перейдем туда, нажав клавишу <F7>) напишем текст, приведенный в листинге 13.14.

Листинг 13.14. Заполнение данных на первой веб-странице

```
// На начальной веб-странице имеем командную кнопку "ПЕРЕХОД" и текстовое
// поле, которое заполняет пользователь. После щелчка на кнопке происходит
// переход на другую веб-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 щелчке на кнопке будет осуществляться на страницу, указанную в этом свойстве. Однако в комментарии мы указали, как альтернативу, использование метода `Redirect` объекта `Response`.

Кстати, веб-формы res.aspx, на которую мы имеем ссылку в программном коде, еще нет в нашем проекте. Добавим эту форму так же, как добавили исходную форму, т. е. выполнив в 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 программного кода **res.aspx.cs** напишем текст, приведенный в листинге 13.15.

Листинг 13.15. Получение данных на второй веб-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:
                var S1 = Request.Form.Get("TextBox1");
                var 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 Ситуация)
            {
                Response.Write("Начальная веб-страница должна " +
                               "содержать<br />текстовое поле и командную "
                               + "кнопку.<br />" + Ситуация.Message);
            }
        }
    }
}
```

Как видно из программного кода, новая, результирующая, форма не содержит ни одного элемента управления. Вывод текста в форму осуществляем с помощью 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`.

На рис. 13.21 показана исходная форма ishod.aspx с текстовым полем, подлежащим заполнению, и командной кнопкой.

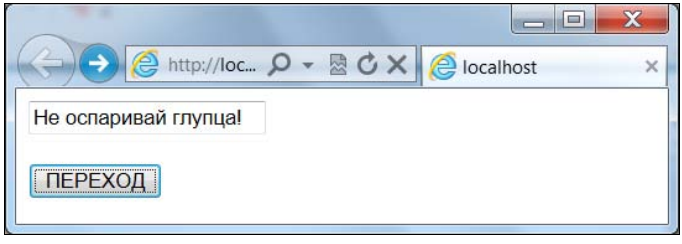


Рис. 13.21. Первоначальная форма

На рис. 13.22 продемонстрирована результирующая форма, где отображены значения элементов управления первоначальной формы.

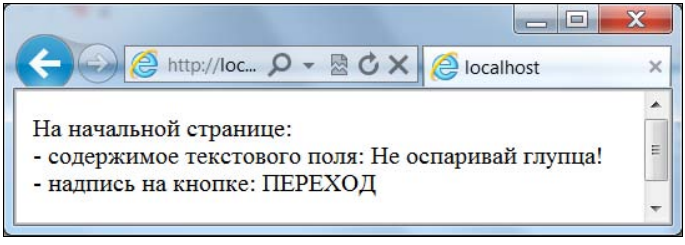


Рис. 13.22. Результирующая форма

Убедиться в работоспособности веб-страницы можно, открыв решение Передача2.sln в папке Передача2 сопровождающего книгу электронного архива.

Пример 117. Отображение табличных данных в веб-форме с помощью элемента управления *GridView*

В этом примере покажем, как можно вывести какие-либо данные, например два строковых массива в виде таблицы, в веб-форму с помощью элемента управления **GridView** (Просмотр/обзор сетки данных в таблице) и объекта **DataTable**. Мы решали подобную задачу для Windows-приложения. Между тем для веб-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 нашей задаче таблица телефонов представлена в виде двух строковых массивов. Требуется минимальными усилиями вывести в веб-форму эти массивы в виде таблицы.

Для решения задачи запускаем Visual Studio 12, заказываем новый проект из шаблона **ASP.NET Empty Web Application Visual C#**, в поле **Name** вводим имя `TabGrdWeb`. К текущему проекту добавим веб-форму. Для этого в меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Form**.

Далее перетащим в форму из панели **Toolbox** (подраздел **Data**) элемент управления **GridView**. Содержимое файла программной поддержки WebForm1.aspx.cs приведено в листинге 13.16.

Листинг 13.16. Вывод табличных данных в веб-форму с помощью элемента управления *GridView*

```
// Вывод табличных данных в веб-форму с помощью элемента управления
// GridView. В данной веб-странице организован вывод двух строковых
// массивов в таблицу веб-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 веб-форму";
            String[] Имена = {"Андрей — раб", "ЖЭК",
                              "Мама — дом", "Карапузова Маша"};
            String[] Тлф = {"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 (var i = 0; i <= 3; i++)
                Таблица.Rows.Add(Имена[i], Тлф[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`. Пример работы веб-страницы показан на рис. 13.23.

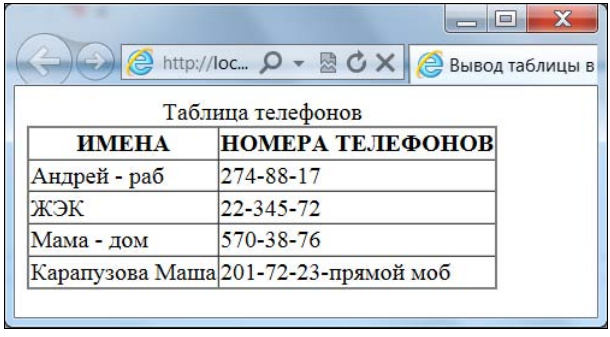


Рис. 13.23. Пример работы веб-страницы с табличными данными

Замечу, что эту 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 веб-страницы можно, открыв решение TabGrdWeb.sln в папке TabGrdWeb сопровождающего книгу электронного архива.

Пример 118. Отображение хэш-таблицы в веб-форме

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 — значе-

ние". 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 *примере 117*, 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). Примерами хэш-таблиц являются англо-русский и другие словари, а также база доменных имен, которая доменному имени сопоставляет IP-адрес. По принципу хэш-таблицы организованы объекты `ViewState` и `Session` технологии ASP.NET.

Поставим следующую задачу — сопоставить в хэш-таблице государства в качестве ключей, а их столицы — в качестве значений. Далее, используя элемент управления **GridView**, выведем эту хэш-таблицу на веб-страницу. Для этого запустим Visual Studio 12, закажем новый проект шаблона **ASP.NET Empty Web Application Visual C#**, в поле **Name** укажем имя `Hash_Grid`. К текущему проекту добавим веб-форму. Для этого в меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком 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 13.17.

Листинг 13.17. Хэш-таблица в веб-форме

```
// Вывод в веб-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 веб-страницу
using System;
using System.Web.UI.WebControls;
// 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 — значение" так:
            Хэш["Украина"] = "Киев";
```

```
            // А можно добавлять так:
            Хэш.Add("Россия", "Москва");
            // Здесь государство — это ключ, а столица — это значение
            Хэш.Add("Беларусь", "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(ОднаПара.Key, ОднаПара.Value);

            // Немного другое свойство, чем в WindowsApplication:
            GridView1.Caption = "Таблица государств"; // Заголовок таблицы
            GridView1.BorderWidth = 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 веб-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, так же как и в *примере 117*, для `GridView1` указываем в качестве источника данных заполненный объект `DataTable`. Пример работы данной веб-страницы показан на рис. 13.24.

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 веб-страницы можно, открыв решение Hash_Grid.sln в папке Hash_Grid сопровождающего книгу электронного архива.

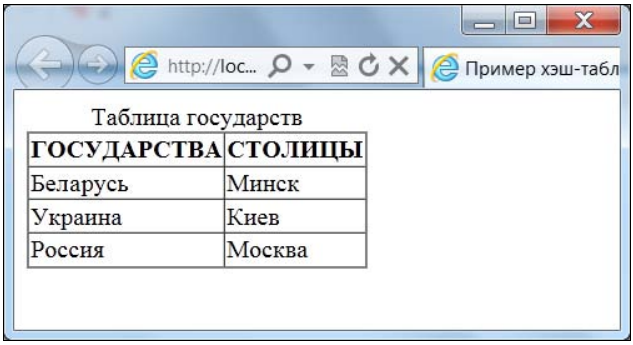
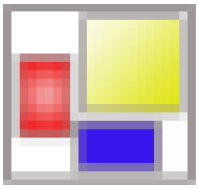


Рис. 13.24. Пример работы веб-страницы с хэш-таблицей

ГЛАВА 14



Типичные веб-ориентированные приложения ASP.NET на Visual C# 12

Пример 119. Чтение/запись текстового файла веб-приложением

Активная веб-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 веб-приложение, читающее текстовый файл в текстовое поле, при этом пользователь имеет возможность редактировать текст и сохранять его в том же файле.

Для решения поставленной задачи запустим Visual Studio 12, закажем новый проект из шаблона **ASP.NET Empty Web Application Visual C#**, имя веб-приложения: `RW_txt`. К текущему проекту добавим веб-форму. Для этого в меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Form**.

В конструкторе формы из панели **Toolbox** раздела **Standard** перетащим мышью текстовое поле **TextBox** и две командные кнопки **Button**. Расположим их на форме, 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 веб-приложением. Веб-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 ИмяФайла; // — имя файла используется в обеих процедурах
        protected void Page_Load(object sender, EventArgs e)
```

```
        {
            Page.Title = "Здесь кодировка Win1251";
            Button1.Width = 95; Button2.Width = 95;
            Button1.Text = "Читать"; Button2.Text = "Сохранить";
            Button1.Focus();
            // Разрешаем многострочие:
            TextBox1.TextMode = TextBoxMode.MultiLine;
            ИмяФайла = 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(ИмяФайла,
                                    System.Text.Encoding.GetEncoding(1251));
                TextBox1.Text = ЧИТАТЕЛЬ.ReadToEnd();
                ЧИТАТЕЛЬ.Close();
            }
            catch (System.IO.FileNotFoundException Ситуация1)
            {
                Response.Write("Нет такого файла <br />" +
                                            Ситуация1.Message);
            }
            catch (Exception Ситуация2)
            { // Отчет о других ошибках
                Response.Write("Ошибка чтения файла  <br />" +
                                            Ситуация2.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(ИмяФайла, false,
                                    System.Text.Encoding.GetEncoding(1251));
                ПИСАТЕЛЬ.Write(TextBox1.Text);
                ПИСАТЕЛЬ.Close();
            }
            catch (Exception Ситуация)
            {
                // Отчет обо всех возможных ошибках
                Response.Write("Ошибка записи файла <br />" +
                                            Ситуация.Message + " <br />");
            }
        }
    }
}
```

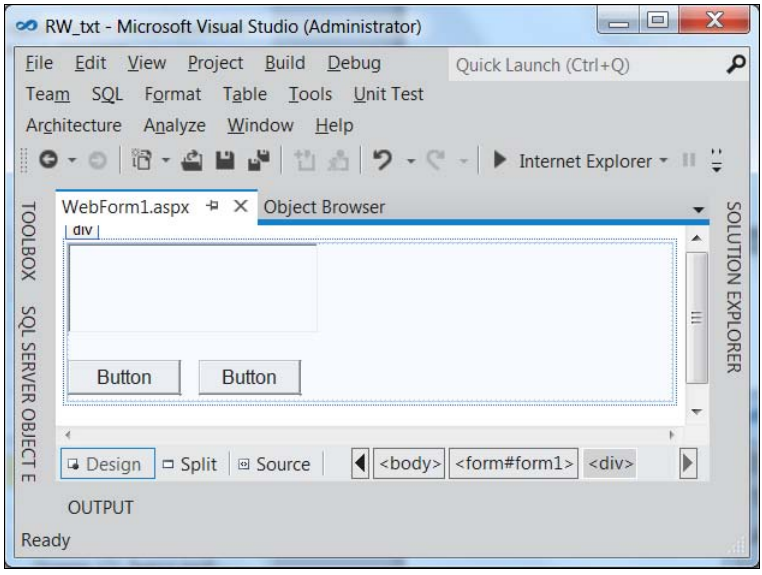


Рис. 14.1. Форма приложения в режиме конструктора

В начале программного кода объявляем имя файла в качестве внешней переменной 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 веб-приложение. При этом мы не сможем воспользоваться методом `GetCurrentDirectory` класса `IO.Directory`, поскольку эта функция нам укажет на системную папку C:\Windows\system32. Поэтому применим метод `PhysicalApplicationPath` объекта `Request`. При обработке события "щелчок на кнопке **Читать**" читаем текстовый файл, используя объект `StreamReader`. Так же как и при записи файла, эта 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 при выполнении вашего приложения выдать сообщение об ошибке сервера вследствие отказа в доступе к задействованному текстовому файлу. То есть мы 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 файле txt.txt и выбрав команду **Свойства**, мы получим свойства файла в таком виде, как показано на рис. 14.2.

Как можно видеть, в окне свойств файла txt.txt отсутствует вкладка **Безопасность**. Чтобы ее добавить, следует в Проводнике выбрать пункт меню **Сервис | Свойства папки** и в открывшемся окне на вкладке **Вид** снять флажок **Использовать простой общий доступ к файлам**, затем щелкнуть на кнопках **Применить** и **OK** (рис. 14.3).

Теперь опять перейдем к свойствам файла txt.txt. Вкладка **Безопасность**, как можно видеть, уже появилась. На этой вкладке щелкнем на кнопке **Добавить**, перейдя таким

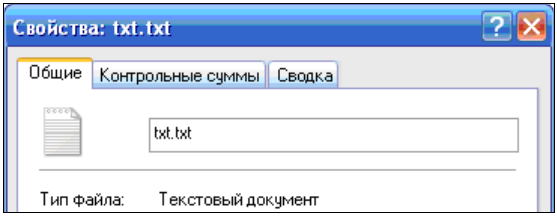


Рис. 14.2. Вкладки окна свойств файла

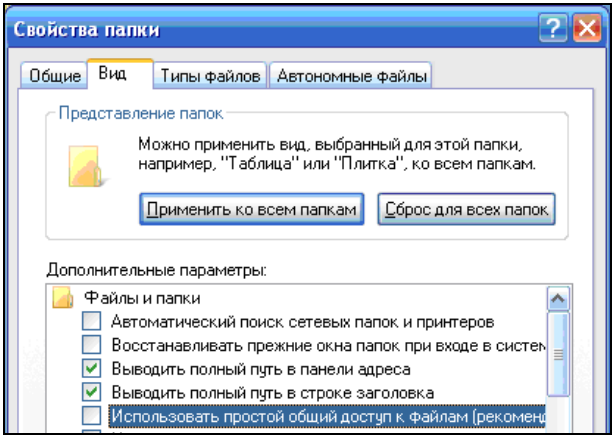


Рис. 14.3. Вкладки окна свойств папки

образом в окно **Выбор**. Далее нажмем кнопки **Дополнительно** и **Поиск** и, выбрав в открывшемся списке имя пользователя **ASP.NET**, щелкнем на кнопке **OK**. Затем на вкладке **Свойства** в строках **Чтение** и **Запись** установим флажки разрешения и нажмем кнопки **Применить** и **OK** (рис. 14.4).

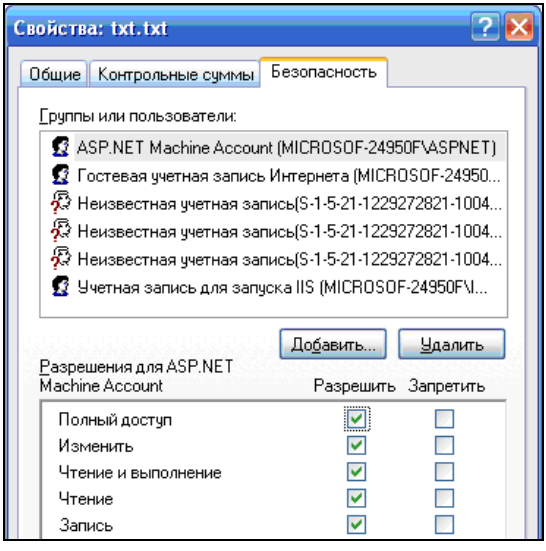


Рис. 14.4. Вкладка **Безопасность** окна свойств файла

Теперь еще раз запускаем веб-приложение и убеждаемся в его корректной работе. На рис. 14.5 показан пример работы созданной веб-страницы.

Используя инструмент чтения/записи текстового файла, можно обмениваться данными между страницами веб-приложения, организовать форум на веб-сайте, гостевую книгу, счетчик посещений и проч.

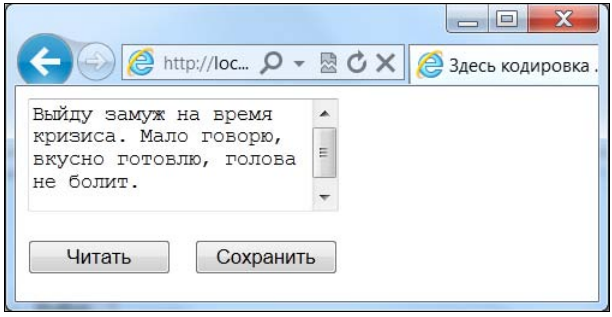


Рис. 14.5. Пример работы веб-страницы, осуществляющей чтение/запись текстового файла

Убедиться в работоспособности веб-страницы можно, открыв соответствующее решение в папке RW_txt сопровождающего книгу электронного архива.

Напоминание

Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 120. Программирование счетчика посещений сайта с использованием базы данных и объекта *Session*

Поставим задачу создать счетчик посещений 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, например, версии 2010, далее в меню **Файл** выберем команду **Создать | Новая база данных**, дадим файлу имя: `Web.mdb`. Затем создадим "шапку" новой таблицы в режиме конструктора. Имена полей приведены на рис. 14.6, типы всех полей — текстовые.

Назовем таблицу: **Таблица посещений веб-страницы**. Как можно видеть, 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 веб-страницы]
```

Теперь запустим Visual Studio 12, закажем новый проект из шаблона **ASP.NET Empty Web Application Visual C#**, назовем его `Counter`. К текущему проекту добавим веб-

форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Form**.

Здесь из панели **Toolbox** раздела **Standard** перенесем на веб-форму метку **Label**, куда будем выводить количество посещений данной страницы. Метку **Label** разместим пока в самом верху формы, но, в конце концов, ее следует разместить скромно где-нибудь в самом низу страницы.

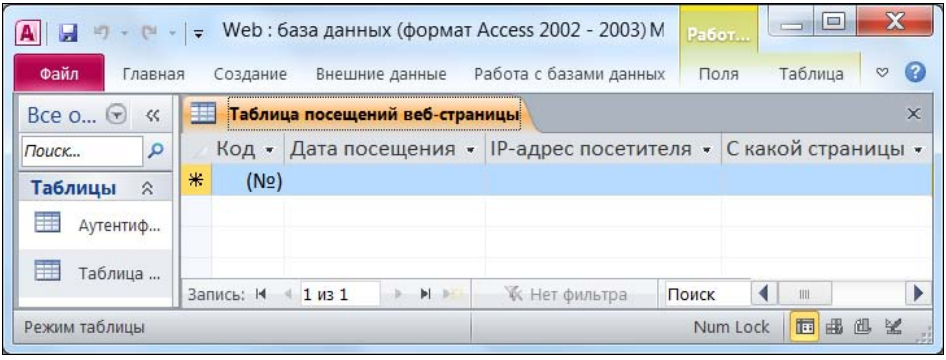


Рис. 14.6. Проектирование полей таблицы базы данных

Содержимое файла программной поддержки **WebForm1.aspx.cs** приведено в листинге 14.2.

Листинг 14.2. Счетчик посещений сайта

```
// Веб-приложение, реализующее счетчик посещений сайта с использованием
// базы данных и объекта Session
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе.
// Добавим директиву для краткости выражений:
using System.Data.OleDb;
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;
            Label1.Text = String.Empty;
            // При первой загрузке страницы выясняем IP-адрес посетителя
            var IP_адрес = Request.UserHostAddress;
            String URL_адрес;
            try
            {
                // Определение, с какой веб-страницы вы сюда пришли
                URL_адрес = Request.UrlReferrer.AbsoluteUri;
```

```
    Response.Write("<br /><br />Вы пришли на эту страницу "
                 + "со страницы " + URL_адрес);
}
catch // Ситуация As NothingReferenceException
{ // 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
var СтрокаПодкл =
  "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
                                    Server.MapPath("Web.mdb");
// Создание экземпляра объекта Connection
var Подключение = new OleDbConnection(СтрокаПодкл);
try
{
    // Открытие подключения
    Подключение.Open();
}
catch (Exception Ситуация)
{
    Response.Write("<br /><br />" + Ситуация.Message);
    Подключение.Close();
    return;
}
String SQL_запрос;
var Команда = new OleDbCommand();
// ЕСЛИ ОБ ЭТОМ ПОСЕЩЕНИИ ЕЩЕ НЕТ ЗАПИСИ
if (Page.Session["ЕСТЬ ЛИ ЗАПИСЬ ОБ ЭТОМ ПОСЕЩЕНИИ?"] == null)
{
    // ЕСЛИ НЕТ, ТО ДОБАВЛЯЕМ ЗАПИСЬ О ПОСЕЩЕНИИ В БД:
    var ДАТА = System.DateTime.Now.ToString();
    // Строка SQL-запроса:
    SQL_запрос =
       "INSERT INTO [Таблица посещений веб-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 веб-страницы]";
            Команда.CommandText = SQL_запрос;
            Команда.Connection = Подключение;
            // ExecuteScalar выполняет запрос и возвращает первую
            // колонку первого ряда таблицы запроса
            var КОЛ_ВО = 0;
            try
            {
                КОЛ_ВО = (int)Команда.ExecuteScalar();
            }
            catch (Exception Ситуация)
            {
                Response.Write("<br /><br />" + Ситуация.Message);
            }
            Подключение.Close();
            Label1.Text = String.Format(
                "Количество посещений страницы = {0}", КОЛ_ВО);
        }
    }
}
```

Как видно из программного кода, программирование счетчика посещений реализовано при обработке событий загрузки веб-страницы `Page_Load`, причем при ее первой загрузке, когда `isPostBack = false`. Используя объект `Request`, определяем IP-адрес посетителя веб-страницы. Далее с помощью этого же объекта `Request` в блоке `try...catch` определяем, с какой веб-страницы пользователь на эту страницу перешел. Если попытка (`try`) определить оказалась неудачной, то управление переходит на `Exception` и делается вывод, что пользователь оказался на этой странице, набрав ее адрес (URL) в адресной строке своего браузера. Для вывода текста в форму использован метод `Response.Write`. Роль этого метода 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 клавишу <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 веб-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 веб-приложения оказывается доступной на любой другой его странице. Даже если пользователь перешел на другое веб-приложение (на другой веб-узел), а затем вернулся в первоначальное приложение, то исходная коллекция данных сессии будет сохранена. Она будет сохранена до тех пор, пока пользователь не закрыл свой веб-браузер. Однако "время жизни" (лимит времени) сессии все же ограничено параметром `timeout`. По умолчанию такое "время жизни" равно 20 минутам, но 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 другой странице нашего веб-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 веб-приложения.

Вернемся к *добавлению* в БД записи о 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 это веб-приложение и затем вернется в него опять. (Но при этом не будет закрывать свой веб-браузер.) Поэтому в программном коде, прежде чем добавлять запись в базу данных о посещении пользователя, предусмотрена проверка с помощью коллекции объекта `Session`.

Пример работы счетчика посещений представлен на рис. 14.7.

Давайте проверим полученный IP-адрес. Является ли он действительно адресом нашего локального компьютера в Интернете? Для этого на панели задач своего компьютера

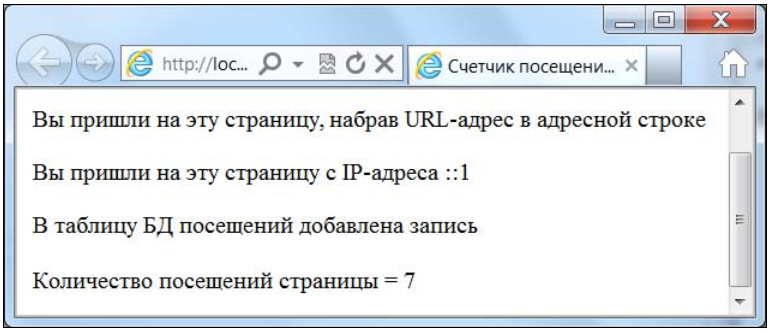


Рис. 14.7. Пример работы веб-страницы, подсчитывающей ее посещения

найдите значок **Подключение по локальной сети**. В контекстном меню укажите пункт **Состояние** и на вкладке **Поддержка** вы увидите свой правильный IP-адрес. Он будет отличаться от выведенного в форму. Почему? Дело в том, что пример работы нашей программы получен на *локальном сервере*, поэтому и IP-адрес мы получили как адрес из внутренней подсети. После того как проект веб-приложения будет скопирован на *удаленный сервер*, предоставляющий веб-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 веб-страницы можно, открыв решение Counter.sln в папке Counter сопровождающего книгу электронного архива.

Пример 121. Чтение/запись cookie-файлов

Cookie (по-английски — печенье) — это небольшой фрагмент данных (обычно до 4096 байт), созданный веб-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, для ведения статистики пользователей и проч. В самом начале использования cookie служили для определения, посещал ли пользователь сайт ранее.

Зайдя на любой сайт, вы можете увидеть активные на данном сайте cookie, набрав в адресной строке браузера следующее:

```
javascript:alert("Cookies: "+document.cookie)
```

В ответ ваш веб-браузер покажет вам примерно такое окно (рис. 14.8).

Рис. 14.8. Отображение активного cookie на некотором сайте

В этом окне содержатся те сведения, которые сервер некоторого веб-узла закодировал о вас в cookie-файле, записанном на ваш компьютер.

В этом примере мы научимся читать cookie при загрузке страницы и записывать cookie-файл при нажатии пользователем командной кнопки. То есть, постановка задачи следующая — веб-страница предлагает посетителю ввести данные о себе: его имя и род занятий. При нажатии кнопки **Запись Cookie**, введенные в текстовые поля сведения окажутся записаны в cookie-файл. Этот cookie-файл будет храниться на компьютере пользователя сутки. В течение этих суток, каждый раз вызывая нашу страницу, в текстовых полях мы будем видеть введенные нами сведения, которые мы можем тут же исправлять и опять записывать в cookie. Пример функционирования подобной веб-страницы приведен на рис. 14.9.

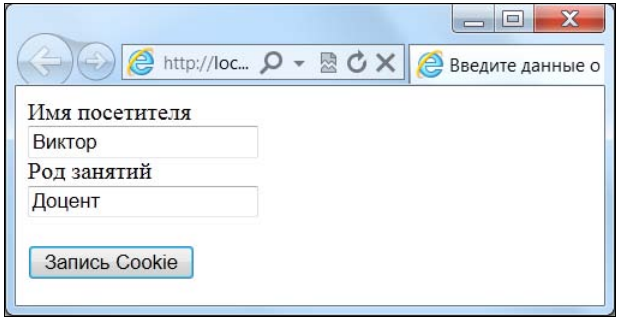


Рис. 14.9. Пример работы веб-страницы записи cookie

Для реализации поставленной задачи запустим Visual Studio 12, закажем новый проект из шаблона **ASP.NET Empty Web Application Visual C#**, назовем его `Cookie`. К текущему проекту добавим веб-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Form**.

Затем из панели **Toolbox** раздела **Standard** перенесем в проектируемую веб-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. Веб-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:
            // Dim CookieN As HttpCookieCollection
            HttpCookie Куки;
            // Читаю только один раздел cookie "О посетителе страницы"
            Куки = Request.Cookies.Get("О посетителе страницы");
            // Если на машине клиента нет такого cookie
            if (Куки == null) return;
            // А если есть, то заполняю текстовые поля из cookie
            TextBox1.Text = Куки["Имя посетителя"];
            TextBox2.Text = Куки["Род занятий посетителя"];
        }
        protected void Button1_Click(object sender, EventArgs e)
        {
            // ЗАПИСЬ COOKIE
            var Куки = new HttpCookie("О посетителе страницы");
            // Запись двух пар "имя (ключ) - значение"
            Куки["Имя посетителя"] = TextBox1.Text;
            Куки["Род занятий посетителя"] = TextBox2.Text;
            // Установка даты удаления cookie: сейчас плюс один день
            Куки.Expires = DateTime.Now.AddDays(1);
            // Добавление раздела "О посетителе страницы" в cookie-файл
            Response.Cookies.Add(Куки);
        }
    }
}
```

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 нашей программе мы предусмотрели только один раздел, хотя их может быть несколько (коллекция `HttpCookieCollection`). Если компьютер пользователя не содержит этого 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`, скопированное из текстовых полей. В предложении `Cookie.Expires` указываем срок хранения cookie. Если срок не указан, то cookies удаляются сразу, как только пользователь закроет браузер. Например, интернет-магазин может использовать cookie для хранения названий товаров, которые пользователь выбрал и поместил в виртуальную корзину покупок. Если срок хранения cookie указан, то, даже если пользователь закроет браузер, не совершив покупки, при последующем посещении интернет-магазина ему не придется формировать корзину заново.

Вернемся к нашему программному коду. Оператор `Add(Куки)` 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 cookie-файлы в различные папки. Скорее всего, вы найдете свой 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 этом примере cookie-файла будет:

```
администратор@localhost[2].txt
```

А раз cookie — это текстовый файл, его можем легко открыть, например, с помощью Блокнота (рис. 14.10).

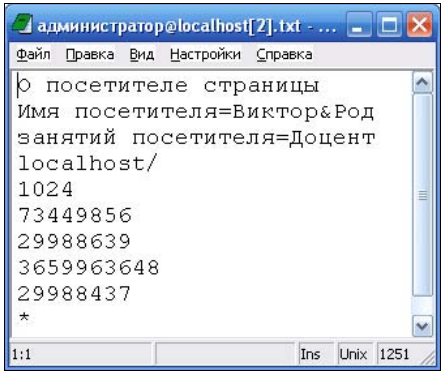


Рис. 14.10. Содержимое cookie-файла

Как можно видеть, первая строка содержимого cookie-файла — это имя раздела, затем идут пары "имя — значение", запрограммированные нами в коде, 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 демонизировать cookie не стоит.

Убедиться в работоспособности веб-страницы можно, открыв решение Cookie.sln в папке Cookie сопровождающего книгу электронного архива.

Пример 122. Вывод изображения в веб-форму

Размещение изображения в веб-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, с помощью `AlternateText` — альтернативный текст, отображаемый в элементе управления **Image**, когда изображение недоступно. Немножко усложним задачу и поставим ее в следующем виде — на странице имеется некоторое изображение, при щелчке мышью на котором оно увеличивается вдвое без перезагрузки веб-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 12 и создадим новый проект из шаблона **ASP.NET Empty Web Application Visual C#**, назовем его `ВебИзображение`. К текущему проекту добавим HTML-страницу. Для этого в меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **HTML Page**. Далее на вкладке **HtmlPage1.html** введем текст, представленный в листинге 14.4.

Листинг 14.4. Увеличение изображения в веб-форме (HTML-код)

```
<!DOCTYPE html>
<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 сгенерирует 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.html, например, в Internet Explorer. При этом мы увидим следующее (рис. 14.11).

Решим эту же задачу программированием на Visual Basic 12. Для этого добавим в текущий проект новую веб-форму. В меню **Project** выполним команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Form**. На вкладке конструктора формы перетащим из панели **Toolbox** (эту панель можно вызвать, например, используя комбинацию клавиш <Ctrl>+<Alt>+<X>) раздела **Standard** элемент управления изображения **Image**, а на вкладке программного кода **WebForm1.aspx.cs** напишем текст, представленный в листинге 14.5.

Рис. 14.11. Щелчок мышью увеличивает изображение

Листинг 14.5. Увеличение изображения в веб-форме (программный код на Visual C# 12)

```
// На странице имеем изображение — файл poryv.png, при щелчке мышью на
// нем изображение увеличивается вдвое без перезагрузки веб-страницы
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ВебИзображение
{
    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
            var 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 объекта `Image1`: `ImageUrl` и `AlternateText` — 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 веб-странице имеется изображение, например, мужчины. Это изображение мы используем для ссылки на другую веб-страницу, например на WebForm1.aspx. Причем при наведении на него указателя мыши происходит смена этого изображения на изображение женщины.

Для решения задачи добавим в текущий проект еще одну HTML-страницу и на вкладке разметки **HtmlPage2.html** напишем текст, приведенный в листинге 14.6.

Листинг 14.6. Смена изображения в веб-форме (HTML-код)

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Смена изображения при наведении</title>
</head>
<body>
    <a href="HTMLPage1.html">
      <img onmouseover="this.src='g.jpg'"
           onmouseout="this.src='m.jpg'"
      alt="Щелкните, чтобы перейти на HTMLPage1.html"
      src="m.jpg" border="0" /> </a>
</body>
</html>
```

Здесь тег `<a>` обеспечивает гиперссылку на другую веб-страницу — HTMLPage1.html, 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`).

Внешний вид данной веб-страницы в браузере показан на рис. 14.12.

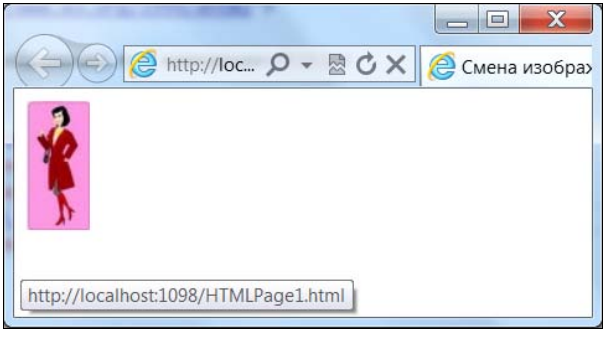


Рис. 14.12. Смена изображения при наведении указателя мыши

Для решения этой же задачи с помощью Visual Studio 12 в текущий проект добавим новую веб-форму — WebForm2.aspx. В конструкторе формы добавим из панели

Toolbox раздела **Standard** 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 веб-форме (программный код на Visual C# 12)

```
// На веб-странице имеем изображение, например, мужчины — файл m.jpg. Это
// изображение используем для ссылки на другую веб-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 ВебИзображение
{
    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 веб-страниц, рассмотренных в этом примере, можно, открыв соответствующее решение в папке ВебИзображение сопровождающего книгу электронного архива.

Пример 123. Формирование изображения методами класса *Graphics* и вывод его в веб-форму

При создании веб-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 вывес-

ти его клиенту в обозреватель. Таким графиком может быть, скажем, график статистики посещений сайта по месяцам, по неделям, по дням.

Задача, решаемая в этом примере, состоит в следующем. Во время загрузки веб-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. Затем сохранить рисунок в текущий каталог сервера и вывести его на веб-страницу.

Для решения этой задачи запустим Visual Studio 12, закажем новый проект шаблона **ASP.NET Empty Web Application Visual C#**, в поле имени проекта напишем `ТекстНаклWeb`. Добавим веб-форму, для этого в меню **Project** выберем команду **Add New Item**, в открывшемся окне выберем 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 веб-форму

```
// Веб-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;
// Другие директивы using удалены, поскольку они не используются в данной
// программе.
// Добавим эту директиву для более кратких выражений:
using System.Drawing;
namespace ТекстНаклWeb
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Page.Title = "Вывод даты";
            // Создаем точечное изображение размером 230x40 точек
            // с глубиной цвета 24
            var Рисунок = new Bitmap(230, 40,
          System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            // Создаем новый объект класса Graphics из изображения Рисунок
            var 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:
            var Дата = String.Format("Сегодня {0:D}", DateTime.Now);
            // Разворачиваем мир на 356 градусов по часовой стрелке.
            // То есть ось X направлена влево, ось Y — вниз, а углы
            // отсчитываются от оси X в сторону Y.
```

```
            Графика.RotateTransform(356.0F);
            var Фонт = new Font("Times New Roman", 14,
                                        FontStyle.Regular);
            // Выводим на изображение текстовую строку Дата,
            // x=5, y=15 — координаты левого верхнего угла строки
            Графика.DrawString(Дата, Фонт, Brushes.Red, 5, 15);
            // Определяем физический путь файла для текущего веб-узла,
            // сохраняем изображение в файле risunok.jpg каталога веб-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();
        }
    }
}
```

Как видно из программного кода, при загрузке страницы создаем точечное изображение указанного размера, формат `Format24bppRgb` указывает, что отводится 24 бита на точку: по 8 битов на красный, зеленый и синий каналы. Это 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 к файлу изображения, методом `Save` сохраняем его в каталоге веб-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 пример работы программы.

Убедиться в работоспособности веб-страницы, рассмотренной в этом примере, можно, открыв соответствующее решение в папке ТекстНаклWeb сопровождающего книгу электронного архива.

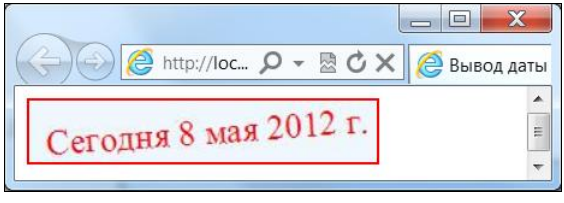


Рис. 14.13. Вывод в форму изображения, сформированного программно

Пример 124. 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 непосредственно для сообщения. Все записи, оставленные посетителями сайта, будем сохранять в текстовом файле kniga.txt на *винчестере сервера*, предоставляющего веб-хостинг. Впрочем, вместо текстового файла можно воспользоваться какой-либо базой данных.

Для программирования гостевой книги запустим Visual Studio 12, в окне **New Project** выберем шаблон **ASP.NET Empty Web Application Visual C#**, в поле **Name** укажем имя нового решения `ГостеваяКнига` и щелкнем на кнопке **OK**. Теперь добавим к текущему проекту веб-форму. Для этого в пункте меню **Project** выберем команду **Add New Item**, в открывшемся 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** 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 станем 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 заполнение 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.

Листинг 14.9. Гостевая книга

```
// Данная веб-страница приглашает посетителя оставить какие-либо записи,
// которые могут прочитать другие посетители страницы. Записи сохраняются
// в текстовом файле kniga.txt. Записи отображаются на веб-странице с
// помощью элемента "сетка 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.ControlToValidate = "TextBox1";
            RequiredFieldValidator1.EnableClientScript = false;
            RequiredFieldValidator1.ErrorMessage =
                          "* Следует заполнить это текстовое поле";

            RequiredFieldValidator2.ControlToValidate = "TextBox2";
            RequiredFieldValidator2.EnableClientScript = false;
            RequiredFieldValidator2.ErrorMessage =
                          "* Следует заполнить это текстовое поле";

            RequiredFieldValidator3.ControlToValidate = "TextBox3";
            RequiredFieldValidator3.EnableClientScript = false;
            RequiredFieldValidator3.ErrorMessage =
                          "* Следует заполнить это текстовое поле";

            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[] Сепаратор = { '\t' }; // — массив из одного символа
    while (Читатель.EndOfStream == false)
    {
        var Одна_строка = Читатель.ReadLine();
        // Функция Split делит строку на четыре части и присваивает
        // каждую часть элементам массива
        var Массив_частей_строки = Одна_строка.Split(Сепаратор);
        // 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"
    // Если проверка текстовых полей показала, что они заполнены
    // верно,
    if (Page.IsValid == false) return;
    // то 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();
        }
    }
}
```

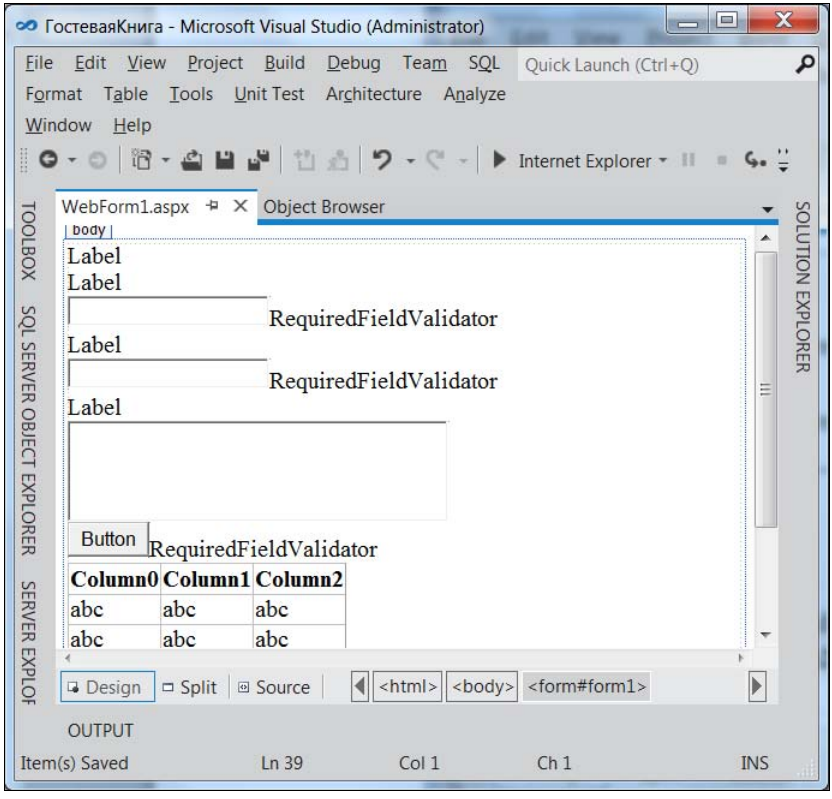


Рис. 14.14. Размещение элементов управления в конструкторе формы

Как видно из программного кода, вначале создаем таблицу данных класса `DataTable` и два потока данных для чтения и для 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()`.

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 веб-страницы, его e-mail и непосредственно его сообщение. Причем в качестве разделителя между этими текстовыми строками мы используем невидимый 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 два раза*: один раз при загрузке формы, а другой при добавлении новой записи.

Пример работы гостевой книги представлен на рис. 14.15.

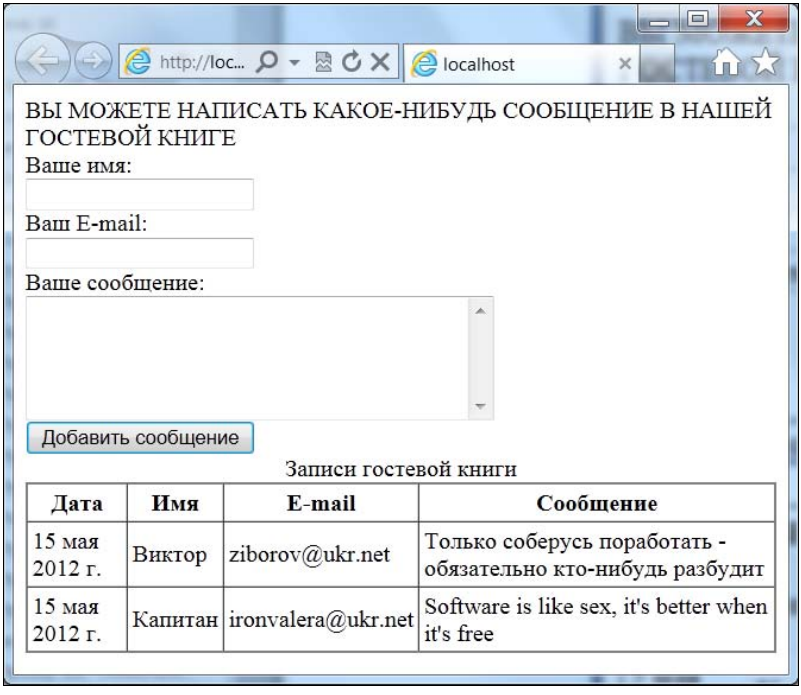


Рис. 14.15. Интерфейс гостевой книги

Убедиться в работоспособности данной программы можно, открыв соответствующее решение в папке ГостеваяКнига сопровождающего книгу электронного архива.

Пример 125. Отображение времени в веб-форме с использованием технологии AJAX

Мы уже обсуждали вопрос отображения времени в Windows-приложении *(см. главу 12, пример 101. Время по Гринвичу в полупрозрачной форме)*. 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 секунде) элемента управления **Timer** в метку **Label** копировалось новое значение времени, при этом изображение формы обновлялось (перерисовывалось). Когда речь идет о веб-форме, технология ASP.NET предполагает операции обратной отправки данных — если пользователь щелкает на кнопке или делает выбор в окне списка с включенной функцией `AutoPostBack`, серверу отсылается соответствующий запрос, после чего с него обратно клиенту отправляется целая страница. Если такое обращение к серверу будет происходить каждую секунду, то возрастет объем сетевого трафика, и вследствие инерции Сети пользователь начнет 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 у нас есть метка **Label** в веб-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 12 и закажем новый проект из шаблона **ASP.NET Empty Web Application Visual C#**, укажем имя: `AjaxTimer`. К текущему проекту добавим веб-форму. Для этого в меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Form**.

Далее, попав на вкладку конструктора веб-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 14.10.

Листинг 14.10. Отображение времени в веб-форме

```
// Веб-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; // миллисекунд = 1 сек
            Label1.Text = "Текущее время: " +
                                DateTime.Now.ToLongTimeString();
        }
        protected void Timer1_Tick(object sender, EventArgs e)
```

```
        {
            // Обновление содержимого метки каждую секунду времени:
            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 веб-форма, а только те элементы, которые расположены на элементе **UpdatePanel**.

Пример работы программы показан на рис. 14.16.

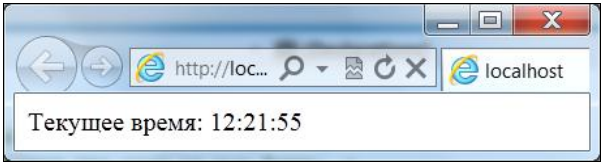


Рис. 14.16. В форме обновляется только AJAX-элемент **UpdatePanel**

Убедиться в работоспособности веб-страницы можно, открыв решение AjaxTimer.sln в папке AjaxTimer сопровождающего книгу электронного архива.

ГЛАВА 15

Создание веб-служб и их клиентов

О веб-службах

Веб-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 (пользоваться сервисом веб-службы, *потреблять* его) со своего клиентского компьютера. При этом возможности удаленной веб-службы будут реализованы в вашем Windows- или веб-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 веб-служб и результатов их выполнения используется протокол SOAP (Simple Object Access Protocol). Протокол SOAP применяет формат сообщений, основанный на XML. Веб-служба и протокол SOAP не зависят от конкретной платформы. Поэтому различные разработчики могут использовать веб-службы друг друга, не беспокоясь о совместимости операционных систем, технологий или языков программирования. Другое название веб-службы — веб-сервис.

Веб-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 веб-приложение путем создания клиентского приложения, потребляющего такой уже существующий сервис веб-службы.

Для поиска веб-служб в Интернете существует несколько сайтов, в которых веб-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 веб-служб. В Сети можно найти и другие списки общедоступных веб-служб различных производителей, например: **http://www.xmethods.org/ve2/index.po**. Список русскоязычных веб-служб можно найти по адресу: **http://ivbeg.bestpersons.ru/feed/post3279396/**.

Упрощенно говоря, веб-служба в среде .NET состоит из двух компонентов: ASMX-файла и файла программной поддержки ASMX.cs. ASMX-файл содержит в себе ин-

формацию о методах веб-службы, о способах их тестирования, это можно посмотреть в любом веб-браузере. Файл программной поддержки написан на языке C# (или Visual Basic) и может быть скомпилирован для получения библиотеки DLL. Именно файл с расширением dll обеспечивает реализацию методов веб-службы.

Начальная страница наиболее популярного сайта **http://www.webservicex.net**, предоставляющего веб-сервис, показана на рис. 15.1.

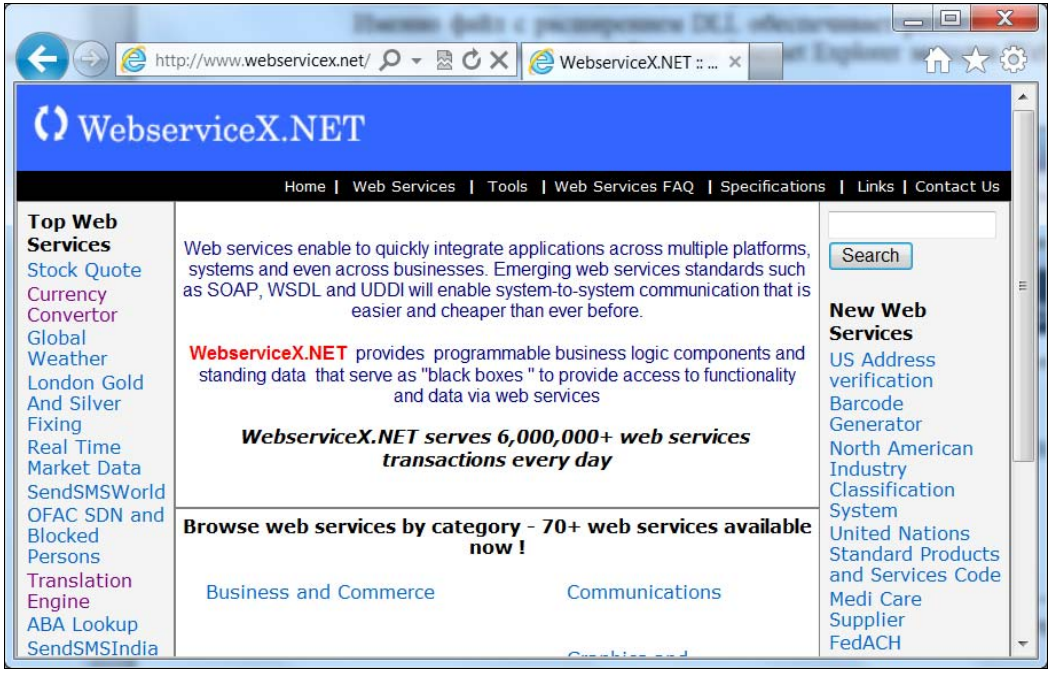


Рис. 15.1. Начальная страница сайта, предоставляющего множество различных веб-сервисов

Как можно видеть, сайт предоставляет множество различных сервисов: это и конвертер валют, погода в любой точке планеты, переводчик на различные языки, проверка адреса и проч.

Потребители подобных веб-сервисов в своем создаваемом клиентском приложении делают ссылку на конкретные классы сервисов и таким образом включают в текущий проект удаленный класс. А затем в своем программном коде объявляют создание нового экземпляра этого класса и потом уже обращаются ко всем методам класса удаленного веб-сервиса.

Пример 126. Клиентское веб-приложение, потребляющее сервис веб-службы "Прогноз погоды"

А теперь продемонстрируем, как конкретно можно использовать удаленный веб-сервис. В этом примере создадим клиентское веб-приложение, потребляющее сервис

веб-службы сайта **http://www.webservicex.net/globalweather.asmx**, и продемонстрируем тем самым, что это — просто! Следует всего лишь в создаваемом нами Windows- или веб-приложении указать ссылку (**Add Web Reference**) на удаленный файл (который может находиться за тысячи километров от нас) с расширением *.asmx. Если веб-сервис был создан с использованием технологии (по шаблону) Web Service, то этот удаленный файл будет иметь расширение *.asmx. Если же была использована другая технология — WCF Service, то соответствующий файл будет иметь расширение *.svc. Веб-сервисов, построенных по шаблону Web Service, — больше, а построенные по технологии WCF Service являются более скоростными, хотя при их использовании разности в скорости вы не ощутите.

Веб-служба, обсуждаемая в этом примере, возвращает прогноз погоды в городе, который пользователь задает в запросе к службе. Эта веб-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 веб-служба готова сделать прогноз погоды.

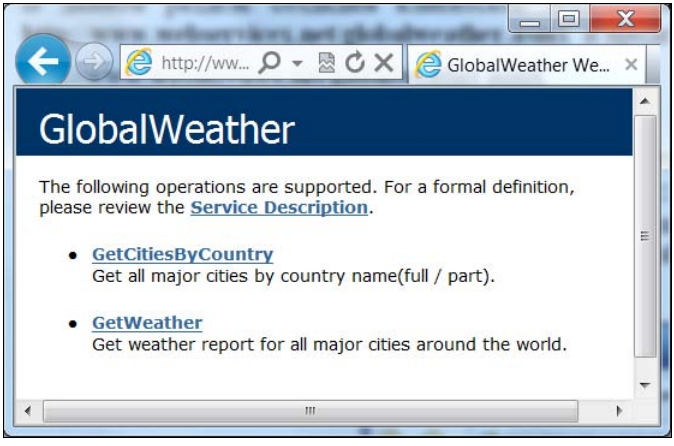


Рис. 15.2. Два метода веб-службы прогноза погоды

На вход второй функции `GetWeather` подают названия города и страны, а на выходе функции получают XML-строку, содержащую прогнозируемые параметры погоды. Давайте щелкнем на второй ссылке и подадим на вход функции `GetWeather`, названия города (Киев) и страны (Украина), см. рис. 15.3.

Щелкнув на кнопке **Invoke** (Запросить), мы получим сгенерированный веб-сервисом XML-код в нашем клиентском браузере (рис. 15.4).

Как видно из рис. 15.4, параметры погоды указываются в XML-элементах: температура (`Temperature`), относительная влажность (`RelativeHumidity`), давление (`Pressure`) и проч. в Киеве 12 апреля 2012 года на 16:30 по Гринвичу (UTC).

Напишем веб-приложение (также легко можно написать и Windows-приложение), обращающееся к функции `GetWeather` этой удаленной веб-службы. Причем программный код должен быть минимальным и демонстрировать лишь принцип подключения и доступ к нужному элементу (например, температуре) XML-документа.

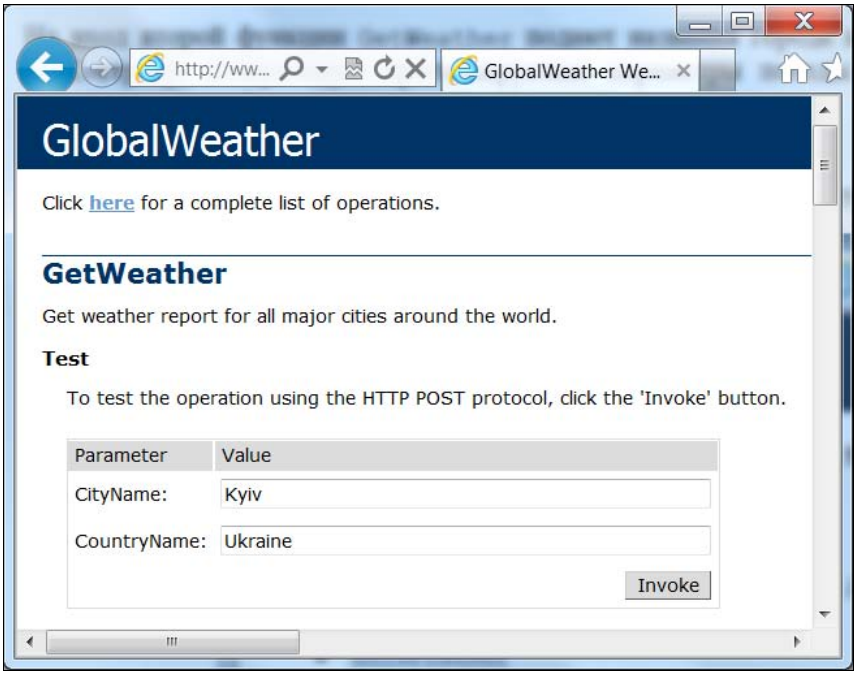


Рис. 15.3. Подаем на вход функции `GetWeather` название города и страны

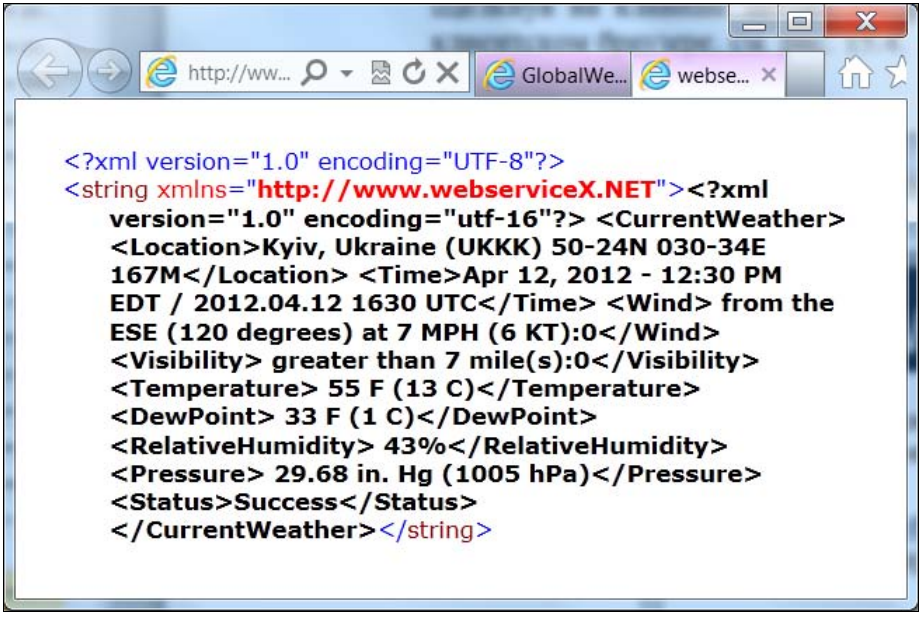


Рис. 15.4. В наш браузер веб-сервис вернул данный XML-код

Для этой цели после запуска Visual Studio 12 выберем проект шаблона **ASP.NET Empty Web Application Visual C#**, укажем имя **Name**: `ВебКлиентПогода`. К текущему проекту добавим веб-форму. Для этого в меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Form**. Далее, попав в конструктор веб-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 веб-службу, содержащую класс с необходимыми методами. Для этого в меню **Project** выберем команду **Add Web Reference**. Такого пункта может в меню **Project** не быть, в этом случае выполняем команду меню **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. 15.5, — здесь в поле **URL** введем адрес веб-службы: **http://www.webservicex.net/globalweather.asmx** и щелкнем на кнопке **Go**.

После этого в окне под полем **URL** мы увидим оба метода данного сервиса в том виде, как это было показано на рис. 15.2 (после нажатия на кнопку **Go**). При этом станет дос-

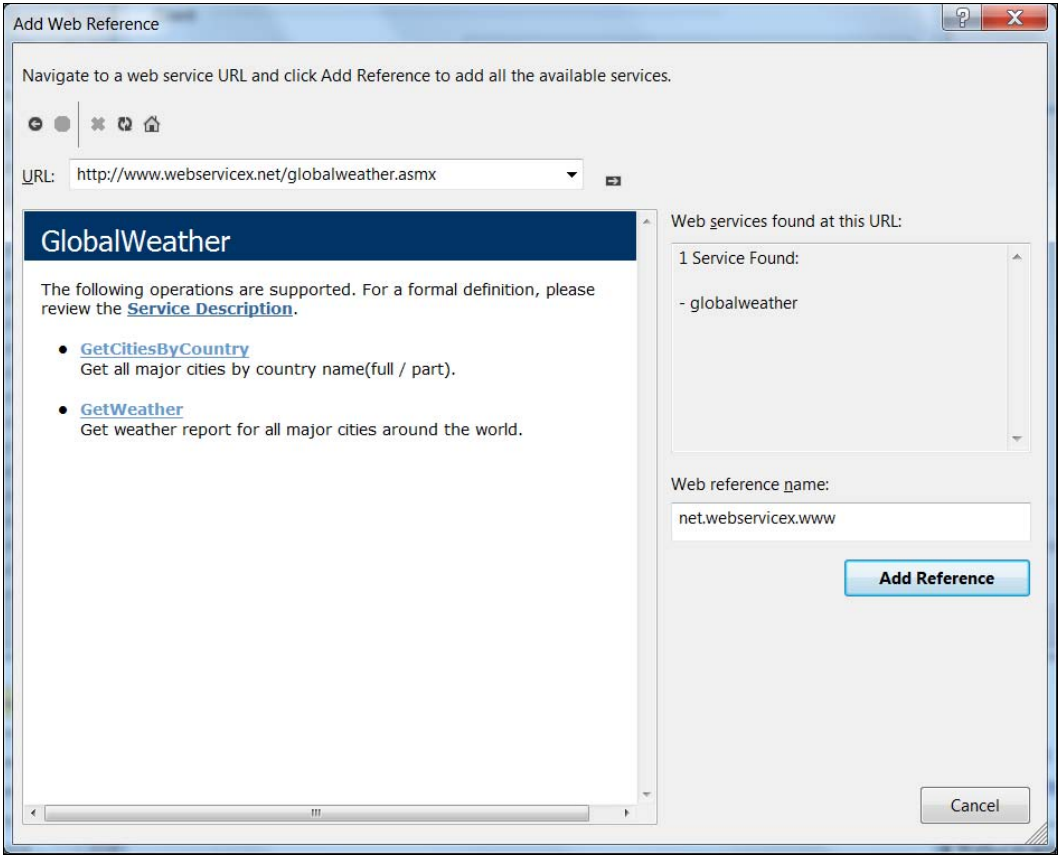


Рис. 15.5. Добавление удаленного сервиса к нашему проекту

тупной кнопка **Add Reference**. Щелкнем на этой кнопке и увидим, что в окне **Solution Explorer** появился новый узел **Web References** со ссылкой на удаленный класс **net.webservicex.www** (рис. 15.6).

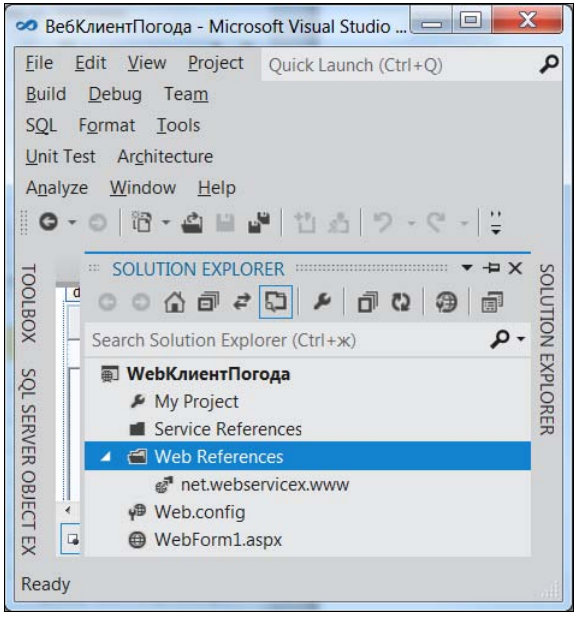


Рис. 15.6. В окне **Solution Explorer** появилась ссылка на удаленный класс

Теперь этот класс, содержащий необходимую нам удаленную веб-службу, мы можем использовать в своем программном коде (листинг 15.1).

Листинг 15.1. Веб-приложение, обращающееся к сервису удаленной веб-службы прогноза погоды

```
// Веб-приложение, потребляющее сервис удаленной веб-службы прогноза
// погоды. Приложение в текстовом поле TextBox демонстрирует XML-строку
// с параметрами погоды для города, указанного во входных параметрах
// при обращении к веб-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 ВебКлиентПогода
{
    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 веб-службы:
            // http://www.webservicex.net/globalweather.asmx
            // Эта веб-служба часто бывает перегружена 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(@"D:\Погода.xml");
            // var Строка_XML = Reader.ReadToEnd();
            // Reader.Close();
            // Функция GetWeather запрашивает строковые параметры с названием
            // города и страны и возвращает строку с XML-документом:
            var Строка_XML = ПОГОДА.GetWeather("Moscow", "Russia");
            // Можно тут же сохранить XML-строку в файл для дальнейшей
            // отладки программы:
            // System.IO.File.WriteAllText(@"D:\Погода.xml", Строка_XML);
            // Какая погода в Киеве:
            // var 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);
            var Имя = String.Empty;
            var Значение = String.Empty;
            // Цикл по узлам XML-документа:
            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 можно видеть, в программном коде при обработке события "щелчок на кнопке **Выяснить погоду**" создается экземпляр класса удаленной веб-службы. Далее происходит непосредственное обращение к методу класса `GetWeather` с входными параметрами `город` и `страна`, 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`.

Пример работы программы показан на рис. 15.7.

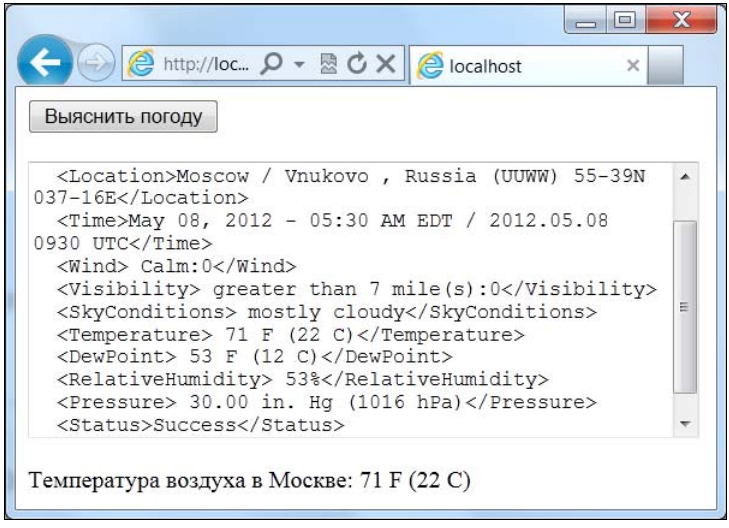


Рис. 15.7. Работа клиента веб-службы "Прогноз погоды"

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 — обращение к удаленному классу, предоставленному веб-службой.

Таким образом, мы продемонстрировали создание клиентского веб-приложения, потребляющего сервис веб-службы. Замечу, что аналогичным образом можно создавать и Windows-приложения (настольные приложения), которые также, будучи клиентами какой-либо веб-службы, могут получать, например, справочную информацию в онлайновом режиме.

Убедиться в работоспособности данной программы можно, открыв соответствующий файл решения в папке ВебКлиентПогода сопровождающего книгу электронного архива.

Напоминание

Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 127. Клиентское Windows-приложение, использующее ту же веб-службу "Прогноз погоды"

Теперь создадим клиентское Windows-приложение, потребляющее сервис той же веб-службы "Прогноз погоды", и продемонстрируем тем самым, что обращаться к удаленному классу можно не только из веб-приложения, но и из Windows-приложения (т. е. из настольного приложения).

Для решения этой задачи запустим Visual Studio 12, выберем шаблон **Windows Forms Application Visual 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 веб-службы. Для этого выберем в меню **Project** команду **Add Service Reference | Advanced | Add Web Reference** — это приведет к появлению одноименного диалогового окна **Add Web Reference**. В этом окне в поле **URL** введем тот же адрес веб-службы, что и в *примере 126*: **http://www.webservicex.net/globalweather.asmx** и нажмем кнопку **Go**. В окне ниже увидим список доступных методов данного сервиса, после чего 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 15.2).

Листинг 15.2. Windows-приложение, использующее сервис удаленной веб-службы прогноза погоды

```
// Windows-приложение, потребляющее сервис удаленной веб-службы прогноза
// погоды. Приложение в текстовом поле TextBox демонстрирует XML-строку
// с параметрами погоды для города, указанного во входных параметрах при
// обращении к веб-службе. Также выводит в текстовую метку значение
// температуры в этом городе
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Добавляем внешнюю ссылку на веб-службу таким образом:
// Проект — Добавить ссылку на службу — Дополнительно -
// Добавить веб-ссылку http://www.webservicex.net/globalweather.asmx
namespace WindowsКлиентПогода
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
```

```
        {
            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(@"D:\Погода.xml");
            // var Строка_XML = Reader.ReadToEnd();
            // Reader.Close();
            // Функция GetWeather запрашивает строковые параметры с названием
            // города и страны и возвращает строку с XML-документом:
            var Строка_XML = ПОГОДА.GetWeather("Moscow", "Russia");
            // Какая погода в Киеве:
            // var Строка_XML = ПОГОДА.GetWeather("Kyiv", "Ukraine")
            textBox1.Text = Строка_XML;
            // Здесь считывание значения узла из XML-строки выполнено более
            // эффективно:
            var XML_элемент = System.Xml.Linq.XElement.Parse(Строка_XML);
            var Значение = XML_элемент.Element("Temperature").Value;
            label1.Text = "Температура воздуха в Москве: " + Значение;
        }
    }
}
```

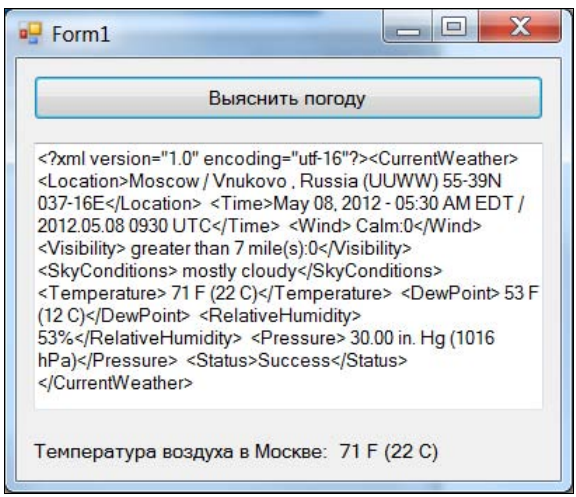


Рис. 15.8. Работа Windows-клиента веб-службы "Прогноз погоды"

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 выяснить прогноз погоды. Этот метод возвращает строку XML. Эту XML-строку мы выводим в текстовое поле `TextBox1`, так же, как мы это делали в веб-приложении, однако извлекаем из узла "`Temperature`" прогнозируемое значение температуры более эффективным образом, используя методы класса `Linq`.

Пример работы программы представлен на рис. 15.8.

Убедиться в работоспособности программы можно, открыв решение в папке WindowsКлиентПогода сопровождающего книгу электронного архива.

Пример 128. Создание простейшей веб-службы

Полагаю, мы убедились, что создать клиентское приложение, потребляющее сервис веб-службы, не сложно. Теперь поставим задачу создать самую простую веб-службу, чтобы убедиться, что это тоже простая задача. Автор понимает, что сейчас наступил очень деликатный момент, — если мы начнем рассматривать какой-либо сложный пример, то можем навсегда отбить интерес у читателя разобраться с принципами работы веб-служб. Поэтому задачу, решаемую данной веб-службой, ставим максимально простой. Например, пользователю предлагается ввести два числа, а веб-служба берет на себя функцию сложения этих двух чисел и вывода (возврата) суммы. При этом необходимо произвести диагностику вводимых данных. От такой веб-службы нет особенной пользы, но на этой маленькой задачке нам будет удобно продемонстрировать все возможности. Здесь, так же как и при отладке активных веб-страниц *(см. главы 13 и 14)*, при отладке нашей веб-службы в качестве удаленного серверного компьютера и клиентского компьютера мы будем использовать один ваш локальный компьютер.

Для решения этой задачи запускаем Visual Studio 12, выбираем шаблон **ASP.NET Empty Web Application**, даем проекту имя — например: `ВебСлужбаСумма`. Теперь нам нужно добавить в пустой проект шаблон **Web Service**, который добавит нам файл ASMX, являющийся "ключевым" в этом сервисе. Однако мы не обнаружим названный шаблон для .NET Framework 4.5. Вместо этого шаблона Microsoft предлагает нам шаблон **WCF Service** (здесь файлы с расширением svc).

> ***Примечание***
>
> WCF-службы рассмотрены в самом конце этой главы *(примеры 135 и 136)*, поскольку мы знаем, что такие сервисы *менее* популярны. Вероятно, Microsoft хочет изменить эту тенденцию и настойчиво предлагает перейти на веб-службы технологии WCF Service. Однако мы не станем спешить, а будем переходить на технологию WCF Service *"медленно"*, имея в виду неудавшийся проект Windows Vista, который многих разочаровал. Поэтому в данном примере мы все-таки воспользуемся шаблоном **Web Service** из .NET Framework 4.0.

Итак, выполним команду меню **Project | ВебСлужбаСумма Properties**, здесь на вкладке **Application** в раскрывающемся списке вместо ссылки на **.NET Framework 4.5** щелкнем ссылку на **.NET Framework 4**. Теперь нужный нам шаблон **Web Service** станет доступным.

Чтобы подключить его в текущий проект, как обычно, в меню **Project** выбираем команду **Add New Item**, в открывшемся окне укажем шаблон **Web Service** и щелкнем

на кнопке **Add**. Сразу после этого попадаем на вкладку **WebService1.asmx.cs** готовой веб-службы **Hello World**, и уже ее можно тестировать. Приводим содержимое этой вкладки в листинге 15.3.

Листинг 15.3. Файл программной поддержки веб-службы Hello World

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

namespace ВебСлужбаСумма
{
    /// <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 **Hello World** — это другая задача, разработчики Visual Studio 12 включили ее для облегчения программистам освоения данной технологии. Нас интересует именно наша задача, поэтому изменим файл программной поддержки, как показано в листинге 15.4.

Листинг 15.4. Простейшая веб-служба

```
// На вход данной веб-службы предлагается ввести два числа, а веб-служба
// берет на себя функцию сложения этих двух чисел и вывода (возврата)
// суммы. При этом веб-служба производит диагностику вводимых данных
// ~ ~ ~ ~ ~ ~ ~ ~
using System;
using System.Web.Services;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
```

```
namespace ВебСлужбаСумма
{
    /// <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, их анализировать, и при
            // "плохом вводе" сообщать по-русски.
            Single X, Y;
            var 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";

            var 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 веб-служба — это удаленный класс, к которому можно подключиться через Интернет из своего, например, Windows- или веб-приложения. На вход функции `Сумма` пользователь веб-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, происходит преобразование строковых переменных в переменные типа `Single`.

Теперь можем протестировать данную веб-службу, для этого нажмем клавишу <F5>. При этом веб-браузер откроет ASMX-файл. Здесь мы увидим единственный метод этой

веб-службы — `Сумма`. Щелчок указателем мыши на этой гиперссылке обеспечит возможность тестирования нашей веб-службы (рис. 15.9).

Попробуйте вводить в 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 веб-службы.

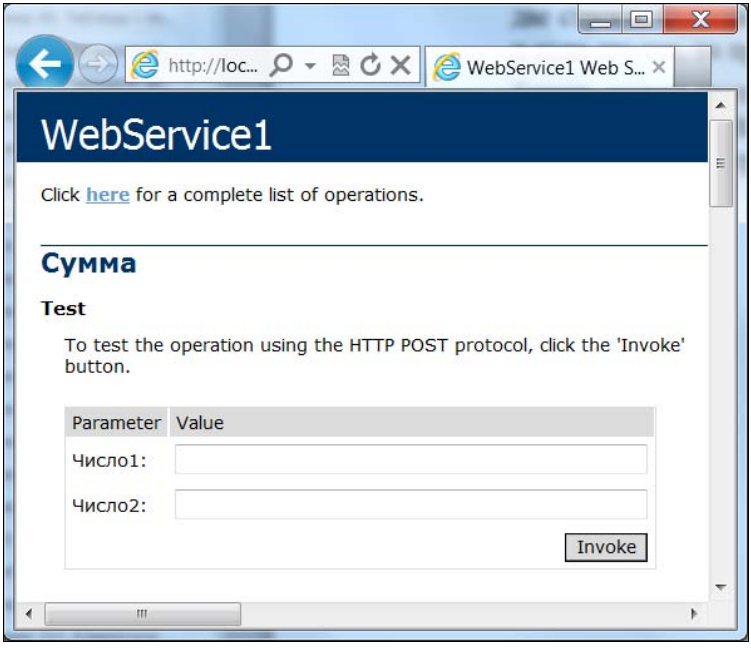


Рис. 15.9. Тестирование созданной веб-службы

Убедиться в работоспособности созданной веб-службы можно, открыв соответствующий файл решения в папке ВебСлужбаСумма сопровождающего книгу электронного архива.

Пример 129. Создание Windows-приложения, потребителя сервиса веб-службы

Теперь напишем Windows-приложение (т. е. создадим EXE-файл), который будет обращаться к веб-службе, созданной в *примере 128*. Такое приложение можно называть *клиентским* или клиентом службы. Для этого запустим Visual Studio 12, выберем шаблон **Windows Forms Application Visual C#** (можно даже **Console Application**), укажем имя проекта: `ВебКлиентСумма`. Из панели **Toolbox** перенесем в форму командную кнопку **Button**. Запланируем, что вся работа с удаленной веб-службой будет происходить при обработке события "щелчок на кнопке **Пуск**".

Чтобы сделать Windows-приложение потребителем сервиса веб-службы, необходимо в его проекте создать веб-ссылку на удаленный класс. Для этого выберем в 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 щелкнем кнопку **Advanced**, а затем, в следующем окне, кнопку **Add Web Reference**.

Поскольку мы отлаживаем и веб-службу, и его клиентское приложение на одном локальном компьютере, в поле URL пишем виртуальный адрес ASMX-файла. Этот адрес мы можем получить, запустив нашу веб-службу из *примера 128*, а затем скопировав в буфер обмена его URL-адрес. Для локального компьютера, на котором автор отлаживал данный пример, URL-адрес был таким: **http://localhost:2961/WebService1.asmx**.

Затем щелкаем на кнопке **Go** и тем самым получаем доступ к кнопке **Add Reference**, после щелчка на которой в окне **Solution Explorer** добавится значок ссылки на класс `localhost`, обеспечивающий доступ к веб-службе (рис. 15.10).

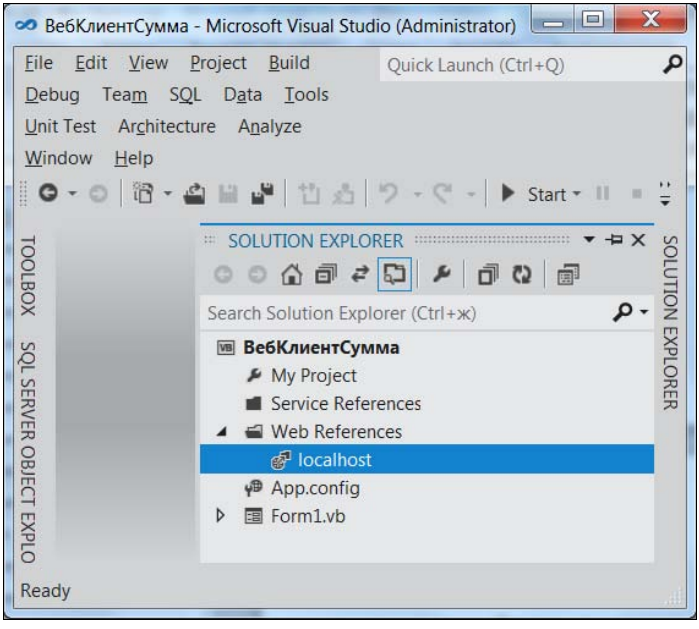


Рис. 15.10. Добавление ссылки на класс `localhost`

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 15.5).

Листинг 15.5. Windows-приложение, потребляющее сервис веб-службы

```
// Клиентское Windows-приложение, потребляющее сервис веб-службы
// предыдущего примера WebСлужбаСумма
using System;
using System.Windows.Forms;
```

```
// Другие директивы using удалены, поскольку они не используются в данной
// программе
// ~ ~ ~ ~ ~ ~ ~ ~
// Чтобы добавить веб-службу к обычному Windows-приложению:
// Project | Add Service Reference | Advanced | Add Web Reference,
// затем в поле URL пишем виртуальный адрес веб-службы
// http://localhost:3002/WebService1.asmx
namespace ВебКлиентСумма
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            button1.Text = "Пуск";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Создаем экземпляр удаленного класса:
            var Удаленный = new localhost.WebService1();
            var 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. 15.11.

Убедиться в работоспособности программы можно, открыв соответствующий файл решения SLN в папке ВебКлиентСумма сопровождающего книгу электронного архива.

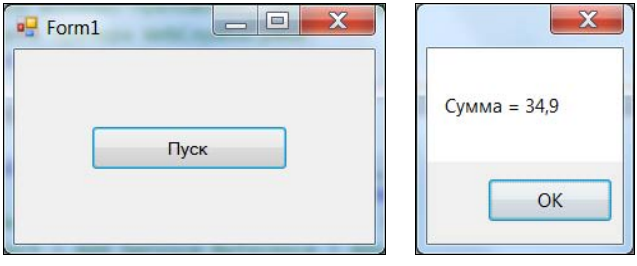


Рис. 15.11. Обращение Windows-приложения к веб-службе

Пример 130. Создание веб-службы "Торговая рекомендация на рынке Forex"

Создадим веб-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.

Пусть 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 веб-службу, которая с помощью синтаксического разбора указанной веб-страницы будет извлекать торговую стратегию, рекомендованную этим 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 задачи запускаем Visual Studio 12, в окне **New Project** щелкнем раскрывающееся меню **.NET Framework 4.5** и выберем другую платформу — **.NET Framework 4**. Далее, указав шаблон **ASP.NET Empty Web Application Visual C#**, введем имя нового проекта: `ВебСлужбаForex`. Затем в меню **Project** выберем пункт **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **Web Service**. Этот шаблон стал доступен благодаря тому, что мы выбрали платформу **.NET Framework 4**. Мы получим простейшую веб-службу **Hello World**, которая по умолчанию запрограммирована в шаблоне **Web Service**. Однако мы ее изменим согласно условию нашей задачи. Содержимое файла программной поддержки приведено в листинге 15.6.

Листинг 15.6. Веб-служба "Торговая рекомендация на рынке Forex"

```
// Веб-служба, которая с помощью синтаксического разбора веб-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 веб-службы в виде строки
// ~ ~ ~ ~ ~ ~ ~ ~
using System;
using System.Web.Services;
// Другие директивы using удалены, поскольку они не используются в данной программе
namespace ВебСлужбаForex
{
    /// <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 веб-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;
            var СТРОКА = "Строка";
            try
            {
                // Попытка открытия веб-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 веб-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:
            var i = СТРОКА.IndexOf("Торговая стратегия:");
            // Ищем стратегию только для EURUSD:
            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 EURUSD на " +
                     DateTime.Now.ToLongDateString() + ":" + "\r\n");
            ПОТОК.Close();
            return СТРОКА;
        }
    }
}
```

Как видно из программного кода, наша веб-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 веб-страницы. Вообще говоря, веб-страница может быть записана на машинном носителе в какой-либо кодировке. Чаще всего это Unicode (UTF-8), русскоязычные же сайты часто имеют кодировку Windows 1251. Чтобы выяснить, в какой кодировке записана веб-страница, следует в браузере в контекстном меню навести указатель мыши на пункт меню **Кодировка**.

При чтении веб-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 разметке страницы раздел с торговой стратегией для EUR–USD с помощью строковых операций.

Теперь запустим созданную веб-службу, нажав клавишу <F5>. Если формальных ошибок нет, мы получим отображение ASMX-файла в браузере (рис. 15.12).

Щелкнув в этом окне на изображении единственного метода данной веб-службы `Рекомендация`, мы попадаем на другую страницу, где получаем возможность его тестировать. Для этого щелкнем на кнопке **Invoke**. В результате тестирования получим

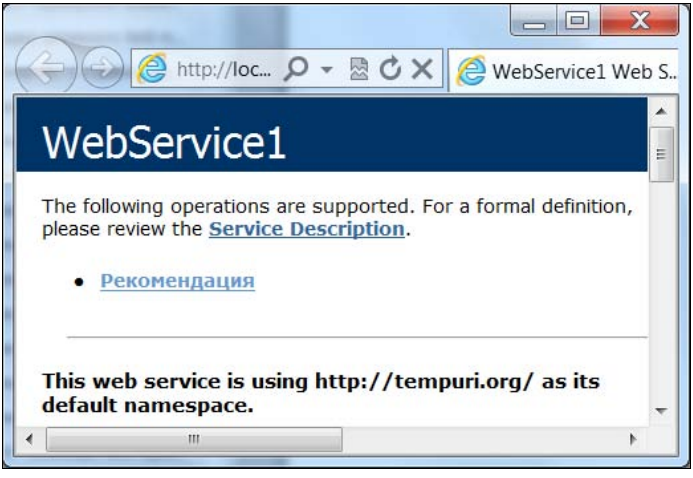


Рис. 15.12. Отображение ASMX-файла в браузере

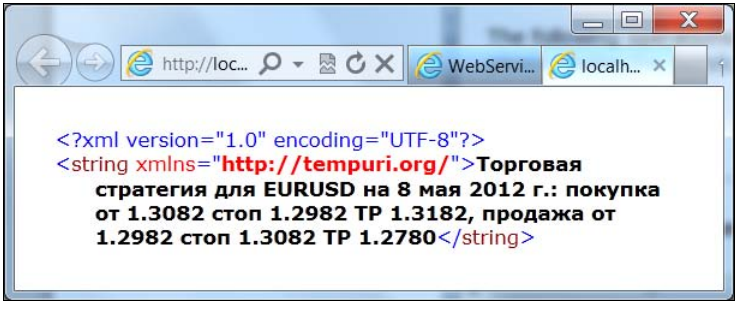


Рис. 15.13. XML-сообщение, как результат работы веб-сервиса

в браузере результат работы заданного метода веб-службы в формате XML-сообщения (рис. 15.13).

Убедиться в работоспособности программы можно, запустив соответствующее решение в папке ВебСлужбаForex сопровождающего книгу электронного архива.

Пример 131. Клиентское приложение, потребляющее сервис веб-службы "Торговая рекомендация на рынке Forex"

Теперь напишем клиентское Windows-приложение, использующее веб-службу, созданную нами в *примере 130*. Для этого запустим Visual Studio 12, выберем шаблон **Windows Forms Application Visual C#** и в поле **Name** зададим имя `ВебКлиентForex`. Из панели элементов **Toolbox** перенесем в форму метку **Label**, куда будем помещать строку торговой рекомендации, полученную от веб-службы "Торговая стратегия на рынке Forex".

Далее необходимо в проекте создать веб-ссылку на удаленный класс веб-службы. Для этого выберем в меню **Project** команду **Add Service Reference** — это приведет к появлению диалогового окна **Add Service Reference**. В этом окне щелкнем на кнопке **Advanced**, а затем, в следующем окне, на кнопке **Add Web Reference**.

Теперь в поле URL пишем виртуальный адрес ASMX-файла используемой веб-службы. Этот адрес мы можем получить, запустив нашу веб-службу из *примера 130* и скопировав в буфер обмена его URL-адрес. На вкладке программного кода набираем простейшую программу (листинг 15.7).

Листинг 15.7. Клиентское приложение, потребляющее сервис веб-службы "Торговая рекомендация на рынке Forex"

```
// Получаем прогноз рынка Forex на текущий день. Клиентское
// Windows-приложение, потребляющее сервис веб-службы предыдущего
// примера ВебСлужбаForex
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ВебКлиентForex
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Создаем клиентское приложение веб-службы:
            // http://localhost:1648/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 веб-службы, затем обращаемся к его методу `Рекомендация` и копируем получаемую строку в метку `Label1`. Пример работы программы приведен на рис. 15.14.

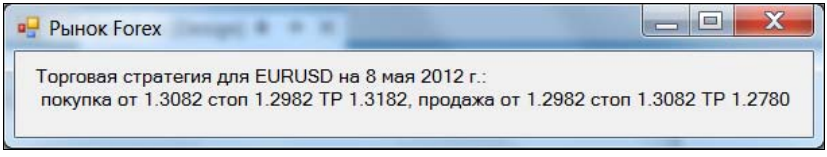


Рис. 15.14. Результат работы клиентского приложения

Убедиться в работоспособности программы можно, запустив соответствующее решение в папке ВебКлиентForex сопровождающего книгу электронного архива.

Пример 132. Клиентское веб-приложение, потребляющее сервис веб-службы "Морфер"

Создадим клиентское Windows-приложение (настольное) *не* для веб-службы, написанной нами, а для удаленного веб-сервиса. Различных готовых веб-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 веб-служб, а именно веб-службу склонения существительных "Морфер", ее адрес в Сети следующий: **http://www.morpher.ru/WebServices/Morpher.asmx**. Набрав этот URL в адресной строке браузера, мы получим (рис. 15.15) операции (методы), поддерживаемые веб-службой Morpher.

Эта веб-служба предназначена для автоматизированной обработки текстов на русском языке. В частности, метод `GetForms` веб-службы (см. рис. 15.15) обеспечивает склонение слов и словосочетаний на русском языке. На вход функции (метода) `GetForms` данной веб-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. Но интереснее и полезнее, на наш субъективный взгляд, в наше "банковско-бухгалтерское" время метод `PropisRUB`. На вход этого метода подаем сумму в рублях цифрами, метод же возвращает сумму прописью.

Для испытания этой веб-службы запустим Visual Studio 12, выберем шаблон **Windows Forms Application Visual C#**. В поле **Name** укажем имя проекта `ВебКлиентMorpher`. В конструкторе формы из панели элементов **Toolbox** перенесем текстовое поле

TextBox, две метки **Label** и кнопку **Button**. Далее, чтобы добавить ссылку на удаленный класс, выберем в меню **Project** (или в контекстном меню окна **Solution Explorer**) команду **Add Service Reference** — это приведет к открытию диалогового окна **Add Service Reference**. В этом окне щелкнем на кнопке **Advanced**, а затем, в следующем окне, на кнопке **Add Web Reference**. Далее в поле **URL** введем ссылку на веб-службу: **http://www.morpher.ru/WebServices/Morpher.asmx**.

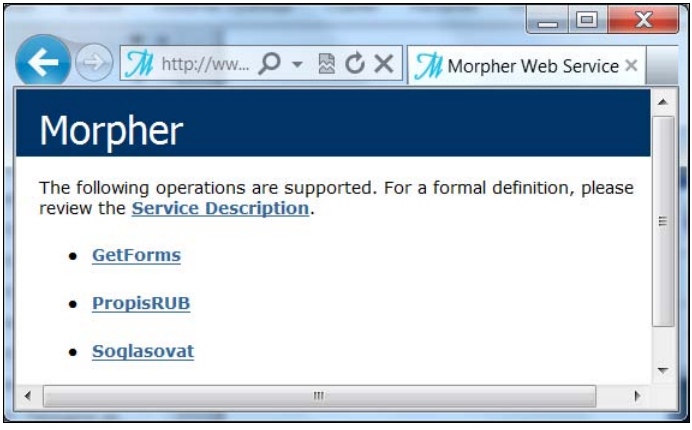


Рис. 15.15. Методы, поддерживаемые веб-службой Morpher

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 15.8).

Листинг 15.8. Клиентское настольное приложение, потребляющее сервис веб-службы "Морфер"

```
// Клиентское Windows-приложение, потребляющее сервис веб-службы
// "Морфер". На вход метода веб-службы PropisRUB подаем сумму
// в рублях цифрами, метод же возвращает сумму прописью
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ВебКлиентMorpher
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            label1.Text = "Введите сумму в рублях цифрами";
```

```
            label2.Text = null;
            button1.Text = "Перевести";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            Decimal Сумма;
            var Число_ли = Decimal.TryParse(textBox1.Text, out Сумма);
            if (Число_ли == false)
            {
                // Если пользователь ввел не число:
                label2.Text = "Следует вводить числа";
                label2.ForeColor = Color.Red; // — цвет текста на метке
                return; // — выход из процедуры
            }
            // Создаем клиентское приложение веб-службы:
            //           http://www.morpher.ru/WebServices/Morpher.asmx
            // Создаем экземпляр удаленного класса:
            var Методы = new ru.morpher.www.Morpher();
            var СуммаПрописью = Методы.PropisRUB(Сумма);
            label2.ForeColor = Color.Blue; // — цвет текста на метке
            label2.Text = СуммаПрописью;
        }
    }
}
```

Как видно из текста программы, при обработке события "щелчок на кнопке **Перевести**" создаем экземпляр удаленного класса `Методы` и далее используем метод `PropisRUB` класса для получения суммы прописью.

Пример работы программы представлен на рис. 15.16.

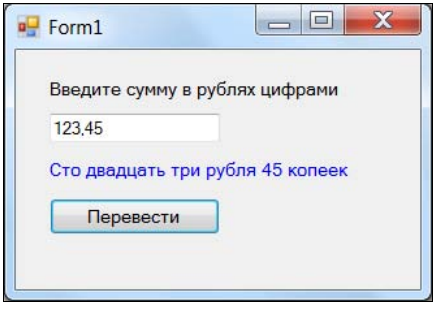


Рис. 15.16. Перевод введенного числа в сумму прописью обеспечивает веб-сервис

Убедиться в работоспособности программы можно, запустив соответствующий файл решения в папке ВебКлиентMorpher сопровождающего книгу электронного архива.

Пример 133. Получение веб-приложением данных от веб-службы Центрального банка РФ

Центральный банк Российской Федерации (ЦБРФ) предоставляет веб-службу для получения ежедневных экономических данных (курсы валют, учетные цены драгоценных металлов и проч.). Сервис этой веб-службы можно получать по адресу: **http://www.cbr.ru/scripts/Root.asp?Prtid=DWS**. Поставим задачу создания клиентского веб-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` этой веб-службы, на вход которой подают дату, а на выходе функции получают ежедневные курсы валют в виде `DataSet` (табличные данные).

Для решения задачи запустим Visual Studio 12, выберем шаблон **ASP.NET Empty Web Application Visual C#**. В поле **Name** зададим имя `ВебКлиентРоссия`. К текущему проекту добавим веб-форму. Для этого в меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком 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 меню **Project** выберем команду **Add Web Reference** и в открывшемся окне в поле **URL** введем адрес веб-сервиса: **http://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx**.

При этом станет доступной кнопка **Add Reference**. Щелкнем на этой кнопке, и 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 15.9).

Листинг 15.9. Получение веб-приложением данных от веб-службы Центрального банка РФ

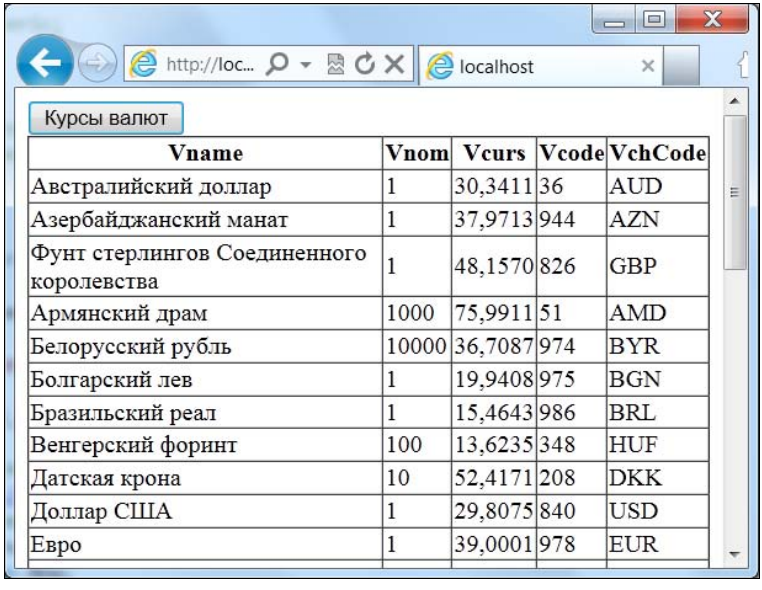
```
// Клиентское веб-приложение, потребляющее сервис веб-службы Центрального
// банка России для получения ежедневных курсов валют. На выходе
// приложения получаем таблицу курсов валют
using System;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ВебКлиентРоссия
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Button1.Text = "Курсы валют";
        }
        protected void Button1_Click(object sender, EventArgs e)
        {
```

```
            // Создаем клиентское приложение веб-службы:
            //        http://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx
            // Создаем экземпляр удаленного класса:
            var Валюта = new ru.cbr.www.DailyInfo();
            var Дата = DateTime.Now;
            // Получение ежедневных курсов валют — уже не поддерживается:
            // var НаборДанных = Валюта.GetSeldCursOnDate(Дата);
            // Функция, актуальная сегодня:
            var НаборДанных = Валюта.GetCursOnDate(Дата);
            // Содержимое DataSet в виде строки XML для отладки:
            var СтрокаXML = НаборДанных.GetXml();
            // Указываем источник данных для сетки данных:
            GridView1.DataSource = НаборДанных;
            var tmp = GridView1.DataMember;
            GridView1.DataBind();
        }
    }
}
```

Как и при создании предыдущих клиентов веб-служб, при обработке события "щелчок на кнопке" создаем экземпляр удаленного класса, в данном случае мы назвали его `Валюта`. На вход его метода `GetCursOnDate` подаем текущую дату. Этот метод возвращает данные типа `DataSet`, которые мы указываем в качестве источника данных `DataSource` для элемента управления "сетка данных" `GridView1`.

Пример работы программы показан на рис. 15.17.

Убедиться в работоспособности программы можно, открыв решение в папке ВебКлиентРоссия сопровождающего книгу электронного архива.



Vname	Vnom	Vcurs	Vcode	VchCode
Австралийский доллар	1	30,3411	36	AUD
Азербайджанский манат	1	37,9713	944	AZN
Фунт стерлингов Соединенного королевства	1	48,1570	826	GBP
Армянский драм	1000	75,9911	51	AMD
Белорусский рубль	10000	36,7087	974	BYR
Болгарский лев	1	19,9408	975	BGN
Бразильский реал	1	15,4643	986	BRL
Венгерский форинт	100	13,6235	348	HUF
Датская крона	10	52,4171	208	DKK
Доллар США	1	29,8075	840	USD
Евро	1	39,0001	978	EUR

Рис. 15.17. Ежедневные курсы валют веб-сервиса ЦБРФ

В заключение отмечу, что подобную веб-службу имеют также банки других стран, например, веб-служба Национального банка Республики Беларусь расположена по адресу: **http://www.nbrb.by/Services/ExRates.asmx**.

Пример 134. Получение Windows-приложением данных от веб-службы Национального банка Республики Беларусь

Веб-служба, предоставляющая текущие курсы валют, есть и у Национального банка Республики Беларусь. На примере потребления сервиса веб-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 12, выберем шаблон **Windows Forms Application Visual C#**. В поле **Name** зададим имя `ВебКлиентБеларусь`. В конструкторе формы добавим кнопку **Button** и сетку данных **DataGridView**. Мы намереваемся на элемент управления **DataGridView** вывести данные с курсами валют. Элемент управления **DataGridView** в Windows-приложении играет ту же роль, что и **GridView** в веб-приложении. Методы этих классов похожи, хотя есть и отличия.

Следующим этапом разработки клиентского приложения является добавление ссылки на удаленный класс искомой веб-службы. Для этого выберем в меню **Project** команду **Add Service Reference** — это приведет к открытию диалогового окна **Add Service Reference**. В этом окне щелкнем на кнопке **Advanced**, а в следующем окне — на кнопке **Add Web Reference**. В открывшемся диалоговом окне **Add Web Reference** в поле URL введем адрес веб-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 15.10).

Листинг 15.10. Получение Windows-приложением данных от веб-службы Национального банка Республики Беларусь

```
// Клиентское Windows-приложение, потребляющее сервис веб-службы
// Национального банка Республики Беларусь для получения ежедневных
// курсов валют. На выходе приложения получаем таблицу курсов валют
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace ВебКлиентБеларусь
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void Form1_Load(object sender, EventArgs e)
        {
            button1.Text = "Курсы валют";
        }
        private void button1_Click(object sender, EventArgs e)
        {        // Создаем клиентское приложение веб-службы:
            // http://www.nbrb.by/Services/ExRates.asmx
            // Создаем экземпляр удаленного класса:
            var Валюта = new by.nbrb.www.ExRates();
            // А этот сайт автор нашел на сайте:
            // http://ivbeg.bestpersons.ru/feed/post3279396/
            // Здесь есть ссылки на другие русскоязычные сервисы
            var Дата = DateTime.Now;
            // На вход метода ExRatesDaily подаем текущую дату:
            var НаборДанных = Валюта.ExRatesDaily(Дата);
            // Метод ExRatesDaily возвращает курсы валют в виде DataSet
            // Содержимое DataSet в виде строки XML для отладки:
            var 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 примерах этой главы 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.

Пример работы программы представлен на рис. 15.18.

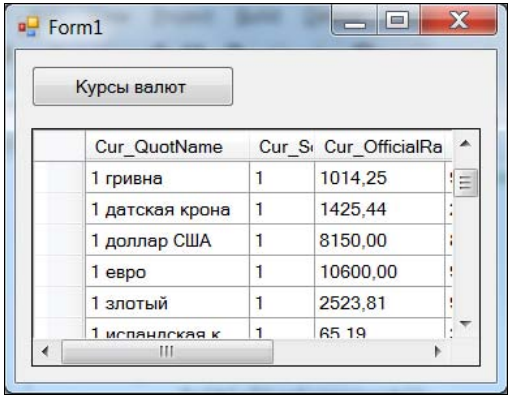


Рис. 15.18. Ежедневные курсы валют веб-службы НБРБ

Убедиться в работоспособности этой Windows-программы можно, открыв решение в папке ВебКлиентБеларусь сопровождающего книгу электронного архива.

Веб-приложение, решающее такую же задачу, — вывода на сетку данных курсов валют, можно тестировать, запустив соответствующий файл решения в папке ВебКлиентБеларусьWeb сопровождающего книгу электронного архива.

Пример 135. Создание веб-службы на основе WCF (WCF Service)

Поскольку Microsoft все большее внимание уделяет веб-службам на основе WCF (Windows Communication Foundation), создадим такую веб-службу. Задачу будем решать *самую простую*, чтобы максимально не запутать читателя. Наша веб-служба будет предлагать пользователю ввести свое имя, а в ответ WCF Service будет возвращать пользователю строку "<Введенное имя>, вы увлекаетесь WCF?".

Для этой цели запускаем Visual Studio 12, выбираем шаблон **ASP.NET Empty Web Application Visual C#**, назначаем имя: `WCF-Служба`. Далее в меню **Project** выберем пункт **Add New Item** и в открывшемся окне двойным щелчком щелкнем на шаблоне **WCF Service**.

В результате в нашем проекте, кроме прочих файлов, появятся два файла: **Service1.svc.cs** и **IService1.cs**. Содержимое этих файлов мы можем увидеть в листингах 15.11 и 15.12.

Листинг 15.11. Веб-служба на основе WCF, файл Service1.svc.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WCF_Служба
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to...
    // NOTE: In order to launch WCF Test Client for testing this service,...
    public class Service1 : IService1
    {
        public void DoWork()
        {
        }
    }
}
```

Листинг 15.12. Веб-служба на основе WCF, файл IService1.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WCF_Служба
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to...
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        void DoWork();
    }
}
```

Как видно из листингов, речь идет об одной и той же процедуре `DoWork()`, которую нам следует изменить так, чтобы она решала поставленную нами задачу. Эта процедура должна быть типа не `void` и возвращать значение строкового типа, а на ее вход мы должны подать одну переменную также строкового типа. И, конечно же, дадим процедуре название русскими буквами соответственно решаемой задаче, чтобы в последующих тестированиях мы сразу видели имя своей процедуры. Теперь сделаем указанные изменения обоих файлов и представим их в листингах 15.13 и 15.14.

Листинг 15.13. Веб-служба на основе WCF, файл Service1.svc.cs, измененный нами

```
using System;

namespace WCF_Служба
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to...
    // NOTE: In order to launch WCF Test Client for testing this service,...
    public class Service1 : IService1
    {
        public String ВводИмени(String Имя)
        {
            return Имя + ", вы увлекаетесь WCF ?";
        }
    }
}
```

Листинг 15.14. Веб-служба на основе WCF, файл IService1.cs, измененный нами

```
using System;
using System.ServiceModel;

namespace WCF_Служба
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to...
    [ServiceContract]
```

```
    public interface IService1
    {
        [OperationContract]
        String ВводИмени(String Имя);
    }
}
```

Это все, задача почти решена, теперь надо просто в этом убедиться. Запускаем службу нажатием клавиши <F5>. В результате на локальном сервере создается файл *.svc, а затем происходит тестирование нашей WCF-службы (рис. 15.19).

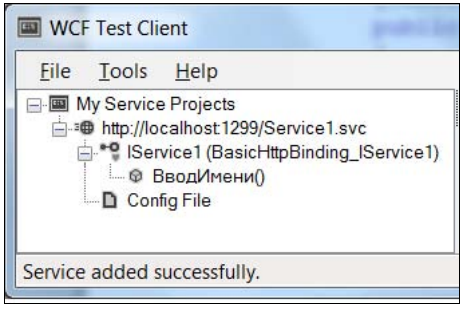


Рис. 15.19. Тестирование созданной WCF-службы

Теперь клиент нашей службы в своем Windows- или веб-приложении может указать ссылку (**Add Web Reference**) на этот удаленный SVC-файл и пользоваться созданной нами функцией. Убедиться в работоспособности этой WCF-службы можно, открыв соответствующий файл решения в папке WCF-Служба сопровождающего книгу электронного архива.

Создать WCF-службу можно также более коротким путем, а именно — после старта Visual Studio 12 выбрать шаблон **WCF Service Application**. Здесь в вашем проекте окажутся те самые два файла: **Service1.svc.cs** и **IService1.cs**. Вы можете изменить оба файла так же, как мы показали в нашем примере. Результат получим абсолютно таким же. Почему мы *не начали* свое изложение именно с шаблона **WCF Service Application**? Потому что он в своем исходном примере содержит больше строк программного кода, что, естественно, усложнило бы восприятие читателем основной сути.

Решив такую маленькую задачу и поняв принцип технологии WCF, можно уже браться за любую другую, по-настоящему серьезную и сложную.

Пример 136. Создание Windows-приложения, потребителя сервиса WCF-службы

Теперь напишем Windows-приложение, которое будет обращаться к WCF-службе, созданной в *примере 135*. Такое приложение можно называть *клиентским* или клиентом службы. Для этого действуем совершенно аналогично. Запустим Visual Studio 12, выберем шаблон **Windows Forms Application Visual C#**, укажем имя проекта `КлиентWCFСлужбы`. Из панели **Toolbox** перенесем в форму командную кнопку **Button**. За-

планируем, что вся работа с удаленной веб-службой будет происходить при обработке события "щелчок на кнопке **Пуск**".

Так же, как и в предыдущих случаях, создадим веб-ссылку на удаленный класс. Для этого выберем в меню **Project** (или в контекстном меню окна **Solution Explorer**) команду **Add Service Reference** — это приведет к открытию диалогового окна **Add Service Reference**. В этом окне щелкнем на кнопке **Advanced**, а в следующем окне — на кнопке **Add Web Reference**.

Поскольку мы отлаживаем и веб-службу, и его клиентское приложение на одном локальном компьютере, в поле **URL** пишем виртуальный адрес SVC-файла. Этот адрес мы можем получить, запустив нашу веб-службу из *примера 135*, а затем скопировав в буфер обмена его URL-адрес. Для локального компьютера, на котором автор отлаживал данный пример, URL-адрес был таким: **http://localhost:2830/Service1.svc**.

Затем щелкаем на кнопке **Go** и тем самым получаем доступ к кнопке **Add Reference**, после щелчка на которой в окне **Solution Explorer** добавится значок ссылки на класс `localhost`, обеспечивающий доступ к WCF-службе.

Теперь, когда в наш проект добавлена ссылка на удаленный класс, мы можем вызывать методы этого класса в нашей программе (листинг 15.15).

Листинг 15.15. Windows-приложение, потребляющее сервис WCF-службы

```
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной программе
namespace КлиентWCFСлужбы
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            button1.Text = "Пуск";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Чтобы добавить веб-службу к обычному Windows-приложению:
            // Project | Add Service Reference | Advanced | Add Web Reference,
            // затем в поле URL пишем виртуальный адрес веб-службы
            // http://localhost:1299/Service1.svc
            // Создаем экземпляр удаленного класса:
            var Удаленный = new localhost.Service1();
            var Ответ = Удаленный.ВводИмени("Виктор");
            MessageBox.Show(Ответ);
        }
    }
}
```

Как видно из программного кода, при обработке события "щелчок мышью на кнопке **Пуск**" создаем экземпляр удаленного класса и обращаемся к его функции (методу) `ВводИмени`. Результат работы программы представлен на рис. 15.20.

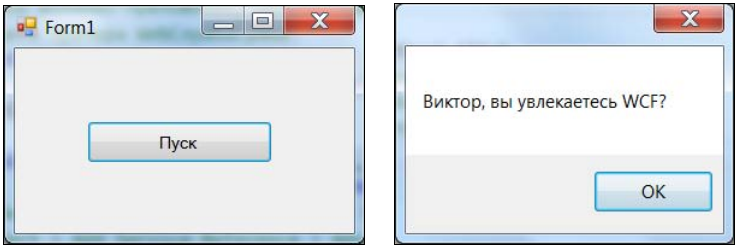
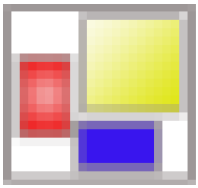


Рис. 15.20. Обращение Windows-приложения к WCF-службе

Убедиться в работоспособности программы можно, открыв соответствующий файл решения SLN в папке КлиентWCFСлужбы сопровождающего книгу электронного архива.

ГЛАВА 16



Использование технологии WPF

Что может нам дать WPF?

До сих пор мы обсуждали возможности программирования на Visual Basic 12 либо настольных Windows-приложений (частный случай — консольные приложения), которые пользователи запускают непосредственно (например, EXE-файлы), либо веб-приложений, выполняющихся внутри браузера. Оба этих приложения имеют свои как положительные качества, так и недостатки. Например, настольные (автономные) приложения обладают большей гибкостью и реактивностью, выражающейся, например, в практическом 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 веб-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 веб-приложения может затянуться на продолжительное время с учетом скорости пропуска сетевого трафика. С другой стороны, к веб-приложениям могут иметь доступ одновременно многие пользователи.

Однако в последнее время мы наблюдаем "сближение" возможностей Windows- и веб-приложений. Так, например, благодаря появлению веб-служб, Windows-приложения (так же, как и веб-приложения) получили возможность обрабатывать не только данные, которые имеются непосредственно на винчестере конкретного настольного компьютера, но и данные, предоставленные той или иной веб-службой. А доступ к помощи (справке по нажатию клавиши <F1>) во многих Windows-приложениях часто осуществляется посредством обращения к Интернету. Все большую гибкость веб-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 веб-страницы выполняется реже, Flash-технологии обеспечивают яркую и скоростную графику и т. д. Тем самым веб-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 веб-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 из области 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 (т. н. XAML Browser Applications, XBAP-приложение), будет работать как в операционной системе Windows, так и в операционных системах семейства UNIX/Linux, операционных системах Solaris и Macintosh (Mac OS). В результате этого практическая польза для веб-разработчиков состоит в следующем. Мы с вами говорили о том, что веб-хостинг, т. е. услуга по размещению в сети веб-страниц, осуществляется, в основном, под управлением операционных систем семейства UNIX/Linux. Хостингов на основе ASP.NET на Windows-платформе сравнительно немного. А это ограничивает написание веб-страниц на языках среды Visual Studio. Однако если мы будем писать веб-страницы на основе WPF-технологии, то проблема поиска необходимого хостинга отпадает, поскольку веб-страница, написанная на основе WPF-технологии, будет благополучно работать как на сервере под управлением UNIX/Linux, так и на Windows-сервере.

Пример 137. Создание простейшего WPF-приложения. Компоновка элементов управления с помощью сетки *Grid*

Поставим следующую задачу — требуется создать интерактивную веб-страницу, содержащую текстовое поле, метку и кнопку. Это 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 выводится результат извлечения корня. Это совсем маленькая задача, но на ее основе мы сможем рассмотреть все этапы создания WPF-приложения.

Итак, запустим Visual Studio 12, закажем новый проект шаблона **WPF Browser Application Visual C#** (из инсталлированных шаблонов **Windows**), назовем новый проект `WpfSqrt`. После щелчка на кнопке **OK** попадаем на вкладку **Page1.xaml**. В WPF-приложении, так же как и в ASPX-приложении, имеются 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-технологии, кроме файлов разметки *.xaml, имеются соответствующие файлы программной поддержки *.xaml.cs. На вкладке **Page1.xaml** разместим элементы управления, перетащив их из панели элементов **Toolbox** на проектируемую веб-страницу. Как видите, 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 веб-страница на языке XAML

```
<Page x:Class="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>
```

Перенесем на форму из панели элементов **Toolbox** (эту панель можно вызвать, как обычно, комбинацией клавиш <Ctrl>+<Alt>+<X>) необходимые нам текстовую метку **TextBox**, метку **Label** и командную кнопку **Button**. Сейчас мы не будем заботиться об правильной компоновке этих элементов на веб-странице, это мы сделаем в программном коде. Вообще говоря, для целей размещения элементов на веб-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.

Вернемся к нашей задаче. В настоящий момент у нас есть четыре программных объекта, которыми мы будем управлять из программного кода на Visual C#, — это упомянутая сетка, текстовое поле, метка и командная кнопка. Если мы посмотрим на вкладку **Page1.xaml** в режиме XAML, то увидим, что ни один из четырех названных объектов не имеет имени (свойство **Name**). Поэтому, чтобы обращаться к ним из программного кода, используя окно **Properties** (Свойства), зададим имя каждому объекту. Нажмите клавишу <F4>, если вы не видите вкладку **Properties**. На вкладке **Properties** задайте

такие имена: сетке **Grid** — `Сетка`, текстовому полю — `ТекстПоле`, метке — `Метка` и командной кнопке — `Кнопка`.

Программный код будет содержать процедуру обработки события загрузки сетки и процедуру обработки события "щелчок на кнопке". Эти процедуры добавим в наш проект, используя вкладку **Properties**. Для этого на вкладке **Properties** щелкнем на значке молнии и получим список возможных событий, доступных для данного объекта. Например, для сетки выберем событие **Loaded** и щелкнем по нему двойным щелчком. В результате среда сгенерирует пустой обработчик для этого события на вкладке программного кода **Page1.xaml.cs**. *Аналогично* получим пустой обработчик для события "щелчок на кнопке".

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 WpfКорень
{
    /// <summary>
    /// Interaction logic for Page1.xaml
    /// </summary>
    public partial class Page1 : Page
    {
        public Page1()
        {
            InitializeComponent();
        }
        private void Сетка_Loaded(object sender, RoutedEventArgs e)
        {
            // Строка заголовка веб-страницы:
            this.WindowTitle = "Извлекаю корень";
            // Компонуем элементы управления с помощью сетки Grid:
            Сетка.Width = 200; Сетка.Height = 100;
            // Сетку размещаем в левом верхнем углу веб-страницы:
            Сетка.HorizontalAlignment = HorizontalAlignment.Left;
            Сетка.VerticalAlignment = VerticalAlignment.Top;
            // Удобно для отладки показать линии сетки:
            // Сетка.ShowGridLines = True
```

```
        // Расстояния от краев веб-страницы до сетки:
        Сетка.Margin = new Thickness(10, 10, 10, 10);
        // Цвет объекта Сетка:
        Сетка.Background = Brushes.LightGray;
        // Сетка имеет одну колонку и три ряда:
        Сетка.ColumnDefinitions.Add(new ColumnDefinition());
        Сетка.RowDefinitions.Add(new RowDefinition());
        Сетка.RowDefinitions.Add(new RowDefinition());
        Сетка.RowDefinitions.Add(new RowDefinition());
        // Указываем, что текстовое поле расположить в нулевой колонке в
        // нулевом ряду:
        Grid.SetRow(ТекстПоле, 0); Grid.SetColumn(ТекстПоле, 0);
        Grid.SetRow(Метка, 1); Grid.SetColumn(Метка, 0);
        Grid.SetRow(Кнопка, 2); Grid.SetColumn(Кнопка, 0);
        // Текстовое поле располагаем в левом нижнем углу ячейки, сдвинув
        // его на 10 пикселов:
        ТекстПоле.HorizontalAlignment = HorizontalAlignment.Left;
        ТекстПоле.VerticalAlignment = VerticalAlignment.Bottom;
        ТекстПоле.Margin = new Thickness(10, 0, 0, 0);
        ТекстПоле.Width = 120; ТекстПоле.Focus();
        ТекстПоле.Text = null;
        Метка.HorizontalAlignment = HorizontalAlignment.Left;
        Метка.VerticalAlignment = VerticalAlignment.Bottom;
        Метка.Margin = new Thickness(10, 0, 0, 0);
        Кнопка.HorizontalAlignment = HorizontalAlignment.Left;
        Кнопка.VerticalAlignment = VerticalAlignment.Top;
        Кнопка.Margin = new Thickness(10, 0, 0, 0);
        Кнопка.Content = "Извлечь корень"; Кнопка.Width = 120;
        Метка.Content = "Введите число в текстовое поле";
        Метка.Foreground = Brushes.Blue;// — синий цвет текста на метке
    }
    private void Кнопка_Click(object sender, RoutedEventArgs e)
    {
        // Поскольку метод Sqrt предусматривает на входе и на выходе
        // переменные Double, будем преобразовывать к этому типу числа,
        // принимаемые от пользователя
        Double X;  // — из этого числа будем извлекать корень
        var Число_ли = Double.TryParse(ТекстПоле.Text,
              System.Globalization.NumberStyles.Number,
              System.Globalization.NumberFormatInfo.CurrentInfo, out X);
        // Второй параметр — это разрешенный стиль числа (Integer,
        // шестнадцатеричное 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:
            Метка.Content = "Следует вводить числа";
```

```
                // Красный цвет текста на метке:
                Метка.Foreground = Brushes.Red;
                return; // — выход из процедуры
            }
            Double Y = Math.Sqrt(X); // — извлечение корня
            Метка.Foreground = Brushes.Blue; // — синий цвет текста на метке
            Метка.Content = String.Format("Корень из {0} равен {1:F5}", X, Y);
        }
    }
}
```

Как видно из программного кода, задачу компоновки элементов управления в ячейках сетки решаем при обработке процедуры загрузки сетки. Мы задали сетку структурой в одну колонку и три ряда. В верхнем ряду расположили текстовое поле, в среднем — метку, а нижнем ряду — командную кнопку. Результат такой компоновки показан на рис. 16.1. При обработке события "щелчок на кнопке" проверяем, число ли ввел пользователь в текстовое поле, и выводим результат извлечения корня в метку.

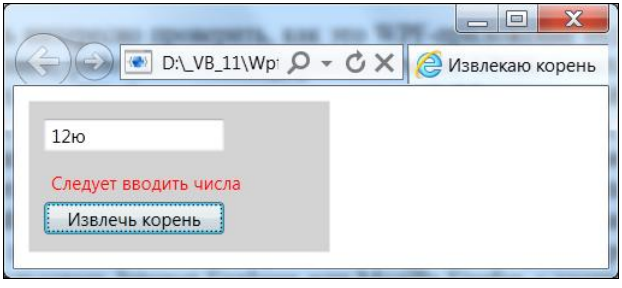


Рис. 16.1. Результат диагностики вводимых данных

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 под Windows значительно меньше). Мы не случайно в самом начале назвали данный проект WpfSqrt, использовав при этом только английские буквы — 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 (файл типа exe — исполняемый файл, он содержит скомпилированный код, манифест приложения и pdb-файл содержат метаданные, связанные с приложением, а файл типа xbap открывает браузер). В качестве браузера следует использовать Internet Explorer или Mozilla Firefox с соответствующими плагинами.

Убедиться в работоспособности программы можно, открыв решение в папке WpfSqrt сопровождающего книгу электронного архива.

Напоминание

Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 138. Использование одного из эффектов анимации

Как уже упоминалось, Microsoft в WPF-технологии стремится по графическим возможностям приблизиться к возможностям Adobe Flash. Покажем на следующем примере возможность анимации (т. е. способности двигаться) кнопки и изображения при щелчке мышью 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 — анимация интерполяцией, при которой свойство модифицируется плавно от начальной точки до конечной.

Для демонстрации этой возможности запустим Visual Studio 12, выберем среди инсталлированных шаблонов **Windows** шаблон **WPF Browser Application** и назовем его `WpfАниме`. Чтобы на вкладке **Page1.xaml** появились командная кнопка и изображение, на панели элементов **Toolbox** двойным щелчком щелкнем по элементам **Button** и **Image**.

Скопируем в папку проекта изображение poryv.png, а затем *добавим это изображение в проект* командой **Project | Add Existing Item | Image Files | Add**. Это важно сделать, поскольку, создавая обычные приложения (не WPF), мы не добавляли в проект файлы изображений, а просто на них ссылались в программном коде. Теперь мы увидим файл poryv.png в окне **Solution Explorer**.

На вкладке дизайнера формы, используя окно **Properties** (Свойства), зададим имя каждому объекту. Нажмите клавишу <F4>, если вы не видите вкладку **Properties**. На вкладке **Properties** зададим такие имена: сетке **Grid** — `Сетка`, изображению (**Image**) — `Изо` и командной кнопке — `Кнопка`.

Далее в программном коде нам понадобятся три пустых обработчика событий: загрузки сетки, щелчка на кнопке и щелчка на изображении. Чтобы получить эти три обработчика, на вкладке дизайнера формы в окне **Properties** щелкнем на значке молнии и получим список возможных событий, доступных для данного объекта. Для сетки выберем событие **Loaded** и двойным щелчком щелкнем на нем. В результате среда сгенерирует пустой обработчик для этого события на вкладке программного кода **Page1.xaml.cs**. *Аналогично* получим пустые обработчики для событий "щелчок на кнопке" и "щелчок на изображении".

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 веб-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;
using System.Windows.Media.Imaging;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace WpfАниме
{
    /// <summary>
    /// Interaction logic for Page1.xaml
    /// </summary>
    public partial class Page1 : Page
    {
        public Page1()
        {
            InitializeComponent();
        }
        private void Сетка_Loaded(object sender, RoutedEventArgs e)
        {
            this.WindowTitle = "Анимационный эффект";
            // Сетку размещаем в левом верхнем углу веб-страницы:
            Сетка.HorizontalAlignment = HorizontalAlignment.Left;
            Сетка.VerticalAlignment = VerticalAlignment.Top;
            // Удобно для отладки показать Grid-линии:
            // Сетка.ShowGridLines = True
            // Расстояния от краев веб-страницы до сетки Сетка:
            Сетка.Margin = new Thickness(10, 10, 10, 10);
            // Цвет объекта Сетка:
            Сетка.Background = Brushes.LightGray;
            // Сетка имеет одну колонку и два ряда:
            Сетка.ColumnDefinitions.Add(new ColumnDefinition());
            Сетка.RowDefinitions.Add(new RowDefinition());
            Сетка.RowDefinitions.Add(new RowDefinition());
            // Указываем, что кнопку расположить в верхнем ряду,
            // а изображение — в нижнем ряду:
            Grid.SetRow(Кнопка, 0); Grid.SetColumn(Кнопка, 0);
            Grid.SetRow(Изо, 1); Grid.SetColumn(Изо, 0);
            Кнопка.Width = 100;
            Кнопка.Content = "Привет!"; Кнопка.Focus();
            Кнопка.Margin = new Thickness(10, 10, 10, 10);
            // Указываем, что изображение добавлено в проект:
            Изо.Source = new BitmapImage(
                         new Uri("poryv.png", UriKind.Relative));
            Изо.Margin = new Thickness(10, 10, 10, 10);
            Изо.Width = 100; Изо.Height = 100;
        }
        private void Кнопка_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:
            Кнопка.BeginAnimation(Button.WidthProperty, Аниме);
        }
        private void Изо_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);
            Изо.BeginAnimation(Image.WidthProperty, Аниме);
            Изо.BeginAnimation(Image.HeightProperty, Аниме);
        }
    }
}
```

При обработке события загрузки сетки 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. Аналогичные установки и команды выполняем при щелчке мышью на изображении, только анимационным преобразованиям подвергаем одновременно и ширину (`Width`) и высоту (`Height`) изображения.

Убедиться в работоспособности программы можно, открыв решение в папке WpfАниме сопровождающего книгу электронного архива.

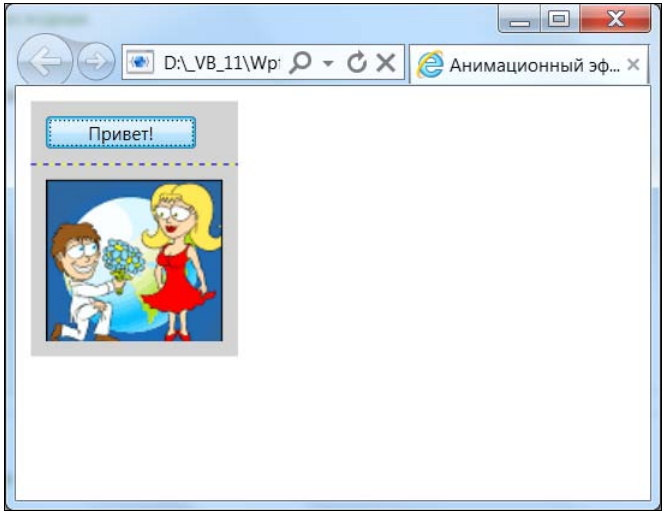


Рис. 16.2. Изображение плавно уменьшается

Пример 139. Эффект постепенной замены (прорисовки) одного изображения другим

Создадим WPF-приложение, выполняемое в браузере и содержащее на веб-странице два изображения. В качестве изображений мы выбрали презентационные рисунки автомобилей Chevrolet Cavalier и BMW M3. Поскольку они расположены на странице одинаково, и их размеры совпадают, пользователь видит только одно — "верхнее" — изображение. Таким образом, при загрузке страницы мы видим лишь изображение 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 изображение Chevrolet Cavalier. Повторный щелчок мышью на изображении автомобиля Chevrolet постепенно "проявляет" опять "верхнее" изображение, и мы вновь имеем честь видеть автомобиль BMW.

Для решения поставленной задачи запустим Visual Studio 12, выберем среди инсталлированных шаблонов **Windows** шаблон **WPF Browser Application** и назовем его `WpfДваИзо`. Далее, перейдя на вкладку дизайна **Page1.xaml**, на панели элементов **Toolbox** двойным щелчком щелкнем по элементу **Image**. Поскольку нам требуются два таких элемента, сделаем это еще раз. Мы можем сейчас совершенно не озабочиваться расположением этих элементов на веб-странице и заданием их размеров, поскольку это мы определим в программном коде. На вкладке дизайнера формы, используя окно свойств **Properties**, зададим следующие имена: для сетки — `Сетка`, для двух изображений (**Image**) соответственно `Изо1` и `Изо2`.

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 | Image Files | Add**. При этом в окне **Solution Explorer** появятся значки этих двух файлов.

На вкладке дизайнера формы в окне **Properties** щелкнем на значке молнии и получим список возможных событий, доступных для данного объекта. Для сетки выберем событие **Loaded** и щелкнем на нем двойным щелчком. В результате среда сгенерирует пустой обработчик для этого события на вкладке программного кода **Page1.xaml.cs**. Аналогично получим пустой обработчик события "щелчок на первом изображении". 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 веб-странице два изображения. Поскольку
// месторасположение обоих изображений задано одинаково, а также совпадают
// размеры изображений, пользователь будет видеть только "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 WpfДваИзо
{
    public partial class Page1 : Page
    {
        Boolean Флажок = false; // Внешняя переменная
        public Page1()
        {
            InitializeComponent();
        }
        private void Сетка_Loaded(object sender, RoutedEventArgs e)
        {
            this.WindowTitle = "Щелкни на изображении";
            // Изображение автомобиля Chevrolet Cavalier:
            Изо1.Source = new BitmapImage(
                                    new Uri("c1.bmp", UriKind.Relative));
            // Изображение автомобиля BMW M3:
            Изо2.Source = new BitmapImage(
                                    new Uri("c2.bmp", UriKind.Relative));
            // Положение обоих изображений на странице задаем одинаковое,
            // поэтому один рисунок закрывает другой:
            Изо1.HorizontalAlignment = HorizontalAlignment.Left;
            Изо2.HorizontalAlignment = HorizontalAlignment.Left;
            Изо1.VerticalAlignment = VerticalAlignment.Top;
            Изо2.VerticalAlignment = VerticalAlignment.Top;
            // Размеры изображений:
            Изо1.Width = 591; Изо1.Height = 258;
            Изо2.Width = 591; Изо2.Height = 258;
            // Расстояния от краев веб-страницы до сетки Grid:
            Изо1.Margin = new Thickness(10, 10, 0, 0);
            Изо2.Margin = new Thickness(10, 10, 0, 0);
            // Присоединяем один обработчик двух событий:
            Изо1.MouseDown += new MouseButtonEventHandler(Изо_MouseDown);
            Изо2.MouseDown += new MouseButtonEventHandler(Изо_MouseDown);
            // Свойство Opacity задает уровень непрозрачности
            // Изо1.Opacity = 1;
        }
        private void Изо_MouseDown(object sender, RoutedEventArgs e)
        {
            // Ниже написана 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:
            Изо2.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. При обработке события загрузки сетки задаем принадлежность фотографий `c1.bmp` и `c2.bmp` изображениям `Изо1` и `Изо2`. Также задаем их размер и месторасположение на странице.

Далее 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 `Изо_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 пример работы программы.

Рис. 16.3. После щелчка на изображении автомобиля BMW постепенно прорисовывается автомобиль Chevrolet

Убедиться в работоспособности программы можно, открыв решение в папке WpfДваИзо сопровождающего книгу электронного архива.

Пример 140. Закрашивание области текста горизонтальным линейным градиентом

Создадим автономное настольное WPF-приложение, включающее 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 12, выберем среди инсталлированных шаблонов **Windows** шаблон **WPF Application**, т. е. автономное WPF-приложение, и назовем его `WpfГрадиент`. Далее, перейдя на вкладку дизайна **MainWindows.xaml**, на панели элементов **Toolbox** двойным щелчком щелкнем по элементу **TextBlock**. Теперь на вкладке дизайнера формы, используя окно свойств **Properties**, зададим следующие имена: для сетки — `Сетка`, для элемента **TextBlock** — `ТекстБлок`. В этом же окне **Properties** щелкнем на значке молнии и получим список возможных событий, доступных для данного объекта. Для сетки выберем событие **Loaded** и щелкнем на нем двойным щелчком. В результате среда сгенерирует пустой обработчик для этого события на вкладке программного кода **Page1.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 WpfГрадиент
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
        private void Сетка_Loaded(object sender, RoutedEventArgs e)
        {
            this.Title = "Игорь Губерман";
```

```
            ТекстБлок.Text =
                "Какие дамы, и не раз, шептали: \"Дорогой," + "\r\n" +
                "Конечно, да, но не сейчас, не здесь, и не с тобой!\"";
            ТекстБлок.Width = 470; ТекстБлок.Height = 50;
            ТекстБлок.FontSize = 20;
            // ТекстБлок.Background = Brushes.Gray
            // Создаем горизонтальный линейный градиент
            var 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:
            ТекстБлок.Foreground = Градиент;
        }
    }
}
```

Как видно из программного кода, сразу при обработке события загрузки сетки написан шуточный афоризм Игоря Губермана для свойства текстового блока `Text`. Далее создан 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).

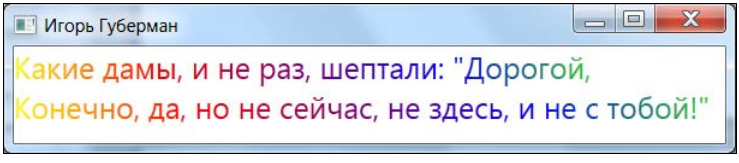


Рис. 16.4. Закрашивание текстового блока горизонтальным линейным градиентом

Убедиться в работоспособности программы можно, открыв решение в папке WpfГрадиент сопровождающего книгу электронного архива.

Пример 141. 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 создадим здесь

автономное настольное WPF-приложение, реализующее проигрыватель мультимедиа с возможностями включения паузы, остановки и регулировки уровня громкости.

Для этой цели запустим среду Visual C# 12, 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** служит для расположения элементов по вертикали (по умолчанию) 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>) двойным щелчком щелкнем на элементе **StackPanel**. Далее, на панели **Toolbox** двойным щелчком щелкнем последовательно на элементах **MediaElement**, а затем еще раз — на **StackPanel** (т. е. вторая "стек-панель" будет вложена в первую). Иными словами, первый элемент **StackPanel** будет содержать (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"), двойным щелчком щелкнем по ним последовательно на панели **Toolbox**. Мы хотим расположить их горизонтально (рис. 16.5), но об этом и о многом другом мы позаботимся в программном коде. Хотя часто это делают на языке XAML (это оказывается более удобным), но мы не забываем, что в книге речь идет о языке Visual C#. Перед тем как перейти на вкладку программного кода, посмотрим содержимое файла разметки **MainWindows.xaml** (листинг 16.6).

Листинг 16.6. 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>
        <MediaElement/>
        <StackPanel>
            <Button Content="Button"/>
            <Button Content="Button"/>
            <Button Content="Button"/>
            <TextBlock Text="TextBlock"/>
            <Slider/>
        </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, используя окно **Properties**, дадим имя каждому элементу. Сетке **Grid** — имя `Сетка`, первой стек-панели — `Ст1`, второй — `Ст2`, кнопкам соответственно: `Кн1`, `Кн2`, `Кн3`, текстовому блоку — `ТБ`, "ползунку" — `Сл` и элементу **MediaElement** — `Медиа`.

Теперь перейдем на вкладку **MainWindows.xaml.cs** и напишем программный код, представленный в листинге 16.7. При этом не забываем, что для каждого программируемого события, скажем, щелчок на кнопке, следует его создать, например, используя окно **Properties**. Созданные события отобразятся в XAML-разметке.

Листинг 16.7. 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
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
        private void Сетка_Loaded(object sender, RoutedEventArgs e)
        {
            // Строка заголовка веб-страницы:
            this.Title = "Медиапроигрыватель";
            this.Width = 435; this.Height = 430;
            // Компонуем элементы управления с помощью панели StackPanel:
            Ст1.Margin = new Thickness(10, 10, 0, 0);
            Ст1.Width = 410; Ст1.Height = 400;
            Ст2.Width = 410; Ст2.Height = 50;
            // При отладке удобно задать фоновый цвет панели:
            //Ст2.Background = Brushes.LightGray;
            Ст1.Orientation = Orientation.Vertical;
            Ст2.Orientation = Orientation.Horizontal;
            Кн1.Height = 23; Кн2.Height = 23; Кн3.Height = 23;
            Кн1.Width = 75; Кн2.Width = 75; Кн3.Width = 75;
            Кн1.Focus(); Кн1.Content = "Play";
            Кн2.Content = "Pause"; Кн3.Content = "Stop";
            // Расстояния от краев текстового блока до краев окна:
            ТБ.Margin = new Thickness(10, 13, 10, 0);
            ТБ.Text = "Громкость:"; ТБ.FontSize = 12;
```

```
            // Элемент выбора из диапазона значений "Ползунок" (Slider):
            Сл.Height = 23; Сл.Width = 100;
            Сл.Minimum = 0; Сл.Maximum = 1; Сл.Value = 0.5;
            // MediaElement представляет элемент, содержащий аудио
            // и/или видео:
            Медиа.Source = new Uri(@"D:/ВысшийПилотаж.mp4");
            Медиа.Width = 400; Медиа.Height = 340;
            Медиа.HorizontalAlignment = HorizontalAlignment.Left;
            // Manual — состояние, используемое для управления элементом
            // MediaElement вручную, при этом можно использовать
            // интерактивные методы, такие как Play и Pause:
            Медиа.LoadedBehavior = MediaState.Manual;
            Медиа.UnloadedBehavior = MediaState.Stop;
            Медиа.Stretch = Stretch.Fill;
        }
        private void Кн1_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:
            Медиа.Play();
            // Задаем громкость воспроизведения файла мультимедиа в
            // зависимости от состояния элемента "ползунок":
            Медиа.Volume = Сл.Value;
        }
        private void Медиа_MediaEnded(Object sender, RoutedEventArgs e)
        {
            // При завершении воспроизведения файла мультимедиа переводим
            // элемент MediaElement в состояние Stop:
            Медиа.Stop();
        }
        private void Сл_ValueChanged(Object sender,
                                     RoutedPropertyChangedEventArgs<double> e)
        {
            // Если значение свойства Value элемента "ползунок" изменяется,
            // то изменяем громкость воспроизведения мультимедиа Volume:
            Медиа.Volume = Сл.Value;
        }
        private void Кн2_Click(Object sender, RoutedEventArgs e)
        {
            // Приостанавливаем воспроизведение файла мультимедиа:
            Медиа.Pause();
        }
        private void Кн3_Click(object sender, RoutedEventArgs e)
        {
            // Останавливаем и возвращаем воспроизведение файла мультимедиа
            // на начало:
            Медиа.Stop();
        }
    }
}
```

Как видно из программного кода, при обработке события загрузки сетки 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.

Пример работы программы представлен на рис. 16.5.

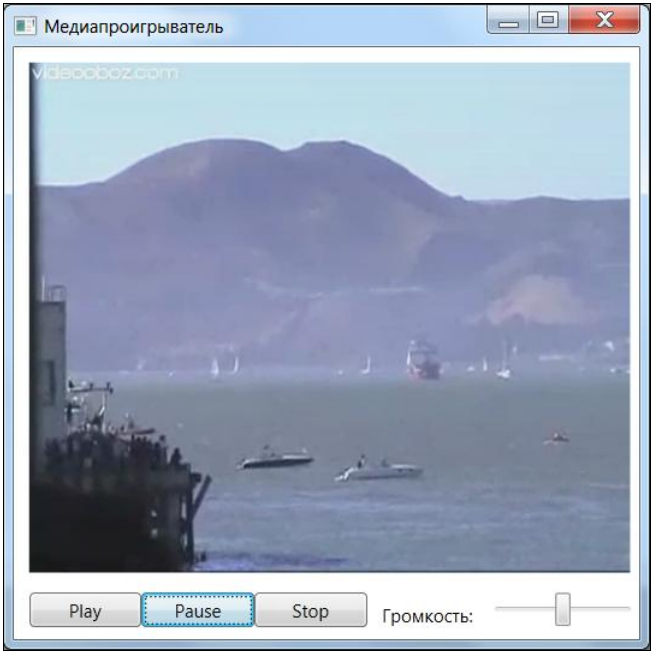


Рис. 16.5. Интерфейс созданного WPF-проигрывателя

Убедиться в работоспособности программы можно, открыв решение в папке WpfПроигрыватель сопровождающего книгу электронного архива.

Пример 142. Наложение текста на видео

Вспомним видеоролик из *примера 141*. Тихая морская гавань, размеренная, неторопливая летняя атмосфера. И вдруг в эту умиротворенность врывается рев самолета. Мы видим очень быстро летящий 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 Studio 12, 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. В окне **Properties** дадим название элементу **MediaElement**: `Медиа`. Сетке **Grid** давать имя не будем, а для того, чтобы получить пустой обработчик события инициализации сетки (можно также загрузки сетки), щелкнем на значке молнии и в списке возможных событий для сетки выберем событие **Initialized**. Затем перейдем на вкладку **MainWindows.xaml.cs** и введем текст, представленный в листинге 16.8.

Листинг 16.8. Вариант использования 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
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
        private void Grid_Initialized_1(object sender, EventArgs e)
        {
            this.Title = "Высший пилотаж";
            // Размеры окна:
            this.Width = 420; this.Height = 355;
            Медиа.Width = 400; Медиа.Height = 300;
            Медиа.Source = new Uri(@"D:/ВысшийПилотаж.mp4");
            // Создаем форматированный текст:
            var ФорматированныйТекст = new FormattedText("ПОЛЕТ",
               System.Globalization.CultureInfo.CurrentCulture,
                FlowDirection.LeftToRight,
                new Typeface(this.FontFamily, FontStyles.Normal,
                    FontWeights.ExtraBold, FontStretches.Normal),
                111.0, Brushes.Red);
            // Строим геометрическую фигуру с отрисованным форматированным
            // текстом:
            var ГеометрическаяФигура = ФорматированныйТекст.
                                 BuildGeometry(new Point(12.0F, -36.0F));
            // Масштабируем геометрическую фигуру, растягивая ее по
            // вертикали:
            var Масштаб = new ScaleTransform();
            // Указываем коэффициент растяжения по вертикали:
            Масштаб.ScaleY = 3.55;
```

```
            ГеометрическаяФигура.Transform = Масштаб;
            // Задаем геометрию в пределах контура медиаэлемента:
            Медиа.Clip = ГеометрическаяФигура;
        }
    }
}
```

Как видно из текста программы, при обработке события инициализации сетки 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.

Пример работы программы показан на рис. 16.6.

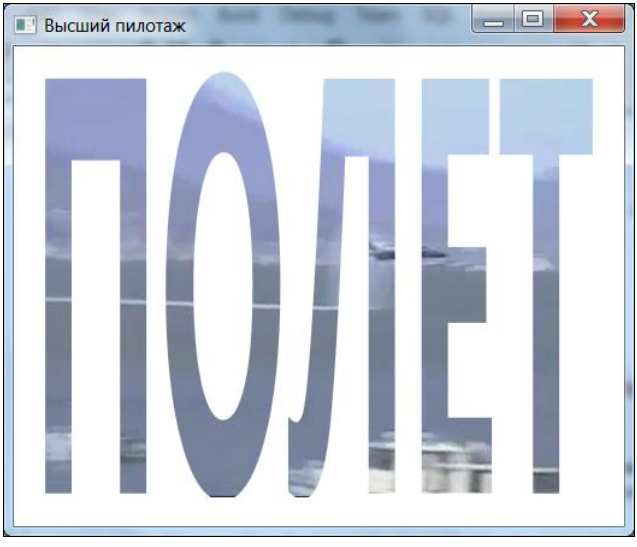


Рис. 16.6. Наложение текста на видео в WPF-приложении

Убедиться в работоспособности программы можно, открыв решение в папке WpfТекстНаВидео сопровождающего книгу электронного архива.

Пример 143. Переходы в WPF-приложениях

Приложения WPF часто состоят из набора страниц. По мере работы с приложением пользователь переходит со страницы на страницу, как в веб-приложении. Windows Presentation Foundation поддерживает переходы в стиле браузера, которые могут использоваться в приложениях обоих типов: автономных приложениях и XBAP-приложениях веб-браузера. Для демонстрации возможности перехода от страницы к странице создадим WPF-приложение, выполняемое в браузере. Это приложение будет содержать две веб-страницы. Первая веб-страница имеет кнопку для перехода на вторую веб-страницу и гиперссылку для перехода на почтовый сервер **www.ukr.net**. Вторая веб-страница содержит только гиперссылку для перехода назад, на первую веб-страницу. Таким образом, мы продемонстрируем возможность перехода с одной страницы на другую с помощью объекта **NavigationService** и гиперссылки **Hyperlink**.

Итак, запустим Visual Studio 12, закажем новый проект шаблона **WPF Browser Application C#** (из инсталлированных шаблонов **Windows**), назовем новый проект `WpfПереходы`. После щелчка на кнопке **OK** попадем на вкладку **Page1.xaml**. Вызвав панель элементов **Toolbox** (например, с помощью комбинации клавиш <Ctrl>+<Alt>+ +<X>), двойным щелчком щелкнем на элементе **Button**, а затем — на элементе **TextBlock**. В результате на вкладке дизайна **Page1.xaml** появятся один элемент над другим: командная кнопка и текстовый блок. Эти элементы будем компоновать с помощью сетки в программном коде. На вкладке дизайна в окне **Properties** дадим имена следующим элементам: сетке **Grid** — `Сетка`, командной кнопке **Button** — `Кнопка` и элементу **TextBlock** — `ТекстБлок`. Также, используя окно **Properties**, получим два пустых обработчика событий: инициализации сетки и щелчка на кнопке.

На вкладке программного кода **Page1.xaml.cs** введем программный код, представленный в листинге 16.9.

Листинг 16.9. Организация переходов на другие ресурсы, страница 1

```
// Данное WPF-приложение содержит две веб-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 веб-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;
using System.Windows.Media;
// Другие директивы using удалены, поскольку они не используются в данной
// программе
namespace WpfПереходы
{
    public partial class Page1 : Page
    {
        public Page1()
```

```
        {
            InitializeComponent();
        }
        private void Сетка_Initialized(object sender, EventArgs e)
        {
            this.WindowTitle = "Первая WPF-страница";
            // Сетку размещаем в левом верхнем углу веб-страницы:
            Сетка.HorizontalAlignment = HorizontalAlignment.Left;
            Сетка.VerticalAlignment = VerticalAlignment.Top;
            // Удобно для отладки показать линии сетки:
            // Сетка.ShowGridLines = true;
            // Расстояния от краев веб-страницы до сетки Сетка:
            Сетка.Margin = new Thickness(10, 10, 10, 10);
            // Цвет объекта Сетка:
            Сетка.Background = Brushes.LightGray;
            // Сетка имеет одну колонку и два ряда:
            Сетка.ColumnDefinitions.Add(new ColumnDefinition());
            Сетка.RowDefinitions.Add(new RowDefinition());
            Сетка.RowDefinitions.Add(new RowDefinition());
            // Указываем, что кнопку расположить в верхнем ряду,
            // а изображение — в нижнем ряду:
            Grid.SetRow(Кнопка, 0); Grid.SetColumn(Кнопка, 0);
            Grid.SetRow(ТекстБлок, 1); Grid.SetColumn(ТекстБлок, 0);
            Кнопка.Width = 170; Кнопка.Focus();
            Кнопка.Content = "Перейти на вторую страницу";
            Кнопка.Margin = new Thickness(10, 10, 10, 10);
            ТекстБлок.Margin = new Thickness(10, 10, 10, 10);
            ТекстБлок.Text = "Перейти ";
            // Создаем объект класса Hyperlink для размещения гиперссылки:
            var 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:
            ТекстБлок.Inlines.Add(Ссылка);
        }
        private void Кнопка_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. В пояснении нуждается только элемент `Hyperlink`. Такого элемента нет в панели **Toolbox**, по-

скольку 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`. Пример работы веб-страницы Page1.xaml показан на рис. 16.7.

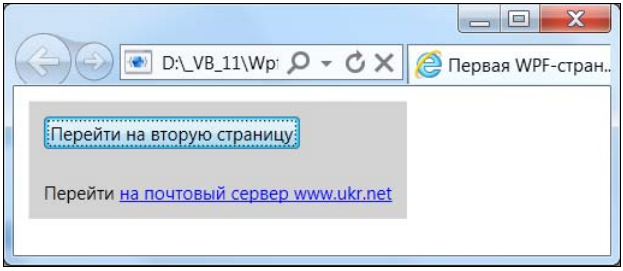


Рис. 16.7. Интерфейс первой веб-страницы WPF-приложения

Теперь создадим вторую веб-страницу, на которую будем переходить, щелкнув на командной кнопке. Для этого в меню **Project** выберем команду **Add New Item** и в открывшемся окне двойным щелчком 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, показанный в листинге 16.10.

Листинг 16.10. Организация переходов на другие ресурсы, страница 2

```
// Вторая страница приложения
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 WpfПереходы
{
    public partial class Page2 : Page
    {
        public Page2()
        {
            InitializeComponent();
        }
        private void Сетка_Initialized(object sender, EventArgs e)
        {
            this.WindowTitle = "Вторая WPF-страница";
            // Сетку размещаем в левом верхнем углу веб-страницы:
            Сетка.HorizontalAlignment = HorizontalAlignment.Left;
            Сетка.VerticalAlignment = VerticalAlignment.Top;
            // Удобно для отладки показать линии сетки:
            // Сетка.ShowGridLines = true;
```

```
            // Расстояния от краев веб-страницы до сетки Сетка:
            Сетка.Margin = new Thickness(10, 10, 10, 10);
            // Цвет объекта Сетка:
            // Сетка.Background = Brushes.LightGray;
            // Сетка имеет одну колонку и два ряда:
            Сетка.ColumnDefinitions.Add(new ColumnDefinition());
            Сетка.RowDefinitions.Add(new RowDefinition());
            Сетка.RowDefinitions.Add(new RowDefinition());
            // Указываем, что кнопку расположить в верхнем ряду,
            // а изображение — в нижнем ряду:
            Grid.SetRow(Метка, 0); Grid.SetColumn(Метка, 0);
            Grid.SetRow(ТекстБлок, 1); Grid.SetColumn(ТекстБлок, 0);
            Метка.Content = "Вы перешли на вторую страницу";
            ТекстБлок.Text = "Щелкните ";
            // Создаем объект класса Hyperlink для размещения гиперссылки:
            var 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:
            ТекстБлок.Inlines.Add(Ссылка);
            // Добавляем текст в текстовый блок:
            ТекстБлок.Inlines.Add(" для перехода назад на первую страницу.");
        }
    }
}
```

Как видно из текста программы, использованный подход аналогичен программной поддержке для первой веб-страницы. Пример работы второй веб-страницы Page2.xaml показан на рис. 16.8.

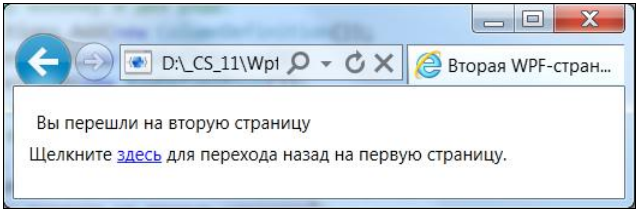


Рис. 16.8. Интерфейс второй веб-страницы WPF-приложения

Убедиться в работоспособности программы можно, открыв решение в папке WpfПереходы сопровождающего книгу электронного архива.

* * *

Желаю вам, уважаемые читатели, получить не только удовольствие от процесса программирования на Visual C#, но и зарабатывать с его помощью достойные вас деньги.

Извините, всё.

ПРИЛОЖЕНИЕ

Содержание электронного архива с примерами из книги

Электронный архив с примерами, рассмотренными в книге (табл. П1), можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Таблица П1. *Программы, рассмотренные в книге*

Название папки	**Описание программы**	**Номер примера**
First	Первая простейшая программа с текстовой меткой и кнопкой, щелчок на которой вызывает появление диалогового окна, в котором написано: "Всем привет!"	1
Hover	Простейшая программа с экранной формой, меткой, командной кнопкой и диалоговым окном, отслеживание события **MouseHover**	2
ВыборДаты	Программа выбора нужной даты	3
Корень	Программа вводит через текстовое поле число, при щелчке на командной кнопке извлекает из него квадратный корень и выводит результат на метку **Label**. В случае ввода не числа сообщает пользователю об этом предупреждением красного цвета также на текстовую метку	4
Паспорт	Программа для ввода пароля в текстовое поле, причем при вводе вместо вводимых символов, некто, "находящийся за спиной пользователя", увидит только звездочки	5
Флажок1	Программа управляет стилем шрифта текста введенного на метку **Label** посредством флажка **CheckBox**	6
Флажок2	Совершенствование предыдущей программы. Побитовый оператор XOR — исключающее ИЛИ	7
Вкладки	Программа, позволяющая выбрать текст из двух вариантов, задать цвет и размер шрифта для этого текста на трех вкладках **TabControl** с использованием переключателей **RadioButton**	8
ВоздШар	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**)	9

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
Комби	Программа, реализующая функции калькулятора. Здесь для отображения вариантов выбора арифметических действий используется комбинированный список **ComboBox**	10
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	11
Сумма	Программа организует ввод двух чисел, их сложение и вывод суммы на консоль	12
ТаблКорней	Консольное приложение задает цвета и заголовок консоли, а затем выводит таблицу извлечения квадратного корня от нуля до десяти	13
ConsoleMessageBox	Консольное приложение выводит в окно **MessageBox** текущую дату и время в различных форматах, используя **String.Format**	14
ССылкаНаVisualBasic	В этом 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**	15
Интервалы	Программирование интервалов с помощью **else if**	16
Месяцы	Программа создает словарь данных типа **Dictionary** и записывает в этот словарь названия месяцев и количество дней в каждом месяце. Ключом словаря является название месяца, а значением — количество дней. Используя цикл `foreach`, программа выводит на консоль только те месяцы, количество дней в которых равно 30	17
Мониторинг	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**	18
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"	19
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	20
Калькулятор	Программа Калькулятор с кнопками цифр. Управление калькулятором возможно только мышью. Данный калькулятор выполняет лишь арифметические операции	21

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
ССылкиLinkLabel	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**	22
СобытияКлавиатуры	Программа, информирующая пользователя о тех клавишах и комбинациях клавиш, которые тот нажал	23
ТолькоЦифры	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	24
ТолькоЦифры+ТчкОрЗпт	Программа разрешает ввод в текстовое поле только цифровых символов, а также разделитель целой и дробной частей числа (т. е. точки или запятой)	25
Пгм-ноВызватьКликКнопки	На экранной форме имеем две кнопки. Щелчок на первой кнопке вызывает появление окна с сообщением о произошедшем событии нажатия первой кнопки. Щелкая на второй кнопке, мы имитируем нажатие первой кнопки путем программного вызова события нажатия первой кнопки	26
TxtUnicode	Программа для чтения/записи текстового файла в кодировке Unicode	27
TXT_1251	Программа для чтения/записи текстового файла в кодировке Windows 1251	28
ТекстовыйРедактор	Простой текстовый редактор	29
Тестирование	Программа тестирует студента по какому-либо предмету обучения	30
RtfРедактор	Программа простейшего RTF-редактора	31
ВводКаталогаКоординат	Чтение текстового файла, содержащего каталог координат. В каждой строке файла должны быть записаны координаты одной точки (четыре числа). При этом в качестве разделителя целой и дробной частей пользователь может использовать точку и/или запятую. Между числами может быть сколько угодно пробельных символов и/или знаков табуляции (`//\t//`). Пользователь может пропускать пустые строки в середине каталога и/или в конце файла. Программа должна "понять" введенный текст и вывести распознанный каталог координат в текстовое поле экранной формы	32
TxtPrint	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	33
ReadWriteBin	Программа для чтения/записи бинарных файлов с использованием потока данных	34
SimpleImage1, SimpleImage2 и SimpleImage3	Три варианта программ, выводящих в форму растровое изображение из графического файла	35
СкроллингБольшогоРисунка	Программа выводит изображение из растрового файла в **PictureBox**, размещенный на элементе управления **Panel**, с возможностью прокрутки изображения	36

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
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**	37
ВыборЦвета1 и ВыборЦвета2	Два варианта программы, меняющей цвет фона формы **BackColor**, перебирая константы цвета, предусмотренные в Visual Studio 12, с помощью элемента управления **ListBox**	38
ПрозрачныйТреугольник	Программирование экранной формы, в которой размещен прозрачный треугольник	39
ПечатьЭллипса	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**)	40
ПечатьBMPфайла	Эта программа выводит на печать файл с расширением bmp	41
Создать_JPG	Программа формирует изображение методами класса **Graphics**, записывает его на диск в формате JPG-файла и выводит его отображение в экранную форму	42
СменаИзображения	Программа выводит в панель рисования текстовую строку. При щелчке на командной кнопке происходит перерисовка изображения с разворотом текстовой строки	43
РисМышью	Программа позволяет при нажатой левой или правой кнопке мыши рисовать в форме	44
Spline	Программа строит сплайн Безье по двум узловым точкам, а две контрольные (управляющие) точки совмещены в одну. Эта одна управляющая точка отображается в форме в виде красного прямоугольника. Перемещая указателем мыши управляющую точку, мы регулируем форму сплайна (кривой)	45
График	Программа рисует график объемов продаж по месяцам. Понятно, что таким же образом можно построить любой график по точкам для других прикладных целей	46
БуферОбменаTXT	Эта программа имеет возможность записи какого-либо текста в буфер обмена, а затем извлечения этого текста из буфера обмена	47
БуферОбменаBitmap	Программа оперирует буфером обмена, когда тот содержит изображение	48
AltPrintScreen	Программная имитация нажатия клавиш <Alt>+<PrintScreen> методом **Send** класса **SendKeys**	49
БуферОбменаSaveBMP	Программа читает буфер обмена, и если данные в нем представлены в формате растровой графики, то записывает их в BMP-файл	50
ПростоTimer	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	51

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
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	52
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 раскрывающегося меню, с помощью которого пользователь выводит сформированную таблицу в Блокнот с целью последующего редактирования и вывода на печать	53
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	54
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 веб-страниц Internet Explorer или другого браузера	55
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	56
ХэшGridView	В данной программе используется структура данных, называемая хэш-таблицей. С ее помощью программа ставит в соответствие государствам их столицы. При этом в качестве ключей указываем названия государств, а в качестве значений — их столицы. Далее, используя элемент управления **DataGridView**, программа выводит эту хэш-таблицу в форму	57
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 D:\tabl.xml. При последующих запусках данной программы таблица будет считываться из этого файла, и пользователь может продолжать редактирование таблицы	58
ГауссGrid	Программа для решения системы линейных уравнений. Ввод коэффициентов предусмотрен через **DataGridView**	59
СвязанныеТаблицы	Программа создает таблицу с данными о клиентах (названия организаций, контакты) и таблицу с данными о заказах (номер заказа, его объем, организация-заказчик). Между таблицами устанавливается связь посредством одинаковых столбцов (названий организаций). При этом таблица с данными о клиентах будет родительской, а таблица с данными о заказах — дочерней. Каждая строка родительской таблицы отображается со знаком "плюс". При щелчке на знаке "плюс" открывается узел, содержащий ссылку на дочернюю таблицу	60

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
ГрафикChart	Программа, используя элементы управления **Chart** и **DataGridView**, выводит график (диаграмму) зависимости объемов продаж от времени по месяцам. При этом в качестве источника данных указываем объект класса **DataTable**	61
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)	62
FlashWeb	Программа использует элемент управления **WebBrowser** для отображения Flash-файлов	63
Split	Эта программа использует элемент управления **WebBrowser** для отображения веб-страницы и ее HTML-кода	64
ЗаполнениеВебФормы	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 веб-страницы по их именам. Заполняем поле поиска ключевыми словами для нахождения соответствующих веб-страниц, а затем для отправки заполненной формы на сервер "программно" нажимаем кнопку **Submit**. В итоге получим в элементе **WebBrowser** результат работы поисковой системы — множество ссылок на страницы, содержащие указанные ключевые слова	65
РазборВебСтраницы	Программа, используя класс **WebClient**, читает веб-страницу Центрального банка РФ **www.cbr.ru**, ищет в ее гипертекстовой разметке курс доллара США и копирует его в текстовую метку **Label**. Кроме того, элемент управления графическое поле **PictureBox** отображает логотип банка, используя URL-адрес этого изображения	66
Орфография1 и Орфография2	Два варианта программы, позволяющей 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	67
ТаблицаWord	Программа вывода таблицы средствами MS Word: запускается программа, пользователь наблюдает, как запускается редактор MS Word и автоматически происходит построение таблицы	68
ExcelПи	Программа обращается к одной простой функции объектной библиотеки MS Excel для получения значения числа Пи = 3,14...	69
ExcelПлт	Программа использует финансовую функцию `Pmt()` объектной библиотеки MS Excel для вычисления суммы периодического платежа на основе постоянства сумм платежей и постоянства процентной ставки	70

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
ExcelСЛАУ	Программа решает систему уравнений с помощью функций объектной библиотеки MS Excel	71
ExcelГрафик	Программа строит диаграмму (график), используя объекты компонентной библиотеки MS Excel	72
ACADЭлементарныйЧертеж	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 2008, рисует два отрезка, одну дугу и два текстовых объекта, сохраняет чертеж в файле D:\Чертеж.dwg и завершает работу AutoCAD	73
MatlabВызов	Программа вызывает простейшую функцию **Matlab**	74
MatlabСлау	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**	75
Создать_PDF_1 и Создать_PDF_2	Первая программа "на лету" генерирует PDF-документ с английским текстом, а вторая — с русским	76
Создать_PDF_Табл_1 и Создать_PDF_Табл_2	Два варианта программ, создающих "на лету" PDF-файл и записывающих в этот файл таблицу данных	77
Создать_PDF_граф_1 и Создать_PDF_граф_2	Два варианта программ, выводящих в PDF-документ изображение	78
БД_SQL_Server	Создание базы данных SQL Server в среде Visual Studio 12. 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**	79
БД_SQL_Server_Консоль	Программа читает все записи из таблицы БД SQL Server (файл *.sdf) и выводит их на консоль с помощью объектов **Command** и **DataReader**	80
БДDataReader	Программа читает все записи из таблицы БД MS Access и выводит их на консоль с помощью объектов **Command** и **DataReader**	81
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 на Visual C#, так и используя MS Access. В этом примере технология ADO.NET не использована	82
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	83
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	84

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
БдReaderGridView	Программа читает все записи из таблицы базы данных с помощью объектов **Command**, **DataReader** на элемент управления **DataGridView** (Сетка данных)	85
БдАдаптерGridView	Программа читает из БД таблицу в сетку данных **DataGridView** с использованием объектов класса **Command**, **Adapter** и **DataSet**	86
БдUpdate	Программа обновляет записи (`Update`) в таблице базы данных MS Access	87
БдУдаленЗаписи	Программа удаляет запись из таблицы БД с использованием SQL-запроса и объекта класса **Command**	88
LinqМассив	Программа в строковом массиве имен выбирает имена, состоящие из шести букв. В списке выбранных имен сортируем их в алфавитном порядке, все строки переводим в верхний регистр и избавляемся от дублирования имен. При этом используем синтаксис LINQ-запросов	89
LinqСписок	Три LINQ-запроса к спискам данных	90
LinqМесяцы1 и LinqМесяцы2	Программы выполняют группировку некоторых данных. То есть список данных данные делят на две группы по заданному критерию	91
LinqЦеныНаПродукты	Программа создает словарь данных **Dictionary** продуктов питания. В словаре — всего два поля: наименование товара и его цена. Методом **GroupBy** из данных словаря создаем две ценовых группы: товары дороже и дешевле 90 рублей. Обе группы товаров выводим на консоль	92
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	93
ПоискXmlЭлемента1 и ПоискXmlЭлемента2	Дана строка XML, содержащая прогнозные метеорологические показатели для Москвы на заданную дату. Варианты программ извлекают из корневого элемента XML-документа значение температуры элемента "Температура"	94
ПоискСтрокВXml	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 методов классов пространства имен `System.Xml.Linq`	95
LinqBhv	Программа читает XML-файл (источник), содержащий сведения о книгах по программированию, и выбирает книги автора Зиборова. Выбранные книги программа записывает в другой (производный) XML-файл	96

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
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, то программа отображает в элемент **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 поиск городов-миллионеров в наборе данных **DataSet** искомой таблицы. Результат запроса выводится в текстовое поле	97
ПроверкаФамилии и ПроверкаЧисла	Проверка данных, вводимых пользователем, на достоверность	98
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.	99
Гринвич	Программа в полупрозрачной экранной форме отображает текущее время по Гринвичу и при этом не закрывает собой другие приложения	100
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	101
ПеремещениеФормы	Нестандартная форма. Программа позволяет перемещать форму мышью, "зацепив" ее не только за заголовок, а в любом месте формы	102
Звуки_ОС	Программа воспроизводит некоторые звуки операционной системы Windows	103
Player	Программа реализует функции проигрывателя Windows Media Player 11	104
PlayerТолькоЗвук	Программа предназначена для воспроизведения только звуковых файлов	105
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	106
Summa	Создание простейшей активной веб-страницы на Visual C# 12. Веб-страница демонстрирует способность складывать числа, введенные пользователем	107

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
Valid1	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`	108
Validations	Проверка достоверности ввода имени, адреса e-mail, URL-адреса и пароля. Веб-страница проверяет ввод пользователем имени, e-mail-адреса, URL-адреса и пароля, например, при регистрации пользователя. Причем если страница (веб-форма) успешно прошла все этапы проверки, то направляем пользователя на другую, уже разрешенную для этого пользователя, веб-страницу	109
Login	Регистрация и аутентификация пользователя с помощью базы данных MS Access. Данный пример включает в себя три веб-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	110
tab	Таблица с переменным числом ячеек, управляемая двумя раскрывающимися списками. Веб-страница позволяет с помощью двух раскрывающихся списков **DropDownList** заказать необходимое число рядов и столбцов в таблице, а затем строить заказанную таблицу	111
Menu	Веб-страница демонстрирует, как можно организовать переход на разные страницы сайта (гиперссылки) с помощью раскрывающегося списка **DropDownList**	112
Ссылка	Передача данных между веб-страницами через параметры гиперссылки. В данном примере имеем две веб-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	113
Передача1	Веб-страница получает имя пользователя и адрес его электронной почты от веб-формы и отображает эти данные на странице	114
Передача2	На начальной веб-странице имеем командную кнопку **ПЕРЕХОД** и текстовое поле, которое заполняет пользователь. После щелчка на кнопке происходит переход на другую веб-страницу. На новой странице отображается содержимое текстового поля и надпись на кнопке из предыдущей страницы	115
TabGrdWeb	Вывод табличных данных в веб-форму с помощью элемента управления **GridView**. В данной веб-странице организован вывод двух строковых массивов в таблицу в веб-форму с помощью элемента управления **GridView** и объекта класса **DataTable**	116

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
Hash_Grid	Вывод в веб-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 веб-страницу	117
RW_txt	Чтение/запись текстового файла веб-приложением. Веб-приложение читает текстовый файл в текстовое поле, а пользователь имеет возможность редактировать текст и сохранять его в том же файле	118
Counter	Веб-приложение, реализующее счетчик посещений сайта с использованием базы данных и объекта **Session**	119
Cookie	Чтение/запись cookie-файлов. Веб-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	120
ВебИзображение	На странице имеем изображение — файл poryv.png, при щелчке мышью на нем изображение увеличивается вдвое без перезагрузки веб-страницы. В этом же проекте решена также другая задача. На веб-странице имеем изображение, например мужчины, — файл m.jpg. Это изображение используем для ссылки на другую веб-страницу, например на WebForm1.aspx. Причем при наведении на него указателя мыши происходит смена изображения на изображение женщины — файл g.jpg	121
ТекстНаклWeb	Веб-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	122
ГостеваяКнига	Данная веб-страница приглашает посетителя оставить какие-либо записи, которые могут прочитать другие посетители страницы. Записи сохраняются в текстовом файле kniga.txt. Записи отображаются на веб-странице с помощью элемента **GridView** (Сетка данных)	123
AjaxTimer	Веб-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	124
ВебКлиентПогода	Веб-приложение, потребляющее сервис удаленной веб-службы прогноза погоды. Приложение в текстовом поле **TextBox** демонстрирует XML-строку с параметрами погоды для города, указанного во входных параметрах при обращении к веб-службе. Также выводит в текстовую метку значение температуры в этом городе	125

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
WindowsКлиентПогода	Windows-приложение, потребляющее сервис удаленной веб-службы прогноза погоды. Приложение в текстовом поле **TextBox** демонстрирует XML-строку с параметрами погоды для города, указанного во входных параметрах при обращении к веб-службе. Также выводит в текстовую метку значение температуры в этом городе	126
ВебСлужбаСумма	На вход данной веб-службы предлагается ввести два числа, а веб-служба берет на себя функцию сложения этих двух чисел и вывода (возврата) суммы. При этом веб-служба производит диагностику вводимых данных	127
ВебКлиентСумма	Клиентское Windows-приложение, потребляющее сервис веб-службы предыдущего примера *ВебСлужбаСумма*	128
ВебСлужбаForex	Веб-служба, которая с помощью синтаксического разбора веб-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 веб-службы в виде строки	129
ВебКлиентForex	Получаем прогноз рынка Forex на текущий день. Клиентское Windows-приложение, потребляющее сервис веб-службы предыдущего примера *ВебСлужбаForex*	130
ВебКлиентMorpher	Клиентское веб-приложение, потребляющее сервис веб-службы "Морфер". На вход метода веб-службы подаем сумму в рублях цифрами, метод же возвращает сумму прописью	131
ВебКлиентРоссия	Клиентское веб-приложение, потребляющее сервис веб-службы Центрального банка России для получения ежедневных курсов валют. На выходе приложения получаем таблицу курсов валют	132
ВебКлиентБеларусь	Клиентское Windows-приложение, потребляющее сервис веб-службы Национального банка Республики Беларусь для получения ежедневных курсов валют. На выходе приложения получаем таблицу курсов валют	133
WCF-Служба	Веб-служба будет предлагать пользователю ввести свое имя, а в ответ WCF Service будет возвращать пользователю строку "<Введенное имя>, вы увлекаетесь WCF?"	134
КлиентWCFСлужбы	Windows-приложение, которое обращается к WCF-службе, созданной в предыдущем примере	135
WpfSqrt	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 на текстовую метку выводит результат извлечения корня	136
WpfАниме	WPF-приложение выводит на веб-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	137

Таблица П1 *(окончание)*

Название папки	**Описание программы**	**Номер примера**
WpfДваИзо	WPF-приложение содержит на веб-странице два изображения. Поскольку месторасположение обоих изображений задано одинаково, а также совпадают размеры изображений, пользователь будет видеть только "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	138
WpfГрадиент	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 ставится в соответствие цвета: желтый, красный, синий и зеленый. Между этими цветами задается плавный переход от одного цвета к другому с помощью градиента	139
WpfОрфография	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	140
WpfПроигрыватель	WPF-проигрыватель, позволяющий воспроизводить мультимедиа, включать паузу, остановку, а также настраивать громкость с помощью "ползунка"	141
WpfТекстНаВидео	Программа воспроизводит видеоролик с помощью элемента **MediaElement** и накладывает на него форматированный текст "ПОЛЕТ"	142
WpfПереходы	Данное WPF-приложение содержит две веб-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 веб-страницу Page2.xaml. Возврат со второй страницы на первую организован также с помощью гиперссылки **Hyperlink**	143

Предметный указатель

A

ADO.NET 178, 256
AJAX 399
ASP.NET 337

C

Cookie 384
◊ чтение/запись 384

D

DataSet 178
DataTable 179

M

Microsoft Access 261
◊ создание БД 253
Microsoft Excel 216, 218, 221, 224
Microsoft Word 213

R

RTF-редактор 102, 106

S

Screen shot 156
Simple Object Access Protocol (SOAP) 401
SQL Server
◊ база данных: создание 249
◊ таблица: отображение 251

W

Windows Media Player 319, 321, 325

А

Аутентификация пользователя 345

Б

База данных
◊ MS Access
▫ регистрация и аутентификация пользователя 345
▫ создание 253, 258
◊ SQL Server
▫ создание 249
▫ чтение записей на консоль 256
Буфер обмена
◊ изображение 153
◊ текст 151

В

Валидатор 337, 340
Веб-сервис 401

Веб-служба 401
◊ "Морфер" 421
◊ "Прогноз погоды" 402, 409
◊ "Торговая рекомендация на рынке Forex" 417, 428
◊ валют 424, 426, 434, 439, 442
◊ создание 411
Веб-хостинг 333
Видимость, свойство 39
Вкладка 36, 136, 141, 145
Время
◊ вывод в веб-форму 398
◊ вывод в Windows-форму 312

Г

Гостевая книга 394
График 224
Графический примитив 124, 130, 132

Д

Данные
◊ передача с веб-страницы на веб-страницу 362, 366
◊ проверка 307
◊ проверка с помощью валидатора 337
Диаграмма 224

И

Изображение
◊ вывод в форму 119
◊ смена в веб-форме 390
◊ увеличение в веб-форме 388
◊ формирование на веб-странице 391

К

Калькулятор 69
Клавиатура 76, 79, 82
Кнопка: создание программное 66
Кодировка
◊ Unicode 85
◊ ANSI 88
Комбинированный список 41
Курсор мыши 63

М

Метод 19
◊ OnPaint 119

Н

Навигация по веб-страницам 356

О

Объект 17

П

Перевод каретки 40
Переключатель 36
Печать
◊ графического примитива 130, 132
◊ графического файла 133
◊ файла 110
Приложение консольное 49
Проверка вводимых данных 307

Р

Регистрация пользователя 345

С

Свойство 17
◊ Visible 39
Сеанс пользователя 383
Сервер 333
Сессия 383
Система
◊ линейных алгебраических уравнений 221
◊ линейных уравнений 181
Снимок экрана 156
Событие 68
◊ клавиатуры 76, 79, 82
◊ от разных объектов 69
Справочная система 327
Счетчик сообщений сайта 379

Т

Таблица 163
◊ MS Access
▫ заполнение 263
▫ редактирование в VB2010 254
◊ MS Word 213
◊ SQL Server
▫ отображение в форме 251
▫ в веб-форме 369
▫ вывод в Internet Explorer 169

- обновление записей 269
- переменное число ячеек 354
- создание в пустой БД 261
- удаление записей 273
- чтение записей 265

Текстовый редактор 91
Точка останова 30

Ф

Файл
- графический
  - вывод в форму 119
  - печать 133
- двоичный
  - чтение/запись 114
- открытие 91
- сохранение 91
- текстовый
  - печать 110
  - чтение/запись 85, 375
- формат
  - ASMX 401
  - CHM 329
  - SWF 197

Фигура
- печать 130, 132
- рисование на форме 124

Флажок 32
Форма 16
- нестандартная 317
- перемещение мышью 317
- прозрачность 310

Х

Хост 332
Хостинг 333
Хэш-таблица 174, 370

Ц

Цвет 127

Э

Элемент управления 68
- Button 65
- CheckBox 32
- ComboBox 41
- DataGridView 177, 265
- DropDownList 356
- GridView 369
- HyperLink 359
- ListBox 63
- Microsoft Web Browser 153
- PictureBox 153
- RadioButton 36
- TabControl 36
- TextBox 27
- Timer 161
- WebBrowser 195