MCAD/MCSD

Training Kit

Exam 70-305 and 70-315

DEVELOPING WEB APPLICATIONS WITH MICROSOFT VISUAL BASIC .NET AND MICROSOFT VISUAL C# .NET.

Microsoft Press

Учебный
курс
MCAD/MCSD

**Сертификационный
экзамен 70-305 и 70-315**

Разработка Web-приложений на Microsoft Visual Basic .NET и Microsoft Visual C# .NET.

*Официальное пособие Microsoft
для самостоятельной подготовки*

Москва, **2003**

РУССКАЯ РЕДАКЦИЯ

УДК 004
ББК 32.973.26-018.2
М59

Microsoft Corporation

М59 Разработка Web-приложений на Microsoft Visual Basic .NET и Microsoft Visual C# .NET. Учебный курс MCAD/MCSD/Пер. с англ. — М.: Издательско-торговый дом «Русская Редакция», 2003. — 704 стр.: ил.

ISBN 5–7502–0221–6

Этот учебный курс посвящен разработке Web-приложений на Microsoft Visual Basic .NET и Microsoft Visual C# .NET. В книге изложены основы программирования для Web, методы создания пользовательского интерфейса и работы с базами данных. Значительное внимание уделено вопросам защиты, тестирования и развертывания Web-приложений, а также их глобализации и созданию справочной системы.

Книга адресована всем тем, кто хочет научиться создавать Web-приложения среднего и высокого уровня сложности в Visual Studio .NET на языках Visual Basic .NET и C#, а также тем, кто собирается самостоятельно подготовиться к сдаче экзаменов по программе сертификации Microsoft (сертификаты Microsoft Certified Application Developer и Microsoft Certified Solution Developer) № 70-305: Developing and implementing Web Applications with Microsoft Visual Basic .NET and Microsoft Visual Studio .NET и 70-315: Developing and Implementing Web Applications with Microsoft Visual C# .NET and Microsoft Visual Studio .NET.

Книга состоит из 15 глав, словаря терминов и предметного указателя, содержит множество иллюстраций и примеров программного кода. На прилагаемом компакт-диске записаны демонстрационные файлы, задания практических работ и другие материалы.

УДК 004
ББК 32.973.26-018.2

Подготовлено к изданию по лицензионному договору с Microsoft Corporation, Редмонд, Вашингтон, США.
ActiveX, FrontPage, IntelliSense, JScript, Microsoft, Microsoft Press, PhotoDraw, Visual Basic, Visual C++, Visual C#, Visual Studio, Windows, Windows Media Player и Windows NT являются товарными знаками или охраняемыми товарными знаками корпорации Microsoft в США и/или других странах. Все другие товарные знаки являются собственностью соответствующих фирм.

Все названия компаний, организаций и продуктов, а также имена лиц, используемые в примерах, вымышлены и не имеют никакого отношения к реальным компаниям, организациям, продуктам и лицам.

© Оригинальное издание на английском языке. Microsoft Corporation, 2002
© Перевод на русский язык, Microsoft Corporation, 2003
© Оформление и подготовка к изданию, издательско-торговый дом «Русская Редакция», 2003

ISBN 0-7356-1581-0 (англ.)
ISBN 5-7502-0221-6

Microsoft Corporation

Разработка Web-приложений на Microsoft Visual Basic .NET и Microsoft Visual C# .NET.
Учебный курс MCAD/MCSD

Перевод с английского под общей редакцией А. Е. **Соловченко**

Редактор Ю. П. **Леонова**

Компьютерный дизайн и подготовка иллюстраций В. Б. **Хильченко**

Дизайнер обложки Е. В. Козлова

Главный редактор А. И. Козлов

Подготовлено к печати издательством «Русская Редакция»
121087, Москва, ул. Заречная, д.9
тел.: (095) 142-0571, тел./факс: (095) 145-4519
e-mail: info@rusedit.ru, http://www.rusedil.ru

РУССКАЯ РЕДАКЦИЯ

Подписано в печать 20.05.2003 г. Тираж 2000 экз.
Формат 70x100 1/16 Физ. п. л. 44

Отпечатано в ОАО «Типография "Новости"»
107005, Москва, ул. Фр. Энгельса, 46

Содержание

Об этой книге . . . XIV

Глава 1. Введение в Web-программирование . . . 1
Занятие 1. Типы Интернет-приложений . . . 2
Четыре типа Интернет-приложений . . . 2
Принцип работы Web-приложений . . . 3
Возможности ASP.NET . . . 5
Преимущества ASP.NET . . . 5
Занятие 2. Использование ASP.N ET . . . 7
Структура Web-приложений . . . 7
Компоненты Web-форм . . . 9
Структура .NET Framework . . . 9
Языки программирования . . . 13
Занятие 3. Работа в Visual Studio .NET . . . 16
Начальная страница Visual Studio .NET . . . 16
Типы окон Visual Studio .NET . . . 19
Запуск проекта . . . 26
Установка точек прерывания и просмотр значений переменных . . . 28
Пошаговое исполнение . . . 29
Работа со справочной системой . . . 30
Резюме . . . 31
Практикум 1. Введение в Visual Studio .NET . . . 33
Упражнение 1. Настройка среды Visual Studio .NET . . . 33
Упражнение 2. Создание учетной записи службы хостинга . . . 34
Упражнение 3. Выбор провайдера услуг хостинга . . . 35
Упражнение 4. Работа с учебниками ASP.NET QuickStart . . . 38
Закрепление материала . . . 40

Глава 2. Создание приложений Web Forms . . . 41
Занятие 1. Создание проекта Web-приложения . . . 42
Создание нового Web-приложения . . . 42
Управление проектом при помощи IIS . . . 43
Размещение элементов управления и написание кода . . . 47
Файлы проекта Web Forms . . . 49
Файлы Web-приложения . . . 50
Занятие 2. Обработка событий . . . 52
События жизненного цикла Web-приложения . . . 52
Сохранение данных Web-формы . . . 53
События Application и Session . . . 55
События Web-формы . . . 57
События серверных элементов управления . . . 58
Занятие 3. Обработка данных . . . 61
Установка границ приложения . . . 61
Управление процессами . . . 63
Определение момента завершения приложения . . . 63
Резюме . . . 65
Практикум 2. Создание простого Web-приложения . . . 66
Упражнение 1. Создание пользовательского интерфейса . . . 66

Упражнение 2. Создание класса FlashCardClass . . . 67
Упражнение 3. Сохранение объекта FlashCardClass в переменной состояния Session . . 70
Упражнение 4. Обработка событий Web-формы с использованием класса FlashCardClass . . . 71
Закрепление материала . . . 74

Глава 3. Работа с Web-объектами . . . 75
Занятие 1. Основы пространств имен . . . 76
Знакомство с пространством имен . . . 76
Классы и модули . . . 82
Лицом к интерфейсу . . . 98
Занятие 2. Пространства имен в Web-приложениях . . . 101
Обзор пространства имен Web . . . 101
Использование объекта Application . . . 102
Использование объекта Page . . . 104
Использование объекта Request . . . 105
Использование объекта Response . . . 107
Занятие 3. Сохранение сведений о состоянии . . . 109
Способы сохранения сведений о состоянии . . . 109
Использование строк запросов . . . 109
Использование файлов cookie . . . 110
Использование состояния отображения . . . 112
Использование переменных состояния Application и Session . . . 113
Резюме . . . 116
Практикум 3. Работа с объектами и сохранение данных . . . 117
Упражнение 1. Создание пользовательского интерфейса . . . 117
Упражнение 2. Создание класса TranslatorClass . . . 118
Упражнение 3. Сохранение объекта Translator в переменной состояния Session . . . 121
Упражнение 4. Обработка событий Web-формы с помощью объекта TranslatorClass . . . 121
Закрепление материала . . . 124

Глава 4. Создание пользовательского интерфейса . . . 125
Занятие 1. Использование элементов управления . . . 126
Выбор способа размещения элементов управления . . . 126
Выбор подходящего элемента управления . . . 128
Работа с текстом . . . 130
Работа с таблицами и списками . . . 132
Исполнение команд . . . 141
Получение и установка значений . . . 145
Отображение графики и графической рекламы . . . 147
Группировка элементов управления . . . 149
Получение дат . . . 151
Получение файлов от клиента . . . 152
Занятие 2. Проверка вводимых данных . . . 156
Использование проверки вводимых данных . . . 156
Комбинирование верификаторов . . . 159
Отмена проверки введенных данных . . . 160
Нестандартная проверка вводимых данных . . . 161
Занятие 3. Переход между формами . . . 164
Методы перехода между страницами . . . 164
Отображение страницы в новом окне браузера . . . 169
Резюме . . . 172

Практикум 4. Создание текстового редактора на основе Web-форм . . . 173
Упражнение 1. Создание входной формы приложения . . . 173
Упражнение 2. Создание формы NewAccount . . . 176
Упражнение 3. Создание формы File Manager . . . 180
Упражнение 4. Создание формы File Editor . . . 184
Упражнение 5. Создание Web-формы Not Authorized . . . 189
Закрепление материала . . . 190

Глава 5. Хранение и извлечение данных с помощью ADO.NET . . . 191
Занятие 1. Доступ к данным с помощью ADO.NET . . . 192
Знакомство с ADO.NET . . . 192
Подключение к БД . . . 194
Создание набора данных . . . 196
Отображение набора данных . . . 197
Модификация записей в базе данных . . . 200
Обращение к объектам в ADO.NET . . . 200
Добавление, модификация и удаление строк . . . 201
Запись содержимого объекта DataSet в базу данных . . . 203
Создание соединения с базой данных во время выполнения . . . 205
Обновление базы данных во время выполнения . . . 207
Занятие 2. Использование наборов данных в Web-формах . . . 209
Отображение набора данных с помощью DataGrid . . . 209
Отображение набора данных с помощью DataList . . . 211
Отображение элементов данных в других элементах управления, представляющих списки . . . 215
Выборка отдельных записей . . . 217
Исполнение команд над базой данных . . . 218
Извлечение значений из базы данных . . . 219
Модификация записей непосредственно в базе данных . . . 220
Извлечение записей непосредственно из базы данных . . . 221
Занятие 3. Обработка транзакций . . . 223
Введение в транзакции . . . 223
Использование транзакций с наборами данных . . . 224
Использование транзакций с базами данных . . . 227
Корпоративные транзакции . . . 232
Резюме . . . 233
Практикум 5. Создание диспетчера контактов . . . 234
Упражнение 1. Создание проекта и формы SwitchBoard . . . 234
Упражнение 2. Добавление компонентов, необходимых для работы с базой данных . . . 235
Упражнение 3. Создание формы AddContact . . . 237
Упражнение 4. Создание формы Calls . . . 243
Упражнение 5. Создание форм DeleteContact и ContaclTypes . . . 252
Закрепление материала . . . 252

Глава 6. Перехват и исправление ошибок . . . 253
Занятие 1. Обработка исключений . . . 254
Способы обработки исключений . . . 254
Генерация исключений . . . 260
Определение новых исключений . . . 262
Использование событий Error . . . 264
Занятие 2. Страницы сообщений об ошибках . . . 267
Обработка исключений, связанных с работой Интернета . . . 267

Использование страниц сообщений об ошибках на уровне приложения . . . 268
Использование страниц сообщений об ошибках на уровне страницы . . . 270
Занятие 3. Регистрация исключений . . . 272
Регистрация ошибок с помощью трассировки . . . 272
Включение и выключение трассировки . . . 272
Запись сообщений в журнал трассировки . . . 274
Чтение журнала трассировки . . . 276
Резюме . . . 277
Практикум 6. Организация обработки исключений в приложении . . . 278
Упражнение 1. Усовершенствование обработки исключений . . . 278
Упражнение 2. Добавление обработчика события Error . . . 281
Упражнение 3. Создание и просмотр журнала трассировки . . . 283
Упражнение 4. Организация обработки исключений в других Web-формах . . . 284
Закрепление материала . . . 284

Глава 7. Усложненное программирование с применением Web Forms . . . 285
Занятие 1. Сохранение и извлечение сведений о пользователе . . . 286
Идентификация пользователей Web-приложения . . . 286
Хранение сведений о пользователе на клиентском компьютере . . . 287
Хранение сведений о пользователе на сервере . . . 291
Создание уникальных ключей для идентификации пользователей . . . 291
Занятие 2. Отправка почты . . . 298
Отправка почты с клиентского компьютера . . . 298
Отправка почты с сервера . . . 299
Занятие 3. Применение фреймов . . . 302
Создание фреймов в Visual Studio . . . 302
Проверка наличия поддержки фреймов в браузере . . . 305
Использование ссылок для перехода между фреймами . . . 306
Занятие 4. Применение клиентских сценариев . . . 308
Что нужно учесть перед применением сценариев . . . 308
Выбор языка сценариев . . . 310
Знакомство с DOM . . . 310
Добавление встроенных сценариев . . . 310
Обработка событий с помощью сценариев . . . 314
Занятие 5. Использование Web-сервисов XML . . . 319
Поиск Web-сервисов XML . . . 319
Использование Web-сервисов XML . . . 320
Использование Web-сервисов XML в клиентских сценариях . . . 321
Резюме . . . 324
Практикум 7. Создание приложения Advanced Features . . . 325
Упражнение 1. Проверка наличия поддержки дополнительных возможностей . . . 325
Упражнение 2. Сохранение сведений о пользователе . . . 328
Упражнение 3. Создание Web-формы Mail . . . 331
Упражнение 4. Создание пользовательского интерфейса на основе фреймов . . . 334
Упражнение 5. Дополнительные упражнения по применению сценариев и файлов cookie . . . 336
Закрепление материала . . . 337

Глава 8. Обеспечение безопасности . . . 339
Занятие 1. Аутентификация и авторизация пользователей . . . 340
Анонимный доступ . . . 340
Доступ аутентифицированных пользователей . . . 342

Аутентификация запросов HTML-страниц . . . 343
Занятие 2. Аутентификация Windows . . . 345
Включение аутентификации Windows . . . 345
Назначение прав доступа отдельным пользователям . . . 348
Применение авторизации, основанной на ролях . . . 348
Извлечение идентификационных данных пользователя . . . 349
Параметры IIS и аутентификация средствами Windows . . . 350
Занятие 3. Аутентификация Forms . . . 352
Как задействовать аутентификацию с помощью форм . . . 352
Создание входной Web-формы . . . 354
Занятие 4. Аутентификация Passport . . . 363
Как задействовать аутентификацию Passport . . . 363
Занятие 5. Защита данных при пересылке . . . 368
Защита взаимодействия с помощью SSL . . . 368
Генерация запроса на выдачу сертификата . . . 369
Запрос сертификата . . . 370
Установка сертификата . . . 371
Использование защищенного подключения . . . 372
Резюме . . . 374
Практикум 8. Создание защищенного приложения . . . 375
Упражнение 1. Установка режима аутентификации . . . 375
Упражнение 2. Создание Web-формы Login . . . 376
Упражнение 3. Отображение сведений о пользователе . . . 383
Упражнение 4. Усложненные задания . . . 385
Закрепление материала . . . 386

Глава 9. Компоновка и развертывание Web-приложений . . . 387
Занятие 1. Компоновка Web-приложения . . . 388
Установка параметров компоновки . . . 388
Ввод сведений о приложении . . . 389
Конфигурирование приложения . . . 391
Занятие 2. Развертывание Web-приложения . . . 393
Загрузка приложений на сервер провайдера . . . 393
Развертывание приложения на собственном сервере . . . 393
Занятие 3. Сопровождение приложения после развертывания . . . 400
Наблюдение за работой сервера . . . 400
Восстановление приложения во время его эксплуатации . . . 411
Настройка развернутых приложений . . . 412
Советы по оптимизации . . . 415
Занятие 4. Развертывание приложения на нескольких серверах . . . 416
Масштабирование путем наращивания числа процессоров . . . 416
Масштабирование путем наращивания числа серверов . . . 417
Резюме . . . 419
Практикум 9. Компоновка и развертывание Web-приложения . . . 420
Упражнение 1. Подготовка приложения . . . 420
Упражнение 2. Развертывание приложения . . . 420
Упражнение 3. Отображение сведений о производительности приложения на графике . . . 420
Упражнение 4. Настройка приложения . . . 422
Закрепление материала . . . 422

Глава 10. Тестирование Web-приложений . . . 423
Занятие 1. Подготовка тестирования . . . 424
Составление плана тестирования . . . 424
Типы тестирования . . . 425
Блочное тестирование . . . 425
Комплексное тестирование . . . 429
Регрессивное тестирование . . . 430
Нагрузочное тестирование . . . 430
Тестирование па платформе . . . 434
Занятие 2. Проведение тестирования . . . 436
Тестирование с помощью ACT . . . 436
Тестирование .NET-сборок . . . 442
Автоматическая компоновка и тестирование . . . 445
Просмотр файлов журнала . . . 449
Занятие 3. Отладка приложения . . . 450
Применение результатов тестирования . . . 450
Использование классов Debug и Trace . . . 454
Удаленная отладка . . . 456
Резюме . . . 459
Практикум 10. Тестирование и отладка . . . 460
Упражнение 1. Анализ плана тестирования . . . 460
Упражнение 2. Блочное тестирование . . . 460
Упражнение 3. Нагрузочное тестирование . . . 461
Упражнение 4. Пошаговое исполнение Web-приложения на удаленном сервере . . . 461
Закрепление материала . . . 462

Глава 11. Создание нестандартных Web-элементов управления . . . 463
Занятие 1. Создание пользовательских элементов управления . . . 465
Создание и применение пользовательских элементов управления . . . 465
Использование элементов управления в коде . . . 471
Объявление событий пользовательского элемента управления . . . 472
Включение сеточной разметки . . . 474
Занятие 2. Создание составных элементов управления . . . 476
Создание и применение составных элементов управления . . . 476
Определение вида составного элемента управления . . . 484
Создание свойств и методов . . . 488
Обработка событий . . . 491
Генерация событий . . . 493
Изменение размеров элемента управления . . . 495
Создание производных элементов управления . . . 495
Занятие 3. Создание генерируемых элементов управления . . . 498
Создание и использование генерируемых элементов управления . . . 498
Обработка действий пользователя . . . 505
Прием данных от пользователя . . . 511
Добавление нестандартных элементов управления на панель Toolbox . . . 515
Резюме . . . 517
Практикум 11. Создание нестандартных элементов управления . . . 518
Упражнение 1. Создание пользовательского элемента управления AddressBox . . . 518
Упражнение 2. Создание составного элемента управления DigitPad . . . 519
Упражнение 3. Создание генерируемого элемента управления BarChart . . . 519
Закрепление материала . . . 520

Глава 12. Работа с мультимедиа 521
Занятие 1. Воспроизведение звука 522
Способы воспроизведения звука 522
Воспроизведение фоновых звуков 522
Внедрение звуковых файлов 523
Добавление звуков в виде объектов ActiveX 524
Работа с объектами ActiveX в Netscape Navigator 529
Занятие 2. Воспроизведение видео 532
Способы воспроизведения видео 532
Работа с разными версиями Media Player 536
Занятие 3. Анимация на Web-страницах 538
Способы отображения анимированного содержимого 538
Анимация текста 539
Анимация простых графических элементов 541
Анимация с помощью объектов и подключаемых модулей 542
Анимация с помощью клиентских сценариев 542
Анимация с помощью HTML+TIME 544
Анимация с помощью атрибутов 548
Резюме 551
Практикум 12. Создание мультимедийного приложения 552
Упражнение 1. Использование звука и движущихся объектов 552
Упражнение 2. Создание слайд-шоу 553
Закрепление материала 554

Глава 13. Форматирование данных, генерируемых Web-приложением 555
Занятие 1. Применение каскадных таблиц стилей 556
Как работают стили 556
Применение таблиц стилей с Web-формами 558
Переключение таблиц стилей во время выполнения 563
Использование динамических элементов стиля 566
Занятие 2. Использование XSL-преобразований 570
Как работают XS L-преобразования 570
Создание XML-файла 570
Создание XSL-файла 573
Изменение разметки 574
Создание дополнительных шаблонов 575
Выполнение повторяющихся действий 578
Добавление гиперссылок и других HTML-элементов 579
Сортировка элементов 581
Выполнение условных действий 581
Резюме 583
Практикум 13. Создание книги рецептов 584
Упражнение 1. Создание и подключение таблицы стилей 584
Упражнение 2. Создание альтернативных таблиц стилей 585
Упражнение 3. Исполнение XSL-преобразования 587
Упражнение 4. Самостоятельная работа 588
Закрепление материала 588

Глава 14. Создание справочной системы 589
Занятие 1. Создание справочной системы в Web-приложениях 590
Отображение всплывающих подсказок 590
Отображение справочных сведений на отдельных Web-формах и HTML-страницах . . . 591

Отображение справочной информации в формате HTML Help . . . 593
Занятие 2. Применение HTML Help Workshop . . . 595
Создание справочной системы в формате HTML Help . . . 595
Установка параметров проекта . . . 597
Создание оглавления . . . 598
Добавление элементов указателя . . . 602
Добавление перекрестных ссылок . . . 603
Организация поддержки поиска . . . 606
Компиляция и предварительный просмотр справочной системы . . . 606
Занятие 3. Подключение справочной системы к Web-формам . . . 607
Применение HTML Help Viewer . . . 607
Отображение справочных файлов в браузере . . . 608
Отображение контекстно-зависимой справки . . . 609
Резюме . . . 611
Практикум 14. Создание и применение справочных файлов HTML Help . . . 612
Упражнение 1. Создание проекта справочной системы . . . 612
Упражнение 2. Создание оглавления, указателя и поддержки поиска . . . 612
Упражнение 3. Добавление ключевых слов к указателю . . . 614
Упражнение 4. Подключение справочной системы к Web-форме . . . 615
Закрепление материала . . . 616

Глава 15. Глобализация Web-приложений . . . 617
Занятие 1. Выбор способа . . . 618
Способы глобализации Web-приложений . . . 618
Определение культуры, к которой принадлежит пользователь . . . 619
Использование нескольких версий Web-приложения . . . 620
Приспособление к текущей культуре во время выполнения . . . 624
Применение сопутствующих сборок . . . 629
Занятие 2. Создание и применение сопутствующих сборок . . . 630
Как работают сопутствующие сборки . . . 630
Создание поддержки ресурсов в HTML-элементах . . . 631
Создание файлов ресурсов . . . 633
Загрузка файлов ресурсов . . . 635
Получение и установка культуры пользовательского интерфейса . . . 635
Отображение строк из файлов ресурсов . . . 637
Занятие 3. Другие аспекты глобализации . . . 639
Общие вопросы программирования . . . 639
Кодировка символов . . . 640
Резюме . . . 642
Практикум 15. Создание конвертера валют . . . 643
Упражнение 1. Создание Web-формы конвертера валют . . . 643
Упражнение 2. Создание ресурсов для пользовательского интерфейса . . . 644
Упражнение 3. Загрузка ресурсов, соответствующих культуре пользователя . . . 645
Упражнение 4. Выполнение действий, зависящих от культуры . . . 647
Закрепление материала . . . 649

Вопросы и ответы . . . 650

Предметный указатель . . . 660

06 этой книге

Мы рады представить вам учебный курс, посвященный разработке Web-приложений на Visual Basic .NET и Visual C# .NET.

Эта книга предназначена разработчикам, которые хотят научиться создавать Web-приложения среднего и высокого уровней сложности. Изучив теоретические материалы и выполнив практические задания из этой книги, вы научитесь разрабатывать серверные приложения, использующие .NET Framework для отображения HTML-содержимого и получения через Интернет данных от браузеров, работающих на клиентских машинах, а также подготовитесь к сдаче экзаменов 70-305 и 70-315 по программе сертификации Microsoft Certified Professional.

Примечание Дополнительную информацию о программах сертификации специалистов Microsoft Certified Application Developer и Microsoft Certified Solution Developer см. далее в разделе «Программа сертификации специалистов Microsoft».

В разделе «Начало работы» этой главы подробно рассказано, как подготовиться к занятиям, в том числе приведен список требований к оборудованию и программному обеспечению, а также указаны параметры сетевой конфигурации, необходимые для выполнения некоторых практических заданий курса. Внимательно прочитайте этот раздел, прежде чем приступить к изучению материала.

Кому адресована эта книга

Данный курс предназначен тем, кто занимается проектированием, реализацией и поддержкой Web-приложений, а также всем, кто желает сдать следующие сертификационные экзамены:

- *Developing and Implementing Web Applications with Microsoft Visual Basic .NET and Microsoft Visual Studio .NET* (экзамен 70-305);
- *Developing and Implementing Web Applications with Microsoft Visual C# .NET and Microsoft Visual Studio .NET* (экзамен 70-315).

Для изучения курса необходимы:

- навыки создания Windows-приложений на языках Visual Basic .NET или Visual C# в среде Visual Studio .NET;
- знание основ объектно-ориентированного программирования, в том числе классов, свойств, методов и событий;
- знание фундаментальных элементов языка HTML, используемого для создания информационного наполнения Web;
- понимание того, как осуществляется хранение Web-содержимого и доступ к нему через Интернет, умение раскрывать роли Web-серверов, протоколов Интернета и Web-клиентов (браузеров) в этих процессах.

Справочные материалы

- Справочная система Visual Studio.NET.
- Web-сайт консорциума по «Всемирной паутине» — World Wide Web Consortium (*http://www.w3c.org*).
- Открытая телеконференция по ASP.NET на сайте Microsoft (*news://msnews.microsoft.com/microsoft.public.dotnet.framework.aspnet*).

Содержимое компакт-диска

Компакт-диск учебного курса содержит ряд вспомогательных материалов, которые потребуются вам при изучении курса:

- электронная книга — полная электронная версия материалов учебного курса;
- примеры. В ходе занятий часто приводятся примеры кода и Web-форм, организованные в проекты Visual Studio. Некоторые из них представляют собой законченные приложения, но большинство — просто фрагменты кода, иллюстрирующие отдельные приемы программирования;
- результаты выполненных упражнений. Каждая глава курса завершается упражнениями для получения навыков работы. На компакт-диске записаны результаты упражнений, с которыми вам следует сравнить результаты, полученные при самостоятельной работе. Кроме того, к этим материалам можно обращаться, если в ходе выполнения упражнения у вас возникнут некоторые затруднения;
- **файлы, требуемые для выполнения практических заданий.** При необходимости эти файлы вы можете скопировать на ваш диск;
- **перечень примерных экзаменационных вопросов.** Ответив на них, вы выясните, насколько полно усвоили материал этого курса, а также потренируетесь в сдаче сертификационного экзамена.

Структура книги

Для повышения эффективности обучения главы этой книги разбиты на стандартные разделы:

- каждая глава начинается с раздела «В этой главе», где дан краткий обзор обсуждаемых тем; в следующем за ним разделе — «Прежде всего» — перечислены материалы, необходимые для изучения главы;
- главы делятся на занятия, в каждом из которых обсуждается отдельная тема;
- в разделе «Резюме» подводятся краткие итоги занятия и формулируются основные выводы;
- в конце занятий предлагаются упражнения (обозначаются специальным значком на полях), позволяющие применить полученные навыки на практике или поэкспериментировать с приложением-примером, о котором шла речь в занятии. Где возможно, упражнения организованы так, чтобы в конце практикума вы получили готовое приложение;
- каждую главу завершает раздел «Закрепление материала». Ответив на вопросы этого раздела, вы сможете проверить, насколько твердо усвоили материал.

В приложении «Вопросы и ответы», расположенном в конце книги, вы найдете ответы на все вопросы для самопроверки.

Примечания

В книге встречаются различные виды примечаний:

- **Совет** — поясняет возможный результат или описывает альтернативный метод решения задачи:
- **Внимание!** — содержит сведения, необходимые для выполнения поставленной задачи;
- **Примечание** — содержит дополнительную информацию.

Обозначения

- Вводимые вами символы или команды набраны **строчными буквами полужирного начертания.**
- *Курсив* в операторах указывает, что в этом месте вы должны подставить собственные значения.
- Имена файлов и каталогов начинаются с Прописных Букв (за исключением имен, которые вы задаете сами). Кроме особо оговоренных случаев, для ввода имен файлов и каталогов в диалоговых окнах или в командной строке можно использовать строчные буквы.
- Расширения имен файлов набраны строчными буквами.
- Аббревиатуры напечатаны ПРОПИСНЫМИ БУКВАМИ.
- Примеры кода, текста, выводимого на экран и вводимого в командной строке, а также фрагменты инициализационных файлов выделены моноширинным шрифтом.
- Значками на полях помечены конкретные разделы:

Значок	Описание
	Файлы на компакт-диске. О назначении и расположении нужных файлов рассказывается в сопутствующем тексте
	Упражнения, выполнив которые вы приобретете необходимые навыки

Обзор глав и приложений

Этот курс, предназначенный для самостоятельного изучения, включает занятия, упражнения и вопросы для самопроверки, которые помогут вам научиться создавать Web-приложения на Visual Studio .NET. Курс рассчитан на последовательное изучение, но не исключена возможность работы лишь с интересующими вас главами. В этом случае обращайте внимание на раздел «Прежде всего» в начале каждой главы, где указаны предварительные требования и даны ссылки на главы, в которых вы найдете материалы для подготовки к выполнению упражнений.

Ниже кратко описаны главы и приложения учебного курса.

- В главе «Об этой книге» собраны сведения о содержании учебника, а также о структурных единицах и условных обозначениях, принятых в нем. Внимательно прочитайте ее: это поможет вам эффективно работать с материалами курса, а также выбрать интересующие вас темы.
- Глава 1 «Введение в Web-программирование» познакомит вас с терминами и понятиями, используемыми при изложении материала. В ней рассказывается о том, как работают Web-приложения и из каких частей они состоят, а также о структуре .NET Framework и приемах работы в среде Visual Studio. В разделах этой главы дается краткий обзор основных понятий, которые более подробно рассматриваются в следующих главах.
- Глава 2 «Создание приложений Web Forms» рассказывает о том, как создают Web-приложения на Microsoft Visual Studio .NET и ASP.NET. Вы научитесь создавать ноше проекты, конструировать простейшие пользовательские интерфейсы, писать код для об-

работки событий и сохранять данные в приложении. Поскольку инфраструктура ASPNET интегрирована со службами Microsoft Internet Information Services (IIS), вы узнаете, что такое IIS и как использовать его для управления файлами Web-приложения.

- В главе 3 «Работа с Web-объектами» речь пойдет о том, как создавать и упорядочивать объекты в Microsoft Visual Studio .NET, применять объекты, поддерживаемые ASP.NET, а также сохранять объекты и данные в промежутках между запросами к Web-форме. Visual Basic .NET, Visual C# и ASP.NET используют общую инфраструктуру, поэтому приемы объектно-ориентированного программирования (ООП), описанные в этой главе, применимы в .NET-программировании.
- В главе 4 «Создание пользовательского интерфейса» обсуждается конструирование пользовательского интерфейса для многостраничных приложений Web Forms с помощью элементов управления ASP.NET, выбор элементов управления, подходящих для решения различных задач, а также различные виды разметки Web-форм. Кроме того, здесь рассказывается, как проверять правильность данных, введенных в элементы управления, и программировать переходы между Web-формами.
- В главе 5 «Хранение и извлечение данных с помощью ADO.NET» описано, как при помощи ADO.NET получать доступ из Web-форм к базам данных (БД) SQL и OLE и модифицировать хранящиеся в них данные. Здесь вы познакомитесь с инструментами для работы с данными, имеющимися в Visual Studio .NET, и научитесь применять их для создания соединений, извлечения записей и исполнения различных команд в БД. Вы также научитесь отображать данные на Web-формах с помощью привязки данных и других программных методов. В завершение вы узнаете, как поддерживать целостность БД при помощи транзакдий.
- В главе 6 «Перехват и исправление ошибок» рассматриваются действия при сбоях в работе приложения, вызванных внешними причинами. Материал этой главы поможет вам эффективно предупреждать, диагностировать и обрабатывать подобные ошибки.
- В главе 7 «Усложненное программирование с применением Web Forms» рассказано о том, как решать усложненные задачи программирования, которые трудно отнести к какой-то определенной категории. В принципе, Web-приложения можно создавать и не зная обсуждаемые здесь предметы. Однако, освоив материал этой главы, вы научитесь решать ряд основных задач Web-программирования — без этого ваш арсенал навыков программирования нельзя считать полным.
- В главе 8 «Обеспечение безопасности» обсуждается управление доступом к Web-приложениям с применением различных методик ASP.NET для идентификации пользователей и предоставления им разрешений на доступ к приложениям. Из этой главы вы также узнаете, как защищать пересылаемые через Интернет данные от внимания посторонних.
- Глава 9 «Компоновка и развертывание Web-приложений» посвящена тому, как опубликовать готовые Web-приложения на Web-сервере, а также наблюдать за работой приложения и поддерживать его производительность на уровне, соответствующем текущим потребностям пользователей.
- В главе 10 «Тестирование Web-приложений» описано, как выполнять систематическое тестирование Web-приложений во время его разработки. Вы научитесь планировать, подготавливать и проводить тесты, а также устранять неполадки, выявленные тестированием.
- Глава 11 «Создание нестандартных Web-элементов управления» посвящена созданию и применению нестандартных компонентов пользовательского интерфейса с Web-фор-

мами. В этой главе демонстрируются различные типы нестандартных элементов управления и способы их создания, а также сравниваются их преимущества и недостатки.

- В главе 12 «Работа с мультимедиа» рассказано, как обогатить Web-приложения звуками, видео и анимацией, а также как управлять в коде их отображением и воспроизведением.
- В главе 13 «Форматирование данных, генерируемых Web-приложением» описано форматирование элементов пользовательского интерфейса Web-приложений при помощи каскадных таблиц стилей (CSS) и XSL-преобразований. Вы узнаете, в каких случаях следует применять эти методы и как их комбинировать.
- Глава 14 «Создание справочной системы» посвящена различным способам отображения справочной информации в Web-приложении, а также применению HTML Help Workshop для создания компилированных справочных файлов. Кроме того, здесь рассказано, как сделать эти файлы доступными пользователю Web-приложения.
- В главе 15 «Глобализация Web-приложений» обсуждаются способы определения культуры, к которой принадлежит пользователь, с целью корректной обработки его запроса путем направления к локализованным версиям Web-приложения либо приспособления к различным культурам в коде единственного приложения, и отображения локализованных элементов пользовательского интерфейса.
- В приложении «Вопросы и ответы» приводятся ответы на вопросы из разделов «Закрепление материала» всех глав учебного курса.
- Словарь с определениями терминов и понятий, используемых в этом учебном курсе, вы найдете на компакт-диске, прилагаемом к этой книге.

С чего начать

Данный курс предназначен для самостоятельного изучения, поэтому вы можете пропускать некоторые занятия, чтобы вернуться к ним позже, Чтобы определить, с чего начать изучение курса, обратитесь к этой таблице.

Если вы	Что делать
Готовитесь к сдаче сертификационного экзамена 70-305 или 70-315	См. раздел «Начало работы», затем проработайте главы 1—4, далее изучайте материал книги в произвольном порядке
Хотите изучить информацию по определенной теме экзамена	См. раздел «Материалы для подготовки к экзаменам»

Материалы для подготовки к экзаменам

В таблицах перечислены темы сертификационных экзаменов 70-305: *Developing and Implementing Web Applications with Microsoft Visual Basic .NET and Microsoft Visual Studio .NET* к 70-315: *Developing u Implementing Web Applications with Microsoft Visual C# .NET and Microsoft Visual Studio .NET*, а также главы настоящего учебного курса, где обсуждаются соответствующие вопросы.

Примечание Конкретное содержание любого экзамена определяется компанией Microsoft и может быть изменено без предварительного уведомления.

Создание сервисов, предназначенных для пользователей

Тема	Где обсуждается Глава	Занятие
Создание страниц ASPNET:	2	11
• создание и установка директив на страницах ASP.NET	6, 11	6 (2), 11 (2)
• отделение ресурсов пользовательского интерфейса от бизнес-логики	3	1
Добавление серверных и HTML-элементов управления, а также нестандартных Web-элементов управления к страницам ASP.NET; написание HTML-кода для страниц ASP.NET	4	1, 2
• установка свойств элементов управления	4	1
• динамическая загрузка элементов управления	4	I
• применение шаблонов	4, 5	4 (1), 5 (2)
• назначение стилей на страницах ASP.NET при помощи каскадных таблиц стилей	13	1
• создание экземпляров и вызов ActiveX-элементов управления	12	1, 2
Реализация переходов между страницами:	2, 3. 4	2 (2), 3 (3), 4 (3)
• управление состоянием отображения	3	3
• управление данными во время событий, возвращающих форму на сервер	3	3
• применение сеансовых переменных состояния для хранения данных между отображениями страницы	3	3
Проверка данных, вводимых пользователем:	4	2
• проверка данных, вводимых на языке с алфавитом, отличным от латинского	15	3
Реализация обработки ошибок в пользовательском интерфейсе:	6	1, 2, 3
• конфигурирование нестандартных страниц сообщений об ошибках	6	*2*
• реализация обработки ошибок при помощи событий, определенных в файле Global.asax и генерируемых на уровне приложения, страницы и отдельных элементов страницы	6	2
Создание **встроенной справочной системы**	14	1, 2, 3
Встраивание **имеющегося** кода **в страницы ASP.NET**	**1**	**2**
Отображение **и обновление данных:**	**4**	**1**
• преобразование и фильтрация данных	5, 13	5 (1, 2), 13 (2)
• связывание элементов пользовательского интерфейса с данными	4	I
• отображение данных при помощи элементов управления	4	I
Создание и вызов экземпляров Web-сервисов и компонентов:	7	4
• создание и вызов экземпляров Web-сервисов	7	4
• создание и вызов экземпляров компонентов COM и COM+	10	*1*
• создание и вызов экземпляров .NET-компонентов	3	I

(окончание)

Тема	Где обсуждается Глава	Занятие
Глобализация Web-приложений:		
• организация поддержки локализации в пользовательском интерфейсе	15	2
• преобразование имеющихся файлов в различные кодировки	15	3
• отображение содержимого страниц слева направо и справа налево	15	1
• подготовка Web-форм к форматированию, зависимому от культуры	15	1
Обработка событий:	2	1
• создание обработчиков событий		
• генерация событий		
Реализация специальных возможностей форматирования	13	1
Применение и модификация внутренних объектов, таких, как объекты, представляющие отклик, запрос, сеанс, сервер и приложение:	3	2
• получение значений свойств внутренних объектов		
• установка значений свойств внутренних объектов		
• выполнение действий с использованием внутренних объектов		

Создание компонентов и **.NET-сборок** и управление ими

Тема	Где обсуждается Глава	Занятие
Создание и модификация .NET-сборок:	9, 11	9 (1), 11 (3)
• создание и применение сопутствующих сборок	15	1
• создание сборок, содержащих только ресурсы	15	2

Работа с данными и манипулирование ими

Тема	Где обсуждается Глава	Занятие
Доступ к базам данных Microsoft SQL Server и манипулирование содержащейся в них информацией при помощи запросов и хранимых процедур	5	1
Доступ к хранилищам данных (таким, как реляционные базы данных, XML-документы и «плоские» файлы) и манипулирование содержащейся в них информацией при помощи ADO.NET и XML	5	1
Обработка ошибок, возникающих при работе с данными	6	1

Тестирование и отладка

Тема	Где обсуждается Глава	Занятие
Создание плана блочного тестирования	10	1
Применение трассировки:		
• добавление к приложению приемников данных трассировки и трассировочных переключателей	10	3
• отображение результатов трассировки	6	3
Отладка, доработка и устранение дефектов кода:		
• конфигурирование отладочной среды	10	3
• написание и применение кода для отладки компонентов, страниц и приложений	10	1
• подготовка тестирования компонентов, страниц и приложений, поддерживающих несколько культур	10	2
• проведение тестирования	10	2
• устранение ошибок и доработка кода	10	3

Развертывание Web-приложений

Тема	Где обсуждается Глава	Занятие
Планирование развертывания Web-приложения:	9	2
• планирование развертывания Web-приложения в Web-саду, на Web-ферме или кластере	9	4
Развертывание Web-приложения	9	2
Добавление сборок в глобальный кэш сборок	9	2

Сопровождение и поддержка Web-приложений

Тема	Где обсуждается Глава	Занятие
Оптимизация производительности Web-приложений	9	3
Диагностика ошибок и устранение сбоев	9, 6, 10	9 (3), 6 (3). 10(3)

Конфигурирование и защита Web-приложений

Тема	Где обсуждается Глава	Занятие
Конфигурирование Web-приложений:	9	1
• модификация файла Web.config	8, 9 15	8 (1), 9 (1) 15 (3)
• модификация файла Machine.config	9	1
• добавление и модификация параметров приложения	9, 8, 15	9(1), 8(1), 15 (3)

(окончание)

Тема	Где обсуждается Глава	Занятие
Конфигурирование зашиты Web-приложений:		
• выбор метода и настройка аутентификации; разрешение анонимного доступа, применение аутентификации средствами Windows, аутентификации на основе форм, службы Microsoft Passport, Internet Information Services (IIS), а также нестандартных методов аутентификации	8	1, 2, 3
Настройка авторизации; применение различных методов авторизации на основе файлов и URL:	8	1, 2, 3, 4
• настройка авторизации на основе ролей	8	*2*
• применение олицетворения	8	1
Конфигурирование и применение кэширования; типы кэширования (кэширование откликов и страниц)	9	3
• применение объекта *cache*		
' применение директив, управляющих кэшированием		
Конфигурирование и реализация совместного использования сеансовых переменных состояния на серверах различной топологии, таких, как Web-фермы и Web-сады:		
• использование сеансовых переменных состояния внутри процесса	3	3
• обмен сеансовыми переменными состояния через службу состояния	9	4
• совместное использование сеансовых переменных состояния при помощи базы данных SQL server	9	4
Установка и настройка серверных служб:	2	1
• установка и настройка Web-сервера	2, 6, 8, 9	2 (I, 3), 6 (2), 8 (1, 5), 9 (2)
• установка и настройка FrontPage Server Extensions	2, 9	2 (I), 9 (3)

Начало работы

Данный курс предназначен для самостоятельного изучения и содержит упражнения и практические рекомендации, которые помогут вам освоить разработку Web-приложений. Для выполнения части упражнений Вам потребуется сеть из двух компьютеров или подключение к большей сети. Возможностей обоих компьютеров должно хватать для запуска Windows XP Professional Edition или Windows 2000.

Внимание! При выполнении части упражнений потребуется изменить конфигурацию серверов. Если вы подключены к большой сети, это может привести к нежелательным результатам. Перед выполнением таких упражнений предварительно проконсультируйтесь с сетевым администратором.

Аппаратное обеспечение

Компьютер должен соответствовать приведенной ниже минимальной конфигурации, а установленное на нем оборудование необходимо выбрать из списка совместимых устройств Microsoft Windows XP или Microsoft Windows 2000 Hardware Compatibility List:

- процессор Pentium II 450 МГц;
- 160 Мб памяти (рекомендуется 256 Мб);
- 12-скоростной привод CD-ROM или более быстрый;
- 3,5 Гб свободного пространства на жестком диске, в том числе 500 Мб в системном разделе;
- монитор SVGA(256 цветов, рекомендуется разрешение 800x600 или более высокое);
- мышь Microsoft или другое аналогичное устройство;
- доступ к Интернету через сеть или модем (минимально необходимую скорость передачи данных способен обеспечить модем (56600 бит/с), однако рекомендуется более быстрое подключение).

Программное обеспечение

Для выполнения упражнений вам потребуется следующее ПО:

- Microsoft Windows XP Professional Edition или Windows 2000;

Примечание Необходимое для разработки Web-приложений ПО Microsoft Internet Information Services (IIS) в этих ОС устанавливается по умолчанию.

- Microsoft Visual Studio .NET Professional Edition или Visual Studio .NET Enterprise Developer*. Рекомендуется Professional Edition, а редакция Enterprise Developer Edition идеально подходит для выполнения упражнений этого учебного курса.

Подготовка компьютера к выполнению упражнений

Прежде всего настройте компьютер согласно инструкциям поставщика.

► Установка файлов, необходимых для занятий

Чтобы установить на жесткий диск файлы, необходимые для выполнения занятий, выполните следующие действия.

1. Вставьте прилагаемый компакт-диск в дисковод CD-ROM своего компьютера.

Примечание Если на вашем компьютере отключена функция автозапуска, следуйте указаниям из файла Readme.txt на компакт-диске.

---

2. Щелкните в открывшемся меню ссылку Lesson Files, чтобы запустить программу установки. После завершения работы установочной программы необходимо подготовить виртуальные каталоги, требуемые для работы приложений-примеров. Для этого выполните следующие действия.

1. Откройте окно командной строки:
   - если у вас на компьютере уже установлена Visual Studio .NET, выберите в меню Start (Пуск) элемент Programs\Microsoft Visual Studio .NET\Visual Studio .NET Toofs\Visual Studio .NET Command Prompt. Это гарантирует, что в переменной окружения *PATH* будет указан каталог, в котором находятся компиляторы .NET Framework;
   - если у вас на компьютере установлен .NET Framework SDK, но не Visual Studio .NET, выберите в меню Start элемент Programs\Accessories\Command Prompt (Программы\Стандартные\ Командная строка). Далее необходимо добавить к переменной *PATH* каталог, в котором находятся компиляторы .NET Framework. Эти компиляторы находятся в каталоге *папка_Windows*\ Microsoft.NET\Framework*номер_версии*.

* Пробную версию Visual Studio .NET можно загрузить с Web-сайта Microsoft (см. *http://msdn.microsoft.com/subscriptions/resources/subdwnld.asp*).— *Прим. ред.*

Например, чтобы обновить переменную *PATH* в системе под управлением Windows **XP,** исполните следующую команду:

```
set path=%path%;C:\Windows\Microsoft.NET\Framework\v1.0.3705
```

(Учтите, что в системах, работающих под управлением Windows 2000, а также в системах, обновленных с Windows 2000 до Windows **XP,** каталог Windows называется WINNT, номер версии .NET Framework также может отличаться от указанного в этом примере).

2. В командной строке перейдите в каталог, куда установлены файлы примеров. Если он называется C:\MCSDWebApps (рекомендованное имя для этого каталога), то для перехода в него следует ввести следующие команды:

```
C:\> cd MCSDWebApps

C:\MCSDWebApps>
```

3. Исполните сценарии, конфигурирующие IIS так, чтобы Web-сервер мог получить доступ к примерам. При этом следует указать в качестве параметра командной строки каталог, куда были установлены файлы примеров (это текущий каталог):

```
C:\MCSDWebApps> makeVirtualDirectories C:\MCSDWebApps
```

Электронная версия книги

На прилагаемом компакт диске вы найдете электронную версию этой книги, доступную через HTML-браузер.

▶ **Как установить электронную книгу**

1. Вставьте прилагаемый компакт-диск в дисковод **CD-ROM** своего компьютера.

Примечание Если на вашем компьютере отключена функция автозапуска, следуйте указаниям из файла Readme.txt на компакт-диске.

2. Щелкните в открывшемся меню ссылку Training Kit eBook и следуйте указаниям программы.

Примечание Для работы с электронной книгой необходимо наличие прилагаемого компакт-диска в приводе CD-ROM вашего компьютера.

Вопросы пробного экзамена

▶ **Как установить на жесткий диск вопросы пробного экзамена**

1. Вставьте прилагаемый компакт-диск в дисковод CD-ROM своего компьютера.

Примечание Если на вашем компьютере отключена функция автозапуска, следуйте указаниям из файла Readme.txt на компакт-диске.

2. Щелкните в открывшемся меню ссылку Sample Exam Questions и выберите нужный тест.

Программа сертификации специалистов Microsoft

Программа сертификации специалистов Microsoft (Microsoft Certified Professional, MCP) — отличная возможность подтвердить Ваше знание современных технологий и программных продуктов этой фирмы. Лидер отрасли в области сертификации, Microsoft, разработала современные методы тестирования. Экзамены и программы сертификации подтвердят

Вашу квалификацию разработчика или специалиста по реализации решений на основе технологий и программных продуктов Microsoft. Сертифицированные Microsoft работники квалифицируются как эксперты и высоко ценятся на рынке труда.

Программа сертификации специалистов предлагает 7 типов сертификации по разным специальностям.

- *Сертифицированный разработчик приложений Microsoft (Microsoft Certified Application Developer, MCAD) для платформы Microsoft .NET* — способен разрабатывать и поддерживать приложения уровня департамента, а также серверные и клиентские компоненты служб для работы с данными, предназначенных для использования как в Web, так и в корпоративных сетях.
- *Сертифицированный разработчик программных решений Microsoft (Microsoft Certified Solution Developer, MCSD)* — предполагает проектирование и создание решений для бизнеса с использованием средств разработки, платформ и технологий корпорации Microsoft, а также архитектуры Microsoft Windows;
- *Сертифицированный специалист Microsoft (Microsoft Certified Professional, MCP)* — предполагает доскональное знание по крайней мере одной ОС из семейства Windows или ключевой платформы Microsoft. Такой специалист обладает навыками внедрения продукта или технологии Microsoft как части бизнес-системы предприятия.
- *Сертифицированный системный инженер Microsoft (Microsoft Certified Systems Engineer, MCSE)* — подразумевает умение эффективно анализировать потребности компаний, а также проектировать и реализовать инфраструктуры для бизнес-решений на базе платформы Windows 2000 и серверных продуктов корпорации Microsoft.
- *Сертифицированный системный администратор Microsoft (Microsoft Certified System Administrator, MCDBA) на Windows 2000* — занимается вопросами реализации, управления и устранения неполадок в существующих системах на основе Windows 2000, включая и Windows .NET Server
- *Сертифицированный администратор баз данных Microsoft (Microsoft Certified Database Administrator, MCDBA) на Microsoft SQL Server 2000* — разработка физической структуры, логических моделей данных, создание физических БД, создание служб доступа к данным с использованием T-SQL, управление и поддержка БД, настройка и управление системой защиты, мониторинг и оптимизация БД, а также установка и настройка Microsoft SQL Server.
- *Сертифицированный преподаватель Microsoft (Microsoft Certified Trainer, MCT)* — теоретическая и практическая подготовка для ведения соответствующих курсов с использованием учебных материалов Microsoft Official Curriculum (MOC) в сертифицированных центрах технического обучения Microsoft (Microsoft Certified Technical Education Centers, CTECs).

Достоинства сертификации Microsoft

Программа сертификации Microsoft — один из самых строгих и полных тестов оценки знаний и навыков в области проектирования, разработки и сопровождения программного обеспечения. Сертифицированным специалистом Microsoft становится лишь тот, кто демонстрирует умение решать конкретные задачи, применяя продукты компании. Программа тестирования позволяет не только оценить квалификацию специалиста, но и служит ориентиром для всех, кто стремится достичь современного уровня знаний в этой области. Как и любой другой тест или экзамен, сертификация Microsoft представляет собой показатель определенного уровня знаний специалиста, что важно для работодателя и всей организации в целом.

Преимущества сертифицированного специалиста

Звание Microsoft Certified Professional дает вам:

- официальное признание ваших знаний и опыта работы с продуктами и технологиями Microsoft;
- подписку на Microsoft Developer Network. MCP получают скидку на годовую подписку на Microsoft Developer Network (см. *msdn.microsoft.com/subscriptions/*) в течение первого года действия сертификации (более точную информацию об условиях подписки вы найдете в MCP Welcome Kit);
- доступ к технической информации о продуктах Microsoft через защищенную область Web-узла MCP (см. *http://www.microsoft.com/traincert/mcp/mcpsecure.asp/*);
- доступ к эксклюзивным скидкам на продукты и услуги от определенных компаний. Сертифицированные специалисты могут узнать больше о предлагаемых скидках на защищенной области сайта MCP в разделе Other Benefits (см. *http://www.microsoft.com/traincert/mcp/mcpsecure.asp/*);
- право использовать логотип MCP, транскрипт, сертификат, карточку и значок MCP, чтобы проинформировать ваших клиентов и коллег об имеющемся статусе Microsoft Certified Professional. Электронные файлы логотипа MCP и ваш персональный транскрипт могут быть получены через защищенную область сайта MCP (*http://www.microsoft. com/traincert/mcp/mcpsecure.asp/*);
- приглашения на конференции, семинары и специальные мероприятия Microsoft, предназначенные для специалистов;
- свободный доступ к защищенной области Microsoft Certified Professional Magazine Online — журнала, посвященного вопросам карьеры и профессионального развития сертифицированных специалистов Microsoft, а также к разделам Web-сайта журнала со статьями и другой информацией, доступной только на сайте, и регулярно проводимым форумам с участием экспертов Microsoft и других компаний;
- скидку на членство в Professional Association for SQL Server (PASS, Профессиональная Ассоциация пользователей SQL Server) — единственной всемирной организации, управляемой пользователями SQL Server и функционирующей с одобрения Microsoft. Ее членам доступны уникальные возможности повышения уровня своих знаний и навыков. Обратитесь на защищенную область сайта MCP (*http://www.microsoft.com/traincert/mcp/mcpsecure.asp/*) для получения дополнительной информации о том, как воспользоваться данным преимуществом.

Кроме того, в зависимости от типа сертификации и страны, обладатели сертификата MCSA или MCDBA получают 50-процентную скидку на годовую подписку на компакт-диски с материалами TechNet и TechNet Plus в течение первого года действия сертификации (более точную информацию об условиях подписки вы найдете в MCP Welcome Kit). Эти специалисты также получают бесплатный доступ к большинству материалов, содержащихся на компакт-дисках, через сайт TechNet (*http://www.microsoft.com/technet/*).

Обладателям сертификата MCDBA дополнительно предлагается бесплатная годовая подписка на журнал SQL Server Magazine, в котором публикуются статьи ведущих специалистов отрасли. Этот журнал, содержащий массу полезной технической информации и практических советов, полезен каждому, кто работает с SQL Server.

О преимуществах сертифицированных преподавателей Microsoft рассказано в материалах на сайте *http://www.microsoft.com/traincert/mcp/mct/benefits.asp*.

Выигрыш от сертификации Microsoft для работодателей и организаций

Сертификация позволяет организациям быстро окупить затраты на технологии Microsoft и извлечь максимум прибыли из этих технологий. Исследования показывают, что сертификация сотрудников по программам Microsoft:

- быстро окупается за счет стандартизации требований к обучению специалистов и методов оценки их квалификации;
- позволяет увеличить эффективность обслуживания клиентов, повысить производительность труда и снизить расходы на сопровождение ПО;
- обеспечивает надежные критерии для найма специалистов и их продвижения по службе;
- предоставляет методы оценки эффективности труда персонала;
- обеспечивает гибкие методы переподготовки сотрудников для обучения новым технологиям;
- позволяет оценить партнеров — сторонние фирмы.

Требования к соискателям

Требования к соискателям определяются специализацией, а также служебными функциями и задачами.

Соискатель сертификата Microsoft должен сдать экзамен, подтверждающий его глубокие знания в области программных продуктов Microsoft. Экзаменационные вопросы, подготовленные с участием ведущих специалистов компьютерной отрасли, отражают реалии применения программных продуктов Microsoft.

- На звание *Сертифицированного специалиста Microsoft* сдают экзамен по работе с одной из операционных систем. Кандидат может сдать дополнительные экзамены, которые подтвердят его право на работу с другими продуктами, инструментальными средствами или прикладными программами Microsoft.
- На звание *Сертифицированного системного инженера Microsoft* сдают семь экзаменов: пять ключевых и два экзамена по выбору.
- На звание *Сертифицированного системного администратора Microsoft* сдают четыре экзамена: три ключевых и один экзамен по выбору.
- На звание *Сертифицированного администратора баз данных Microsoft* сдают три ключевых экзамена и один экзамен по выбору.
- На звание *Сертифицированного разработчика программных решений на основе Microsoft* сдают три экзамена по основам технологий ОС Microsoft Windows и один — по технологиям интегрированного семейства серверных продуктов Microsoft BackOffice.
- На звание *Сертифицированного преподавателя Microsoft* надо подтвердить свою теоретическую и практическую подготовку для ведения соответствующих курсов в авторизованных учебных центрах Microsoft. Участие в программе требует соответствия требованиям, предъявляемым при ежегодном обновлении статуса сертифицированного преподавателя. Более подробные сведения о сертификации по этой программе можно получить на сайте *http://www.microsoft.com/traincert/mcp/mcp.asp* или в местном отделении компании Microsoft.

Подготовка к экзаменам

Существует три режима подготовки: самостоятельная работа, интерактивный режим, а также занятия с инструктором в авторизованных центрах подготовки.

Самостоятельная подготовка

Самостоятельная подготовка — наиболее эффективный метод подготовки для инициативных соискателей. Издательство Microsoft Press предлагает весь спектр учебных пособий для подготовки к экзаменам по программе сертификации специалистов Microsoft. Учебные курсы для самостоятельного изучения, адресованные специалистам компьютерной отрасли, содержат теоретические и практические материалы, мультимедийные презентации, упражнения и необходимое ПО. Серия «Mastering» — это интерактивные обучающие компакт-диски для опытных разработчиков. Все эти пособия позволят Вам наилучшим образом подготовиться к сдаче сертификационных экзаменов.

Интерактивная подготовка

Интерактивная подготовка средствами Интернета — альтернатива занятиям в учебных центрах. Вы можете выбрать наиболее удобный распорядок занятий в виртуальном классе, где Вы научитесь работать с продуктами и технологиями Microsoft и подготовитесь к сдаче экзаменов. Интерактивное обучение охватывает множество курсов Microsoft — от обычных официальных до специальных, доступных лишь в интерактивном режиме. Интерактивные ресурсы доступны круглосуточно в авторизованных центрах подготовки.

Сертифицированные центры технического обучения

Сертифицированные центры технического обучения (Microsoft Certified Technical Education Center, CTEC) — самый простой способ пройти курс обучения под руководством опытного инструктора и стать сертифицированным специалистом. Microsoft CTEC — всемирная сеть учебных центров, которые позволяют специалистам повысить свой технический потенциал под руководством сертифицированных инструкторов Microsoft.

Список центров CTEC в США и Канаде можно получить на Web-сайте *http://www.microsoft.com/traincert/ctec/*.

Техническая поддержка

Мы постарались сделать все от нас зависящее, чтобы и учебный курс, и прилагаемый к нему компакт-диск не содержали ошибок. Издательство Microsoft Press публикует постоянно обновляемый список исправлений и дополнений к своим книгам по адресу http://mspress.microsoft.com/support/.

Если все же у вас возникнут вопросы или Вы захотите поделиться своими предложениями или комментариями, обращайтесь в издательство Microsoft Press по одному из указанных ниже адресов:

Электронная почта: TKINPUT@MICROSOFT.COM

Почтовый адрес:
Microsoft Press
Attn: *MCAD/MCSD Training Kit: Developing Web Applications with Microsoft Visual Basic .NET and Visual C# .NET* Editor
One Microsoft Way
Redmond, WA 98052-6399

Требования к системе

Для эффективного освоения учебного курса MCAD/MCSD «Разработка Web-приложений на Microsoft Visual Basic .NET и Microsoft Visual C# .NET» ваш компьютер должен соответствовать указанной в этом разделе минимальной конфигурации:

- Microsoft Windows XP Professional Edition или Windows 2000;

Примечание Необходимый для разработки Web-приложений ПО Microsoft Internet Information Services (IIS) в этих ОС устанавливается по умолчанию.

- Microsoft Visual Studio .NET Professional Edition или Visual Studio .NET Enterprise Developer*: Professional Edition рекомендуется, а редакция Enterprise Developer Edition идеально подходит для выполнения упражнений этого учебного курса;
- процессор Pentium II 450 МГц;
- 160 Мб памяти (рекомендуется 256 Мб);
- 12-скоростной привод CD-ROM или более быстрый;

Примечание Дисковод CD-ROM требуется для установки файлов, содержащих практические задания по электронной версии учебного курса.

- 3,5 Гб свободного пространства на жестком диске, в том числе 500 Мб в системном разделе;
- монитор SVGA (256 цветов, рекомендуется разрешение 800x600 или более высокое);
- мышь Microsoft или другое аналогичное устройство;
- доступ к Интернету через сеть или модем (минимально необходимую скорость передачи данных способен обеспечить модем (56600 бит/с), однако рекомендуется более быстрое подключение).

---

* Пробную версию Visual Studio .NET загрузить с Web-сайта Microsoft (см. *http://msdn.microsoft.com/subscriptions/resources/subdwnld.asp*). — *Прим. ред.*

ГЛАВА 1

Введение в Web-программирование

Занятие 1. Типы Интернет-приложений 2

Занятие 2. Использование ASP.NET 7

Занятие 3. Работа в Visual Studio .NET 16

В этой главе

В этой главе описаны различные типы Интернет-приложений и инструменты, при помощи которых их создают, а также определены понятия и термины, используемые в этой книге. Вы узнаете, как работают Web-приложения, из каких частей они состоят, познакомитесь с организацией инфраструктуры .NET Framework и научитесь работать в Visual Studio .NET. В этой главе дается краткий обзор основных понятий, подробно рассмотренных в следующих главах.

Прежде всего

Для изучения материалов этой главы вам потребуются:

- Visual Studio .NET;
- доступ в Интернет через локальную сеть (ЛВС), широкополосный канал или модем.

Занятие 1. Типы Интернет-приложений

Web-приложения — это один из четырех типов Интернет-приложений, которые можно создавать при помощи Visual Studio .NET и Microsoft ASP.NET. Из этого занятия вы узнаете о различных типах Интернет-приложений и познакомитесь с принципами работы Web-приложений.

Изучив материал этого занятия, вы сможете:

- ✓ описать четыре типа Интернет-приложений;
- ✓ рассказать, как работает Web-приложение и чем оно отличается от традиционного статического Web-сайта;
- ✓ понять роль ASP.NET в создании Web-приложений;
- ✓ перечислить компоненты ASP.NET и описать ее преимущества по сравнению с другими технологиями разработки Web-приложений, такими, как Common Gateway Interface (CGI). Кроме того, в этом занятии указаны справочные и учебные материалы, посвященные разработке этих типов приложений.

Продолжительность занятия — около 5 минут.

Четыре типа Интернет-приложений

Строго говоря, Интернет-приложением можно считать любой программный продукт, так или иначе использующий Интернет. Согласно такому определению, приложения, пользователи которых регистрируются или получают доступ к справочной системе через Интернет, тоже в какой-то степени являются Интернет-приложениями.

Но программ, попадающих под это определение, слишком много для одной книги. Давайте попробуем ограничить область нашего интереса, выделив четыре типа Интернет-приложений;

- **Web-приложения,** которые работают на сервере, передавая через Интернет данные на клиентские машины. Для их применения требуются Web-браузеры, такие, как Microsoft Internet Explorer и Netscape Navigator;
- **Web-сервисы,** которые позволяют приложениям обрабатывать их данные на сервере. При этом передача подлежащих обработке данных на сервер и возврат результатов осуществляется через Интернет;
- **приложения с поддержкой Интернета** — автономные программы со встроенными механизмами, позволяющими их пользователям регистрироваться, получать обновления, а также предоставляющими доступ к справочной системе и другим вспомогательным службам через Интернет;
- **одноранговые приложения** — автономные программы, использующие Интернетдля взаимодействия с другими программными продуктами этого же типа.

В Visual Studio .NET можно создавать все вышеперечисленные типы приложений, но в этой книге речь пойдет о разработке первого типа программ — Web-приложений. В таблице 1-1 перечислены разделы справочной системы Visual Studio .NET и учебные курсы, в которых можно найти сведения о разработке различных типов Интернет-приложений.

Таблица 1-1. Источники сведений об Интернет-приложениях

Тип приложений	Темы разделов справочной системы	Пособие для подготовки к экзамену MCSD
Web-приложения	ASP.NET, Web Forms, пространство имен *System. Web*	Эта книга
Web-сервисы	ASRNET, Web-сервисы XML, пространство имен *System, Web.Services*	Учебный курс *Developing Web Services and ServerComponents*
Приложения с поддержкой Интернета	Windows-формы, справочная система на основе HTML, элемент управления WebBrowser, пространство имен *System. Net*	Учебные курсы *Developing Windows Applications with the Microsoft .NET Framework* и *Developing Desktop Applications with Microsoft Visual C++ 6.0*
Одноранговые приложения	Доступ к Интернету, подключаемые протоколы, пространство имен *System.Net.Sockets*	Учебный курс *Developing Wet Services and Server Components*

Принцип работы Web-приложений

Web-приложения используют архитектуру «клиент — сервер». Собственно, Web-приложение находится на сервере и обрабатывает запросы, которые передают ему через Интернет многочисленные клиенты (рис. 1-1).

На стороне клиента Web-приложение работает в браузере, например в Internet Explorer или Netscape Navigator. Пользовательский интерфейс Web-приложения передается на клиентскую машину в виде страниц на языке HTML (Hypertext Markup Language), где браузер интерпретирует и отображает их.

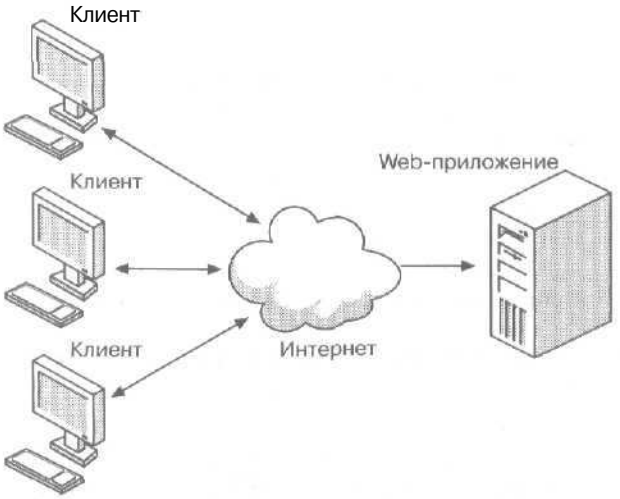


Рис. 1-1. Архитектура Web-приложений

На стороне сервера Web-приложение работает под управлением IIS (Internet Information Services). IIS управляет работой приложения, передает ему клиентские запросы и возвращает клиентам результаты исполнения их запросов. Запросы и результаты их исполнения передаются через Интернет по протоколу HTTP (Hypertext Transport Protocol). *Протокол —*

это набор правил, регламентирующих взаимодействие двух и более сущностей, которое реализуется через среду, такую, как Интернет. На рис. 1-2 показано, как клиент и сервер взаимодействуют через Интернет.



Рис. 1-2. Взаимодействие клиента **и сервера при работе Web-приложения**

Web-приложение компонует отклик из серверных ресурсов, к которым относятся исполняемый код, работающий на сервере (то, что традиционно считается «приложением» в Windows-программировании), Web-формы, HTML-страницы, графические файлы и иное содержимое, составляющее информационное наполнение приложений.

Web-приложения во многом напоминают традиционные Web-сайты, но в отличие от них отображают пользователю динамическое содержимое, генерируемое исполняемым кодом приложения, а не статические страницы, хранящиеся на сервере в готовом виде. На рис. 1-3 показано, как Web-приложение компонует HTML-страницы, возвращаемые пользователям.



Рис. 1-3. Web-приложение генерирует отклик из серверных ресурсов

Исполняемая часть Web-приложения способна делать многое из того, чего не могут статические Web-сайты, а именно:

- принимать данные от пользователя и сохранять их на сервере;
- выполнять для пользователя различные действия: размешать заказы, делать сложные вычисления и извлекать информацию из баз данных (БД);
- опознавать пользователя и отображать интерфейс, настроенный в соответствии с его предпочтениями;
- отображать постоянно меняющееся содержимое, например инвентарные списки, обрабатываемые заказы и сведения об отгружаемых товарах.

Этот перечень далеко не полон. В принципе, Web-приложения способны решить любые вообразимые задачи, доступные и клиент-серверному приложению. Особенность Web-приложений в том, что взаимодействие между клиентом и сервером осуществляется через Интернет.

Возможности ASP.NET

ASP.NET — это платформа для создания Web-приложений и Web-сервисов, работающих под управлением IIS. Конечно, есть и другие технологии, позволяющие создавать Web-приложения, например CGI. ASP.NET отличается от них высокой степенью интеграции с серверными продуктами, а также с инструментами Microsoft для разработки, доступа к данным и обеспечения безопасности.

Возможности ASP.NET сделали разработку Web-приложений намного более упорядоченной, подобно тому как Microsoft Office позволил навести порядок в мире приложений для настольных систем. ASP.NET является частью инфраструктуры .NET Framework и состоит из следующих компонентов:

- **инструментов Visual Studio .NET для Web-разработки** — графических средств разработки Web-страниц, шаблонов Web-приложений и инструментов для управления проектами и развертывания Web-приложений;
- **пространств имен System.Web**, которые являются частью .NET Framework и включают классы для работы с элементами, специфичными для Web, такими, как HTTP-запросы и отклики, браузеры и электронная почта;
- **серверных элементов управления и HTML-элементов управления** — компонентов пользовательского интерфейса, применяемых для приема данных от пользователей и отображения им отклика приложения,

Кроме того, ASP.NET применяет ряд более универсальных компонентов и инструментальных средств из ОС Windows. Хотя они не считаются частью ASP.NET, они имеют ключевое значение для программирования с использованием ASP.NET:

- **Internet Information Services (IIS)** — как сказано выше, IIS управляет Web-приложениями на серверах Windows;
- **языки программирования Visual Basic .NET, Visual C# и JScript** — Visual Studio .NET поддерживает все эти языки, поэтому Web-приложения можно писать на любом им них;
- **.NET Framework** — полный набор классов, необходимый для Windows-программирования. Помимо классов ASP.NET, в него входят классы для решения других задач программирования: файлового доступа, преобразования типов, манипулирования массивами и строками и т. д,;
- **инструменты и классы ADO.NET для** работы с **БД,** предоставляющие доступ к БД SQL Server и ODBC. Механизмы доступа к данным часто являются ключевыми компонентами Web-приложений;
- **Application Center Test (ACT)** — компонент Visual Studio .NET, автоматизирующий нагрузочное тестирование Web-приложений.

Хотя ASP.NET является наиболее полной платформой для разработки Web-приложений, важно помнить, что она функционирует исключительно на серверах Windows, так как требует наличия IIS. Для создания Web-приложений, не требующих IIS (а использующих, скажем, Web-сервер Apache) и работающих на серверах под управлением ОС, отличной от Windows (например. Linux), применяются другие технологии (как правило. CGI).

Преимущества ASP.NET

ASP.NET обладает целым рядом преимуществ по сравнению с другими платформами для создания Web-приложений. Возможно, самое важное из них — интеграция с серверами Windows и инструментами программирования. Применение ASP.NET упрощает разработку, отладку и развертывание Web-приложений, поскольку все эти задачи удается решать в одной и той же среде разработки — Visual Studio .NET.

Вот основные преимущества, которые дает разработчикам Web-приложений использование ASP.NET:

- компиляция исполняемых частей Web-приложения, благодаря которой они работают быстрее, чем интерпретируемые сценарии;
- обновление развернутых Web-приложений «на лету» без перезапуска сервера;
- доступ к инфраструктуре .NET Framework, дополняющей возможности API Windows;
- использование известного языка программирования Visual Basic, который теперь полностью поддерживает объектно-ориентированное программирование (ООП), и нового языка Visual C# — более совершенной версии C, поддерживающей ООП и контроль типов;
- автоматическое управление состоянием элементов управления Web-страниц [называемых *серверными элементами управления* (server controls)]. Благодаря этой особенности их поведение мало отличается от обычных элементов управления Windows;
- возможность создания новых, нестандартных серверных элементов управления на основе существующих;
- обеспечение безопасности Web-приложений с помощью встроенных защитных механизмов сервера Windows и других методов аутентификации и авторизации;
- интеграция с **ADO.NET,** позволяющая получать доступ к БД и применять инструменты для конструирования БД из Visual Studio .NET;
- полная поддержка языка Extensible Markup Language (XML), каскадных таблиц стилей (CSS) и других новых стандартов Web;
- встроенные механизмы кэширования часто запрашиваемых Web-страниц на сервере, локализации содержимого (приспособления его для использования различных языков и региональных стандартов) и определения возможностей браузера.

Занятие 2. Использование ASP.NET

Из этого занятия вы узнаете, как организованы Web-приложения ASP.NET, как называются их компоненты и какие роли они выполняют. Вы также познакомитесь с Web-формами — центральным элементом пользовательского интерфейса Web-приложений,

ASP.NET входит в состав более крупной инфраструктуры, .NET Framework, поэтому мы начнем с рассказа об организации .NET Framework и отличиях между исполнением приложений .NET и традиционных Windows-приложений.

В конце занятия мы обсудим языки программирования, на которых можно писать Web-приложения. Поскольку ASP.NET позволяет писать Web-приложения на разных языках, в заключительной части занятия приводится сравнительный анализ языков программирования, представленных в этой книге (Visual Basic .NET и Visual C#).

Изучив материал этого занятия, вы сможете:

- ✓ перечислить компоненты Web-приложения и рассказать, как они работают на сервере;
- ✓ указать черты сходства и отличия Web-форм, HTML-страниц и Windows-форм;
- ✓ описать некоторые элементы управления, размещаемые на Web-формах;
- ✓ перечислить компоненты .NET Framework и рассказать, как общеязыковая исполняющая среда (CLR) исполняет .NET-приложения;
- ✓ понять структуру .NET Framework и найти классы, предназначенные для решения общих задач прикладного программирования;
- ✓ дать сравнительную характеристику языков программирования Visual Basic .NET и Visual C#.

Продолжительность занятия — около 10 минут.

Структура Web-приложений

Любое Web-приложение состоит из трех частей: информационного наполнения, кода, реализующего логику приложения, и сведений о конфигурации. Кратко эти части с указанием их места в структуре Web-приложений ASP.NET описаны в таблице 1-2.

Таблица 1-2. Компоненты Web-приложения ASP.NET

Компонент	Типы файлов	Описание
Информационное наполнение	Web-формы, HTML, графика, аудио, видео и другие данные	Эти файлы определяют вид Web-приложения. Содержат как статические (текст или изображения), так и динамические элементы, компонуемые кодом приложения «на лету» (например, запросы к БД)
Код	Исполняемые файлы и сценарии	Реализуют логику, определяющую реакцию приложения на запросы пользователя. Код Web-приложений ASP.NET находится в динамически подключаемой библиотеке (DLL), работающей на сервере, и (в некоторых случаях) сценариях, исполняемых на клиентских машинах
Сведения о конфигурации	Файл Web.config, таблицы стилей, параметры IIS	Конфигурационные файлы и параметры определяют способ исполнения приложения на сервере; круг лиц, имеющих доступ к приложениям; способы обработки ошибок и ряд других аспектов

Web-форма (Web form) — это ключевой элемент Web-приложения. Она в равной мере похожа на HTML-страницу и на Windows-форму: Web-форма выглядит и работает как HTML-страница, но, подобно Windows-форме, содержит элементы управления, реагирующие на события и исполняющие код.

В готовом Web-приложении исполняемый код Web-формы хранится в библиотеке (.dll-файл), работающей под управлением IIS на сервере. Информационное наполнение Web-формы находится в каталоге содержимого Web-сервера (рис. 1-4).

Когда пользователь открывает в браузере страницу Web Forms, происходит следующее:

1. IIS запускает исполняемую часть Web-приложения, если это еще не сделано;
2. исполняемая часть приложения компонует отклик для данного пользователя из содержимого запрошенной им страницы Web Forms и содержимого, динамически сформированного кодом;
3. IIS возвращает пользователю результат исполнения его запроса в виде HTML.

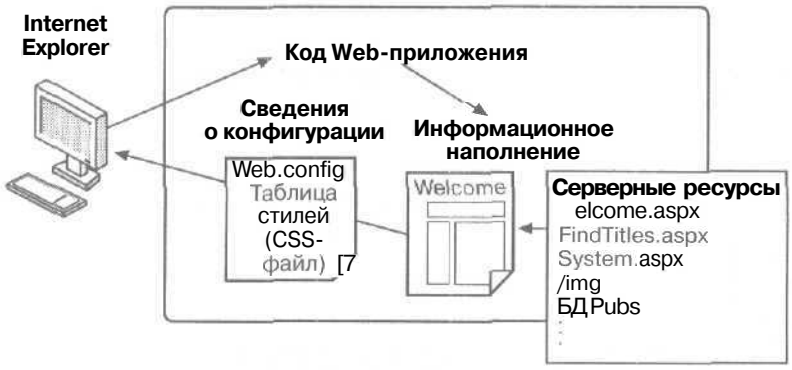


Рис. 1-4. Размещение частей **Web-приложения на сервере**

Получив запрошенную Web-форму, пользователь может вводить в нее данные, устанавливать нужные переключатели, щелкать кнопки, то есть работать с элементами управления формы. Манипуляции над некоторыми элементами управления (например, щелчки кнопок) инициируют возврат формы на сервер для обработки событий, и все повторяется снова (рис. 1-5).

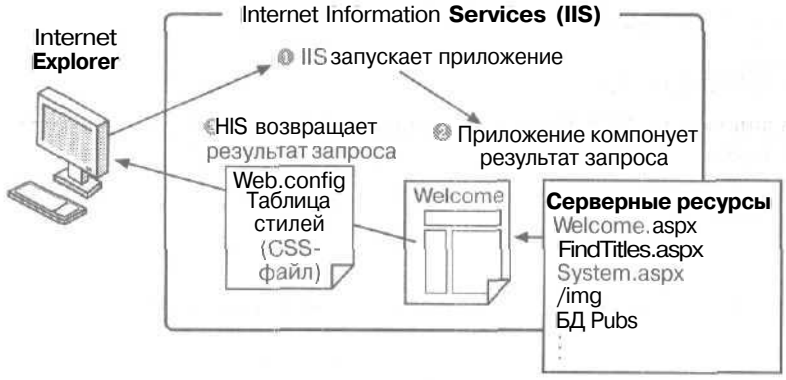


Рис. 1-5. Схема взаимодействия частей Web-приложения

Более подробно эти действия описаны в занятии 2 главы 2.

Компоненты Web-форм

На Web-формах могут размещаться различные компоненты (таблица 1-3),

Таблица 1-3. Компоненты, размещаемые на Web-формах

Компонент	Примеры	Описание
Серверные элементы управления	*TextBox*, *Label*, *Button*, *ListBox*, *DropDownList*, *DataGrid*	Эти элементы управления реагируют на события, происходящие в ответ на действия пользователя, — исполнение соответствующих процедур на сервере. Имеют встроенные механизмы для сохранения введенной пользователем информации при отправке страницы на сервер. Используются для конструирования пользовательского интерфейса Web-формы
HTML-элементы управления	*Text area*, *Table*, *Image*, *Submit Button*, *Reset Button*	HTML-версии стандартных графических элементов управления. Удобны в случаях, когда требуется лишь часть богатой функциональности серверных элементов управления
Элементы управления, связанные с данными	*SqlConnection*, *SqlCommand*, *OleDbConnection*, *OleDbCommand*, *DataSet*	**Позволяют** подключаться к БД SQL и OLE, а также к файлам данных XML; исполнять над ними различные команды и извлекать данные из этих источников
Системные компоненты	*FileSystem Watcher*, *EventLog*, *MessageQueue*	Эти элементы управления предоставляют доступ к системным событиям сервера

Серверные элементы управления и HTML-элементы необходимы для создания пользовательского интерфейса Web-формы. Элементы управления, связанные с данными, как и системные компоненты, видны на поверхности формы только во время разработки, что позволяет определять их свойства и создавать обработчики для их событий. У этих элементов управления нет графического представления времени выполнения. На рис. 1-6 показано, как выглядит Web-форма с размещенными на ней компонентами.

Более подробно применение серверных элементов управления и HTML-элементов управления с Web-формами рассматривается в главе 4.

Структура .NET Framework

ASP.NET — это всего лишь часть .NET Framework, хотя и очень важная, Зная весь спектр возможностей .NET Framework, вы сможете более эффективно писать приложения ASP.NET и избегать дублирования кода, уже имеющегося в .NET Framework.

Но сначала немного теории. .NET Framework — это новая платформа, созданная Microsoft для разработки программного обеспечения для Windows и Web. Она состоит из двух частей:

- исполняющего ядра под названием *общеязыковая исполняющая среда* (common language runtime, CLR);
- библиотеки классов, открывающих доступ к системным функциям, включая и те, что прежде были доступны только через API Windows. Кроме того, библиотека классов предоставляет доступ к прикладным функциям для Web-разработки (ASP.NET), для доступа к данным (ADO.NET), обеспечения безопасности и удаленного управления.

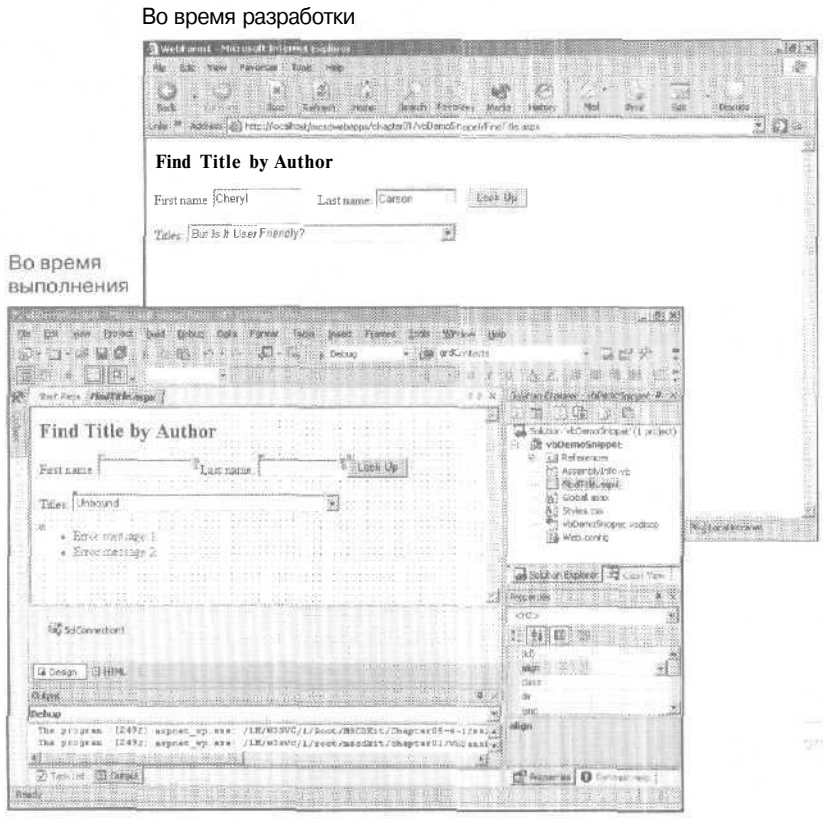


Рис. 1-6. Web-форма с размещенными на ней компонентами

.NET-приложения исполняются иначе, чем традиционные Windows-приложения, которые вы писали раньше. При компиляции .NET-приложения вместо исполняемого файла с машинными кодами получается *сборка* (assembly), содержащая команды промежуточного языка Microsoft intermediate language (MSIL). Во время выполнения CLR осуществляет окончательную компиляцию сборки. При исполнении программы CLR берет на себя управление памятью, контроль типов и решает за приложение ряд других задач. Как все это работает, показано на рис. 1-7.

Работающие в CLR приложения называются *управляемым кодом* (managed code), поскольку исполнение большинства задач, за которые раньше отвечал прикладной код, берет на себя CLR. Применение управляемого кода решает проблему Windows-программирования, связанную с регистрацией и управлением версиями компонентов (известную также как «ад DLL»), поскольку сборка содержит все сведения о версиях и типах, необходимые CLR для исполнения приложения. Вместо статической регистрации компонентов в системном реестре, как делают COM-приложения, CLR динамически регистрирует компоненты во время выполнения.

Библиотека классов .NET открывает доступ ко всем возможностям CLR. Классы, составляющие эту библиотеку, организованы при помощи пространств имен. Каждое пространство имен заключает в себе классы, выполняющие близкие функции. В таблице 1-4 кратко описаны пространства имен .NET, особенно интересных разработчикам Web-приложений.

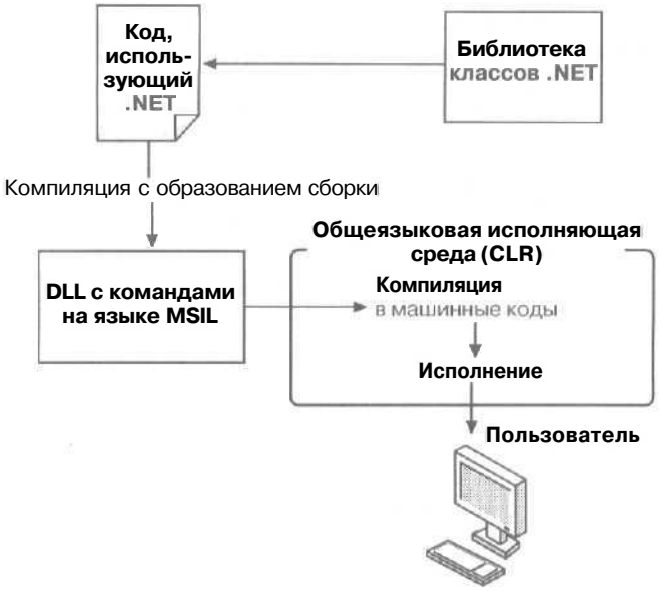


Рис. 1-7. Схема работы .NET-приложения

Таблица 1-4. Структура библиотеки классов .NET Framework

Категория	Пространства имен	Функции расположенных в них классов
Общие типы	*System*	Представляют все общие типы данных, в том числе строки, массивы и числовые типы. Поддерживают методы для преобразования типов, манипулирования строками и массивами, математических вычислений и генерации случайных чисел
Доступ к данным	*System.Data*, *System.Data.Common*, *System.Data.OleDb*, *System.Data.SqlClient*, *System.Data.SqlTypes*	Обеспечивают доступ к БД. Здесь находятся классы, предоставляющие методы для подключения к БД, исполнения над ними различных команд, а также для извлечения и модификации данных
Отладка	*System.Diagnostics*	Служат для отладки приложений и трассировки ихисполнения
Файловый доступ	*System.IO*, *System.IO.IsolatedStorage*, *System.DirectoryServices*	Предоставляют доступ к файловой системе. Поддерживают методы для чтения и записи файлов, а также для получения имен файлов и путей к ним
Сетевое взаимодействие	*System.Net*, *System.Net.Sockets*	Обеспечивают взаимодействие через Интернет с использованием низкоуровневых протоколов, например TCP/IP. Эти классы необходимы для создания одноранговых приложений

Таблица 1-4. *(окончание)*

Категория	Пространства имен	Функции расположенных в них классов
Безопасность	*System. Security, System.Security. Cryptography, System.Security. Permissions, System. Security.Policy, System. Web.Security*	Обеспечивают аутентификацию и авторизацию пользователей, а также шифрование данных
Web-сервисы	*System. Web.Services, System. Web. Services.Configuration, System. Web.Services.Description, System.Web.Services. Discovery, System.Web, Services. Protocols*	Позволяют создавать и публиковать компоненты. с которыми можно работать через Интернет. К ним относятся основные классы, используемые для создания Web-сервисов ASPNET
Windows-приложения	*System. Windows. Forms, System. Windows. Forms. Design*	Служат для создания приложений, обращающихся к компонентам пользовательского интерфейса Windows. Представляют Windows-формы и Windows-элементы управления. а также позволяют создавать нестандартные элементы управления
Данные в форме XML	*System.Xml, System.Xml. Schema, System.Xml.Serialization, System. Xml.Xpath, System. Xml. Xsl*	**Позволяют** создавать XML-файлы и получать доступ к ним

Поскольку классы распределяются по пространствам имен .NET в соответствии с их функциями, пространства имен служат ориентиром для поиска классов и членов, предоставляющих доступ к нужным функциям CLR. Например, пространство имен *System* — одно из самых востребованных, поскольку в нем находятся классы фундаментальных типов данных. Вы обращаетесь к нему всякий раз, объявляя переменную числового или строкового типа либо массив.

Такой подход позволяет .NET Framework поддерживать встроенные методы для преобразования типов данных и манипулирования строками и массивами. Например, следующий фрагмент кода сортирует список при помощи встроенных методов классов *String* и *Array*.

Visual Basic .NET

```
' Объявить и инициализировать строку,
Dim strFruit As String - "oranges apples peaches kumquats nectarines mangos"
' Объявить массив.
Dim arrFruit As String()
' Записать слова из строки е элементы массива.
arrFruit = strFruit.Split(" ")
' Выполнить сортировку массива.
System.Array.Sort(arrFruit)
' Записать отсортированные данные из массива обратно в строку.
strFruit = String.Join(" ", arrFruit)
```

Visual C#

```
// Объявить и инициализировать строку.
string strFruit - "oranges apples peaches kumquats nectarines mangos";
// Объявить массив.
string[] arrFruit;
// Записать слова из строки в элементы массива.
arrFruit = strFruit.Split(" ".ToCharArray());
// Выполнить сортировку массива,
System.Array.Sort(arrFruit);
// Записать отсортированные данные из массива обратно в строку.
strFruit = System.String.Join(" ", arrFruit);
```

У классов из пространства имен *System* многие методы доступны напрямую, беи создания объекта на основе соответствующего класса. В Visual Basic .NET такие методы называются *общими* (shared), а в Visual C# — *статическими* (static). Общие и статические члены можно вызывать прямо из класса, обращаясь к нему по имени. В показанном выше коде так и делается: *System.Array.Sort*. Еще один класс со статическими (общими) членами - *Math*. В следующем примере используются его методы *Pi* и *Pow*:

Visual Basic .NET

```
' Получить площадь окружности.
dblCircArea = System.Math.Pi * System.Math.Pow(intRadius, 2)
```

Visual C#

```
// Получить площадь окружности.
dblCircArea - System.Math.PI * System.Math.Pow(intRadius, 2);
```

В обшей сложности .NET Framework включает около сотни пространств имен, в таблице 1-4 описаны лишь самые востребованные. Полный список пространств имен .NET Framework можно найти в разделе «Class Library» справочной системы Visual Studio .NET.

Языки программирования

Платформа ASP.NET (как и .NET Framework в целом) не зависит от языка программирования. Это означает, что можно писать код на любом языке, для которого существует CLS-совместимый компилятор. Microsoft не только создала собственные языки программирования, но и разработала (в сотрудничестве с другими производителями) поддержку .NET для таких языков, как Perl, Pascal, Eiffel, Cobol, Python, Smalltalk и др.

В этой книге описано создание Web-приложений на Visual Basic .NET и Visual C#. Эти языки функционально эквивалентны, то есть предоставляют равные возможности для создания Web-приложений, отличия между ними касаются лишь синтаксиса и стиля написания кода.

Большинство программистов, скорее всего, предпочтут язык, с которым они уже знакомы. Те, кто сейчас пишет на Visual Basic, будут создавать Web-приложения с помощью Visual Basic .NET; а программирующим на C и C++ подойдет Visual C#.

Если вы недавно занимаетесь программированием или хотите обогатить свои знания, освоив новый язык, то можете изучать Visual Basic .NET и Visual C# одновременно. В случае разработки Web-приложений это еще проще, так как большинство задач решается в них через классы .NET Framework. Это означает, что написанный на Visual Basic .NET и Visual C# код Web-приложений зачастую выглядит почти идентично.

В таблице 1-5 вы найдете сводку ключевых отличий между Visual Basic .NET и Visual C#, о которых полезно помнить, планируя переход на другой язык программирования.

Таблица 1-5. Основные отличия языков Visual Basic .NET и Visual C#

Возможности языка	Visual Basic .NET	Visual C# .NET

Чувствительность к регистру символов	Нет
	`response.write("Yo") ' Верно`
		Есть
		`response.write("Yo"); //Это ошибка!`
		`Response.Write("Yo"); // Верно`

Функциональные блоки	Объявляются специальными операторами
	`Sub Show(strX as String)`
	`Response.Write(strX)`
	`End Sub`
		Объявляются фигурными скобками
		`void Snow (string strX)`
		`{`
		`Response.Write(strX);`
		`}`

Преобразование типов	По умолчанию разрешено неявное преобразование типов:
	`Dim intX As Integer`
	`intX - 3.14 ' Это допустимо`
	Можно ограничивать набор типов, для которых разрешено преобразование, помещая в начало модуля оператор
	`Option Strict On`
		Преобразование типов выполняется явно с использованием приведения:
		`int intX;`
		`intX = 3.14; // Ошибка!`
		`intX - (int)3.14; // приведение`
		// допустимо.
		либо специальных методов:
		`string strX;`
		`strX = intX.ToString();`

Массивы	Элементы массива задают при помощи круглых скобок: `arrFruit(1) = "Apple"`	Элементы массива задают при помощи квадратных скобок: `arrFruit[1] = "Apple";`
Методы	При вызове метода без аргументов скобки можно опустить: `strX - objX.ToStrinc`	При вызове метода после его имени всегда должны стоять скобки: `strX - objX.ToString();`
Завершение операторов	Операторы завершаются символом «возврат каретки»: `Response.Write(«Hello»)`	Операторы завершаются точкой с запятой: `Response.Write(«Hello»);`
Символ продолжения строки	Символ подчеркивания (_) свидетельствует о том, что строка продолжается: `intX - System.Math.Pi * intRadius`	Оператор завершается символом «;» и при необходимости может продолжаться на следующей строке: `intX = System.Math.PI * intPadius;`
Строковые операторы	Для соединения строк используются знаки & и +: `strFruit = "Apples" & "Oranges"`	Для соединения строк используется знак +: `strFruit - "Apples" + "Oranges";`
Операторы сравнения	Для сравнения значений используют операторы =, >, <. >=, <= и О: `If intX >= 5 Then`	Для сравнения значений используют операторы ==, >, <, >=, <= и !=: `if (intX >= 5)`
Отрицание	Логическое отрицание выражается ключевым словом *Not*. `If Not IsPostBack The-`	Логическое отрицание выражается оператором Л `if (!IsPostBack)`

Таблица 1-5. (*окончание*)

Возможности языка	Visual Basic .NET	Visual C# .NET
Сравнение объектов	Для сравнения переменных-объектов используется ключевое слово *is:* `If objX Is cbjY Then`	Для сравнения переменных-объектов используется оператор ==: `if (objX == objY)`
Проверка существования объекта	Для проверки существования объекта используется ключевое слово *Nothing* или функция *IsNothing*: `If IsNothing(objX) Then`	Для проверки существования объекта используется ключевое слово *null:* `if (objX == null)`

Помимо различий, указанных в таблице 1-5, эти языки существенно отличаются ключевыми словами, поэтому все примеры кола в этой книге приводятся на обоих языках. Полное сравнение Visual Basic .N ET, Visual C# и других языков от Microsoft дано в разделе «Language Equivalents» справочной системы Visual Studio .NET.

Занятие 3. Работа в Visual Studio .NET

Среди особенностей среды разработки Visual Studio .NET следует отметить новые типы окон и новые способы управления ими, а также более тесную интеграцию с содержимым Интернета. В этом занятии мы кратко рассмотрим новые возможности Visual Studio .NET, а также ряд функций отладки и справочной системы, пришедших из прежних версий, с точки зрения Web-программирования.

Если у вас есть опыт работы с прежними версиями Visual Studio .NET и вы хотите пропустить этот раздел, знайте: в новой версии Visual Studio .NET нельзя модифицировать код приложения во время отладки без перезапуска приложения: функция «edit-and-continue» больше не доступна в языках Visual Basic .NET и Visual C#.

Изучив материал этого занятия, вы сможете:

- ✓ использовать начальную страницу для открытия новых и существующих проектов, получения последних сведений о программных продуктах, а также для настройки среды разработчика;
- V назвать два типа окон Visual Studio .NET и освобождать на экране максимум места для редактирования документов при помощи функции Auto Hide;
- ✓ пользоваться инструментом Clipboard Ring в окне Toolbox для копирования и вставки;
- ✓ редактировать Web-формы и HTML-страницы в графическом режиме или в виде HTML-текста;
- ✓ автоматизировать написание кода при помощи возможностей Code Editor, **а также включать и выключать их, меняя параметры Visual Studio** .NET;
- ✓ **компоновать, запускать** и отлаживать приложения в Visual **Studio** .NET;
- ✓ пользоваться **встроенной справочной** системой **и устанавливать в ней фильтр языка программирования.**

Продолжительность занятия — около 30 минут.

Начальная страница Visual Studio .NET

Начальная страница, или Start page (рис. 1-8), — первое, что вы видите после запуска Visual Studio .NET. Она содержит ряд панелей, которые облегчают поиск нужной информации, а также исполнение некоторых общих задач (например, помогают открыть недавно открывавшийся файл).

Панель Projects, показанная на рис. 1-8, содержит ссылки на четыре последних проекта, сохраненных в Visual Studio .NET. Любой из них можно открыть, щелкнув ссылку с именем нужного проекта. Кроме того, здесь предусмотрены средсгва для создания нового проекта или открытия существующего, но отсутствующего в списке последних проектов. Для этого нужно щелкнуть соответствующую кнопку на панели Projects.

Слева на начальной странице располагается список разделов, содержащих последние сведения о Visual Studio .NET и других программных продуктах Microsoft, материалы по разным вопросам программирования, список провайдеров хостинга и массу другой полезной информации. Чтобы вывести содержимое раздела, достаточно щелкнуть соответствующую ссылку (рис. 1-9).

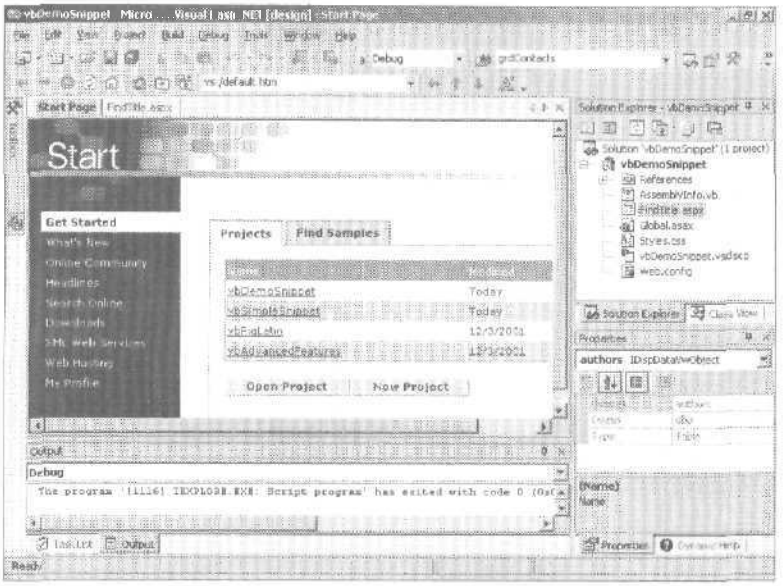


Рис. 1-8. Начальная страница Visual Studio .NET

Содержимое начальной страницы динамично, большая часть информации обновляется из Интернета, что гарантирует ее актуальность. Полезно время от времени заглядывать в разделы Headlines и Downloads, чтобы быть в курсе последних новостей.

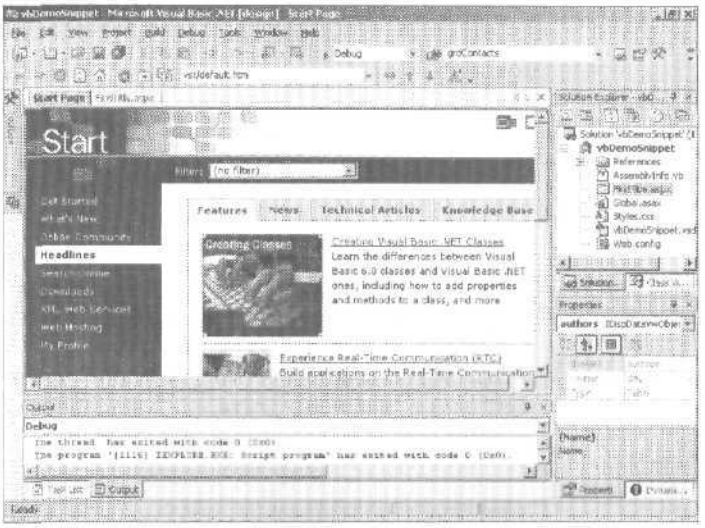


Рис. 1-9. Раздел Headlines

Особый интерес для разработчиков Web-приложений представляет раздел Web Hosting, показанный на рис. 1-10. В нем располагаются ссылки на Web-сайты поставщиков услуг хостинга Web-приложений ASP.NET.

Ряд провайдеров (например, Brinkster) предлагают ограниченный набор услуг хостинга бесплатно. Эта возможность чрезвычайно полезна при изучении ASP.NET, поскольку

она позволяет опубликовать свою работу, чтобы сделать ее доступной всему миру, без труда и расходов, связанных с созданием собственного Web-сервера.

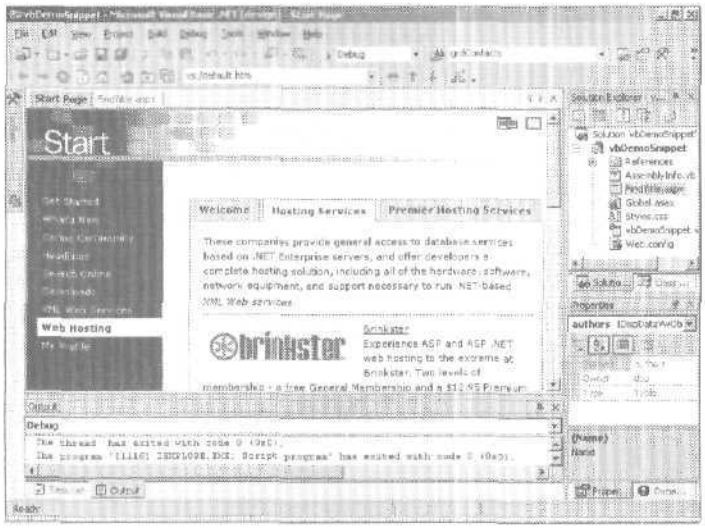


Рис. 1-10. Раздел Web Hosting

Раздел My Profile, показанный на рис. 1-11, позволяет настраивать Visual Studio .NET с учетом личных предпочтений.

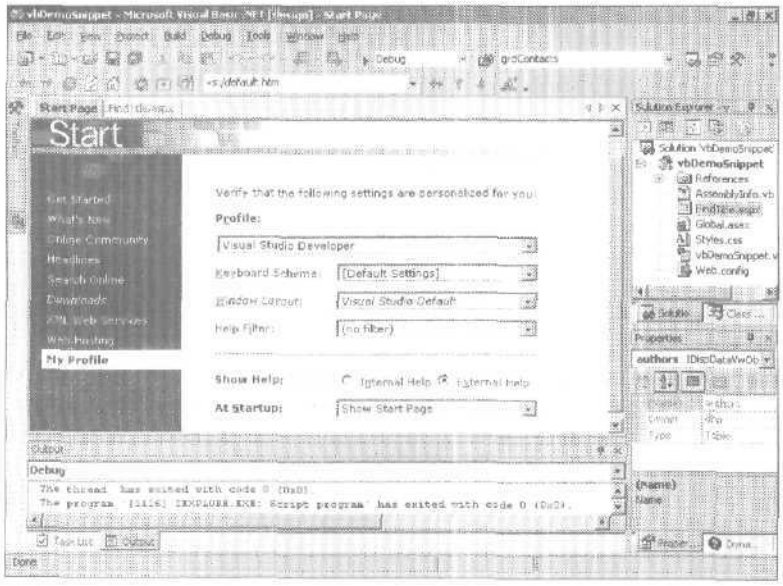


Рис. 1-11. Раздел My Profile с персональными параметрами среды разработки

Эти параметры позволяют изменить размещение окон Visual Studio .NET, заданное по умолчанию, выбрать предпочитаемый язык программирования, а также определить, как следует отображать содержимое справки: прямо на панелях Visual Studio .NET либо в от-

дельном окне (как отдельное приложение). Поскольку в Visual Studio. NET много других окон, места для окна Help остается мало, так что лучше выбрать External Help, если, конечно, у вас не 19-дюймовый монитор.

Типы окон Visual Studio .NET

В Visual Studio .NET два типа окон: окна документов и инструментальные окна. Первые отображают содержимое приложения: формы, Web-страницы и код. Одновременно можно открыть несколько окон документов; для переключения между ними достаточно щелкнуть ярлычки, расположенные вверху экрана (рис. 1-12).

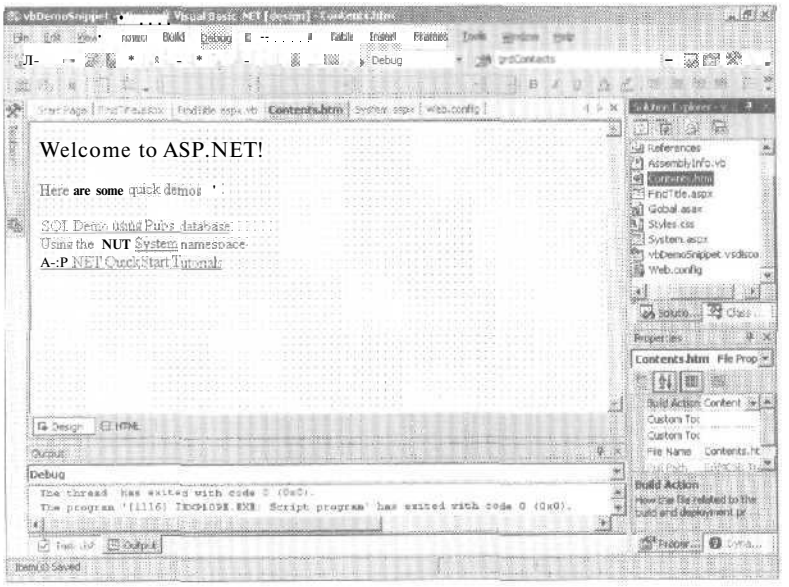


Рис. 1-12. Окно **документа**

В инструментальных окнах отображаются компоненты приложений: элементы управления, соединения с базами данных, а также классы и свойства, используемые в проектах. Инструментальные окна располагаются слева и справа от окна документа. Их разрешается настроить так, чтобы можно было прятать и снова открывать их, щелкая ярлычки окон (рис. 1-13).

Чтобы инструментальное окно находилось на экране постоянно, выключите кнопку Auto Hide (она располагается в правом верхнем углу окна и похожа на маленькую канцелярскую кнопку), а если щелкнуть ее еще раз, инструментальное окно снова будет скрываться, оставляя на виду лишь ярлычок. В этом режиме неиспользуемые инструментальные окна скрываются, освобождая место для редактирования документа (рис. 1-14).

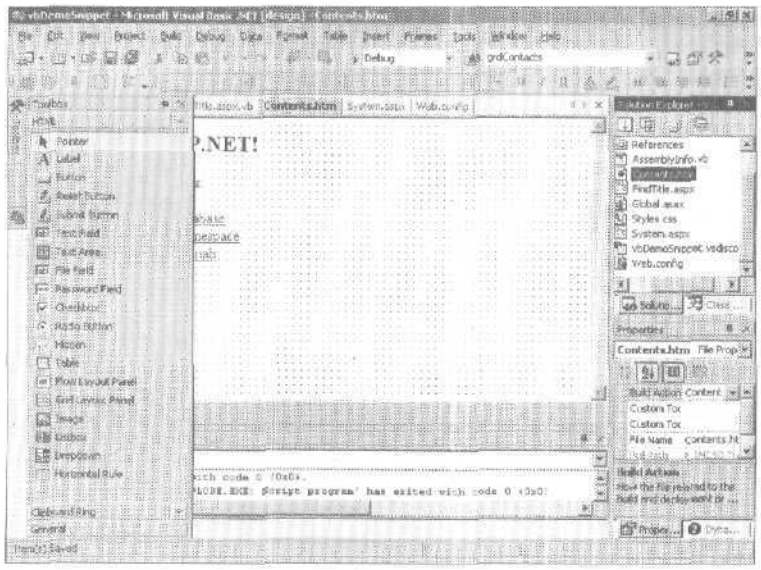


Рис. 1-13. Инструментальное окно (на примере Toolbox)

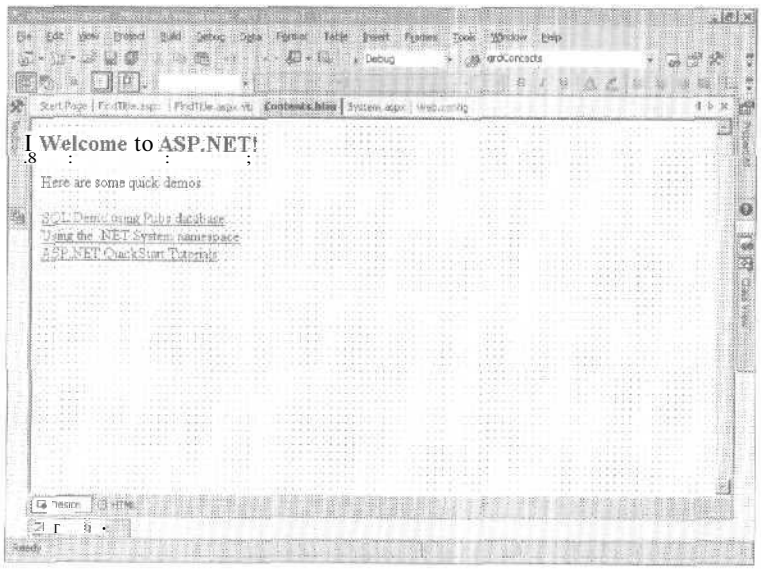


Рис. 1-14. Ярлычки скрытых инструментальных окон, расположенных вокруг окна документа

По умолчанию инструментальные окна и окна документов в Visual Studio .NET отображаются с ярлычками Эту функцию можно отключить, чтобы использовать более привычный режим, когда каждый документ отображается в обычном окне. Для этого следует вызвать окно Options (выбрав в меню элемент Tools\Options) и настроить в нем нужные параметры, как показано на рис. 1-15.

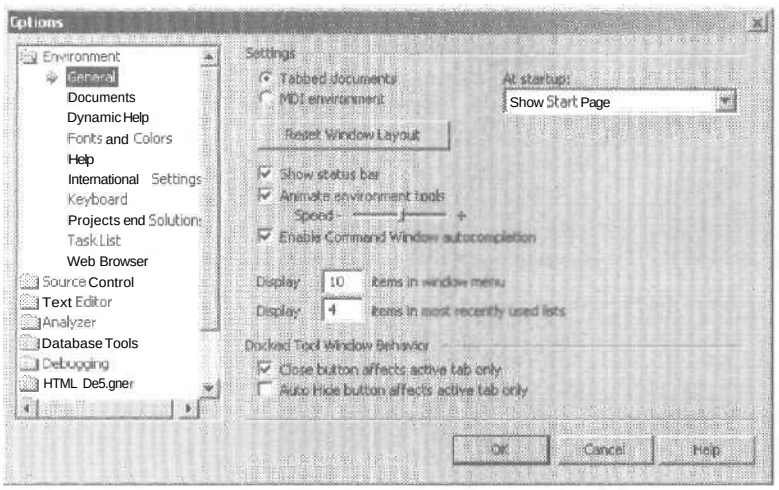


Рис. 1-15. Диалоговое окно Options

Окно Toolbox

В Visual Studio .NET окно Toolbox отображает элементы управления и компоненты, которые можно перетаскивать в окно документа. Содержимое Toolbox зависит от типа редактируемого документа. Например, при редактировании Web-формы Toolbox содержит серверные элементы управления, HTML-элементы управления, элементы управления, связанные с данными, и другие компоненты, которые можно размешать на поверхности Web-форм (рис. 1-16).

Как видно из рис. 1-16, компоненты в окне Toolbox упорядочены по категориям. Щелкнув категорию, вы отобразите элементы из этой категории. Содержимое окна Toolbox можно прокручивать, щелкая стрелки, расположенные сверху и снизу списка компонентов.

Когда текущим документом является код, в окне Toolbox находится только один инструмент — Clipboard Ring (рис. 1-17). Clipboard Ring хранит последние 20 элементов, вырезанных или скопированных в буфер обмена (нажатием Ctrl+X или Ctrl+C соответственно) — любой из них вы можете вставить в документ.



Рис. 1-16. Окно Toolbox

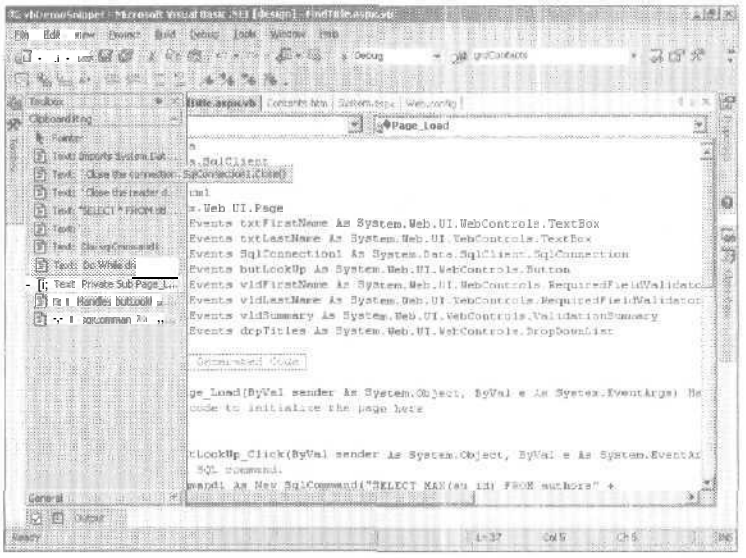


Рис. 1-17. Clipboard Ring в окне Toolbox

Чтобы сделать это, щелкните элемент в Clipboard Ring и перетащите в нужное место. **Если** навести указатель мыши на элемент в Clipboard Ring, Visual Studio отобразит этот элемент полностью, чтобы облегчить его просмотр.

РедактированиеWeb-документов

Web-формы и HTML-документы можно редактировать в графическом режиме с использованием drag-and-drop, как обычные Windows-формы, либо в текстовом виде. Для переключения между режимами достаточно щелкнуть ярлычки Design и HTML, расположенные внизу окна документа (рис. 1-18).

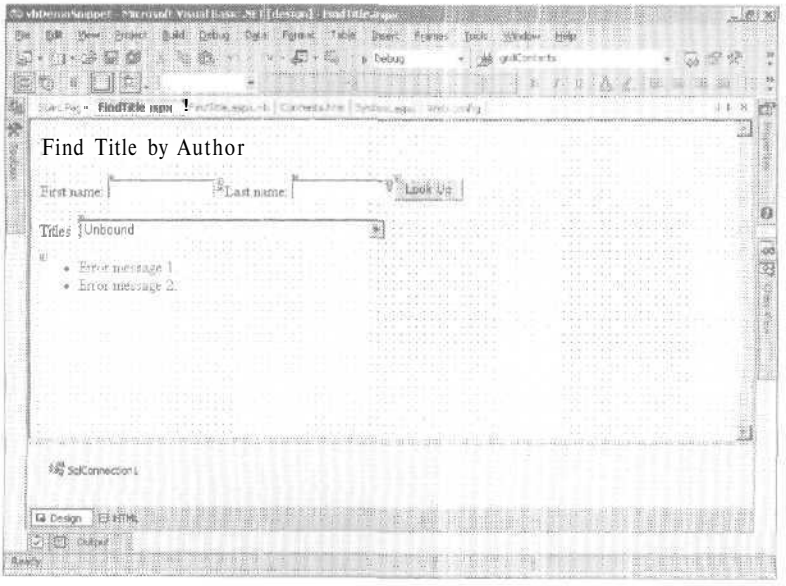


Рис. 1-18. Web-документ в режиме конструирования (Design)

Некоторые задачи невозможно решить в графическом режиме, поэтому приходится редактировать Web-документы в текстовом виде. Если вы знаете **HTML,** то этот способ может оказаться даже удобнее использования графических инструментов. Поддерживаемая Visual Studio .NET технология IntelliSense помогает завершать элементы HTML, как показано на рис. 1-19.

Для просмотра результатов изменений, внесенных в режиме HTML, стоит переключиться обратно в режим Design; для этого достаточно щелкнуть ярлычок Design, расположенный внизу окна документа.

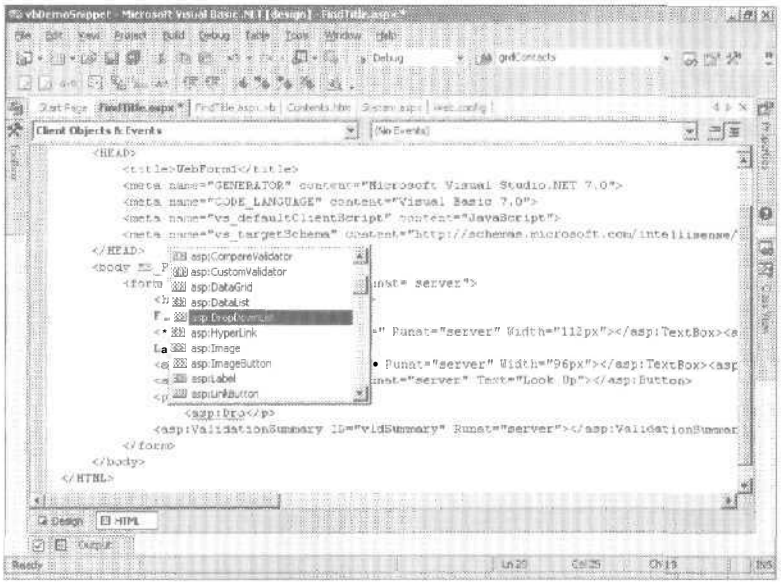


Рис. 1-19. Технология IntelliSense в действии при редактировании HTML-текста

Редактирование кода

С каждой страницей Web Forms связан файл, содержащий объект кода этой страницы, Эти файлы автоматически генерируются при создании новой Web-формы. Имя файла с кодом составляется из имени Web-формы и расширения .vb или .cs (рис. 1-20 и 1-21).

В каждом файле кода Visual Studio .NET генерирует определение класса, инициализирующие процедуры и обработчик события *Page_Load.* Не изменяйте код в секциях с меткой *Web Form Designer Generated Code,* поскольку в дальнейшем Visual Studio .NET может автоматически модифицировать его, уничтожив ваши изменения.

Чтобы скрыть сгенерированный код, следует щелкнуть знак «минус» (–) слева от директивы #Region. При этом область сворачивается, а знак «минус» меняется на «плюс» (+), щелкнув который, вы снова раскроете эту область. Аналогичным образом сворачивают и раскрывают и другие блоки кода, например определения классов и процедуры.

Инструмент Visual Studio .NET под названием Code Editor также поддерживает функцию autocomplete, автоматически завершающую ключевые слова и члены классов при помощи технологии IntelliSense (рис. 1-22).

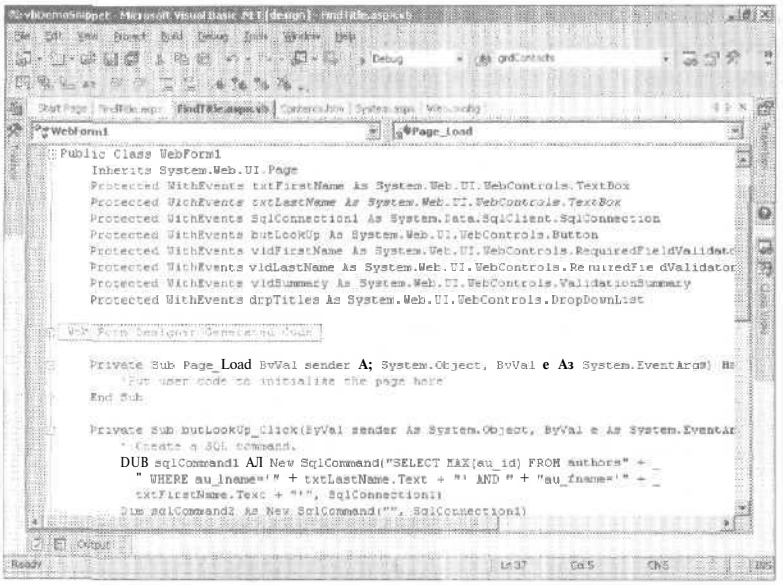


Рис. 1-20. Файл объекта кода Web-формы (на Visual Basic .NET)



Рис. 1-21. Файл объекта кода Web-формы (на Visual C#)

Если вы **пишете на Visual Basic, то после завершения** ввода **каждой строки функция** autocomplete будет делать **первые буквы** ключевых **слов и имен членов** заглавными. **Однако в программе, написанной** на Visual C#, Visual Studio .NET **не распознает ключевое слово или имя члена,** если **при его** вводе **использован** неверный **регистр.** Дело в том, **что** Visual C#, в **отличие от** Visual **Basic** .NET, чувствителен к **регистру символов.**

Кроме того, после ввода каждой строки Code Editor выделяет синтаксические ошибки и необъявленные переменные, подчеркивая их волнистой линией. При наведении указателя мыши на выделенную ошибку отображается ее описание (рис. 1-23).

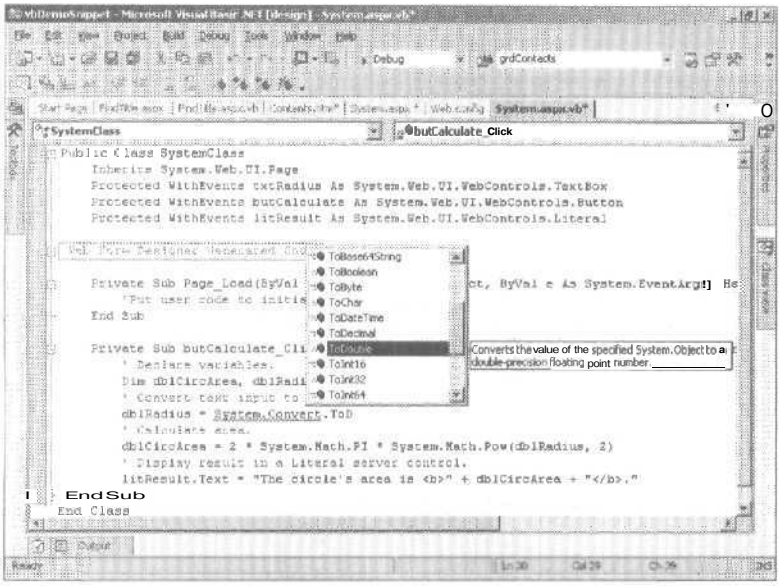


Рис. 1-22. Функция autocomplete

Большинство автоматизирующих возможностей Code Editor можно включать и отключать, изменяя параметры в диалоговом окне Options (рис. 1-15). Кроме того, в нем предусмотрена корректировка автоматических абзацных отступов, завершения блоков кода и других параметров, определяющих работу с языком.

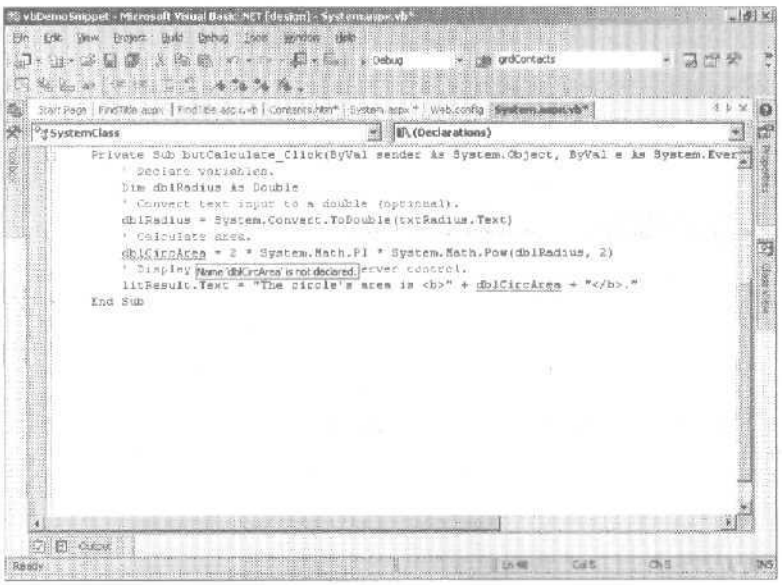


Рис. 1-23. Автоматическое выделение ошибок в окне Code Editor

Окно Solution Explorer

Visual Studio .NET упорядочивает приложения при помощи проектов и решений *Проект* (project) — это набор файлов, из которых в итоге компонуется исполняемый файл. *Решение* (solution) — это группа проектов, образующих функциональную единицу. Файлы, составляющие решение, можно просматривать в окне Solution Explorer, показанном на рис. 1-24.

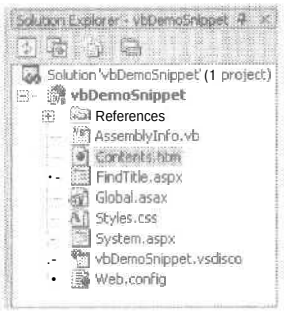


Рис. 1-24. Окно Solution Explorer

Проект, название которого выделено в окне Solution Explorer полужирным шрифтом, является стартовым. *Стартовый проект* (start-up project) запускается первым по щелчку кнопки Start в Visual Sludio .NET. Если решение состоит из нескольких проектов, то стартовый проект обычно вызывает другие проекты этого решения.

Сведения о решении хранятся в файле решения, который имеет расширение .sln и по умолчанию размещается в папке My Documents (Мои документы). Эти файлы позволяют открывать решения, а файлы проектов (файлы с расширением .vbproj или .csproj) позволяют напрямую открывать проекты, расположенные в соответствующих папках. При сохранении проекта, открытого таким образом, Visual Studio .NET создаст новый файл решения.

Запуск проекта

Запустить проект в Visual Studio .NET можно одним из следующих способов; щелкнуть кнопку Start на панели инструментов, выбрать из меню Debug команду Start или нажать клавишу F5. После запуска проекта Visual Studio .NET компонует его файлы, показывая в окне Task List сообщения об ошибках, если таковые возникнут (рис. 1-25).

Дважды щелкнув описание ошибки в окне Task List, вы быстро найдете строку с ошибкой, чтобы исправить ее.

Если во время компоновки ошибки не обнаружены, Visual Studio .NET запускает приложение в режиме отладки, а в случае Web-приложения — запускает Internet Explorer и открывает в нем начальную страницу Web-приложения (рис. 1-26). Если при исполнении приложения в режиме отладки возникает ошибка, Visual Studio .NET показывает в браузере соответствующее сообщение (см., например, рис. 1-26).

Дальше можно выполнить одно из следующих действий:

- если причина ошибки известна, стоит остановить приложение, закрыв окно браузера, и вернуться в Visual Studio .NET, чтобы исправить ошибку;
- если причина ошибки неизвестна, следует щелкнуть кнопку Back (Назад) в браузере и переключиться в Visual Studio .NET, чтобы установить в коде точку прерывания перед предполагаемым местом возникновения ошибки. Затем, переключившись обратно в браузер, попробуйте еще раз выполнить действие, вызывавшее ошибку. Встретив точку

прерывания, Visual Studio .NET остановит исполнение приложения. Далее путем пошагового исполнения вы локализуете ошибку.

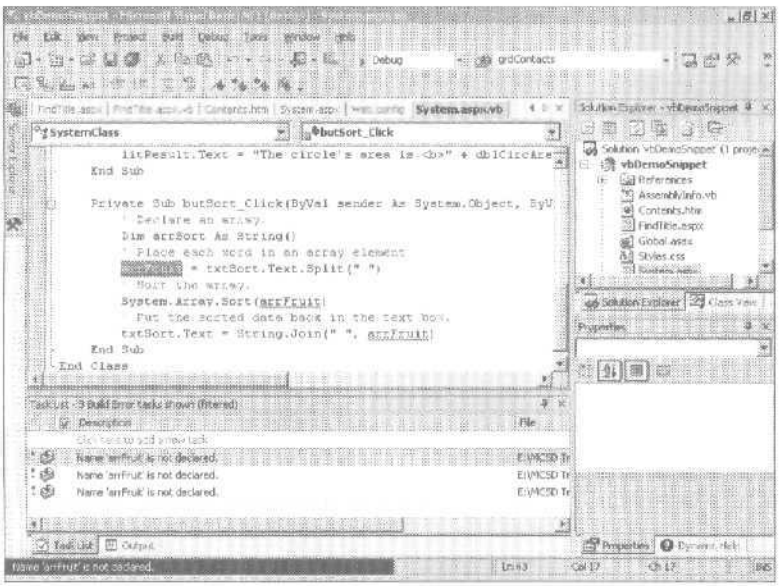


Рис. 1-25. Сообщения об ошибках, возникших при компоновке проекта

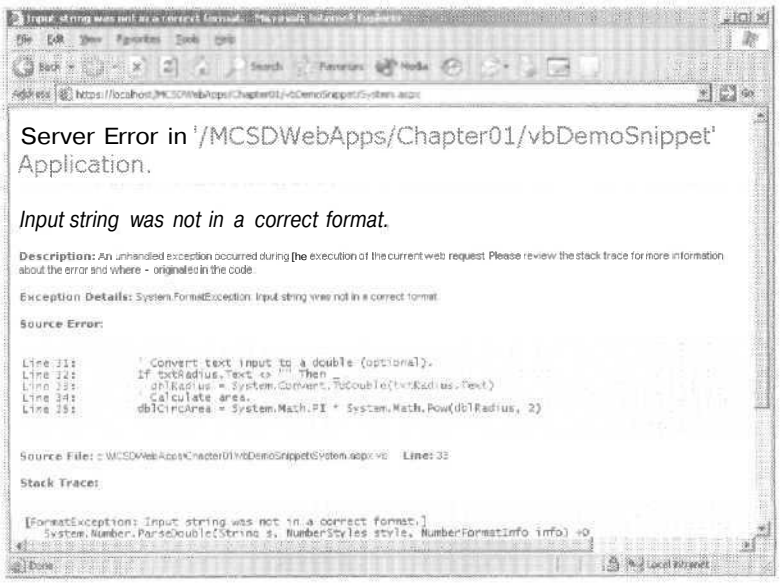


Рис. 1-26. Сообщение об ошибке периода выполнения в проекте Web Forms

После обнаружения причины ошибки, до ее устранения необходимо остановить приложение. В прежних версиях Visual Studio .NET разрешалось исправлять ошибки прямо в отладочном режиме, не прерывая работы приложения.

Установка точек прерывания и просмотр значений переменных

Можно остановить исполнение проекта перед некоторой строкой кода, установив в этой строке точку прерывания. Встретив при исполнении проекта строку с точкой прерывания, Visual Studio .NET остановит приложение перед этой строкой и откроет в ее окне Code Editor, выделив, как показано на рис. 1-27.

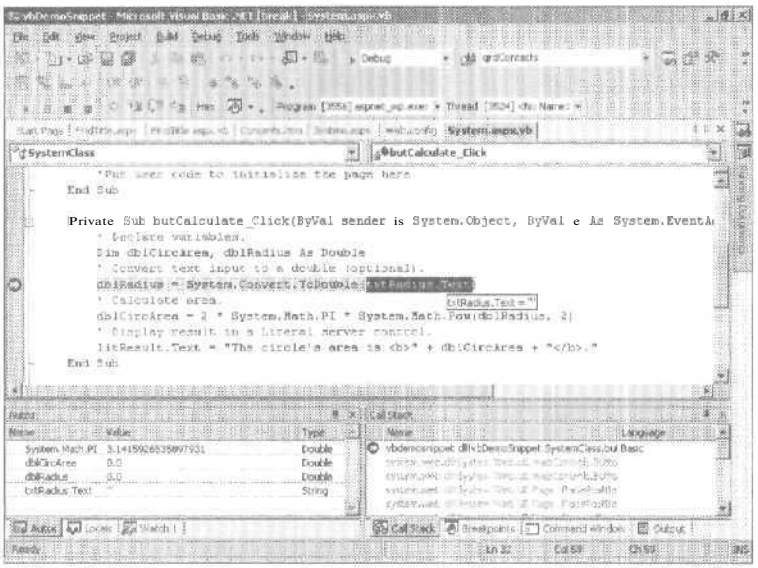


Рис. 1-27. Проект, остановленный перед **точкой прерывания**

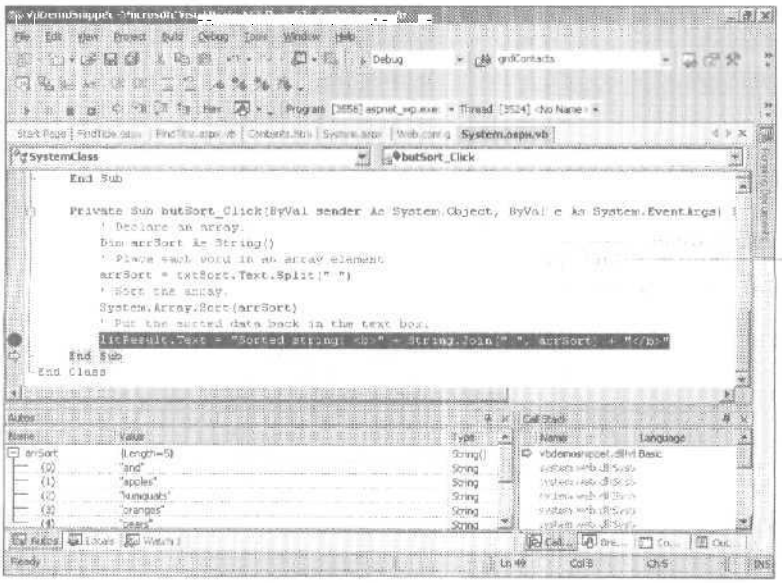


Рис. 1-28. Установка точки прерывания

Чтобы установить точку прерывания, щелкните серое поле слева от строки, перед которой нужно остановить исполнение программы, либо выделите эту строку и нажмите F9. Установленная точка прерывания показана на рис. 1-28.

После останова приложения перед точкой прерывания стоит просмотреть значения активных переменных, наводя на них указатель мыши. Для просмотра значений переменных сложного типа, таких, как переменные-объекты или массивы, следует добавить эти переменные в окно Watch, показанное на рис. 1-29.

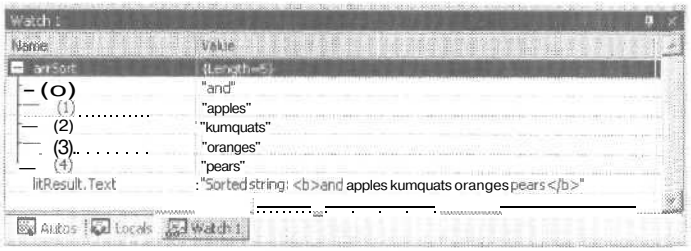


Рис. 1-29. Окно Watch

Чтобы сделать это, выберите элемент и перетащите его в окно Watch. Для просмотра значений вложенных элементов (например, элементов массива или свойств объекта) щелкните знак «+» справа от сложного элемента в окне Watch.

Пошаговое исполнение

Чтобы возобновить исполнение приложения, остановленного перед точкой прерывания, щелкните кнопку Continue на панели инструментов либо нажмите F5. Можно также исполнять приложение построчно, нажимая F10 или F11.

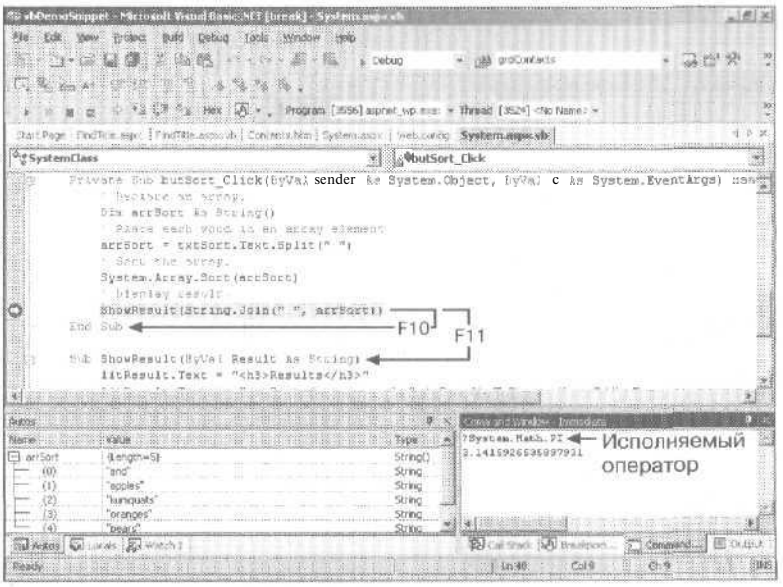


Рис. 1-30. Различные способы пошагового исполнения

F10 заставляет при вызове процедуры исполнить ее как одну команду. Другими словами, процедура исполняется *без захода* в нее (step over), после чего исполнение останавливается на следующей после вызова этой процедуры строке. Нажатие клавиши F11 вызывает исполнение процедуры *с заходом в* нее (step in), при этом исполнение останавливается на первой строке вызванной процедуры. Окно Command позволяет исполнить строку кода вне контекста проекта. Все эти способы пошагового исполнения показаны на рис. 1-30.

Результаты исполнения кода в окне Command выводятся в том же окне строкой ниже. Например, в результате исполнения оператора *?System.Math.PI* на следующей строке выводится число 3.1415926535897931.

Работа со справочной системой

В Visual Studio .NET имеется объединенная справочная система, включающая сведения по среде разработки, языкам программирования и .N ET Framework, а также информацию технической поддержки и статьи из MSDN. В зависимости от параметров, установленных на начальной странице в диалоговом окне Options, содержимое справочной системы отображается в окне документа либо в отдельном окне вне Visual Studio .NET.

В справочной системе предусмотрены три поисковых окна: Contents, Index и Search. Они работают подобно инструментальным окнам Visual Studio .NET: их можно «пристыковывать» к другим окнам, а также прятать и открывать, щелкая их ярлычки (рис. 1-31).

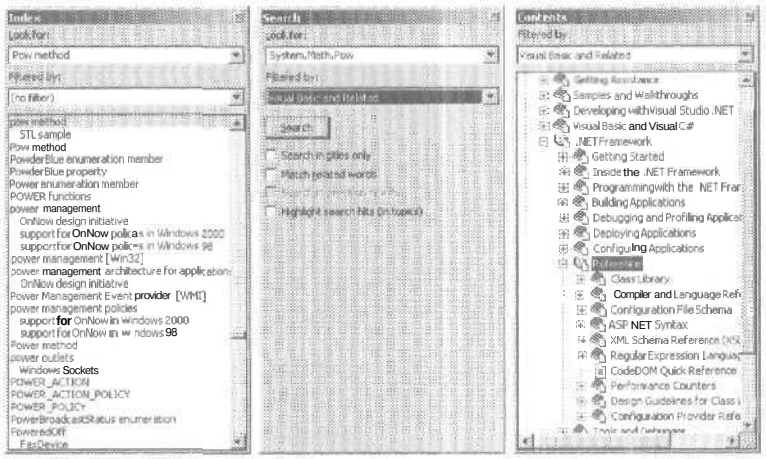


Рис. 1-31. Поисковые окна справочной системы

В каждом поисковом окне есть раскрывающийся список Filter, в котором выбирают нужный язык программирования или раздел для поиска. Эта возможность особенно полезна при работе с окнами Search и Index, поскольку объединенная справочная система содержит немало информации. Фильтры Visual Basic And Related и Visual C# And Related охватывают большинство разделов, которые, вероятно, пригодятся вам при изучении материала этой книги.

В разделах с примерами операторов и кода на каждой странице установлен фильтр языка. Он имеет вид кнопки с изображением воронки и расположен в левом верхнем углу страницы. Щелкнув значок фильтра, вы выведете примеры к этому разделу на другом языке программирования либо заставите справочную систему показать примеры на всех языках (рис. 1-32).

Помимо справочной системы Visual Studio .NET вам доступны материалы Web-сайта GotDotNet (*www.gotdotnet.com*), поддерживаемого Microsoft. В них можно найти учебники по ASP.NET, а также ссылки на другие Web-сайты по смежной тематике.

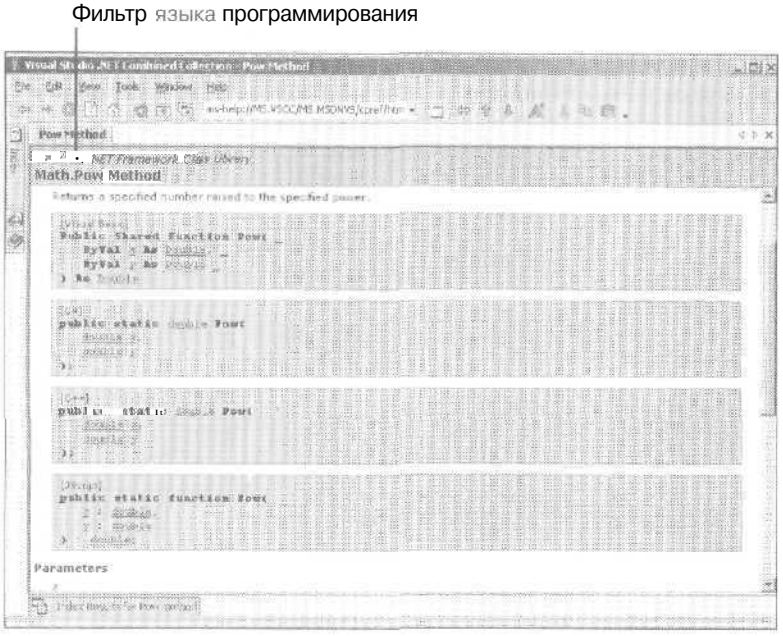


Рис. 1-32. Настройка фильтра языка

Резюме

- Существует четыре типа Интернет-приложений: Web-приложения, Web-сервисы, приложения с поддержкой Интернета и одноранговые приложения.
- Web-приложения работают на сервере, предоставляя пользователям запрошенные ими страницы, которые скомпонованы из результатов исполнения кода и статических ресурсов сервера.
- Web-приложения способны «на лету» формировать и отображать содержимое, созданное на основе динамических ресурсов сервера (например, баз данных) и информации, введенной пользователем. Так создается и отображается таблица выплат по займам на основе сведений о займе, предоставленных пользователем.
- ASP.NET — это платформа для создания Web-приложений, работающих на серверах Windows и использующих IIS и .NET Framework.
- Web-приложения состоят из информационного наполнения, исполняемого кода и конфигурационных файлов.
- Содержимое Web-приложений отображается через Web-формы, которые используют HTML-компоненты подобно обычным HTML-страницам и реагируют на инициируемые пользователем события (например, на щелчки мыши) подобно Windows-формам.
- Исполняемая часть Web-приложения хранится в .dll-файле, называемом сборкой. Сборки образуются в результате компиляции исходного текста в команды промежуточного языка (IL). Окончательная компиляция сборок осуществляется CLR непосредственно перед исполнением приложения.

- .NET Framework состоит из CLR и библиотеки классов .NET, открывающей программистам доступ к возможностям времени выполнения CLR.
- Классы .NET сгруппированы в пространства имен в соответствии с решаемой ими задачей программирования. Пространство имен может служить ориентиром при поиске нужных классов, методов и свойств.
- Начальная страница Visual Studio .NET позволяет просматривать текущую информацию о программных продуктах, открывать новые и существующие проекты, настраивать личные параметры среды разработки и подписываться на услуги хостинга,
- В режиме Design окно документа позволяет редактировать Web-формы и HTML-страницы в графическом виде, а в режиме HTML — редактировать эти документы в текстовом виде.
- Установив соответствующий фильтр языка в справочной системе, можно просматривать примеры кода на нескольких языках программирования либо на каком-нибудь одном.
- Диалоговое окно Options позволяет изменять параметры среды разработчика Visual Studio .NET.

Практикум 1. Введение в Visual Studio .NET

В этом практикуме вы познакомитесь со средой разработки Visual Studio .NET, узнаете, как подписаться на услуги хостинга и как работать с учебниками ASP.NET QuickStart. Навыки, полученные при выполнении упражнений этого практикума, послужат основой для изучения материала следующих глав.

Продолжительность практикума — около 30 минут.

Упражнение 1. Настройка среды Visual Studio .NET



Сейчас вы настроите окна Visual Studio .NET, чтобы освободить максимум места для работы, вывести содержимое справки в отдельном внешнем окне и установить фильтр языка в справочной системе. После выполнения этого упражнения окно Visual Studio .NET должно выглядеть, как на рис. 1-33.

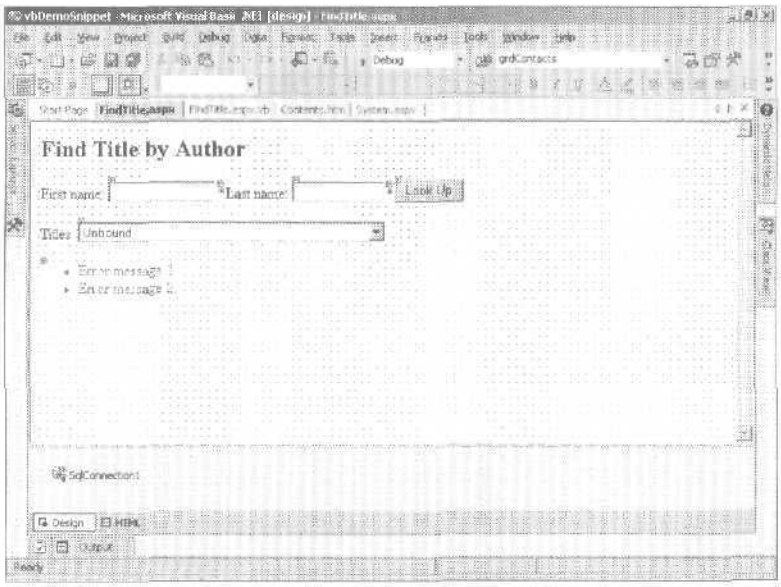


Рис. 1-33. Вид среды **Visual Studio .NET** после **настройки**

При первом запуске Visual Studio .NET отображает слева окна Server Explorer и Toolbox, а справа — окна Solution Explorer и Properties (все они отображаются в виде ярлычков).

► Как освободить **максимум места для редактирования Web-документов и кода**

- Выберите из меню элемент Window\Auto Hide All

или

- щелкните кнопку Auto Hide (это кнопка с изображением канцелярской кнопки расположена в верхнем правом углу окон Solution Explorer и Properties).

При включенной функции Auto Hide инструментальное окно спрячется, если вывести указатель мыши за его пределы, и снова откроется, если установить его над ярлычком окна, расположенным рядом с краем экрана.

По умолчанию Visual Studio .NET настроена для отображения содержимого справочной системы в окне документа. Поскольку у справочной системы много собственных окон, вложенные окна сильно загромождают экран.

► **Отображение содержимого справочной системы в собственном окне**

1. Вызовите диалоговое окно Options, выбрав из меню элемент Tools\Options.
2. В диалоговом окне Options раскройте папку Environment и щелкните Help — Visual Studio .NET покажет параметры справочной системы (рис. 1-34).

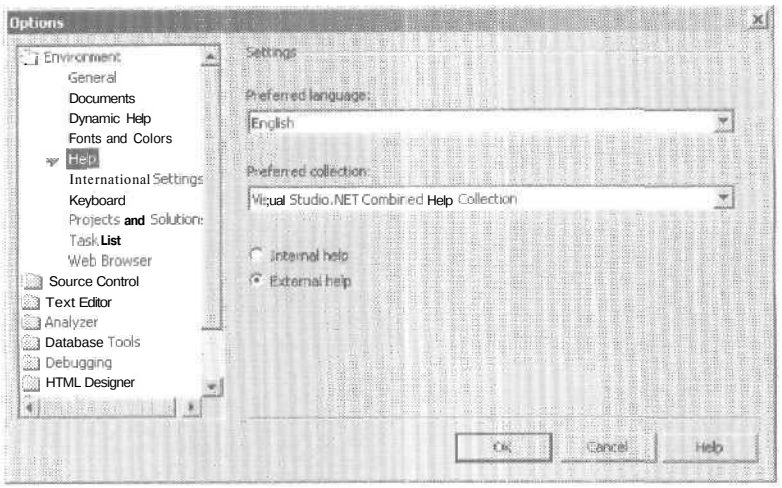


Рис. 1-34. Настройка справочной системы

3. Установите переключатель в позицию External help и шелкните OK.

Если результаты ваших экспериментов с размещением окон Visual Studio .NET вас не устраивают, можно восстановить размещение окон, заданное по умолчанию.

► **Как восстановить параметры окон по умолчанию**

1. Выберите из меню элемент Tools\Options — Visual Studio .NET откроет диалоговое окно Options.
2. В диалоговом окне Options раскройте папку Environment и шелкните General.
3. Щелкните кнопку Reset Window Layout. Visual Studio .NET покажет окно предупреждения с просьбой подтвердить ваши намерения.
4. Щелкните OK два раза, чтобы закрыть окно предупреждения и диалоговое окно Options. В результаге Visual Studio .NET восстановит размещение окон, заданное по умолчанию,

Упражнение 2. Создание учетной записи службы хостинга



Сейчас вы создадите учетную запись службы хостинга, которая позволит вам опубликовать свое Web-приложение в Интернете, чтобы каждый смог попробовать и оценить его. Учетная запись хостинга не обязагельна для занятий по материалам этой книги, поскольку Web-приложения можно исполнять и отлаживать локально, но ее наличие позволит вам не только проверить способность Web-приложения одновременно обрабатывать запросы нескольких пользователей, но и развлечься, продемонстрировав другим свои успехи в программировании.

▶ **Создание учетной записи хостинга**

1. Выберите провайдера.
2. Зарегистрируйтесь в его службе хостинга.
3. Подпишитесь на услуги хостинга и загрузите свое приложение на сервер провайдера.
4. Чтобы отобразить раздел Web Hosting в Visual Studio .NET, выберите окно документа с начальной страницей, щелкните ссылку Web Hosting и перейдите на вкладку Hosting Services. В результате откроется панель Web Hosting, показанная на рис. 1-35.

Более подробно действия, необходимые для создания учетной записи хостинга, описаны далее.

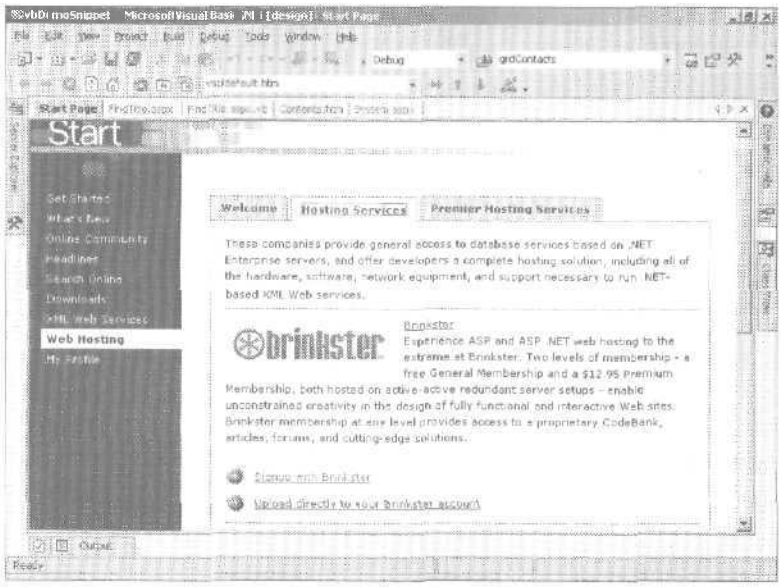


Рис. 1-35. Провайдеры хостинга Web-приложений ASP.NET

Упражнение 3. Выбор провайдера услуг хостинга

В разделе Web Hosting вы найдете список провайдеров, поддерживающих Web-приложения ASP.NET и предлагающих различные комбинации платных и бесплатных услуг хостинга. Чтобы узнать, подходит ли вам данный провайдер, посетите его сайт, щелкнув соответствующую ссылку на начальной странице Visual Studio. Помимо стоимости услуг, примите во внимание еще несколько моментов.

- Качество услуг. Есть ли на сайте провайдера пользовательская телеконференция или другое место, где можно получить ответы на свои вопросы? Если да, узнайте мнение других пользователей об этом провайдере.
- Возможность наращивания. Легко ли перейти от пробного развертывания к полноценному? Достаточно ли места выделяет провайдер? Хватит ли производительности его сервера?
- Поддержка баз данных. Услуга хостинга БД SQL Server есть у большинства провайдеров, но стоит она у всех по-разному.

Регистрация у провайдера

Примечание Для создания учетной записи необходимо зарегистрироваться в службе хостинга. Конкретная процедура зависит от провайдера, но обычно требуется выполнить действия, описанные в этом разделе.

1. В разделе Web Hosting щелкните ссылку Sign Up With для выбранного вами провайдера — Visual Studio .NET откроет в окне Document описание соответствующей процедуры регистрации (рис. 1-36).

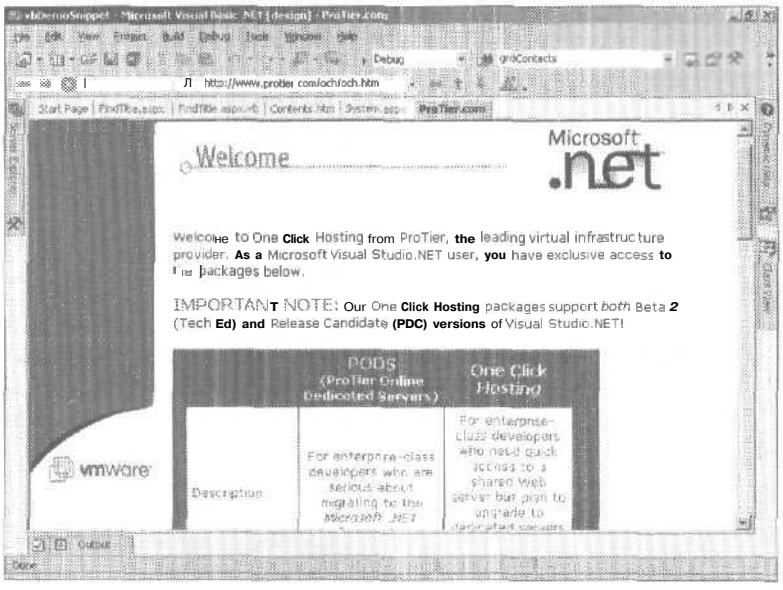


Рис. 1-36. Начальная страница провайдера ProTier

2. Следуйте инструкциям провайдера. Обычно он просит указать имя, а также адреса обычной и электронной почты. Получив эту информацию, провайдер высылает вам имя пользователя и пароль для доступа к вашей учетной записи.

Все провайдеры, указанные в разделе Web Hosting, предоставляют услугу *One-Click Hosting*, позволяя загружать готовые Web-приложения на свой сервер прямо из Visual Studio .NET (с панели Web Hosting).

Как загрузить приложение на сервер провайдера

1. В разделе Web Hosting начальной страницы Visual Studio .NET щелкните ссылку Upload Directly To Your Account для выбранного вами провайдера. Visual Studio .NET откроет в окне Document входную страницу службы хостинга, похожую на ту, что показана на рис. 1-37.
2. Введите в соответствующие поля имя пользователя и пароль вашей учетной записи (обычно провайдер присылает их по электронной почте, когда вы подпишетесь на услуги хостинга). После входа на сайт провайдера в окне Document открывается страница загрузки (рис. 1-38).
3. Выберите из списка слева папку, в которой находится загружаемое приложение, а из списка справа — папку на сервере, в которую нужно его поместить. Щелкните кнопку Upload, чтобы загрузить проект на сервер службы хостинга.

После окончания процедуры вы и все остальные смогут открыть его в браузере, набрав соответствующий URL в строке адреса. Для просмотра новых приложений подписка не требуется — она нужна лишь для загрузки приложений на сервер провайдера.

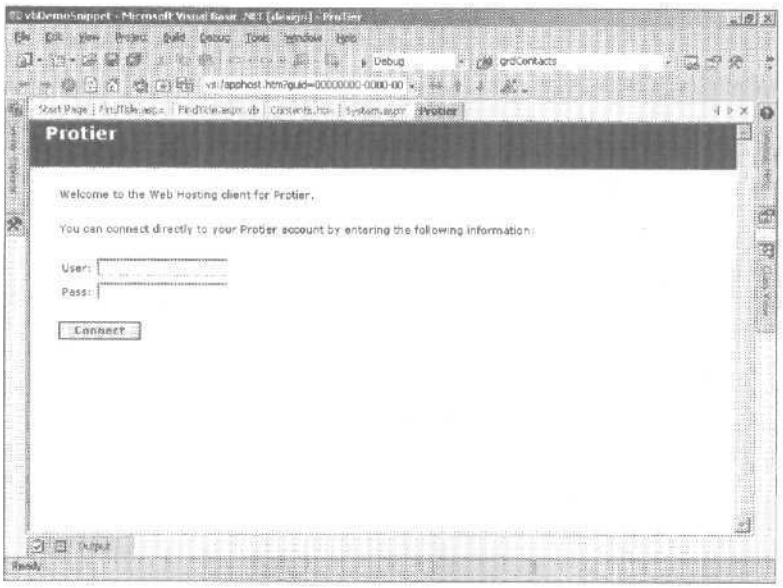


Рис. 1-37. Входная страница службы хостинга провайдера ProTier

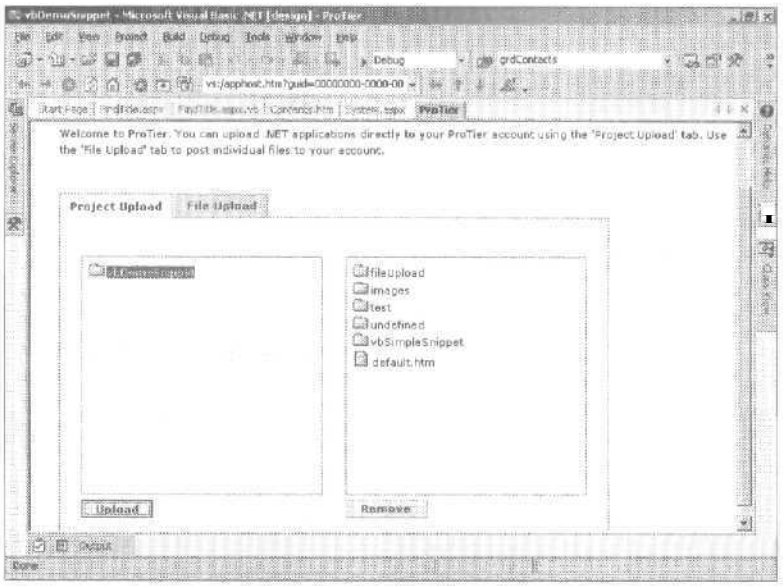


Рис. 1-38. Страница закачки у провайдера ProTier

Упражнение 4. Работа с учебниками ASP.NET QuickStart



Сейчас вы установите учебники и откроете QuickStart. Учебники QuickStart содержат сведения о .NET Framework и считаются одним из лучших источников информации для изучения программирования с использованием ASP.NET.

▶ **Установка и просмотр учебников QuickStart для ASP.NET**

1. Выберите в Главном меню элемент Start\All Programs\Microsoft .NET Framework SDK\ Samples And QuickStart Tutorials — Windows откроет страницу, показанную на рис. 1-39.
2. Щелкните ссылку Step 1: Install The .NET Framework Samples Database, чтобы установить и настроить базу данных с примерами.
3. Щелкните ссылку Step 2: Set Up The QuickStarts, чтобы установить и сконфигурировать Web-сайты, используемые QuickStart,
4. В следующий раз, когда вы откроете учебники QuickStart при помощи меню Start. Internet Explorer пропустит установочную страницу (рис. 1-40) и сразу перейдет на страницу, показанную на рис. 1-41.
5. Для просмотра содержимого учебников (рис. 1-41) щелкните ссылку ASP.NET QuickStarts на странице, показанной на рис. 1-40.

В учебниках ASP.NET QuickStart показано решение различных задач программирования на языках Visual Basic .NET, Visual C# и JScript. В отличие от этой книги, в учебниках QuickStart большинство примеров Web-форм показано в режиме HTML, а не Design.

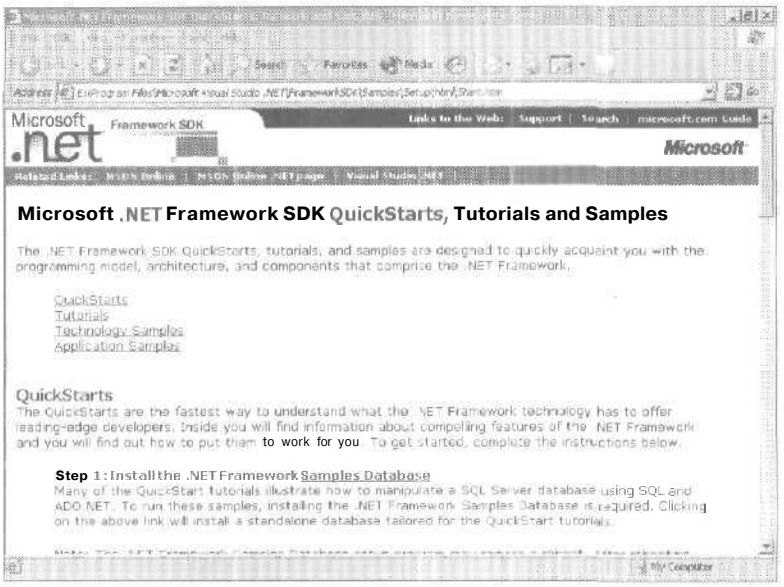


Рис. 1-39. Страница ASP.NET QuickStart Tutorial

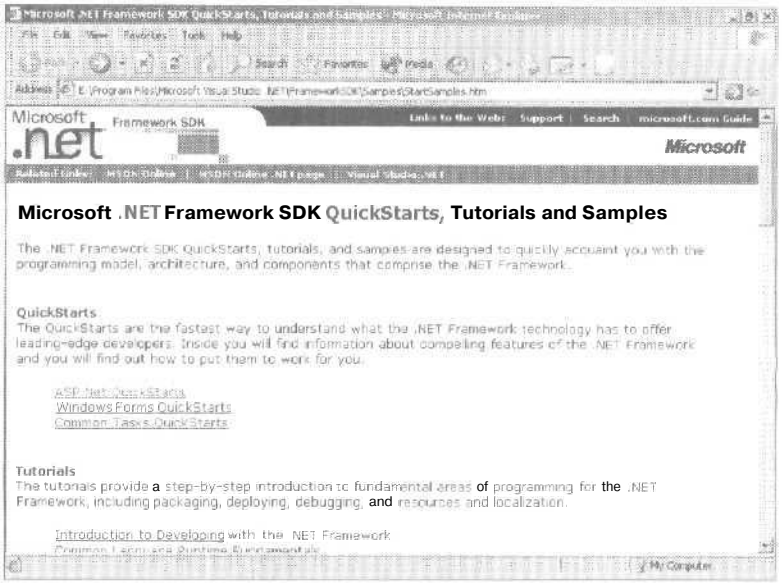


Рис. 1-40. Страница установки Microsoft .NET Framework QuickStarts, **Tutorials and** Samples

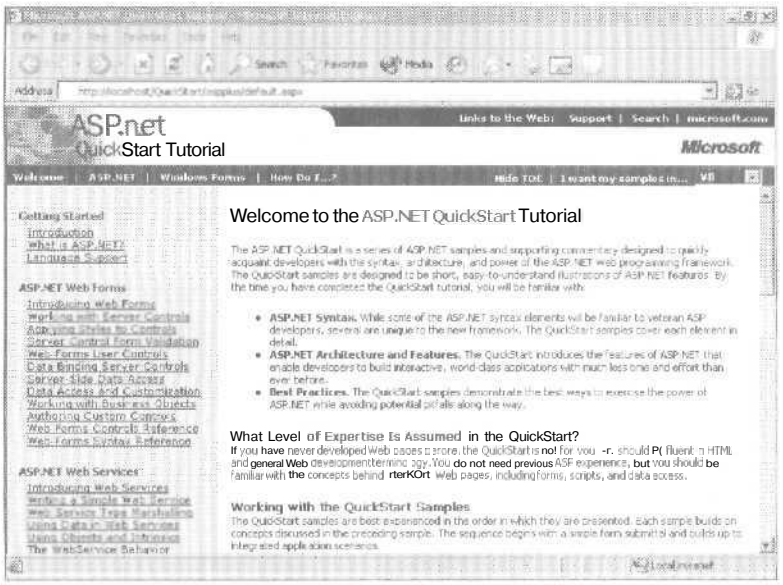


Рис. 1-41. Страница **Microsoft .NET Framework QuickStarts, Tutorials and Samples**

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. Назовите три отличия Web-приложений ASP.NET от традиционных приложений для Windows.

2. Перечислите основные компоненты .NET Framework.

3. Как в Visual Studio .NET восстановить параметры окон, заданные по умолчанию?

4. Почему Code Editor не сможет автоматически завершить следующую строку (на языке Visual C#)?

```
int intX = system.math
```

5. В каких случаях нельзя использовать ASP.NET для создания Web-приложений?

ГЛАВА 2

Создание приложений Web Forms

Занятие 1. Создание проекта Web-приложения 42

Занятие 2. Обработка событий 52

Занятие 3. Обработка данных 61

В этой главе

Из этой главы вы узнаете, как создают Web-приложения на Microsoft Visual Studio .NET и ASP.NET. Вы научитесь создавать новые проекты, конструировать простейшие пользовательские интерфейсы, писать код для обработки событий и сохранять данные в приложении. Поскольку инфраструктура ASP.NET интегрирована со службами Microsoft Internet Information Services (IIS), вы узнаете, что такое IIS и как использовать его для управления файлами Web-приложения.

Прежде всего

Для занятий по материалам это главы вам потребуются:

- Visual Studio .NET, Internet Information Services (IIS) и Microsoft FrontPage Server Extensions;
- базовые навыки работы в Visual Studio .NET (умение размещать элементы управления на форме и добавлять к ним код, а также запускать программы). Дополнительные сведения об этом см. в учебниках «Quick Start» из Visual Studio .NET.

Занятие 1. Создание проекта Web-приложения

Web-формы — это объекты Web-приложения, определяющие его пользовательский интерфейс. От размещенных на Web-форме текста и элементов управления зависит, что увидит пользователь, запустивший созданное вами приложение, — в этом Web-формы сходны с формами Windows-приложений.

Однако, в отличие от Windows-приложений, Web-приложения работают на сервере, обмениваясь данными с клиентами через Интернет. Поэтому в Visual Studio .NET создание проекта Web-приложения и управление его файлами существенно отличаются от аналогичных процедур для Windows-приложений. Эти отличия рассматриваются в следующих разделах.

Изучив материал этого занятия, вы сможете:

- ✓ создавать новые проекты Web Forms в Visual Studio .NET;
- ✓ задавать физический каталог для размещения файлов проекта;
- ✓ размещать на Web-форме серверные элементы управления и писать код для обработки их событий;
- ✓ запускать Web-приложения в среде разработки;
- ✓ рассказать о назначении файлов проекта Web Forms.

Продолжительность занятия — около 35 минут.

Создание **нового** Web-приложения

Чтобы создать в Visual Studio .NET новое приложение Web Forms, прежде всего нужно открыть новый проект. Для каждого типа приложений в Visual Studio .NET имеется соответствующий шаблон. Шаблон Web-приложений называется «ASP.NET Web Application». При создании нового проекта на основе этого шаблона Visual Studio .NET генерирует файл проекта, пустую Web-форму и ряд вспомогательных файлов, необходимые приложению.

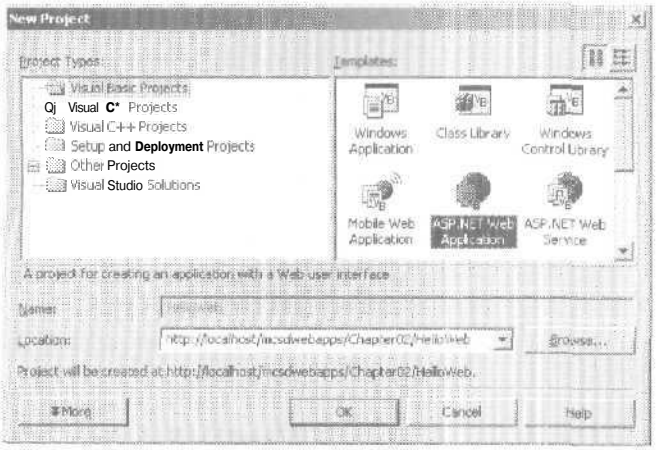


Рис. 2-1. Диалоговое окно New Project

Чтобы создать проект Web-приложения в Visual Studio .NET, выполните следующие действия.

1. На начальной странице Visual Studio .NET щелкните ссылку New Project. В результате откроется диалоговое окно New Project, показанное па рис. 2-1. Visual Studio .NET размешает Web-приложения в виртуальном каталоге localhost.
2. В диалоговом окне New Project выберите шаблон ASP.NET Web Application, введите в поле Location имя нового проекта и щелкните OK.

При создании проекта Web-приложения Visual Studio .NET создает новый каталог и генерирует в нем файлы проекта. Visual Studio .NET называет новый каталог по имени проекта и размешает его в корневом каталоге Web-сайта по умолчанию, который обслуживается IIS на данном компьютере. В поле Location диалогового окна New Project указан адрес каталога проекта в виде *http://localhost/имя_проекта*.

Управление проектом при помощи IIS

Важно понимать, что Web-приложение может существовать только в *виртуальном каталоге* (virtual folder), опубликованном IIS. Виртуальный каталог — это разделяемый ресурс, идентифицируемый псевдонимом, который представляет физический каталог, расположенный на сервере. Если в диалоговом окне New Project попытаться выбрать для размещения Web-приложения обычный каталог (например, c:\MyFiles), кнопка OK будет недоступна, поскольку Web-приложения нельзя создавать в обычных каталогах.

Виртуальный каталог //localhost является корневым Web-каталогом компьютера, его физическое размещение определяет IIS. По умолчанию это каталог \Inetpub\wwwroot в загрузочном разделе жесткого диска. Но если хранить все проекты в корневом Web-каталоге, как задано по умолчанию, он быстро заполнится хаотической смесью проектов-примеров, кола рабочих приложений и других файлов. Во избежание путаницы лучше размешать примеры, тестируемые приложения и рабочий код в отдельных каталогах. Вложенные каталоги, созданные в корневом Web-каталоге, необходимо опубликовать в Web *до* создания в них новых проектов. Поскольку Visual Studio .NET автоматически создает новые проекты в новых каталогах, вам достаточно создать систему каталогов, упорядочивающую проекты по типам.

Создание виртуальных каталогов для Web-приложений

IIS позволяет создавать новые виртуальные каталоги и управлять размещенными па компьютере Web-сайтами. Основные этапы создания виртуального каталога, предназначенного для использования в Visual Studio .NET, таковы:

- **создание** виртуального каталога. Виртуальные каталоги определяют физическое размещение проектов Web-приложений и служат для их упорядочения;
- добавление к виртуальному каталогу FrontPage Server Extensions для создания **каталогов subweb**. Subweb — это просто виртуальный каталог, содержаший Web-сайт. Добавив к виртуальному каталогу FrontPage Server Extensions, вы позволите Visual Studio .NET создавать и обслуживать Web-приложения в этом каталоге.

Создание виртуального каталога

Чтобы создать в IIS новый виртуальный каталог, выполните следующие действия

1. В IIS щелкните значок Default Web Site правой кнопкой мыши и выберите из контекстного меню команду New\Virtual Directory, как показано на рис. 2-2.
2. IIS запустит мастер Virtual Directory Creation Wizard, который будет направлять ваши действия во время создания виртуального каталога. На заглавной странице мастера щелкните кнопку Next, в результате откроется страница Virtual Directory Alias, показанная на рис. 2-3.

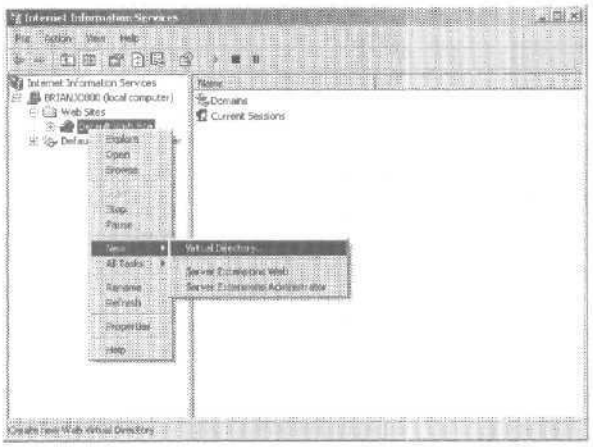


Рис. 2-2. Создание нового виртуального каталога

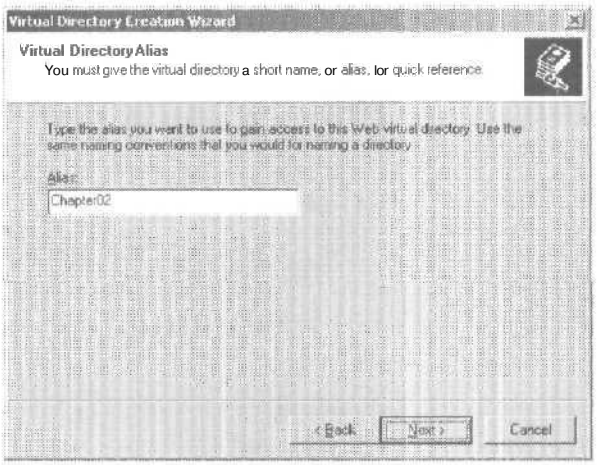


Рис. 2-3. Страница Virtual Directory Alias

3. Введите *псевдоним* (alias) для каталога. Псевдоним понадобится для обращения к ресурсам, расположенным в этом каталоге, а в Visual Studio .NET — для указания размещения проекта. Щелкните кнопку Next, чтобы открыть страницу Web Site Content Directory (рис. 2-4).
4. Введите путь к физическому каталогу, с которым следует связать новый виртуальный каталог. Этот физический каталог станет базовым, в нем будут размещаться папки ваших проектов. Щелкните Next, чтобы открыть страницу Access Permissions, показанную на рис. 2-5.
5. Оставьте без изменений разрешения, заданные по умолчанию (они показаны на рис. 2-5). Щелкните кнопку Next, затем Finish, чтобы создать виртуальный каталог и завершить" работу мастера.

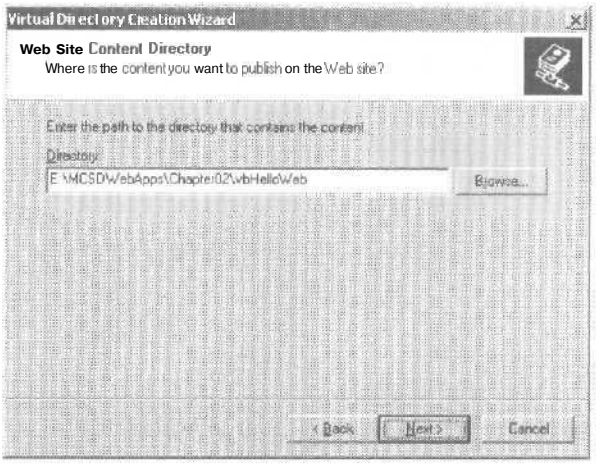


Рис. 2-4. Страница Web Site Content Directory

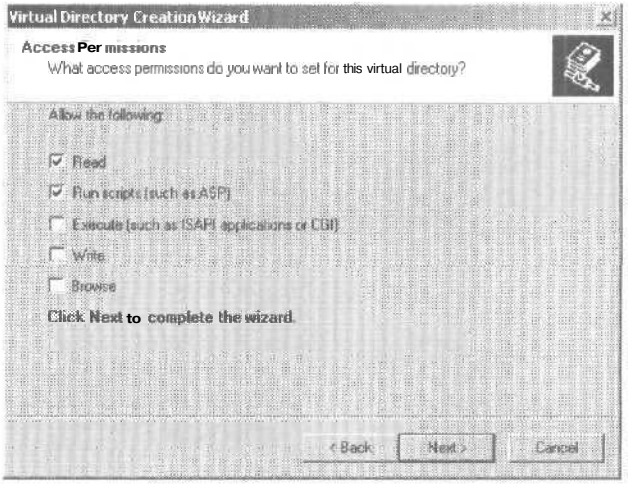


Рис. 2-5. Страница Access Permissions

Создание каталога subweb

Чтобы добавить к виртуальному каталогу FrontPage Server Extensions, выполните следующие действия.

3. В IIS щелкните правой кнопкой значок Defaut Web Site и выберите из контекстного меню пункт New Server Extensions Web.
2. IIS запустит мастер New Subweb Wizard, который поможет вам добавить к виртуальному каталогу FrontPage Server Extensions. Щелкните кнопку Next на заглавной странице мастера, чтобы перейти к странице Subweb Name, показанной на рис. 2-6.
3. В поле Directory name введите имя виртуального каталога (оно соответствует псевдониму, указанному вами при исполнении действия 3 предыдущей процедуры). В поле Title можно ввести описание каталога либо оставить его незаполненным. Щелкните Next — мастер откроет страницу Access Control (рис. 2-7).
4. Примите параметры управления доступом, заданные по умолчанию, щелкнув кнопку Next. Затем щелкните кнопку Finish, чтобы создать вложенный виртуальный каталог.

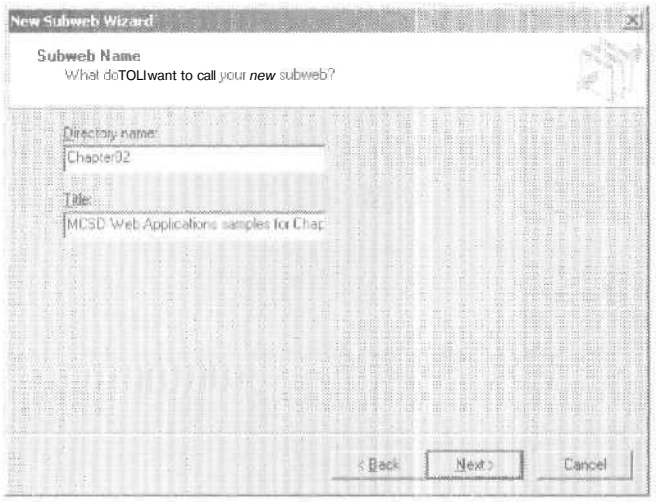


Рис. 2-6. Страница Subweb **Name**

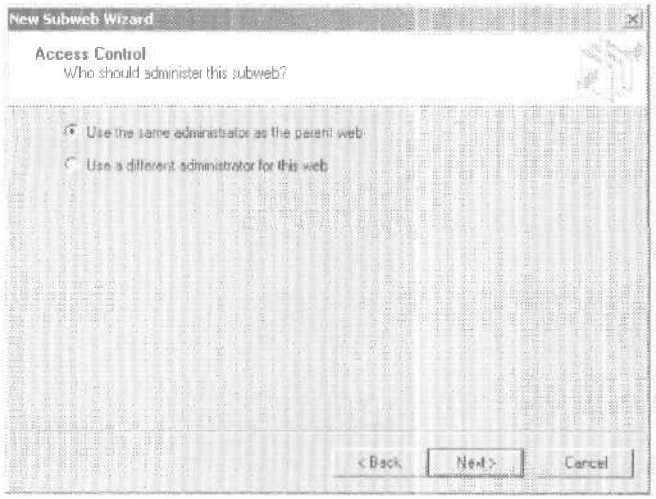


Рис. 2-7. Страница Access Control

Создание нового проекта в виртуальном каталоге

После добавления FrontPage Server Extensions виртуальный каталог готов к созданию в нем новых проектов при помощи Visual Studio .NET. Чтобы создать проект Web-приложения в новом виртуальном каталоге, выберите в Visual Studio .NET команду New Project из меню File и укажите в диалоговом окне New Project имя виртуального каталога, как показано на рис. 2-8.

В поле Location диалогового окна New Project необходимо указать путь к проекту в следующем виде: *http://имя_сервера/имя_виртуального_каталога*. Имя сервера localhost свидетельствует о том, что Web-сервер работает на той же машине, где ведется разработка Web-приложения. *Виртуальный_каталог* — это псевдоним, присвоенный до этого виртуальному каталогу.

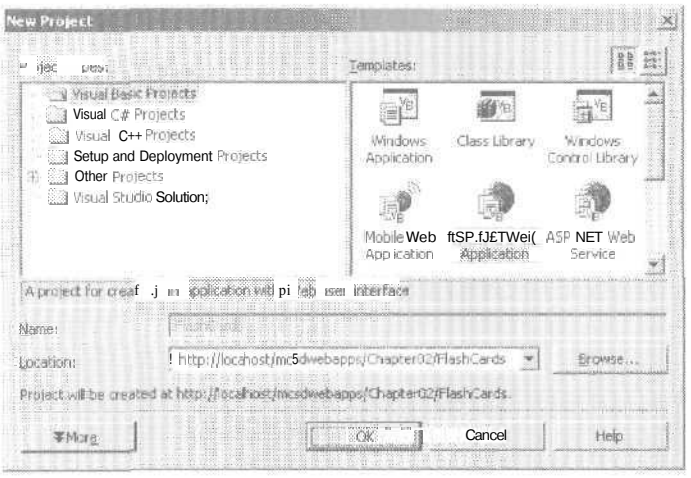


Рис. 2-8. Диалоговое **окно New** Project

Размещение элементов управления и написание кода

После создания Web-приложения Visual Studio .NET отображает в центральном окне новую Web-форму (рис. 2-9). Элементы управления можно перетаскивать с панели Toolbox на поверхность такой формы, как при создании обычной Windows-формы.

Чтобы добавить к Web-форме код для обработки событий, дважды щелкните ее — Visual Studio .NET откроет окно Code. Вставьте нужный код в процедуру обработчика события этого элемента. Например, следующий код покажет в Internet Explorer страницу с кнопкой, щелкнув которую, вы увидите надпись «Hello, Web!»:

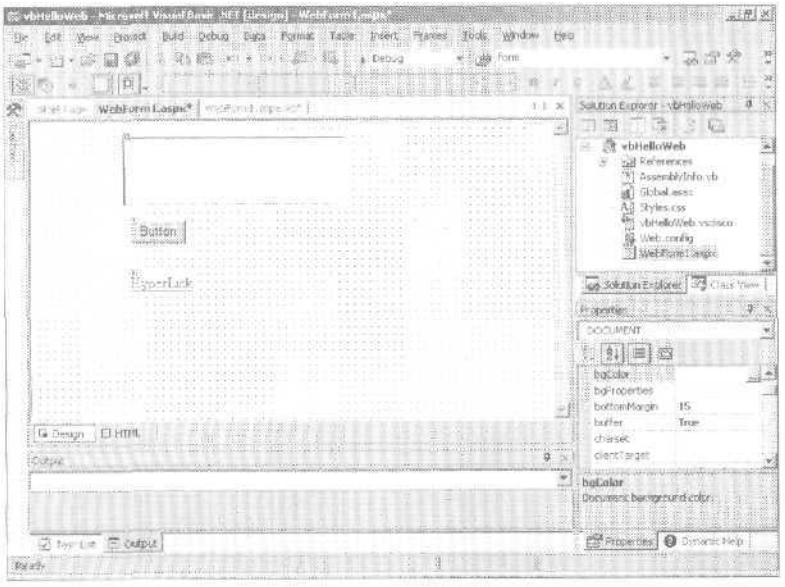


Рис. 2-9. Новая Web-форма

Visual Basic.NET

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
        Response.Write("Hello, Web!<br>")
End Sub
```

Visual C#

```
private void Button1_Click(object sender, System.EventArgs e)
{
    Response.Write("Hello, Web!<br>");
}
```

Запустите приложение, нажав клавишу F5. Visual Studio .NET скомпонует приложение, запустит браузер и откроет в нем созданную страницу. Нажав кнопку Button1, вы заставите браузер отобразить приветствие «Hello, Web!» (рис. 2-10). Закрыв браузер, вы завершите работу Web-приложения.

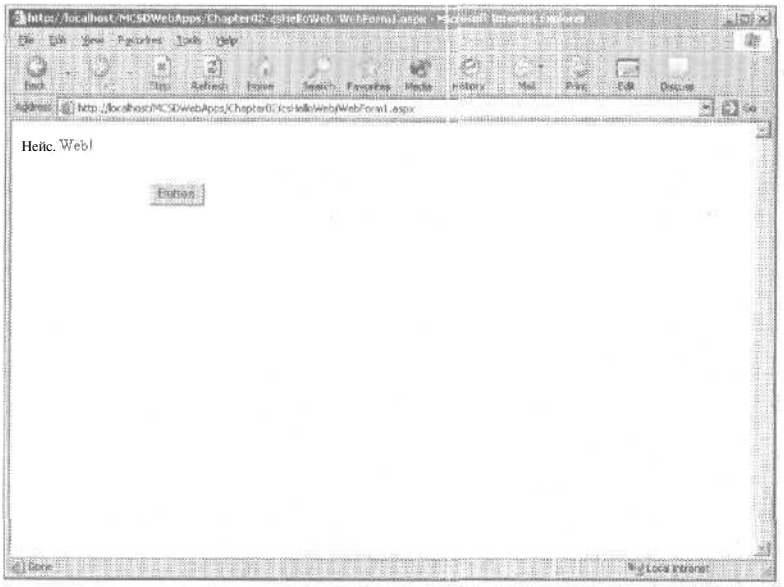


Рис. 2-10. Вид Web-формы в браузере

Этот пример показывает, насколько похожи приложения Web Forms и Windows Forms. Тем не менее по ряду параметров приложения Web Forms существенно отличаются:

* **элементами управления.** На Web-формах нельзя использовать стандартные элементы управления Windows, вместо них применяются специальные элементы управления для Web-форм: *серверные элементы управления* (server controls), *HTML-элементыуправления* (HTML controls), а также *нестандартные элементы управления* (custom controls);
- **пользовательским интерфейсом.** Вид Web-формы определяется отображающим ее браузером. Web-приложение можно открыть в Internet Explorer, Netscape Communicator или в любом другом HTML-совместимом браузере. Разные браузеры (и даже различные версии одного браузера) отличаются наборами поддерживаемых возможностей языка HTML, что влияет как на внешний вид, так и на поведение Web-форм. Но не стоит

особенно беспокоится из-за этого, так как серверные элементы управления корректно справляются с подобными различиями — в этом вы еще убедитесь (см. главу 4);

- **временем жизни.** Windows-формы существуют столько, сколько нужно программе, а затем уничтожаются. У Web-форм похожий жизненный цикл, но это сходство чисто внешнее. В действительности они создаются, передаются браузеру и сразу после этого уничтожаются. Следовательно, все объявленные в Web-форме переменные и объекты после отображения этой формы тут же уничтожаются! Чтобы сделать что-то полезное в такой ситуации, необходимо сохранять информацию в специальных объектах состояния, поддерживаемых ASP.NET. Подробнее об этих объектах рассказано в главе 3;
- **способом исполнения.** Исполняемая часть Web-приложения «живет» на Web-сервере. С этой точки зрения Web-приложения являются разновидностью клиент-серверных приложений, у которых единственным клиентским компонентом является браузер, а все остальное (прикладная логика и пользовательский интерфейс) работает на сервере. Все взаимодействие между клиентом и сервером происходит с использованием HTML. Это означает, что даже сложные Web-приложения практически не угрожают безопасности клиентов, поэтому брандмауэры не становятся помехой для работы Web-приложений. Физическая организация и обработка данных Web-приложений подробно рассматриваются в занятии 3.

Файлы проекта Web Forms

Файл Web-формы — это лишь один из 11 файлов, генерируемых Visual Studio .NET при создании нового проекта Web Forms, назначение каждого из них описано в таблице 2-1. Для каждого нового проекта Web-приложения Visual Studio .NET генерирует полный набор файлов, но в окне Solution Explorer отображаются лишь те, что выделены в таблице 2-1 **полужирным шрифтом.**

Таблица 2-1. Файлы проекта Web Forms

Имя	Содержимое
AssemblyInfo.vb или AssemblyInfo.cs	Все данные, необходимые для идентификации проекта, в том числе версия, название компании, GUID, параметры компилятора и др.
Global.asax	Обработчики глобальных событий Web-приложения, таких как запуск и завершение. В проекте может быть только один файл Global.asax, он размещается в корневом каталоге проекта
Global.asax.vb (Global.asax.cs для проектов на C#)	Код, используемый файлом Global.asax
Styles.css	Определение стилей оформления **HTML-** элементов, генерируемых для проекта
Web.config	Параметры Web-сервера при обслуживании этого проекта; определяют способ сообщения об ошибках, используемый тип аутентификации пользователей и т. п. В проекте может быть только один такой файл, он размещается в корневом каталоге проекта
***Имя_проекта*.disco**	**Описание** Web-сервисов, предоставляемых проектом
WebForm1.aspx	**Описание визуального** представления Web-формы
WebForm1.aspx.vb (WebForml.aspx.cs для проектов на C#)	Код, обрабатывающий события Web-формы. По умолчанию этот файл не отображается в окне Solution Explorer

Таблица 2-1. *(окончание)*

Имя	Содержимое
WebForm1.aspx.resx	XML-ресурсы, используемые Web-формой
Имя_проекта.vbproj (*Имя_проекта*.csproj для проектов на C#)	Список файлов и параметров, используемых во время разработки
Имя_проекта.vbproj.webinfo (*Имя_проекта*.csproj.webinfo для проектов на C#)	Параметры Web, используемые во время разработки проекта

Помимо файлов, перечисленных в таблице 2-1, проект Web-приложения может содержать сколько угодно файлов других типов. В таблице 2-2 описаны основные типы исходных файлов проектов.

Таблица 2-2. Типы файлов проекта Web Forms

Расширение	Элемент проекта	Описание
.aspx	Web-форма	Каждая Web-форма составляет Web-страницу приложения ASP.NET. В приложении может быть одна или несколько Web-форм. С Web-формами связаны файлы, содержащие код. Эти файлы имеют расширение .aspx.vb (либо .aspx.cs, если формы написанны на Visual C#)
.htm	HTML-страница	Web-страницы, не имеющие серверного кода
.vb или .cs	Класс или модуль	Код с определениями объектов приложения, хранимый в виде классов
.ascx	пользовательские Web-элементы управления	Пользовательские элементы управления, созданные в Visual Studio .NET из других Web-форм и серверных элементов управления
.xsd	*DataSet*	Создает схему XML при помощи класса *DataSet*

Файлы Web-приложения

Компонуя проект Web Forms, Visual Studio .NET компилирует весь исходный текст в исполняемый DLL-файл и помещает его в каталог /bin. Часть Web-приложения, отвечающая за его вид, остается в .aspx- и .html-файлах. Файлы, которые получаются в результате компоновки Web-приложения, показаны на рис. 2-11.

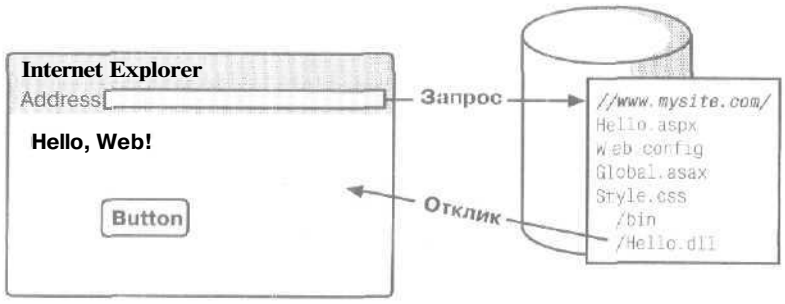


Рис. 2-11. Файлы Web-приложения

Как показано на рис. 2-11, когда браузер запрашивает начальную страницу приложения (http://www.mysite.com/Hello.aspx), IIS запускает исполняемую часть приложения (Hello.dll) и генерирует отклик. В этом случае отклик содержит страницу, определение которой находится в файле Hello.aspx. Если на странице произошло событие (например, пользователь щелкнул кнопку Button), страница возвращается на сервер, где DLL приложения обрабатывает этот событие.

О том, как обрабатываются события и где происходит обработка данных Web-приложения, вы узнаете из следующих занятий.

Занятие 2. Обработка событий

Из этого занятия вы узнаете о событиях, составляющих жизненный цикл Web-приложения, а также о том, чем он отличается от жизненного цикла Windows-приложений. В Web-приложениях события происходят на трех уровнях: на уровне приложения, страницы и серверного элемента управления. Очередность событий и обстоятельства, при которых они произошли, определяют обработку событий кодом приложения.

Изучив материал этого занятия, вы сможете:

- ✓ понять очередность событий жизненного цикла приложения;
- ✓ объяснить взаимодействие событий в приложении;
- ✓ перечислить события основных объектов Web-приложения;
- ✓ назвать три типа событий серверных элементов управления;
- ✓ сохранять данные в Web-приложении при помощи переменных состояния.

Продолжительность занятия — около 35 минут.

События жизненного цикла Web-приложения

Web-приложения живут, пока у них есть хоть один активный сеанс, а Web-формы живут буквально мгновения, Жизнь Web-приложения начинается, когда браузер запрашивает его начальную страницу (рис. 2-12). Тут в игру вступает Web-сервер, который запускает исполняемую часть приложения (DLL), формирующую отклик. Исполняемая часть создает экземпляр запрошенной Web-формы, генерирует HTML-текст отклика и посылает его браузеру, после чего DLL приложения уничтожает созданный ей экземпляр Web-формы.

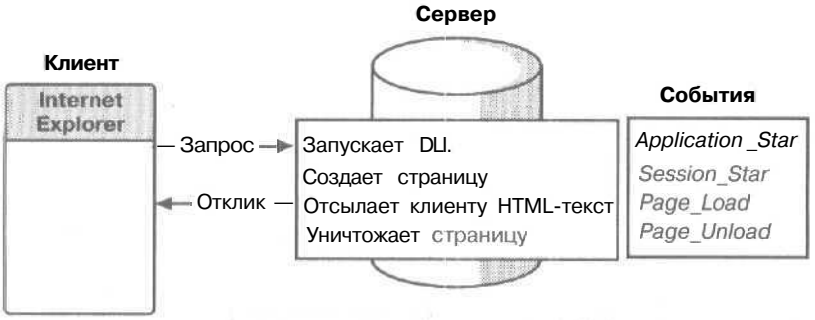


Рис. 2-12. Жизнь началась!

После получения браузером HTML, сгенерированного приложением, пользователь может заполнять текстовые поля формы, устанавливать нужные переключатели и выполнять другие действия, пока он не спровоцирует событие, вызывающее отправку формы на сервер (например, щелкнув кнопку). Это событие заставит браузер отослать данные страницы (ее состояние отображения) обратно на сервер для обработки события. Получив состояние отображения, сервер создает новый экземпляр Web-формы, заполняет его данными состояния отображения и обрабатывает все события (см. рис. 2-13). Завершив обработку, сервер возвращает браузеру полученный в результате HTML и уничтожает созданный им экземпляр Web-формы.

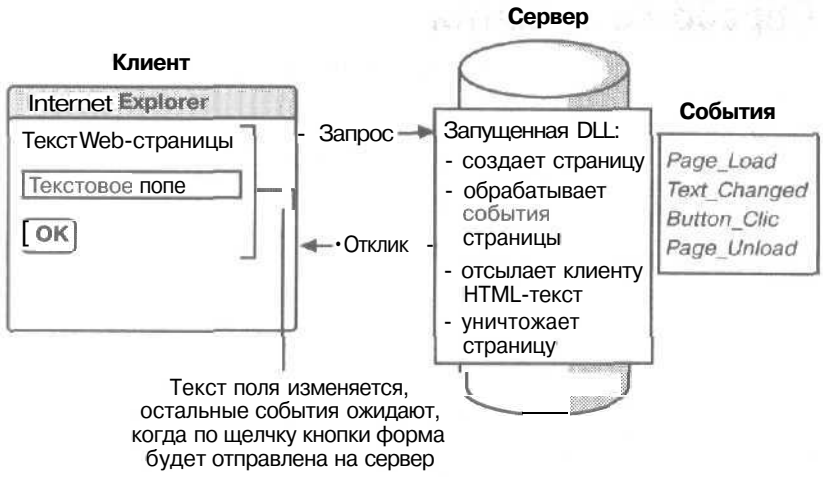


Рис. 2-13. Жизнь продолжается

Закончив работу, пользователь закрывает браузер либо переходит на другой Web-сайт, в результате сеанс этого пользователя завершается (рис. 2-14). Если с приложением больше никто не работает (то есть сеансов больше нет), оно завершается. Однако это не всегда происходит сразу. В отличие от OLE, где управление памятью базируется на подсчете ссылок, **ASP.NET** управляет памятью при помощи сбора мусора. При использовании этого механизма сервер периодически проверяет ссылки объектов. Обнаружив объект, который больше не используется, сервер уничтожает его, освобождая занятую им память. Как следствие, нельзя сказать наверняка, когда именно сработает событие *Aplication_End*.

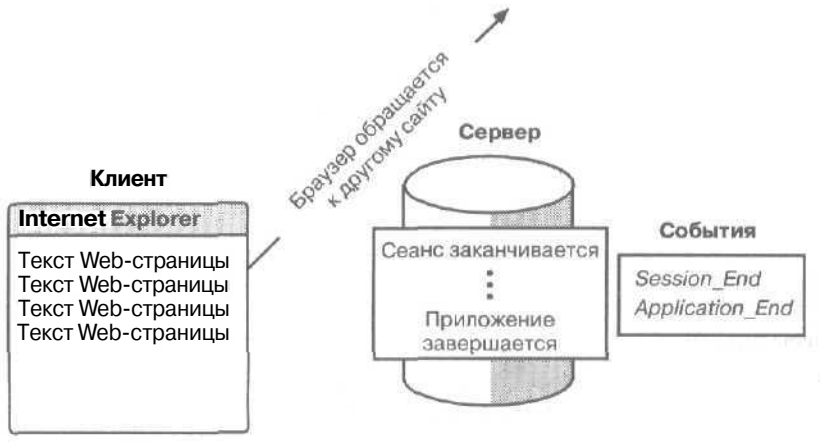


Рис. 2-14. Это конец...

Сохранение данных Web-формы

Поскольку Web-формы живут очень недолго, ASP.NET принимает специальные меры для сохранения данных, введенных в элементы управления Web-формы (рис. 2-15). С каждым запросом на сервер отправляются все данные, которые были ведены в элементы управления, где они записываются в элементы управления во время обработки события *Page_Init*. Теперь эти данные будут доступны обработчику события *Page_Load*.

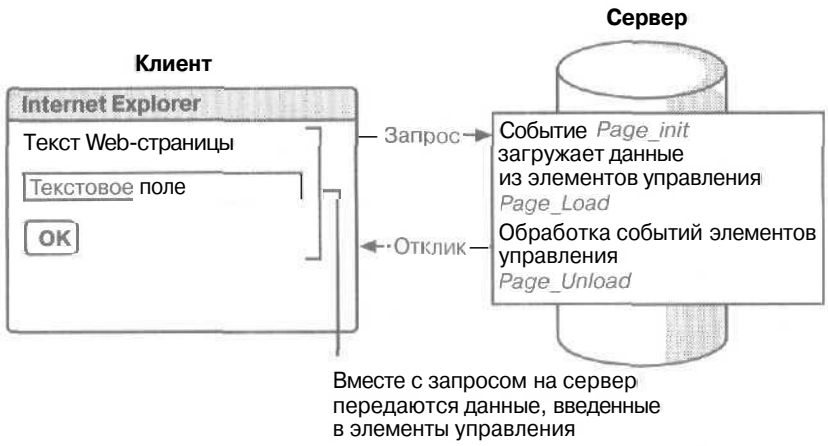


Рис. 2-15. ASP.NET сохраняет данные Web-формы

Данные, которые ASP.NET сохраняет между передачей запросов, называются *состоянием отображения* (view state) Web-формы. Состояние отображения Web-формы доступно только внутри этой Web-формы. Чтобы сделать введенные в Web-форму данные доступными другим Web-формам приложения, необходимо сохранить их в переменных состояниях (в объектах *Application* либо *Session*). Эти объекты отличаются областью действия:

- переменные состояния на основе объекта *Application* доступны всем пользователям приложения, поэтому их можно рассматривать как глобальные переменные, доступные нескольким пользователям. Эти переменные разрешается читать и записывать из любых сеансов;
- переменные состояния на основе объекта *Session* доступны в пределах одного сеанса (только одному пользователю). Переменные Session напоминают глобальные переменные стандартных Windows-приложений. Эти переменные состояния доступны только в текущем сеансе.

Переменные *Application* и *Session* объявляются иначе, чем обычные. Переменные состояния создаются кодом «на лету». Следующий код подсчитывает, сколько раз щелкнули кнопку, и записывает результат в переменную состояния *Session*.

Visual Basic .NET

```
Private Sub Button1_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button1.Click
    'Сохранить число щелчков в переменной состояния Session,
    Session("Clicks") = Session("Clicks") + 1
    ' Отобразить число щелчков.
    Response.Write("Number of clicks: " & Session("Clicks"))
End Sub
```

Visual C#

```
// Код из файла Global.asax.cs
protected void Session_Start(Object sender, EventArgs e)
{
    // Инициализировать переменную состояния Clicks Session.
    Session["Clicks"] = 0;
```

```
}

// Этот код взят из файла StateNEvents.asax.cs
private void Button1_Click(object sender, System.EventArgs e)
{
    // Увеличить число щелчков.
    Session["Clicks"] = (int)Session["Clicks"] + 1;
    // Показать число щелчков,
    Response.Write("Number of
    clicks: " + Session["Clicks"] + "<br>");
}
```

В переменных состояния можно хранить данные любого типа, от обычных целочисленных значений до объектов. Поскольку переменные состояния относятся к глобальным, необходимо выработать стратегию их использования в приложении. Управление глобальными переменными в программе подробно рассматривается в главе 3.

Внимание! Как правило, в программах на Visual C# переменные состояния *Application* следует инициализировать перед использованием. Например, перед преобразованием типов (int)Session[«Clicks»] переменную состояния *Clicks* необходимо инициализировать, в противном случае во время выполнения вы получите сообщение об ошибке: «Value null was found where an instance of an object was required».

События Application и Session

Код, обрабатывающий события объектов *Application* и *Session,* можно поместить в файл Global.asax. События *Application* используются для инициализации объектов и данных, которые должны быть доступными всем текущим сеансам Web-приложения. События *Session* служат для инициализации данных, которые нужно хранить в пределах одного сеанса, не делая их доступными другим сеансам. В таблице 2-3 перечислены обработчики различных событий *Application* с указанием обстоятельств, при которых они вызываются.

Таблица 2-3. Обработчики событий *Application*

Имя	Когда **вызывается**
Application_Start	Первый пользователь открыл начальную страницу Web-приложения
Application_End	С приложением больше никто не работает
Application_Begin Request	В начале каждого запроса к серверу. Запрос происходит каждый раз, когда браузер открывает какую-либо страницу приложения
Application_EndRequest	В конце каждого запроса к серверу
Session_Start	Новый пользователь открыл начальную страницу приложения
Session_End	Пользователь покинул приложение, закрыв браузер, либо наступил тайм-аут по бездействию

В приложениях Web Forms *сеанс* (session) представляет уникальный экземпляр браузера. Если пользователь запустит у себя на компьютере несколько экземпляров браузера и откроет в каждом из них страницы Web-приложения, для каждого экземпляра браузера будет создан отдельный сеанс.

Чтобы увидеть события *Application* и *Session* в действии, добавьте в файл Global.asax проекта Web Forms следующий код:

Visual Basic .NET

```
Sub Application_Start(ByVal Sender As Object, ByVal E As EventArgs)
    ' Зарегистрировать запуск приложения.
    Application("AppCount") = Application("AppCount") + 1
End Sub

Sub Session_Start(ByVal Sender As Object, ByVal E As EventArgs)
    ' Подсчитать сеансы.
    Application("SessCount") = Application("SessCount") + 1
    ' Показать число экземпляров приложения.
    Response.Write("Number of applications: " & _
        Application("AppCount") & "<br>")
    ' Показать число сеансов.
    Response.Write("Number of sessions: " & _
        Application("SessCount") & "<br>")
End Sub

Sub Session_End(ByVal Sender As Object, ByVal E As EventArgs)
    ' Уменьшить число сеансов.
    Application("SessCount") = Application("SessCount") - 1
End Sub
```

Visual C#

```
protected void Application_Start(Object sender, EventArgs e)
{
    // Объявить переменные состояния Application,
    Application["AppCount"] = 0;
    Application["SessCount"] = 0;
    // Зарегистрировать запуск приложения.
    Application["AppCount"] = (int)Application["AppCount"] + 1;
}

protected void Session_Start(Object sender, EventArgs e)
{
    // Подсчитать сеансы.
    Application["SessCount"] = (int)Application["SessCount"] + 1;
    // Показать число экземпляров приложения.
    Response.Write("Number of applications: " +
      Application["AppCount"] + "<br>");
    // Показать число сеансов.
    Response.Write("Number of sessions: " +
      Application["SessCount"] + "<br>");
}
```

```
protected void Session_End(Object sender, EventArgs e)
{
    // Уменьшить число сеансов.
    Application["SessCount"] = (int)Application["SessCount"] - 1;
}
```

Чтобы увидеть, как работают эти события, исполните предыдущий код, затем запустите новый экземпляр браузера и откройте в нем свое Web-приложение, набрав его адрес. С каждым новым экземпляром браузера число сеансов увеличивается, но число приложений остается равным 1.

Обратите внимание, что внутренние объекты, такие, как *Session* и *Response,* не доступны на момент события *Application_Start.* Чтобы задействовать эти объекты, придется дождаться событий, которые происходят при их создании.

События Web-формы

События Web-формы используются для обработки и сохранения данных, введенных на Web-странице, связывания данных, а также для обработки исключений Web-страницы. В таблице 2-4 эти события перечислены в порядке их генерации. Чаще всего в коде приходится иметь дело с обработчиками событий *Page_Load* и *Page_Error.*

Таблица 2-4. **События Web-формы**

Имя обработчика	**Когда** происходит
Page_Init	При загрузке и инициализации серверных элементов управления с использованием состояния отображения Web-формы. Это первый этап жизненного цикла Web-формы
Page_Load	При загрузке серверных элементов управления в объект *Page.* В этот момент доступны данные состояния отображения, поэтому код для изменения параметров элементов управления или отображения текста на странице следует поместить в обработчик именно этого события
Page_PreRender	Перед визуализацией объекта *Page*
Page_Unload	При выгрузке страницы из памяти
Page_Error	При возникновении необработанного исключения
Page_AbortTransaction	При отмене транзакции
Page_CommitTransaction	При фиксации транзакции
Page_DataBinding	При привязке к источнику данных серверного элемента управления, расположенного на странице
Page_Disposed	При освобождении объекта *Page.* Это последнее событие жизненного цикла *Page*

Чтобы выполнить инициализацию при первом посещении клиентом Web-формы, можно использовать событие *Page_Load* вместе со свойством *IsPostback.* В результате получится «событие», которое напоминает *Session_Start,* но происходит на уровне страницы, а не на уровне приложения. Следующий код инициализирует объект и при первом просмотре страницы сохраняет его в переменной состояния *Session:*

Visual Basic .NET

```
' Объявить новый объект.
Dim FlashCard As New FlashCardClass()

Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' ...Если это первый просмотр страницы. . ,
    If Not (Page.IsPostBack) Then
        ' перемешать карточки.
        FlashCard.Shuffle()
        ' Сохранить объект в переменной Session.
        Session("FlashCard") = FlashCard
    End If
    ' Получить значение переменной Session.
    FlashCard = Session("FlashCard")
    RefreshDisplay()
End Sub
```

Visual C#

```
// Объявить новый объект.
FlashCardClass FlashCard = new FlashCardClass();

private void Page_Load(object sender, System.EventArgs e)
{
    if(!IsPostBack)
    {
            // Перемешать карточки.
        FlashCard.Shuffle();
        // Сохранить объект в переменной Session ,
        Session["FlashCard"] = FlashCard;
    }
   // Получить значение переменной FlashCard из объекта Session .
    FlashCard = (FlashCardClass)Session["FlashCard"];
    RefreshDisplay();
}
```

Прочие страничные события позволяют настраивать вид страницы и обрабатывать события, связанные с данными. Привязка к данным, обработка транзакций и визуализация страницы подробно описаны в следующих главах.

События серверных элементов управления

События, срабатывающие в ответ на определенные действия пользователя, есть у всех серверных элементов управления (таких, как Button, TextBox и DropDownList), но не все эти события одинаковы. Существует три типа событий серверных элементов управления:

1. **события, инициирующие возврат** формы (post-back events) — вызывают возврат Web-страницы на сервер для немедленной обработки событий. Заметно снижают быстродействие из-за дополнительного обмена данными с сервером;

2. кэшируемые события (cashed events) — сохраняются в состоянии отображения страницы. Обрабатываются, когда происходит событие, инициировавшее возврат формы;
3. события проверки (validation events) — обрабатываются прямо на странице, не требуют возврата формы или кэширования.

На рис. 2-16 показана очередность событий серверных элементов управления Web-формы. События проверяющих элементов управления обрабатываются до отправки страницы на сервер. События *Page_Init* и *Page_Load* обрабатываются при возврате страницы, затем выполняется обработка кэшированных событий и в последнюю очередь обрабатывается событие, инициировавшее возврат формы.

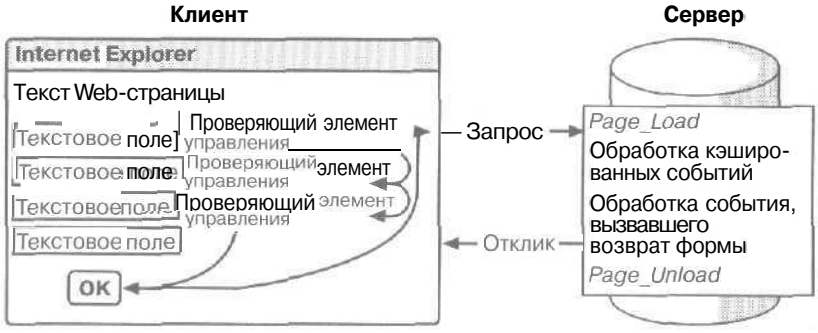


Рис. 2-16. Последовательность генерации событий Web-формой

Элементы управления Button, Link Button и Image Button генерируют события, вызывающие возврат формы. В отличие от них элементы управления TextBox, DropDownList, ListBox, RadioButton и CheckBox генерируют кэшируемые события; но их повеление можно изменить, присвоив свойству *AutoPostBack* значение *True*.

Чтобы пронаблюдать за взаимодействием событий различных типов, создайте Web-форму и поместите на нее серверные элементы управления TextBox, RequiredFieldValidator и Button. Занесите в свойство *ControlToValidate* проверяющего элемента управления (RequiredFieldValidator) значение TextBox1, затем добавьте в обработчики событий текстового поля и кнопки следующий код:

Visual Basic .NET

```
Private Sub Button1_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button1.Click
    Response.Write("ButtonClicked!<br>")
End Sub

Private Sub TextBox1_TextChanged(ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles TextBox1.TextChanged
    Response.Write("Text has changed!<br>")
End Sub
```

Visual C#

```
private void Button1_Click(object sender, System.EventArgs e)
{
    Response.Write("Button Clicked!<br>");
}
```

```
private void TextBox1_TextChanged(object sender, System.EventArgs e)
{
    Response.Write("Text has changed!<br>");
}
```

Если оставить поле пустым и щелкнуть OK, сработает только RequiredFieldValidator, отобразив сообщение — больше никакие события не обрабатываются и страница на сервер не возвращается. Если ввести в поле какой-нибудь текст и щелкнуть OK, страница будет отправлена на сервер, при этом событие *TextChanged* обработывается раньше события *Click*.

Занятие 3. **Обработка данных**

Из предыдущих занятий вы узнали, что Web-приложения работают на сервере. Однако в отличие от Windows-приложений, которые запускаются при вызове исполняемого файла пользователем и завершаются, когда пользователь закрывает приложение, Web-приложения запускаются при первом запросе браузером Web-страницы и завершаются после исполнения всех запросов.

В связи с этими особенностями для управления Web-приложениями используется IIS. Из этого занятия вы узнаете, как IIS находит Web-страницы, принадлежащие некоторому приложению, как сервер исполняет DLL приложений и как он определяет момент, когда приложение следует завершить,

Изучив материал этого занятия, вы сможете:

- ✓ установить границы приложения с помощью каталогов;
- ✓ назвать три варианта исполнения Web-приложения в IIS;
- ✓ объяснить, как параметры IIS влияют на производительность и устойчивость приложений;
- ✓ задавать значение тайм-аута, косвенно определяющее момент завершения приложения. Кроме того, вы узнаете о роли структуры каталогов и параметров исполнения при разработке Web-приложений.

Продолжительность занятия — около 15 минут.

Установка границ приложения

IIS считает Web-приложением любой исполняемый им файл из каталогов Web-сайта. Границы Web-приложения определяются структурой его каталогов: граница начинается в каталоге, содержащем начальную страницу приложения, и заканчивается в последнем вложенном каталоге. На рис. 2-17 показаны отдельные Web-приложения.

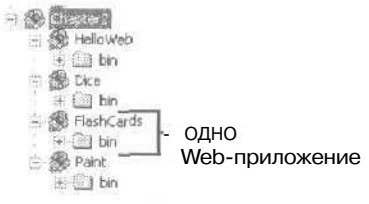


Рис. 2-17. Границы Web-приложения в IIS

Создавая проект Web-приложения, Visual Studio .NET генерирует каталог проекта и делает первую Web-форму проекта (Webform1.aspx) его *начальной страницей* (start page). Начальной считается страница, выбранная Visual Studio .NET в качестве первой отображаемой страницы приложения. Все вложенные каталоги, такие, как /bin, становятся частями данного приложения,

Если в ходе разработки изменилась структура приложения или во время отладки понадобилось оперативно загрузить текущую страницу, может потребоваться изменить начальную страницу приложения.

Чтобы сделать HTML-страницу или Web-форму начальной страницей приложения, выполните действия, описанные далее.

1. В окне Project выберите Web-форму или HTML-страницу, которую нужно сделать начальной страницей приложения.
2. В меню Project выберите команду Web Project\Set As Start Page. Теперь Visual Studio .NET будет использовать эту страницу как первую отображаемую страницу приложения.

У каждого созданного вами проекта есть своя начальная страница, которая делает его отдельным приложением. IIS позволят объединить несколько проектов в одно Web-приложение.

Чтобы передать подчиненное Web-приложение приложению, занимающему более высокий уровень иерархии, выполните действия, описанные далее.

1. В IIS выберите подчиненное Web-приложение, которое нужно включить в состав другого приложения,
2. В меню Action выберите команду Properties. В результате IIS откроет окно свойств приложения, показанное на рис. 2-18.

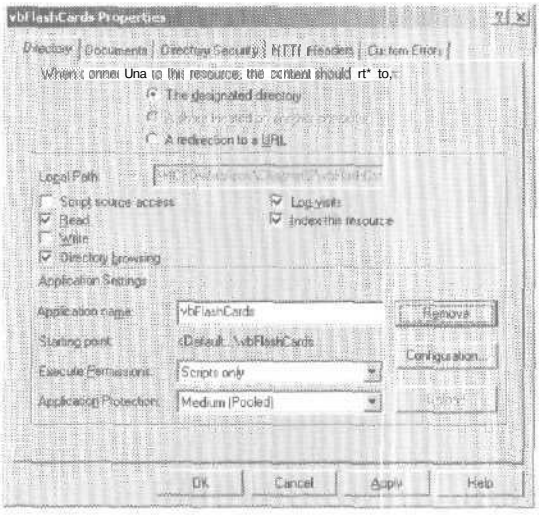


Рис. 2-18. Диалоговое окно Properties

3. Щелкните кнопку Remove в этом окне. При этом начальная точка приложения будет удалена, а само приложение включено в состав приложения, расположенного на более высоком уровне иерархии каталогов. Щелкните OK, чтобы принять изменения и вернуться в IIS.

При изменении структуры приложения может потребоваться разделить большое приложение, состоящее из нескольких каталогов, на несколько меньших.

Чтобы разделить Web-приложение, состоящее из нескольких каталогов, на меньшие приложения, выполните действия, описанные далее.

1. В IIS выберите подкаталог, который нужно сделать отдельным Web-приложением.
2. В меню Action выберите Properties, чтобы вызвать окно свойств приложения.
3. Щелкните Create, чтобы сделать выбранный каталог начальной точкой приложения. Затем щелкните OK, чтобы принять изменения и вернуться в IIS.

Изменение границ приложения может влиять на переменные состояния *Application*. Важно понимать, что переменные состояния *Application* доступны всем DLL, расположенным внутри границ приложения. Если не учитывать это при разработке DLL, результаты взаимодействия между ними могут оказаться непредсказуемыми.

Управление процессами

Важность границ приложения, устанавливаемых в IIS, еще и в том, что они определяют способ исполнения приложения на сервере. Web-приложение в IIS исполняется по одному из трех вариантов.

1. **В одном процессе с IIS** (в процессе **Inetinfo.exe).** Этот вариант обеспечивает большую производительность, поскольку все вызовы происходят *в пределах одного процесса* (In-process), но делает сервер и другие приложения уязвимыми. Если в приложении возникнут сбои, оно может испортить данные как сервера (Inetinfo.exe), так и других Web-приложений, выполняющихся в этом процессе.
2. **В одном процессе е другими Web-приложениями (в процессе DLLHost.exe).** Этот вариант, заданный по умолчанию, считается компромиссным между безопасностью и производительностью. Сбои в приложении могут затронуть лишь его соседей по процессу, но не Inetinfo.exe.
3. **Изолированно в собственном экземпляре DLLHost.exe.** Изолированные приложения защищены от влияния ошибок в других приложениях, а сбои в изолированном приложении не влияют на другие приложения. Однако вызовы, адресованные другим приложениям, должны пересекать границы процессов, что снижает производительность.

Чтобы выбрать способ исполнения приложения в IIS, выполните действия, описанные далее.

1. В IIS выберите каталог с начальной точкой приложения.
2. В меню Action выберите Properties. В результате IIS откроет окно свойств приложения (см. рис. 2-18).
3. В раскрывающемся списке Application protection выберите подходящий вариант. Щелкните OK, чтобы принять изменения и вернуться в IIS.

Чтобы не снижать надежности сервера, старайтесь выбирать варианты Pooled или Isolated. Вариант In-process — это наследие предыдущих версий IIS, поддерживавших исполнение приложений только в одном процессе с сервером.

Для новых приложений по умолчанию выбирается вариант Pooled. Этот вариант оптимален для большинства приложений, поскольку позволяет использовать для доступа к другим компонентам внутрипроцессные вызовы, не требующие маршаллинга через границы процесса.

Вариант Isolated обеспечивает наивысшую защиту от сбойных приложений, но снижает производительность из-за необходимости маршаллинга всех вызовов, адресованных другим приложениям. К тому же существует практический предел числа изолированных процессов — не более 10 на один сервер, поэтому исполнение в изолированном режиме следует использовать только для защиты приложений, критичных для бизнеса.

Определение момента завершения приложения

Когда пользователь в первый раз запрашивает страницу Web-приложения, IIS запускает это приложение (если оно еще не запушено) и создает для этого пользователя сеанс. Все последующие запросы этого пользователя являются частью его сеанса. Сеансы важны по двум причинам:

- они позволяют ASP.NET сохранять специфичные для пользователя данные в переменных состояния *Session*. Переменные состояния *Session* и *Application* обсуждаются в занятии 2;
- они определяют момент завершения приложения. Как только заканчивается последний сеанс, MS завершает приложение.

Продолжительность сеанса определяется значением тайм-аута, заданным в файле Web.config. По умолчанию оно составляет 20 минут, то есть по прошествии 20 минут после последнего запроса пользователя его сеанс завершается, а если нет других сеансов, завершается и приложение.

Помните: IIS считает Web-приложением любой исполняемый им файл из каталогов Web-сайта, поэтому, если в течение 20 минут пользователь обратится к любому .aspx- или .htm-файлу в одном из каталогов приложения, его сеанс продолжится.

Примечание Можно лишь косвенно определить момент завершения приложения, задавая время тайм-аута. Это важный момент, поскольку от него зависит жизненный цикл приложения на сервере.

Меньшее значение тайм-аута сеанса позволяет быстрее освобождать ресурсы сервера, поскольку в этом случае быстрее истекает срок действий сеансовых данных. Однако слишком малое значение тайм-аута может заставить сервер завершить сеансы, которые еще используются, что чревато сбоями в работе приложения.

Чтобы изменить значение тайм-аута сеанса, выполните действия, описанные далее.

1. Откройте проект в Visual Studio .NET.
2. Откройте файл Web.config, дважды щелкнув его значок в окне Solution Explorer.
3. Измените значение тайм-аута (оно выделено полужирным шрифтом), отредактировав следующую строку:

```
<sessionState mode="InProc" stateConnectionString="tcpip=127.0.0.1:42424"
sqlConnectionString="data source=127.0.0.1;user id=sa;password="
cookieless="false" timeout="20" />
```

4. Сохраните файл Web.config.

Резюме

- Web-формы используются для создания интерфейса Web-приложений, отображаемого на пользовательском компьютере посредством Интернет-браузера.
- Ресурсы и код, обрабатывающий события, а также выполняющий полезные вычисления, хранятся и работают на Web-сервере, на котором размешается приложение.
- В силу распределенной структуры Web-приложений, они отличаются от приложений для Windows по четырем основным признакам:
  - вместо элементовуправления WindowsWeb-приложения используют серверные элементы управления и HTML-элементы;
  - Web-приложения не имеют собственных окон и отображаются в окне браузера;
  - срок жизни Web-форм очень мал, поэтому во время обработки страничных событий и событий элементов управления необходимо сохранять нужные данные в переменных состояния;
  - обработка данныхWeb-приложений происходитна сервере, клиент и сервер обмениваются данными в цикле запросов и откликов.
- Web-приложения управляются событиями, которые происходят на уровнях приложения, страницы и элемента управления.
- Существует три типа событий серверных элементов управления. Эти события происходят и обрабатываются в следующем порядке:
  - события проверяющих элементов управления происходят и обрабатываются до отправки страницы на сервер;
  - кэшируемые события накапливаются во время отображения и обрабатываются, как только страница отправит запрос на сервер;
  - события, инициирующие возврат формы, заставляют страницу отправить на сервер запрос, но обработчики этих событий исполняются в последнюю очередь.
- Границы Web-приложения определяются структурой его каталогов,
- Границы приложения влияют на область видимости данных, хранимых в переменных состояния *Application*, и позволяют указывать процесс, в котором сервер должен выполнять приложение.
- IIS позволяет создавать корневые каталоги приложений, устанавливать границы приложения, а также выбирать процесс, в котором он будет исполнять приложение,

Практикум 2. Создание простого Web-приложения

В этом практикуме вы создадите простое Web-приложение FlashCards с одной формой. Оно позволяет решать простые арифметические задачи и проверять введенный ответ – совсем как карточки, которыми пользуются ученики на уроках математики в начальной школе.

Чтобы создать приложение FlashCards, следует поместить на новую Web-форму серверные элементы управления и добавить код, управляющий ими, в обработчик события *Page_Load*. Готовая Web-форма может выглядеть так, как показано на рис. 2-19.

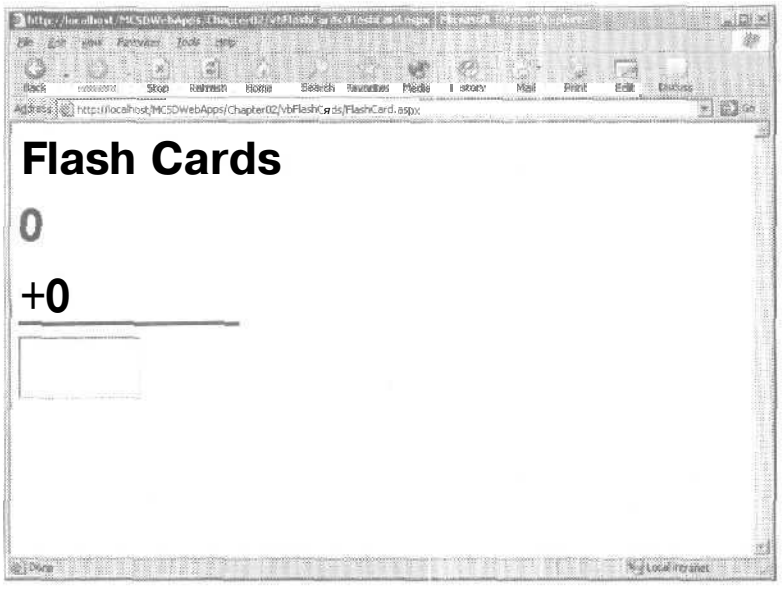


Рис. 2-19. Готовая Web-форма приложения FlashCard

Продолжительность практикума — около 20 минут.

Упражнение 1. Создание пользовательского интерфейса



Вы должны создать проект FlashCard и сконструировать пользовательский интерфейс приложения, поместив на Web-форму FlashCard серверные элементы управления.

▶ Создание нового проекта Web-приложения

1. Откройте новый проект ASP.NET Web application. В диалоговом окне New Project введите имя **FlashCards** и щелкните OK.
2. В окне Properties переименуйте файл WebForm1.aspx в FlashCard.aspx.

▶ Добавление элементов управления к пользовательскому интерфейсу

Поместите на Web-форму серверные элементы управления, показанные на рис. 2-19, и задайте для них свойства, перечисленные в таблице.

Имя элемента управления	Тип	Свойство	Значение
LabelI	Надпись	ID	lblFeedback
		Font	Arial, Bold, XXL
		Text	Flash Cards
Label2	Надпись	[D	lblFirst
		Font	Arial, Bold, XXL
Label3	Надпись	[D	IblSecond
		Font	Arial, Bold, XXL
<HR>	Горизонтальная линия	Size	4
		Color	#000000
Text Box1	Текстовое поле	ID	txtAnswer
		AutoPostBack	True
		Font	Arial, Bold, XXL

Совет Если необходимо создать несколько серверных элементов управления с одинаковыми свойствами, создайте первый элемент, присвойте его свойствам нужные значения, затем размножьте его при помощи операций копирования-вставки.

Упражнение 2. Создание класса FlashCardClass



Сейчас вы создадите класс с логикой и данными приложения FlashCard. *Класс* (class) — это определение объекта, используемого в программе для решения некоторой задачи. Подробнее об объектно-ориентированном программировании рассказано в главе 3,

Мы создадим класс *FlashCardClass,* который случайным образом генерирует арифметические задачи, отображаемые на Web-форме, это и будет логикой нашего приложения. В этом приложении, как и в большинстве Web-приложений, логика (класс *FlashCardClass*) и интерфейс (Web-форма) приложения разделены.

▶ Как создать класс

1. В меню Project выберите команду Add Class. Visual Studio .NET откроет диалоговое окно Add New Item. Назовите класс FlashCardClass и щелкните OK.
2. В окне Code введите следующий код:

Visual Basic .NET

```
Public Class FlashCardClass

    Dim mintFirstNumber, mintSecondNumber As Integer
    Dim mstrOp As String = "+"
    Dim mrndNumber As Random

    Public Sub New()
        ' Инициализировать объект генератора случайных чисел.
        mrndNumber = New Random()
    Enc Sub
```

```
    Public Sub Shuffle(Optional ByVal Min As Integer = 1, _
            Optional ByVal Max As Integer = 12)
        ' Генерировать случайные числа.
        mintFirstNumber = mrndNumber.Next(Min, Max)
        mintSecondNumber = mrndNumber.Next(Min, Max)
    End Sub

    Public Readonly Property FirstNumber()
        Get
            FirstNumber - mintFirstNurnber
        End Get
    End Property

    Public Readonly Property SecondNumber()
        Get
            SecondNumber = mintSecondNumber
        End Get
    End Property

    Public Property Operation() As String
        Get
            Operation = mstrQp
        End Get
        Set(ByVal Value As String)
            mstrQp = Value
        End Set
    End Property

    ' Рассчитать ответ на основе текущего арифметического действия.
    Public Function Answer() As Integer
        Select Case mstrOp
            Case "+"
                Answer - mintFirstNurnber + mintSecondNumber
            Case "x", "*"
                Answer = mintFirstNurnber * mintSecondNumber
            Case "-"
                Answer = mintFirstNumber - mintSecondNumber
        End Select
    End Function

End Class
```

Visual C#

```
public class FlashCardClass
{
    int mintFirstNumber, mintSecondNumber;
    string mstrOp - "+";
```

```
Random mrndNumber;

public FlashCardClass()
{
      // Инициализировать объект генератора случайных чисел.
mrndNumber = new Random();
}

public void Shuffle(int Mm, int Max)
{
    // Генерировать случайные числа.
    mintFirstNumber = mrndNumber.Next(Min, Max);
    mintSecondNumber - mrndNumber.Next(Min, Max);
}

// Вызов метода Shuffle без параметров, по умолчанию Min - 0, Max - 12.
public void Shuffle()
{
    // Генерировать случайные числа.
    mintFirstNumber = mrndNumber.Next(0, 12):
    mintSecondNumber = mrndNumber.Next(0, 12);
}

public int FirstNumber
{
    get
    {
        return mintFirstNumber;
    }
}

public int SecondNumber
{
    get
    {
        return mintSecondNumber;
    }
}

public string Operation
{
    get
    {
        return mstrOp;
    }
    set
    {
```

```
            mstrOp = value;
        }
    }

    // Рассчитать ответ на основе текущего арифметического действия,
public int Answer()
    {
        switch(mstrOp)
        {
            case "+":
                return mintFirstNumber + mintSecondNumber;
            case "x":
                return mintFirstNumber * mintSecondNumber;
            case "*":
                return rcintFirstNumber * nintSecondNumber;
            case "-":
                return mintFirstNumber - mintSecondNumber;
            default :
                return 0;
        }
    }

}
```

Упражнение 3. Сохранение объекта FlashCardClass в переменной состояния Session



Сейчас вы добавите в файл Global.asax код, инициализирующий переменную состояния *Session*. Эта переменная предназначена для хранения объекта *FlashCardClass*, используемого Web-формой. Файл Global.asax содержит описание событий, которые происходят при запуске приложения и открытии новою сеанса, Объект *FlashCardClass* записывается в переменную *Session* для сохранения на протяжении сеанса пользователя.

▶ **Создание переменной состояния** Session

1. Дважды щелкните файл Global.asax в окне Project, затем щелкните ссылку «Click Here To Switch To Code View».
2. Чтобы создать обработчик события *Session_Start*, введите в окне Code View следующий код:

Visual Basic .NET

```
Sub Session_Start(ByVai Sender As Object, ByVal E As EventArgs)
        ' Создать новый объект FlashCard
    Dim FlashCard As New FlashCardClass()
        ' Сохранить созданный объект в переменной Session.
    Session("FlashCard") = FlashCard
End Sub
```

Visual C#

```
protected void Session_Start(Object sender. EventArgs e)
{
    // Создать новый объект FlashCard.
        FlashCardClass FlashCard - new FlashCardClass();
    // Сохранить созданный объект в переменной Session.
    Session["FlashCard"] - FlashCard;
};
```

Упражнение 4. Обработка событий Web-формы с использованием класса FlashCardClass



Сейчас вы добавите к Web-форме код, который позволит ей использовать объект *FlashCardClass,* сохраненный в переменной состояния *Session*, чтобы соединить логику приложения (класс *FlashCardClass)* с его интерфейсом (Web-формой FlashCard.aspx).

▶ **Как задействовать объект *FlashCardClass* в обработчиках событий Web-формы**

1. Откройте в окне Code модуль кода Web-формы (FlashCard.aspx), дважды щелкнув ее любую область.
2. Объявите на уровне модуля переменную-объект для сохранения объекта *FlashCardClass* из переменной состояния *Session.*

Visual Basic .NET

```
Dim FlashCard As New FlashCardClass
```

Visual C#

```
FlashCardClass  FlashCard;
```

3. Чтобы присвоить значение переменной состояния *Session* только что объявленной переменной, добавьте в обработчик события *Page_Load* следующий код:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    ' Получить объект FlashCard из переменной состояния Session.
    FlashCard = Session("FlashCard")
    RefreshDisplay()
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Получить значение FlashCard из переменной состояния Session,
    FlashCard = (FlashCardClass)Session["FlashCard"];
    RefreshDisplay();
}
```

4. Дважды щелкните файл FlashCard.aspx в окне проекта, затем дважды щелкните текстовое поле. Добавьте в обработчик события *Text_Changed* следующий код:

Visual **Basic .NET**

```
Private Sub txtAnswer_TextChanged(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles txtAnswer.TextChanged
     If txtAnswer.Text = FlashCard.Answer Then
         lblFeedback.Text = "Correct!"
         ' Получить другой набор чисел.
         FlashCard.Shuffle()
         ' Обновить страницу, чтобы показать новые числа.
         RefreshDisplay()
         ' Сбросить ответ.
         txtAnswer.Text = ""
     Else
         IblFeedback.Text - "Oops! Try again."
     End If
 End Sub
```

Visual C#

```
private void txtAnswer_TextChanged(object sender, System.EventArgs e)
 {
     if(txtAnswer.Text == FlashCard.Answer().ToString())
     {
         lblFeedback.Text - "Correct!";
         // Получить другой набор чисел.
         FlashCard.Shuffle();
         // Обновить страницу, чтобы показать новые числа.
         RefreshDisplay();
         // Сбросить ответ.
         txtAnswer.Text = "";
     }
     else
     {
         IblFeedback.Text - "Cops! Try again.";
     }
 }
```

5. Введите кол вспомогательной процедуры, обновляющей текст надписей:

Visual Basic .NET

```
Private Sub RefreshDisplay()
     lblFirst.Text = FlashCarc.FirstNumber
     lblSecond.Text = FlashCard.Operation & _
       FlashCard  SecondNumber
 End Sub
```

Visual C#

```
private void RefreshDisplay()
 {
    lblFirst.Text = FlashCard.FirstNumber.ToString();
     lblSecond.Text = FlashCard.Operation +
       FlashCard.SecondNumber.ToString();
 }
```

6. **Запустите приложение и проверьте его работу.**

Совет Хранение значений переменных состояния *Session* и *Aplpication* в переменных, объявленных на уровне формы (см. пункты 2 и 3) помогает избежать ошибок. Поскольку в Visual Studio .NET объявлять переменные состояния не обязательно, опечатка в имени переменной состояния просто приведет к созданию новой пустой переменной состояния.

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. Где в Visual Studio .NET хранятся проекты Web-приложений?

2. Перечислите основные отличия Web- и Windows-приложений.

3. Опишите жизненный цикл Web-приложения. Когда создаются Web-формы и как долго они существуют?

4. Как сохранить данные (например, значение простой переменной) в Web-приложении?

5. Что определяет границы Web-приложения?

ГЛАВА 3

Работа с Web-объектами

Занятие 1. Основы пространств имен 76

Занятие 2. Пространства имен в Web-приложениях 101

Занятие 3. Сохранение сведений о состоянии 109

В этой главе

В этой главе рассказано, как создавать и упорядочивать объекты в Microsoft Visual Studio .NET, применять объекты, поддерживаемые ASP.NET, а также сохранять объекты и. данные в промежутках между запросами к Web-форме. Visual Basic .NET, Visual C# и ASP.NET используют общую инфраструктуру, поэтому приемы объектно-ориентированного программирования (ООП), описанные в этой главе, будут применимы ко всем языкам программирования Web Forms.

Прежде всего

Для изучения материалов этой главы вам необходимо:

- знать основы Visual Basic, C# или C++, в том числе понятия переменной, процедуры, *структуры решений* (decision structure) и области видимости. Эти сведения приводятся в разделах «Language Changes in Visual Basic» и «C# Language Tour» справочной системы Visual Studio;
- понимать события, составляющие жизненный цикл Web-приложений (он обсуждается в занятии 2 главы 2).

Занятие 1. Основы пространств имен

Языки Visual Basic .NET и Visual C# поддерживают все концепции ООП, включая абстрактные классы, интерфейсы, а также перегрузку и переопределение членов класса. Эти фундаментальные языковые возможности повсеместно используются в .NET Framework. Если вы научитесь применять их самостоятельно, то поймете, как устроена инфраструктура .NET Framework и сумеете более эффективно использовать ее в своих приложениях.

Изучив материал этого занятия, вы сможете:

- ✓ упорядочивать код с использованием пространств имен;
- ✓ создавать классы и управлять доступом к ним;
- ✓ объявлять абстрактные и базовые классы и порождать от них новые классы;
- ✓ понять, как используются перегрузка, переопределение и замещение методов, унаследованных от базового класса;
- ✓ создавать интерфейсы и объяснять, чем они отличаются от абстрактных классов.

Продолжительность занятия — около 40 минут.

Знакомство с пространством имен

Возможно, вы заметили, что в примере файла Global.asax из предыдущей главы код, сгенерированный Visual Studio, содержит следующие строки:

Visual Basic .NET

```
Imports System
Imports System.Web
```

Visual C#

```
using System
using System.Web;
```

Эти операторы позволяют использовать классы из пространств имен *System* и *System.Web*, не указывая их полное имя. Не будь их, вызывать простые методы из пространства имен *System* пришлось бы так:

Visual Basic .NET

```
System.Array.Sort(strArray)
```

Visual C#

```
System.Array.Sort(strArray);
```

Импортировав в начале кода пространство имен *System*, удается сократить вызов метода *Array:*

Visual Basic .NET

```
Array.Sort(strArray)
```

Visual C#

```
Array.Sort(strArray);
```

Пространства имен позволяют упорядочить код. Они защищают от конфликтов имен, иногда также называемых *коллизиями пространств имен* (namespace collisions), что особенно важно в больших проектах, где велика вероятность случайного появления элементов с одинаковыми именами. Упорядочение кода при помощи пространств имен снижает вероятность подобных конфликтов. Чтобы создать пространство имен, следует заключить класс или модуль в блок *Namespace...End Namespace*.

Можно поместить в пространство имен несколько классов или модулей, назначив для них одинаковые пространства имен.

Операторы, выделенные в следующем примере полужирным шрифтом, создают пространство имен для модуля *Strings*:

Visual Basic .NET

```
' Имя проекта: Wombat.
Namespace Utils
    Public Module Strings
        Public Function Sort(ByVal strText As String, _
          Optional ByVal bAlphaOrder As Boolean = True) As String
            ' Объявить и инициализировать строковый массив.
            Dim strArray As String() = {""}
            ' Преобразовать строку в массив при помощи класса
            ' System.String.strArray - strText.Split(" ")
            ' Выполнить сортировку с использованием класса
            ' System.Array.System.Array.Sort(strArray)
            ' Если массив не упорядочен по алфавиту,
            ' отсортировать его в обратном порядке.
            If Not bAlphaOrder Then
                ' Отсортировать элементы в обратном порядке
                ' с помощью System.Array.
                System.Array.Reverse(strArray)
            End If
            ' Вернуть строку.
            Sort - System.String.Join(" ", strArray)
        End Function
    End Module
End Namespace
```

Visual C#

```
// Имя проекта: Wombat.
namespace Utils
{
    class Strings
    {
        // Этот код принимает строку и возвращает ее в упорядоченном виде
        public static string Sort(string strText , bool bAlphaOrder)
```

```
    {
        // Объявить и инициализировать строковый массив.
        string[] strArray = {""};
        char[] strSep = {' '};
        // Преобразовать строку в массив при помощи класса System.String.
        strArray = strText.Split(strSep);
        // Выполнить сортировку с использованием класса System.Array.
        System.Array.Sort(strArray);
        // Если массив не упорядочен по алфавиту,
        // отсортировать его в обратном порядке.
        if (!bAlphaOrder)
        {
            // Отсортировать элементы в обратном порядке с
            // помощью System.Array.
            System.Array.Reverse(strArray);
        }
        // Вернуть строку.
        return System.String.Join(" ", strArray);
    }

    // Тот же метод, но с одним параметром.
    public static string Sort(string strText)
    {
        return Sort(strText, true);
    }

}
}
```

Внимание! В предыдущем примере помимо *Utils* использованы два пространства имен .NET: *System.Array* и *System.String*. Пространство имен *System* — новый механизм доступа к функциям, которые в API Windows были доступны через оператор *Declare*. Теперь добраться до нужных функций еще легче, поскольку языки Visual Basic .NET, Visual C# и .NET Framework используют общие скалярные типы данных и соглашения о передаче параметров по умолчанию. В отличие от API Windows, пространства имен *System* обеспечивают для функций Windows объектно-ориентированный интерфейс. Если вы привыкли к API Windows, использование пространств имен поначалу потребует некоторых усилий, однако это сполна компенсируется выгодой от использования общеязыковой исполняющей среды и управляемого кода.

Поскольку код, упорядоченный при помощи пространств имен, должен оставаться открытым, пространства имен являются открытыми по определению. Ссылки в тексте программ на код, расположенный в некотором пространстве имен, имеют следующий вид:

ИмяПроекта.ИмяПространстваИмен.ИмяМодуля.ИмяЧлена

Есть два способа вызова кода из некоторого пространства имен в приложениях.

- **При помощи полного имени. Код, выделенный в показанном ниже примере полужирным шрифтом, вызывает функцию *Sort* из пространства имен *Wombat. Utils:***

Visual Basic .NET

```
Private Sub Button1_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles Button1.Click
        ' Выполнить сортировку текста.
        TextBox1.Text = Wombat.Utils.Strings.Sort(TextBox1.Text)
    End Sub
End Class
```

Visual C#

```
private void Button1_Click(object sender, System.EventArgs e)
    {
        TextBox1.Text = Wombat.Utils.Strings.Sort(TextBox1.Text);
    }
```

- **При помощи сокращенного имени.** Добавив в начало модуля Visual Basic .NET оператор *Imports* (или в случае класса Visual C# — оператор *using),* можно ссылаться на члены по сокращенным именам. Этот способ реализован в следующем коде, использующем пространство имен *Utils* из проекта *Wombat* (см. строку, выделенную **полужирным** шрифтом):

Visual Basic .NET

```
' Импортировать пространство имен Utils из проекта Wombat.
Imports Wombat.Utils

Public Class WebForm1
    ' Объявления и инициализирующий код опущены...

    Private Sub Button1_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles Button1.Click
        ' Вызвать вспомогательную функцию
TextBox1.Text = Strings.Sort(TextBox1.Text)
    End Sub
End Class
```

Visual C#

```
// Импортировать пространство имен Utils из проекта Wombat.
using Wombat.Utils;
namespace csNamespaces
{
    public class WebForm1 : System.Web.UI.Page
    {
        // Объявления и инициализирующий код опущены...

        private void Button1_Click(object sender, EventArgs e)
        {
            TextBox1.Text = Strings.Sort(TextBox1.Text);
```

```
        }
    }
}
```

В ссылках пространства имен имена, занимающие разные иерархические уровни, разделяются точкой. Рассмотрим следующее объявление:

Visual Basic .NET

```
' Имя проекта: Wombat
Namespace Utils
    Namespace Types
        Namespace Digits
            Public Enum Numeric
                Zero
                One
                Two
                Three
                Four
                Five
                Six
                Seven
                Eight
                Nine
            End Enum
        End Namespace
    End Namespace
End Namespace
```

Visual C#

```
// Имя проекта: Wombat
namespace Utils
{
    namespace Types
    {
        class Digits
        {
            public enum Numeric
            {
                Zero, One, Two, Three, Four, Five, Six, Seven, Eight, Nine
            }
        }
    }
}
```

Это объявление эквивалентно такому:

Visual Basic .NET

```
' Имя проекта: Wombat
Namespace Utils.Types.Digits
```

```
            Public Enum Numeric
             Zero
             One
             Two
             Three
             Four
             Five
             Six
             Seven
             Eight
             Nine
        End Enum
End Namespace
```

Visual C#

```
// Имя проекта: Wombat
namespace Utils.Types
{
    class Digits
    {
        public enum Numeric
        {
            Zero, One, Two, Three, Four, Five,
            Six, Seven. Eight, Nine
        }
    }
}
```

Для ссылки на пространство имен, объявленное выше, можно использовать полное имя, например, так:

Basic .NET

```
Dim numVar As Wombat.Utils.Types.Digits.Numeric
```

Visual C#

```
Wombat.Utils.Types.Digits.Numeric numVar;
```

Альтернативный способ — добавить оператор *Imports* или *using* и указать сокращенное имя:

Visual Basic .NET

```
Imports Wombat.Utils.Types.Digits
Dim numVar As Numeric
```

Visual C#

```
using Wombat.Utils.Types;
Digits.Numeric numVar;
```

Ссылки проекта и оператор Imports

Чтобы использовать пространство имен, расположенное вне текущего проекта, следует добавить в проект ссылку на компонент, в котором оно находится, и импортировать это пространство имен с помощью оператора *Imports*. В таких случаях оператор *Imports* — это просто механизм, позволяющий применять сокращенные имена для обращения в коде к классам и членам из внешнего пространства имен.

Чтобы добавить в проект ссылку, выполните следующие действия:

1. В меню Project выберите команду Add Reference — Visual Studio .NET откроет диалоговое окно Add Reference, показанное на рис. 3-1.
2. Перейдите на вкладку.NET, COM или Project, выберите из списка компонент, на который нужно добавить ссылку, и щелкните OK.
3. В результате Visual Studio добавит в окно Project ссылку на выбранный компонент.

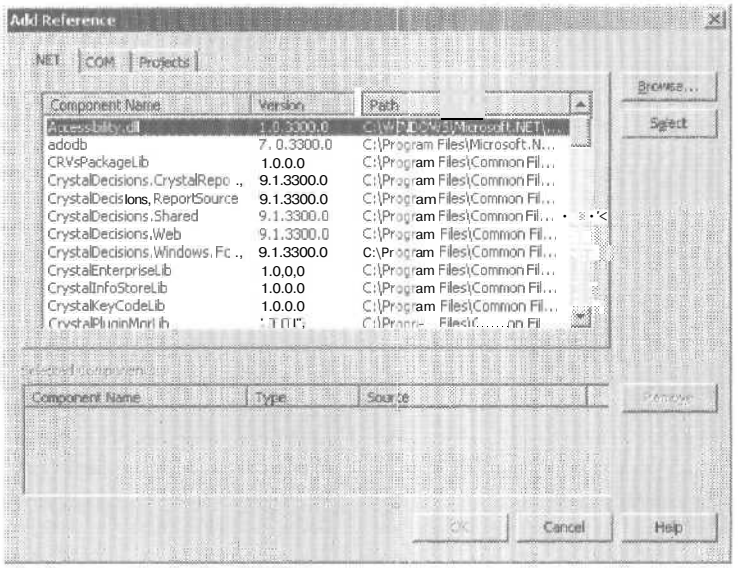


Рис. 3-1. Добавление ссылки в проект

Классы и модули

Проекты Visual Studio хранят код в классах и модулях. Поддержка концепции классов, отличающихся от модулей кода, введена в прежних версиях Visual Basic и продолжается в Visual Basic .NET: классы представляют элементы, выделяющие для себя память (объекты), а модули содержат код, не имеющий постоянных данных. Работая с классом, вы сначала создаете объект этого класса, а затем пользуетесь им, а модули кода вы просто вызываете напрямую.

В отличие от Visual Basic .NET, в Visual C# весь код размещается в классах. Если нужно создать методы или свойства, доступные без предварительного создания объекта, их следует определить как статические. Метод *Sort* класса *Unit,* созданный в предыдущем разделе, может служить примером эквивалента модуля Visual Basic в Visual C#.

В этих языках при работе с модулями и классами используются пять ключевых понятий, новых для Visual Basic .NET (естественно, и для Visual C# тоже, поскольку это абсолютно новый язык). Однако эти понятия наверняка знакомы программистам на C++,

В таблице 3-1 описаны ключевые понятия объектно-ориентированного программирования (ООП) применительно к языкам Visual Basic .NET и Visual C#.

Таблица **3-1. Ключевые понятия ООП**

Понятие	Visual Basic .NET	Visual C#
Определение	Некая сущность определяется как класс или модуль при помощи блоков *Class...End Class* или *Module...End Module*. В предыдущих версиях это неявно определялось типом файла, поэтому в одном файле могло быть не больше одного класса или модуля	Классы определяются при помощи ключевого слова *class*, весь исполняемый код является частью класса
Доступ	Существует пять уровней доступа к классам, модулям и их членам: *Public*, *Protected*, *Friend*, *Protected Friend* и *Private*. Уровень доступа явно задается в определении элемента, а не выводится из свойств файла	Существует пять уровней доступа к классам и их членам: *public*, *protected*, *internal*, *protected internal* и *private*
Наследование	Классы могут наследовать члены от других классов, а затем переопределять, замещать или перегружать унаследованные члены	Классы могут наследовать члены от базовых классов и переопределять либо перегружать унаследованные члены
Конструкторы и деструкторы	У классов есть методы *New* и *Finalise*, вызываемые соответственно при создании и уничтожении объекта данного класса и объектов его производных классов	Классы имеют методы-конструкторы и деструкторы, вызываемые при создании и уничтожении объектов данного класса и объектов его производных классов. Имена конструкторов и деструкторов совпадают с именем их класса, только имя деструктора предваряется тильдой (~)
Абстрактные классы и интерфейсы	Допускается создание интерфейсов, а также абстрактных классов, методов и свойств. Интерфейс содержит список имен членов с указанием их параметров для классов, использующих данный интерфейс. Абстрактные члены являются шаблонами членов, наследуемых производными классами	Допускается определение интерфейсов, а также абстрактных классов, методов и свойств. Интерфейссодержит список имен членов с указанием их параметров для классов, использующих данный интерфейс. Абстрактные члены являются шаблонами членов, наследуемых производными классами

Создание классов, модулей и управление доступом к ним

В Visual Basic .NET для определения классов и модулей применяются блоки *Class...End Class* и *Module...End Module*; в Visual C# классы определяются с использованием ключевого слова *class*. В файле может быть несколько классов и/или модулей. Классы и модули, которым будут доступны члены данного класса или модуля, определяются с помощью модификаторов доступа, перечисленных в таблице 3-2.

Таблица 3-2. Уровни доступа к классам и модулям

Visual Basic	Visual C#	Кому доступен модуль или класс
Public	*public*	Всем членам любых классов и проектов
Friend	*internal*	Всем членам текущего проекта
Protected	*protected*	Всем членам текущего класса и его производным классам. Этот модификатор годится только для определений членов, его нельзя применять в определениях классов или модулей
Protected Friend	*protected internal*	Всем членам текущего класса и его производным классам. Этот модификатор годится только в определениях членов, его нельзя применять для определений классов или модулей
Private	*private*	Только членам текущего класса

Например, следующий класс доступен только в текущем проекте:

Visual Basic .NET

```
Friend Class Translator
    Private mstrText As String

    ' Этот код управляет доступом к переменным,
    ' объявленным на уровне модуля.
    Public Property Text() As String
        Get
            Text = mstrText
        End Get
        3et(ByVal Value As String)
            mstrText = Value
        End Set
    End Property

    ' Транслировать значение свойства Text.
    Public Sub Translate()
        Dim strWord As String, intCount As Integer
        Dim arrWords() As String
        Dim bCaps As Boolean
        ' Преобразовать строку в массив при помощи System.String.
        arrWords = mstrText.Split(" ")
        For intCount = 0 To UBound(arrWords)
            ' Проверить, начинается ли слово с заглавной буквы,
            If LCase(arrWords(intCount)) o arrWords(intCount) Then
                bCaps = True
                arrWords(intCount) = LCase(arrWords(intCount))
            End If
            strWord = arrWords(intCount)
            ' Выполнитьтрансляцию.
            If strWord <> "" Then
                strWord - Right(strWord, Len(strWord) - 1) & _
```

```
                Left(strWord, 1) & "ay"
                ' Сделать первую букву -заглавной, если нужно.
                If bCaps Then
                    strWord = UCase(Left(strWord, 1)) & _
                      Right(strWord, Len(strWord) - 1)
                End If
            End If
            ' Записать слово обратно в массив.
            arrWords(intCount) - strWord
            ' Сбросить флаг заглавной буквы,
            bCaps - False
        Next
        ' Собрать строку из элементов массива.
        mstrText = String.Join("", arrWords)
    End Sub
End Class
```

Visual C#

```
internal class Translator
{
    string mstrText;

    // Этот код управляет доступом к переменным,
    // объявленным на уровне класса.
    public string Text
    {
        get
        {
            return mstrText;
        }
        set
        {
            mstrText - value;
        }
    }

    // Транслировать значение свойства Text.
    public void Translate()
    {
        string strWord;
        string[] arrWords;
        bool bCaps = false;

        // Преобразовать строку в массив при помощи System.String.
        arrWords = mstrText.Split(' ');
        for(int intCount - 0; intCount <=
```

```
        arrWords.GetUpperBound(0);
             intCount++)
        {
            // Преобразовать в нижний регистр.
            strWord = arrWords[intCount].ToLower();
            // Проверить, начинается ли слово с заглавной буквы.
            if(!arrWords[intCount].Equals(strWord))
                bCaps - true;
            // Выполнить трансляцию.
            if(strWord != "")
            {
                strWord - strWord.Substring(1,strWord.Length
                - 1) + strWord.Substring(0,1) + "ay";
                // Сделать первую букву заглавной, если нужно,
                if(bCaps)
                    strWord = strWord.Substring(0,1).ToUpper() +
                      strWord.Substring(1, strWord.Length - 1);
            }
            // Записать слово обратно в массив.
            arrWords[intCount] = strWord;
            // Сбросить флаг заглавной буквы.
            bCaps = false;
        }
        // Собрать строку из элементов массива.
        mstrText - String.Join(" ", arrWords);
    }
}
```

Свойство *Text* и метод *Translate* класса *Translator* в этом примере удается вызывать только из проекта, в котором находится этот класс; из других проектов эти члены не доступны. Следующий код демонстрирует использование членов *Text* и *Translate* в Web-форме (см. строки, выделенные **полужирным** шрифтом):

Visual Basic .NET

```
Private Sub butTranslate_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butTranslate.Click
    ' Создать новый объект Translator,

Dim TranslateText As New Translator()
    ' Передать введенный текст объекту Translator,
    TranslateText.Text = txtInput.Text
    ' Нажми на кнопку...
    TranslateText.Translate()
    ' ...получишь результат!
    txtOutput.Text = TranslateText.Text
End Sub
```

Visual C#

```
private void butTranslate_Click(object sender, System.EventArgs e)
{
    // Создать новый объект Translator.
    Translator TranslateText = new Translator();
    // Передать введенный текст объекту Translator.
    TranslateText.Text = txtInput.Text;
    // Нажми на кнопку...
    TranslateText.Translate();
    // ...получишь результат!
    txtInput.Text = TranslateText.Text;
}
```

Изучая сгенерированный Visual Studio .NET код Web-формы, обратите внимание, что серверные элементы управления Web-формы объявляются с модификатором *Protected:*

Visual Basic .NET

```
Protected System.Web.UI.WebControls.TextBox txtInput
Protected System.Web.UI.WebControls.Button butTranslate
```

Visual C#

```
protected System.Web.UI.WebControls.TextBox txtInput;
protected System.Web.UI.WebControls.Button butTranslate;
```

По умолчанию объекты серверных элементов управления делают доступными только текущей Web-форме и производным от нее формам.

Наследование: кто от кого родится?

Наследование — это объявление нового *(производного)* класса на основе существующего *(базового).* При этом производный класс использует методы, свойства и прочие члены базового класса. Наследование облегчает повторное применение и сопровождение кода: достаточно поместить код в базовый класс, чтобы сделать его доступным для всех потомков этого базового класса. Степень знакомства программиста с наследованием обычно определяется языком, на котором он пишет.

- С Visual Basic.NET наследование пришло и в Visual Basic. Это был огромный шаг вперед для языка, обусловивший многочисленные изменения в Visual Basic .NET по сравнению с прежними версиями Visual Basic. В отличие от всех предыдущих версий Visual Basic Windows-формы и типы данных теперь происходят от соответствующих классов .NET Framework, а взаимодействие Visual Basic .NET и Windows стало намного более гармоничным и упорядоченным.
- C#-программисты, прежде писавшие на C++, скорее всего хорошо знакомы с концепцией наследования. Однако в C# есть ряд синтаксических особенностей, отсутствовавших в C++, в частности ключевые слова *interface, abstract* и *virtual.*

Выражение наследования зависит от языка программирования. В Visual Basic .NET для объявления нового класса на основе существующего используется оператор *Inherits,* а в C# это делается вовсе без ключевых слов, в определении класса:

Visual Basic .NET

```
Public Class DerivedClass
    Inherits BaseClass
End Class
```

Visual C#

```
public class DerivedClass : BaseClass
{
}
```

В таблице 3-3 приводятся ключевые слова, используемые в Visual Studio для объявления базовых и производных классов. В следующих разделах их применение разъясняется более подробно.

Таблица 3-3. Обзор ключевых слов, используемых при наследовании

Visual Basic	Visual C#	Функция ключевого слова
Inherits	*Имя Базового Класса : ИмяПроизводного-Класса*	Объявляет **на основе базового** класса производный класс, наследующий **члены базового класса**
Overridable	*virtual*	Указывает, что этот член базового класса может быть переопределен в производном классе
Overrides	*override*	Объявляет, что данный член производного класса переопределяет одноименный член базового класса
Shadows	*new*	Указывает, что данный член производного класса замещает одноименный член базового класса
MustInherit	*abstract*	Объявляет данный класс шаблоном для производных классов. Такие классы называют *абстрактными* (abstract), объекты этих классов создавать нельзя
MustOverride	*abstract*	Объявляет данный член шаблоном для членов производных классов. Такие члены называются *абстрактными,* их нельзя вызывать
MyBase	*base*	Вызывает из производного класса член базового класса
Me	*this*	Вызывает член из текущего экземпляра класса
Interface	*interface.*	Объявляет интерфейс с определением членов, которые должен поддерживать класс
Implements	*ИмяКласса : ИмяИнтерфейса*	Использует в классе определение из интерфейса

Прежде чем изучать особенности наследования, следует запомнить, чего *нельзя* делать при помощи наследования. Таких вещей две:

- **нельзя объявлять производный класс на основе двух и более базовых классов. Концепция *множественного наследования* (multiple inheritance) поддерживается во многих языках программирования, но на практике применяется редко. Тем не менее у производного класса может быть несколько предков (базовых классов), если они занимают разные уровни иерархии классов;**

- нельзя породить класс Web-формы от аналогичного базового класса. Хотя Web-формы определяются в виде классов, Visual Studio не поддерживает объявление новых классов Web-форм на основе существующих.

Переопределение, перегрузка и замещение членов

Производный класс наследует члены базового класса. Если производный класс определяет член с той же сигнатурой, считается, что член производного класса *переопределяет* (override) член базового класса. *Сигнатура* (signature) члена включает имя, список его параметров с указанием их типов и тип возвращаемого результата.

Если производный класс определяет член с тем же именем, но с другим набором (или типом) параметров и типом возвращаемого значения, говорят, что член производного класса перегружает или замешает член базового класса. *Перегрузка* (overload) имеет место, если член базового класса остается доступным, в противном случае выполняется *замещение* (shadowing).

В следующем примере класс *Sphere* будет объявлен на основе класса *Circle*. В Visual Basic .NET члены, которые могут быть переопределены, следует объявлять с ключевым словом *Overridable (1),* а в Visual C# — с ключевым словом *virtual (1)*. Для начала приведем код класса *Circle*:

Visual Basic .NET

```
Public Class Circle
    Private sxCenter, syCenter As Single
    Private sRadius As Single

    Public Property Top() As Single
        Get
            Top - sxCenter - sRadius
        End Get
        Set(ByVal Value As Single)
            sxCenter - Value + sRadius
        End Set
    End Property

    Public Property Left() As Single
        Get
            Left - syCenter - sRadius
        End Get
        Set(ByVal Value As Single)
            syCenter = Value + sRadius
        End Set
    End Property

    Public Overridable Function Area() As Single        ' (1)
        Area = System.Math.PI * (sRadius " 2)
    End  Function

    Public Function Perimeter() As Single
```

```
        Perimeter = 2 * sRadius * System.Math.PI
    End Function

    Public Property Radius() As Single
        Get
            Radius - sRadius
        End Get
        Set(ByVal Value As Single)
            sRadius = Value
        End Set
    End Property

    Public Overridable Sub Center(ByVal X As Single, _
      ByVal Y As Single) ' (1)
        sxCenter = X
        syCenter = Y
    End Sub
End Class
```

Visual C#

```
public class Circle
{
    float fxCenter, fyCenter, fRadius;
    // Это конструктор,
    public Circle()
    {
        // Инициализировать внутренние переменные,
        fxCenter = 0;
        fyCenter = 0;
        fRadius = 0;
    }
    public float Top
    {
        get
        {
            return fxCenter - fRadius;
        }
        set
        {
            fxCenter = value + fRadius;
        }
    }
    public float Left
    {
        get
        {
            return fyCenter - fRadius;
```

```
        }
        set
        {
            fyCenter - value + fRadius;
        }
    }

    public virtual float Area()              // (1)
    {
        return (float)(System.Math.PI * Math.Pow((double)fRadius, 2));
    }

    public float Perimeter()
    {
        return 2 * fRadius * (float)System.Math.PI;
    }

    public float Radius
    {
        get
        {
            return fRadius;
        }
        set
        {
            fRadius = value;
        }
    }

    public virtual void Center(float X, float Y)          // (1)
    {
            fxCenter = X;
            fyCenter = Y;
    }
}
```

Класс *Sphere* (см. пример кода далее) наследует все методы и свойства, определенные в классе *Circle (1). Sphere* переопределяет метод *Area (2),* так как площадь сферы рассчитывается иначе, чем окружности. *Sphere* замещает метод *Center (3),* поскольку у сферы есть дополнительная координата (z). Замещение здесь позволяет подстраховаться от невнимательности пользователя, который, задавая координаты центра сферы, может забыть про Z. Заметьте, что *Sphere* использует ключевое слово *MyBase* (это в случае Visual Bask, а для Visual C# — *base),* чтобы из замещающего метода вызывать метод *Center* базового класса *(4).*

Visual Basic .NET

```
Public Class Sphere
    Inherits Circle                                    ' (1)
    Private sCenter As Single
```

```
    Public Overrides Function Area() As Single          ' (2)
        Area = 4 * System.Math.PI * (MyBase.Radius " 2)
    End Function

    Public Shadows Sub Center(ByVal X As Single, _
      ByVal Y As Single, ByVal Z As Single)             ' (3)
        MyBase.Center(X, Y)                             ' (4)
        sCenter - Z
    End Sub

    Public Function Volume() As Single
        Volume = (4 / 3) * System.Math.PI * (Radius " 3)
    End Function

    Public Property Front() As Single
        Get
            Front – sCenter - Radius
        End Get
        Set(ByVal Value As Single)
            sCenter = Value + MyBase.Radius
        End Set
    End Property
End Class
```

Visual C#

```
public class Sphere : Circle                    // (1)
{
    float fCenter;

    // Конструктор.
    public Sphere()
    {
        // Инициализация внутренней переменной.
        fCenter = 0;
    }

    public override float Area()                // (2)
    {
        return (float)(4 * Math.PI * Math.Pow((double)base.Radius, 2));
    }

    public new void Center(float X, float Y)    // (3)
    {
        this.Center(X, Y, 0);
    }

    public void Center(float X, float Y, float Z)
```

```
    {
        base.Center(X, Y);                    //   (4)
        fCenter = Z;
    }

    public float Volume()
    {
        return (float)((4 / 3) * System.Math.PI *
           Math.Pow((double)base.Radius, 3));
    }

    public float Front
    {
        get
        {
            return fCenter - base.Radius;
        }
        set
        {
            fCenter - value + base.Radius;
        }
    }
}
```

Чтобы увидеть наследование в действии, создайте объекты *Circle* и *Sphere* и попробуйте вызвать их свойства и методы. Следующий код демонстрирует работу классов *Circle* и *Sphere,* отображая рассчитанные ими значения на Web-форме:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    Dim MyCircle As New Circle()
    MyCircle.Radius = 2
    MyCircle.Center(10, 2)
    Response.Write("Circle area: " & MyCircle.Area & "<br>")
    Response.Write("Circle circumference: " & MyCircle.Perimeter & "<br>")
    Dim MySphere As New Sphere()
    MySphere.Radius = 10
    MySphere.Center(10, 20, 25)
    Response.Write("Sphere top: " & MySphere.Top & "<br>")
    Response.Write("Sphere left: " & MySphere.Left & "<br>")
    Response.Write("Sphere front: " & MySphere.Front & "<br>")
    Response.Write("Sphere volume: " & MySphere.Volume & "<br>")
    Response.Write("Sphere surface area: " & MySphere.Area & "<br>")
    Response.Write("Sphere circumference: " & MySphere.Perimeter & "<br>")
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
    {
        Circle MyCircLe = new Circle();
        MyCircle.Radius = 2:
        MyCircle.Center(10, 2);
        Response.Write("Circle area: " + MyCircle.Area() + "<br>");
        Response.Write("Circle circumference: " + MyCircle.Perimeter() + "<br>");
        Sphere MySphere = new Sphere();
        MySphere.Radius = 10;
        MySphere.Center(10,20,25);
        Response.Write("Sphere top  " + MySphere.Top + "<br>");
        Response.Write("Sphere left: " + MySphere.Left + "<br>");
        Response.Write("Sphere front: " + MySphere.Front + "<br>");
        Response.Write("Sphere volume: " + MySphere.Volume() + "<br>");
        Response.Write("Sphere surface area: " + MySphere.Area() + "<br>");
        Response.Write("Sphere circumference: " + MySphere.Perimeter() + "<br>");
    }
```

Немного абстракции

Среди прочего Visual Studio допускает объявление абстрактных классов. *Абстрактный* класс определяет интерфейс производных классов, по существу являясь соглашением о том, что все объявленные на его основе классы будут поддерживать определенные методы и свойства. Нельзя создать объект абстрактного класса, можно лишь использовать его как предок новых классов.

В Visual Basic абстрактные классы объявляются при помощи ключевого слова *Must-Inherit,* а в Visual C# для этого служит ключевое слово *abstract.* Все методы и свойства, которые должны быть шаблонами членов производных классов, объявляются в Visual Basic с ключевым словом *MustOverride*; а в Visual C# — с ключевым словом *abstract.* Примером абстрактного класса может служить следующий класс *Shape:*

Visual Basic .NET

```
' Определение абстрактного класса,
Public MustInherit Class Shape
    Public MustOverride Property Top() As Single
    Public MustOverride Property Left() As Single
    Public MustOverride Function Area() As Single
    Public MustOverride Function Perimeter() As Single
End Class
```

Visual C#

```
// Определение абстрактного класса.
public abstract class Shape
{
    public Shape()
    {
```

```
    }
    public abstract float Top
    {
        get;
        set;
    }
    public abstract float Left
    {
        get;
        set;
    }
    public abstract float Area();

    public abstract float Perimeter();

}
```

Обратите внимание, что члены, объявленные с ключевым словом *MustOverride* (на Visual Basic.NET, а для Visual C# — *abstract*) — это пустые определения, в них нет операторов, составляющих тело процедур, поскольку эти члены следует определять (переопределять) в производных классах.

Следующий класс *Circle* демонстрирует наследование от абстрактного класса. В программе на Visual Basic .NET оператор *Inherits* (*1*) объявляет, что данный класс является производным от абстрактного класса *Shape*. В объявлении каждого из членов, переопределяющих члены абстрактного класса, должно быть ключевое слово *Overrides (2)*. В варианте примера, написанном на Visual C#, определение класса объявляет абстрактный класс *Shape* базовым классом *Circle (1)*. При этом в определении каждого члена, переопределяющего член абстрактного класса, также должно быть ключевое слово *override (2)*.

Visual Basic .NET

```
Public Class Circle
    Inherits Shape                              ' (1)
    Private sxCenter, syCenter As Single
    Private sRadius As Single

    Public Overrides Property Top() As Single   ' (2)
        Get
            Top = sxCenter - sRadius
        End Get
        Set(ByVal Value As Single)
            sxCenter - Value + sRadius
        End Set
    End Property

    Public Overrides Property Left() As Single
        Get
            Left = syCenter - sRadius
        End Get
```

```
        Set(ByVal Value As Single)
            syCenter - Value + sRadius
        End Set
    End Property

    Public Overrides Function Area() As Single
        Area = 2 * System.Math.PI * (sRadius " 2)
    End Function

    Public Overrides Function Perimeter() As Single
        Perimeter = 2 - sRadius * System.Math.PI
    End Function

    Public Property Radius() As Single
        Get
            Radius = sRadius
        End Get
        Set(ByVal Value As Single)
            sRadius = Value
        End Set
    End Property

    Public Overridable Sub Center(ByVal X As Single, ByVal Y As Single)
        sxCenter = X
        syCenter - Y
    End Sub
End Class
```

Visual C#

```
public class Circle : Shape                    // (1)
{
    float fxCenter, fyCenter, fRadius;

    // Конструктор.
    public Circle()
    {
        // Инициализация внутренних переменных.
        fxCenter = 0:
        fyCenter = 0;
        fRadius - 0;
    }

    public override float Top                  // (2)
    {
        get
        {
```

```
            return fxCenter - fRadius;
        }
        set
        {
            fxCenter = value + fRadius;
        }
    }

    public override float Left
    {
        get
        {
            return fyCenter - fRadius;
        }
        set
        {
            fyCenter - value + fRadius;
        }
    }

    public override float Area()
    {
        return (float)(2 - System.Math.PI - Math.Pow((double)fRadius, 2));
    }
        public override float Perimeter()
    {
        return 2 * fRadius * (float)System.Math.PI;
    }
        public float Radius
    {
        get
        {
            return fRadius;
        }
        set
        {
            fRadius - value;
        }
    }

    public virtual void Center(float X, float Y)
    {
            fxCenter - X;
            fyCenter = Y;
    }
}
```

Лицом к интерфейсу

Интерфейсы (interfaces) подобны абстрактным классам в том, что и те и другие предоставляют шаблон для новых классов. Отличаются они тем, что в интерфейсах начисто отсутствует реализация членов, в то время как в абстрактных классах могут быть реализованы некоторые члены, которые впоследствии станут общими для всех производных от них классов.

Интерфейсы больше похожи на типы, чем на классы. Если реализовать в классе некоторый интерфейс, то объекты этого класса можно использовать как значения аргументов и переменных, объявленных с типом этого интерфейса. Например, следующий код объявляет интерфейс для объектов фигур, которые создавались в примерах, показанных ранее:

Visual Basic .NET

```
' Интерфейс для всех фигур.
Public Interface IFigure
    Property Top() As Single
    Property Left() As Single
    Function Area() As Single
    Function Perimeter() As Single
End Interface
```

Visual C#

```
// Интерфейс для всех фигур.
public interface IFigure
{
    float Top
    {
        get;
        set;
    }
        float Left
    {
        get;
        set;
    }
        float Area();
        float Perimeter();
}
```

Чтобы использовать этот интерфейс, следует объявить его в классе следующим образом (см. строку, выделенную **полужирным шрифтом):**

Visual Basic .NET

```
' Определение абстрактного класса.
Public MustInherit Class Shape
    Implements IFigure
    Public MustOverride Property Top() As Single Implements IFigure.Top
    Public MustOverride Property Left() As Single Implements IFigure.Left
```

```
    Public MustOverride Function Area() As Single Implements IFigure.Area
    Public MustOverride Function Perimeter() As Single Implements
       IFigure.Perimeter
End Class
```

Visual C#

```
public abstract class Shape : IFigure
{
    // Конструктор.
    public Shape()
    {
    }

    public abstract float Top
    {
        get;
        set;
    }
        public abstract float Left
    {
        get;
        set;
    }

    public abstract float Area();

    public abstract float Perimeter();

}
```

Поскольку здесь интерфейс *IFigure* реализован в абстрактном классе *Shape,* реализацию интерфейса *IFigure* унаследуют все классы, производные от *Shape.* Это означает, что объекты классов *Circle* и *Sphere* (классов-потомков *Shape)* могут служить значениями аргументов, объявленных с типом *IFigure,* как показано в следующих примерах:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Создать окружность.
    Din MyCircle As New Circle()
    MyCircle.Radius - 2
    MyCircle.Center(10, 2)
    ' Создать сферу.
    Dim MySphere As New SphereC)
    MySphere.Radius - 10
    MySphere.Center(10, 20, 25)
    ' Показать сведения о каждой из фигур,
```

```
    ShowShapeInfo(MySphere)
    ShowShapeInfo(MyCircle)
End Sub

' Отобразить сведения о фигуре на Web-форме.
Sub ShowShapeInfo(ByVal Shape As IFigure)
    Response.Write("Shape top: " & Shape.Top & "<br>")
    Response.Write("Shape left: " & Shape.Left & "<br>")
    Response.Write("Shape volume: " & Shape.Volume & "<br>")
    Response.Write("Shape surface area: " & Shape.Area & _
      "<br>")
    Response.Write("Shape perimeter: " & _
      Shape.Perimeter & "<br>")
End Sub
```

Visual C#

```
private void Page_Load(object serder, System.EventArgs e)
{
    // Создать окружность.
    Circle MyCircle - new Circle();
    MyCircle.Radius = 2;
    MyCircle.Center(10, 2);
    // Создать сферу.
    Sphere MySphere = new Sphere();
    MySphere.Radius - 10;
    MySphere.Center(10,20,25);
    // Показать сведения о каждой из фигур.
    ShowShapeInfo(MySphere);
    ShowShapeInfo(MyCircle);
}

// Отобразить сведения о фигуре на Web-форме.
private void ShowShapeInfo(IFigure Shape)
{
    // Поскольку аргумент Shape имеет тип IFigure,
    // мы знаем, что у него есть необходимые члены.
    Response.Write("Shape top: " + Shape.Top + "<br>");
    Response.Write("Shape left: ' + Shape.Left + "<br>");
    Response.Write("Shape perimeter: " + Shape.Perimeter() + "<br>");
    Response.Write("Shape area: " + Shape.Area() + "<br>");
}
```

Важно, чтобы в каждом классе, где реализован **интерфейс,** обязательно были все элементы, **определенные в этом** интерфейсе. **Если пропустить хоть один из них, Visual Studio генерирует ошибку времени** компиляции.

Занятие 2. Пространства имен в Web-приложениях

Из этого занятия вы узнаете, как найти среди пространств имен ASP.NET объекты, необходимые для создания Web-приложений. Кроме того, вы научитесь работать с объектами *Application, Page, Request* и *Response,* составляющими основу программирования Web-приложений.

Изучив материал этого занятия, вы сможете:

- ✓ найти в .NET Framework объекты, необходимые для создания Web-приложения;
- ✓ понять взаимосвязи объектов в архитектуре ASP.NET;
- ✓ получать объекты ASP.NET и управлять ими в коде;
- ✓ просматривать иерархию объектов для получения подчиненных объектов ASP.NET, таких, как *Browser* и *Cookies.*

Продолжительность занятия — около 30 минут.

Обзор пространства имен Web

Определения классов, представляющих объекты, которые используются в Web-приложениях, находятся в пространстве имен *System-Web.* В таблице 3-4 определена иерархия пространств имен, расположенных в *System. Web,* и описаны определения классов, которые находятся в каждом из них. Эта таблица пригодится вам при поиске объектов, необходимых для выполнения различных задач,

Таблица 3-4. Иерархия пространства имея *System. Web*

Пространство имен	Какие классы содержит
System. Web	*Application, Browser, Cache, Cookies, Exception, Request, Response, Server* и *Trace.* Эти классы необходимы для решения большинства задач Web-программирования. Объект *Application,* определяемый в файле Global.asax, является экземпляром класса *Application*
System. Web.Session-Stare	*Session.* Используется для сохранения и извлечения элементов из переменных состояния *Session*
System. Web.Services	*WebService.* Требуется для создания Web-сервисов и работы с ними
System.Web.UI	*Page* и *Control.* Применяется в Web-формах для создания пользовательского интерфейса и управления им, Web-формы представлены классом *Page*
System.Web.UI.Web-Controls	Классы всех серверных элементов **управления,** используемых в Web-формах
System. Web. UI.-HTMLControls	Классы всех HTML-элементов управления, применяемых в Web-формах
System. Web.Caching	*Cache.* Необходим для управления кэшированием на стороне сервера с целью повышения производительности приложения
System. Web.Mail	*MailMessage, MailAttachment и SmtpMail.* **Требуются для отправки из приложения сообщений** по электронной **почте**
System. Web.Security	Содержит классы для объектов и модулей аутентификации, используемых для аутентификации пользователей и обеспечения конфиденциальности при работе с приложениями

При программировании Web-приложений приходится непосредственно иметь дело с двумя типами объектов, созданными на основе классов из пространства имен Web:

- объект ***Application***. Происходит от класса *HttpApplication*. В приложении соответствующее определение находится в файле Global.asax;
- объекты Web-форм. Происходят от класса *Page*. В приложении соответствующее определение хранится в модулях форм.

Следующий код демонстрирует объявления классов *Global* и *WebForm1*, генерируемые Visual Studio. ASP.NET автоматически создает экземпляры этих классов во время выполнения.

Visual Basic .NET

```
Public Class Global
    Inherits System.Web.HttpApplication
End Class

Public Class WebForm1
    Inherits System.Web.UI.Page
End Class
```

Visual C#

```
public class Global : System.Web.HttpApplication
{
}

public class WebForm1 : System.Web.UI.Page
{
}
```

Объекты *Global* и *WebForm1* — точки входа, открывающие доступ к остальным Web-объектам приложения. С ними, а также с объектами *Request* и *Response* в коде приходится работать особенно часто. В следующем разделе разъясняется, как с помощью объектов *Application*, *Page*, *Request* и *Response* получать другие объекты.

Использование объекта Application

Объект *Application* занимает верхний уровень иерархии объектов Web-приложения и служит для конфигурирования приложения и сохранения сведений о состоянии. Свойства и методы объекта *Application* открывают доступ к другим объектам Web-приложения (рис. 3-2).

Запуская приложение, ASP.NET автоматически создает экземпляр объекта *Global*, определенного в файле Global.asax. С помощью событий объекта *Global* удается конфигурировать приложение и инициализировать переменные состояния, объявленные на уровне приложения.

От базового класса (*HttpApplication*) объект *Global* наследует свойства и методы (таблица 3-5) для доступа к своим подчиненным объектам. Эти свойства и методы служат для получения других объектов из объекта *Application*.

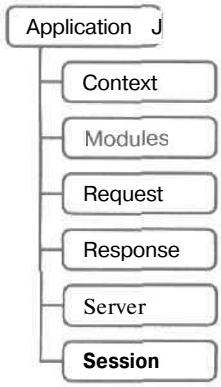


Рис. 3-2. Объект *Application*

Таблица 3-5. Свойства и методы объекта *Application*

Имя	Назначение
Application	Сохранение элементов данных в переменных состояния *Application*
Context	Получение объектов *Trace, Cache, Error* и других для текущего контекста
Modules	Доступ к модулям HTTP
Request	Чтение запросов и получение объектов *Browser, ClientCertificates, Cookies* и *Files* из текущего запроса
Response	Запись в отклик текста и данных, а также получение объектов *Cache, Cookies* и *Output* из текущего отклика
Server	Обработка запросов и откликов. Вспомогательные методы для кодирования и декодирования URL поддерживаются объектом *Server*
Session	Сохранение данных в переменных состояния *Session*
User	Получение результатов аутентификации пользователя, сделавшего текущий запрос. По умолчанию Web-приложения разрешают анонимный доступ

Следующий код проверяет при помощи объектов *Request* и *Browser,* доступных из объекта *Application,* прислан ли запрос браузером с номером версии 4.0 или выше.

Visual Basic .NET

```
Sub Application_BeginRequest(ByVal sender As Object, _
  ByVal e As EventArgs)
    ' Этот код исполняется перед обработкой каждого запроса.
    If Request.Browser.MajorVersion < 4 Then
        ' Отключить дополнительные возможности.
    End If
End Sub
```

Visual C#

```
protected void Application_BeginRequest(Object sender, EventArgs e)
{
```

```
    // Этот код исполняется перед обработкой каждого запроса.
    if (Request.Browser.MajorVersion < 4)
    {
        // Отключить дополнительные возможности.
    }
}
```

Использование объекта Page

Объект *Page* управляет пользовательским интерфейсом приложения. Свойства и методы объекта *Page* обеспечивают доступ к другим объектам пользовательского интерфейса (рис. 3-3).

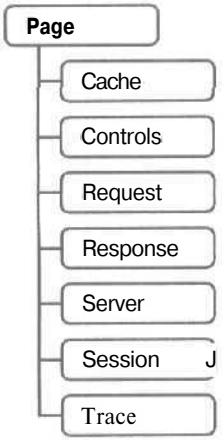


Рис. 3-3. Объект *Page*

Когда пользователь запрашивает страницу У приложения, ASP.N **ET** автоматически создает экземпляр Web-формы и отображает запрошенную страницу, а написать обработчики событий, управляющие пользовательским интерфейсом и взаимодействующие с пользователем, — забота разработчика.

Основные методы и свойства, которые чаще всего требуются при программировании Web-форм (их сводка дана в таблице 3-6), наследуются от базового класса *Page*.

Таблица 3-6. Свойства и методы объекта *Page*

Имя	Назначение
Application	Сохранение данных в переменных состояния *Application*
Cache	Управление кэшированием откликов на сервере
Controls	Получение элементов управления страницы
Request	Чтение запросов и получение из текущего запроса объектов *Browser, ClientCertificates, Cookies* и *Files*
Response	Запись текста и данных в отклик, а также получение из текущего отклика объектов *Cache, Cookies* и *Output*
Server	Обработка запросов и откликов. Объект *Server* предоставляет вспомогательные методы для кодирования и декодирования URL
Session	Сохранения элементов данных в переменных состояния *Session*
Trace	Включение и выключение трассировки, а также запись трассировочных данных в журнал

Следуюший код добавляет новый элемент управления на Web-форму во время выполнения:

Visual Basic

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
        ' Создать новый элемент управления,
        Dim txtNew As New TextBox()
        ' Записать в поле элемента управления какой-нибудь текст.
        txtNew.Text = "Some new text"
        ' Добавить элемент управления, заключенный между тэгами
        ' <form> и </form>.
        FindControl("Form1").Controls.Add(txtNew)
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
        // Создать новый элемент управления.
    TextBox txtNew = new Text8ox();
        //  Записать в поле элемента управления какой-нибудь текст.
            txtNew.Text - "Some new text";
        // Добавить элемент управления, заключенный между тэгами
        // <form> и </form>.
    FindControl["Form1"].Controls.Add(txtNew);
}
```

Использование объекта Request

Объект *Request* содержит информацию, присланную клиентским браузером при !.апросе страницы приложения. Свойства и методы объекта *Request* открывают доступ к другим объектам запроса (рис. 3-4).



Рис. 3-4. Объект *Request*

Свойства и методы объекта *Request* открывают доступ к его подчиненным объектам (таблица 3-7).

Таблица 3-7. Свойства и методы объекта *Request*

Имя	Назначение
Browser	Определение возможностей запрашивающего браузера. Свойства этого объекта указывают номер версии браузера, является ли он браузером AOL, поддерживает ли он файлы cookie и некоторую другую информацию
ClientCertificates	Аутентификация клиента
Cookies	Получение информация от клиента в виде файлов cookie
Files	Получение файлов, закачиваемых клиентом
InputStream	Чтение и запись присланного запроса в виде неструктурированных данных

Следующий код использует объект *Request*, чтобы перед сохранением содержимого cookie в переменной состояния *Session* проверить, поддерживает ли браузер файлы cookie и существует ли этот файл cookie.

Visual Basic .NET

```
Private Sub Page_Loac(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    ' Этот код исполняется при первом отображении страницы.
    If Not IsPostBack Then
        ' Проверить, поддерживает ли браузер файлы cookie.
        If Request.Browser.Cookies Then
            ' Проверить, существует ли cookie по имени UName.
            If Not IsNothing(Request.Cookies("UName")) Then
                ' Сохранить значение cookie.
                Session("User") - Request.Cookies("UName").Value
            End If
        End If
    End If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Этот код исполняется при первом отображении страницы.
    if(!IsPostBack)
        // Проверить, поддерживает ли браузер файлы cookie.
        if(Request.Browser.Cookies)
            // Проверить, существует ли cookie по имени UName,
            if(Request.Cookies["UName"] != null)
                // Получить значение cookie.
                Session["User"] = Request.Cookies["UName"].Value;
}
```

Использование объекта Response

Объект *Response* применяется для формирования отклика сервера, пересылаемого клиентскому браузеру. Свойства и методы объекта *Response* обеспечивают доступ к остальным объектам, составляющим запрос (рис. 3-5).

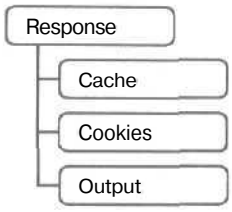


Рис. 3-5. Объект *Response*

В таблице 3-8 перечислены свойства и методы объекта *Response,* обеспечивающие доступ к его подчиненным объектам.

Таблица 3-8. Свойства и методы объекта *Response*

Имя	Назначение
Cashe	Определяет, как сервер будет кэшировать отклики перед отправкой их клиенту
Cookies	Задает содержимое файлов cookie, предназначенных для пересылки клиенту
Output	Чтение и запись неструктурированных данных, возвращаемых клиенту в виде отклика

Следующий код создает cookie и посылает его клиенту в составе отклика.

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
        ' Этот код исполняется при первом отображении страницы,
        If Not IsPostBack Then
            ' Если браузер поддерживает cookies...
            If Request.Browser.Cookies Then
                ' ...создать cookie...
                Dim cookUname As New HttpCookie("UName")
                cookUname.Value = "Wombat"
                ' ... и добавить его к отклику.
                Response.Cookies.Add(cookUname)
            End If
        End If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{       // Исполняется при первом отображении страницы
    if(!IsPostBack)
        // Если браузер поддерживает cookies...
```

```
        if (Request.Browser.Cookies)
        {
            // ...создать cookie...
            HttpCookie cookUname = new HttpCookie("UName");
            cookUname.Value = "Wombat";
            // ... и добавить его к отклику.
            Response.Cookies.Add(cookUname);
        }
}
```

Занятие 3. Сохранение сведений о состоянии

Из этого занятия вы узнаете, как сохранять данные Web-формы в промежутках между запросами. В ASP.NET существует несколько механизмов сохранения данных формы, которые применяются в зависимости от природы и области действия данных.

Изучив материал этого занятия, вы сможете:

- ✓ выбирать подходящий способ сохранения данных Web-формы;
- ✓ передавать данные между Web-формами с помощью строк запросов;
- ✓ сохранять данные у клиента в виде cookies;
- ✓ хранить страничные данные в свойстве *View State* Web-формы;
- ✓ сохранять значения переменных, объявленных на уровне приложения, в переменных состояния *Application* и *Session;*
- ✓ упорядочивать доступ к переменным состояния *Application* и *Session* во избежание ошибок при программировании.

Продолжительность занятия — около 30 минут.

Способы сохранения сведений о состоянии

Как вы знаете из главы 2, Web-формы создаются и уничтожаются при каждом запросе клиентского браузера. В силу этих особенностей значения переменных, объявленных в Web-форме, пропадают после ее отображения. В ASP.NET существуют различные средства сохранения переменных между запросами, позволяющие решить эту проблему:

- **строки запросов** — необходимы для передачи информации между запросами и откликами в составе Web-адреса. Строки запросов видны пользователю, поэтому в них нельзя передавать секретную информацию, например пароли;
- **файлы** cookie — применяются для хранения небольших порций информации на клиентской машине. Клиент может отказаться от записи cookie, это нужно предусмотреть при написании кода;
- **состояние отображения** — ASP.NET хранит элементы, записанные в страничное свойство *View State*, в скрытых полях страницы;
- **переменные состояния Session** — требуются для хранения данных, локальных для текущего сеанса (доступных единственному пользователю);
- **переменные состояния *Application*** — применяются для хранения данных, которые должны быть доступны всем пользователям приложения.

Использование строк запросов

Строки запросов позволяют пересылать вместе с адресом дополнительную информацию. В HTML-тэгах гиперссылок строка запроса следует после знака вопроса:

```
<A HREF= "WebForm1.aspx?UName=Wombat">Query string sample.</A>
```

Чтобы переслать строку запроса в коде, ее следует добавить к адресу, передаваемому методу *Redirect.* Ниже показан обработчик события *Click,* функционально эквивалентный предыдущему HTML-тэгу:

Visual Basic .NET

```
Private Sub Button1_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button1.Click
```

```
        ' Отобразить страницу еще раз с учетом строки запроса.
        Server.Redirect("WebForm1.aspx?UName=Wombat")
End Sub
```

Visual C#

```
private void Button1_.Click(object sender. System. EventArgs e)
{
    // Отобразить страницу еще раз с учетом строки запроса.
      Response.Redirect("Webform1.aspx?UName=Wombat");
}
```

Для извлечения строк запросов в коде применяется метод *QueryString* объекта *Request*. Следующий код отображает элемент *Uname*, извлеченный из строки запроса, созданной в предыдущих примерах:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
        ' Отобразить строку запроса.
        Response.Write (Request.QueryString("UName"))
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Отобразить строку запроса,
    Response.Write(Request.QueryString["UName"]);
}
```

Использование файлов cookie

Файлы cookie применяются для хранения небольших порций данных на компьютере клиента. Web-сайты часто обращаются к файлам cookie для хранения личных параметров и иной информации клиентов. Поскольку клиент может отказаться от cookie, перед попыткой записи cookie важно убедиться, что это разрешено браузером.

Следующий код проверяет, разрешает ли браузер **запись cookie, и если да, сохраняет** личные **параметры пользователя.**

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Hardies MyBase.Load
        ' Этот код исполняется при первом отображении страницы.
        If Not IsPostBack Then
            ' Если браузер поддерживает файлы cookie...
            If Request.Browser.Cookies Then
                ' .. создать cookie...
                Din cookUPrefs As New HttpCookie("UPrefs")
                cookUPrefs.Value = "English"
```

```
            ' ...и добавить в отклик,
            Response.Cookies.Add(cookUPrefs)
        End If
    End If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Этот код исполняется при первом отображении страницы,
    if(!IsPostBack)
        // Если браузер поддерживает файлы cookie...
        if(Request.Browser.Cookies)
        {
            // ...создать cookie...
            HttpCookie cookUPrefs - new HttpCookie("UPrefs");
            cookUPrefs.Value = "English";
            // ...и добавить в отклик.
            Response.Cookies.Add(cookUPrefs);
        }
}
```

Следующий код проверяет наличие файла cookie, и если проверка успешна, получает его.

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
        ' Этот код исполняется при первом отображении страницы.
        If Not IsPostBack Then
            ' Если браузер поддерживает файлы cookie...
            If Request.Browser.Cookies Then
                ' ...проверить, существует ли cookie UPrefs.
                If Not IsNothing(Request.Cookies("UPrefs")) Then
                    ' Сохранить значение cookie.
                    Session("Lang") = Request.Cookies("UPrefs").Value
                End If
            End If
        End If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Этот код исполняется при первом отображении страницы.
    if(!IsPostBack)
        // Если браузер поддерживает файлы cookie...
```

```
        if(Request.Browser.Cookies)
            // ...проверить, существует ли cookie UPrefs.
            if(Request.Cookies["UPrefs"] != null)
                // Сохранить значение cookie.
                Session["Lang"] = Request.Cookies["UPrefs"].Value;
}
```

Использование состояния отображения

Свойство *ViewSiate* служит **для хранения** данных **в скрытых полях страницы. Поскольку** *ViewState* **сохраняет** данные **в странице, этот способ** годится **лишь** для тех **элементов, которые можно** сериализовать. **Для сохранения в** *ViewState* более **сложных** элементов **их придется** сначала **преобразовать в строку, а** после **извлечения —** вернуть в **исходный вид.**

Следующий код извлекает текст из текстового поля и помещает его в ячейки таблицы, расположенной на странице. Поскольку нельзя хранить объекты непосредственно в свойстве *ViewState*, процедура *Button1_Click* сначала сохраняет полученные строки в *ViewSiate,* а затем процедура *Page Load* создает на основе сохраненных текстовых строк элементы управления, составляющие таблицу.

Visual Basic .NET

```
Private Sub Button1_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button1.Click
        ' Записать текст в ViewState,
        ViewState.Add(ViewState.Count, TextBox1.Text)
End Sub

Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
        Dim iCount As Integer
        ' Для каждого элемента ViewState
        For iCount = 0 To ViewState.Count
            Dim rowNew As New TableRow()
            Dim celNew As New TableCell()
            ' Записать текст в ячейку.
            celNew.Text = ViewState(iCount)
            ' Добавить ячейку в строку.
            rowNew.Cells.Add(celNew)
            ' Добавить строку в таблицу.
            Table1.Rows.Add(rowNew)
        Next
End Sub
```

Visual C#

```
private void Button1_Click(object sender, System.EventArgs e)
{
    // Записать текст в ViewState.
    ViewState.Add(ViewState.Count.ToString(), TextBox1.Text);
```

```
}

private void Page_Load(object sender, System.EventArgs e)
{
    if (IsPostBack)
        // Для каждого элемента ViewState
        foreach(StateItem staItem in ViewState.Values)
        {
            TableRow rowNew = new TableRow();
            TableCell celNew = new TableCell();
            // Записать текст в ячейку.
            celNew.Text - staItem.Value.ToString();
            // Добавить ячейку в строку.
            rowNew.Cells.Add(celNew);
            // Добавить строку в таблицу.
            Table1.Rows.Add(rowNew);
        }
}
```

ASP.NET кодирует скрытые страничные данные, поэтому пользователь не сможет прочитать их из исходного текста страницы. Если, воспользовавшись предшествующим кодом, добавить несколько элементов в таблицу, а после выбрать в меню браузера элемент View\View Source (Вид\В виде HTML), скрытые поля будут выглядеть примерно так:

```
</HEAD>
<body MS_POSITIONING="GridLayout">
<form name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE" value="dDwtMTMwNzIzMzUOMzt0PHA8b
DwwOzE7Mj szOzQ7PjtsPFxlO1RoaXMgaXMgc29tZSBOZXh
OO1RoaXMgaXMgc29tZSBOZXhOO1RoaXMgaXMgc29tZSBOZXhOO1RoaXMgaXMgc29tZSBOZXh
OOz4+Ozs+Oz4=" />
```

Использование переменных состояния Application и Session

Переменные состояния *Application* и *Session* необходимы для хранения данных, которые требуются на протяжении времени жизни приложения или отдельного сеанса соответственно. В переменных состояния *Application* и *Session* можно хранить данные любого типа, включая объекты. Но учтите следующее, прежде чем пользоваться этим механизмом:

- переменные состояния *Application* и *Session* создаются «на лету», при этом не выполняются обычная проверка имени переменной и контроль типов. Следует ограничивать число точек доступа к этим переменным;
- обслуживание переменных состояния *Session* снижает производительность. Эти переменные разрешается отключать на уровне приложения или страницы;
- переменные состояния *Application* доступны только в текущем процессе, доступ к ним из другого процесса невозможен. В случае масштабируемого приложения, работающего на нескольких серверах (или на многопроцессорном сервере), у каждого процесса этого приложения будет своя переменная состояния *Application*;
- область видимости переменных состояния *Application* определяется границами приложения, заданными IIS,

Упорядочение доступа к переменным состояния

Переменные состояния *Application* и *Session* — мощное и потому опасное средство. Если использовать их бессистемно, проще простого наделать ошибок в коде, как часто бывало в старом Basic. Например, следующий код создает две переменных состояния *Application*:

Visual Basic .NET

```
Application("Uname") = "Wombat"
Response.Write(Application("Unamme"))
```

Visual C#

```
Application["Uname"] = "Wombat";
Response.Write(Application["Unamme"]);
```

Код в первой строке создает переменную и сохраняет в ней некоторый текст. Из-за банальной опечатки в имени переменной код следующей строки создаст и отобразит на странице новую, пустую переменную, в результате на странице тоже будет пусто. В этом случае найти ошибку просто, чего не скажешь, если такая ошибка возникнет где-то в глубине *структуры решений* (decision structure).

Во избежание подобных проблем следует упорядочивать доступ к переменным состояния *Application* и *Session*. Проще всего сделать это следующим образом. Для каждого необходимого элемента данных объявляются переменные на уровне страницы, затем при исполнении обработчика события *Page_Load* в них записываются значения переменных состояния *Application* и *Session*, а при исполнении обработчика события *Page_Unload* значения переменных, объявленных на уровне страницы, возвращаются в переменные состояния. Упорядоченное чтение и запись переменных состояния — полезный прием программирования, который рекомендуется взять на вооружение.

Следующий код иллюстрирует упорядоченный доступ к переменным состояния:

Visual Basic .NET

```
Public Class WebForm1

    ' Объявить переменную на уровне страницы.
    Dim mstrUname As String

    Private Sub Page_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load
        ' Получить значение переменной состояния.
        mstrUname = Application("Uname")
        ' Присвоить значение.
        mstrUname = "Wombat"
        ' Использовать переменную.
        Response.Write(mstrUname)
    End Sub

    Private Sub Page_Unload(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles MyBase.Unload
        ' Вернуть значение в переменную состояния,
```

```
        Application("Uname") = mstrUname
    End Sub

End Class
```

Visual C#

```
string mstrUname = "";

private void Page_Load(object sender, System.EventArgs e)
{
    // Проверить, существует ли переменная состояния.
    if(Application["Uname"] != null)
        // Получить значение переменной состояния.
        mstrUname = Application["Uname"].ToString();
    // Присвоить значение.
    mstrUname = "Wombat";
    // Использовать переменную,
    Response.Write(mstrUname);
}

private void Page_UnLoad(object sender, System.EventArgs e)
{
    // Вернуть значение в переменную состояния.
    Application["Uname"] = mstrUname;
}
```

Внимание! В случае Visual C#, прежде чем вызывать любые методы переменной состояния, например *ToString()* обязательно убедитесь, что ее значение отлично от *null,* иначе вы получите ошибку времени выполнения из-за пустой переменной состояния.

Отключение переменных состояния Session

По умолчанию ASP.NET поддерживает переменную состояния *Session* для каждой страницы приложения. Если странице не нужна сеансовая переменная состояния, то для повышения производительности ее можно отключить.

Чтобы отключить переменную состояния *Session* для одной Web-формы:

- вызовите окно Web Form Properties для формы и установите *EnableSessionState* как *False.*

Чтобы отключить переменные состояния *Session* во всем приложении:

- в файле Web.config установите для тега *<sessionstate mode>* значение *off:*

```
<sessionstate mode="off" ...>
```

Резюме

- Пространства имен позволяют упорядочить собственный код и использовать код из .NET Framework. Можно также обращаться к пространствам имен, внешним по отношению к проекту. Для этого нужно добавить ссылку на содержащий его компонент при помощи диалогового окна References, доступного из меню Project. Чтобы использовать в коде сокращенные имена для ссылки на члены, расположенные в некотором пространстве имен, добавьте в исходный текст оператор *Imports* (в случае Visual Basic .NET) или *using* (в случае Visual C#).
- Классы являются определениями объектов в пространстве имен. Наследование позволяет объявлять новые классы на основе существующих, при этом производный класс может использовать методы базового класса в неизменном виде, либо модифицировать их посредством переопределения, перегрузки или замещения.
- Web-приложения используют пространства имен *System. Web* и *System. Web. VI.* Эти пространства имен определяют большинство объектов Web-приложений, в том числе *Application, Page, Request* и *Response,* открывающие доступ к большинству своих подчиненных объектов, используемых в Web-приложении.
- Поскольку срок жизни обычных переменных, объявленных в Web-приложении, краток, необходимо сохранять их значения в промежутках между запросами. **ASP.NET** позволяет сохранять элементы данных в строке запросов, файлах cookie, состоянии отображения (в значении свойства *View State),* а также в переменных состоянии *Session* и *Application.*

Практикум 3. Работа с объектами и сохранение данных

В этом практикуме вы создадите приложение Translator. Для этого потребуется сконструировать Web-форму и написать модуль Class. Вы создадите экземпляр объекта из модуля Global, затем сохраните новый объект в переменной состоянии *State,* которая наряду с состоянием отображения применяется в Web-форме. Готовое приложение выглядит так, как показано на рис. 3-6.

Продолжительность практикума — около 20 минут.

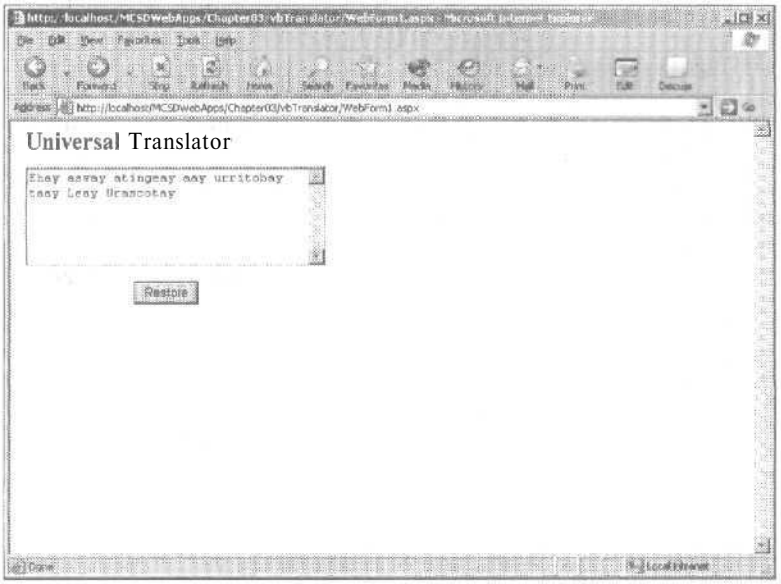


Рис. 3-6. Готовое приложение Translator

Упражнение 1. Создание пользовательского интерфейса



Сейчас вы создадите пользовательский интерфейс приложения, поместив на Web-форму серверные элементы управления.

▶ Создание нового проекта Web-приложения

1. Откройте новый проект ASP.NET Web application. В диалоговом окне New Project введите имя **Translator** и щелкните OK.
2. В окне Properties переименуйте файл WebForm1.aspx в **Translator.aspx**.

▶ Добавление элементов управления к пользовательскому интерфейсу

Поместите на Web-форму серверные элементы управления, показанные на рис. 3-6, и задайте для них следующие свойства:

Имя элемента управления	Тип	Свойство	Значение
Label1	Надпись	Text	Universal Translator
		Font	Bold, Large
TextBoxl	Текстовое поле	[D	txtSource
		Text Mode	Multiline
Button1	Кнопка	ID	butTranslate
		Text	Translate

Упражнение 2. Создание класса TranslatorClass



Сейчас вы объявите класс с модификатором доступа *Friend,* заключающий в себе логику и данные приложения Translator.

► **Как создать класс**

1. В меню Project выберите команду Add Class. Visual Studio .NET откроет диалоговое окно Add New Item. Назовите класс **TranslatorClass.vb** и щелкните OK.
2. В окне Code введите следующий код:

Visual Basic .NET

```
Friend Class TranslatorClass
    Private mstrText As String
    Private mstrOriginal As String

    ' Этот код управляет доступом к переменным,
    ' объявленным на уровне модуля.
    Public Property Text() As String
        Get
            Text = mstrText
        End Get
        Set(ByVal Value As String)
            mstrText = Value
            ' Сохранить копию исходной строки для Restore.
            mstrOriginal = Value
        End Set
    End Property

    ' Вернуть транслированный текст в исходный вид.
    Public Sub Restore()
        mstrText = mstrOriginal
    End Sub

    ' Транслировать значение свойства Text.
    Public Sub Translate()
        Dim strWord As String, intCount As Integer
        Dim arrWords() As String
        Dim bCaps As Boolean
```

```
        ' Преобразовать строку в массив при помощи System.String.
        arrWords = mstrText.Split(" ")
        For intCount = 0 To UBound(arrWords)
            ' Проверить, начинается ли слово с заглавной
            ' буквы.
            If LCase(arrWords(intCount)) <> _
            arrWords(intCount) Then
                bCaps = True
                arrWords(intCount) = LCase(arrWords(intCount))
            End If
            strWord = arrWords(intCount)
            ' Выполнить трансляцию.
            If strWord <> "" Then
                strWord = Right(strWord, Len(strWord) - 1) & _
                  Left(strWord, 1) & "ay"
                ' Сделать первую букву заглавной, если нужно.
                If bCaps Then
                    strWord = UCase(Left(strWord, 1)) & _
                      Right(strWord, Len(strWord) - 1)
                End If
            End If
            ' Записать слово обратно в массив.
            arrWords(intCount) = strWord
            ' Сбросить флаг заглавной буквы,
            bCaps = False
        Next
        ' Собрать строку из массива слов.
        mstrText = String.Join(" ", arrWords)
    End Sub

End Class
```

Visual C#

```
internal class TranslatorClass
{

    string mstrText, mstrOriginal;

    // Этот код управляет доступом к переменным, объявленным
     // на уровне класса.
    public string Text
    {
        get
        {
            return mstrText;
        }
        set
```

```
        {
            mstrText - value;
                // Сохранить копию исходной строки для Restore,
            mstrOriginal = value;
}
    }

    // Вернуть транслированный текст в исходный вид.
    public void Restore()
    {
        mstrText - mstrOriginal;
    }

    // Транслировать значение свойства Text.
    public void Translate()
    {
        string strWord;
        string[] arrWords;
        bool bCaps - false;
        // Преобразовать строку в массив при помощи
        // System.String.
        arrWords - mstrText.Split(' ');
        for(int intCount - 0; intCount <= arrWords.GetUpperBound(0);
intCount++)
        {
            // Преобразовать в нижний регистр.
            strWord = arrWords[intCount].ToLower();
            // Если первая буква - заглавная.
            if(!arrWords[intCount].Equals(strWord))
                bCaps = true:
               // Выполнить трансляцию.
            if(strWord != "")
            {
                strWord = strWord.Substring(1,strWord.Length - 1) +
                    strWord.Substring(0, 1) + "ay";
                // Сделать первую букву заглавной, если
                // нужно,
                if(bCaps)
                    strWord = strWord.Substring(0, 1).ToUpper() +
                      strWord.Substring(1, strWord.Length - 1);
            }
            // Записать слово обратно в массив.
            arrWords[intCount] = strWord;
            // Сбросить флаг заглавной буквы.
            bCaps - false;
        }
        // Собрать строку из элементов массива.
```

```
            mstrText = String.Join("", arrWords);
        }
    }
```

Упражнение 3. Сохранение объекта Translator в переменной состояния Session



Сейчас вы добавите в файл Global.asax код, инициализирующий переменную состояния *Session*. Переменная состояния *Session* предназначена для хранения объекта *TranslatorClass*, используемого Web-формой.

► **Создание переменной состояния Session**

1. Дважды щелкните файл Global.asax в окне Project, затем щелкните ссылку «Click Here To Switch To Code View».
2. Чтобы создать обработчик события *Session _Start*, введите в окне Code View следующий код:

Visual Basic .NET

```
Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Создать новый объект TranslatorClass.
    Dim TransClass As New TranslatorClass()
    ' Сохранить объект в переменной состояния Session.
    Session("TransClass") = TransClass
End Sub
```

Visual C#

```
protected void Session_Start(Object sender, EventArgs e)
{
    // Создать новый объект класса TranslatorClass.
    TranslatorClass TransClass - new TranslatorClass0;
    // Сохранить объект в переменной состояния Session.
    Session["TransClass"] = TransClass;
}
```

Упражнение 4. Обработка событий Web-формы с помощью объекта TranslatorClass



Сейчас вы добавите к Web-форме код, который позволит ей использовать объект *TransClass*, сохраненный в переменной состояния *Session*. Кроме того, вы примените переменную типа *Boolean*, хранимую в состоянии отображения страницы, для переключения режимов *Translate* и *Restore* кнопки.

► **Как задействовать объект *TransClass* в обработчиках событий Web-формы**

1. Откройте в окне Code модуль кода Web-формы.
2. Объявите на уровне модуля переменную-объект для получения объекта *TransClass* из переменной состояния *Session*.

Visual Basic .NET

```
Dim TransClass As TranslatorClass
```

Visual C#

```
TranslatorClass  TransClass;
```

3. Чтобы присвоить значение переменной состояния *Session* только что объявленной переменной *TransClass*, добавьте в обработчик события *Page_Load* следующий код:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
        ' Получить объект TransClass из переменной состояния Session
        TransClass = Session("TransClass")
    End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
    {
        // Убедиться, что переменная состояния Session существует.
        if(Session["TransClass"] != null)
            // Получить объект TransClass из переменной состояния Session,
            TransClass = (TranslatorClass)Session["TransClass"];
    }
```

4. Добавьте в обработчик события *butTranslate_Click* следующий код:

Visual Basic .NET

```
Private Sub butTranslate_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles butTranslate.Click
    ' Объявить переменную-"переключатель" типа Boolean.
    Dim bSwitch As Boolean
    ' Получить значение "переключателя" из ViewState и инвертировать его.
    bSwitch = Not Viewstate("bSwitch")
    ' Сохранить новое значение в ViewState.
    ViewState("bSwitch") = bSwitch
    ' Транслировать текст, введенный в текстовом поле TextBox1, или зернуть
    ' его в исходный вид, в зависимости от значения "переключателя".
    If bSwitch Then
        ' Получить текст.
        TransClass.Text = txtSource.Text
        ' Транслировать текст.
        TransClass.Translate()
        ' Показать текст.
        txtSource.Text = TransClass.Text
        ' Поменять надпись на кнопке.
        butTranslate.Text = "Restore"
    Else
```

```
        ' Вернуть текст в исходный вид.
        TransClass.Restore()
        ' Показать текст,
        txtSource.Text = TransClass.Text
        ' Поменять надпись на кнопке.
        butTranslate.Text = "Translate"
    End If
End Sub
```

Visual C#

```
private void butTranslate_Click(object sender, System.EventArgs e)
{
    // Объявить переменную-"переключатель" типа Boolean.
    bool bSwitch;
    // Убедится, что переменная ViewState существует.
    if(ViewState["bSwitch"] != null)
        // Получить значение "переключателя" из ViewState и
        // инвертировать его.
        bSwitch = ! (bool)ViewState["bSwitch"];
    else
        // Задать значение "переключателя".
        bSwitch = true;
    // Сохранить новое значение в ViewState.
    ViewState["bSwitch"] = bSwitch;
    // Транслировать текст, введенный в текстовом поле TextBox1, или вернуть
    // его висходный вид, в зависимости от значения "переключателя".
    if (bSwitch)
    {
        // Получить текст.
        TransClass.Text = txtSource.Text;
        // Транслировать текст.
        TransClass.Translate();
        // Показать текст.
        txtSource.Text = TransClass.Text;
        // Поменять надпись на кнопке.
        butTranslate.Text = "Restore";
    }
    else
    {
        // Вернуть текст в исходный вид.
        TransClass.Restore();
        // Показать текст.
        txtSource.Text = TransClass.Text;
        // Поменять надпись на кнопке.
        butTranslate.Text = "Translate";
    }
}
```

5. Запустите приложение и проверьте его работу.

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. Каким образом упорядочены классы .NET Framework?

2. Чем различаются модули класса и кода в Visual Basic .NET?

3. Как в Visual C# объявить метол, чтобы он был доступен без предварительного создания экземпляра этого класса?

4. Как вызвать член базового класса из производного класса?

5. Назовите четыре основных объекта, используемых при разработке Web-приложений.

6. Куда следует поместить следующие элементы данных: элемент управления, создаваемый во время выполнения; объект, предоставляющий сервисы всем пользователям приложения; личные настройки пользователя, чтобы сохранить их между запросами к Web-форме?

ГЛАВА 4

Создание пользовательского интерфейса

Занятие 1. Использование элементов управления 126
Занятие 2. Проверка вводимых данных 156
Занятие 3. Переход между формами 164

В этой главе

В этой главе рассказано, как конструировать пользовательский интерфейс для многостраничных приложений Web Forms с помощью элементов управления **ASP.NET,** выбирать элементы управления, подходящие для решения различных задач, а также описаны различные виды разметки Web-форм. Кроме того, вы научитесь проверять правильность данных, введенных в элементы управления, и программировать переходы между Web-формами.

Прежде **всего**

Для изучения материалов этой главы вам необходимо:

- **знать основные** элементы **HTML;**
- понимать жизненный цикл Web-форм (он обсуждается во 2 главе).

Занятие 1. Использование элементов управления

Элементы управления размещаются на Web-формах, они позволяют Web-формам решать задачи, для которых те предназначены, а также определяют внешний вид формы, позволяют получать данные и выполняют различные действия от имени пользователя,

В Microsoft Visual Studio .NET предусмотрено два типа элементов управления, которые применяются с Web-формами, и два вида разметки самих Web-форм, с ними вы познакомитесь, изучив материал этого занятия. Кроме того, вы узнаете, какие элементы управления необходимы для решения основных задач программирования.

Изучив материал этого занятия, вы сможете:

- ✓ выбирать разметку Web-формы в зависимости от типа создаваемого приложения;
- ✓ объяснять различия между серверными и HTML-элементами управления, а также выбирать тип элементов управления для решения конкретной задачи;
- ✓ осуществлять привязку элементов управления к элементам данных, например к переменным в приложении;
- ✓ создавать шаблоны для сложных элементов управления, состоящих из повторяющихся наборов других элементов управления;
- ✓ писать код для обработки событий элементов управления, инициированных пользователем;
- ✓ получать файлы от клиента и сохранять их на сервере.

Продолжительность занятия — около 60 минут.

Выбор способа размещения элементов управления

Разметка элементов управления на Web-форме бывает двух видов:

- сеточная (grid layout) — задана по умолчанию; элементы управления располагаются в точности там, куда вы их помещаете, и имеют абсолютное положение на странице. Такую разметку следует применять с приложениями в стиле Windows, когда элементы управления не перемежаются длинными текстовыми фрагментами;
- **потоковая** (flow **layout)** — положение элементов управления определяется относительно других элементов управления, размещенных на странице ранее. Новые элементы управления, добавляемые во время выполнения, размещаются ниже тех, что добавлены перед этим. Такой способ размещения подходит для форм, где элементы управления чередуются с текстом.

Чтобы указать способ разметки страницы, предусмотрен следующий порядок действий.

1. В окне Design выберите нужную Web-форму.
2. Выберите объект *DOCUMENT* в окне Properties.
3. Установите для свойства *pageLayout* объекта *DOCUMENT* значение *FlowLayout* либо *GridLayout.*

Результат присваивания *pageLayout* различных значений показан на рис. 4-1. Значение *GridLayout* применяют для Web-форм с постоянным внешним видом, а *FlowLayout* — для Web-форм, которые содержат помимо элементов управления много текста,

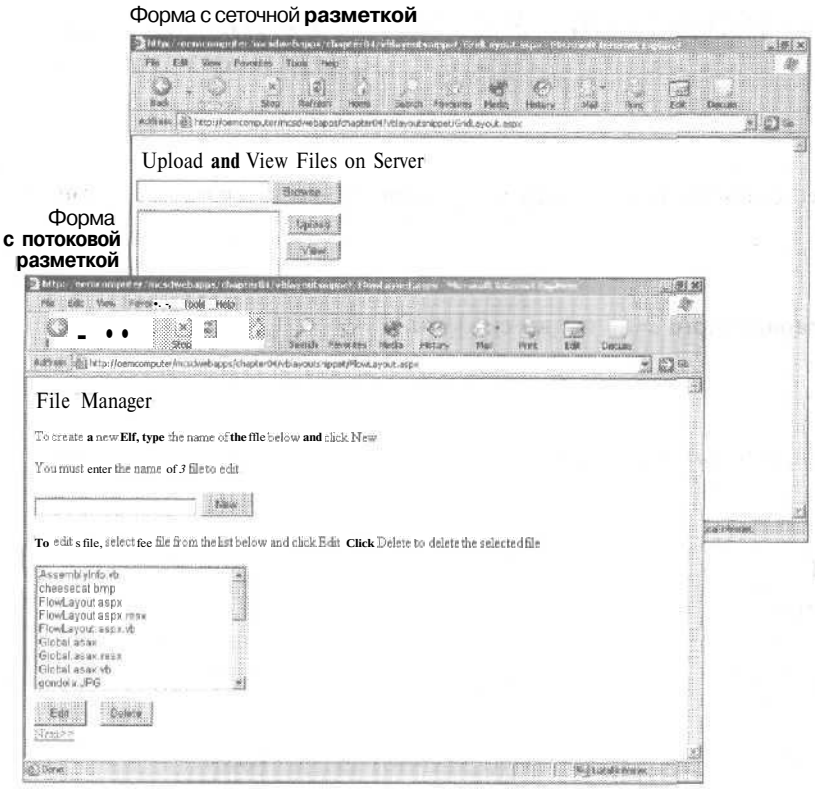


Рис. 4-1. Разные виды разметки Web-форм

Если значение свойства ***pageLayout*** **равно** *GridLayout*, **то при создании каждого элемента управления Visual Studio добавляет к нему атрибуты стиля, которые задают его положение на странице** (показаны **полужирным шрифтом в** следующем **определении формы):**

```
<form action="webform1.aspx" method="post"
    enctype="multipart/form-data" runat="server" ID="Form1">
<h2>
Upload and View Files on Server
</h2>
<INPUT id="filUpload" style="Z-INDEX: 101; LEFT: 86px; WIDTH: 347px;
    POSITION: absolute; TOP: 20px; HEIGHT: 27px" runat="server"
    type="file" size="38">
    <asp:Button id="butUpload" style="Z-INDEX: 102; LEFT: 357px;
     POSITION: absolute; TOP; 60px" runat="server" Text="Upload"
     Width="74px" Height="31px"></asp:Button>
    <asp:ListBox id="lstServerFiles" style="Z-INDEX: 103; LEFT: 86px;
     POSITION: absolute; TOP: 79px" runat="server" Height="140px"
     Width="168px"></asp:ListBox>
    <asp:Button id="butView" style="Z-INDEX: 104; LEFT: 268px;
     POSITION: absolute; TOP: 178px" runat="server" Height="31px"
     Width="76px" Text="View"></asp:Button>
```

```
    <asp:Image id="imgView" style="Z-INDEX: 106; LEFT: 441px;
        POSITION: absolute; TOP: 24px" runat="server"
        Height="132px" Width="162px"></asp:Image>
</form>
```

Если же значение свойства *pageLayout* равно *FlowLayout*, Visual Studio не добавляет атрибуты стиля. Элементы управления можно создавать не только перетаскиванием их на форму, но и дважды щелкнув изображение нужного элемента на панели Toolbox. Использование потоковой разметки упрощает создание форм, на которых элементы управления чередуются с текстом, особенно если потом эту форму приходится редактировать в виде HTML-текста:

```
<form id="Form1" method="post" runat="server">
    <h2>
        File Manager
    </h2>
    <P>
    To create a new file, type the name of the file below and
    click New.
    </p>
    <asp:Literal id="litNoFile" runat="server"></asp:Literal>
    <asp:TextBox id="txtNewFile" runat="server" width="191px"> </asp:TextBox>
    <asp:Button id="butNew" runat="server" Width="62px"
    Height="29px" Text="New"></asp:Button>
    <p>
        To edit a file, select the file from the list below and click Edit.
        Click Delete to delete the selected file.
    </P>
    <asp:Literal id="litNoneSelected" runat="server">
    <asp:Literal>
    <asp:ListBox id="lstFiles" runat="server" Width="252px"
        Height="161px"></asp:ListBox>
    <br>
    <asp:Button id="butEdit" runat="server" Width="62"
    Height="29"
        Text="Edit"></asp:Button>
      
    <asp:Button id="butDelete" runat="server" Width="62px"
Height="29px"
        Text="Delete"></asp:Button>
</form>
```

Выбор подходящего элемента управления

С Web-формами применяются как HTML-элементы управления, так и серверные элементы управления. В чем разница между ними? В сущности, серверные элементы управления — это надмножество HTML-элементов управления, их особенности перечислены в таблице 4-1.

Таблица 4-1. Отличия серверных и HTML-элементов управления

Предмет рассмотрения	Серверные элементы управления	HTML-элементы управления
События на сервере	Генерируют на сервере события, специфичные для элемента управления	Могут генерировать на сервере только страничные события (инициирующие отправку данных на сервер)
Управление состоянием	Данные, введенные в элемент управления, сохраняются между запросами	Введенные в них данные не сохраняются; их необходимо сохранять при помощи страничных сценариев
Приспособление к возможностям браузера	Автоматически распознают браузер и выполняют рендеринг страницы в зависи мости от его возможностей	Не способны автоматически приспосабливаться к возможностям браузера. В общем случае при использовании HTML-элементов управления нужен код, определяющий браузер, либо при написании кода следует исходить из минимальных возможностей браузера
Свойства	Для каждого элемента управления .NET Framework поддерживает набор свойств, позволяющих изменять внешний вид элемента управления и его поведение в серверном коде	Поддерживаюттолько атрибуты HTML

Но зачем нужен второй набор элементов управления со сходной функциональностью? Всем стандартным элементам HTML соответствуют HTML-элементы управления, которые позволяют в большей степени контролировать поведение объектов на странице. Причины применения HTML-элементов управления перечислены далее:

- **переход с более ранних версий ASP.** Прежние версии ASP поддерживали только элементы HTML; при использовании Visual Studio можно не переписывать старые ASP- приложения «с нуля», а постепенно обновить их ло ASP.NET. При загрузке проекта в Visual Studio .NET элементы HTML преобразуются в HTML-элементы управления;
- не **все** элементы **управления требуют серверной обработки и управления состоянием,** особенно при использовании привязки данных. При каждом запросе к серверу элементы управления, связанные с данными, обновляются значениями из источника данных. Поэтому в таких случаях лучше применять HTML-элементы управления, а если используются серверные элементы управления, связанные с данными, — отключать для них управление состоянием;
- **элементы управления HTML дают полный контроль над отображением. ASP.NET** автоматически настраивает внешний вид серверных элементов управления в зависимости от возможностей запрашивающего браузера, а HTML-элементы управления не поддерживают автоматическую настройку, поэтому ими управляют напрямую.

Как видите, серверные и HTML-элементы управления в большинстве случаев обладают сходной функциональностью. Как правило, с серверными элементами управления работать проще. В таблице 4-2 перечислены серверные и HTML-элементы управления, сгруппированные по видам задач программирования, которые они позволяют решать.

Таблица 4-2. Серверные и HTML-элементы управления

Задача	Серверные элементы управления	HTML-элементы управления
Отображение текста	Label, TextBox, Literal	Label, Text Field, Text Area, Password Field
Отображение таблиц	Table, DataGrid	Table
Выбор значения из списка	DropDownList, ListBox, DataList, Repeater	List Box, Dropdown
Исполнение команд	Button, LinkButton, ImageButton	Button, Reset Button, Submit Button
Установка значений	CheckBox, CheckBoxList, RadioButton, RadioButtonList	Checkbox, Radio Button
Отображение графики	Image, ImageButton	Image
Переходы	Hyperlink	Нет (для этого используется HTML-тэг <a>)
Группировка элементов управления	Panel, Placeholder	Flow Layout, Grid Layout
Работа с датами	Calendar	Нет
Демонстрация рекламных изображений	AdRotator	Нет
Отображение горизонтальной линии	Literal	Horizontal Rule
Получение файлов от клиентов	Нет	File Field
Сохранение данных, введенных на странице	Обеспечивается механизмом управления состоянием	Input Hidden
Проверка вводимых данных	RequiredFieldValidator, CompareValidator, RangeValidator, RegularExpressionVaidator, CustomValidator, Validationsummary	Нет (для этого используются страничные сценарии)

В следующих разделах рассматриваются основные задачи программирования, возникающие при создании пользовательского интерфейса, и решаемые посредством элементов управления.

Работа с текстом

Существует много способов отображения текста на странице. Текст, доступный только для чтения, можно записать прямо и объект *Response,* скажем, так: Response.Write («Some text»). Кроме того, для этого можно воспользоваться элементом управления Label или TextBox со свойством *ReadOnly*, установленным в *True;* наконец, для отображения текста применяют элемент управления Literal либо компонуют текст в виде HTML (как при использовании метода *Response. Write*).

Для отображения текста, доступного для редактирования, служит элемент управления TextBox, ключевые свойства которого перечислены в таблице 4-3.

Таблица 4-3. Свойства элемента управления TextBox

Имя	Назначение
Text	Получение и установка значения текстового поля
TextMode	Отображение текста в одну (*SingleLine*) или несколько строк (*MultiLine*) (с прокруткой), а также пароля *(Password)*. В последнем случае вместо вводимых символов отображаются точки
ReadOnly	Запрет на изменение отображаемого текста пользователем
AutoPostBack	Если это свойство установлено в *True,* то, когда пользователь покидает текстовое поле, изменив его содержимое, поле генерирует событие *TextChanged*, вы-зывающее отправку страницы на сервер. По умолчанию это свойство установлено в *False,* поэтому событие *TextChanged*кэшируется, пока не произойдет событие, вызывающее возврат страницы на сервер

Чтобы познакомиться с текстовыми полями на практике, мы создадим входную страницу Web-приложения.

1. Создайте новое Web-приложение с двумя Web-формами: Webform1 и Webform2.
2. Разместите на Webforml элементы управления Label, TextBox, Button и задайте их свойства, как показано в следующей таблице:

Элемент управления	Свойство	Значение
Label 1	Text	Username:
LabeI2	Text	Password:
Text Box 1	ID	txtUser
TextBox2	ID	txt Password
	Text Mode	Password
B Litton 1	ID	butOK
	Text	OK

3. Добавьте к обработчику события *butOK_Click* следующий код:

Visual Basic .NET

```
Private Sub butOK_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles butOK.Click
        If txtUser.Text = "Guest" And txtPassWord.Text = "Moondog" Then
            Response.Redirect("Webform2.aspx")
        Else
            txtPassword.Text - ""
        End If
    End Sub
```

Visual C#

```
private void butOK_Click(object sender, System.EventArgs e)
    {
        if ((txtUser.Text == "Guest") &&
              (txtPassword.Text == "Moondog"))
```

```
            Response.Redirect("Webform2.aspx");
        else
            txtPassword.Text = "";
    }
```

4. Запустите приложение. Если ввести имя пользователя и пароль, прописанные в коде, приложение покажет форму Webform2 (рис. 4-2). Заметьте, что в результате присвоения свойству *TextMode*этого поля значения *Password* вместо вводимых в него символов отображаются точки.

Естественно, хранить пароли и пользовательские имена в тексте программы не рекомендуется, в главе 8 рассказывается, как сохранять и извлекать пароли из защищенных серверных файлов.

Работа с таблицами и списками

Элементы управления Label и TextBox отображают текст единым блоком. Для отображения текста в виде списков и таблиц следует воспользоваться одним из элементов управления, перечисленных в таблице 4-4. Элементы управления ListBox, DropDownList и Table служат для отображения простых динамических таблиц и списков, а DataGrid, DataList и Repeater — для более сложных таблиц и списков, содержащих другие элементы управления и связанных с данными,

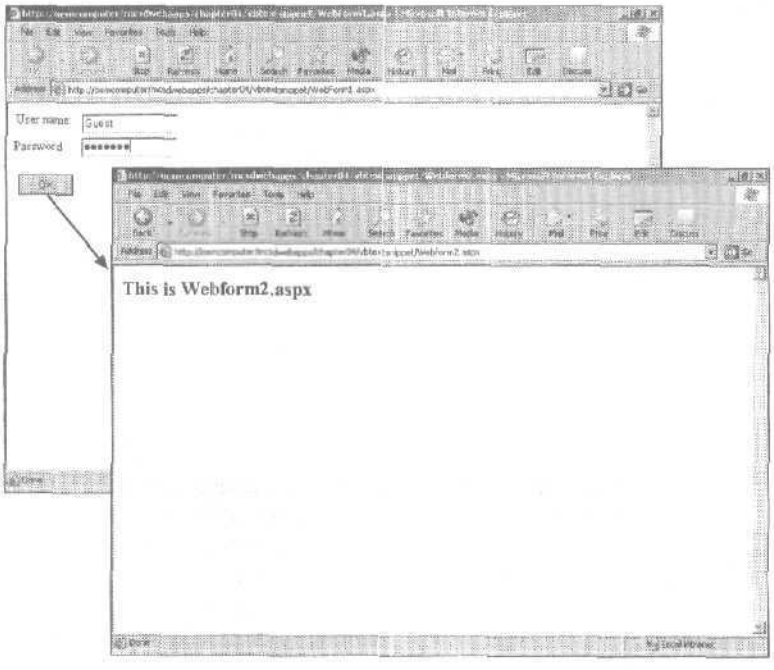


Рис. 4-2. Webform1 **и Webform2**

Таблица 4-4. Элементы управления, отображающие списки и таблицы

Имя	Назначение
ListBox	Отображение неизменяемого текста в виде простого списка с прокруткой
DropDownList	Отображение неизменяемого текста в виде простого раскрывающегося списка
Table	Отображение текста и элементов управления в виде таблицы. Позволяет динамически создавать таблицы в коде при помощи наборов *TableRows* и *TableCells*
DataGrid	Отображение текста и элементов управления в виде таблицы; внешний вид этого элемента контролируется шаблонами. DataGrid обладает встроенными возможностями форматирования, сортировки и постраничного просмчтра данных
DataList	Отображение текстовых строк и элементов управления с использованием шаблона, задающего их внешний вид. Обладает встроенными возможностями форматирования и выбора элементов списка
Repeater	Отображение строк, составленных из различных элементов управлении, внешний вид которых контролируется шаблоном. Встроенные возможности, присущие DataGrid и DataList, отсутствуют

Добавление элементов к спискам и таблицам во время разработки

К элементам управления ListBox, DropDownList и Table можно добавлять статические элементы данных на этапе разработки. Это делается средствами диалогового окна Collection Editor, показанного на рис. 4-3.

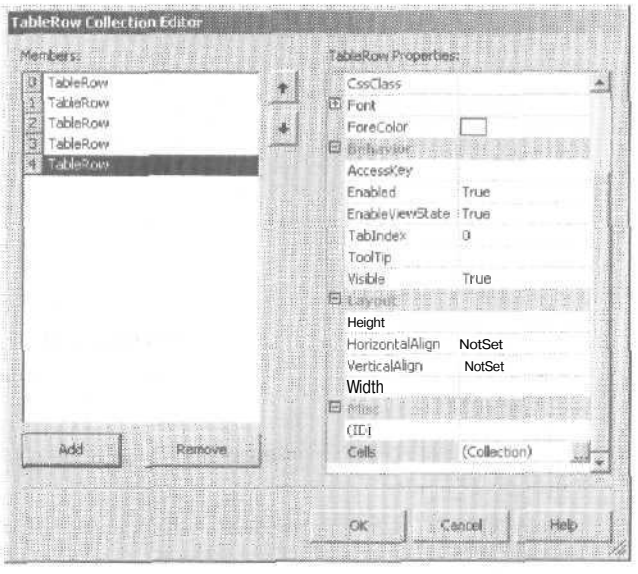


Рис. 4-3. Collection Editor

Для этого нужно выполнить одно из следующих действий:

- чтобы добавить постоянный элемент к ListBox или DropDownList, выберите в окне Properties свойство Items и щелкните кнопку рядом с надписью (Collection);
- чтобы добавить постоянный элемент к Table, выберите в окне Properties свойство Rows и щелкните кнопку рядом с надписью (Collection).

Добавление элементов в список или таблицу во время выполнения

Для добавления элементов к списку во время выполнения используется метод *Add* набора *Items,* принадлежащего элементу управления. Например, следующий код добавляет к элементам управления ListBox и DropDownList данные, введенные в текстовое поле:

Visual Basic .NET

```
Private Sub butAdd_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butAdd.Click
    ListBox1.Items.Add(txtSource.Text)
    DropDownList1.Items.Add(txtSource.Text)
End Sub
```

Visual C#

```
private void butAdd_Click(object sender,
     System.EventArgs e)
{
    ListBox1.Items.Add(txtSource.Text);
    DropDownList1.Items.Add(txtSource.Text);
}
```

И ListBox, и DropDownList автоматически сохраняют элементы, добавленные к ним во время выполнения. В отличие от них, элемент управления Table автоматически сохраняет только те данные, которые были записаны в ячейки во время разработки при помощи Collection Editor. Чтобы добавить строки или ячейки к Table во время выполнения, необходимо заново создать таблицу на основе данных, сохраненных в переменной состояния. Ниже показан код, отображающий в ячейках таблицы элементы, которые пользователь вводит в текстовое поле (через запятую). Каждый щелчок кнопки Add добавляет строку к таблице:

Visual Basic .NET

```
Private Sub butAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles butAdd.Click
        ' Записать текст в состояние отображения страницы.
        ViewState.Add(Viewstate.Count, txtSource.Text)
        ' Построить таблицу заново.
        RebuildTable()
    End Sub

Private Sub RebuildTableO
    Dim iCount1, iCount2 As Integer
    Dim arrWords As String()
    Dim strWords As String
```

```
    ' Для каждой строки, сохраненной в состоянии отображения...
    For iCount1 = 0 To ViewState.Count - 1
        ' создать новую строку таблицы
        Dim rowNew As New TableRow()
        ' взять строку из ViewState
        strWords - ViewState(iCount1)
        ' разбить строку и поместить в массив
        arrWords = Split(strWords, ",")
        ' Для каждого элемента массива...
        For iCount2 = 0 To UBound(arrWords)
            ' создать новую ячейку
            Dim celNew As New TableCell()
            ' записать в ячейку текст
            celNew.Text - arrWords(iCount2)
            ' ...добавить ячейку к строке.
            rowNew.Cells.Add(celNew)
        Next
        ' ...добавить строку к таблице.
        Table1.Rows.Add(rowNew)
    Next
End Sub
```

Visual C#

```
private void butAdd_Click(object sender,
    System.EventArgs e)
{
    ListBox1.Items.Add(txtSource.Text);
    DropDownList1.Items.Add(txtSource.Text);
    // Записать текст в состояние отображения страницы,
    ViewState.Add(ViewState.Count.ToString(), txtSource.Text);
    RebuildTable();
}


private void RebuildTable()
{
    string[] arrWords;
    string strWords;
    TableRow rowNew;
    TableCell celNew;
    // Для каждой строки, сохраненной в состоянии отражения...
    for (int iCount1 = 0; iCount1 < ViewState.Count;
        iCount1++)
{
    char[] strSep = {','};
    // Create a new table row.
    rowNew = new TableRow();
```

```
        // взять строку из ViewState.
        strWords = ViewState[iCount1.ToString()].ToString();
        // разбить строку и поместить результаты в массив.
        arrWords = strWords.Split(strSep);
        // Для каждого элемента массива...
        for (int iCount2 = 0; iCount2 <= arrWords.GetUpperBound(0); iCount2++)
        {
           // создать новую ячейку,
           celNew = new TableCell();
           // записать текст в ячейку,
           celNew.Text = arrWords[iCount2];
           // добавить ячейку к строке таблицы.
           rowNew.Cells.Add(celNew);
        }
        // ...добавить строку к таблице.
        Table1.Rows.Add(rowNew);
    }
}
```

Получение выбранного элемента списка

Для **получения** текущего **выбранного** элемента списка используется его **свойство** ***SelectedItem***. Например, следующий **код отображает выбранный элемент** списка в **надписи на поверхности Web-формы:**

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Проверить, выбран ли какой-нибудь элемент списка.
    If Not IsNothing(ListBox1.SelectedItem) Then
        ' Отобразить выбранный элемент.
        Labell.Text = "The selected item is: " _
          & ListBox1.SelectedItem.Text
    Else
        Labell.Text = "No item is selected."
    End If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Проверить, выбран ли какой-нибудь элемент списка.
    if (ListBox1.SelectedItem == null)
        Labell.Text = "No item is selected.";
    else
        // Отобразить выбранный элемент.
```

```
        Label1.Text = "The selected item is: " +
            ListBox1.SelectedItem.Text;
}
```

Свойство *SelectedItem* возвращает объект, поэтому перед использованием значения этого свойства непременно убедитесь, что он существует.

Использование простого связывания данных со списками

В приложениях значения элементов управления можно получать из любого источника данных. Таким источником могут быть любые открытые данные — таблицы базы данных, массивы, свойства объектов или выражения, объединяющие несколько элементов. В простейшем варианте привязка данных позволяет легко инициализировать значения элементов управления ListBox и DropDownList, расположенных на Web-формах.

Чтобы увидеть, как работает привязка данных, выполните следующие действия.

1. Создайте Web-форму и добавьте к ней элемент управления DropDownList и следующий код:

Visual Basic .NET

```
' Открытый элемент данных для DropDownList.
Public arrData As String() = {"This", "that", "and", _
  "the", "other"}

Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Связать с данными элементы управления на этой странице.
    Page.DataBind()
End Sub
```

Visual C#

```
public string[] arrData= {"This", "that", "and",
                          "the", "other"};

private void Page_Load(object sender, System.EventArgs e)
{
    Page.DataBind()
}
```

2. Выберите элемент управления DropDownList и щелкните в окне Properties кнопку свойства *DataBinding* — Visual Studio откроет диалоговое окно DataBindings, показанное на рис. 4-4. Оно служит для загрузки данных из любых открытых источников в элементы управления, отображающие списки и таблицы.
3. Из списка Bindable Properties выберите свойство *DataSource*, установите переключатель в позицию Custom Binding Expression и введите в текстовое поле для выражений с гроку arrData. После запуска приложения в раскрывающемся списке DropDownList отобразятся элементы массива *arrData* (рис. 4-5).

При связывании данных с серверным элементом управления отключите для него управление состоянием. Это позволит повысить производительность, поскольку метод *DataBind* заменяет автоматическое управление состоянием отображения ASP.NET.

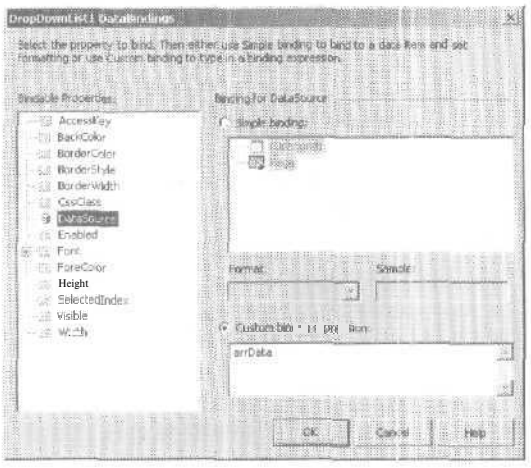


Рис. 4-4. Диалоговое окно DataBindings

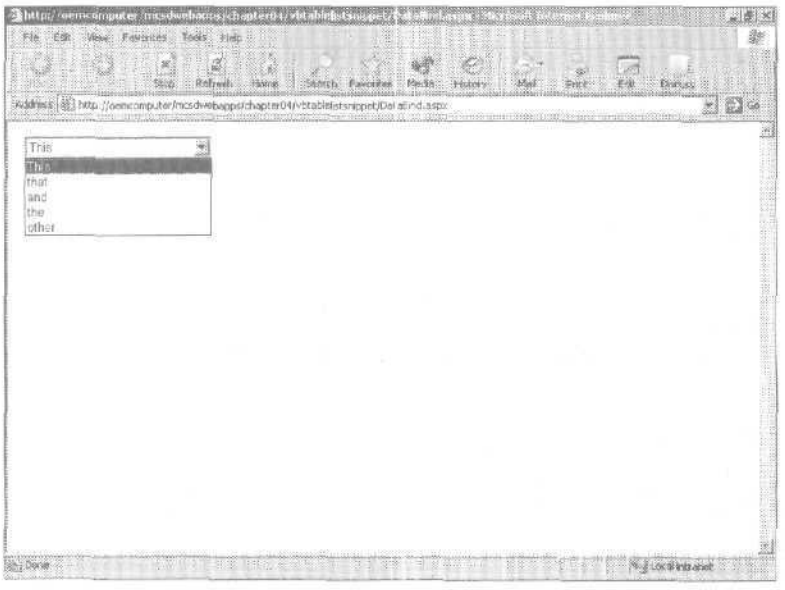


Рис. 4-5. Привязка списка к массиву

Чтобы отключить управление состоянием для серверного элемента управления, установите его свойство *EnableViewState* в *False*.

Добавление элементов к DataGrid, DataList и Repeater

Для добавления элементов к DataGrid, DataList и Repeater используется привязка данных. Внешний вид этих элементов управления во время выполнения определяется шаблонами. *Шаблон* (template) — это набор серверных или HTML-элементов элементов управления (либо тех и других вместе), составляющих повторяющиеся элементы данных в DataGrid, DataList или Repeater.

Чтобы добавить элемент данных к DataGrid, DataList или Repeater, выполните с. [едующие действия.

1. Определите источник данных.
2. Поместите на форму элемент управления DataGrid, DataList или Repeater и свяжите его с источником данных.
3. Задайте повторяющийся набор серверных или HTML-элементов управления, отредактировав шаблон элемента управления.
4. Установите свойства серверных элементов управления, содержащихся в списке или таблице, чтобы привязать их к источнику данных, связанному с их контейнером.

В следующем примере показано, как добавить к элементу управления DataGrid столбцы шаблона и привязать элементы управления из этих столбцов к простым источникам данных.

1. Создайте в приложении открытый источник данных, как это делалось в предыдущих примерах. Код, показанный ниже, создает массив для демонстрации использования простого источника данных:

Visual Basic .NET

```
' Открытый элемент данных для DataGrid.
    Public arrData As String() = {"This", "that", "and", _
      "the", "other"}
```

Visual C#

```
// Открытый элемент данных для DataGrid.
    public string[] arrData - new string[] {"This", "that",
        "and", "the", "other"};
```

2. Поместите на поверхность Web-формы элемент управления DataGrid.
3. Добавьте к DataGrid столбцы шаблона. Для этого выберите его и щелкните ссылку Property Builder, расположенную пол списком свойств в окне Properties, — Visual Studio откроет диалоговое окно Properties (рис. 4-6).

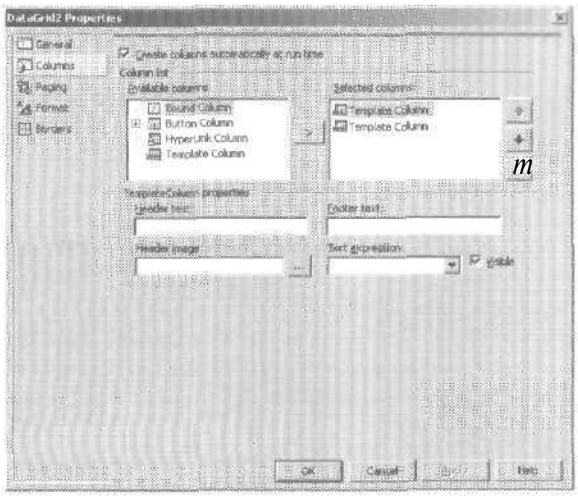
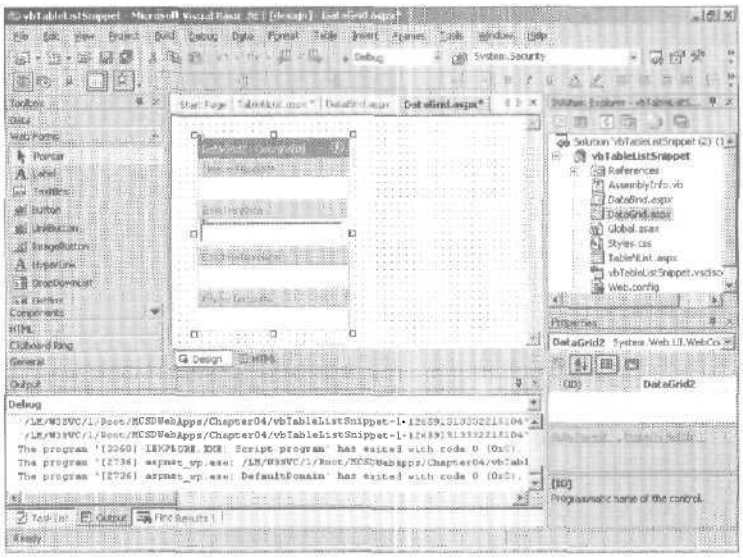


Рис. 4-6. Диалоговое окно Properties

4. Чтобы добавить столбец шаблона к элементу управления DataGrid, в диалоговом окне Properties щелкните Columns, выберите в списке Columns пункт Template Column и щелкните кнопку Add (>), В этом примере вы должны добавить два столбца шаблона. Щелкните OK.
5. В окне Properties выберите свойство *DataSource* и задайте источник данных для DataGrid. В нашем примере нужно ввести **arrData**, т. е. имя массива, созданного на этапе 1.
6. Создайте шаблон, который будет использован для отображения данных в DataGrid. Для этого щелкните DataGrid правой кнопкой и выберите из контекстного меню команду Edit Template\Column(0) — элемент управления перейдет в режим редактирования, как показано на рис. 4-7.



Рис, 4-7. Добавление элементов управления к DataGrid

7. Чтобы добавить к DataGrid элементы управления, поместите их на Web-форму и перетащите на шаблон. В этом примере нужно добавить текстовое поле и перетащить его на столбец 0.
8. В окне Properties выберите свойство *DataBindings* для элемента управления, только что добавленного к шаблону, и щелкните появившуюся кнопку, В результате Visual Studio покажет диалоговое окно DataBinding (рис. 4-8).
9. В списке BindableProperties выберите свойство, которое станет получателем данных. В нашем случае ~ *Text.* Установите переключатель в положение Simple Binding. Щелкните Container, затем — DataItem, чтобы задать элемент данных, помещаемый в указанное свойство. Щелкните OK, чтобы закрыть диалоговое окно DataBinding.
10. Отредактируйте второй столбец шаблона. Для этого щелкните правой кнопкой элемент управления DataGrid и выберите из контекстного меню команду Edit Template\Column(l).
11. Повторите этапы с 4 по 7, чтобы отредактировать столбец Column(1). Поместите на форму элемент управления Button и перетащите его на столбец шаблона Column(1) в DataGrid.
12. Закончив, закройте шаблон. Для этого щелкните шаблон правой кнопкой и выберите из контекстного меню команду End Template Editing. После этого Visual Studio покажет

элементы управления, вложенные в DataGrid (рис. 4-9). Любое свойство вложенных элементов управления можно изменить, отредактировав шаблон, как показано выше.

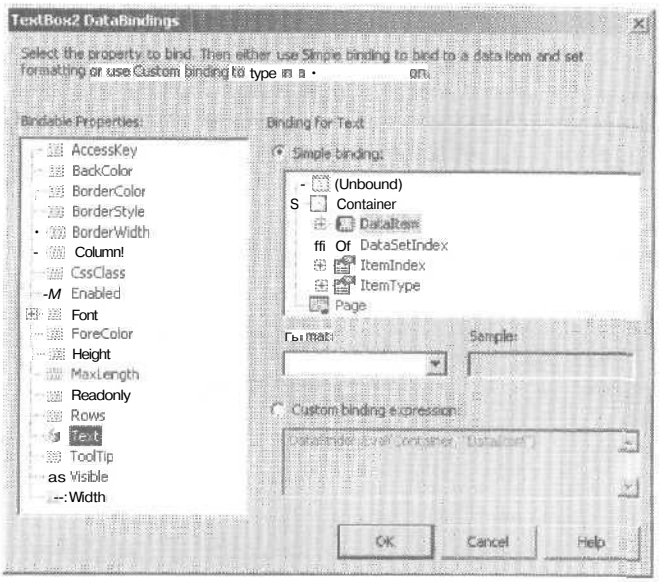


Рис. 4-8. Диалоговое окно DataBindings

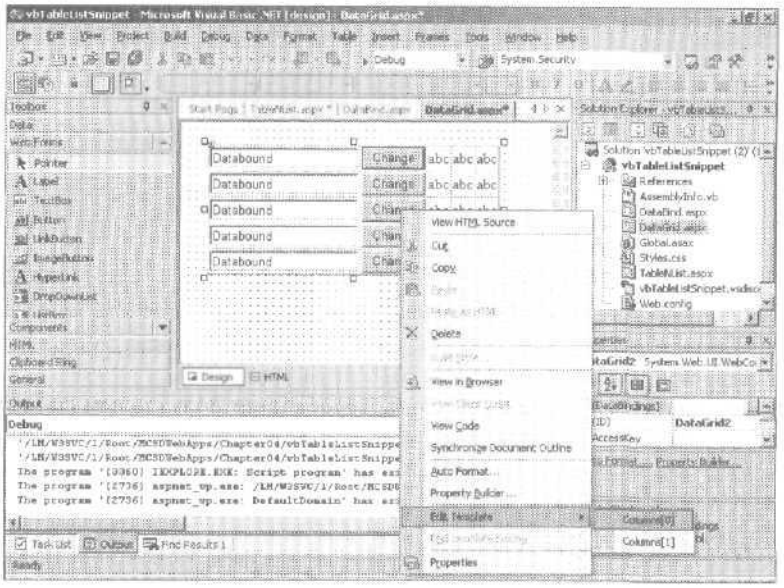


Рис. 4-9. Элементы управления, связанные с данными

Исполнение команд

Серверные элементы управления Button, LinkButton и ImageButton генерируют события, инициирующие возврат страницы на сервер (события postback) для исполнения команд. Такие события инициируют запрос браузера, который заставляет сервер обработать собы-

тия, **генерированные страницей. Обработчик события *Click* кнопки, щелчок которой отправил** страницу **на сервер, выполняется только после обработки всех событий, генерированных проверяющими элементами управления** (верификаторами), **а также кэшированных событий.**

Чтобы наглядно продемонстрировать порядок генерации событий на странице, поместите на Web-форму элементы управления TextBox, ListBox, Button и добавьте к ней следующий код:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load
        Response.Write("Page load.<br>")
' При первой загрузке страницы заполнить список,
If Not IsPostBack Then
ListBox1.Items.Add("This")
ListBox1.Items.Add("That")
ListBox1.Items.Add("The other")
End If
End Sub

Private Sub TextBox1_TextChanged(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles TextBox1.TextChanged
    Response.Write("Text changed.<br>")
End Sub

Private Sub ListBox1_SelectedIndexChanged(ByVal sender As _
System.Object, _
  ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged
    Response.Write("Item selected.<br>")
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button1.Click
    Response.Write("Button clicked.<br>")
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    Response.Write("Page load.<br>");
    //  При первой загрузке страницы заполнить список.
    if (!IsPostBack)
    {
        ListBox1.Items.Add("This");
        ListBox1.Items.Add("That");
        ListBox1.Items.Add("The other");
    }
```

```
}

private void TextBox1_TextChanged(object sender,
    System.EventArgs e)
{
    Response.Write("Text changed.<br>");
}

private void ListBox1_SelectedIndexChanged(object sender,
    System.EventArgs e)
{
    Response.Write("Item selected.<br>");
}
private void Button1_Click(object sender, System.EventArgs e)
{
    Response.Write("Button clicked.<br>");
}
```

Если запустить этот код, страница отобразит сведения о событиях в порядке их генерации (рис. 4-10). Обработчик события *Click* исполняется последним из всех обработчиков, генерированных страницей.

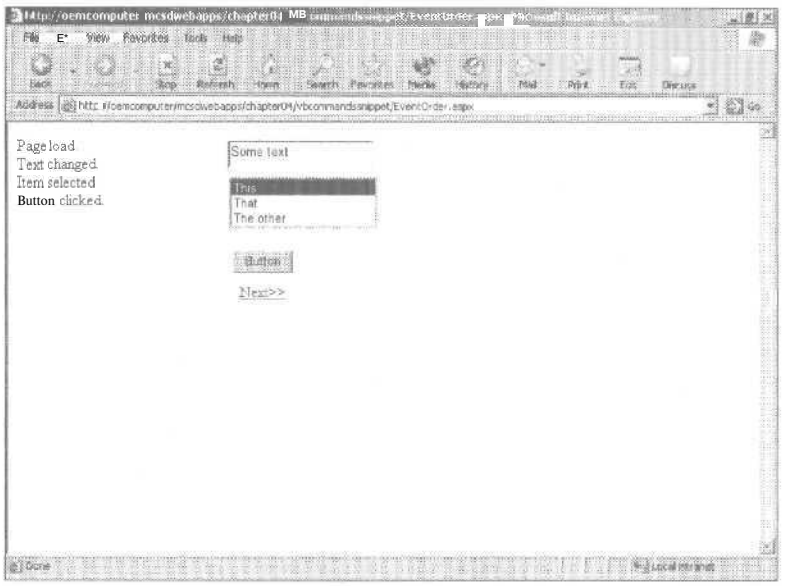


Рис. 4-10. Порядок генерации событий

Как видите, в использовании обработчиков события *Click* элементов управления Button и LinkButton нет ничего особого, В отличие от них элемент управления ImageButton предоставляет ряд дополнительных возможностей. Аргумент обработчика события *Click* этого элемента управления содержит координаты точки (X и Y), в которой пользователь щелкнул этот элемент управления. Реакция изображения зависит от того, где его щелкнул пользователь (рис. 4-11). Графический элемент, реагирующий на события подобным образом, называется *картой изображения* (image map).

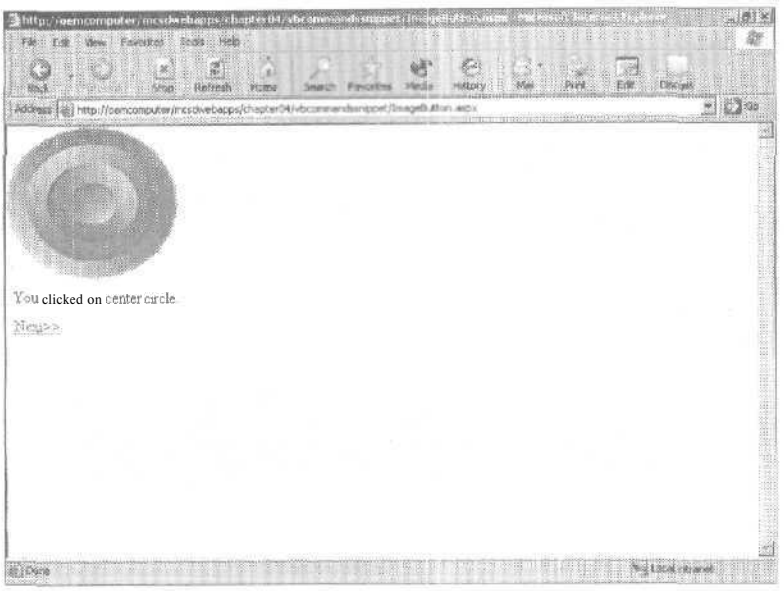


Рис. 4-11, Элемент управления ImageButton

Следующий код использует аргументы e.X и e.Y элемента управления ImageButton, что позволяет определить, по какому кругу кнопки (рис. 4-11) пришелся щелчок пользователя.

Visual Basic .NET

```
Private Sub ImageButton1_Click(ByVal sender As System.Object, _
    ByVal e As System.Web.UI.ImageClickEventArgs) Handles ImageButton1.Click
      Dim xOffset, yOffset, X, Y, Radius As Single
      Dim strMessage As String
      ' Вычислить расстояние от центра до точки щелчка на основе
      ' аргументов X и Y.
      xOffset - CSng(sender.width.value) / 2
      yOffset = CSng(sender.height.value) / 2
      X = System.Math.Abs(CSng(e.X) - xOffset)
      Y = System.Math.Abs(CSng(e.Y) - yOffset)
      Radius = System.Math.Sqrt((X ^ 2) + (Y ^ 2))
      ' Создать строку сообщения.
      Select Case CInt(Radius)
          Case 0 To 21
              strMessage = "on center circle."
          Case 22 To 42
              strMessage = "on second circle."
          Case 43 To 64
              strMessage - "on third circle."
          Case 65 To 86
              strMessage - "on fourth circle."
          Case Else
              strMessage - "outside of circle."
```

```
        End Select
        ' Показать сообщение.
        Label1.Text = "You clicked " & strMessage
End Sub
```

Visual C#

```
private void ImageButton1_Click(System.Object sender,
        System.Web.UI.ImageClickEventArgs e)
{
    double xOffset, yOffset, X, Y, Radius:
    string strMessage;
// Вычислить расстояние от центра до точки щелчка на основе аргументов X и Y.
    xOffset = Convert.ToDouble(ImageButton1.Width.Value) / 2;
    yOffset = Convert.ToDouble(ImageButton1.Height.Value) / 2;
    X = Math.Abs(Convert.ToDouble(e.X) - xOffset);
    Y = Math.Abs(Convert.ToDouble(e.Y) - yOffset);
    Radius = Math.Sqrt(Math.Pow(X, 2) + Math.Pow(Y, 2));
// Создать строку сообщения,
    if (Radius < 22)
    {
        strMessage - "on center circle.";
    }
    else if ((Radius >= 22) && (Radius <= 42))
    {
        strMessage = "on second circle.";
    }
    else if ((Radius > 42) && (Radius <= 64))
    {
        strMessage = "on third circle.";
    }
    eise if ((Radius > 64) && (Radius <= 86))
    {
        strMessage = "on fourth circle.";
    }
    else
    {
        strMessage - "outside of circle.";
    }
    // Показать сообщение,
    Labell.Text - "You clicked " + strMessage;
}
```

Получение и установка значений

Для получения от пользователя **булевых значений применяются** элементы **управления RadioButton,** RadioButtonList, CheckBox **и** CheckBoxList. **Подобно** ListBox **и DropDownList, элементы** управления **RadioButtonList и CheckBoxList** можно **заполнить** элементами при

помощи инструмента Collection Editor. Для этого щелкните у выбранного элемента управления свойство *Items,* чтобы вызвать Collection Editor.

Для получения значений элементов управления CheckBox и RadioButton используется их свойство *Checked.* Например, следующий код проверяет, установлен ли флажок CheckBox 1:

Visual Basic .NET

```
Private Sub Button1_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles Button1.Click
    Response.Write("Checkbox1 is " & CheckBox1.Checked.ToString)
End Sub
```

Visual C#

```
private void Button1_Click(object sender,
             System.EventArgs e)
{
    Response.Write("Checkbox1 is " + CheckBox1.Checked.ToString());
}
```

В отличие от элементов управления OptionButton из Windows-форм, элемент RadioButton не может автоматически взаимодействовать с аналогичными элементами управления, размещенными на той же форме. Чтобы элементы управления RadioButton взаимодействовали, задайте для каждого из них одинаковое значение свойства *GroupName.* На рис. 4-12 показаны три элемента управления RadioButton с одинаковыми значениями свойства *GroupName.* В результате при установке одного из этих переключателей остальные сбрасываются.

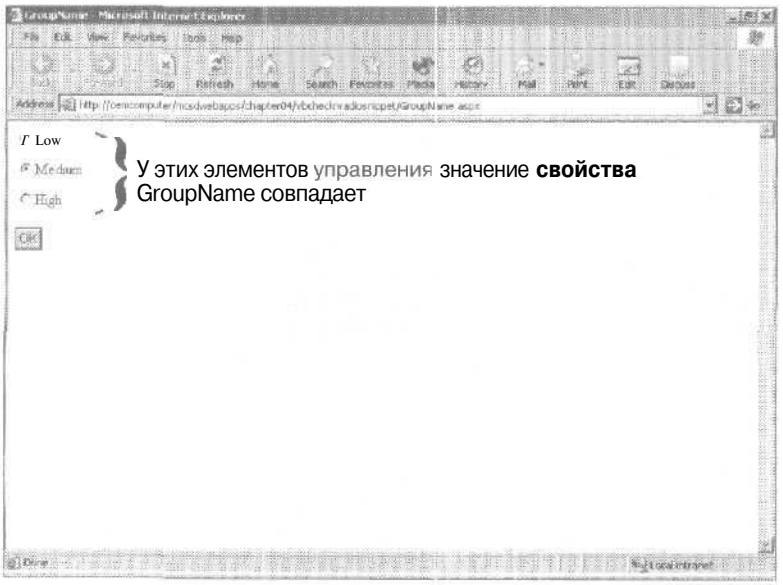


Рис. 4-12. Сгруппированные элементы управления RadioButton

Для получения и установки значений элементов управления CheckBoxList или RadioButtonList используется цикл *For Each,* который проверяет все элементы управления из списка. Вопреки ожиданиям, элемент управления CheckBoxList содержит не CheckBox (а RadioButtonList — не RadioButton). Вместо них CheckBoxList и RadioButtonList содержат элементы управления ListControl. Состояние ListControl можно определить посредством свойства *Selected,* как показано ниже:

Visual Basic .NET

```
Private Sub Button1_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button1.Click
    Dim lstItem As ListItem
    For Each lstItem In RadioButtonList1.Items
        If lstItem.Selected Then
            Response.Write(lstItem.Text & " is selected.<br>")
        End If
    Next
End Sub
```

Visual C#

```
private void Button1_Click(object sender,
        System.EventArgs e)
{
    foreach (ListItem lstItem in RadioButtonList1.Items)
    {
        if (lstItem.Selected)
            Response.Write(lstItem.Text +
                        " is selected.<br>");
    }
}
```

Отображение графики и графической рекламы

Графику можно отображать на Web-форме по-разному:

- как фон, заполняющий всю страницу (при помощи свойства *Background* Web-формы) либо ее часть (при помощи свойства *BackImageUrl* элемента управления Panel);
- на переднем плане — при помощи элемента управления Image;
- на кнопке — при помощи элемента управления ImageButton; так отображаются элементы, реагирующие на действия пользователя (см. пример в предыдущем разделе);
- как рекламное изображение — при помощи элемента управления AdRotator, который поочередно показывает картинки рекламного содержания, определенные в списке. Элементы AdRotator, помимо изображений, содержат гиперссылки на Web-сайт рекламодателя.

Элемент управления Image сам не реагирует на действия пользователя, но позволяет динамически отображать графические элементы в зависимости от действий, выполненных над другими элементами управления. Чтобы показать изображение при помощи элемента управления Image во время выполнения, воспользуйтесь его свойством *ImageUrl*. Например, следующий код в ответ на щелчок кнопки показывает картинку с видом Венеции:

Visual Basic .NET

```
Private Sub Button1_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles Button1.Click
    Image1.ImageUrl = "gondola.jpg"
End Sub
```

Visual C#

```
private void Button1_Click(object sender,
              System.EventArgs e)
{
    Image1.ImageUrl = "gondola.jpg";
}
```

Графику часто используют в рекламных целях — большинство рекламных баннеров в Интернете хранится в файлах формата **GIF.** Элемент управления AdRotator применяется в Visual Studio для демонстрации подобной рекламы. Для управления демонстрацией рекламных изображений AdRotator использует XML-файл, где хранятся URL изображения и страницы, на который попадет пользователь, щелкнувший изображение, а также сведения о приоритете и прочие свойства рекламных изображений. Во время выполнения AdRotator выбирает изображение из списка XML-файла и показывает его на странице.

Чтобы воспользоваться элементом управления AdRotator, выполните следующие действия.

1. Поместите на Web-форму элемент управления AdRotator.
2. Добавьте к текущему проекту XML-файл и откройте его.
3. В окне Properties выберите свойство *TargetSchema*, затем выберите из раскрывающегося списка значение AdRotator Schedule File. В результате Visual Studio добавит в XML-файл следующие строки:

```
<Advertisements xmlns="http://schemas.microsoft.com/AspNet/AdRotator-
Schedule-File">

</Advertisements>
```

4. В секции *<Advertisements>* добавьте для каждого рекламного изображения тег <Ad>.
5. Сохраните XML-файл и вернитесь к Web-форме в окне Design.
6. Выберите элемент управления AdRotator и поместите в свойство *AdvertisementFile* имя только что созданного XML-файла.

Теперь нужно отредактировать XML-файл, чтобы AdRotator удалось отобразить нужную информацию. Например, следующий XML-файл показывает два рекламных изображения:

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements xmlns="http://schemas.microsoft.com/AspNet/AdRotator-Schedule-
File">
    <Ad>
        <ImageUrl>../ads/sponsorad.gif</ImageUrl>
        <NavigateUrl>/ads/sponsorad.htm</NavigateUrl>
        <AlternateText>Click here to visit our sponsor.</AlternateText>
        <Keywo rd></Keyword>
        <Impressions>1</Impressions>
```

```
    </Ad>
    <Ad>
        <ImageUrl>../ads/net.gif</ImageUrl>
        <NavigateUrl>//www.gotdotnet.com</NavigateUrl>
        <AlternateText> ASP.NET tutorials and more.</AlternateText>
        <Keyword>ASP</Keyword>
        <Keyword>.NET</Keyword>
        <Impressions>5</Impressions>
    </Ad>
</Advertisements>
```

Теги, составляющие схему XML элемента управления AdRotator, описаны в таблице 4-5.

Таблица 4-5. Схема AdRotator

Тег	Значение
<Ad>	Открывает описание рекламного изображения
<ImageUrl>	Задает адрес рекламного изображения
<NavigateUrl>	Адрес страницы, на которую попадет пользователь, щелкнувший рекламное изображение
<AlternateText>	Текст всплывающей подсказки, появляющейся, когда пользователь наводит указатель мыши на изображение. Этот текст также появляется на месте изображения, если оно не доступно по адресу *ImageUrl* и не может быть показано
<Keyword>	Название категории для фильтрации показываемых изображений
<Impressions>	Число, представляющее вероятность показа; чем оно больше, тем чаще отображается соответствующее изображение

Группировка элементов управления

Элементы управления группируют, чтобы получить возможность манипулировать в коде отдельными областями Web-формы. Таким образом, например, удается выделить отдельную область Web-формы для обслуживания входа пользователя в систему, а после успешного входа — отключить или вовсе скрыть эту панель.

Элементы управления, размещенные на Web-форме, группируют при помощи элемента управления Panel (панель). Применение Panel с Web-формами отличается от применения одноименного элемента управления с Windows-формами. Во-первых, на поверхности Panel нельзя размещать элементы управления прямо; сначала нужно поместить их на Web-форму и лишь потом перетащить на Panel. Во-вторых, элемент управления Panel из Web Forms использует потоковую разметку. Это означает, что разместить элемент управления в определенном месте поверхности Panel нельзя.

Примечание Чтобы заставить элементы управления RadioButton взаимодействовать, не обязательно размещать их на одной панели; достаточно задать для них одинаковое значение свойства *GroupName* (подробнее об этом — в разделе «Получение и установка значений»).

Чтобы поместить элемент управления на поверхности элемента управления Panel, выполните следующие действия.

1. Разместите на Web-форме необходимые элементы управления.

2. Перетащите добавленные элементы управления на поверхность Panel в том порядке, в каком они должны отображаться; при этом Visual Studio располагает их на панели один за другим.
3. При помощи символов перевода каретки и пробелов поместите элементы управления в нужном месте формы.

На рис. 4-13 показана группа элементов управления, составляющая входную страницу Web-сайта. После входа пользователя панель становится скрытой (см. код ниже), а вместо нее отображается приветственное сообщение.

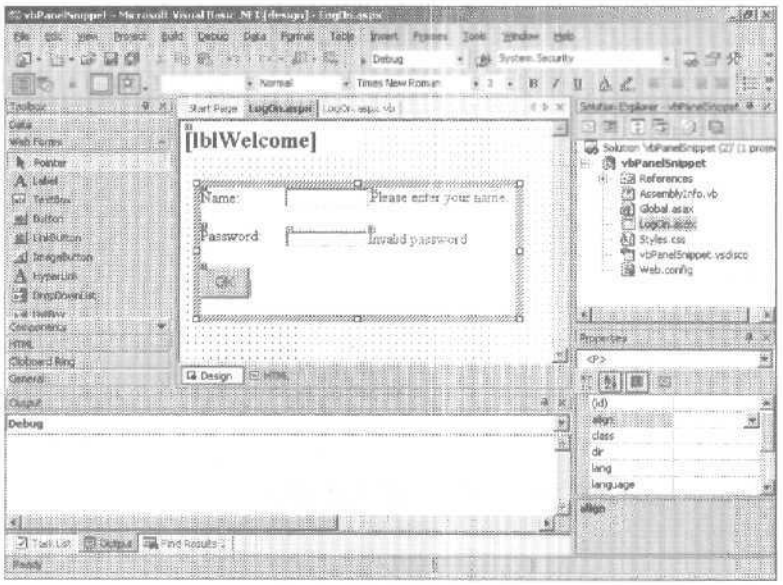


Рис. 4-13. Размещение элементов управления на панели

Этот код скрывает панель после входа пользователя на сайт:

Visual Basic .NET

```
Private Sub butOK_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butOK.Click
    ' Если имя и пароль введены верно, скрыть панель и показать сообщение.
    If txtName.Text = "Guest" And txtPassword.Text = "Wombat" Then
        pnlLogon.Visible = False
        lblWelcome.Text = "Welcome " & txtName.Text
    Else
        txtPassword.Text = ""
        vldPassword.Validate()
    End If
End Sub
```

Visual C#

```
private void butOK_Click(object sender, System.EventArgs e)
{
    // Если имя и пароль введены верно, скрыть панель и показать сообщение.
```

```
    if ((txtName.Text == "Guest") &&
            (txtPassword.Text == "Wombat"))
    {
        pnlLogon.Visible = False;
        lblWelcome.Text = "Welcome " + txtName.Text;

    }
    else
    {
        txtPassword.Text = "";
        vldPassword.Validate();
    }
}
```

Получение дат

Для получения и отображения дат используется элемент управления Calendar (календарь). Его внешний вид контролируется его свойствами, показанными на рис. 4-14.

Для получения и установки дат, заданных при помощи Calendar, используется обработчик события *SelectionChanged*, *а* также свойства *SelectedDate* и *Selected Dates.* Событие *SelectionChanged* инициирует возврат страницы на сервер, поэтому следующий код о гображает выбранную дату (или даты) сразу же после их установки:

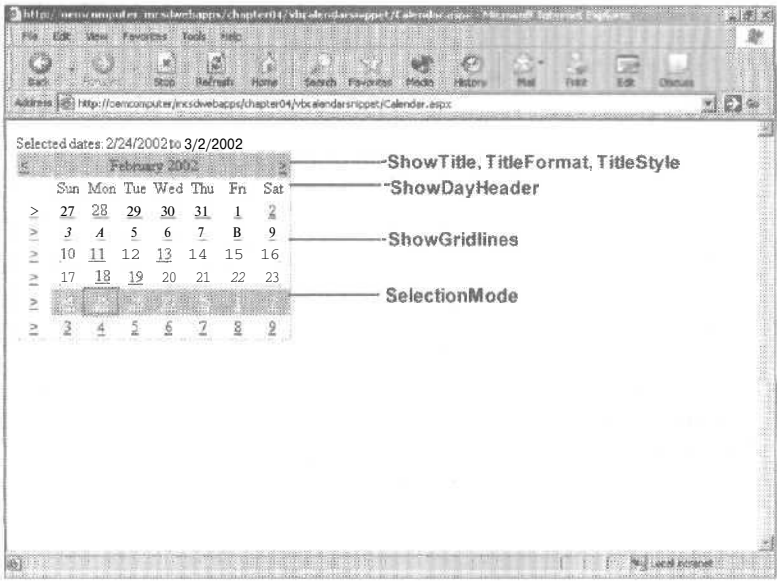


Рис. 4-14. Свойства элемента управления Calendar

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    ' После загрузки страницы показать в надписи текущую дату.
```

```
    lblDate.Text = "Current date: " & Date.Now.Date
End Sub

Private Sub calSource_SelectionChanged(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles calSource.SelectionChanged
    If calSource.SelectedDates.Count = 1 Then
        ' Если выбрана одна дата, показать ее в надписи.
        lblDate.Text = "Selected date: " & calSource.SelectedDate
    Else
        ' Если выбрано несколько дат, показать их,
        lblDate.Text = "Selected dates: " & calSource.SelectedDates(0) & _
        " to " & _
        calSource.SelectedDates(calSource.SelectedDates.Count - 1)
    End If
End Sub
```

Visual C#

```
private void Page_Load(object sender,
          System.EventArgs e)
{
    // Показать сегодняшнее число,
    lblDate.Text = "Current date: " +
           calSource.TodaysDate;
}

private void calSource_SelectionChanged(object sender,
      System.EventArgs e)
{
    // Показать сегодняшнее число.
    lblDate.Text = "Current date: " +
           calSource.TodaysDate;
    if (calSource.SelectedDates.Count == 1)
        // Если выбрана одна дата, показать её.
        lblDate.Text = "Selected Gate: " +
                  calSource.SelectedDate;
    else
        // Если выбрано несколько дат, показать их.
        lblDate.Text = "Selected dates: " +
                  calSource.SelectedDates[0] + " to " +
                  calSource.SelectedDates
                  [calSource.SelectedDates.Count - 1];

}
```

Получение файлов от клиента

Для загрузки файлов с клиента на сервер служит HTML-элемент управления File Field. На самом деле он состоит из связанных HTML-элементов управления Text Field и Submit

Button. Щелчок кнопки Browse запускает встроенный сценарий, отображающий на клиентском компьютере диалоговое окно выбора файла (рис. 4-15).

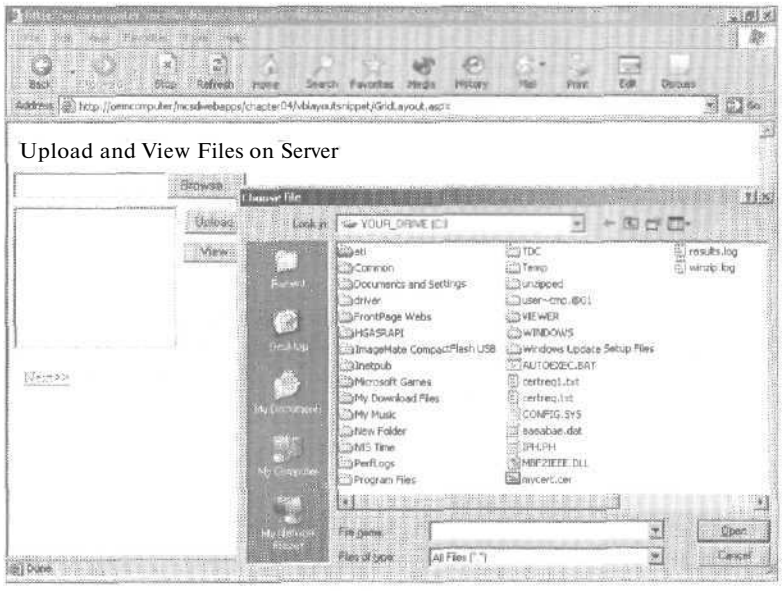


Рис. 4-15. Диалоговое окно Choose File

Чтобы загрузить на сервер выбранный файл с клиента, выполните следующие действия.

1. Перетащите на Web-форму элемент управления File Field.
2. Щелкните его правой кнопкой и выберите из контекстного меню команду Run As Server Control.
3. Щелкните правой кнопкой Web-форму и выберите команду View HTML Source — Visual Studio покажет эту Web-форму в виде HTML.
4. Добавьте к тегу *<form>* в **HTML-тексте** формы атрибут *enctype*. Этот атрибут устанавливает тип MIME-кодировки формы. Для загрузки файла MIME-кодировка должна быть установлена как multipart/form-data, а атрибут method — как post (см. текст, выделенный **полужирным шрифтом** в следующем примере):

```
<form action="webform1.aspx" method="post"
enctype="multipart/form-data" runat="server" ID="Form1">
```

5. Щелкните Web-форму правой кнопкой и выберите из контекстного меню команду View Design — Visual Studio откроет дизайнер Web-формы.
6. Добавьте к Web-форме элемент управления Button и обработчик события Click, которые позволят получить и сохранить выбранный файл. Например, следующий обработчик события сохраняет файл на сервере:

Visual Basic .NET

```
Private Sub butUpload_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles butUpload.Click
  Dim strFilename As String()
  ' Разбить полное имя файла на части,
  strFiiename = filUpload.PostedFile.FileName.Split("\")
  ' Сохранить закачанный файл на сервере с использованием
```

```
    ' полученного имени.
    filUpload.PostedFile.SaveAs_
      (Request.MapPath(Request.ApplicationPath) & "\" & _
      strFilename(UBound(strFilename)))
    ' Добавить файл к списку серверных файлов,
    lstServerFiles.Items.Add(strFilename(UBound(strFilename)))
    ' Выбрать первый элемент списка.
    lstServerFiles.SelectedIndex = 0
End Sub
```

Visual C#

```
private void butUpload_Click(object sender,
      System.EventArgs e)
{
    // Получить полное имя файла.
    string strFilename = filUpload.PostedFile.FileName;
          // Получить имя файла.
    strFilename =
          System.IO.Path.GetFileName(strFilename);
    // Сохранить закачанный файл на сервере с использованием
    // полученного имени.
    filUpload.PostedFile.SaveAs(Request.MapPath
          (Request.ApplicationPath) +  "\\" + strFilename);
    // Добавить файл к списку серверных файлов.
    lstServerFiles.Items.Add(strFilename);
    // Выбрать первый элемент списка.
    lstServerFiles.SelectedIndex = 0;
}
```

Метол *PostedFile* HTML-элемента управления File Field представляет закачиваемый файл. Этот метод возвращает объект, свойство *FileName* и метод *SaveAs* которого позволяют сохранить файл на сервере (см. пример выше). Следующий код демонстрирует один из способов получения файла с сервера; в этом случае полученный файл отображается в браузере.

Visual Basic .NET

```
Private Sub butView_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles butView.Click
    Dim strFilename As String
    ' Получить имя выбранного файла.
    strFilename = lstServerFiles.SelectedItem.ToString
    ' Отобразить файл как Web-страницу.
    Response.Redirect(Request.ApplicationPath & "/" & strFilename)
End Sub
```

Visual C#

```
private void butView_Click(object sender,
      System.EventArgs e)
{
    // Получить имя выбранного файла.
    string strFilename =
            lstServerFiles.SelectedItem.ToString();
    // Отобразить файл как Web-страницу.
    Response.Redirect(Request.ApplicationPath +
            "/" + strFilename);
}
```

Занятие 2. Проверка вводимых данных

Один из важнейших этапов получения данных от пользователя — их проверка. Критерии проверки могут быть разными: вводились ли данные вообще, верен ли их тип (например, действительно ли введенная строка является номером телефона), находятся ли они в пределах допустимого диапазона.

В ASP.NET есть особые элементы управления — *верификаторы* (validation controls), позволяющие перед использованием введенных данных проверять их соответствие различным критериям. На этом занятии рассказано, как применять верификаторы для выявления недопустимых данных и оказания пользователю помощи в устранении ошибок.

Изучив материал этого занятия, вы сможете:

- ✓ объяснить, как ASP.NET проверяет правильность вводимых данных и почему проверка выполняется как на клиентской, так и на серверной стороне;
- ✓ проверять данные, вводимые в текстовое поле, на соответствие одному или нескольким критериям;
- ✓ отображать сообщения, которые помогут пользователю исправить ошибки, допущенные им при вводе данных па Web-форме;
- ✓ отменять проверку, чтобы пользователь смог покинуть Web-форму, даже если при проверке возникают ошибки;
- ✓ создавать собственные критерии для проверки вводимых данных на серверной, клиентской или на обеих сторонах.

Продолжительность занятия — около 40 минут.

Использование проверки вводимых данных

Перед отправкой страницы на сервер верификаторы проверяют данные, которые были введены на стороне клиента в связанные с ними серверные элементы управления. Такая схема проверки существенно лучше тех, что были до нее, так как пользователь может сам обнаружить и исправить большинство ошибок, допущенных им при вводе без обмена данными с сервером.

Таблица 4-6. Верификаторы ASP.NET

Имя	Функция
RequiredFieldValidator	Проверяет наличие данных в элементе управления
CompareValidator	Проверяет, совпадают ли введенные данные с данными, введенными в другой элемент управления
RangeValidator	Проверяет, находится ли введенный элемент данных в заданном диапазоне
RegularExpressionValidator	Проверяет, соответствует ли формат введенных данных заданному
CustomValidator	Проверяет правильность вводимых данных при помощи серверного или клиентского сценария (либо того и другого)
ValidationSummary	Централизованно отображает сообщения об ошибках при вводе данных или краткое описание ошибки

Проверка вводимых данных на стороне клиента поддерживается посредством JScript-библиотеки WebUIValidation.js, которую клиент загружает отдельно. Несмотря на достаточно широкую поддержку JScript, используемая этой библиотекой объектная модель документа (Document Object Model, DOM) доступна только в Microsoft Internet Explorer 4.0 и выше. В связи с этим верификаторы поддерживают автоматическую проверку не только на клиентской, но и на серверной стороне. Проверка на серверной стороне выполняется всегда, независимо от результатов проверки на клиентской стороне. Такого рода двойной контроль обеспечивает корректность нестандартной проверки и не позволяет обойти проверку на стороне клиента.

Верификаторы проверяют значения серверных элементов управления, заданных их свойством *ControlToValidate*. В таблице 4-6 описаны шесть верификаторов.

Чтобы использовать верификаторы, выполните следующие действия.

1. Поместите верификатор на Web-форму и занесите в его свойство *ControlToValidate* имя проверяемого им элемента управления (в случае Compare Validator следует также определить свойство *ControlToCompare*).
2. Введите в свойство *ErrorMessage* верификатора текст сообщения об ошибке, отображаемого при вводе неверных данных элемент управления, проверяемый верификатором.
3. Если помимо сообщения об ошибке, заданного свойством *ErrorMessage,* нужно отобразить дополнительное сообщение, занесите его текст в свойство *Text.* Это позволит кратко указать, в каком месте формы возникла ошибка, а также показать более подробное описание ошибки при использовании свойства *ErrorMessage* элемента управления ValidationSummary.
4. Поместите на форму элемент управления ValidationSummary — это позволит централизованно отображать сообщения об ошибках, генерируемые всеми верификаторами.
5. Добавьте элемент управления, генерирующий событие, который вызывает возврат страницы на сервер. Хотя проверка вводимых данных должна осуществляться на стороне клиента, она не выполняется, пока не будет инициирован возврат страницы.

На рис. 4-16 показано, как выполняется простая проверка данных, вводимых на Web-форме.

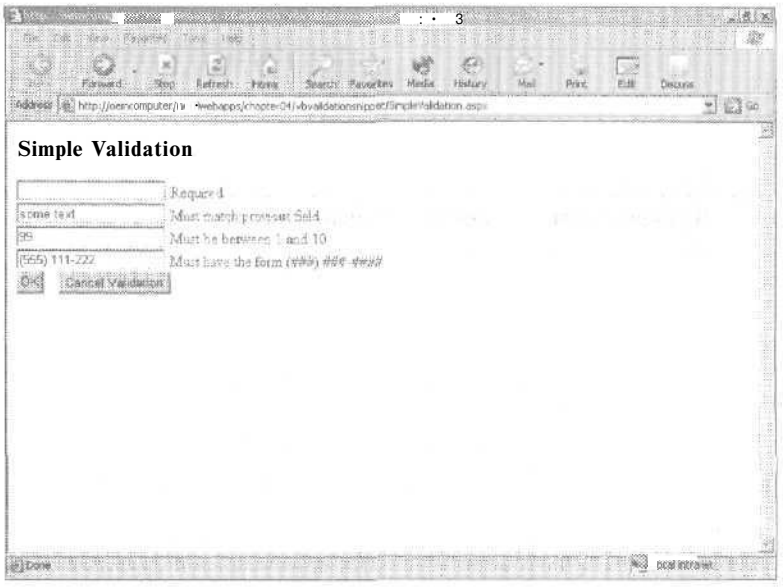


Рис. 4-16. Проверка данных, вводимых на **Web-форме**

На рис. 4-17 показана проверка с применением элемента управления ValidationSummary. Заметьте, что на рис. 4-16 значения свойства *Text* отображаются у каждого верификатора, а при использовании элемента управления ValidationSummary все сообщения об ошибках поступают в его свойство *ErrorMessage* и централизованно отображаются на форме.

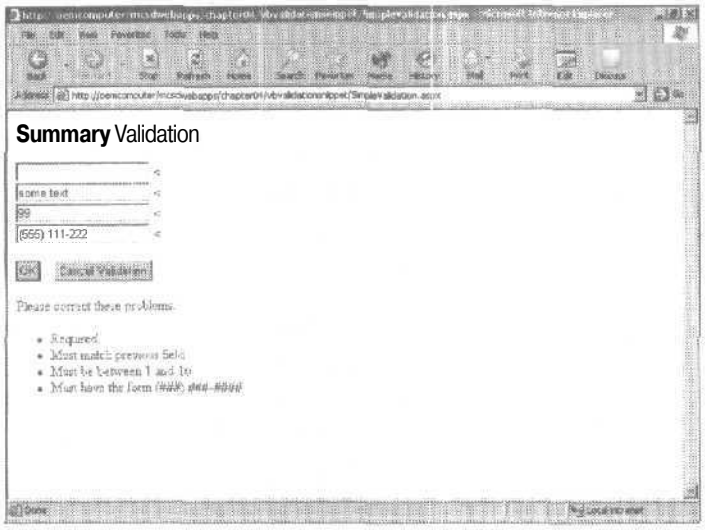


Рис. 4-17, Использование элемента управления ValidationSummary

Сообщения об ошибках, генерируемые при проверке вводимых данных, можно отображать и в диалоговом окне (рис. 4-18), для этого установите свойство *ShowMessage* элемента управления ValidationSummary в *True*.

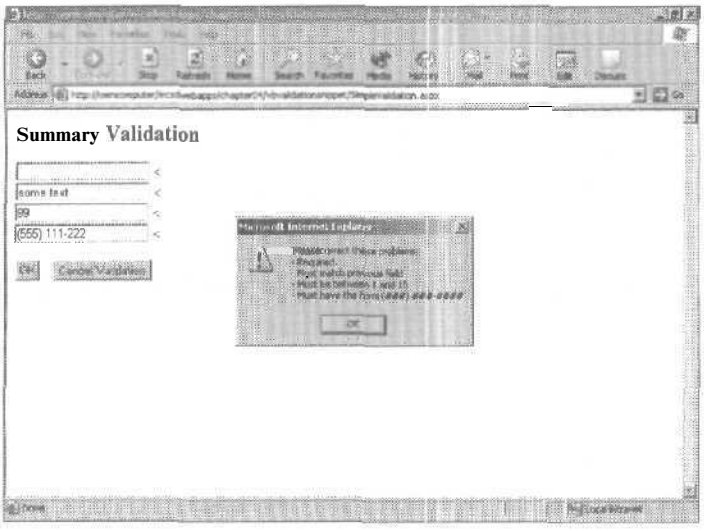


Рис. 4-18. Диалоговое окно Validation errors

Комбинирование верификаторов

С серверным элементом управления может быть связано несколько верификаторов. Например, для текстового поля, предназначенного для ввода телефонного номера (рис. 4-19), нужно проверять не только наличие значения, но и правильность формата введенных данных, таким образом, в этом случае требуются два верификатора.

Для интерпретации сообщений об ошибках, генерируемых при использовании нескольких верификаторов с одним элементом управления, необходимо понимать правила проверки вводимых данных, принятые в ASP.NET. Смысл этих правил понятен не сразу, но они позволяют генерировать сообщения, указывающие на конкретные ошибки.

Например, все верификаторы за исключением RequireFieldValidator приравнивают отсутствие введенного значения к вводу правильного значения. Таким образом, генерируются одни сообщения при отсутствии введенных данных (это проверяет RequireFieldValidator) и другие, если введенное значение находится вне допустимого диапазона или имеет неверный формат. Пример с полем для ввода телефонного номера показан на рис. 4-19.

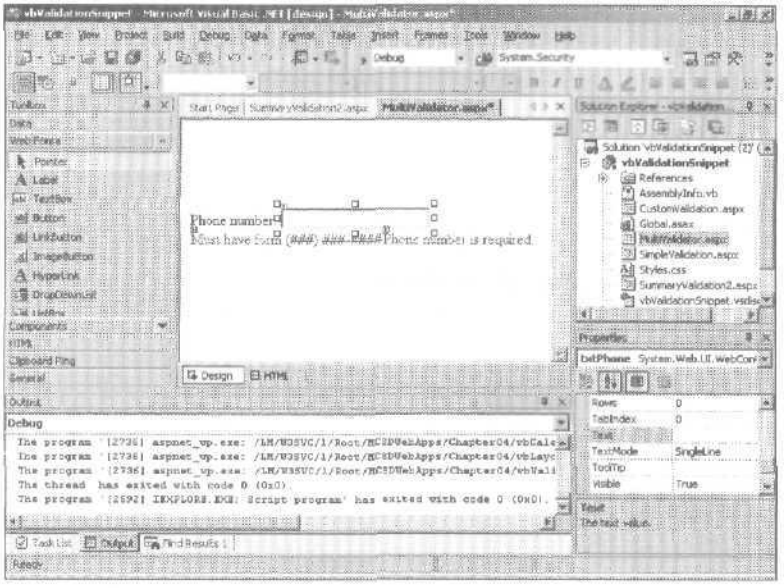


Рис. 4-19. Использование нескольких верификаторов

В таблице 4-7 перечислены правила преобразования данных, принятые элементом управления CompareValidator.

Таблица 4-7. Правила проверки, применяемые CompareValidator

Ситуация	Результат
Значение элемента управления, заданного свойством *ControlToValidate*, не может быть преобразовано в подходящий тип данных	Значение недопустимо
Значение элемента управления, заданного свойством *ControlToCompare*, не может быть преобразовано в подходящий тип данных	Значение допустимо

В последнем случае из таблицы 4-7 необходимо отдельно проверять данные, вводимые в элемент управления, заданный свойством *ControlToCompare*, чтобы гарантировать ввод в него допустимых данных.

Отмена проверки введенных данных

Поскольку введенные данные проверяются перед обработкой страницы сервером, пользователь может попасть в ловушку при проверке, если не предусмотрен способ отменить проверку, не возвращая страницу на сервер.

Для отмены **проверки** введенных данных необходим HTML-элемент **управления Submit, устанавливающий атрибут** *Page_ValidationActive* **в** *false* **(см. строку, выделенную полужирным шрифтом в следующем** HTML):

```
<INPUT id="butCancel" onclick="Page_ValidationActive=false;" style="Z INDEX:
114; LEFT; 386px; WIDTH: 62px; POSITION: absolute; TOP: 130px; HEIGHT: 30px"
type="submit" value="Cancel">
```

Этот HTML-текст определяет кнопку, которая отменяет проверку и возвращает страницу на сервер. Проверив свойство *IsValid* объекта *Page* при обработке события *Page_Load*, можно определить, что проверка прервана пользователем. В этом случае необходимо заново проверить страницу, поскольку при отмене проверки свойство *IsValid* получает значение *True*. Следующий код позволяет узнать, отменил ли пользователь проверку:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Проверить страницу, если пользователь отменил проверку на клиенте.
    If Page.IsPostBack Then Page.Validate()
    ' Проверить, допустимы ли введенные на странице данные.
    If Not Page.IsValid Then
        ' Пользователь отменил операцию, вернуться на страницу по умолчанию.
        Response.Redirect("default.htm")
    End If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Проверить страницу, если пользователь отменил проверку на клиенте.
    if (Page.IsPostBack)
    {
        Page. ValidateO;
        if (!Page.IsValid)
            // Пользователь отменил операцию.
            Response.Redirect("default.htm");
    }
}
```

Нестандартная проверка вводимых данных

Сложную проверку, которую не в состоянии обеспечить стандартные верификаторы, реализуют при помощи элемента управления CustomValidator и собственного кода, выполняющего проверку на серверной и при необходимости — на клиентской стороне.

Серверный код, выполняющий верификацию, помещают в обработчик события *ServerValidate,* аргументы которого дают доступ к проверяемому элементу управления. Показанный ниже обработчик проверяет, является ли введенное число простым:

Visual Basic .NET

```
' Этот код проверяет, является ли введенное число простым.
Private Sub CustomValidator1_ServerValidate _
  (ByVal source As System.Object, ByVal args As _
  System.Web.UI.WebControls.ServerValidateEventArgs) _
  Handles CustomValidator1.ServerValidate
    Try
        Dim iPrime, iCount As Integer
        ' Получить значение из ControlToValidate (оно передается в args)
        iPrime = Integer.Parse(args.Value)
        For iCount = 2 To iPrime \ 2
            ' Если число делится без остатка,
            ' оно не является простым - вернуть False.
            If iPrime Mod iCount = 0 Then
                args.IsValid = False
                Return
            End If
        Next
        ' Это простое число, вернуть True.
        args.IsValid = True
        Return
    Catch ex As Exception
        ' Если при синтаксическом разборе возникла ошибка, вернуть False.
        args.IsValid = False
       Return
    End Try
End Sub
```

Visual C#

```
private void vldtxtPrime_ServerValidate(object source,
      System.Web.UI.WebControls.ServerValidateEventArgs args)
{
    try
    {
        // Получить значение из ControlToValidate
        // (оно передается как args),
        int iPrime = Int32.Parse(args.Value);
        for (int iCount = 2; iCount <= (iPrime / 2);
```

```
                iCount++)
        {
            // Если число делится без остатка,
            // оно не является простым - вернуть False.
            if((iPrime % iCount) == 0)
            {
                args.IsValid = false;
                return;
            }
            // Это простое число, вернуть True.
            args.IsValid = true;
            return;
        }
    }
    catch(Exception e)
    {
        // Если при синтаксическом разборе возникла ошибка, вернуть False,
        args.IsValid = false;
        return;
    }
}
```

Для проверки вводимых данных на клиентской стороне определите проверяющий сценарий в свойстве *ClientValidationFunction* элемента управления Custom Validator. Проверка на клиентской стороне необязательна, но если она используется, клиентский и серверный проверяющий код должны быть идентичными. Следующий сценарий является клиентской версией предыдущего серверного кода:

VBScript

```
<script language="vbscript">
        Sub ClientValidate(source, arguments)
            For iCount = 2 To arguments.Value \ 2
                ' Если число делится без остатка,
                ' оно не является простым - вернуть False.
                If arguments.Value Mod iCount = 0 Then
                    arguments.IsValid = False
                    Exit Sub
                End If
            Next
            arguments.IsValid = True
        End Sub
</script>
```

JScript

```
<script language="jscript">
        function ClientValidate(source, arguments)
        {
```

```
        for (var iCount - 2; iCount <= arguments.Value / 2;
                iCount++)
        {
            ' Если число делится без остатка,
            ' оно не является простым - вернуть False.
            if ((arguments.Value % iCount) == 0)
            {
                arguments.IsValid = false;
                return false;
            }
        }
        // Это простое число, вернуть True.
        arguments.IsValid - true;
        return true;
    }
</script>
```

Занятие 3. Переход между формами

Данные, связанные ссылками, — это сущность Web. В Web Forms гиперссылки и методы перехода являются тем «цементом», который связывает Web-формы в единое приложение. ASP.NET поддерживает несколько способов перехода между страницами в приложениях, отличающихся методикой отображения и обмена данными между страницами.

На этом занятии вы узнаете об использовании различных способов перехода между страницами в коде и **HTML.**

Изучив материал этого занятия, вы сможете:

- ✓ перечислить способы перехода между страницами в приложении Web Forms;
- ✓ выбрать метод перехода, соответствующий задаче программирования;
- ✓ сохранять состояние отображения формы между запросами;
- ✓ обрабатывать вторую Web-форму, не покидая текущую Web-форму;
- ✓ отображать страницу в новом окне браузера и управлять новым окном из серверного кода.

Продолжительность занятия — около 35 минут.

Методы перехода между страницами

ASP.NET поддерживает пять различных способов перехода между страницами в приложениях (таблица 4-8).

Таблица 4-8. Способы перехода между страницами

Способ перехода	Назначение
Элемент управления Hyperlink	Переход на другую страницу
Метод *Response.Redirect*	Переход на другую страницу в коде, эквивалент щелчка гиперссылки
Метод *Server. Transfer*	**Закрывает текущую** Web-форму **и начинает** исполнение другой Web-формы. Работает только при переходе между страницами Web Forms (.aspx)
Метод *Server.Execute*	Начинает исполнение новой Web-формы, не закрывая текущую, при этом содержимое обеих форм объединяется. Работает только при переходе между страницами Web Forms (.aspx)
Сценарий *Window.Open*	**Открывает** страницу **на клиенте в** новом окне браузера

Использование гиперссылок и перенаправления

Серверный элемент управления Hyperlink отвечает на инициированное пользователем событие click отображением страницы, заданной его свойством *NavigateURL.* Элемент управления Hyperlink не поддерживает никаких пользовательских событий на стороне сервера. Для обработки события click в коде воспользуйтесь серверными элементами управления LinkButton или ImageButton.

Объекты LinkButton и ImageButton выполняют переход при помощи метода *Redirect* объекта *Response:*

Visual Basic .NET

```
Private Sub LinkButton1_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles LinkButton1.Click
    ' Отобразить следующую страницу.
    Response.Redirect("NextPage.aspx")
End Sub
```

Visual C#

```
private void LinkButton1_Click(object sender. System.EventArgs e)
{
    // Отобразить следующую страницу.
    Response.Redirect("NextPage.aspx");
}
```

Использование метода Transfer

Применение метола *Transfer* весьма напоминает обработку гиперссылки или использование метода *Redirect* за исключением того, что метод *Transfer* может сохранять между запросами некоторые данные исходной страницы. Если установить его аргумент *preserveForm в True,* то свойства формы *QueryString, ViewState* и сведения об обработчиках события текущей формы станут доступными из целевой формы.

Прежде чем вызывать метод *Transfer* со свойством *preserve Form,* установленным в *True,* необходимо присвоить значение *False* атрибуту *EnableViewState Mac* страничной директивы Page. По умолчанию ASP.NET хеширует данные состояния отображения. Если атрибут *EnableView State Mac* определен как *False,* то хеширование не выполняется и данные состояния отображения могут быть прочитаны после перехода на другую Web-форму, Следующий код демонстрирует, как отменить хеширование и сделать состояние отображения доступным после перехода на другую страницу:

Visual Basic .NET

```
<%@ Page language="vb" EnableViewStateMac="false"
Codebehind="RedirectNTransfer.aspx.vb"
Inherits="vbNavigateSnippet.WebForm1" %>
```

Visual C#

```
<%@ Page language="c#" EnableViewStateMac="false"
Codebehind="RedirectNTransfer.aspx.cs" AutoEventWireup="false"
Inherits="csNavigateSnippet.WebForm1" %>
```

Следующий обработчик события ImageButton демонстрирует передачу данных между формами с помощью метода *Transfer*

Visual Basic .NET

```
' Webform1.aspx
Private Sub imgTransfer_Click(ByVal sender As System.Object, _
  ByVal e As System.Web.UI.ImageClickEventArgs) _
  Handles imgTransfer.Click
```

```
    ' Перейти на другую Web-форму, сохранив состояние отображения.
    Server.Transfer("Webform2.aspx", True)
End Sub
```

Visual C#

```
// Webform1.aspx
private void ImageButton1_Click(object sender,
    System.Web.UI.ImageClickEventArgs e)
{
    // Перейти на другую Web-форму, сохранив состояние отображения.
    Server.Transfer("Webform2.aspx", true);
}
```

Для извлечения данных из состояния отображения исходной страницы используется метод *Form* объекта *Request*. Следующий код отображает на форме Webform2.aspx значения элементов управления (значение TextBox, а также координаты X и Y щелчка ImageButton), принадлежащих форме Webform1.aspx.

Visual Basic .NET

```
' Webform2.aspx
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    Dim colForm As _
      System.Collections.Specialized.NameValueCollection
    ' Получить данные из исходной Web-формы.
    colForm = Request.Form
    ' Отобразить значение поля, принадлежащего Webforml.
    Response.Write("TextBox1.Text: " & _
      colForm.Item("TextBox1") & "<br>")
    ' Отобразить координаты щелчка.
    Response.Write("ImageButton X, Y coords: " & _
      colForm.Item("imgTransfer.x") & ", " & _
      colForm.Item("imgTransfer.y") & "<br>")
End Sub
```

Visual C#

```
// Webform2.aspx
    private void Page_Load(object sender, System.EventArgs e)
    {
        System.Collections.Specialized.NameValueCollection colForm;
        // Получить данные из исходной Web-формы.
        colForm = Request.Form;
        // Отобразить значение поля, принадлежащего Webforml.
        Response.Write("TextBox1.Text: " + colForm["TextBox1"] + "<br>");
        // Отобразить координаты щелчка.
        Response.Write("ImageButton X, Y coords: " +
             colForm["imgTransfer.x"] + ", "
```

```
            + colForm["imgTransfer.y"] + "<br>");
    }
```

Метода *Transfer* объекта *Server* позволяет передавать значения элементов управления между последовательными запросами (рис. 4-20).

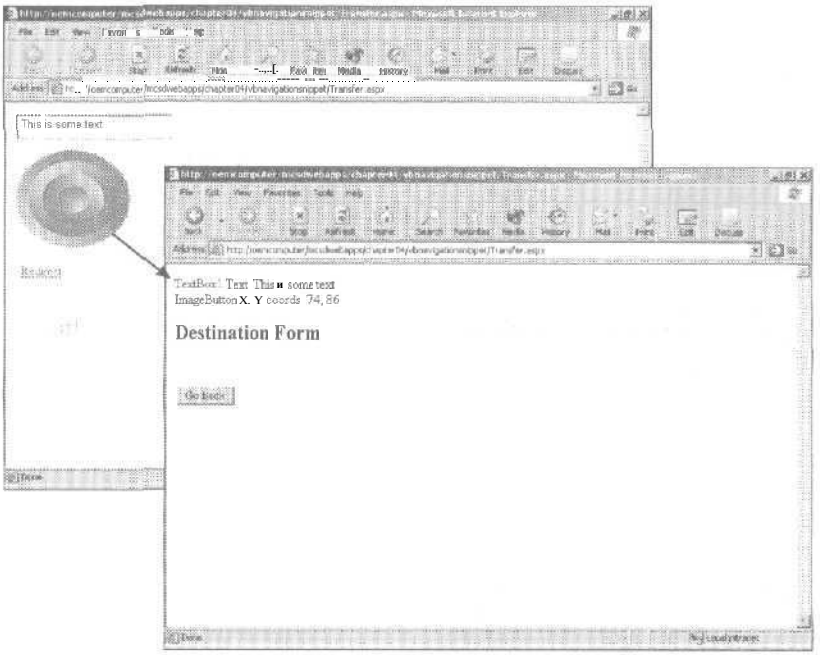


Рис. 4-20, Работа метода *Transfer* объекта *Server*

Примечание Методы *Transfer* и *Execute* объекта Server работают исключительно с Web-формами. Попытка использования этих методов для перехода на HTML-страницу приведет к ошибке времени выполнения.

Использование метода Execute

Метод *Execute* объекта *Server* позволяет обрабатывать вторую Web-форму, не покидая первую. Таким образом, при помощи этого метода удается направлять результаты обработки одной Web-формы в определенную область другой формы. Как и в случае *Transfer* перед вызовом метода *Execute* необходимо установить атрибут *EnableViewStateMac* в *False,* чтобы отменить хеширование состояние отображения.

Следующий код исполняет Web-форму Table.aspx и отображает результаты на текущей странице при помощи элемента управления Literal:

Visual Basic .NET

```
Private Sub butExecute_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butExecute.Click
    Dim swrTarget As New System.IO.StringWriter()
    ' Обработать Web-форму и сохранить результат.
    Server.Execute("Table.aspx", swrTarget)
```

```
    ' Отобразить результат с помощью элемента управления Literal.
    litTable.Text = "<h2>Table Results</h2>" & swrTarget.ToString
End Sub
```

Visual C#

```
private void butExecute_Click(object sender, System.EventArgs e)
    {
        System.IO.StringWriter swrTarget = new
            System.IO.StringWriter();
        // Обработать Web-форму и сохранить результат.
        Server.Execute("Table.aspx", swrTarget);
        // Отобразить результат с помощью элемента управления Literal.
        litTarget.Text = "<h2>Table Results</h2>" +
            swrTarget.ToString();
    }
```

Второй аргумент метода *Execute* является необязательным. **Если** опустить его, результат обработки Web-формы будет направлен на текущую страницу. В результате содержимое страниц объединяется, а их серверные элементы управления могут реагировать на события, инициированные пользователем (рис. 4-21).

Объединяя Web-формы с помощью метода *Execute,* не забывайте вот о чем: если вторая Web-форма генерирует событие postback, то первая Web-форма будет очищена. Поэтому объединение Web-форм полезно, главным образом, при отсутствии на второй Web-форме элементов управления, вызывающих возврат страницы.

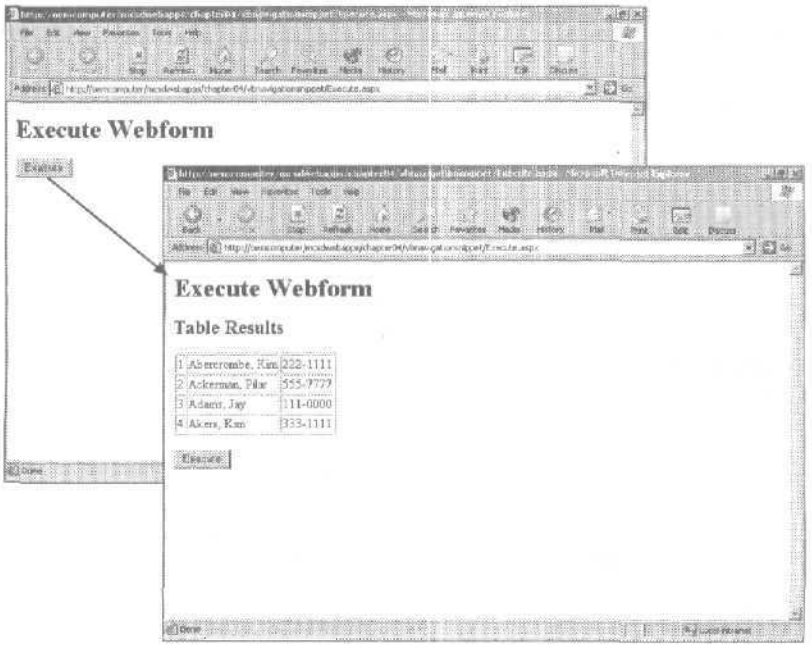


Рис. 4-21. Работа метода Execute объекта Server

Отображение страницы в новом окне браузера

Для запуска нового экземпляра браузера предназначен метод *Open* клиентского объекта *Window*. Поскольку новое окно генерируется на стороне клиента, его удается создать только при помощи клиентского сценария, но его содержимым и внешним видом допустимо управлять и с сервера.

Следующее определение HTML-элемента управления Button демонстрирует простейший вариант использования метода *Window.Open* (см. строку, выделенную **полужирным** шрифтом):

```
<INPUT style="Z-INDEX: 102; LEFT: 55px; WIDTH: 81px; POSITION: absolute;
TOP: 156px; HEIGHT: 24px" onclick="window.open('webform2.aspx')" type="submit"
value="New Window">
```

Чтобы получить возможность задавать целевой URL с помощью переменной, замените **webform2.aspx** следующим тегом:

```
<INPUT style="Z-INDEX: 102; LEFT: 55px; WIDTH: 81px; POSITION: absolute;
TOP: 156px; HEIGHT: 24px" onclick="window.open('<%# urlTarget %>')"
type="submit" value="New Window">
```

Чтобы серверный код получил возможность обновлять целевой URL, используйте открытую (*Public*) переменную и привязку данных. Следующий обработчик события *Page_Load* задает целевой URL и во время загрузки страницы обновляет его с использованием связанных данных:

Visual Basic .NET

```
Public urlTarget As String
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    urlTarget = "webform2.aspx"
    Page.DataBind()
End Sub
```

Visual C#

```
public string urlTarget;
private void Page_Load(object sender, System.EventArgs e)
{
    urlTarget = "webform2.aspx";
        Page.DataBind();
}
```

Поскольку метод *Window.Open* поддерживает множество аргументов, управляющих различными функциями новых окон браузера, для управления ими можно создать класс. Классы инкапсулируют всевозможные параметры, облегчая управлением ими в ООП. Обработчик события *Page_Load* и определение класса, показанные ниже, демонстрируют управление размерами, размещением и URL нового окна из серверного кода с использованием класса *BrowserWindow*:

Visual Basic .NET

```
Public urlTarget As New BrowserWindow()

Private Sub Page_Load(ByVal sender As System.Object, _
```

```
  ByVal e As System.EventArgs) Handles MyBase.Load
    With urlTarget
        ' Задать URL для нового окна.
        .URL = "HTMLPage1.htm"
        .Top = 100
        .Left = 100
        .Width = 400
        .Height = 400
    End With
    ' Обновить HTML-кнопку.
    Page.DataBind()
End Sub

' Это класс, управляющий новыми окнами браузера, создаваемыми
' при помощи сценариев,
' Показанные здесь настройки по умолчанию совпадают с таковыми браузера,
Public Class BrowserWindow
    ' Строка адреса по умолчанию пуста (blank).
    Public URL As String = "about:blank"
    ' Целочисленные аргументы по умолчанию равны 0.
    Public Height As Integer = 0
    Public Width As Integer = 0
    Public Top As Integer = 0
    Public Left As Integer = 0
    ' Аргументы типа булевых, 0 = "нет": 1 = "да",
    Public ChannelMode As Integer = 0
    Public Directories As Integer = 1
    Public FullScreen As Integer = 0
    Public Location As Integer = 1
    Public Resizable As Integer - 1
    Public ScrollBars As Integer = 1
    Public Status As Integer = 1
    Public TitleBar As Integer = 1
    Public ToolBar As Integer = 1
    Public MenuBar As Integer - 1
End Class
```

Visual C#

```
public BrowserWindow urlTarget = new BrowserWindow();

private void Page_Load(object sender, System.EventArgs e)
{
    urlTarget.URL = "Execute.aspx";
    urlTarget.Top - 100;
    urlTarget.Left = 100;
    urlTarget.Width = 400;
    urlTarget.Height = 400;
```

```
    // Обновить HTML-кнопку.
    Page.DataBind();
}

// Это класс, управляющий новыми окнами браузера, создаваемыми
// при помощи сценариев.
// Показанные здесь настройки по умолчанию совпадают с таковыми браузера,
public class BrowserWindow
{
    // Строка адреса по умолчанию пуста (blank).
    public string URL = "about:blank";
    // Целочисленные аргументы по умолчанию равны 0.
    public int Height = 0;
    public int Width - 0;
    public int Top = 0;
    public int Left - 0;
    // Аргументы типа булевых, 0 - "нет"; 1 - "да",
    public int ChannelMode - 0:
    public int Directories = 1;
    public int FullScreen = 0;
    public int Location = 1;
    public int Resizable - 1;
    public int ScrollBars = 1;
    public int Status - 1;
    public int TitleBar = 1;
    public int ToolBar = 1:
    public int MenuBar = 1;
}
```

HTML-код кнопки, берущий параметры нового окна из экземпляра показанного выше класса, выглядит примерно так (переменные, предоставляемые сервером, выделены полужирным шрифтом):

```
<INPUT style="Z-INDEX: 103; LEFT: 25px; WIDTH: 126px; POSITION: absolute;
TOP: 60px; HEIGHT: 33px" type="button" value="Show New Window"
onclick="window.open('<%# urlTarget.URL %>', null,
'height=<%# urlTarget.Height %>,
width=<%# urlTarget.Width %>, top=<%# urlTarget.Top %>,
left=<%# urlTarget.Left %>, channelmode=<%# urlTarget.ChannelMode %>,
directories=<%# urlTarget.Directories %>,
fullscreen=<%# urlTarget.FullScreen %>,
location=<%# urlTarget.Location %>,
menubar=<%# urlTarget.MenuBar 56>,
resizable=<%# urlTarget.Resizable %>,
scrollbars=<%# urlTarget.ScrollBars %>,
status=<%# urlTarget.Status %>,
titlebar=<%# urlTarget.TitleBar %>,
toolbar=<%# urlTarget.ToolBar %>')">
```

Резюме

- Серверные элементы управления поддерживают свойства, методы и события, которые можно использовать в серверном коде.
- HTML-элементы управления поддерживают атрибуты и события, используемые в клиентских сценариях. Чтобы получить доступ к свойствам HTML-элемента управления на сервере, следует добавить к его определению атрибут *runat=server*.
- ASP.NET проверяет данные, вводимые в элементы управления на стороне клиента, непосредственно перед возвратом страницы на сервер. После успешной проверки на стороне клиента Web-форма проходит аналогичную проверку на сервере (до срабатывания события *Page_Load*). Это гарантирует проверку вводимых данных, даже если в результате модификации Web-формы проверку на клиенте удалось обойти, и обеспечивает поддержку проверки данных для браузеров старше Internet Explorer 4.0.
- Для проверки содержимого поля данных на соответствие нескольким критериям используется несколько верификаторов. Например, текстовое поле для ввода телефонного номера следует проверять верификаторами RequiredFieldValidator (проверяет наличие значения) и RegularExpressionValidator (проверяет формат).
- Существует несколько возможностей для перехода между страницами в приложениях Web Forms (методы объекта *Server* работают только с Web-формами):
  - гиперссылки;
  - метод *Response.Redirect*;
  - метод *Server. Transfer;*
  - метод *Server.Execute*;
  - сценарии, работающие на стороне клиента.
- Действия в браузере на стороне клиента выполняются при помощи клиентских сценариев. Чтобы открыть страницу в новом окне, применяют HTML-элемент управления Button с атрибутом onclick="window.open()", поскольку клиентские сценарии — это единственный вид исполняемого кода Web-приложения, работающего на стороне клиента.

Практикум 4. Создание текстового редактора на основе Web-форм

В этом практикуме вы создадите многостраничное приложение Web Forms, позволяющее создавать и редактировать текстовые файлы, хранимые на сервере. Приложение определяет каталог, в котором хранятся файлы пользователя, на основе его регистрационного имени. Оно не выполняет строгой проверки паролей, у всех пользователей этого приложения один пароль — Guest. Аутентификация с использованием пароля в деталях обсуждается в главе 8.

Продолжительность практикума - около 45 минут.

Внимание! Чтобы приложение смогло создавать и удалять файлы, учетная запись ASPNET должна иметь права на запись и модификацию для корневого каталога приложения. По умолчанию учетная запись ASPNET обладает только правами на чтение и исполнение файлов. Для получения сведений об определении прав доступа к каталогу приложения см. в главе 9.

Упражнение 1. Создание входной формы приложения



Сейчас вы создите Web-форму, которая позволит пользователям войти в текстовый редактор. Эта Web-форма проверяет вводимые данные, выполняет потоковую разметку, переход на другие страницы, кроме того, здесь применяется пространство имен *System.IO* (для работы с каталогом), В законченном виде эта Web-форма выглядит, как на рис. 4-22.

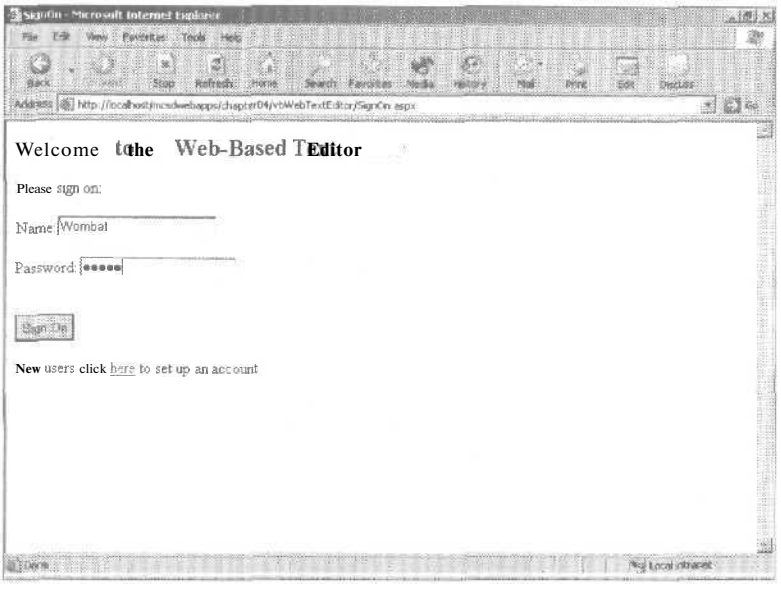


Рис. 4-22. Готовая Web-форма SignOn.aspx

► Создание нового проекта Web-приложения и входной Web-формы

1. Откройте новый проект ASP.NET Web application. В диалоговом окне New Project введите имя **WebTextEditor**.
2. В окне Properties переименуйте файл WebForml.aspx в **SignOn.aspx**.
3. В окне Properties выберите объект *DOCUMENT* и установите свойство *pageLayout* в *FlowLayout*.

После этого разместите на странице элементы управления. Для этого дважды щелкните изображение элемента управления на Toolbox. Если создавать элементы управления в том порядке, в котором они будут появляться на странице, дело пойдет быстрее. В следующей таблице перечислены элементы управления в порядке их создания, а также их свойства.

Элемент управления	Свойство	Значение
Literal	ID	litNoAccount
TextBox	ID	txtName
RequiredFieldValidator	ID	vldtxtName
	ControlToValidate	txtName
	ErrorMessage	Please enter a name.
TextBox	ID	txtPassword
RequiredFieldValidator	ID	vldtxt Password
	ControlToValidate	txtPassword
	ErrorMessage	Please enter the password.
RangeValidator	ID	vldtxtPasswordGuest
	ControlToValidate	txtPassword
	ErrorMessage	The password is «Guest».
	MinimumValue	Guest
	Maximum Value	Guest
Button	ID	butSignOn
	Text	Sign On

Создав элементы управления и определив необходимые свойства, переключите Designer в режим HTML. Далее вы будете редактировать страницу в виде HTML-текста, добавляя текст и элементы HTML прямо в HTML-текст формы.

► Редактирование Web-формы в режиме HTML

1. Щелкните Web-форму правой кнопкой и выберите в контекстном меню команду View HTML Source. Добавьте к форме текст и HTML-теги, выделенные в показанном ниже примере **полужирным шрифтом:**

```
<form id="Form1" method="post" runat="server">
<h2>
Welcome to the Web-Based Text Editor
</h2>
<asp:Literal id="litNoAccount" runat="server"></asp:Literal>
<P>
Please sign on:
```

```
</p>
Name:
<asp:TextBox id="txtName" runat="server" Width="165px"></asp:TextBox>
<asp:RequiredFieldValidator id="vldtxtName" runat="server"
ErrorMessage="Please enter a name."
ControlToValidate="txtName"></asp:RequiredFieldValidator>
<br>
<br>
Password:
<asp:TextBox id="txtPassword" runat="server" Width="163px" Height="21px"
TextMode="Password"></ asp:TextBox>
<asp:RequiredFieldValidator id="vldtxtPassword" runat="server"
ErrorMessage="Please enter a password."
ControlToValidate="txtPassword"></asp:RequiredFieldValidator>
<br>
<asp:RangeValidator id="vldtxtPasswordGuest" runat="server"
ControlToValidate="txtPassword" ErrorMessage='The password is "Guest".'
MaximumValue="Guest" MinimumValue="Guest"></asp:RangeValidator>
<br>
<br>
<asp:Button id="butSignOn" runat="server" Text="Sign On" Width="61px"
Height="28px"></asp:Button>
<P>
New users click <a href="NewAccount.aspx">here<a> to set up an account,
</p>
</form>
```

2. Для просмотра результатов изменений щелкните Web-форму правой кнопкой и выберите из контекстного меню команду View Design.
3. Закончив редактирование Web-формы, переключитесь в режим Design и дважды щелкните SignOn, чтобы вызвать окно Code. Перед определением класса Web-формы поместите следующий оператор:

Visual Basic .NET

```
Imports System.IO
```

Visual C#

```
using System.IO;
```

4. Добавьте в обработчик события *butClick* следующий код:

Visual Basic .NET

```
Private Sub butSignOn_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles butSignOn.Click
    Dim strPath As String
    'Если такой пользователь существует, должен быть каталог
    ' с таким же именем. . .
    strPath = Server.MapPath(Request.ApplicationPath) & "\" & txtName.Text
```

```
    If Directory.Exists(strPath) Then
        ' Определить переменную состояния Session.
        Session("Path") = strPath
        Server.Transfer("FileManager.aspx")
    Else
        ' ...В противном случае сообщить, что пользователь не
        ' найден.
        litNoAccount.Text = ("<p>The name " & txtName.Text & _
        " wasn't found.  & _
        "Check the name, or click <A href='NewAccount.aspx'>here
</A>" & " if you are a new user.</p>")
    End If
End Sub
```

Visual C#

```
private void butSignOn_Click(object sender, System.EventArgs e)
{
    string strPath;
    // Если такой пользователь существует, должен быть
    // каталог с таким же именем...
    strPath = Server.MapPath(Request.ApplicationPath) + "\\"
        + txtName.Text;
    if(Directory.Exists(strPath))
    {
        // Определить переменную состояния Session.
        Session["Path"] = strPath;
        Server.Transfer("FileManager.aspx");
    }
    else
        // ...В противном случае сообщить, что пользователь
        // не найден.
        litNoAccount.Text = "<p>The name " + txtName.Text +
            " wasn't found. Check the name, or click " +
                "<A href='NewAccount.aspx'>here</A> if you " +
                "are a new user.</p>";
}
```

Упражнение 2. Создание формы NewAccount



Сейчас вы создадите Web-форму, которая генерирует учетную запись для нового пользователя. В контексте этого приложения «учетная запись» — всего лишь каталог, расположенный в корневом каталоге приложения и названный по имени пользователя. В этом случае требования безопасности также игнорируются; дополнительные сведения о безопасности перечислены в главе 8.

Эта Web-форма использует проверку вводимых данных, потоковую разметку, переход между страницами, HTML-элементы управления и пространство имен *System.IO* (для работы с каталогами). Имя пользователя совпадает с именем каталога для хранения файлов

этого пользователя, созданного **в корневом каталоге приложения. Готовая** Web-форма должна **выглядеть, как на рис.** 4-23.

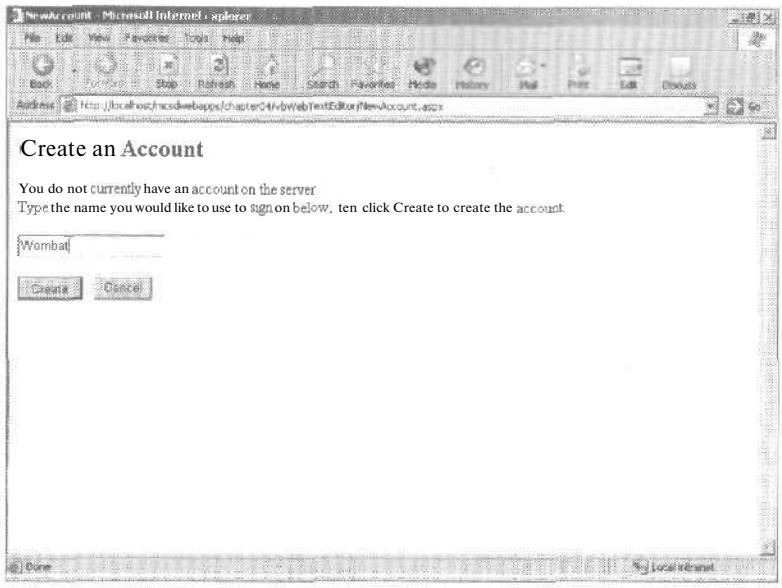


Рис. 4-23. Готовая Web-форма NewAccount.aspx

► **Создание Web-формы NewAccount.aspx**

1. **Создайте новую Web-форму с именем** NewAccount.aspx.
2. **В окне Properties выберите объект *DOCUMENT к* установите его свойство** *pageLayout* как FlowLayout.
3. **Добавьте** элементы **управления и** задайте их свойства, как показано в следующей таблице.

Элемент управления	Свойство	Значение
Literal	ID	litName Exists
TextBox	ID	txtName
Required Field Validator	ii)	vldtxtName
	ControlToValidate	txtName
	ErrorMessage	Please enter a name.
Button	ID	butCreate
	Text	Create
Button HTML control	id	butCancel
	value	Cancel

4. **Переключите** Web-форму **в режим HTML и добавьте к ней текст и HTML-теги, выделенные в показанном ниже примере полужирным шрифтом:**

```
<form id="Form1" method="post" runat="server">
<h2>
Create an Account
```

```
</h2>
<asp:Literal id="litNameExists" runat="server"></asp:Literal>
<P>
You do not currently have an account on the server.
<br>
Type the name you would like to use to sign on below, then click Create to
create the account.
</p>
<asp:TextBox id="txtName" runat="server" Width="153px"
Height="24px"></asp:TextBox>
<asp:RequiredFieldValidator id="vldtxtName" runat="server"
ErrorMessage="Please enter a name."
ControlToValidate="txtName"></asp:RequiredFieldValidator>
<p>
</p>
<asp:Button id="butCreate" runat="server" Text="Create" Width="67"
Height="24"></asp:Button>
   <INPUT type="button" onclick="history.back()" value="Cancel"></
BUTTON>
</form>
```

Последняя **из** вставленных **выше** строк (`onclick="history.back()"`) **позволяет отменить операцию без проверки введенных** значений. **Ключевой момент** здесь **— использование HTML-элемента управления. Это** позволяет **вернуться на** входную **форму** (SignOn), **не возвращая форму на сервер.**

5. **Закончив редактирование** Web-формы, **переключитесь в режим Design** и дважды щелкните **Create, чтобы вызвать окно Code. Поместите перед** определением **класса Web-формы следующий** оператор:

Visual Basic .NET

```
Imports System.IO
```

Visual C#

```
using System.IO;
```

6. **Добавьте к обработчику события** *butCreate_Click* **следующий код:**

Visual Basic .NET

```
Private Sub butCreate_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butCreate.Click
    Dim strPath As String
    ' Проверить, существует ли <аталог.
    strPath = Server.MapPath(Request.ApplicationPath) & "\" & txtName.Text
    If Directory.Exists(strPath) Then
        ' Предложить пользователю выбрать другое имя.
        litNameExists.Text = "<p>The name " & txtName.Text & _
          " already exists. Please choose a different one.</p>"
        Return
```

```
    Else
        Try
            ' Создать каталог.
            Directory.CreateDirectory(strPath)
            ' Определить переменную состояния Session.
            Session("Path") = strPath
            ' Перейти на страницу управления файлами,
            Server.Transfer("FileManager.aspx")
        Catch ex As System.UnauthorizedAccessException
            Server.Transfer("NotAuthorized.aspx")
      End Try
     End If
End Sub
```

Visual C#

```
private void butCreate_Click(object sender, System.EventArgs e)
{
        string strPath;
    // Проверить, существует ли каталог.
    strPath = Server.MapPath(Request.ApplicationPath) + "\\"
      + txtName.Text;
      if (Directory.Exists(strPath))
      {
          // Предложить пользователю выбрать другое имя.
          HtNameExists.Text = "<p>The name " + txtName.Text +
            " already exists. Please choose a different one.</p>";
          return;
      }
      else
      {
        try
        {
            // Создать каталог.
            Directory.CreateDirectory(strPath);
            // Создать и присвоить значение переменной
            // состояния Session.
            Session["Path"] = strPath;
            // Перейти на страницу управления файлами.
            Server.Transfer("FileManager.aspx");
        }
            catch (System.UnauthorizedAccessException ex)
        {
            Server.Transfer("NotAuthorized.aspx");
        }
    }
}
```

Упражнение 3. Создание формы File Manager



Ваша задача — создать Web-форму, которая позволит пользователям создавать, открывать и удалять файлы в своих каталогах на сервере. Для этой Web-формы потребуются привязка к данным, потоковая разметка, переход между формами, а также пространства имен *QueryStrings* и *System.IO* для файлового доступа. Готовая форма выглядит, как на рис. 4-24.

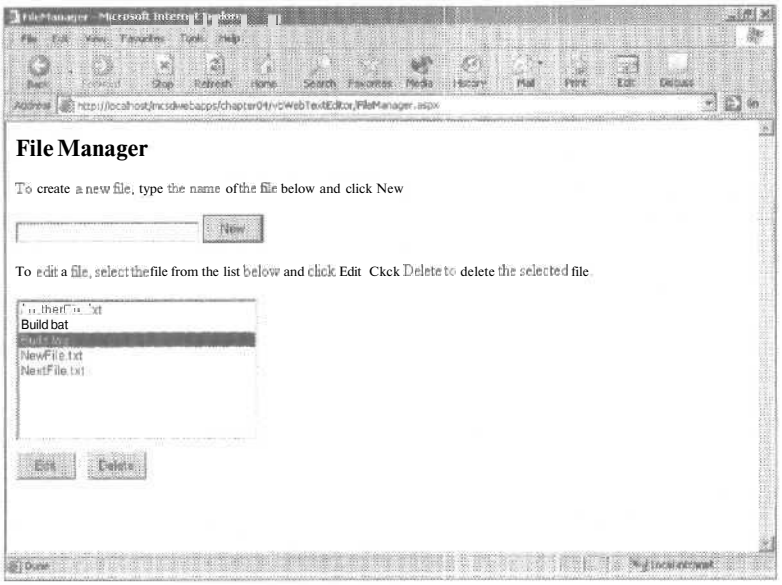


Рис. 4-24. Готовая Web-форма FileManager.aspx

▶ Создание Web-формы FileManager.aspx

1. Создайте новую форму с именем FileManager.aspx.
2. В окне Properties выберите объект *DOCUMENT* и установите его свойство *pageLayout* как *FlowLayout*.
3. Добавьте элементы управления и задайте их свойства, как показано в следующей таблице.

Элемент управления	Свойство	Значение
Literal	ID	litNoFile
Text Box	ID	txtNewFile
Button	ID	butNew
	Text	New
Literal	ID	litNoneSelected
List Box	ID	lstFiles
	DataSource	strFiles
Button	ID	butEdit
	Text	Edit
Button	ID	butDelete
	Text	Delete

4. Переключите Web-форму в режим HTML и к ней добавьте HTML-теги, выделенные в показанном ниже тексте полужирным шрифтом:

```
<form id="Form1" method="post" runat="server">
<h2>
File Manager
</h2>
<P>
To create a new file, type the name of the file below and click New.
</p>
<asp:Literal id="litNoFile" runat="server"></asp:Literal>
<asp:TextBox id="txtNewFile" runat="server" Width="191px">
</asp:TextBox>
<asp:Button id="butNew" runat="server" Width="62px" Height="29px"
Text="New"></asp:Button>
<P>
To edit a file, select the file from the list below and click Edit. Click
Delete to delete the selected file.
</p>
<asp:Literal id="litNoneSelected" runat="server"></asp:Literal>
<asp:listbox id="lstFiles" runat="server" Width="252px" Height="161px"
DataSource="<%# strFiles %>"></ asp:listbox>
<br>
<asp:Button id="butEdit" runat="server" Width="62" Height="29"
Text="Edit"></asp:Button>
  
<asp:Button id="butDelete" runat="server" Width="62px" Height="29px"
Text="Delete"></asp:Button>
</form>
```

5. Закончив редактирование Web-формы, переключитесь в режим Design и дважды щелкните New, чтобы вызвать окно Code. Поместите перед определением класса Web-формы следующий оператор:

Visual Basic .NET

```
Imports System.IO
```

Visual C#

```
using System.IO;
```

6. Добавьте к обработчикам событий формы следующий код:

Visual Basic .NET

```
Public strPath As String, strFiles As String()
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Получить путь к файлу,
    strPath = Session("Path")
    ' Если это событие не инициирует возврат страницы...
    If Not IsPostBack Then
```

```
        Dim iCount As Integer
        ' Получить список файлов в текущем каталоге.
        strFiles = Directory.GetFiles(strPath)
        ' Получить имена файлов.
        For iCount = 0 To UBound(strFiles)
            strFiles(iCount) = Path.GetFileName(strFiles(iCount))
        Next
    End If
    ' Привязать lstFiles к массиву с именами файлов.
    lstFiles.DataBind()
End Sub

Private Sub butNew_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butNew.Click
    ' Если выбран файл, запустить редактор,
    If txtNewFile.Text <> "" Then
        Response.Redirect("EditFile.aspx?file=" & txtNewFile.Text)
    Else
        ' в противном случае показать сообщение.
        litNoFile.Text = "<p>You must enter the name of a file" & _
        "" to create.<p>"
    End If
End Sub

Private Sub butEdit_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butEdit.Click
    ' Если выбран файл, запустить редактор,
    If Not IsNothing(lstFiles.SelectedItem) Then
        Response.Redirect("EditFile.aspx?file=" & _
          lstFiles.SelectedItem.ToString)
    Else
        ' в противном случае показать сообщение.
        litNoneSelected.Text = "<p>You must select a file.<p>"
    End If
End Sub
Private Sub butDelete_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles butDelete.Click
    ' Если выбран файл, запустить редактор,
    If Not IsNothing(lstFiles.SelectedItem) Then
            Try
                Dim filToDelete As File
    filToDelete.Delete(strPath & "\" & _
      lstFiles.SelectedItem.ToString)
                lstFiles.Items.Remove(lstFiles.SelectedItem)
            Catch ex As System.Security.SecurityException
                Server.Transfer("NotAuthorized.aspx")
            End Try
    Else
```

```
        ' в противном случае показать сообщение.
        litNoneSelected.Text = "<p>You must select a file.<p>"
    End If
End Sub
```

Visual C#

```
string strPath;
oublic string[] strFiles;

private void Page_Load(object sender, System.EventArgs e)
{
    // Получить путь к файлу,
    strPath = Session["Path"].ToString();
    // Если это событие не инициирует возврат страницы...
    if(!Page.IsPostBack)
    {
        // Получить список файлов в текущем каталоге.
        strFiles - Directory.GetFiles(strPath);
        // Получить имена файлоа.
        for (int iCount = 0; iCount <= strFiles.GetUpperBound(0);
              iCount++)
            strFiles[iCount] = Path.GetFileName(strFiles[iCount]);
    }
// Привязать lstFiles к массиву с именами файлов.
lstFiles.DataBind();
}
private void butNew_Click(object sender, System.EventArgs e)
{
    // Если выбран файл, запустить редактор,
    if (!(txtNewFile.Text == ""))
        Response.Redirect("EditFile.aspx?file=" +
              txtNewFile.Text);
else
    // в противном случае показать сообщение.
    litNoFile.Text = "<p>You must enter the name of a file " +
      "' to create.<p>";
}

private void butEdit_Click(object sender, System.EventArgs e)
{
  // Если выбран файл, запустить редактор,
  if (lstFiles.SelectedItem != null)
      Response.Redirect("EditFile.aspx?file=" +
        lstFiles.SelectedItem.ToString());
  else
      // в противном случае показать сообщение.
      litNoneSelected.Text - "<p>You must select a file.<p>";
```

```
}

private void butDelete_Click(object sender, System.EventArgs e)
{
// Если выбран файл, запустить редактор,
if (lstFiles.SelectedItem != null)
    try
    {
        File.Delete(strPath + "\\" +
          lstFiles.SelectedItem.ToString());
        lstFiles.Items.Remove(lstFiles.SelectedItem);
    }
    catch(System.UnauthorizedAccessException ex)
    {
        Server.Transfer("NotAuthorized.aspx");
    }
else
    // в противном случае показать сообщение.
    litNoneSelected.Text = "<p>You must select a file.<p>";
}
```

Упражнение 4. Создание формы File Editor



Сейчас вы создадите Web-форму для редактирования файлов, хранящихся в каталоге пользователя на сервере. Эта Web-форма использует ту же разметку и методики программирования, что и Web-форма из упражнения 3. Готовая форма выглядит, как на рис. 4-25.

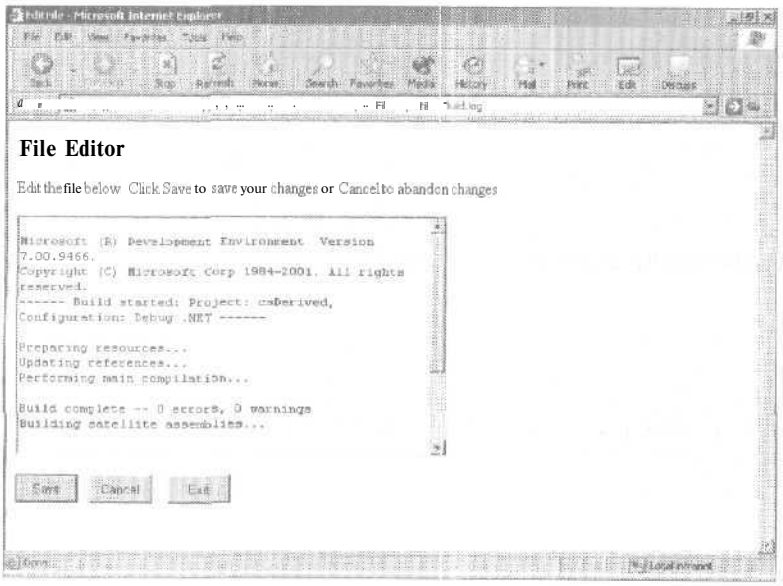


Рис. 4-25. Web-форма EditFile.aspx

► **Создание Web-формы EditFile.aspx**

1. Создайте новую форму с именем EditFile.aspx.
2. В окне Properties выберите объект *DOCUMENT* и установите его свойство *pageLayout* в *FlowLayout*
3. Добавьте элементы управления и задайте их свойства, как показано в следующей таблице.

Элементуправления	Свойство	Значение
Text Box	ID	txtEditFile
	TextMode	Multiline
Button	ID	butSave
	Text	Save
HTML Button	id	butCancel
	value	Cancel
Button	ID	butExit
	Text	Exit

4. Переключите Web-форму в режим HTML и добавьте HTML-теги, выделенные в показанном ниже тексте **полужирным шрифтом:**

```
<form id="Form1" method="post" runat="server">
<h2>
File Editor
</h2>
<p>
Edit the file below. Click Save to save your changes or Cancel to abandon changes.
</p>
<asp:TextBox id="txtEditFile" runat="server" Width="448px" Height="253px"
TextMode="MultiLine"></ asp:TextBox>
<p>
</p>
<asp:Button id="butSave" runat="server" Width="65" Height="30"
Text="Save"></asp:Button>
   <INPUT id="butCancel" onclick="history.back()" type="button"
value="Cancel" style="WIDTH: 65px; HEIGHT: 30px">
  
<asp:Button id="butExit" runat="server" Width="65" Height="30"
Text="Exit"></asp:Button>
</form>
```

5. Закончив редактирование Web-формы, переключитесь в режим Design и дважды щелкните Save, чтобы вызвать окно Code. Поместите перед определением класса Web-формы следующий оператор:

Visual Basic .NET

```
Imports System.IO
```

Visual C#

```
using System.IO;
```

6. **Добавьте к обработчикам событий формы следующий код:**

Visual Basic .NET

```
Dim strPath, strFile As String
Private Sub Page_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    ' Получить имя файла и путь к нему.
    strPath = Session("Path")
    strFile = Request.QueryString("file")
    ' Если это событие не инициирует возврат страницы...
    If Not IsPostBack Then
            Try
                ' Открыть файл.
                strmEditFile = File.OpenText(strPath & "\" & _
                  strFile)
                ' Прочитать текст из файла.
                txtEditFile.Text = strmEditFile.ReadToEnd.ToString
                ' Закрыть файл.
                strmEditFile.Close()
            Catch ex As System.IO.FileNotFoundException
                ' Если такого файла нет, создать его.
                File.CreateText(strPath + "\\"
                + strFile).Close()
                ' Очистить текстовое поле.
                txtEditFile.Text = ""
            Catch ex As System.UnauthorizedAccessException
                Server.Transfer("NotAuthorized.aspx")
            End Try
    End If
   End Sub

Private Sub butExit_Click( ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butExit.Click
    If viewstate("changed") Then
        SaveFile()
    End If
    ' Вернуться на страницу управления файлами,
    Response.Redirect("FileManager.aspx")
End Sub

Private Sub txtEditFile_TextChanged(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles txtEditFile.TextChanged
    ' Установить флаг изменения файла.
    viewstate("changed") = True
```

```
End Sub

Private Sub butSave_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butSave.Click
    SaveFile()
End Sub

Private Sub SaveFileO
    Dim strmEditWrite As StreamWriter
    ' Удалить файл.
    File.Delete(strPath & strFile)
    ' Создать файл с новым текстом.
    strmEditWrite = File.CreateText(strPath & "\" & strFile)
    strmEditWrite.Write(txtEditFile.Text)
    strmEditWrite.Close()
    ' Сбросить флаг изменения файла.
    viewstate("changed") = False
End Sub
```

Visual C#

```
// Перед определением класса Web-формы необходимо
// импортировать пространство имен System. IO
using System.IO;
string strPath;
string strFile;

private void Page_Load(object sender, System.EventArgs e)
{
    // Получить имя файла и путь к нему.
    Session["Path"] = "c:";
    strPath = Session["Path"].ToString();
    strFile - Request.QueryString["file"];
    // Если это событие не инициирует возврат страницы...
    if (! Page.IsPostBack)
    {
    StreamReader strmEditFile;
    try
    {
        // Открыть файл.
        strrnEditFile = File.OpenText(strPath + "\\" +
          strFile);
        // Прочитать текст из файла.
        txtEditFile.Text =
          strmEditFile.ReadToEnd().ToString();
        // Закрыть файл.
        strmEditFile.Close();
```

```
    }
        catch (FileNotFoundException ex)
    {
        // Если такого файла нет, создать его.
        File.CreateText(strPath + "\\" +
          strFile).Close();
        // Очистить текстовое поле.
        txtEditFile.Text = "";
    }
    catch(System.UnauthorizedAccessException ex)
    {
        Server.Transfer("NotAuthorized.aspx");
    }
}

private void butExit_Click(object sender, System.EventArgs e)
{
    if (ViewState["Changed"].ToString() == "true")
        SaveFile();
    // Вернуться на страницу управления файлами,
    Response.Redirect("FileManager.aspx");
}

private void txtEditFile_TextChanged(object sender,
    System.EventArgs e)
{
    // Установить флаг изменения файла.
    ViewState["Changed"] = true.ToString();
}

private void butSave_Click(object sender, System.EventArgs e)
{
    SaveFileO;
}
void SaveFile()
{
    StreamWriter strmEditWrite;
    // Удалить файл.
    File.Delete(strPath + strFile);
    // Создать файл с новым текстом.
    strmEditWrite = File.CreateText(strPath + "\\" + strFile);
    strmEditWrite.Write(txtEditFile.Text);
    strmEditWrite.Close();
    // Сбросить флаг изменения файла.
    ViewState["Changed"] = false.ToString();
}
```

Упражнение 5. Создание Web-формы Not Authorized



Как говорилось в начале практикума, учетная запись ASPNET должна обладать правами на создание и удаление файлов в корневом каталоге приложения, в противном случае любая попытка создания, сохранения или удаления файла вызовет исключение System.UnauthorizedAccessException.

Код, показанный в предыдущих упражнениях, перехватывает это исключение при помощи блоков обработки исключений *Try...Catch/try...catch* и отображает Web-форму с указаниями по устранению ошибки (см. рис. 4-26).

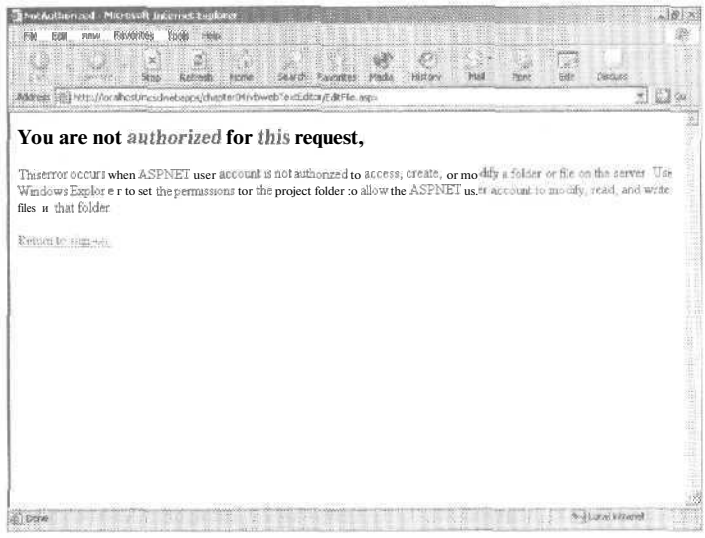


Рис. 4-26. Web-форма Not Authorized.aspx

▶ **Создание новой Web-формы Not Authorized. aspx**

1. Создайте новую форму с именем NotAuthorized.aspx.
2. Переключитесь в режим HTML и добавьте к Web-форме следующий HTML-текст:

```
<h2>You are not authorized for this request.</h2>
<P>
This error occurs when ASPNET user account is not authorized to access,
create, or modify a folder or file on the server. Use Windows
Explorer to set the permissions for the project folder to allow
the ASPNETuser account to modify, read, and
write files to that folder.
</p>
<a href="SignOn.aspx">Return to sign-on.</a>
```

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. В чем главное отличие между серверным элементом управления Button и одноименным HTML-элементом управления?

2. Как заставить несколько элементов управления RadioButton, размещенных на Web-форме взаимодействовать, чтобы одновременно удавалось установить только один из них?

3. Почему ASP.NET проверяет вводимые данные как на клиентской, так и на серверной стороне?

4. Какие верификаторы следует использовать, чтобы убедиться, что клиент ввел допустимый номер телефона?

5. Найдите ошибку в следующем коде:

Visual Basic .NET

```
Server.Transfer("Default.htm")
```

Visual C#

```
Server.Transfer("Default.htm");
```

6. Почему нельзя открыть новое окно браузера из серверного кода?

ГЛАВА 5

Хранение и извлечение данных с помощью ADO.NET

Занятие 1. Доступ к данным с помощью ADO.NET 192

Занятие 2. Использование наборов данных в Web-формах 209

Занятие 3. Обработка транзакций 223

В этой главе

Из этой главы вы узнаете, как при помощи ADO.NET получать доступ из Web-форм к базам данных (БД) SQL и OLE и модифицировать хранящиеся в них данные. Вы познакомитесь с инструментами для работы с данными, имеющимися в Visual Studio .NET, и научитесь применять их для создания соединений, извлечения записей и исполнения различных команд в БД. Вы также научитесь отображать данные на Web-формах с помощью привязки данных и других программных методов. В завершение вы узнаете, как поддерживать целостность БД при помощи транзакций.

Прежде всего

Для изучения материалов этой главы необходимо:

- установить Microsoft Data Engine (MSDE — устанавливается вместе с Visual Studio .NET при выборе установки по умолчанию) либо Microsoft SQL Server 2000;
- установить с прилагаемого компакт-диска **БД** Contacts (Contacts.mdf) — она потребуется для примеров из этой главы;
- знать основы языка Transact-SQL. Для получения сведений о SQL обратитесь к разделу «Transact-SQL Language Reference» справочной системы SQL Server Books Online.

Совет Ознакомительную версию Microsoft SQL Server 2000 можно бесплатно загрузить с Web-сайта Microsoft *(http://vfww.microsoft.com/sqf)*.

Занятие 1. Доступ к данным с помощью ADO.NET

Visual Studio .NET обеспечивает доступ к БД с помощью набора инструментов и пространств имен, в совокупности именуемых ADO.NET. Благодаря стандартизации доступ к данным в ADO.NET практически не зависит от источника данных: после подключения к БД для работы с ней используются стандартный набор объектов, свойств и методов для любого типа БД.

Из этого занятия вы узнаете, как с помощью ADO.NET подключится к БД, а также как читать, модифицировать и удалять записи БД. Во всех примерах этой главы используется БД Contacts (Contacts.mdf), которую вы найдете на прилагаемом к книге компакт-диске. Обязательно установите ее, прежде чем выполнять следующие упражнения.

Изучив материал этого занятия, вы сможете:

- ✓ понять принцип получения доступа к данным через ADO.NET и узнаете, как его осуществить;
- ✓ подключаться к БД и создавать наборы данных в период разработки;
- ✓ объяснить, как ADO.NET обращается к элементам данных, используя информацию о типах;
- ✓ добавлять, удалять и модифицировать записи в наборе данных;
- ✓ записывать в БД содержимое набора данных;
- ✓ подключаться к БД и создавать наборы данных в период выполнения.

Продолжительность занятия - около 35 минут.

Знакомство с ADO.NET

В ADO.NET существует три уровня доступа к данным:

- физическое хранилище данных — им может быть БД OLE, БД SQL либо XML-файл;
- провайдер данных — состоит из объекта Connection и объектов-команд, которые создают представление данных в памяти;
- набор данных — это представление в памяти таблиц и связей, с которыми работает приложение.

Уровень провайдера данных абстрагирует набор данных, используемый программой, от физического хранилища. После создания набора данных ни его происхождение, ни физическое размещение уже не имеет значения. Поскольку здесь набор данных не зависит от хранилища, такая архитектура называется *отсоединенной* (disconnected).

На рис. 5-1 показана модель объектов ADO.NET в действии.

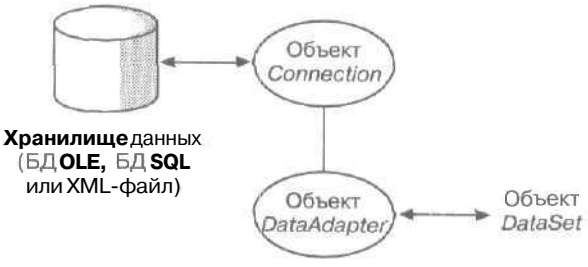


Рис. 5-1. ***Connection*** **и другие объекты ADO.NET**

В ADO.NET предусмотрены два типа соединений с БД:

- **соединения с локальными БД (устанавливаются через объект *OleDbConnection*).** Соединения с БД OLE используют для исполнения команд и получения данных объект *OleDbDataAdapter*;
- **соединения с серверными БД (устанавливаются через объект *SqlConnection*).** Соединения с БД SQL используют для выполнения команд и получения данных объект *SqlDbDataAdapter*.

Помимо соединений с БД, для получения данных применяются XML-файлы, к которым удается обращаться напрямую из наборов данных при помощи методов *ReadXML* и *WriteXML* объекта *DataSet*. XML-файлы — это статические представления наборов данных. Все данные ADO.NET передает через Интернет в формате XML.

Доступ к объектам, свойствам и методам ADO.NET предоставляют три пространства имен, описанных в таблице 5-1. Пространства имен *System.Data.SqlClient* и *System.Data.OleDb* предоставляют аналогичные возможности для работы соответсвенно с БД SQL и OLE.

Таблица 5-1. Пространства имен ADO.NET

Имя	Что поддерживает
System.Data	Классы, типы и сервисы для создания наборов данных и доступа к ним и их дочерним объектам
System.Data.SqlClient	Классы и типы для доступа к БД Microsoft **SQL** Server
System.*Data.OleDb*	Классы и типы для доступа к БД **OLE**

Чтобы получить возможность работы с БД в программе, необходимо поместить в начале модуля кода следующие операторы:

Visual Basic .NET

```
Imports System.Data
' Этот оператор нужен для подключения к БД SQL.
Imports System.Data.SqlClient
' Этот оператор нужен для подключения < БД OLE.
Imports System.Data.OleDb
```

Visual C#

```
using System.Data
// Этот оператор нужен для подключения к БД SQL.
using System.Data.SqlClient
// Этот оператор нужен для подключения к БД OLE.
using System.Data.OleDb
```

Чтобы получить доступ к БД с помощью ADO.NET, выполните следующие действия.

1. Создайте соединение с БД посредством объекта соединения,
2. Исполните команду, создающую объект *DataSet* при помощи объекта адаптера.
3. Задействуйте объект *DataSet* в программе для отображения данных или модификации БД.
4. Исполните команду, записывающую данные из *DataSet* обратно в БД при помощи объекта адаптера.

5. Закройте соединение с БД при помощи метода *Open,* если оно было явно открыто на этапе 2. Исполнение каждой команды, вызванной без предварительного вызова метода *Open,* сопровождается неявным открытием и закрытием (после завершения команды) соединения с БД.

В следующих разделах эти действия обсуждаются более подробно.

Подключение к БД

Для подключения к БД в Visual Studio предназначено окно Server Explorer.

Чтобы установить соединение с БД в среде Visual Studio, выполните следующие действия.

1. Выберите из меню View элемент Server Explorer — Visual Studio откроет окно Server Explorer.
2. В окне Server Explorer щелкните Connect To Database, в результате Visual Studio .NET откроет диалоговое окно DataLink Properties, показанное на рис. 5-2.

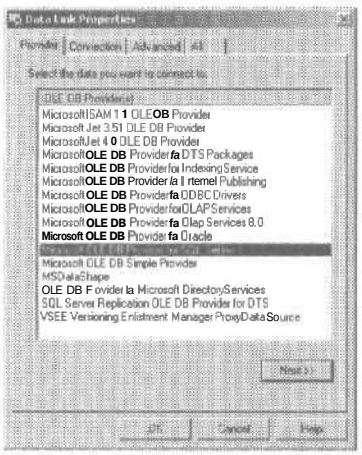


Рис. 5-2. **Диалоговое окно DataLink Properties**

3. Щелкните вкладку Provider, чтобы указать тип БД для подключения. Провайдером БД по умолчанию является Microsoft OLE DB Provider for SQL Server. Он подходит для работы с БД Microsoft SQL Server, а для доступа к БД другого типа следует выбрать соответствующий провайдер. Например, чтобы напрямую подключиться к БД Microsoft Access 2000, укажите Microsoft Jet 4.0 OLE DB Provider.
4. Щелкните вкладку Connection, чтобы задать БД, к которой нужно подключиться. Если на этапе 3 выбрать другой провайдер БД, параметры соединения также изменятся (рис. 5-3).
5. Введите в поле имени сервера **(local)\NETSDK** (рис. 5.3). Щелкните Attach database file as a database name и введите **Contacts** в первое поле, а во втором укажите путь к файлу Contacts.mdf (он должен быть в каталоге C:\MCSDWebApps\Chapter05\). Щелкните кнопку Test Connection, чтобы проверить правильность ввода. После успешного завершения проверки щелкните OK. Visual Studio добавит соединение с источником данных в окно Server Explorer, как показано на рис. 5-4.
6. Щелкая знаки «+», можно раскрывать узлы дерева в окне Server Explorer. Для просмотра таблиц источника данных раскройте узел соединения, затем узел Tables — Visual Studio отобразит список таблиц, как показано на рис. 5-5.

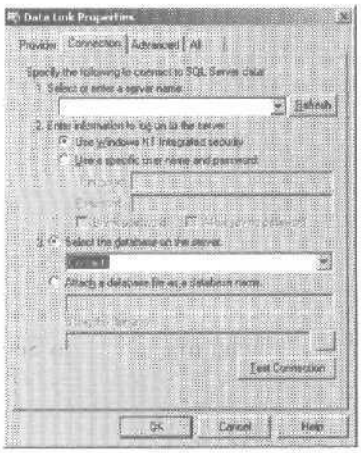


Рис. 5-3. **Вкладка Connection**

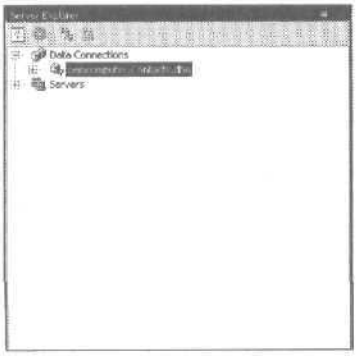


Рис. 5-4. Окно Server Explorer

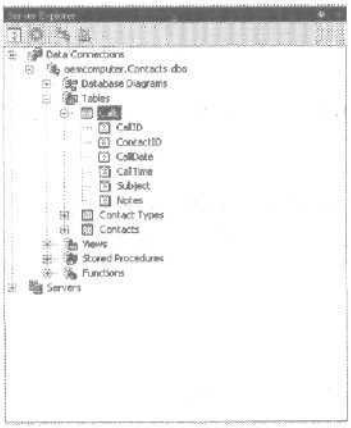


Рис. 5-5. **Раскрытые узлы в окне Server Explorer**

7. Чтобы добавить к приложению элемент данных, перетащите выбранный элемент из окна Server Explorer на Web-форму. Например, если перетащить на Web-форму таблицу

Contacts (рис. 5-5), Visual Studio создаст объекты соединения и адаптера с соответствующими параметрами (рис. 5-6).

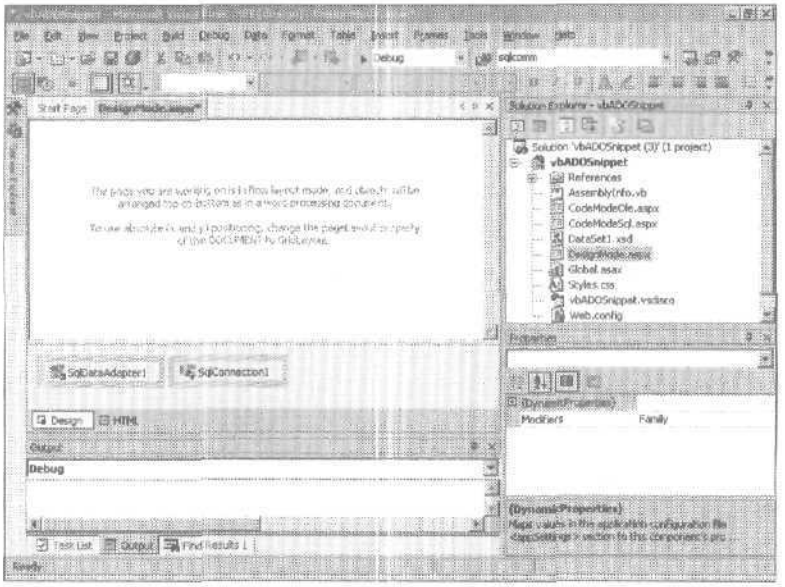


Рис. 5-6. Объект-соединение и объект-адаптер

Внимание! Web-приложения работают под учетной записью ASPNET. Прежде чем ваше Web-приложение попытается получить доступ к БД SQL, администратор БД SQL должен создать эту учетную запись и установить для нее соответствующие разрешения. Для работы с БД Access необходимо задать для учетной записи ASPNET разрешения на доступ к файлу БД с помощью параметров безопасности доступа к файлам Windows.

Создание набора данных

Для создания набора данных используются объекты соединения и адаптера, созданные на этапе 7 описанной в предыдущем разделе последовательности действий.

Чтобы создать набор данных в режиме Design, выполните следующие действия.

1. Щелкните правой кнопкой объект адаптера и выберите из контекстного меню команду Generate Dataset — Visual Studio откроет диалоговое окно Generate Dataset, показанное на рис. 5-7.
2. Укажите таблицы для добавления к набору данных и щелкните OK. Visual Studio создаст новый объект *DataSet* и добавит его к Web-форме.

Для просмотра данных набора в режиме Design щелкните правой кнопкой объект *DataSet,* затем выберите из контекстного меню команду View Schema. Visual Studio отобразит набор данных в окне **XML** Designer (рис. 5-8).

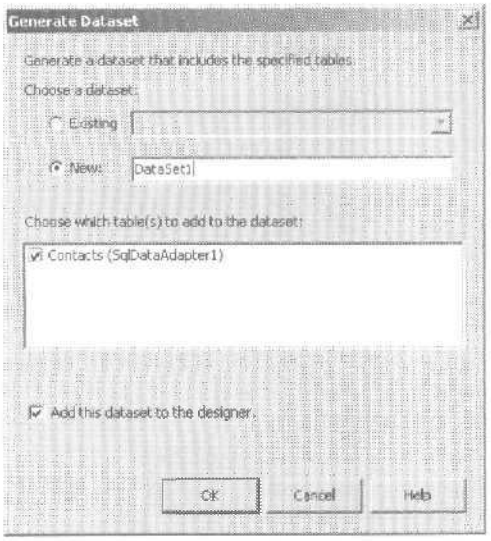


Рис. 5-7. Диалоговое окно Generate Dataset

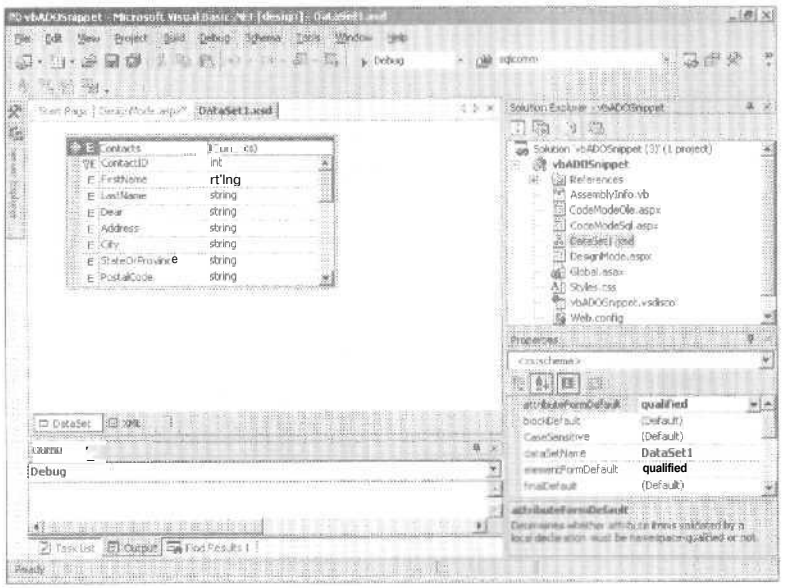


Рис. 5-8. **Просмотр содержимого набора данных в режиме Design**

Отображение набора данных

Для отображения набора данных на Web-форме во время выполнения вам надо проделать следующее.

1. Добавьте к Web-форме элемент управления, в котором будут отображаться данные, например DataGrid.
2. Сделайте набор данных источником данных для этого элемента управления. В случае DataGrid для этого следует щелкнуть ссылку Property Builder в окне Properties Window

и занести в свойство *DataSource* этого элемента управления имя объекта *DataSet*, *а в* свойство *DataMember* — имя таблицы из набора данных, как показано на рис. 5-9.

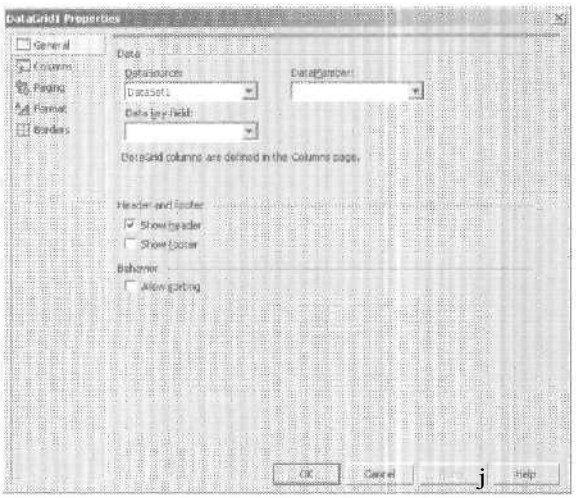


Рис. 5-9. Связывание элемента управления DataGrid с данными

3. Залайте столбцы, которые будут отображаться в элементе управления. В случае DataGrid для этого необходимо щелкнуть элемент Columns в диалоговом окне Properties, снять флажок Create Columns Automatically At Run Time и добавить столбцы для отображения из списка Available Columns, как показано на рис. 5-10. Закончив добавление столбцов, щелкните OK.

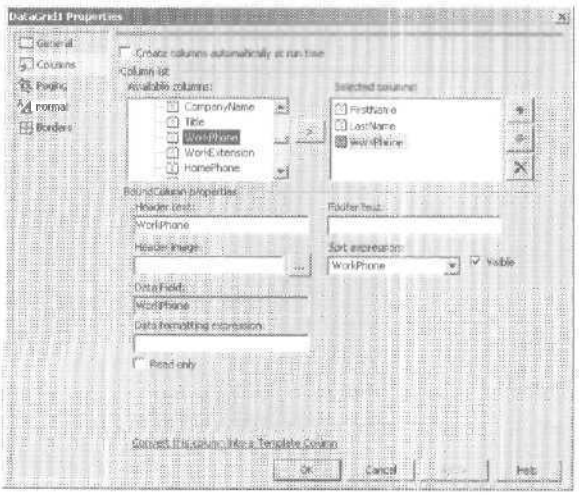


Рис. 5-10. Добавление столбцов для отображения в DataGrid

4. Добавьте к обработчику события *Page_Load* Web-формы код, который заполняет набор данными из объекта адаптера и привязывает элемент управления, отображающий данные, к объекту *DataSet*. Например, следующий код отображает данные посредством созданного ранее элемента управления DataGrid:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
        ' Заполнить набор данных,
        SqlDataAdapter1.Fill(DataSet1)
        ' Обновить DataGrid.
        DataGrid1.DataBind()
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
        // Заполнить набор данных.
        sqlDataAdapter1.Fill(DataSet1);
        // Обновить DataGrid.
        DataGrid1.DataBind();
}
```

5. Запустите приложение. После загрузки страницы на ней отображается набор данных, как показано на рис. 5-11.

Совет Поскольку заполнение набора данных и его привязка к элементу управления DataGrid осуществляется во время обработки события *Page_Load*, сведения о состоянии элемента управления DataGrid можно не сохранять. Отключение сведений о состоянии повышает производительность, так как при этом отпадает необходимость сохранять данные DataGrid между запросами в страничном поле *ViewState*. Чтобы отключить сохранение сведений о состоянии для элемента управления, установите его свойство *EnableViewState* в *False*.

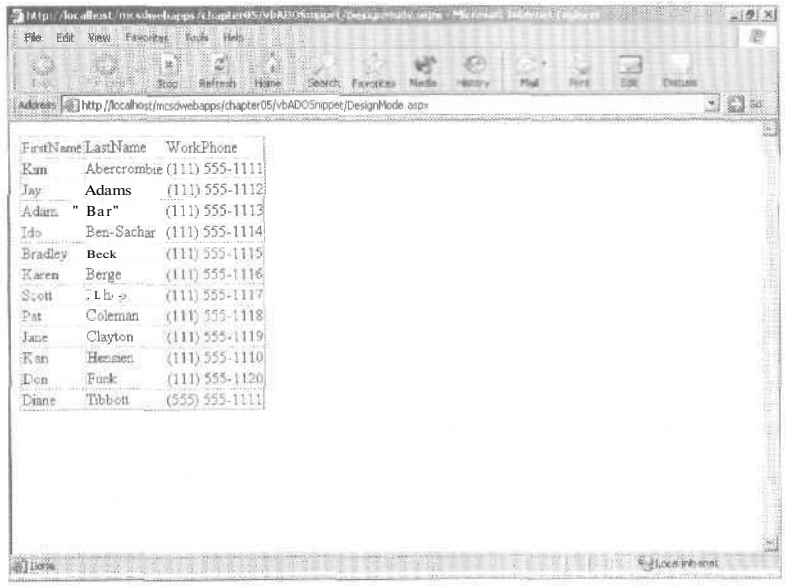


Рис. 5-11. Вид DataGrid во время выполнения

Модификация записей в базе данных

DataSet — это центральный объект ADO.NET. Как правило, любые операции добавления, удаления или модификации записей осуществляются с его помощью.

Для модификации записей БД с помощью объекта *DataSet* выполните следующие действия.

1. Получите объект *DataSet*, как описано в предыдущих разделах.
2. Внесите нужные изменения в полученный объект *DataSet*.
3. Запишите в БД содержимое *DataSet*, вызвав метод *Update* объекта адаптера данных.

Наборы *Tables*, *Rows* и *Columns* служат для извлечения соответствующих элементов данных и их записи в объект *DataSet* (рис. 5-12).

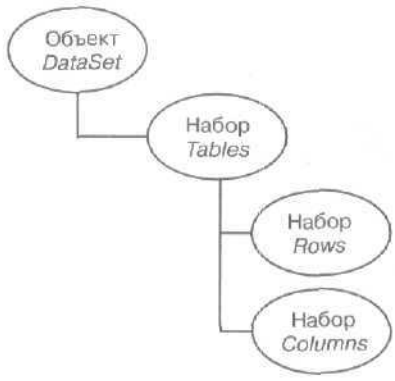


Рис. 5-12. Наборы объекта *DataSet*

Обращение к объектам в ADO.NET

Создание объектов соединения, адаптера и набора данных в режиме Design активирует типизацию данных этих объектов. Это означает, что на таблицы, строки и поля можно ссылаться по их именам в схеме БД. Какое существенное усовершенствование по сравнению с ADO, где поддерживались только нетипизированные ссылки на объекты данных!

Ниже показаны эквивалентные фрагменты кода, демонстрирующие в сравнении типизированные и нетипизированные ссылки на объект из БД:

Visual Basic .NET

```
' Типизированная ссылка на столбец HomePhone таблицы Contacts.
DataSet1.Contacts.HomePhoneColumn.Caption - "@Home"
' Нетипизированная ссылка на столбец HomePhone таблицы Contacts.
DataSet1.Tables("Contacts").Columns("HomePhone").Caption = "@Home"
```

Visual C#

```
// Типизированная ссылка на столбец HomePhone таблицы Contacts.
DataSet1.Contacts. HornePhoneColumn, Caption = "@Home";
// Нетипизированная ссылка на столбец HomePhone таблицы Contacts.
DataSet1.Tables["Contacts"].Columns["HomePhone"].Caption = "@Home";
```

Первый вариант этого кода не только понятнее и проще, но и менее подвержен ошибкам, поскольку Visual Studio проверяет типы данных во время работы и помечает опечатки как нераспознанные ссылки.

Как правило, при работе с объектами данных следует использовать типизированные ссылки. Исключение составляют лишь случаи, когда объект данных заранее не известен. Обычно такая ситуация возникает, когда источник данных определяется во время выполнения, а не во время разработки.

Сведения о типах объектов данных берутся из схемы XML, генерируемой Visual Studio при создании набора данных в режиме Design.

Для просмотра сведений о типах набора данных щелкните его правой кнопкой в окне Design и выберите из контекстного меню команду View Schema. В результате Visual Studio откроет набор данных в окне XML Designer, как показано на рис. 5-13.

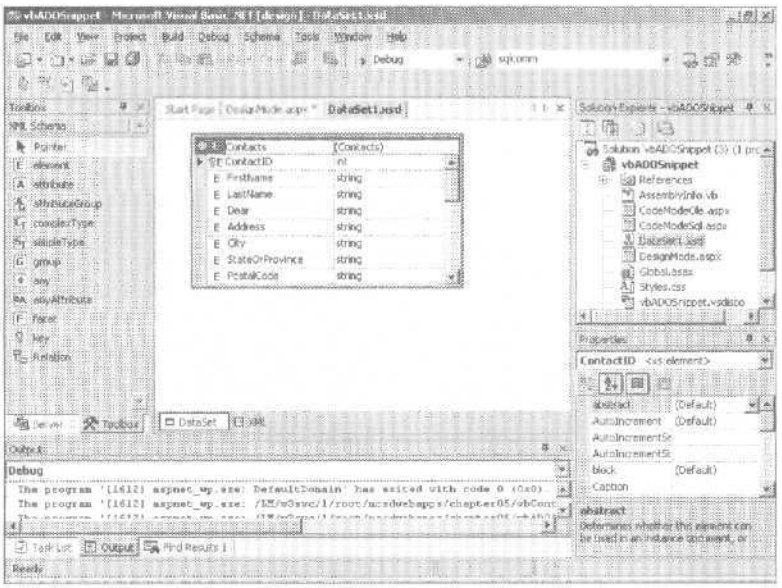


Рис. 5-13. Просмотр сведений о типах

При помощи XML Designer можно добавлять элементы, задавать уникальные ключи, а также просматривать и изменять типы данных элементов набора данных.

Добавление, модификация и удаление строк

Для добавления, модификации и удаления строк объекта *Table,* составляющего объект *DataSet*, служит набор *Rows.* Чтобы добавить запись к набору данных, нужно создать новый объект *Row* и добавить его к набору *Rows* объекта *DataSet*, как показано в следующем фрагменте кода:

Visual Basic .NET

```
' В этом коде используются соединение, адаптер и набор данных, ' созданные в
режиме Design.
    Private Sub butAddRow_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles butAddRow.Click
        ' Создать объект "строка" для таблицы Contacts.
        Dim rowNew As DataSet1.ContactsRow =
DataSet1.Contacts.NewRow
        ' Заполнить поля строки.
```

```
        rowNew.ContactID = 42
        rowNew.FirstName = "Danielle"
        rowNew.LastName = "Tiedt"
        rowNew.WorkPhone = "(111) 555-1212"
        ' Добавить строку к набору данных.
        DataSet1.Contacts.Rows.Add(rowNew)
    End Sub
```

Visual C#

```
// В этом коде используются соединение, адаптер и набор данных.
// созданные в режиме Design.
private void butAddRow_Click(object sender, System.EventArgs e)
{
    // Создать объект "строка" для таблицы Contacts.
    DataSet1.ContactsRow rowNew =
        (DataSet1.ContactsRow)DataSet1.Contacts.NewRow();
    // Заполнить поля строки.
    rowNew.ContactID = 42;
    rowNew.FirstName = "Danielle";
    rowNew.LastName = "Tiedt";
    rowNew.WorkPhone = " (111) 555-1212";
}
```

Чтобы модифицировать строку набора данных, следует получить объект *Row* из объекта *Table* при помощи метода *FindBy*, а затем изменить значения полей строки. Этот прием демонстрирует следующий код:

Visual Basic .NET

```
' В этом коде используются соединение, адаптер и набор данных, созданные в
режиме Design.
    Private Sub butChangeRow_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles butChangeRow.Click
        ' Объявить объект "строка",
        Dim rowChange As DataSet1.ContactsRow
        ' Получить модифицируемую строку при помощи первичного ключа,
        rowChange = DataSet1.Contacts.FindByContactID(42)
        ' Изменить значение поля строки.
        rowChange.WorkPhone = "(111) 555-9000"
    End Sub
```

Visual C#

```
// В этом коде используются соединение, адаптер и набор данных, // созданные в
режиме Design.
   private void butChangeRow_Click(object sender, System.EventArgs e)
   {
        // Объявить объект "строка".
        DataSet1.ContactsRow rowChange;
```

```
        // Получить модифицируемую строку при помощи первичного ключа,
        rowChange = DataSet1.Contacts.FindByContactID(42);
        // Изменить значение поля строки.
        rowChange.WorkPhone = "(111) 555-9000";
    }
```

Чтобы удалить строку из набора данных, необходимо получить объект *Row* из объекта *Table* при помощи метода *FindBy*, а затем удалить строку, вызвав метод *Delete* объекта *Row*, как показано ниже:

Visual Basic .NET

```
' В этом коде используются соединение, адаптер и набор данных,
' созданные в режиме Design.
Private Sub butDeleteRow_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butDeleteRow.Click
    ' Объявить объект "строка".
    Dim rowDelete As DataSet1.ContactsRow
    ' Получить удаляемую строку.
    rowDelete = DataSet1.Contacts.FindByContactID(42)
    ' Удалить строку.
    rowDelete.Delete()
End Sub
```

Visual C#

```
// В этом коде используются соединение, адаптер и набор данных.
// созданные в режиме Design.
private void butDeleteRow_Click(object sender, System.EventArgs e)
{
    // Объявить объект "строка",
    DataSet1.ContactsRow rowDelete;
    // Получить удаляемую строку.
    rowDelete = DataSet1.Contacts.FindByContactID(42);
    // Удалить строку.
    rowDelete.Delete();
}
```

Запись содержимого объекта DataSet в базу данных

Для переноса в БД изменений, сделанных в *DataSet*, предусмотрен метод *Update* объекта адаптера. Обычно перенос модифицированных данных выполняется после обработки всех событий, генерированных элементами управления страницы, поэтому метод *Update* обычно вызывают при обработке события *Page_PreRender*, как показано ниже:

Visual Basic .NET

```
' В этом коде используются соединение, адаптер и набор данных,
' созданные в режиме Design.
Private Sub Page_PreRender(ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles MyBase.PreRender
```

```
    ' Перенести в БД изменения, сделанные в наборе данных.
    SqlDataAdapter1.Update(DataSet1)
End Sub
```

Visual C#

```
// В этом коде используются соединение, адаптер v набор данных,
// созданные в режиме Design.
   private void Page_PreRender(object sender, System.EventArgs e)
   {
        // Перенести в БД изменения, сделанные в наборе данных.
        sqlDataAdapter1.Update(DataSet1);
   }
```

При переносе изменений из набора данных в БД ADO.NET выполняет следующие действия:

1. проверяет свойство *RowState* объекта *DataRow*, чтобы определить, модифицирован ли набор данных. Это свойство может принимать одно из следующих значений: *Added*, *Deleted*, *Modified*, *Unchanged* или *Detached*;
2. вызывает свойства *InsertCommand*, *DeleteCommand* или *UpdateCommand* объекта-адаптера, чтобы внести в БД необходимые изменения;
3. сбрасывает свойство *RowState* модифицированного объекта *DataRow*, устанавливая его в *Unchanged*.

Свойства объекта-адаптера *InsertCommand*, *DeieteCommand* и *UpdateCommand* автоматически генерируются на основе другого свойства этого объекта, *SelectCommand*. Это происходит во время создания набора данных в режиме Design с использованием этого объекта адаптера. Каждое из этих свойств представляет объект *OleCommand* либо *SqlCommand*. Способ исполнения команды определяется следующими свойствами объекта, представляющего команду:

- *CommandText* — содержит оператор SQL или имя хранимой процедуры, выполняющей команду;
- *CommandType* — определяет способ исполнения команды, может иметь одно из следующих значений:
  - *Text* — исполняет команду как оператор SQL (это значение задано по умолчанию);
  - *StoredProcedure* — исполняет команду как хранимую процедуру;
  - *TableDirect* — возвращает таблицу целиком; это значение используется только с провайдерами OLE DB .NET.

Содержимое и способ модификации набора данных разрешается изменять, задав другие операторы SQL для указанных объектов-команд.

Для просмотра или модификации этих команд выполните следующие действия.

1. В окне Properties дважды щелкните левой кнопкой объект, представляющий команду, которую нужно изменить. В результате раскроется список свойств объекта-команды (рис. 5-14).
2. Выберите свойство *CommandText* и щелкните расположенную справа кнопку со знаком (...) — Visual Studio откроет диалоговое окно Query Builder, показанное на рис. 5-15.
3. В диалоговом окне Query Builder выберите столбцы, которые нужно включить в текст команды, пометив флажками соответствующие поля данных на панели таблицы, или введите оператор SQL прямо в панель команд Query Builder. После этого щелкните OK для того, чтобы закрыть диалоговое окно.

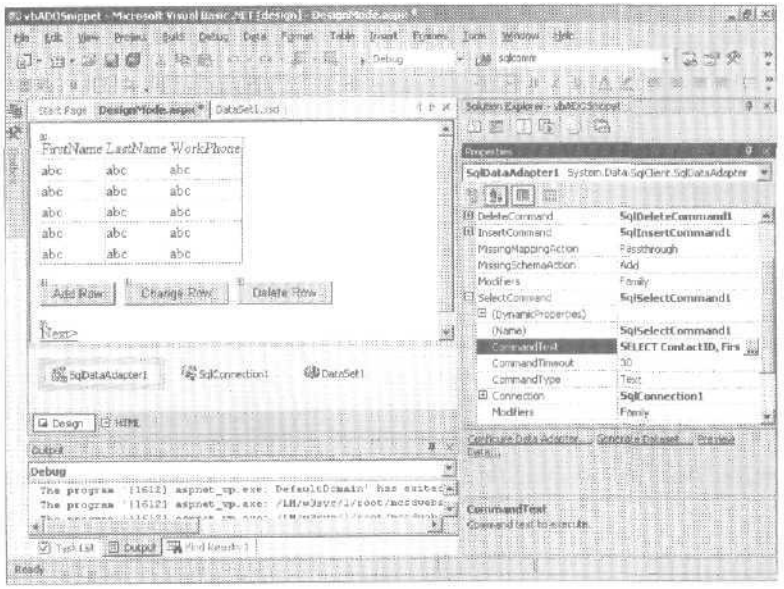


Рис. 5-14. Просмотр и модификация команд SQL

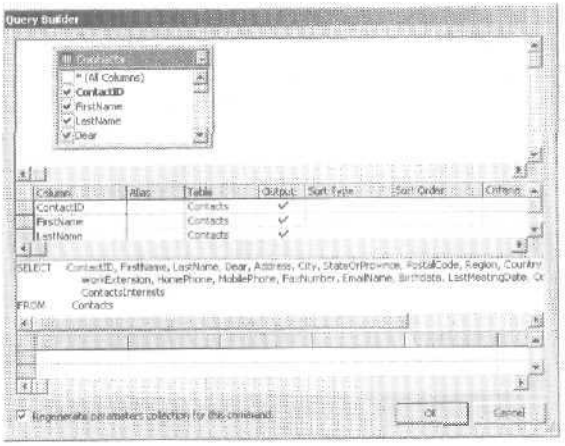


Рис. 5-15. Диалоговое окно Query Builder

Создание соединения с базой данных во время выполнения

Создание объектов данных в режиме Design — это замечательный способ изучения доступа к данным в ADO.NET, поскольку при этом Visual Studio автоматически генерирует значения свойств объектов-соединений и объектов-адаптеров. Значения некоторых свойств, например *ConnectionString*, очень трудно определить без помощи, которую предоставляет Visual Studio в режиме Design. Однако разрешается создать необходимое соединение в режиме Design, а затем перенести значения нужных свойств путем копирования-вставки, чтобы создать аналогичное соединение в коде.

Порядок использования объектов доступа к данным в коде не отличается от такового в режиме Design:

1. **создайте объект** соединения;
2. **создайте объект адаптера данных;**
3. **создайте объект набора данных;**
4. **при помощи методов объекта адаптера заполните и модифицируйте набор данных;**
5. **для отображения содержимого набора данных воспользуйтесь привязкой данных или другим методом.**

Например, следующий код создает объекты данных и отображает данные, которые предоставляет провайдер SQL:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles MyBase.Load
    ' (1) Создать соединение с источником данных.
    Dim sqlconnectContactMgmt As New SqlConnection _
      ("server=(local);database-Contacts;Trusted_Connection=yes")
    ' (2) Создать объект адаптера данных.
    Dim sqladaptContactMgmt As New _
      SqlDataAdapter("select - from Contacts", sqlconnectContactMgmt)
    ' (3) Создать набор данных.
    Dim dsContacts As New DataSet()
    ' (4) Заполнить набор данных.
    sqladaptContactMgmt.Fill(dsContacts, "Contacts")
    ' (5) Отобразить содержимое таблицы в DataGrid с помощью
    ' связывания данных.
    DataGrid1.DataSource = dsContacts.Tables("Contacts").DefaultView
    DataGrid1.DataBind()
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
     // (1) Создать соединение с источником данных.
     SqlConnection sqlconnectContactMgrnt = new
       SqlConnection("server=(local);
       database=Contacts;Trusted_Connection=yes");
     // (2) Создать объект адаптера данных.
     SqlDataAdapter sqladaptContactMgmt = new
       SqlDataAdapter("select * from Contacts", sqlconnectContactMgrnt);
     // (3) Создать набор данных.
     DataSet dsContacts = new DataSet();
     // (4) Заполнить набор данных
     sqladaptContactMgmt.Fill(dsContacts, "Contacts");
     // (5) Отобразить содержимое таблицы в DataGrid с помощью
      // связывания данных.
     DataGrid1.DataSource = dsContacts.Tables["Contacts"].DefaultView;
     DataGrid1.DataBind();
}
```

Обновление базы данных во время выполнения

Как говорилось ранее, в ADO.NET для записи содержимого набора данных в БД используются свойства объекта-адаптера *InsertCommand*, *DeleteCommand* и *UpdateCommand*. Если адаптер данных создается во время выполнения, эти свойства необходимо сгенерировать до вызова его метода *Update*.

Чтобы сгенерировать свойства *InsertCommand*, *DeleteCommand* и *UpdateCommand* во время исполнения, проделайте следующее.

1. Задайте свойство *SelectCommand* объекта-адаптера. На основе свойства *CommandText* объекта *SelectCommand* ADO.NET генерирует значения свойств *InsertCommand*, *DeleteCommand* и *UpdateCommand*.
2. Создайте объект генератора команд для объекта-адаптера. Поскольку этот объект зависит от провайдера данных, он существует в двух формах: *OleDbCommandBuilder* и *SqlCommandBuilder*.

Следующий код подключается к БД (1), создает набор данных (2), генерирует команды (3), модифицирует набор данных (4) и записывает модификации в БД (5):

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' (1) Создать соединение с источником данных.
    Dim ContactMgmt As New SqlConnection _
      ("server=(local);database=Contacts;Trusted_Connection=yes")
    ' Создать объект-адаптер данных.
    Dim adptContactMgmt As New _
      SqlDataAdapter("select * from Contacts", ContactMgmt)
    ' (2) Создать набор данных.
    Dim dsContacts As New DataSet()
    ' Задать команду SELECT.
    adptContactMgmt.SelectCommand.CommandText = _
      "SELECT * FROM Contacts"
    ' (3) Генерировать команды для добавления, удаления и
    ' обновления данных.
    Dim cmdContactMgmt As SqlCommandBuilder - New _
      SqlCommandBuilder(adptContactMgmt)
    ' (4) Создать новую строку.
    Dim rowInsert As DataRow = dsContacts.Tables("Contacts").NewRow
    ' Записать значения полей строки.
    rowInsert("ContactID") = 42
    rowInsert("FirstName") - "Danielle"
    rowInsert("LastName") - "Tiedt"
    rowInsert("WorkPhone") = "(555) 555-1212"
    ' Добавить строку к набору данных.
    dsContacts.Tables("Contacts").Rows.Add(rowInsert)
    ' (5) Записать данные в БД.
    adptContactMgmt.Update(dsContacts.Tables("Contacts"))
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
   // (1) Создать соединение с источником данных.
   SqlConnection ContactMgmt = new
   SqlConnection("server=(local);database=Contacts;Trusted_Connection=yes");
   // Создать объект-адаптер данных.
   SqlDataAdapter adptContactMgmt =
      new SqlDataAdapter("select * from
      Contacts", ContactMgmt);
   // (2) Создать набор данных.
   DataSet dsContacts = new DataSet();
   // Установить команду SELECT.
   adptContactMgmt.SelectCommand.CommandText =
      "SELECT * FROM Contacts";
   // (3) Генерировать команды для добавления, удаления и                    //
обновления данных.
   SqlCommandBuilder cmdContactMgmt = new
      SqlCommandBuilder(adptContactMgmt);
   // (4) Создать новую строку.
   DataRow rowInsert = dsContacts.Tables["Contacts"].NewRow();
   // Записать значения полей строки.
   rowInsert["ContactID"] = 42;
   rowInsert["FirstName"] = "Danielle";
   rowInsert["LastName"] = "Tiedt";
   rowInsert["WorkPhone"] = "(222) 555-1212";
   // Добавить строку к набору данных.
   dsContacts.Tables["Contacts"].Rows.Add(rowInsert);
   // (5) Записать данные в БД.
   adptContactMgmt.Update(dsContacts.Tables["Contacts"]);
}
```

Перед применением метода *Update* в вышеприведенном коде необходимо создать объект *SqlCommandBuilder*. Этот объект предоставляет значение для свойства *InsertCommand*, используемого объектом-адаптером для вставки в БД новой строки.

Занятие 2. Использование наборов данных в Web-формах

Большинство серверных элементов управления поддерживают *привязку данных* (data binding), которая позволяет в приложениях связывать данные, такие как наборы данных, со свойствами элементов управления. В частности, конструкция серверных элементов DataGrid и DataList оптимизирована, чтобы упростить отображение содержимого наборов данных на Web-формах и сделать его эффективным.

Из этого занятия вы узнаете, как отображать набор данных на Web-форме с помощью DataGrid, DataList и других серверных элементов управления, научитесь извлекать записи из набора данных через Web-форму, а также исполнять над БД команды SQL из обработчиков событий элементов управления.

Изучив материал этого занятия, вы сможете:

- ✓ оперативно отображать наборы данных на Web-формах с помощью привязки данных;
- ✓ форматировать данные, отображаемые в DataGrid, DataList и других серверных элементах управления, связанных с данными;
- ✓ программно отображать содержимое набора данных в серверном элементе управления без использования привязки данных;
- ✓ извлекать записи из набора данных, отображаемого на Web-форме;
- ✓ исполнять над БД команды SQL, извлекающие значения или модифицирующие данные напрямую.

Продолжительность занятия — около 30 минут.

Отображение набора данных с помощью DataGrid

Простейший способ отображения набора данных на Web-форме — использование элемента управления DataGrid и привязки данных.

Чтобы отобразить набор данных указанным способом, выполните следующие действия.

1. Создайте объекты соединения с БД, адаптера и набора данных, как описано в предыдущем занятии.
2. Поместите на Web-форму элемент управления DataGrid.
3. Поместите в свойство *DataSource* элемента управления DataGrid имя набора данных.
4. Добавьте к обработчику события *Page_Load* Web-формы код, заполняющий набор данных с помощью адаптера и связывающий элемент управления DataGrid с набором данных.

Следующий обработчик события создает набор данных на основе таблицы Contacts из БД Contacts, а затем отображает его содержимое с помощью элемента управления DataGrid:

Visual Basic .NET

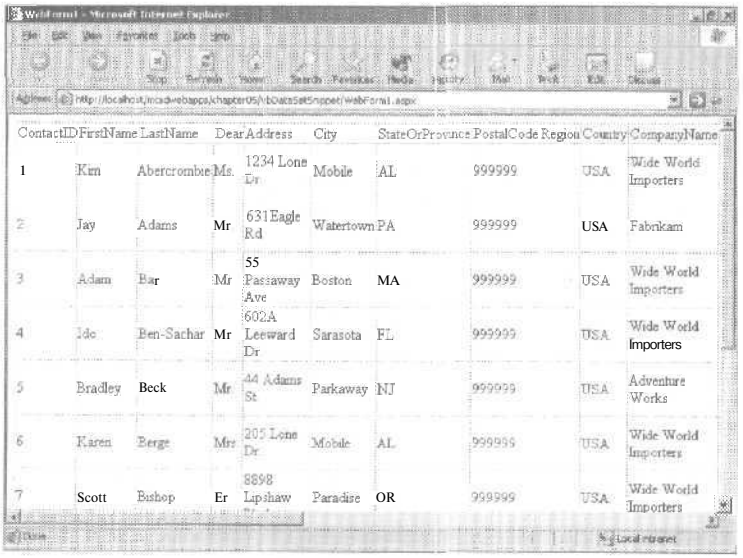
```
Private Sub Page_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load
        ' Заполнить набор данных.
        adptContacts.Fill(dsContacts)
```

```
        ' Связать данные с элементом управления DataGrid.
        grdContacts.DataBind()
    End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
    {
        // Заполнить набор данных.
        adptContacts.Fill(dsContacts);
        // Связать данные с элементом управления DataGrid.
        grdContacts.DataBind();
    }
```

Вышеприведенный код отображает все столбцы набора данных без форматирования, как показано на рис. 5-16.



ContactID	FirstName	LastName	Dear	Address	City	StateOrProvince	PostalCode	Region	Country	CompanyName
1	Kim	Abercrombie	Ms.	1234 Lone Dr	Mobile	AL	999999		USA	Wide World Importers
2	Jay	Adams	Mr	631 Eagle Rd	Watertown	PA	999999		USA	Fabrikam
3	Adam	Bar	Mr	55 Passaway Ave	Boston	MA	999999		USA	Wide World Importers
4	Ido	Ben-Sachar	Mr	602A Leeward Dr	Sarasota	FL	999999		USA	Wide World Importers
5	Bradley	Beck	Mr	44 Adams St	Parkaway	NJ	999999		USA	Adventure Works
6	Karen	Berge	Mrs	205 Lone Dr	Mobile	AL	999999		USA	Wide World Importers
7	Scott	Bishop	Er	8898 Lipshaw	Paradise	OR	999999		USA	Wide World Importers

Рис. 5-16. Вид DataGrid без форматирования

Но этого, как правило, недостаточно: например, столбец Birthdate (он не показан на рис. 5-16) включает не только дату, но и время, что нежелательно.

Чтобы выборочно отобразить столбцы в DataGrid с использованием форматирования данных, выполните следующие действия.

1. Щелкните ссылку Property Builder в окне Properties — Visual Studio откроет диалоговое окно свойств элемента управления DataGrid.
2. Щелкните элемент Columns слева, чтобы выбрать отображаемые столбцы набора данных и задать их форматирование. При этом Visual Studio показывает свойства столбцов (рис. 5-17).

Чтобы выбрать столбцы набора данных для отображения, выполните следующие действия.

1. Снимите флажок Create Columns Automatically At Run Time, в результате только выбранные столбцы будут отображаться в DataGrid.

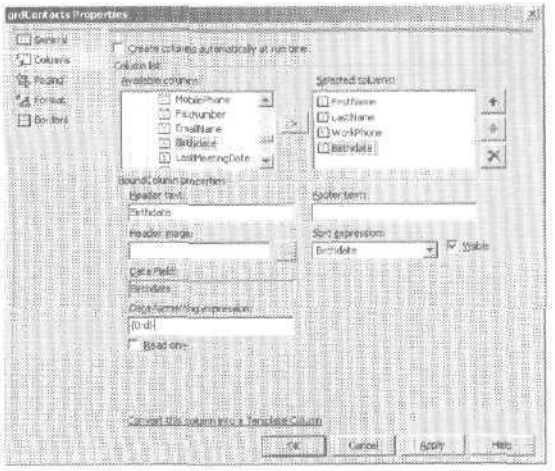


Рис. 5-17. Выбор столбцов, отображаемых DataGrid, и их форматирование

2. Выберите нужный столбец в списке Available Columns и щелкните кнопку Add (>), чтобы добавить его к списку Selected Columns.

Чтобы задать форматирование данных в столбце, выполните следующие действия.

1. В списке Selected Columns выделите столбец, для которого нужно задать формат.
2. В текстовое поле Data Formatting Expression введите форматирующее выражение, оно выглядит примерно так:

```
{0:formattingexpression}
```

где *formattingexpression* — одно из предопределенных форматирующих выражений .NET Framework. Дополнительные сведения о форматирующих выражениях см. в разделах справочной системы Visual Studio, озаглавленных «Date and Time Format Strings», «Numeric Format Strings» и «Custom Format Strings».

В диалоговом окне свойств DataGrid можно определить все особенности отображения и поведения этого элемента управления, в том числе формат заголовков столбцов, способ прокрутки сетки, отображение границ и линий, а также многое другое.

На рис. 5-18 показан набор данных Contacts, отображаемый в DataGrid, но на этот раз с применением форматирования данных. Как видите, DataGrid отображает лишь четыре столбца из набора данных, а столбцу Birthdate назначен короткий формат даты (*{0:d}*).

Отображение набора данных с помощью DataList

Чтобы отобразить содержимое набора данных в виде списка, а не сетки (как делает DataGrid), воспользуйтесь элементом управления DataList.

Для отображения содержимого набора данных с помощью элемента управления DataList выполните следующие действия.

1. Создайте объекты соединения с БД, адаптера и набора данных, как описано в предыдущем занятии.
2. Поместите на Web-форму элемент управления DataList.
3. Занесите в свойство *DataSource* элемента управления DataGrid имя набора данных.
4. Добавьте к обработчику события *Page_Load* Web-формы код, заполняющий набор данных из адаптера и связывающий DataGrid с набором данных.
5. Чтобы задать внешний вид DataList, отредактируйте шаблоны его шапки, элементов и разделителей.

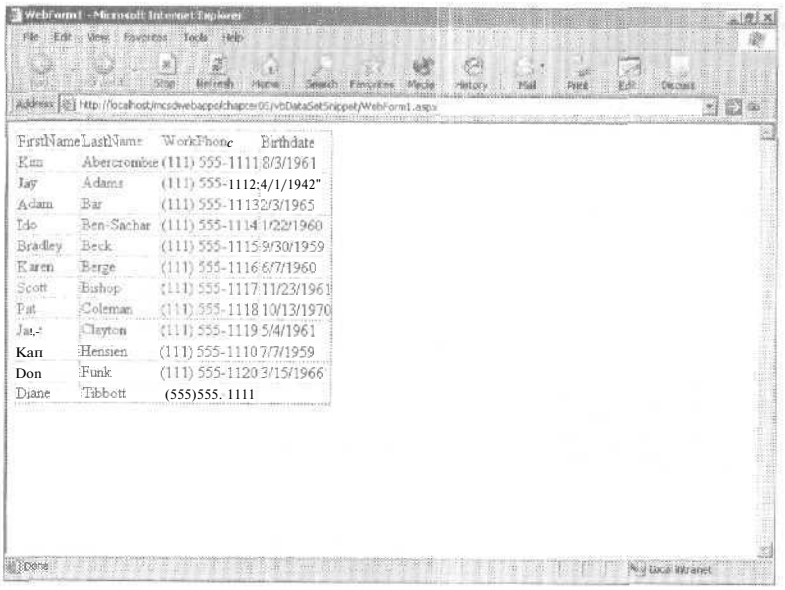


Рис. 5-18. Выборочное отображение столбцов набора данных с форматированием

Содержимое верхнего и нижнего колонтитулов, а также строк элемента управления DataList определяется шаблонами. Редактируя их, можно добавить к шаблонам текст и элементы управления, связанные с набором данных. DataList поддерживает три категории шаблонов, управляющих различными параметрами его отображения:

- **шаблоны верхнего и нижнего колонтитулов** — включают заголовок и линии, отображаемые в начале и в конце списка строк;
- **шаблоны элементов** — определяют содержимое строк, а также позволяют изменять вид четных, нечетных, выделенных и редактируемых строк;
- **шаблоны разделителей** — определяют разделители строк в виде линий (и не только).

Чтобы отредактировать шаблон DataList, выполните следующие действия.

1. Щелкните DataList правой кнопкой и выберите из контекстного меню команду Edit Template, затем укажите в списке шаблон, который нужно отредактировать.
2. Как показано на рис. 5-19, вид элемента управления DataList в режиме редактирования шаблона изменяется. Отредактируйте шаблон, добавив к нему нужный текст и элементы управления.
3. Чтобы поместить в шаблон элемент набора данных, добавьте к шаблону элемент управления и щелкните свойство *DataBindings* этого элемента управления. Затем щелкните кнопку со знаком многоточия (...) справа от свойства *DataBindings* — Visual Studio откроет диалоговое окно DataBindings для добавленного элемента управления (рис. 5-20),
4. Закончив редактирование шаблонов элемента управления DataList, щелкните DataList правой кнопкой и выберите из контекстного меню команду End Template Editing.

Для связывания с данными элементов управления, добавленных к шаблону элемента DataList, применяют как обычную привязку к элементу данных через контейнер (которым является DataList, рис. 5-20), так и более сложные выражения, использующие метод *Eval*. Например, следующее выражение объединяет имя и фамилию, чтобы они отображались в одном элементе управления:

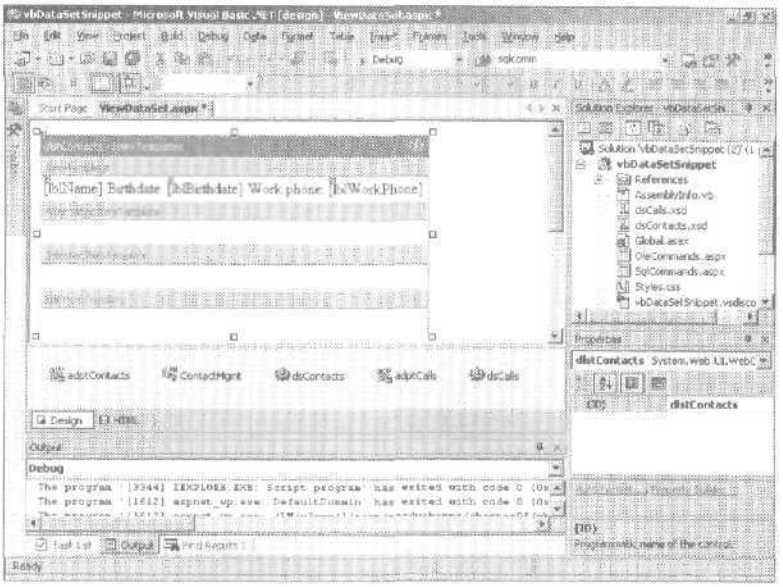


Рис. 5-19. Вил элемента управления DataList в режиме редактирования шаблона

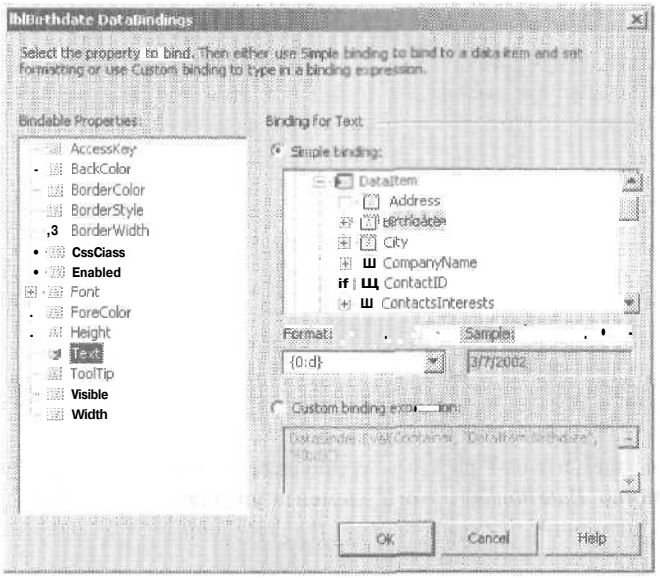


Рис. 5-20. Диалоговое окно DataBindings

Visual Basic .NET

```
DataBinder.Eval(Container, "DataItem.FirstName") & " " & _
  DataBinder.Eval(Container, "DataItem.LastName")
```

Visual C#

```
DataBinder.Eval(Container, "DataItem.FirstName") + " " +
  DataBinder.Eval(Container, "DataItem.LastName");
```

На самом деле при создании шаблона элемента управления DataList со связыванием данных Visual Studio автоматически вставляет метод *Eval* объекта *DataBinder* в HTML-код Web-формы. Иногда проще отредактировать шаблоны прямо в HTML-коде, чем использовать указанную процедуру.

Чтобы отредактировать шаблон DataList прямо в HTML-тексте, щелкните Web-форму правой кнопкой и выберите из контекстного меню команду View HTML Source — Visual Studio откроет Web-форму в режиме HTML.

Следующий HTML-код отображает элемент управления DataList с именем, датой рождения и номером телефона, взятыми из набора данных Contacts.

```
<asp:DataList id="dlstContacts" runat="server" DataSource="<%# dsContacts %>">
<HeaderTemplate>
<h2>
    Contact Information
   <HR width="100%" SIZE="1">
   </h2>
</HeaderTemplate>

<FooterTemplate>
<HR>
</FooterTemplate>

<ItemTemplate>
    <asp:Label id="lblName" runat="server"
        Text='<%# DataBinder.Eval(Container, "DataItem.FirstName") + " "
        + DataBinder.Eval(Container,"DataItem.LastName") %>'>
    </asp:Label>
     Birthdate:
    <asp:Label id="lblBirthdate" runat="server"
        Text='<%# DataBinder.Eval(Container, "DataItem.Birthdate",
    "{0:d}) %>'>
    </asp:Label>
     Work phone:
    <asp:Label id="lblWorkPhone" runat="server"
        Text='<%# DataBinder.Eval(Container, "DataItem.WorkPhone") %>'>
    </asp:Label>
</ItemTemplate>

<SeparatorTemplate>
   <HR>
</SeparatorTemplate>
</asp:DataList>
```

Для отображения содержимого набора данных в этом элементе управления необходимо заполнить набор данных с помощью адаптера и связать его с элементом управления DataList, как демонстрирует следующий обработчик события *Page_Load*:

Visual **Basic .NET**

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Заполнить набор данных.
    adptContacts.Fill(dsContacts)
    ' Связать данные с элементом управления DataList.
    dlstContacts.DataBind()
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
   // Заполнить набор данных.
   adptContacts.Fill(dsContacts);
   // Связать данные с элементом управления DataList.
   dlstContacts.DataBind();
}
```

Во время выполнения заголовок, элементы и разделители этого элемента управления DataList выглядят так, как показано на рис. 5-21.

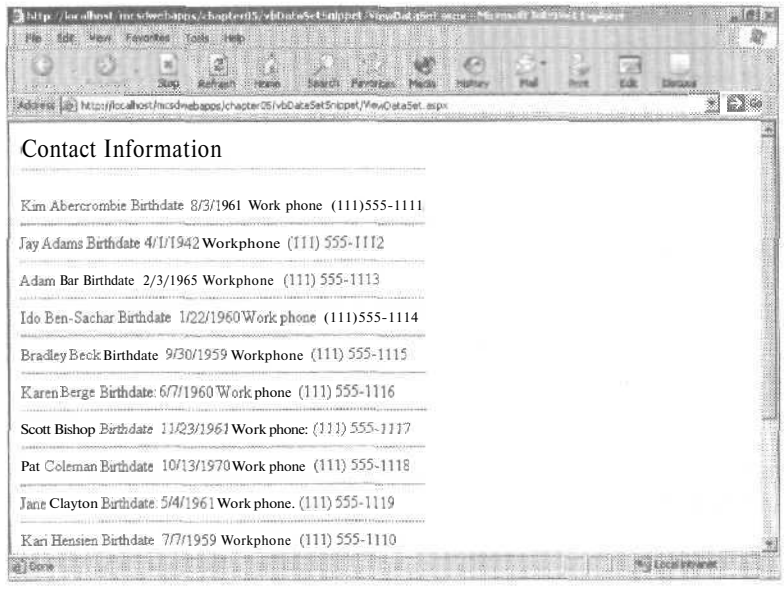


Рис. 5-21. Вид связанного с данными элемента управления DataList во время выполнения

Отображение элементов данных в других элементах управления, представляющих списки

Естественно, элементы набора данных можно связывать с любым элементом управления Web-формы, представляющим список, DataGrid и DataList — это просто примеры наиболее гибких средств отображения содержимого набора данных.

Чтобы отобразить содержимое набора данных при помощи элементов управления ListBox, DropDownList, CheckBoxList или RadioButtonList, выполните следующие действия.

1. Занесите имя набора данных в свойство *DataSource* выбранного элемента управления.
2. Занесите в свойство *DataText* этого элемента управления член набора данных, отображаемый как свойство *Text* элемента списка.
3. Поместите в свойство *DataValue* элемента управления член набора данных, возвращаемый как свойство *Value* элемента списка.
4. В коде заполните набор данных при помощи объекта-адаптера и свяжите его с элементом управления.

Следующий HTML-текст демонстрирует значения свойств элемента управления DropDownList, связанного с набором данных Contacts:

```
<asp:DropDownList id="drpContacts" runat="server" Width="384px"
Height="22px" DataSource="<%# dsContacts %>" DataTextField='LastName'
DataValueField="ContactID"></ asp:DropDownList>
```

Этот фрагмент кода заполняет набор данных и связывает данные с элементом управления DropDownList:

Visual Basic .NET

```
Private Sub Page_Load(6yVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Заполнить набор данных.
    adptContacts.Fill(dsContacts)
    ' Сзязать данные с элементом управления.
    drpContacts.DataBind()
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
   // Заполнить набор данных,
   adptContacts.Fill(dsContacts);
   // Связать данные с элементом управления DataList.
   dlstContacts. DataBindO;
}
```

Во время выполнения DropDownList отображает список фамилий из таблицы Contacts, а свойство *SelectedItem.Value* возвращает значение поля ContactID, что упрощает поиск остальных персональных сведений.

Однако у этого способа есть ограничение. Оно связано с тем, что свойства *DataText* и *DataValue* элемента управления, представляющего список, поддерживают не больше одного значения. Поэтому, если нужно отображать в DropDownList обновременно имя и фамилию контактного лица, это можно сделать только из кода. Следующий обработчик события *Page_Load* добавляет в DropDownList сцепленные имя и фамилию и записывает в свойство *Value* значение ContactID для каждого элемента списка:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
```

```
    ' Этот код исполняется при первом отображении страницы,
    If Not IsPostBack Then
        ' Заполнить набор данных Contacts,
        adptContacts.Fill(dsContacts)
        ' Для каждой строки таблицы...
        Dim drowItem As dsContacts.ContactsRow
        For Each drowItem In dsContacts.Contacts
            ' Создать новый элемент списка.
            Dim lstNew As New ListItem()
            lstNew.Text - drowItem.FirstName & " " & drowItem.LastName
            lstNew.Value - drowItem.ContactID
            ' Добавить элемент к раскрывающемуся списку.
            drpContacts.Items.Add(lstNew)
        Next
    End If
End Sub
```

Visual C#

```
private void Page_Load(object sender. System.EventArgs e)
  {
       ,// Этот код исполняется при первом отображении страницы.
       if (!IsPostBack)
       {
            // Заполнить набор данных Contacts.
            adptContacts.Fill(dsContacts);
            // Для каждой строки таблицы...
            foreach (dsContacts.ContactsRow drowItem in
              dsContacts.Contacts)
            {
                // Создать новый элемент списка.
                ListItem lstNew = new ListItem();
                lstNew.Text = drowItem.FirstName + " " +
                  drowItem.LastName;
                lstNew.Value - drowItem.ContactID.ToString();
                // Добавить элемент к раскрывающемуся списку.
                drpContacts.Items.Add(lstNew);
            }
       }
  }
```

Выборка отдельных записей

Элемент управления DropDownList, созданный в предыдущем разделе, годится и для выборки строк из таблицы Calls в БД Contacts. Поскольку поле ContactID является уникальным **ключом в БД. его можно использовать для выборки из таблицы** Call **записей** для **отдельных контактных** лиц. Следующий **код создает набор данных со сведениями о звонках контактного лица, выбранного в DropDownList:**

Visual Basic .NET

```
Private Sub drpContacts_SelectedIndexChanged(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles drpContacts.SelectedIndexChanged
    ' Отобразить сведения о звонках контактного лица, выбранного
    ' в списке.
    adptCalls.SelectCommand.CommandText = "SELECT * FROM Calls" & _
        " WHERE ContactID =" & drpContacts.SelectedItem.Value
    ' Очистить набор данных.
    dsCalls.Clear
    adptCalls.Fill(dsCalls)
    ' Отобразить результаты в сетке.
    grdCalls.DataBind()
    End Sub
```

Visual C#

```
private void drpContacts_SelectedIndexChanged(object sender,
  System.EventArgs e)
{
    // Отобразить сведения о звонках контактного лица,
    // выбранного в списке.
    adptCalls.SelectCommand.CommandText = "SELECT * FROM Calls" +
      " WHERE ContactID =" + drpContacts.SelectedItem.Value;
    // Очистить набор данных.
    dsCalls.Clear();
    adptCalls.Fill(dsCalls);
    // Отобразить результаты в сетке.
    grdCalls.DataBind();
}
```

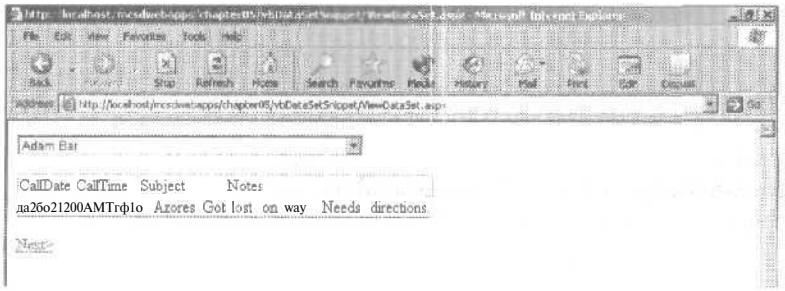


Рис. 5-22. **Список DropDownList**

Во время выполнения Web-форма отображает список звонков контактного лица, указанного в списке DropDownList, как показано на рис. 5-22.

Исполнение команд над базой данных

Помимо манипулирования наборами данных, **ADO.NET** позволяет напрямую исполнять команды в БД через соединение. Объект соединения поддерживает три метода исполнения команд:

- *ExecuteScalar* — исполняет запросы, возвращающие единственное значение, такие, как подсчет числа строк в таблице;
- *ExecuteNonQuery* — выполняет команды, модифицирующие БД (например, добавляющие и удаляющие элементы БД), но не возвращающие никаких значений. Метод *ExecuteNonQuery* также возвращает количество строк, на которые повлияло исполнение команды;
- *ExecuteReader* — последовательно читает записи из БД.

Чтобы использовать эти методы, выполните следующие действия.

1. Создайте соединение с БД.
2. Откройте соединение.
3. Создайте объект команды, содержащий текст команды SQL или имя хранимой процедуры.
4. Вызовите метод объекта команды.
5. Закройте соединение с БД.

Извлечение значений из базы данных

В занятии 1 рассказано, как добавлять и удалять строки из набора данных, а также записывать в БД его содержимое. При этом приходится учитывать одну важную деталь: перед добавлением записей в БД часто приходится извлекать из нее данные. Самый распространенный пример, когда в записи имеется поле с уникальным ключом, такое, как ContactID в БД Contacts. При этом перед добавлением записей в БД необходимо получить новое значение ключа.

Разработано несколько способов извлечения нового значения уникального ключа БД. Возможно, самый прямолинейный из них — это создание объекта команды и вызов его метода *ExecuteScalar*, возвращающего искомое значение. Существует два типа объектов команд:

- *SqlCommand* — используются при работе с БД SQL;
- *OleDBCommand* — применяются при работе с БД OLE.

Следующий код при помощи объекта *SqlCommand* извлекает максимальное значение ContactID, увеличивает его и указывает полученное число в качестве уникального ключа для новой строки, вставляемой в таблицу Contact:

Visual Basic .NET

```
' В этом коде используются соединение, адаптер и набор данных,
' созданные в режиме Design.
Private Sub butAddRow_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butAddRow.Click
    ' Получить новое уникальное значение ContactID.
    Dim intID As Integer
    Dim cmdNewID As New SqlCommand("SELECT MAX(ContactID)" & _
      " FROM Contacts", ContactMgmt)
    ' Открыть соединение с БД.
    ContactMgmt.Open()
    intID = CInt(cmdNewID.ExecuteScalar())
    ' Создать объект, представляющий новую строку таблицы Contacts.
    Dim rowNew As dsContacts.ContactsRow = _
    dsContacts.Contacts.NewRow
    ' Записать значения е поля строки.
    rowNew.ContactID = intID + 1
```

```
    rowNew.FirstName = "Max"
    rowNew.LastName = "Benson"
    rowNew.WorkPhone = "(111) 555-1212"
    ' Добавить строку к набору данных.
    dsContacts.Contacts.AddContactsRow(rowNew)
    ' Закрыть соединение с БД.
    ContactMgmt.Close()
End Sub
```

Visual C#

```
private void butAddRow_Click(object sender, System.EventArgs e)
{
    // Получить новое уникальное значение ContactID.
    int intID;
    SqlCommand cmdNewID = new SqlCommand("SELECT MAX(ContactID)" +
      " FROM Contacts", ContactMgmt);
    // Открыть соединение с БД.
    ContactMgmt.Open();
    intID = (int)sqlcommNewID.ExecuteScalar();
    // Создать объект, представляющий новую строку таблицы Contacts.
    dsContacts.ContactsRow rowNew = (dsContacts.ContactsRow)
    dsContacts.Contacts.NewRow();
    // Записать значения в поля строки.
    rowNew.ContactID = intID + 1;
    rowNew.FirstName = "Max";
    rowNew.LastName = "Benson";
    rowNew.WorkPhone = "(111) 555-1212";
    // Добавить строку к набору данных.
    dsContacts.Contacts.AddContactsRow(rowNew);
    // Закрыть соединение с БД.
    ContactMgmt.CloseO;
}
```

Метод *ExecuteScalar* работает с любыми операторами **SQL,** возвращающими единственное **значение; для исполнения других** видов операторов **SQL применяются методы** *Execute-NonQuery* **и** ***Execute****Reader.*

Модификация записей непосредственно в базе данных

Метод *ExecuteNonQuery* **исполняет** команды, **не возвращающие наборы** данных, **такие, как операторы SQL INSERT, DELETE или UPDATE. Например, следующий код использует метод** *ExecuteNonQuery* **для** удаления **строк прямо из БД:**

Visual Basic .NET

```
Private Sub butDelete_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles butDelete.Click
  Dim intRows As Integer
  ' Создать объект-команду.
  Dim cmdDeleteRow As New SqlCommand("DELETE ROW" & _
```

```
        " FROM Contacts WHERE ContactID=1", ContactMgmt)
    ' Открыть соединение с БД.
    ContactMgmt.Open()
    ' Исполнить команду.
    intRows = cmdDeleteRow.ExecuteNonQuery()
    ' Отобразить число удаленных строк.
    Response.Write("Records deleted: " & intRows)
    ' Закрыть соединение.
    ContactMgmt.Close()
End Sub
```

Visual C#

```
private void butDeleteRow_Click(object sender, System.EventArgs e)
{
     int intRows;
     // Создать объект-команду.
         SqlCommand cmdDeleteRow - new SqlCommand("DELETE ROW" +
     " FROM Contacts WHERE ContactID=1", ContactMgmt);
     // Открыть соединение с БД.
     ContactMgmt.Open();
     // Исполнить команду.
     intRows = cmdDeleteRow. ExecuteNonQueryO;
     // Отобразить число удаленных строк.
     Response.Write("Records deleted: " + intRows.ToString());
     // Закрыть соединение.
     ContactMgrnt. Close();
}
```

Метод *ExecuteNonQuery* не обращается к БД через адаптер или набор данных, а взаимодействует с ней напрямую. Если для модификации таблицы использовался метод *ExecuteNonQuery*, необходимо обновить все наборы данных, на которые повлияли эти модификации, вызвав метод *Fill* адаптера данных.

Извлечение записей непосредственно из базы данных

Метод *ExecuteReader* исполняет команды, возвращающие записи, такие, как оператор SQL SELECT. Каждая запись возвращается в виде объекта чтения данных. Этот объект напоминает набор данных, доступный только для чтения. Поскольку метод *ExecuteReader* взаимодействует с БД напрямую, существует две версии объекта чтения данных: *OleDbDataReader* и *SqlDataReader*.

Использование метода *ExecuteReader* для создания объектов чтения данных обеспечивает более высокое быстродействие, чем создание набора данных на основе объекта-адаптера данных, но не дает такой же гибкости. Объекты чтения данных неизменны и выполняют только последовательное однонаправленное чтение записей БД, тогда как наборы данных позволяют извлекать записи в любом порядке и, что важнее, записывать модификации обратно в БД.

Следующий код извлекает сведения обо всех звонках некоторого контактного лица и отображает их на Web-форме с помощью объекта чтения данных и его команды *Execute*:

Visual Basic .NET

```
Private Sub butShowCalls_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles butShowCalls.Click
  ' Создать объект команды.
  Dim cmdGetCalls As New SqlCommand("SELECT * FROM Calls" & _
    " WHERE ContactID=3", ContactMgmt)
  ' Открыть соединение с БД.
  ContactMgmt.Open()
  ' Создать объект чтения данных.
  Dim dreadCalls As SqlDataReader
  ' Исполнить команду.
  dreadCalls = cmdGetCalls.ExecuteReader()
  ' Отобразить записи.
  Do While dreadCalls.Read()
    Response.Write(dreadCalls.GetString(4))
  Loop
  ' Закрыть объект чтения данных.
  dreadCalls.Close()
  ' Закрыть соединение с БД.
  ContactMgmt.CloseO
End Sub
```

Visual C#

```
private void butShowCalls_Click(object sender, System.EventArgs e)
{
    // Создать объект команды.
    SqlCommand cmdGetCalls = new SqlCommand("SELECT * FROM Calls" +
    " WHERE ContactID=3", ContactMgmt);
    // Открыть соединение с БД.
    ContactMgmt.Open();
    // Создать объект чтения данных.
    SqlDataReader dreadCalls;
    // Исполнить команду.
    dreadCalls = cmdGetCalls.ExecuteReaderO;
    // Отобразить записи.
    while (dreadCalls.Read())
    Response.Write(dreadCalls.GetString(4));
    // Закрыть объект чтения данных.
    dreadCalls.CloseO;
    // Закрыть соединение с БД.
    ContactMgmt.Close();
}
```

Поскольку объект чтения данных представляет собой набор записей для однонаправленного чтения, метод *Read* последовательно читает строки одну за другой, пока не достигнет конца набора. На время своей работы объект чтения данных блокирует соединение с БД, поэтому его метод *Close* следует вызвать только после завершения извлечения записей, как показано в приведенном выше коде.

Занятие 3. Обработка транзакций

ADO.NET позволяет группировать операции БД в транзакции. *Транзакция*(transaction) — это группа команд, модифицирующих хранимые в БД данные. Транзакция обрабатывается как единое целое, гарантируя исполнение команд по принципу «все или ничего»: если при исполнении хоть одной команды транзакции произойдет сбой, исполнение остальных команд этой транзакции тоже окончится неудачей, а любые модификации БД, сделанные этими командами, будут отменены. Таким образом, транзакции обеспечивают целостность данных БД.

Из этого занятия вы узнаете, как реализовать обработку транзакций для наборов данных и БД в Web-приложении.

Изучив материал этого занятия, вы сможете:

- ✓ понять, почему обработка транзакций так важна для Web-приложений, использующих доступ к данным;
- ✓ рассказать, на каких уровнях поддерживается обработка транзакций в ADO.NET;
- ✓ управлять модификацией набора данных при помощи транзакций;
- ✓ создавать объекты транзакций БД;
- ✓ отслеживать команды, исполняемые в БД посредством объекта транзакции;
- ✓ фиксировать или отменять (откатывать) изменения с помощью объекта транзакции.

Продолжительность занятия — около 30 минут.

Введение в транзакции

Как отмечено выше, транзакция — это группа команд БД, обрабатываемых как единое целое. Программисты, занимающиеся разработкой БД, определили следующие критерии транзакции: атомарность, согласованность, изолированность и устойчивость (так называемые ACID-критерии: atomic, consistent, isolated, durable). Итак, команды составляют транзакцию, если они:

- атомарны, то есть составляют единицу работы. Например, при изменении адреса клиента оператор, занимающийся вводом данных о клиентах, должен отредактировать все поля адреса клиента (улицу, город и т. д.) сразу, а не по отдельности;
- **согласованны,** то есть не нарушают связей между данными БД. Например, если сведения о клиенте включают ставку налога, которая берется из таблицы с налоговыми ставками в различных штатах, то штат, указанный в сведениях о клиенте, непременно должен фигурировать в этой таблице;
- изолированы — изменения, вносимые другими клиентами, не влияют на текущие изменения. Например, если два оператора, выполняющие ввод данных, одновременно попытаются изменить данные об одном и том же клиенте, будут приняты изменения, вносимые только одним из операторов, а другой получит уведомление о том, что сделанные им изменения не были внесены, либо оба оператора получат такое уведомление. В любом случае сведения о клиенте не останутся в неопределенном состоянии;
- **устойчивы** — однажды внесенное изменение становится постоянным. Если во время исполнения команд возник сбой системы или произошла аварии электропитания, то при следующем запуске системы все незавершенные команды отменяются, а данные возвращаются в исходное состояние.

Обработка транзакций особенно важна для Web-приложений, использующих доступ к данным, поскольку такие Web-приложения, как правило, являются распределенными и с ними одновременно работает множество клиентов. Базы данных представляют собой общий ресурс Web-приложений, поэтому при использовании БД в приложениях, с которыми работают множество клиентов, рассредоточенных по обширной территории, возникают следующие проблемы:

- **конкуренция за ресурсы** — возникает, когда несколько клиентов одновременно пытаются изменить одну и ту же запись. Чем больше клиентов работает с приложением, тем серьезнее эта проблема;
- **неожиданные сбои.** Даже если ваше Web-приложение и Web-сервер обладают 100-процентной надежностью, то Интернет далеко не самая надежная сеть. В любой момент клиенты могут быть неожиданно отключены из-за проблем у провайдера услуг Интернета, неполадок модема или сбоев электропитания;
- **проблемы, связанные с жизненным циклом Web-приложения**. Жизненный цикл Web-приложения отличается от жизненного цикла приложений Windows. Web-формы живут очень недолго, а клиент в любой момент может покинуть ваше приложение, просто набрав новый адрес в своем браузере.

Транзакции обрабатываются в несколько этапов:

1. начало транзакции;
2. обработка команд БД;
3. проверка наличия ошибок;
4. если обнаружены ошибки, БД возвращается в состояние, в котором она находилась до начала транзакции, в противном случае происходит фиксация транзакции.

ADO.NET поддерживает различные средства для обработки транзакций на разных уровнях:

- **наборы данных поддерживают обработку транзакций посредством методов *RejectChanges* и *Update*,** а также предоставляют метод *AcceptChanges*, сбрасывающий состояние записей в наборе данных, в результате чего они помечаются как немодифицированные;
- **объекты соединения с БД поддерживают обработку транзакций при помощи объекта транзакции.** Объекты транзакций отслеживают команды, выполняемые в БД, и предоставляют методы *Rollback, Commit* и *Save,* позволяющие соответственно вернуть БД в исходное состояние, зафиксировать сделанные изменения и создать внутри транзакции точку сохранения;
- **пространство имен *System. Enterprise Services* поддерживает транзакции на корпоративном уровне посредством класса ContextUtil.** При помощи службы Distributed Transaction Coordinator (DTC), поставляемой с Microsoft SQL Server 2000, корпоративные транзакции способны отслеживать транзакции, совершаемые множеством Web-форм и компонентов COM+.

Использование транзакций с наборами данных

Наборы данных поддерживают неявную обработку транзакций, так как изменения, сделанные в наборе данных, не вносятся в БД до тех пор, пока не будет вызван метод *Update* адаптера данных. Таким образом, можно выполнить несколько команд обработки данных, а затем выбрать точку, в которой изменения будут записаны в БД и станут постоянными.

Если при исполнении метода *Update* возникает ошибка, ни одно из изменений не переносится из набора данных в БД. На этом этапе можно исправить ошибку и снова попытаться вызвать *Update* либо отменить все незавершенные изменения в наборе данных с

помощью метода *RejectChanges*, поддерживаемого набором данных. Например, следующий код отображает содержимое списка Contacts посредством элемента управления DataGrid, позволяя пользователю удалять строки, щелкнув кнопку Delete в DataGrid:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    ' Проверить, отображается ли страница впервые
If Not IsPostBack Then
        ' Если страница отображается впервые, заполнить набор данных,
        adptContacts.Fill(dsContacts)
        ' сохранить набор данных в переменной состояния,
        Session("dsContacts") = dsContacts
    Else
        ' Если страница отображается не первый раз,
        ' получить данные из переменной состояния,
        dsContacts = Session("dsContacts")
    End If
    ' осуществить привязку к набору данных.
    grdContacts.DataBind()
End Sub

Private Sub grdContacts_ItemCommand(ByVal source As Object, _
  ByVal e As System.Web.UI.WebControls.DataGridCommandEventArgs) _
  Handles grdContacts.ItemCommand
    ' Если пользователь щелкнул кнопку Delete...
   If e.CommandName = "Delete" Then
        Dim intContactID
        ' ...получить ContactID для выделенной строки,
        intContactID = _
          CInt(grdContacts.Items(e.Item.ItemIndex).Cells(3).Text)
        Dim rowDelete As dsContacts.ContactsRow
        ' получить строку, удаляемую из набора данных,
        rowDelete = _
        dsContacts.Contacts.FindByContactID(intContactID)
        ' удалить строку.
        rowDelete.Delete()
        ' Обновить DataGrid.
        grdContacts.DataBind()
    End If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Проверить, отображается ли страница впервые
    if (!IsPostBack)
```

```
    {
        // Если страница отображается впервые, заполнить набор данных,
        adptContacts.Fill(dsContacts);
        // сохранить набор данных в переменной состояния.
        Session["dsContacts"] = dsContacts;
    }
    else
        // Если страница отображается не первый раз,
        // получить данные из переменной состояния.
        dsContacts = (dsContacts)Session["dsContacts"];
    // Осуществить привязку к набору данных.
    grdContacts.DataBind();
}

private void grdContacts_ItemCommand(object sender,
  System.Web.UI.WebControls.DataGridCommandEventArgs e)
{
    // Если пользователь щелкнул кнопку Delete...
    if (e.CommandName == "Delete")
    {
        int intContactID;
        // ...получить ContactID для выбранной строки.
        intContactID = Convert.ToInt16
        (grdContacts.Items[e.Item.ItemIndex].Cells[3].Text);
        dsContacts.ContactsRow rowDelete;
        // получить строку, удаляемую из набора данных.
        rowDelete = _
        dsContacts.Contacts.FindByContactID(intContactID);
        // удалить строку.
        rowDelete.Delete();
        // Обнозить DataGrid.
        grdContacts.DataBind();
    }
}
```

Следующие обработчики событий кнопок Restore и Commit позволяют пользователю вернуть набор данных в исходное состояние либо перенести в БД результаты удаления, сделанного предыдущим кодом:

Visual Basic .NET

```
Private Sub butRestore_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles butRestore.Click
    ' Вернуть набор данных в исходный вид.
    dsContacts.RejectChanges()
    ' Обновить DataGrid.
    grdContacts.DataBind()
End Sub
```

```
Private Sub butCommit_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butCommit.Click
    ' Записать в БД содержимое набора данных.
    adptContacts.Update(dsContacts)
    ' Сохранить изменения в переменной состояния.
    Session("dsContacts") = dsContacts
    ' Обновить DataGrid.
    grdContacts.DataBind()
End Sub
```

Visual C#

```
private void butRestore_Click(object sender, System.EventArgs e)
{
    // Вернуть набор данных в исходный вид.
    dsContacts.RejectChanges();
    // Обновить DataGrid.
    grdContacts.DataBind();
}

private void butCommit_Click(object sender, System.EventArgs e)
{
    int intRows;
    // Записать в БД содержимое набора данных.
intRows = adptContacts.Update(dsContacts);
    // Сохранить изменения в переменной состояния.
    Session["dsContacts"] = dsContacts;
    // Обновить DataGrid.
    grdContacts.DataBindO;
}
```

Метод *RejectChanges,* используемый в показанном обработчике события *butRestore_Click,* отменяет удаление строки, возвращая набор данных в исходное состояние. Действие метода *AcceptChanges* противоположно *RejectChanges*: этот метод сбрасывает свойство *DataRowState* у всех измененных строк набора данных, устанавливая его в *Unchanged,* и реально удаляет все удаляемые строки.

Однако при использовании метода *AcceptChanges* нельзя вносить аналогичные изменения в БД при помощи метода *Update,* поскольку последний определяет модифицированные строки на основе их свойства *DataRowState.* В силу этих обстоятельств *AcceptChanges* разрешается применять, только когда вы не планируете переписывать в БД содержимое набора данных.

Использование транзакций с базами данных

Управлять транзакциями на уровне БД позволяет объект транзакции. Так как в ADO NET существует два типа соединений с БД, возможны и два типа объектов транзакций: *SqlTransaction* и *OleDbTransaction.*

Чтобы воспользоваться любым из этих объектов, выполните следующие действия:

1. Откройте соединение с БД.
2. Создайте объект транзакции с помощью метода *Begin Transaction* объекта соединения с БД.

3. Создайте объекты команд, за которыми будет следить эта транзакция. Для этого занесите в свойство *Transaction* каждого объекта команды имя транзакции, созданной на этапе 2.
4. Исполните команды. Поскольку назначение обработки транзакции — обнаруживать и исправлять ошибки до записи данных в БД, обычно код заключается в структуру обработки ошибок,
5. Зафиксируйте сделанные изменения в БД, если команды исполнены успешно, в противном случае верните ее в исходное состояние.
6. Закройте соединение с БД.

Следующий код отображает список контактных лиц из БД Contacts при помощи DataGrid, в котором также имеется столбец с кнопками; последние позволяют удалять из списка соответствующие строки. Чтобы при удалении контактного лица из списка гарантировать удаление всех сведений о его звонках, функция *DeleteContact* использует транзакцию. Эти меры обеспечивают целостность БД.

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load
    ' Заполнить набор данных.
    adptContacts.Fill(dsContacts)
    ' Осуществить привязку к набору данных.
    grdContacts.DataBind()
End Sub

Private Sub grdContacts_ItemCommand(ByVal source As Object, _
  ByVal e As System.Web.UI.WebControls.DataGridCommandEventArgs) _
  Handles grdContacts.ItemCommand
    ' Если пользователь щелкнул кнопку Delete...
    If e.CommandName = "Delete" Then
        Dim intContactID
        ' ...получить ContactID для выбранной строки,
        intContactID = _
          CInt(grdContacts.Items(e.Item.ItemIndex).Cells(3).Text)
        ' Удалить сведения о контактном лице.
        lblStatus.Text = DeleteContact(intContactID)
    End If
    ' Обновить набор данных,
    adptContacts.Fill(dsContacts)
    ' Обновить DataGrid.
    grdContacts.DataBind()
End Sub

Function DeleteContact(ByVal intContactID As Integer) As String
    ' Открыть соединение с БД.
    ContactMgmt.Open()
    ' Объявить объект транзакции.
    Dim transDelete As SqlTransaction
```

```
    ' Создать транзакцию,
    transDelete = ContactMgmt.BeginTransaction _
     CIsolationLevel.ReadCommitted)
    ' Создать команду для удаления строк из таблицы Contacts.
    Dim cmdDelete As New SqlCommand( "DELETE FROM Contacts" & _
      " WHERE ContactID=" & intContactID, _
      ContactMgmt, transDelete)
    ' Выполнить команды.
    Try
        Dim intRows As Integer
        ' Удалить строку из таблицы Contacts.
        intRows = cmdDelete.ExecuteNonQuery()
        ' Удалить из таблицы Calls строки с этим значением ContactID.
        cmdDelete.CommandText = "DELETE FROM Calls WHERE " & _
          " ContactID=" & intContactID
        intRows - intRows + cmdDelete.ExecuteNonQuery()
        ' Зафиксировать транзакцию.
        transDelete.Commit()
        ' Вернуть сообщение об успешном завершении операции.
        Return intRows & " deleted."
    Catch
        ' В случае ошибки восстановить состояние БД.
        transDelete.Rollback()
        ' Вернуть сообщение об ошибке,
        Return "Contact could not be deleted."
    Finally
        ' Закрыть БД.
        ContactMgmt.Close()
    End Try
End Function
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Заполнить набор данных.
     adptContacts.Fill(dsContacts);
        // Осуществить привязку к набору данных.
        grdContacts.DataBind();
}

   private void grdContacts_ItemCommand(object sender,
     System.Web.UI.WebControls.DataGridCommandEventArgs e)
   {
        // Если пользователь щелкнул кнопку Delete...
        if (e.CommandName == "Delete")
        {
```

```
            int intContactID;
            // ...получить ContactID для выбранной строки.
            intContactID = Convert.ToInt16(grdContacts.Items
               [e.Item.ItemIndex].Cells[3].Text);
            // Удалить сведения о контактном лице.
            lblStatus.Text = DeleteContact(intContactID);
        }
        // Обновить набор данных.
        adptContacts.Fill(dsContacts);
        // Обновить DataGrid.
        grdContacts.DataBind();
}

string DeleteContact(int intContactID)
{
        // Открыть соединение с БД.
        ContactMgmt.Open();
        // Объявить объект-транзакцию.
        SqlTransaction transDelete;
        // Создать транзакцию.
        transDelete = ContactMgmt.BeginTransaction
          (IsolationLevel.ReadCommitted);
        // Создать команду для удаления строк из таблицы Contacts.
        SqlCommand cmdDelete = new SqlCommand("DELETE FROM
        Contacts" + " WHERE ContactID=" + intContactID.ToString(),
          ContactMgmt, transDelete);
        // Исполнить команды.
        try
        {
            int intRows;
            // Удалить строку из таблицы Contacts.
            intRows = cmdDelete.ExecuteNonQuery();
            // Удалить из таблицы Calls строки с этим значением ContactID.
            cmdDelete.CommandText = "DELETE FROM Calls WHERE " +
              " ContactID=" + intContactID.ToString();
            intRows = intRows + cmdDelete.ExecuteNonQuery();
            // Зафиксировать транзакцию.
            transDelete.Commit();
            // Вернуть сообщение об успешном завершении,
            return intRows.ToString() + " deleted.";
        }
        catch
        {
            // В случае ошибки восстановить состояние БД,
            transDelete.Rollback();
            // Вернуть сообщение об ошибке.
            return "Contact could not be deleted.";
```

```
        }
        finally
        {
            // Закрыть БД.
            ContactMgmt.Close();
        }
    }
```

Объект-транзакция определяет с помощью свойства *IsolationLevel*, как обрабатываются одновременные коррективы БД. Уровень изоляции транзакции изменяется в зависимости от значений, указанных в таблице 5-2.

Таблица 5-2. Значения свойства *IsolationLevel* (уровень изоляции транзакции)

Значение	Описание
Read Uncommitted	Не блокирует читаемые записи. Это означает, что возможно чтение изменения, еще не зафиксированного в БД либо уже отмененного другим клиентом; в результате локальная копия записи будет отличаться от записи, хранимой в БД. Такая операция называется *«грязное» чтение* (dirty read), поскольку при этом согласованность данных может нарушаться
Chaos	Действует аналогично *ReadUncommitted*, отличаясь лишь тем, что при записи проверяет уровень изо-ляции незавершенных транзакций. Это делается, чтобы случайно не переписать результаты транзакций с более высоким уровнем изоляции
ReadCommitted	Блокирует записи на время чтения, освобождая блокировку сразу после завершения чтения. Это запрещает другим клиентам читать неподтвержденные изменения, но позволяет им добавлять, удалять и модифицировать записи во время транзакции. Такой уровень изоляции задан по умолчанию
ReapeatableRead	Блокирует записи на время чтения и удерживает блокировку до завершения транзакции, гарантируя, что читаемые данные не изменятся в ходе транзакции
Serializable	Блокирует весь набор данных на время чтения и удерживает блокировку до завершения транзакции, гарантируя, что данные и их порядок в БД не изменятся в ходе транзакции

При обработке транзакций соединения с БД SQL предоставляют одну возможность, не доступную при использовании соединений с БД OLE, а именно — создание внутри транзакции так называемых *точек сохранения* (save point) или контрольных точек. Точка сохранения позволяет вернуть БД в состояние, в котором она находилась на момент прохождения этой точки во время текущей транзакции. Для установки точки сохранения в транзакциях БД SQL служит метод *Save:*

Visual Basic .NET

```
transDelete.Save("FirstStep")
```

Visual C#

```
transDelete.Save("FirstStep");
```

Чтобы откатить транзакцию SQL до точки сохранения, при вызове метода *Rollback* укажите имя точки сохранения:

Visual Basic .NET

```
transDelete.Rollback("FirstStep")
```

Visual C#

```
transDelete.Rollback("FirstStep");
```

Корпоративные транзакции

Поскольку транзакции могут затрагивать несколько Web-форм и даже компонентов распределенного приложения, в ASP.NET предоставляется возможность Web-формам работать со службой DTC.

Для использования DTC из Web-формы выполните следующие действия:

1. Перед началом новой или продолжением имеющейся транзакции установите для элемента документа атрибут *transaction*; например, следующая директива *@Page* начинает новую транзакцию:

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="PageTransactions.aspx.vb"
Inherits="vbTransactionSnippet.PageTransactions" Transaction="RequiresNew"%>
```

2. Далее добавьте к проекту ссылку на пространство имен *System.EnterpriseServices*.
3. Чтобы задать для транзакции нужное состояние, используйте методы *SetAbort* и *SetComplete* класса *ContextUtil*.
4. В зависимости от состояния транзакции, вызовите метод *CommitTransaction* либо *AbortTransaction* класса *Page*.

Дополнительные сведения о DTC вы найдете в справочной системе Visual Studio .NET.

Резюме

- Для доступа к данным в ADO.NET используются три ключевых компонента: соединение с БД, адаптер данных и набор данных.
- Чтобы получить доступ к данным, выполните следующие действия:
  - создайте соединение с БД;
  - создайте набор данных, используя объект адаптера;
  - используйте набор данных для отображения или модификации содержимого БД;
  - запишите в БД содержимое набора данных;
  - закройте соединение с БД.
- ADO.NET поддерживает типизированные наборы данных, которые допускают явное применение имен и типов данных, что снижает вероятность ошибок при программировании.
- Привязка данных позволяет оперативно отображать наборы данных на Web-формах с помощью DataGrid, DataList и других элементов управления, представляющих списки.
- Объекты команд создаются для исполнения команд SQL непосредственно в БД. Объекты команд могут возвращать единственное значение (*ExecuteScalar*), модифицировать БД (*ExecuteNonQuery*) или возвращать наборы данных (*ExecuteReader*).
- Транзакции — это группы команд БД, обеспечивающие успешное (или неудачное) исполнение всех команд группы по принципу «все или ничего», что исключает возможность внесения изменений по частям, поддерживая тем самым целостность БД,
- Для определения команд, составляющих транзакцию, используются критерии ACID: транзакция должна быть *атомарной, согласованной, изолированной и устойчивой.*
- Чтобы использовать транзакцию, выполните следующие действия:
  - начните транзакцию;
  - выполните команды и внесите изменения, составляющие транзакцию;
  - проверьте наличие ошибок;
  - если обнаружены ошибки, отмените (откатите) изменения, в противном случае зафиксируйте изменения. После этого транзакция будет завершена.
- Для управления транзакциями в наборах данных предназначены методы *Update* и *RejectChanges*, которые соответственно вносят или отменяют (откатывают) изменения.
- Для управления транзакциями в БД предусмотрены методы *Commit* и *Rollback* объекта транзакции.

Практикум 5. Создание диспетчера контактов



Цель этого практикума — создание многостраничного Web-приложения, которое будет использовать БД для хранения сведений о контактных лицах и их звонках. Назначение приложения — добавление контактных лиц, просмотр и добавление звонков контактного лица, а также добавление новых типов контактных лиц. Диспетчер контактов использует БД Contacts (Contacts.mdb), которая хранится на прилагаемом компакт-диске.

При создании приложения применяются следующие ключевые понятия:

- **переход между страницами** — для решения определенных задач пользователи переходят с главной страницы на страницы, которые позволяют решать эти задачи; а затем возвращаются на главную страницу приложения;
- **централизованное соединение с источником данных** — объекты соединения с источником данных, адаптера и набора данных находятся в файле Global.asax и инициализируются однократно для каждого пользователя; эти компоненты являются общими для всех Web-форм, которые применяют сеансовые переменные состояния;
- **транзакции в наборах данных** — все модификации БД обрабатываются через наборы данных, а код, отвечающий за внесение любых изменений, заключен в структуры обработки ошибок. Если модификация завершилась неудачей, пользователь может повторить попытку или отменить операцию — в любом случае целостность БД не нарушается;
- **использование свободных и привязанных к данным элементов управления** — звонки отображаются с помощью элемента управления DataList, заполненного элементами управления Label. Последние привязаны к данным, а сведения о контактных лицах — в раскрывающемся списке с помощью кода, а не привязки данных.
- **команды БД** — приложение получает значение уникального ключа путем исполнения команды БД.

Продолжительность практикума — около 90 минут.

Упражнение 1. Создание проекта и формы SwitchBoard



Сейчас вы создадите новый проект и главную Web-форму, которая будет использоваться для перехода на другие Web-формы приложения. Это очень простая форма без исполняемого кода.

Готовая форма SwitchBoard выглядит, как на рис. 5-23.

▶ Создание нового приложения и Web-формы SwitchBoard

1. Создайте новый проект ASP.NET Web application и назовите его Contact Management.
2. Переименуйте файл Webform1.aspx в SwitchBoard.aspx и отобразите окно с Web-формой в режим HTML.
3. Введите между тэгами <*form*> и </*form*> следующий HTML-код:

```
<h2>Contact Management Switchboard</h2>
    <P>What rtould you like to do? </P>
    <P><a href="Calls.aspx">View/add calls.</a></P>
    <PXa href="AddContact.aspx">Add a contact.</a></P>
    <P><a href="DeleteContact.aspx">Delete a contact.</a></P>
    <P><a href="ContactTypes.aspx">Add a contact type.</a></P>
```

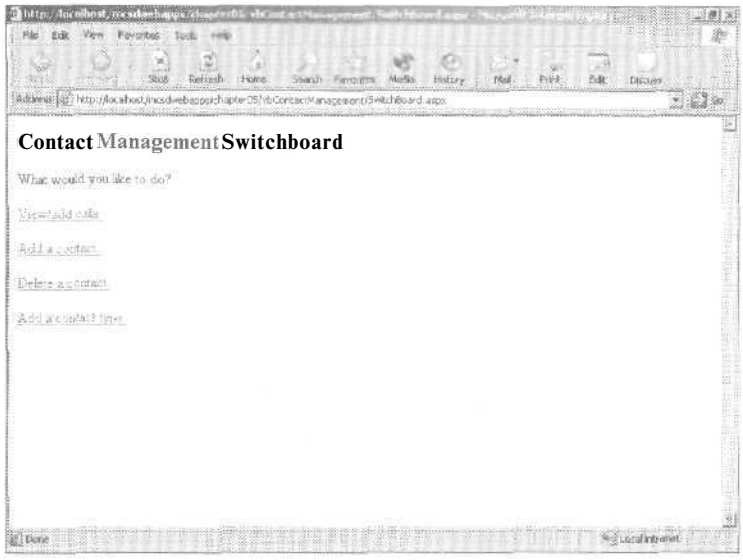


Рис. *5-23.* **Форма SwitchBoard**

Упражнение *2.* Добавление компонентов, необходимых для работы с базой данных



Приложение Contact Management осуществляет доступ к данным из нескольких Web-форм. Чтобы объекты соединения с БД, адаптера и набора данных оказались доступными соответствующим Web-формам, они определяются в файле Global.asax и сохраняются в сеансовых переменных состояния.

Все Web-формы и их пользователи применяют одно общее соединение с БД. Это экономит ресурсы и облегчает управление параметрами объектов соединения и адаптера во время разработки приложения, поскольку все компоненты приложения, требуемые для работы с БД. собраны в одном месте — в файле Global.asax.

▶ **Добавление к приложению компонентов для работы с БД**

1. Откройте файл Global.asax и щелкните ссылку Server Explorer — Visual Studio откроет окно Server Explorer.
2. Щелкните кнопку Connect To A Database в окне Server Explorer, в результате Visual Studio откроет диалоговое окно DataLink Properties.
3. Выберите сервер SQL, на котором установлена БД Contacts, щелкните Use Windows NT integrated security и введите в текстовое поле Select database on the Server имя **Contacts**. Щелкните кнопку Test Connection, чтобы проверить заданные параметры, затем OK. Visual Studio добавит созданное соединение с БД в окно Server Explorer.
4. В окне Server Explorer откройте список элементов созданного соединения, щелкнув знак «+» слева от значка соединения Contacts.dbo, затем аналогичным образом раскройте узел Tables. Перетащите таблицы Calls, ContactTypes и Contacts из окна Solution Explorer на страницу Global.asax. Visual Studio добавит компонент соединения с БД и компоненты адаптера данных для каждой из этих таблиц.
5. Каждый адаптер данных представляет одну из таблиц, которую вы перетащили на страницу. Щелкните правой кнопкой один из адаптеров данных и выберите из контекст-

ного меню команду Generate Data Set — Visual Studio откроет диалоговое окно Generate DataSet.

6. Введите в текстовое поле New имя набора данных, который нужно создать. Имя набора данных должно быть связано с именем таблицы, которую он представляет (например, у набора данных, созданного на основе таблицы Calls, должно быть имя *dsCalls*). После этого установите флажок Add This Dataset To The Designer и щелкните OK, Visual Studio добавит набор данных к странице Global.asax.

Примечание При добавлении набора данных Visual Studio добавляет к его имени цифру; чтобы удалить ее, отредактируйте имя набора данных в окне его свойств.

7. Повторите пункты 5 и 6 для каждого адаптера данных.
8. Дважды щелкните страницу Globabl.asax в окне Code, чтобы открыть код страницы. Добавьте к нему следующий код обработчика события *Session_Start*, объявляющий сеансовые переменные состояния, которые необходимы другим Web-формам для доступа к компонентам, обеспечивающим работу с БД:

Visual Basic .NET

```
Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Соединение с БД.
    Session("ContactMgmt") = SqlConnection1
    ' Адаптеры данных.
    Session("adptCalls") - SqlDataAdapter1
    Session("adptContactTypes") = SqlDataAdapter2
    Session("adptContacts") - SqlDataAdapter3
    ' Наборы данных.
    Session("dsCalls") = dsCalls
    Session("dsContactTypes") = dsContactTypes
    Session("dsContacts") - dsContacts
End Sub
```

Visual C#

```
protected void Session_Start(Object sender, EventArgs e)
{
    // Соединение с БД.
    Session["ContactMgmt"] = SqlConnection1;
    // Адаптеры данных.
    Session["adptCalls"] = SqlDataAdapter1;
    Session["adptContactTypes"] = SqlDataAdapter2;
    Session["adptContacts"] = SqlDataAdapter3;
    // Наборы данных.
    Session["dsCalls"] = dsCalls;
    Session["dsContactTypes"]=dsContactTypes;
    Session["dsContacts"] = osContacts;
}
```

Упражнение 3. Создание формы AddContact

Для добавления сведений о контактах в приложении служит Web-форма AddContact — простая форма с текстовыми полями. Она позволяет ввести значения для всех полей таблицы Contact. Готовая форма AddContact показана на рис. 5-24.

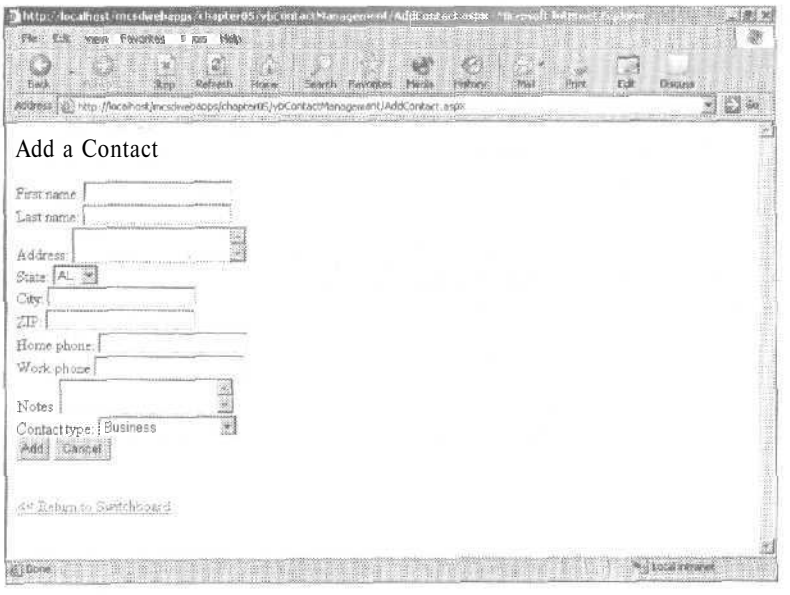


Рис. 5-24. Web-форма AddContact

► Создание Web-формы для ввода данных AddContact

1. Добавьте к проекту новую Web-форму и назовите ее **AddContact.aspx**.
2. В окне Properties выберите объект *DOCUMENTu* задайте для свойства *pageLayout* значение *FlowLayout.*
3. Разместите на Web-форме элементы управления, перечисленные в следующей таблице, и задайте их свойства, как указано в этой таблице.

Элемент управления	Свойство	Значение
Text Box	ID	txtFirstName
TextBox	ID	txtLastName
Text Box	ID	txtAddress
	Text Mode	MultiLine
TextBox	ID	txtCity
Drop Down List	ID	drpStates
	DataSource	arrState
TextBox	ID	txtZIP
TextBox	ID	txtHomePhone
TextBox	ID	txtWorkPhone
TextBox	ID	txtNotes

(окончание)

Элемент управления	Свойство	Значение
DropDownList	ID	drpContactTypes
Button	ID	butAdd
	Text	Add
Button	ID	butCancel
	Text	Cancel
Literal	ID	litStatus
Hyperlink	Text	Return to SwitchBoard
	NavigateUrl	SwitchBoard.aspx

4. Введите подписи полей прямо на Web-форме, для перехода на новую строку используйте Enter.
5. Вызовите окно Code, дважды щелкнув Web-форму, и добавьте в начало модуля следующие операторы:

Visual Basic .NET

```
Imports System.Data.SqlClient
```

Visual C#

```
using System.Data.SqlClient;
```

6. Web-форма AddContact использует компоненты БД. созданные в файле Global.asax во время выполнения упражнения 2. Чтобы получить возможность применять эти компоненты в Web-форме, объявите переменные на уровне класса и получите ссылки на эти компоненты из переменных состояния, где они хранятся. Следующий код демонстрирует, как это делается:

Visual Basic .NET

```
Dim ContactMgmt As SqlConnection
Din adptContactTypes As SqlDataAdapter
Dim adptContacts As SqlDataAdapter
Dim dsContacts As New dsContacts()
' Это открытые переменные цля привязки данных.
Public dsContactTypes As New dsContactTypes()
Public arrState As String() = {"AL", "AK", "AR", "AZ", _
  "CA", "CO", "CT", "DE", "FL","GA","HI". "ID". "IL", _
  "IN", "IA", "KS", "KY", "LA","MA","ME","MD","MI", _
  "MN", "MO", "MS", "MT", "NC", "ND", "NE", "NH", "NJ", _
  "NM", "NV", "NY", "OH", "OK","OR". "PA", "RI","SC", _
  "SD", "TN", "TX", "UT", "VA", "VT", "WA", "WI", "WY"}

Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Получить обьекты соединения и адаптера из переменной
    ' состояния Session,
```

```
    ContactMgmt - Session("ContactMgmt")
    adptContactTypes - Session("adptContactTypes")
    adptContacts = Session("adptContacts")
    ' Заполните набор дачных ContactType,
    adptContactTypes.Fill(dsContactTypes)
    ' Заполните набор данных Contacts.
    adptContacts.Fill(dsContacts)
    ' Связать с данными списки drpContactTypes и drpState.
    drpContactTypes.DataBind()
    drpStates.DataBind()
End Sub
```

Visual C#

```
SqlConnectionContactMgmt;
SqlDataAdapter adptContactTypes:
SqlDataAdapter  adptContacts;
dsContacts dsContacts = new dsContacts();
// Это открытые переменные для привязки данных,
public dsContactTypes dsContactTypes - new dsContactTypes();
oublic string[] arrState = {"AL", "AK", "AR", "AZ",
  "CA", "CO", "CT", "DE", "FL", "GA", "HI", "ID", "IL",
  "IN", "IA", "KS", "KY", "LA", "MA", "ME", "MD", "MI",
  "MN", "MO", "MS", "MT", "NC", "ND", "NE", "NH", "NJ",
  "NM", "NV", "NY", "OH", "OK", "OR", "PA", "RI", "SC",
  "SD", "TN", "TX", "UT", "VA", "VT", "WA", "WI", "WY"};

private void Page_Load(object sender, System.EventArgs e)
{
    // Получить объекты соединения и адаптера из переменной
     // состояния Session.
    ContactMgmt - (SqlConnection)Session["ContactMgmt"];
    adptContactTypes - (SqlDataAdapter)Session["adptContactTypes"];
    adptContacts = (SqlDataAdapter)Session["adptContacts"];
    // Заполнить набор данных ContactType.
    adptContactTypes.Fill(dsContactTypes, "Contact_Types");
    // Заполнить набор данных Contacts.
    adptContacts.Fill(dsContacts);
    // Связать с данными списки drpContactTypes и drpState.
    drpContactTypes.DataBind();
    drpStates.DataBind();
}
```

7. Каждый контакт в таблице Contacts идентифицируется значением поля ContactID. Это первичный ключ таблицы, следовательно, его значение должно быть уникальным для каждого контактного лица. Поэтому перед добавлением к набору данных Contacts сведений о контактном лице необходимо получить из **БД** новое уникальное значение ContactlD, для чего к классу Web-формы потребуется добавить следующую вспомогательную функцию:

Visual Basic .NET

```
' Эта функция возвращает новое уникальное значение ContactID.
Private Function GetNewContactID() As Integer
    Dim cmdNewID As New SqlCommand("SELECT MAX(ContactID)" & _
      " FROM Contacts", ContactMgmt)
    ' Открыть соединение с БД.
    ContactMgmt.Open()
    ' Получить наибольшее значение ключа.
    Dim intID As Integer
    ' Проверить, существует ли запись.
    If Not (cmdNewID.ExecuteScalar() Is DBNull.Value) Then
    ' Получить наибольшее число...
        intID = CInt(cmdNewID.ExecuteScalar())
    Else
        ' ...или присвоить intID значение 0.
        intID = 0
    End If
    ' Закрыть соединение.
    ContactMgmt.Close()
    ' Вернуть новое значение ключа (оно будет на единицу больше прежнего).
    Return intID + 1
End Function
```

Visual C#

```
// Эта функция возвращает новое уникальное значение ContactlD.
int GetNewContactID()
{
    SqlCommand cmdNewID = new SqlCommand("SELECT " +
    "MAX(ContactID) FROM Contacts", ContactMgmt);
    // Открыть соединение с БД.
    ContactMgmt.OpenO;
    // Получить наибольшее значение ключа.
 int intID;
    // Проверить, существует ли запись,
    if (cmdNewID.ExecuteScalar() != DBNull.Value)
        // Получить наибольшее число...
        intID = (int)cmdNewID.ExecuteScalar();
    else
       // ...или присвоить intID значение 0.
       intID = 0;
    // Закрыть соединение.
    ContactMgmt.Close();
    // Вернуть новое значение ключа (оно будет на единицу больше прежнего).
 return intID + 1;
}
```

8. Далее необходимо добавить обработчик события *butAdd_Click*, который позволяет вставить в БД сведения о новом контактном лице:

Visual Basic .NET

```
Private Sub butAdd_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butAdd.Click
    ' Создать новую строку набора данных.
    Dim rowNew As dsContacts.ContactsRow
    rowNew = dsContacts.Contacts.NewContactsRow()
    ' Записать в строку данные.
    rowNew.ContactID - GetNewContactID()
    rowNew.FirstName = txtFirstName.Text
    rowNew.LastName = txtLastName.Text
    rowNew.Address = txtAddress.Text
    rowNew.City = txtCity.Text
    rowNew.StateOrProvince - drpStates.SelectedItem.Text
    rowNew.PostalCode - txtZip.Text
    rowNew.HomePhone = txtHomePhone.Text
    rowNew.WorkPhone = txtWorkPhone.Text
    rowNew.Notes = txtNotes.Text
    rowNew.ContactTypeID - drpContactTypes.SelectedItem.Value
    ' Добавить строку к набору данных.
    dsContacts.Contacts.AddContactsRow(rowNew)
    Try
        ' Обновить БД.
        adptContacts.Update(dsContacts)
        ' Уведомить об успешном завершении операции.
        litStatus.Text = rowNew.FirstName & " " & rowNew.LastName & _
          " added successfully.<br>"
        ' Перезагрузить страницу, чтобы очистить поля.
        Server.Transfer("AddContact.aspx")
    Catch ex As Exception
        litStatus.Text = "The following error occurred:<br>" & _
          ex.Message & "<br>" & _
          "Correct the error and click Add to add the contact " & _
          "or click Cancel to abort.<br>"
    End Try
End Sub
```

Visual C#

```
private void butAdd_Click(object sender, System.EventArgs e)
{
     // Создать новую строку набора данных.
     dsContacts.ContactsRow rowNew:
     rowNew = dsContacts.Contacts.NewContactsRow();
     // Записать в строку данные,
     rowNew.ContactID = GetNewContactID();
```

```
        rowNew.FirstName = txtFirstName.Text;
        rowNew.LastName = txtLastName.Text;
        rowNew.Address = txtAddress.Text;
        rowNew.City = txtCity.Text;
        rowNew.StateOrProvince = drpStates.SelectedItem.Text;
        rowNew.PostalCode = txtZip.Text;
        rowNew.HomePhone = txtHomePhone.Text;
        rowNew.WorkPhone = txtWorkPhone.Text;
        rowNew.Notes = txtNotes.Text;
        rowNew.ContactTypeID = Convert.ToInt16
          (drpContactTypes.SelectedItem.Value);
        // Добавить строку к набору данных.
        dsContacts.Contacts.AddContactsRow(rowNew);
        try
        {
           // Обновить БД.
           adptContacts.Update(dsContacts);
           // Уведомить об успешном завершении операции.
           litStatus.Text = rowNew.FirstName + " " + rowNew.LastName +
             " added successfully.<br>";
           // Перезагрузить страницу, чтобы очистить поля.
           Server.Transfer("AddContact.aspx");
        }
        catch (Exception ex)
        {
           litStatus.Text = "The following error occurred:<br>" +
           ex.Message + "<br>" +
           "Correct the error and click Add to add the contact " +
                "or click Cancel to abort.<br>";
        }
    }
```

9. Добавьте обработчик события *butCancel_Click*, чтобы пользователь смог отменить операцию и очистить текстовые поля формы:

Visual Basic .NET

```
Private Sub butCancel_ServerClick(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butCancel.ServerClick
    ' Перезагрузить страницу, чтобы очистить поля.
    Server.Transfer("AddContact.aspx")
End Sub
```

Visual C#

```
private void butCancel_ServerClick(object sender, System.EventArgs e)
{
    // Перезагрузить страницу, чтобы очистить поля.
    Server.Transfer("AddContact.aspx");
}
```

Упражнение 4. Создание формы Calls



Web-форма Calls позволяет пользователям вводить и просматривать сведения о телефонных звонках для некоторого контактного лица. Форма содержит элемент управления DataList, позволяющий просматривать звонки контактного лица, указанного в списке элемента управления DropDownList, а также элементы управления, добавляющие сведения о звонке в элемент управления Panel. Таким образом удается скрывать и отображать их, когда пользователь щелкает кнопку Add.

Готовая Web-форма Calls выглядит, как на рис. 5-25.

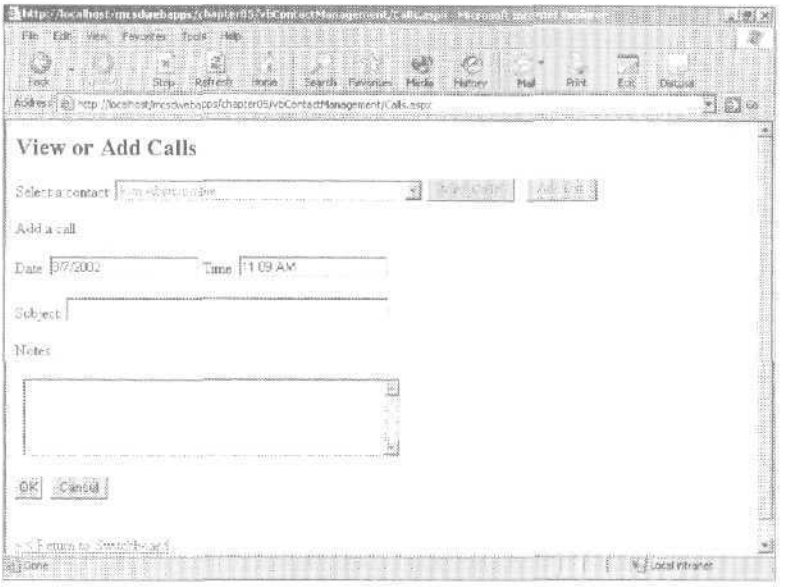


Рис. 5-25. Форма Calls

▶ **Создание Web-формы Calls**

1. Добавьте к проекту новую Web-форму и назовите ее Calls.aspx.
2. Разместите на Web-форме элементы управления, перечисленные в следующей таблице, и задайте их свойства, как указано в этой таблице. Элементы управления, расположенные в таблице ниже Panel, следует разместить внутри панели, чтобы облегчить контроль их отображения во время выполнения.

Элемент управления	Свойство	Значение
DropDownList	ID	DrpContacts
Button	ID	ButView
	Text	View Call
Button	ID	ButAddCall
	Text	Add Call
Panel	ID	PnlAdd
Text Box	ID	TxtDate
Text Box	ID	TxtTime
Text Box	ID	TxtSubject

(окончание)

Элемент управления	Свойство	Значение
TextBox	ID	TxtNotes
	TextMode	Multiline
Button	ID	ButOKAdd
Button	ID	butCancelAdd
Literal	ID	litError

3. В Toolbox перейдите на вкладку Data и перетащите на Web-форму компонент DataSet — Visual Studio откроет диалоговое окно Add Dataset, в котором установлен переключатель Typed Dataset.
4. В раскрывающемся списке Name выберите типизированный набор данных ContactManagement.dsCalls и щелкните OK. Visual Studio добавит набор данных к Web-форме.
5. Перетащите на Web-форму элемент управления DataList с вкладки WebForms в Toolbox, поместив его ниже панели. Задайте его свойства и шаблоны, как указано в следующей таблице.

Элемент управления	**Свойство**	**Значение**
DataList	ID	dlstCalls
	DataSource	dsCalls
	Шаблон верхнего колонтитула DataList	
Текст	Нет	Calls
HTML-тэг, определяющий горизонтальную линию	Нет	<HR>
	Шаблон верхнего колонтитула DataList	
HTML определяющий горизонтальную линию	Нет	<HR>
	Шаблон элемента DataList	
Label	ID	lblDate
	(DataBindings)	В диалоговом окне DataBinding выберите в дереве Simple Binding элемент Container.DataItem.CallDate и задайте форматирующее выражение {0:d}
Label	ID	LblTime
	(DataBindings)	В диалоговом окне DataBinding выберите в дереве Simple Binding элемент Container.DataItem.CallTime и задайте форматирующее выражение {0:d}
Текст	Нет	Subject:
Label	ID	LblSubject
	(DataBindings)	В диалоговом окне DataBinding выберите в дереве Simple Binding элемент Container.DataItem.Subject
Текст	Нет	Notes:

(окончание)

Элемент управления	Свойство	Значение
Label	**ID**	LblNotes
	(DataBindings)	**В диалоговом окне** DataBinding выберите в дереве Simple Binding элемент Container.DataItem.Notes
	***Шаблон** разделителя*	
HTML-тэг, определяющий горизонтальную линию	**Нет**	<HR>

6. Скопируйте элемент управления Hyperlink из формы AddContact (он расположен в нижней части этой формы) и поместите его внизу формы Calls. Чтобы скопировать элемент управления, выберите его и нажмите Ctrl+C. Чтобы вставить скопированный элемент в форму Calls, шелкните форму (в режиме Design) и нажмите Ctrl+V.
7. Вызовите окно Code, дважды щелкнув левой кнопкой мыши Web-форму Calls, и добавьте в начало модуля следующий оператор:

Visual Basic .NET

```
Imports System.Data.SqlClient
```

Visual C#

```
using System.Data.SqlClient;
```

8. Web-форма Calls использует для доступа к данным компоненты, определенные в файле Global.asax, поэтому вам понадобятся ссылки на компоненты (они хранятся в переменных состояния), созданные во время выполнения упражнения 2. Ниже показан код, объявляющий ряд объектов и обрабатывающий событие *Page_Load*. Этот код получает ссылки на компоненты для работы с БД, делая их доступными коду формы:

Visual Basic .NET

```
Dim ContactMgmt As SqlConnection
Dim adptContacts As SqlDataAdapter
Dim adptCalls As SqlDataAdapter
Dim dsContacts As New dsContacts()

Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Получить сеансовые переменные состояния,
   ContactMgmt = Session("ContactMgmt")
   adptContacts = Session("adptContacts")
   adptCalls = Session("adptCalls")
   dsCalls - Session("dsCalls")
   dsContacts = Session("dsContacts")
   ' Этот код выполняется при первом отображении формы.
   If Not IsPostBack Then
       ' Заполнить набор данных Contacts.
       adptContacts.Fill(dsContacts)
```

```
        ' Для каждой строки таблицы...
        Dim rowNext As dsContacts.ContactsRow
        For Each rowNext In dsContacts.Contacts
            ' ...создать новый элемент списка,
            Dim lstNew As New ListItem()
            lstNew.Text = rowNext.FirstName & " " & _
            rowNext.LastName
            lstNew.Value = rowNext.ContactID
            ' добавить элемент к раскрывающемуся списку.
            drpContacts.Items.Add(lstNew)
        Next
        ' Выбрать первый элемент в списке.
        drpContacts.SelectedIndex = 0
    End If
End Sub
```

Visual C#

```
SqlConnection ContactMgmt;
SqlDataAdapter adptContacts;
SqlDataAdapter adptCalls;
dsContacts dsContacts = new dsContacts();

private void Page_Load(object sender, System.EventArgs e)
{
    // Получить сеансовые переменные состояния.
    ContactMgmt = (SqlConnection)Session["ContactMgmt"];
    adptContacts = (SqlDataAdapter)Session["adptContacts"];
    adptCalls = (SqlDataAdapter)Session["adptCalls"];
    dsCalls = (dsCalls)Session["dsCalls"];
    dsContacts = (dsContacts)Session["dsContacts"];
    // Этот код выполняется при первом отображении формы,
    if (!IsPostBack)
    {
        // Заполнить набор данных Contacts.
        adptContacts.Fill(dsContacts);
        // Для каждой строки таблицы...
        foreach (dsContacts.ContactsRow rowNext in
        dsContacts.Contacts)
        {
            // ...создать новый элемент списка,
            ListItem lstNew = new ListItem();
            lstNew.Text = rowNext.FirstName + " " +
        rowNext.LastName;
            lstNew.Value = rowNext.ContactID.ToString();
            // добавить элемент к раскрывающемуся списку.
            drpContacts.Items.Add(lstNew);
        }
```

```
        // Выбрать первый элемент в списке,
        drpContacts.SelectedIndex = 0;
    }
}
```

9. Web-форма Calls модифицирует набор данных Calls во время выполнения. Чтобы после обработки страницы записать набор данных обратно в переменную состояния, необходимо добавить к обработчику события *Page_PreRender*(это последнее событие, которое обрабатывается перед отображением страницы) следующий код:

Visual Basic .NET

```
Private Sub Page_PreRender(ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles MyBase.PreRender
    ' Записать набор данных в сеансовую переменную состояния.
    Session("dsCalls") = dsCalls
End Sub
```

Visual C#

```
private void Page_PreRender(object sender, System.EventArgs e)
{
        // Записать набор данных в сеансовую переменную состояния.
        Session["dsCalls"] = dsCalls;
}
```

10. Добавьте к обработчикам события *Click* кнопок View и Add следующий код:

Visual Basic .NET

```
Private Sub butView_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butView.Click
    ' Создать команду для выборки из набора данных.
    adptCalls.SelectCommand.CommandText = "SELECT * FROM _
    Calls " & _
      "WHERE ContactID=" & drpContacts.SelectedItem.Value
    ' Очистить набор данных.
     dsCalls.Clear
    ' Заполнить набор данных сведениями о звонках,
    adptCalls.Fill(dsCalls)
    ' Связать элемент управления DataList с данными.
    dlstCalls.DataBind()
End Sub

Private Sub butAddCall_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butAddCall.Click
    ' Задать исходные значения даты и времени.
    txtDate.Text = Now.ToShortDateString()
    txtTime.Text = Now.ToShortTimeString()
    ' Сделать панель для добавления звонков видимой.
    pnlAdd.Visible = True
```

```
    ' Деактивировать кнопки и раскрывающийся список.
    butView.Enabled = False
    butAddCall.Enabled = False
    drpContacts.Enabled - False
End Sub
```

Visual C#

```
private void butView_Click(object sender, System.EventArgs e)
{
    // Создать команду для выборки из набора данных.
    adptCalls.SelectCommand.CommandText = "SELECT * FROM_
     Calls " +
      "WHERE ContactID=" + drpContacts.SelectedItem.Value;
    // Очистить набор данных.
    dsCalls.Clear();
    // Заполнить набор данных сведениями о звонках,
    adptCalls.Fill(dsCalls);
    // Связать элемент управления DataList с данными.
    dlstCalls.DataBind();
}

private void butAddCall_Click(object sender, System.EventArgs e)
{
    // Задать исходные значения даты и времени.
    txtDate.Text - System.DateTime.Now.ToShortDateString();
    txtTime.Text = System.DateTime.Now.ToShortTimeString();
    // Сделать панель для добавления звонков видимой.
    pnlAdd.Visible = true;
    // Деактивировать кнопки и раскрывающийся список.
    butView.Enabled = false;
    butAddCall.Enabled = false;
    drpContacts.Enabled = false;
}
```

11. Добавьте вспомогательную функцию, которая получает новое значение CallID. Поскольку поле CallID является первичным ключом в таблице Calls, его значение должно быть уникальным внутри БД:

Visual Basic .NET

```
' Эта функция возвращает новое уникальное значение CallID.
Private Function GetCallID() As Integer
    Dim cmdNewID As New SqlCommand("SELECT MAX(CallID)" & _
      " FROM Calls", ContactMgmt)
    ' Открыть соединение с БД,
    ContactMgmt.Open()
    ' Получить наибольшее значение ключа.
    Dim intID As Integer
```

```
        ' Проверить, существует ли запись,
        If Not (cmdNewID.ExecuteScalar() Is DBNull.Value) Then
            ' Получить наибольшее число...
            intID = CInt(cmdNewID.ExecuteScalar())
        Else
            ' ...или присвоить intID значение 0,
            intID = 0
        End If
        ' Закрыть соединение.
        ContactMgmt.Close()
        ' Вернуть новое значение ключа (оно будет на единицу
        ' больше прежнего),
        Return intID + 1
    End Sub
```

Visual C#

```
int GetCallID()
{
    SqlCommand cmdNewID = new SqlCommand("SELECT MAX(CallID)" +
      " FROM Calls", ContactMgmt);
    // Открыть соединение с БД.
    ContactMgmt.Open();
    // Получить наибольшее значение ключа,
    int intID;
    // Проверить, существует ли запись,
    if (cmdNewID.ExecuteScalar() != DBNull.Value)
       // Получить наибольшее число...
       intID = (int)cmdNewID.ExecuteScalar();
    else
       // ...или присвоить intID значение 0.
       intID = 0;
    // Закрыть соединение
    ContactMgmt.CloseO;
    // Вернуть новое значение ключа (оно будет на единицу больше прежнего),
    return intID + 1;
}
```

12. Добавьте к обработчикам событий кнопок OK и Cancel код, показанный ниже. Он очень похож на обработчики событий, использованные для добавления (или отмены добавления) записей в таблицу Contacts (см. предыдущее упражнение).

Visual Basic .NET

```
Private Sub butOKAdd_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butOKAdd.Click
    Dim rowNew As dsCalls.CallsRow
    rowNew = dsCalls.Calls.NewCallsRow
    rowNew.ContactID = drpContacts.SelectedItem.Value
```

```
    rowNew.CallID = GetCallID()
    rowNew.CallDate = txtDate.Text
    rowNew.CallTime = txtDate.Text & " " & txtTime.Text
    rowNew.Subject = txtSubject.Text
    rowNew.Notes = txtNotes.Text
    ' Добавить строку к набору данных со сведениями о звонках,
    dsCalls.Calls.AddCallsRow(rowNew)
    Try
        adptCalls.Update(dsCalls)
        ' Скрыть панель добавления звонков,
        pnlAdd.Visible = False
        ' Очистить поля,
        txtSubject.Text = ""
        txtNotes.Text = ""
        ' Активировать другие элементы управления.
        butView.Enabled = True
        butAddCall.Enabled = True
        drpContacts.Enabled = True
    Catch ex As Exception
        ' Отобразить сообщение об ошибке.
        litError.Text = "The following error occurred while _
        adding " & _
            "the call:<br>" & _
            ex.Message & "<br>" & _
            "Correct the error and try again, or click Cancel _
            to " & _
            "abort.<br>"
    End Try
End Sub

Private Sub butCancelAdd_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butCancelAdd.Click
    ' Скрыть панель добавления звонков.
    pnlAdd.Visible = False
    ' Очистить поля.
    txtSubject.Text = ""
    txtNotes.Text = ""
    ' Активизировать другие элементы управления.
    butAddCall.Enabled = True
    drpContacts.Enabled = True
End Sub
```

Visual C#

```
private void butOKAdd_Click(object sender, System.EventArgs e)
{
    dsCalls.CallsRowrowNew;
```

```
        rowNew = dsCalls.Calls.NewCallsRow();
        rowNew.ContactID = Convert.ToInt16(drpContacts.SelectedItem.Value);
        rowNew.CallID = GetCallID();
        rowNew.CallDate = Convert.ToDateTime(txtDate.Text);
        rowNew.CallTime = Convert.ToDateTime(txtDate.Text + " "
           + txtTime.Text);
        rowNew.Subject = txtSubject.Text;
        rowNew.Notes = txtNotes.Text;
        // Добавить строку к набору данных со сведениями о
         // звонках.
        dsCalls.Calls.AddCallsRow(rowNew);
        try
        {
           adptCalls.Update(dsCalls);
           // Скрыть панель добавления звонков.
           pnlAdd.Visible = false;
       // Очистить поля.
       txtSubject.Text = "";
       txtNotes.Text = "";
           // Активировать другие элементы управления.
           butView.Enabled = true;
           butAddCall.Enabled = true;
           drpContacts.Enabled = true;
        }
        catch (Exception ex)
        {
           // Показать сообщение об ошибке.
           litError.Text = "The following error occurred while adding " +
           "the call:<br>" +
           ex.Message + "<br>" +
           "Correct the error and try again, or click Cancel to " +
           "abort.<br>";
        }
}

private void butCancelAdd_Click(object sender, System.EventArgs e)
{
        // Скрыть панель добавления звонков,
        pnlAdd.Visible = false;
       // Очистить поля.
       txtSubject.Text = "";
       txtNotes.Text = "";
        // Активировать другие элементы управления.
        butAddCall.Enabled = true;
        drpContacts.Enabled = true;
}
```

Упражнение 5. Создание форм DeleteContact и ContactTypes



До сих пор мы разбирали создание каждой Web-формы шаг за шагом. Теперь вы попробуете сделать это самостоятельно. Итак, вам необходимо создать Web-формы DeleteContact и ContactTypes, которые должны:

- использовать компоненты доступа к данным, определенные в файле *Global.asax*;
- манипулировать БД. Форма DeleteContact отвечает за удаление из БД сведений о контактных лицах, а форма ContactTypes — за просмотр и добавление новых типов контактных лиц;
- использовать транзакции с наборами данных (см. описание форм AddContact и Calls);
- предоставлять пользователю возможность вернуться на форму SwitchBoard.



Закончив, сравните свои результаты с приложением Contact Management (оно находится на прилагаемом компакт-диске). Удачи вам!

Закрепление материала

Приведенные ниже вопросы помогут Вам лучше усвоить основные темы данной главы, Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении *«Вопросы и ответы»* в конце книги.

1. Как и с помощью каких объектов удается быстро подсчитать число записей в таблице БД?

2. Назовите отличие типизированных наборов данных от нетипизированных; в чем преимущества типизированных наборов данных?

3. Как сделать, чтобы все формы Web-приложения использовали общие компоненты для соединения с БД?

4. Объясните разницу между обработкой транзакций на уровнях набора данных и БД.

ГЛАВА 6

Перехват и исправление ошибок

Занятие 1. Обработка исключений 254

Занятие 2. Страницы сообщений об ошибках 267

Занятие 3. Регистрация исключений 272

В этой главе

Из этой главы вы узнаете, **что** делать **при сбоях в** работе **приложения, вызванных внешними причинами.** Изучив предлагаемый **материал,** вы сумеете **эффективно** предупреждать, **диагностировать и** обрабатывать **подобные ошибки.**

Первые два занятия из этой главы посвящены основным подходам к обработке программных ошибок в Web-приложениях ASP.NET. В занятии 3 разъясняются способы мониторинга приложения во время тестирования и после развертывания, гарантирующего эффективное устранение возникающих неполадок.

Прежде всего

Для изучения материалов этой главы необходимо:

- **знать основы языков Visual Basic или Visual** C#, **включая понятия переменной,** процедуры, структуры решений и области **видимости.** Подробнее **об этих языках — в разделах справочной системы** Visual **Studio** .NET, **озаглавленных «Language Changes in** Visual Basic» **и** «C# **Language** Tour»;
- **уметь** создавать **простейшие** Web-приложения **с** помощью Web-форм, **серверных элементов управления и обработчиков событий, с которыми вы познакомились в** главе **2;**
- **иметь готовый** диспетчер контактов **(созданный** в практикуме **из главы** 5),

Занятие 1. Обработка исключений

Исключения — это непредвиденные обстоятельства, возникающие при работе приложения. В документации Visual Studio .NET термины *исключение* (exception) и *ошибка* (error) считаются взаимозаменяемыми, поскольку те и другие обрабатываются с использованием одинаковых приемов программирования.

Из этого занятия вы узнаете, какие приемы программирования применяются для обработки исключений в Web-приложениях. Здесь обсуждаются только современные методики, доступные в Visual Studio .NET и ASP.NET, Прежний синтаксис обработки исключений типа *On Error* здесь не рассматривается, хотя и поддерживается в Visual Basic .NET.

Изучив материал этого занятия, вы сможете:

- ✓ разъяснять суть основных методов обработки исключений, используемых в Web-приложениях;
- ✓ создавать блоки для перехвата и обработки исключений внутри процедуры;
- ✓ генерировать исключения при возникновении непредвиденной ситуации;
- ✓ определять новые типы исключений, позволяющих приложению описать исключительную ситуацию;
- ✓ обрабатывать исключения при помощи событий *Error*.

Продолжительность занятия — около 30 минут.

Способы обработки исключений

Непредвиденный сбой в работе Web-приложения сразу заметен пользователю. На рис. 6-1 показано, что видит пользователь, когда Web-приложение пытается открыть несуществующий серверный файл.

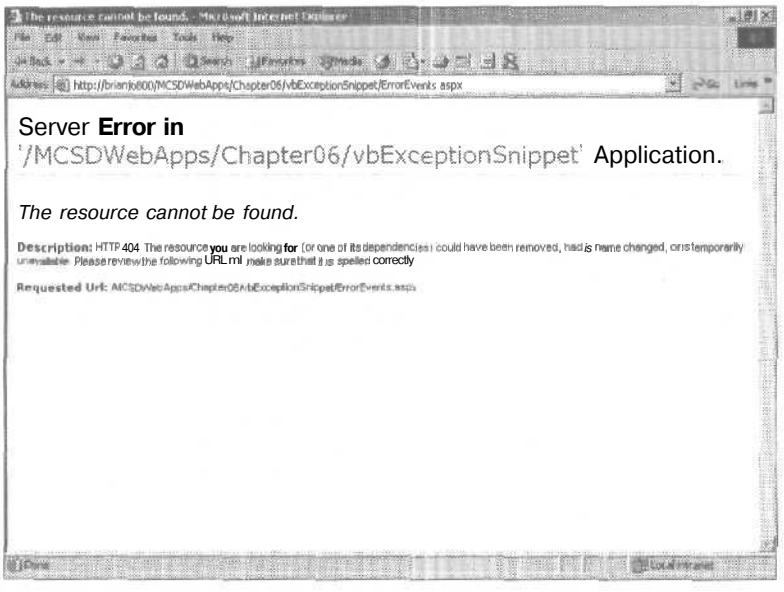


Рис. 6-1. Необработанное исключение

В таких случаях пользователю остается лишь щелкнуть кнопку Back в браузере и повторить попытку. Как разработчик, вы отвечаете за прогнозирование и заблаговременное разрешение подобных ситуаций при написании кода. Процесс разрешения таких ситуаций называется *обработкой исключений* (exception handling), а ошибки, которые не удается обработать программе, — *необработанными исключениями* (unhandled exceptions). Пример необработанного исключения показан на рис. 6-1.

Исключения в Web-приложении обрабатываются двумя способами;

- **с помощью блоков обработки исключений** — применяется для обработки исключений, возникающих в пределах области видимости процедуры. В документации Visual Studio .NET этот метод называется *структурной обработкой исключений* (structured exception handling, SEH);
- **с помощью событий *error*** — служит для обработки исключений, возникающих в пределах области видимости объекта.

Как вы узнаете далее, комбинирование этих подходов позволяет эффективно обрабатывать любые исключения, возникающие во время работы приложения.

Использование блоков обработки исключений

Блоки обработки исключений рекомендуется использовать для защиты операторов, при исполнении которых вероятны сбои по причинам, не зависящим от самого оператора, например оператора, обращающегося к файлу. Причины ошибок, возникающих при сохранении файла на сервере, могут быть самыми разными — переполнение диска, занятое имя или недопустимый путь сохраняемого файла.

Для решения потенциальных проблем такого рода операторы для доступа к файлу следует поместить в блок обработки исключений. Ключевые слова, необходимые для создания блока обработки исключений, перечислены в таблице 6-1.

Таблица 6-1. Ключевые слова, применяемые для обработки исключений

Visual Basic .NET	Visual C#	Назначение
Try	*try*	Открывает структуру обработки исключений. Если при исполнении следующего за ним оператора возникает исключение, управление немедленно передается следующему оператору *Catch* (в C# — *catch*)
Catch	*catch*	Получение и обработка любых исключений. При переходе управления к блоку *Catch (catch)* выполняются расположенные в нем операторы, исправляющие ошибку или обрабатывающие исключение иным способом
Finally	*finally*	Исполнение любых операторов из блока обработки исключения независимо от того, возникло исключение или нет
End Try	нет	Закрывает **блок обработки исключений**
Throw	*throw*	Генерирует исключение. Применяется внутри блоков обработки исключений для немедленной передачи управления оператору *Catch (catch)*

Проще всего разобраться в применении этих ключевых слов на примере. Рассмотрим Web-форму (рис. 6-2), позволяющую выбирать файл на локальном компьютере и загружать его на сервер, где файлы добавляются к списку,

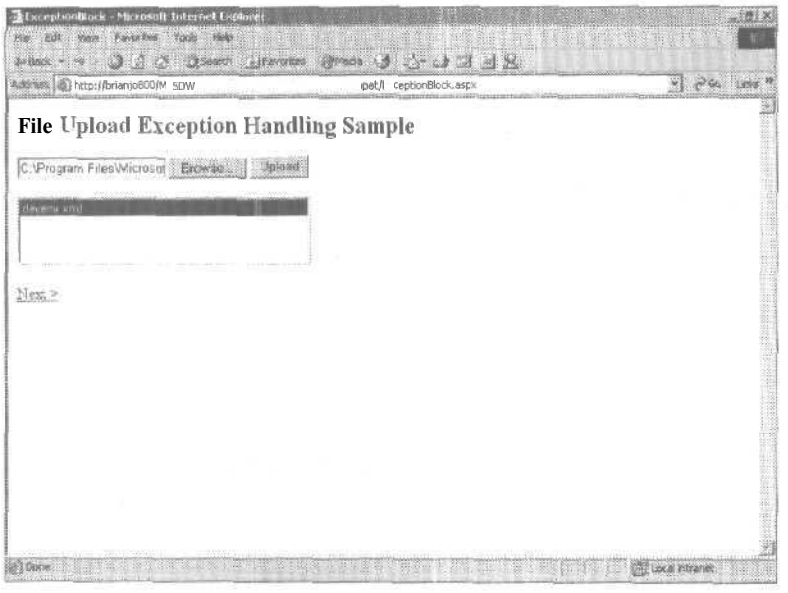


Рис. 6-2. Пример Web-формы для загрузки файлов

После того как пользователь ввел имя файла на Web-форме и щелкнул кнопку Upload, возможны следующие причины исключения:

- **выбранный файл не существует на клиентском компьютере;**
- **возник сбой подключения к Интернету во время загрузки файла;**
- файл слишком велик для передачи через Интернет;
- на диске сервера не хватило свободного места для сохранения файла.

Чтобы справиться с этими и другими исключительными ситуациями, поместите операторы, при исполнении которых они могут возникать, в блок обработки исключений:

Visual Basic .NET

```
Private Sub butUpload_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butUpload.Click
    Dim strFilename As String = fileUpload.PostedFile.Filename
    ' Разбить заданный путь на части.
    strFilename = System.IO.Path.GetFilename(strFilename)
    Try
        ' (1) Проверить, равен ли размер файла нулю (существует ли файл).
        If filUpload.PostedFile.ContentLength = 0 Then
            Throw New System.IO.FileNotFoundException()
        End If
        ' (2) Сохранить закачиваемый файл на сервере
        '     с помощью полученного имени.
        filUpload.PostedFile.SaveAs _
          (Request.MapPath(Request.ApplicationPath) & "\" & _
          strFilename)
        ' Добавить файл к списку серверных файлов.
        lstFiles.Items.Add(strFilename)
```

```
        ' Выделить первый элемент в списке.
        lstFiles.SelectedIndex = 0
    ' (3) Обработать возможные исключения.
    Catch ex As Exception
        litError.Text = "<p>The following error occurred:</p>" & _
          ex.Message & "<br>"
    ' (4) Освободить любые ресурсы.
    Finally
        ' Сбросить значение поля для ввода имени файла.
        filUpload.Value = ""
    End Try
End Sub
```

Visual C#

```
private void butUpload_Click(object sender, System.EventArgs e)
{
    string strFilename = filUpload.PostedFile.FileNarie;
    // Разбить заданный путь на части.
    strFilename = System.IO.Path.GetFileName(strFilename);
    try
    {
    // (1) Проверить, равен ли размер файла нулю (существует ли файл),
    if (filUpload.PostedFile.ContentLength == 0)
        tnrow new System.IO.FileNotFoundException();
    // (2) Сохранить закачиваемый файл на сервере с помощью полученного имени.
    filUpload.PostedFile.SaveAs
        (Request.MapPath(Request.ApplicationPath) + "\\" +
        strFilename);
    // Добавить файл к списку серверных файлов,
    lstFiles.Items.Add(strFilename);
    // Выделить первый элемент в списке.
    lstFiles.SelectedIndex = 0;
    }
    // (3) Обработать возможные исключения,
    catch (Exception ex)
    {
    litError.Text = "<p>The following error occurred:</p>" +
        ex.Message + "<br>";
    }
    // (4) Освободить любые ресурсы.
    finally
    {

    // Сбросить значение поля для ввода имени файла.
    filUpload.Value = "";
    }
}
```

Этот код выполняет следующие действия:

1. проверяет, существует ли заданный файл на клиентском компьютере. Если его размер равен нулю, такой файл не существует и программа генерирует исключение *System.IO.FileNotFoundException*. В результате управление немедленно передается операторам, обрабатывающим исключения *(3)*;
2. сохраняет файл на сервере. Если эта операция прошла успешно, имя файла добавляется к списку в ListBox. В противном случае управление передается операторам, обрабатывающим исключения *(3), а* добавление файла к ListBox не выполняется;
3. обрабатывает возникающие исключения. В этом случае обработка заключается просто в отображении сведений об исключениях на Web-форме при помощи элемента управления Literal. Если не возникает ни одного исключения, управление передается операторам, освобождающим ресурсы *(4)*;
4. выполняет завершающую очистку. Эти операторы завершают блок обработки исключения.

В предыдущем примере ключевое слово *Catch (catch* в примере на C#) используется для перехвата обших типов исключений; оно также позволяет перехватывать специфические типы исключений и по-разному обрабатывать их, что демонстрирует следующий код:

Visual Basic .NET

```
Try
    ' Здесь могут быть какие-нибудь операторы...
Catch ex As System.IO.FileNotFoundException
    litError.Text - "<p>The file does not exist.</p>"
Catch ex As System.IO.IOException
    litError.Text = "<p>The file could not be saved " & _
  "to the server.</p>"
Catch ex As System.Net.WebException
    litError.Text = "<p>An Internet error occurred while " & _
  "uploading the file.</p>"
Catch ex As Exception
    litError.Text - "<p>The following error occurred:</p>" & _
  ex.Message & "<br>"
End Try
```

Visual C#

```
try
{
    // Здесь могут быть какие-нибудь операторы...
}
    catch (System.IO.FileNotFoundException ex)
{
    litError.Text = "<p>The file does not exist.</p>";
}
catch (System.IO.IOException ex)
{
    litError.Text = "<p>The file could not be saved " +
      "to the server.</p>";
}
```

```
catch (System.Net.WebException ex)
;
    litError.Text = "<p>An Internet error occurred while" +
        "uploading the file.</p>";
}
catch (Exception ex)
{
    litError.Text - "<p>The following error occurred:</p>" +
        ex.Message + "<br>";
}
```

В этом фрагменте кода обработка каждого из заданных типов исключений (*FileNotFoundException*, *WebException* или другой) выполняется особым оператором.

Операторы *Catch* обрабатываются последовательно, поэтому первыми следует располагать операторы, обрабатывающие наиболее специфичные типы исключений, а последними — операторы, обрабатывающие общие типы исключений. Например, в следующем фрагменте операторы для обработки исключений *System.Ю.Ю Exception, System.IO. File Not-Found* и *System.Net.WebException* никогда не будут исполнены, поскольку все исключения перехватываются первым блоком, обрабатывающим общий тип исключения *(Exception):*

Visual Basic .NET

```
Catch ex As Exception
    litError.Text = "<p>The following error occurred:</p>" & _
      ex.Message & "<br>"
Catch ex As System.IO.IOExcecption
    litError.Text = "<p>The file could not be saved " & _
      "to the server.</p>"
Catch ex As System.IO.FileNotFoundException
    litError.Text - "<p>The file does not exist.</p>"
Catch ex As System.Net.WebException
    litError.Text = "<p>An Internet error occurred while  & _
      "uploading the file.</p>"
```

Visual C#

```
eaten (Exception ex)
{
    litError.Text = "<p>The following error occurred:</p>" +
      ex.Message + "<br>";
}
catch (System.IO.IOException ex)
{
    litError.Text = "<p>The file could not be saved " +
      "to the server.</p>";
}
catch (System.IO.FileNotFoundException ex)
{
    litError.Text = "<p>The file does not exist.</p>";
!
```

```
catch (System.Net.WebException ex)
{
    litError.Text = "<p>An Internet error occurred while" +
        "uploading the file.</p>";
}
```

В .NET Framework у каждого класса имеется собственный тип исключений, поэтому в документации по .NET вы не найдете единого списка типов исключений. Для просмотра отдельных типов исключений в среде Visual Studio используется диалоговое окно .NET Exceptions.

Для просмотра типов исключений.NET Framework выполните следующие действия.

1. Выберите в меню Debug элемент Exceptions — Visual Studio .NET откроет диалоговое окно Exceptions, показанное на рис. 6-3.

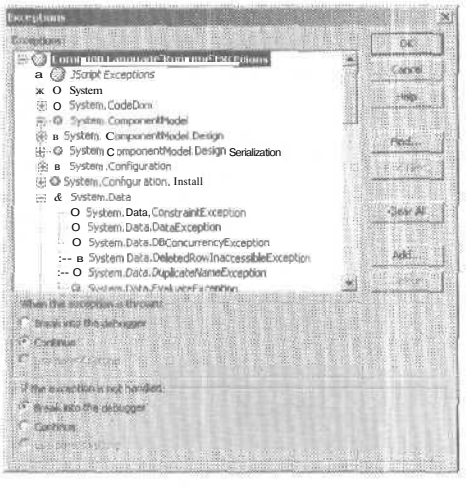


Рис. 6-3. Диалоговое окно Exceptions

2. Щелкните значки «+» слева от элементов списка Exceptions, чтобы просмотреть списки исключений в каждом из элементов.

Генерация исключений

Как говорилось ранее, ключевое слово *Throw* (в C# — *throw)* служит для генерации различных исключений. Для демонстрации методики обработки исключений это полезно, но возникает вопрос: зачем генерировать исключения в *реальных* приложениях?

Обработка исключений — это один из способов организации логики приложения, когда вы перечисляете ситуации, которые выходят за рамки обычной работы приложения, и определяете их как «исключительные». Это позволяет вынести сложный алгоритм обработки исключений за пределы основной логики вашего приложения и, следовательно, упростить ее.

Скажем, в примере с загрузкой файла на сервер разумно генерировать исключение *FileNotFoundException*, если загружаемый файл имеет нулевой размер, в противном случае проверить перед загрузкой существование файла на клиентском компьютере намного сложнее. Для наших целей в этом случае хорошо подходит *FileNotFoundException* — стандартное исключение, определенное в пространстве имен *System.IO*.

Visual Basic .NET

```
Private Sub butUpload_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butUpload.Click
    Dim strFilename As String - filUpload.PostedFile.Filename
    ' Разбить заданный путь на части.
    strFilename = System.IO.Path.GetFilename(strFilename)
    Try
        ' Проверить, равен ли размер файла нулю (существует ли файл).
        If filUpload.PostedFile.ContentLength = 0 Then
            Throw New System.IO.FileNotFoundException()
        End If
        ' Сохранить закачиваемый файл на сервере с помощью полученного имени.
        filUpload.PostedFile.SaveAs _
          (Request.MapPath(Request.ApplicationPath) & "\" & _
          strFilename)
        ' Добавить файл к списку серверных файлов.
        lstFiles.Items.Add(strFilenarne)
        ' Выделить первый элемент в списке.
        lstFiles.SelectedIndex = 0
        ' Обработать возможные исключения.
   Catch ex As System.IO.FileNotFoundException
      litError.Text = "<p>The file does not exist.</p>"
   Catch ex As System.IO.IOExcecption
      litError.Text - "<p>The file could not be saved " & _
     "to the server.</p>"
   Catch ex As System.Net.WebException
      litError.Text = "<p>An Internet error occurred while " & _
     "uploading the file.</p>"
   Catch ex As Exception
      litError.Text - "<p>The following error occurred:</p>" & _
     ex.Message & "<br>"
     ' Освободить ресурсы.
    Finally
        ' Сбросить значение поля для ввода имени файла.
        filUpload.Value = ""
    End Try
End Sub
```

Visual C#

```
private void butUpload_Click(object sender, System.EventArgs e)
{
    string strFilename - filUpload.PostedFile.FileName;
    // Разбить заданный путь на части.
    strFilenarne = System.IO.Path.GetFileName(strFilename);
    try
    {
```

```
        // (1) Проверить, равен ли размер файла нулю (существует ли файл).
        if (filUpload.PostedFile.ContentLength == 0)
            throw new System.IO.FileNotFoundException();
        // (2) Сохранить закачиваемый файл на сервере с помощью
        // полученного имени.
        filUpload.PostedFile.SaveAs
          (Request.MapPath(Request.ApplicationPath) + "\\" +
          strFilename);
        // Добавить файл к списку серверных файлов.
        lstFiles.Items.Add(strFilename);
        // Выделить первый элемент в списке.
        lstFiles.SelectedIndex = 0;
    }
    // (3) Обработать возможные исключения.
    catch (System.IO.FileNotFoundException ex)
    {
        litError.Text = "<p>The file does not exist.</p>";
    }
    catch (System.IO.IOException ex)
    {
        litError.Text = "<p>The file could not be saved " +
          "to the server.</p>";
    }
    catch (System.Net.WebException ex)
    {
        litError.Text = "<p>An Internet error occurred while" +
          "uploading the file.</p>";
    }
    catch (Exception ex)
    {
        litError.Text = "<p>The following error occurred:</p>" +
          ex.Message + "<br>";
    }
    // (4) Освободить ресурсы.
    finally
    {
        // Сбросить значение поля для ввода имени файла.
        filUpload.Value = "";
    }
}
```

Определение новых исключений

Иногда исключения, генерируемые приложениями, не имеют среди типов .NET Framework эквивалентов, достаточно эффективных и способных описать исключительную ситуацию. В таких случаях приходится создавать новые исключения на основе типа *ApplicationException*. Например, следующий фрагмент кода генерирует исключение, свидетельствующее о повторной попытке входа пользователя, уже вошедшего на сервер:

Visual Basic .NET

```
Throw New ApplicationException("User is already logged on.")
```

Visual C#

```
throw new ApplicationException("User is already logged on.");
```

Для обработки подобных исключений также применяются операторы *Catch (catch* в C#):

Visual Basic .NET

```
Catch ex As ApplicationException
```

Visual C#

```
catch (ApplicationException ex)
```

Класс *ApplicationException* не дает дополнительных возможностей по сравнению со стандартным классом *Exception*, он просто позволяет различать исключения, определенные в .NET Framework, и исключения, определенные в приложениях.

Если вы пишете большое приложение или создаете компоненты, предназначенные для использования другими приложениями, скорее всего вам понадобится объявить собственные классы исключений на основе класса *ApplicationException*. Например, следующий фрагмент кода объявляет класс исключения *UserLoggedOnException*:

Visual Basic .NET

```
Public Class UserLoggedOnException
   Inherits ApplicationException

   ' Конструктор исключения,
   Sub New(Optional ByVal Message As String = "The user is already " & _
     "logged on to the server.", _
     Optional ByVal Inner As Exception = Nothing)
         MyBase.New(Message, Inner)
   End Sub

End Class
```

Visual C#

```
public class UserLoggedOnException : System.ApplicationException
{
    // Конструктор исключения (перегруженный).
    public UserLoggedOnException() :
  this("The user is already logged on to the server", null)
    {
    }

    public UserLoggedOnException(string message) : this(message, null)
    {
    }
```

```
    public UserLoggedOnException(string message, Exception inner) :
  base(message, inner)
    {
    }

}
```

Все свойства и методы кроме конструктора, устанавливающего сообщение по умолчанию, класс *UserLoggedOnException* наследует от базового класса *ApplicationException* — это стандартный прием программирования.

Использование событий Error

Другой способ обработки исключений основан на использовании встроенных событий *Error* Web-объектов. При возникновении в Web-приложении необработанного исключения ASP.NET генерирует события error, описанные в таблице 6-2.

Таблица 6-2. **События *Error***

Имя обработчика	**Обстоятельства** возникновения **события**
Page_Error	При возникновении на странице необработанного исключения; обработчик этого события находится в Web-форме
Global_Error	При возникновении необработанного исключения в приложении; обработчик этого события находится в файле Global.asax
Application_Error	При возникновении необработанного исключения в приложении; обработчик этого события находится в файле Global.asax

Обработчики событий *Error* позволяют централизованно обрабатывать все исключения, генерируемые объектом. События *Error* можно использовать по-разному:

- **вместо блоков обработки исключении.** Поскольку события *Error* генерируются за пределами области видимости процедуры, в которой произошла ошибка, они дают меньше информации о последовательности действий, вызвавших исключение, и значит, меньше возможностей для исправления исключительной ситуации. Однако это не имеет значения, когда ошибку нельзя исправить программно;
- **вместе с блоками обработки исключений.** События *Error* позволяют организовать централизованную резервную «линию зашиты» на случай возникновения непредвиденных исключений или исключений, обрабатываемых в другом месте. Комбинируя эти методы обработки исключений, удается перехватывать любые исключения до того, как пользователь увидит их последствия. После этого можно отобразить соответствующее сообщение или даже записать сведения об исключении в журнал — эти данные пригодятся во время работы по улучшению приложения.

Таблица 6-3. Методы объекта Server, применяемые при обработке исключений **с помощью событий**

Имя метода	**Назначение**
GetLastError()	Получение последнего исключения, возникшего на сервере
ClearError()	Сброс последнего исключения, возникшего на сервере. После вызова *ClearError* исключение считается обработанным, поэтому другие события error не срабатывают и пользователь не видит их последствий в браузере

При обработке исключений с помощью событий error для получения сведений об исключениях применяется объект *Server*. Этот объект поддерживает методы для работы с исключениями, описанные в таблице 6-3.

Чтобы обработать исключение с помощью события error, выполните следующие действия.

1. Вызовите метод *GetLastError* в обработчике события *Page_Error*, *Global_Error* или *Application_Error*, чтобы получить возникшее исключение.
2. Обработайте исключение: покажите сообщение для пользователя, исправьте ошибку или сделайте запись в журнале ошибок.
3. Выполните сброс исключения с помощью метода *ClearError*.
4. Отобразите страницу заново. Это необходимо, так как при возникновении исключения обработка Web-формы немедленно прекращается, поэтому после сброса исключения серверные элементы управления и другие компоненты страницы могут остаться непрорисованными.

Следующий фрагмент кода иллюстрирует эти действия:

Visual Basic .NET

```
Private Sub Page_Error(ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles MyBase.Error
      ' Получить ошибку.
      Dim ex As Exception = Server.GetLastError()
      ' Записать сообщение об ошибке,
      Session("Error") - ex.Message()
      ' Сбросить ошибку.
      Server.ClearError()
      'Перезагрузитьстраницу.
      Response.Redirect("ErrorEvents.aspx")
End Sub
```

Visual C#

```
private void Page_Error(object sender, System.EventArgs e)
{
    // Получить ошибку.
    Exception ex - Server.GetLastError();
    // Записать сообщение об ошибке,
    Session["Error"] - ex.Message;
    // Сбросить ошибку.
    Server.ClearError();
    ,// Перезагрузить страницу.
    Response.Redirect("ErrorEvent.aspx");
}
```

Перед сбросом исключения предыдущий код записывает текст сообщения об ошибке в переменную состояния *Session*, чтобы показать его после перезагрузки страницы, осуществляемой вызовом метода *Redirect*. Следующий фрагмент кода отображает сохраненное сообщение об ошибке после перезагрузки страницы:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles MyBase.Load
      ' Если сообщение об ошибке есть, отобразить его.
     If Session("Error") <> Nothing Then
         litError.Text = "<p>The following error occurred:</p>" & _
           Session("Error") & "<br>"
         ' Сбросить переменную состояния Session.
         Session("Error") = Nothing
     End If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Если сообщение об ошибке есть, отобразить его.
    if(Session["Error"] != null)
    {
        litError.Text = "<p>The following error occurred:</p>" +
          Session["Error"].ToString();
        // Сбросить переменную состояния Session.
        Session["Error"] = null;
    }
}
```

Занятие 2. Страницы сообщений об ошибках

Изучив материалы занятия 1, вы освоили различные способы обработки исключений, возникающих на сервере, на котором работает Web-приложение, Но, поскольку во время работы Web-приложение обменивается данными через Интернет, целый класс исключений нельзя обнаружить в программе. Для перехвата и максимально эффективной обработки таких исключений предназначены страницы сообщений об ошибках.

В этом занятии вы научитесь применять страницы сообщений об ошибках ASP.NET для перехвата и обработки ошибок HTTP на уровне приложения и страницы.

Изучив материал этого занятия, вы сможете:

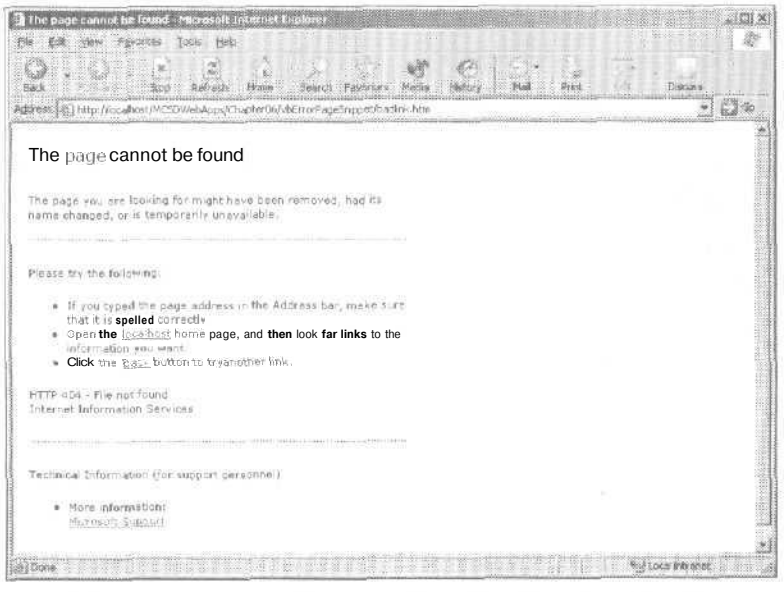
- ✓ определять типы исключений, которые нельзя обнаружить в программе;
- ✓ реагировать на исключения HTTP на уровне приложения и страницы;
- ✓ задавать страницы для отображения при возникновении определенных ошибок HTTP при помощи файла Web.config.

Продолжительность занятия — около 15 минут.

Обработка исключений, связанных с работой Интернета

При исполнении Web-приложения в браузере пользователя ряд исключений может возникнуть за пределами области видимости приложения. Например, ошибки на сервере из-за тайм-аута при ожидании отклика пользователя или отсутствия запрошенной пользователем страницы.

При возникновении подобных событий ASP.NET отображает страницу с сообщением об ошибке, похожую на ту, что показана на рис. 6-4.



Рис, 6-4. Страница с сообщением об ошибке «Page Not Found»

Страницы сообщений об ошибках (error pages) — это серверные .htm- или .aspx-страницы, на которые направляется пользователь при возникновении необработанных исключений. ASP.NET позволяет задавать страницы сообщений об ошибке на двух уровнях:

- **для всего приложения — в** разделе customErrors файла Web.config; параметры, задаваемые в этом разделе, определяют вид страницы, отображаемой при возникновении определенных ошибок HTTP;
- **для отдельной Web-формы** — атрибутом *Error Page* страничной директивы Page; значение этого атрибута определяет страницу с сообщением об ошибке, которая должна отобраться при возникновении необработанного исключения во время обработки Web-формы.

В следующих разделах использование страниц сообщений об ошибках на разных уровнях рассматривается более подробно.

Использование страниц сообщений об ошибках на уровне приложения

Страницы, отображаемые по умолчанию при возникновении определенных ошибок, определяются Internet Information Service **(IIS).** Чтобы просмотреть или изменить соответствующие параметры IIS для приложения, выполните следующие действия.

1. Щелкните правой кнопкой папку Web-приложения и выберите из контекстного меню команду Properties — IIS откроет диалоговое окно свойств приложения.
2. В диалоговом окне Properties перейдите на вкладку Custom Errors. IIS откроет список параметров, определяющих страницы, отображаемые при получении различных кодов состояния HTTP (рис. 6-5).

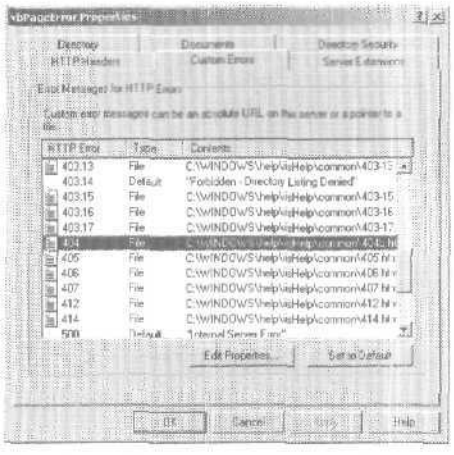


Рис. 6-5. Параметры вкладки Custom Errors в IIS

3. Выберите в списке Error Messages For HTTP Errors код состояния, для которого нужно задать другую страницу сообщения об ошибке, и щелкните кнопку Edit Properties - IIS откроет диалоговое окно Error Mapping Properties, показанное на рис. 6-6.
4. Щелкните Browse, чтобы заменить страницу сообщения об ошибке, заданную по умолчанию, страницей из папки вашего приложения. Закончив, щелкните OK.

В результате назначения других страниц сообщений об ошибках на уровне приложения при помощи IIS изменяются параметры сервера, на котором работает приложение. После переноса приложения на другой сервер с IIS, на новом сервере придется заново

назначать все нестандартные страницы. Другой вариант — включить сведения о нестандартных страницах сообщений об ошибках, назначенных на уровне приложения, и само приложение. Это делается при помощи файла Web.config из проекта.

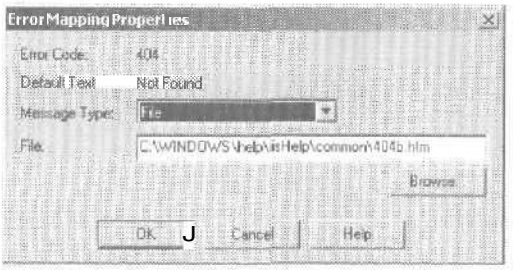


Рис. 6-6. Диалоговое окно Error Mapping Properties в IIS

Страницы, которые должны отображаться при возникновении во время работы Web-приложения определенных необработанных ошибок HTTP, задают в разделе custom Errors файла Web.config. Ошибки HTTP идентифицируются кодами состояния, определенными в спецификации HTTP 1.1. В ASP.NET эти коды находятся в перечислении HTTPStatus-Code. Коды наиболее распространенных состояний перечислены в таблице 6-4.

Таблица 6-4. Коды наиболее распространенных состояний HTTP

Код	Член HTTPStatusCode	Значение
200	*OR*	Запрос выполнен успешно
204	*NoContent*	Запрос выполнен успешно, но данные были преднамеренно удалены из отклика
301	*Moved, MovedPermanently*	Запрос перенаправляется на другой адрес
302	*Found, Redirect*	Запрос перенаправляется на другой адрес
400	*BadRequest*	Запрос не распознан сервером
401	*Unauthorized*	С запросом не предоставлены свидетельства о наличии у пользователя прав на доступ к запрашиваемому им ресурсу
403	*Forbidden*	Сервер отказывается исполнить запрос
404	*NotFound*	Запрашиваемого ресурса нет на сервере
408	*RequestTimeOut*	На сервере наступил тайм-аут до получения запроса от клиента
500	*InternalServerError*	Произошла внутренняя ошибка сервера
503	*Service Unavailable*	Сервер временно недоступен
505	*HttpVersionNotSupported*	Запрашиваемая версия **HTTP** не поддерживается сервером

Чтобы назначить страницы, отображаемые в ответ на получение различных кодов состояния, поместите в раздел *customErrors* файла Web.config соответствующие тэги *<error>*. Ниже показан пример раздела *customErrors,* в котором задана страница сообщения об ошибке. отображаемая по умолчанию, а также определены нестандартные страницы сообщений для некоторых кодов ошибки HTTP:

```
<customErrors mode="On" defaultRedirect="ErrDefault.aspx">
    <error statusCode="401" redirect="ErrUnauthorized.aspx" />
```

```
        <error statusCode="404" redirect="ErrPageNotFound.aspx" />
        <error statusCode="500" redirect="ErrServer.htm" />
    </customErrors>
```

Для просмотра страниц сообщений об ошибках во время отладки приложения на локальном компьютере поле *mode* атрибута *custom Errors* должно быть установлено в *On*. Если оно установлено в *RemoteOnly* (это значение задано по умолчанию), назначенные страницы сообщений об ошибках будут отображаться только при доступе к приложению с других клиентских компьютеров, но не с локального.

Использование страниц сообщений об ошибках на уровне страницы

Для отображения определенных страниц при возникновении необработанного исключения во время обработки Web-формы применяется атрибут *ErrorPage*объекта *Page*. На страницах сообщений об ошибках также можно использовать методы *GetLastError*и *ClearError* объекта *Server* (это показано выше при обсуждении обработки исключений с помощью событий *Error)*.

Например, следующий HTML-код устанавливает атрибут *ErrorPage* для Web-формы:

Visual Basic .NET

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
Inherits="vbExceptionSnippet.WebForm1" errorPage="errDefault.aspx"%>
```

Visual C#

```
<%@ Page language="c#" Codebehind="ErrorPage.aspx.cs"
    AutoEventWireup="false" Inherits="csErrorPageSnippet.WebForm1"
    errorPage="ErrDefault.aspx"%>
```

В этом случае щелчок кнопки butError генерирует исключение:

Visual Basic .NET

```
' Webform1.aspx
Private Sub butError_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butError.Click
      ' Генерировать исключение.
      Throw New System.Net.WebException()
End Sub
```

Visual C#

```
// Webform1.aspx
private void butError_Click(object sender. System.EventArgs e)
{
   // Генерировать исключение.
   throw new System.Net.WebException();
}
```

В результате пользователь видит страницу ErrDefault.aspx с описанием ошибки:

Visual Basic .NET

```
' ErrDefault.aspx
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Показать сведения об ошибке.
    litError.Text - "<p>The following error occurred:</p>" & _
      Server.GetLastError.Message
    ' Сбросить ошибку.
    Server.ClearError()
End Sub
```

Visual C#

```
// ErrDefault.aspx
 private void Page_Load(object sender, System.EventArgs e)
 {
     // Показать сведения об ошибке.
     litError.Text = "<p>The following error occurred:</p>" +
       Server.GetLastError().Message;
     // Сбросить ошибку.
     Server.ClearError();
 }
```

Занятие 3. Регистрация исключений

Во время разработки полезно следить за исключениями, регистрируя их в журнале. *Журнал исключений* (exception log) — это список обработанных исключений, возникших при исполнении приложения. Периодический просмотр журнала исключений позволяет удостовериться, что исключения обрабатываются корректно, не возникают слишком часто и не мешают пользователям приложения решать свои задачи.

В этом занятии вы научитесь пользоваться трассировкой ASP.NET для регистрации исключений, возникающих во время работы Web-приложения.

Изучив материал этого занятия, вы сможете:

- ✓ указать преимущества трассировки как способа регистрации исключений:
- ✓ включать и выключать трассировку приложения или отдельной Web-формы;
- ✓ записывать сообщения в журнал трассировки;
- ✓ просматривать журнал трассировки;
- ✓ очищать журнал трассировки и получать новые трассировочные данные.

Продолжительность занятия — около 20 минут.

Регистрация ошибок с помощью трассировки

Трассировка (tracing) — это способ регистрации событий, возникающих во время исполнения приложений, например исключений. Есть и другие способы регистрации ошибок в приложениях. Обычно для этого открывают файл, куда записывают сообщения об ошибках, но трассировка предпочтительнее по следующим причинам:

- **из-за стандартизации.** Встраивание механизмов трассировки в .NET Framework позволяет применять унифицированные программные методы при разработке любых приложений, использующих .NET Framework;
- **из-за встроенной поддержки Web.** ASP.NET дополняет инструментарий .NET Framework средствами для получения информации о производительности и поведении Web-запросов;
- **из-за простоты настройки.** Трассировку можно включать и выключать, задавая соответствующие параметры в конфигурационном файле приложения, при этом удается обойтись без перекомпиляции приложения;
- **из-за производительности,** В отключенном состоянии трассировочные операторы не влияют на производительность приложения.

Чтобы использовать трассировку в Web-приложении, выполните следующие действия.

1. Включите трассировку.
2. Занесите записи в журнал трассировки.
3. Прочитайте журнал трассировки.

Подробно эти действия рассматриваются в следующих разделах.

Включение и выключение трассировки

Можно включать и выключать трассировку как для всего Web-приложения, так и для отдельной страницы.

- Чтобы включить трассировку приложения, откройте его файл Web.config и установите атрибут *Enabled* элемента *<trace>* в *True*.
- Чтобы включить трассировку отдельной страницы, установите свойство *Trace* объекта DOCUMENT в *True*.

По умолчанию результаты трассировки отображаются в нижней части Web-страниц (рис. 6-7).

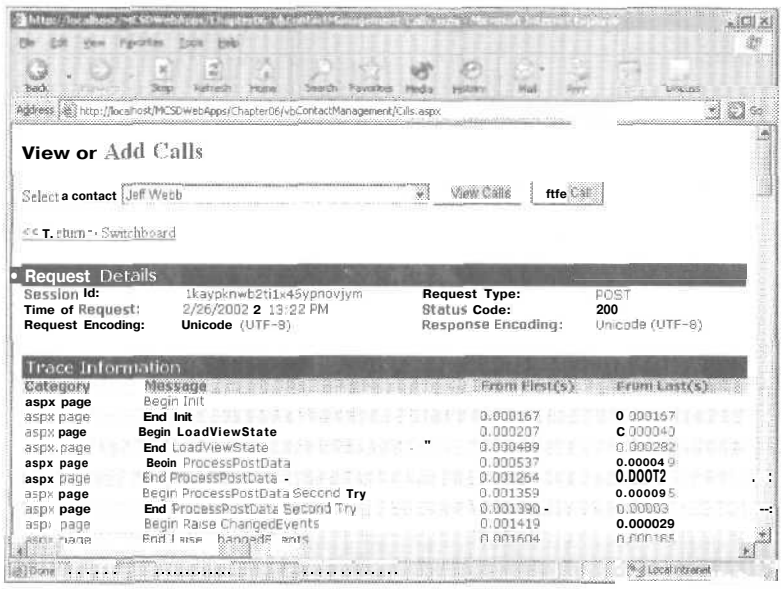


Рис. 6-7. **Отображение** результатов **трассировки**

Этот вариант отображения трассировочных данных прекрасно подходит для отладки. Однако при тестировании готового приложения трассировочные данные, как правило, требуется записывать в файл журнала. Чтобы записывать в файл журнала трассировочные сообщения для всего приложения, откройте файл Web.config и установите атрибут *PageOutput* элемента *<trace>* в *False.* При этом ASP.NET будет записывать трассировочные данные в файл Trace.axd, расположенный в корневом каталоге приложения.

У элемента *<trace>* также имеется атрибут *RequestLimit*, определяющий максимальное число запросов страниц, которые могут быть зарегистрированы в журнал трассировки. Например, следующая строка из Web.config включают трассировку приложения и записывает в файл Trace.axd сведения о первых 20 запросах:

```
<trace enabled="true" requestLimit="20" pageOutput="false"
traceMode="SortByTime" localOnly="true" />
```

Запись трассировочных сообщений в журнал осуществляется независимо от трассировки страниц. Если свойство *Trace* объекта DOCUMENT установлено в *True,* все трассировочные сообщения для этой страницы будут отображаться на самой странице.

Чтобы в файл журнала записывались только трассировочные сообщения для определенных страниц приложения, выполните следующие действия.

1. Откройте файл Web.config и установите атрибут *Enabled* элемента *<trace>* в *True.* а атрибут *PageOutput* — в *False.*
2. У всех Web-страниц, для которых нужно отключить трассировку, установите свойство *Trace* объекта DOCUMENT в *False.*

После изменения значения свойства *Trace* объекта DOCUMENT невозможно восстановить параметры по умолчанию в окне свойств страницы в Visual Studio .NET. Для этого придется отредактировать HTML-код Web-формы, удалив поле *trace* из элемента *Page.* В следующем примере HTML подлежащий удалению текст выделен полужирным шрифтом:

Visual Basic .NET

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="Trace.aspx.vb"
    Inherits="vbErrorLogSnippet.WebForm1" trace="True" %>
```

Visual C#

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs" AutoEventWireup="false"
    Inherits="csTracing.WebForm1" trace="True"%>
```

Запись сообщений в журнал трассировки

Объект *Trace* поддерживает методы *Write* и *Warn,* позволяющие записывать сообщения в трассировочные данные запроса. Эти методы похожи во всем, кроме цвета текста сообщений: у метод *Write* он черный, а у метода *Warn* — красный.

Например, следующий фрагмент кода записывает в журнал трассировки сведения о необработанных исключениях при помощи метода *Warn,* поэтому текст сообщений будет красным.

Visual Basic .NET

```
Private Sub Page_Error(ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles MyBase.Error
    ' Записать сообщение в журнал трассировки,
    Trace.Warn("Error", "", Server.GetLastError())
    ' Сбросить ошибку для того, чтобы продолжить исполнение приложения.
    Server.ClearError()
    ' Отобразить страницу еще раз.
    Response.Redirect("Trace.aspx")
End Sub
```

Visual C#

```
private void Page_Error(object sender, System.EventArgs e)
{
    // Записать сообщение в журнал трассировки.
    Trace.Warn("Error", "", Server.GetLastError());
    // Сбросить ошибку для того, чтобы продолжить исполнение приложения,
    Server.ClearError();
    // Отобразить страницу еще раз.
    Response.Redirect("Trace.aspx");
}
```

Следующий код генерирует необработанное исключение в ответ на щелчок кнопки butError:

Visual Basic .NET

```
Private Sub butError_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butError.Click
    Throw New System.IO.FileNotFoundException()
End Sub
```

Visual C#

```
private void butError_Click(object sender, System.EventArgs e)
{
    throw new System.IO.FileNotFoundException();
}
```

В некоторых случаях во время трассировки необходимо выполнять дополнительные действия, например получать сведения о пользователе, чтобы добавить их к трассировочному сообщению. Свойство *IsEnabled* объекта *Trace* позволяет исполнять подобный код, только когда трассировка включена, поэтому он не снижает производительность приложения при отключенной трассировке. Например, следующий код добавляет к трассировочному сообщению сведения о браузере лишь при включенной трассировке:

Visual Basic .NET

```
Private Sub Page_Error(ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles MyBase.Error
    If Trace.IsEnabled Then
        Dim strMessage As String
        If Request.Browser.AOL Then
            strMessage - "AOL Browser"
        Else
            strMessage = "Non-AOL Browser"
        End If
        Trace.Warn("Error", strMessage, Server.GetLastError())
    End If
    Server.ClearError()
    Response.Redirect("Trace.aspx")
End Sub
```

Visual C#

```
private void Page_Error(object sender, System.EventArgs e)
{
    if (Trace.IsEnabled)
    {
        string strMessage;
        if (Request.Browser.AOL)
            strMessage - "AOL Browser";
        else
            strMessage = "Non-AOL Browser";
        // Записать сообщение в журнал трассировки.
        Trace.Warn("Error", strMessage, Server.GetLastError());
    }
    // Сбросить ошибку для того, чтобы продолжить исполнение приложения,
    Server.ClearError();
    // Отобразить страницу еще раз.
    Response.Redirect("Trace.aspx");
}
```

Чтение журнала трассировки

По умолчанию результаты трассировки отображаются в нижней части всех Web-страниц, для которых включена трассировка. Как говорилось ранее, если установить в *False* атрибут *PageOutput* элемента *<trace>* из файла Web.config, результаты трассировки Web-страницы будут записываться в файл Trace.axd, расположенный в корневом каталоге приложения. Для просмотра этого файла просто наберите его URL в браузере или воспользуйтесь следующей гиперссылкой:

```
<A href="Trace.axd">View trace log.</A>
```

По умолчанию Trace.axd можно просматривать только с локального компьютера, то есть с сервера, на котором работает приложение. Чтобы получить возможность просмотра журнала трассировки с удаленного компьютера (например, при удаленной отладке), откройте файл Web.config и установите атрибут *LocalOnly* элемента *<trace>* в *False:*

```
<trace enabled="true" requestLimit="20" pageOutput="false"
    traceMode="SortByTime" localOnly="false" />
```

Открытый в браузере журнал трассировки (Trace.axd) выглядит, как на рис. 6-8.

ASP.NET останавливает трассировку, когда число полученных сервером HTTP-запросов достигает предела, заданного атрибутом *RequestLimit* элемента *<trace>*. Для сброса трассировочных данных и повторного запуска трассировки щелкните ссылку Clear Current Trace в правом верхнем углу страницы Trace.axd. Эта ссылка заново отображает Trace.axd, передавая строку запроса "clear=1". Вот HTML этой ссылки:

```
<a href="Trace.axd?clear=1">Clear trace log.</a>
```

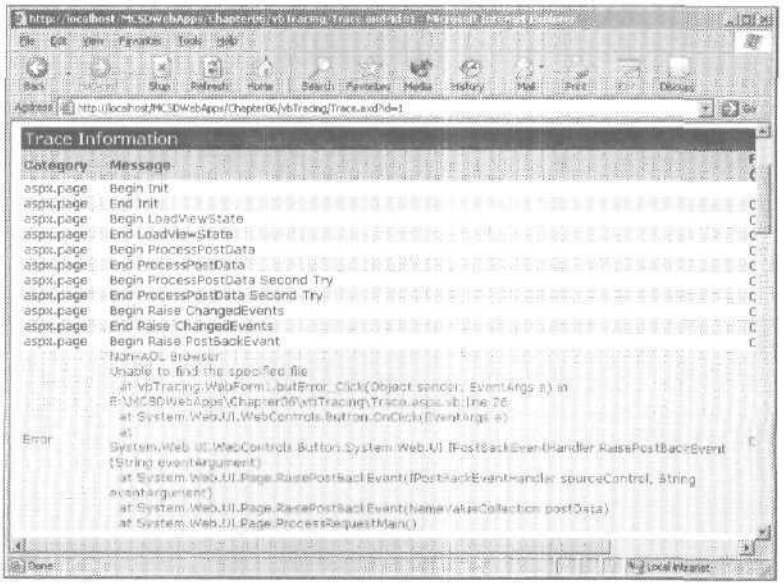


Рис. 6-8. Файл журнала Trace.axd

Резюме

- Обработка исключений позволяет разрешать ситуации, не предвиденные программой. Независимо от используемого подхода, обработка исключений позволяет вынести код для обработки исключений за пределы основной логики приложения, и следовательно, упростить ее.
- Для обработки исключений внутри процедур используются блоки, образованные ключевыми словами *Try, Catch, Finally* (в Visual Basic .NET) или *try, catch, finally* (в Visual C#).
- Следует перехватывать исключения в порядке от специфичных типов к общим.
- Для генерации исключений служат ключевое слово *Throw* (в Visual Basic .NET, в Visual C# — *throw).*
- События error служат для обработки исключений при помощи обработчиков определенных событий, вызывающих методы *GetLastError* и *ClearError* объекта *Server.*
- Страницы сообщений об ошибках применяются для обработки исключений, возникающих за пределами Web-приложения, например из-за сбоев в работе Интернета.
- Следует регистрировать обработанные исключения, возникающие во время работы приложения, записывая в журнал трассировки сообщения с помощью методов *Write* или *Warn* объекта *Trace.*
- Чтобы включить трассировку на уровне приложения, необходимо соответствующим образом установить атрибуты элемента *<trace>* в файле Web.config.
- Чтобы включить трассировку на уровне страницы, необходимо соответствующим образом установить атрибут *Trace* страничной директивы Page.

Практикум 6. Организация обработки исключений в приложении

В этом практикуме вы добавите код для обработки исключений к диспетчеру контактов, созданному в практикуме предыдущей главы. Сейчас в этом приложении для устранения возможных ошибок при обновлении записей в БД применяются простые блоки обработки исключений. Ваша задача — дополнить их кодом, который позволит перехватывать и обрабатывать определенные типы ошибок, связанные с данными, а также реализовать возможность трассировки для наблюдения за обработкой этих ошибок.

Продолжительность практикума — около 30 минут.

Упражнение 1. Усовершенствование обработки исключений



Диспетчер контактов обрабатывает исключения, возникающие при работе с БД, очень просто: отображает информацию об ошибке и позволяет пользователю отменить операцию. Приведенный ниже пример кода демонстрирует реализацию этого подхода в обработчике события *butAdd_Click* Web-формы AddContact:

Visual Basic .NET

```
Try
' Модифицировать БД.
    adptContacts.Update(dsContacts)
    ' Уведомить об успешном завершении операции.
    litStatus.Text = rowNew.FirstName & "  " & rowNew.LastName & _
    " added successfully.<br>"
Catch ex As Exception
    LitStatus.Text = "The following error occurred:<br>" & _
    ex.Message & "<br>" & _
    "Correct the error and click Add to add the contact " & _
    "or click Cancel to abort.<br>"
End Try
```

Visual C#

```
try
{
    // Модифицировать БД.
    adptContacts.Update(dsContacts);
    // Уведомить об успешном завершении операции.
    litStatus.Text = rowNew.FirstName + "  " + rowNew.LastName +
      " added successfully.<br>";
    // Включить и выключить ViewState, чтобы очистить скрытые страничные поля,
    Page.EnableViewState = false;
    Page.EnableViewState = true;
}
catch (Exception ex)
{
```

```
    litStatus.Text - "The following error occurred:<br>" +
    ex.Message + "<br>" +
    "Correct the error and click Add to add the contact " +
      "or click Cancel to abort.<br>";
}
```

Чтобы приложение смогло выборочно обрабатывать исключения, возникающие при работе с данными, модифицируйте операторы *Catch (catch в* C#), добавив к ним код для обработки следующих исключений.

- **ConstraintException** — возникает при нарушении ограничения«уникальный ключ», заданного для поля набора данных. На практике это происходит, когда два пользователя почти одновременно пытаются добавить контактную информацию и получают одинаковое значение ContactID. Если пользователь, добавивший свою информацию чуть позже, попытается записать ее в БД, возникнет исключение *ConstraintException,* так как в базе уже есть строка с таким ContactID, добавленная первым пользователем. Чтобы разрешить эту ситуацию, нужно получить другой ContactID и снова попробовать записать информацию в БД, например, так:

Visual Basic .NET

```
' Это исключение возникает при нарушении уникальности ContactID.
Catch ex As ConstraintException
      ' Получить новый идентификатор.
   rowNew.ContactID = GetNewContactID()
' Попытаться еще раз.
Try
      adptContacts.Update(dsContacts)
   ' Уведомить об успешном завершении операции.
   litStatus.Text = rowNew.FirstName & " " & rowNew.LastName & _
     " added successfully.<br>"
Catch
   litStatus.Text - "The record could not be added. " & _
     "Click Add to try again or click Cancel to abort."
End Try
```

Visual C#

```
// Это исключение возникает при нарушении уникальности ContactID.
catch (ConstraintException ex)
{
    // Получить новый идентификатор,
    rowNew.ContactID - GetNewContactIDO;
    // Попытаться еще раз,
    try
    {
        adptContacts.Update(dsContacts);
        // Уведомить об успешном завершении операции.
        litStatus.Text = rowNew.FirstName + " " + rowNew.LastName +
          " added successfully.<br>";
    }
```

```
    catch
    {
        litStatus.Text - "The record could not be added. " +
        "Click Add to try again or click Cancel to abort.";
    }
}
```

- **DBConcurrencyException** — возникает, когда пользователь пытается модифицировать БД в то время, когда другой пользователь обладает монопольным доступом к ней. Лучшее решение в этом случае — уведомить первого пользователя о неудачной записи его данных в БД и дать ему еще одну попытку:

Visual Basic .NET

```
' БД может быть заблокирована другим сеансом,
' поэтому дать пользователю еще одну попытку.
Catch ex As DBConcurrencyException
    litStatus.Text - "The database is currently locked. Wait " & _
      "a few seconds and then click Arid again."
```

Visual C#

```
// БД может быть заблокирована другим сеансом,
// поэтому дать пользователю еще одну попытку.
catch (DBConcurrencyException ex)
{
    litStatus.Text = "The database is currently locked. Wait " +
      "a few seconds and then click Add again.";
}
```

- **DataException** — это общее исключение, возникающее при работе с данными. Добавив код для обработки этого исключения, вы позаботитесь о разрешении непредвиденных проблем. Следующий фрагмент уведомляет пользователя об ошибке и записывает сведения об исключении в журнал трассировки для просмотра и диагностики его причин:

Visual Basic .NET

```
' Возникло общее исключение при работе с данными,
' его нужно зарегистрировать.
Catch ex As DataException
    litStatus.Text = "The following database error occurred:<br>" & _
      ex.Message & "<br>" & _
      "Correct the error and click Add to add the contact " & _
      "or click Cancel to abort.<br>"
    Trace.Warn("Error", "Data exception", ex)
End Try
```

Visual C#

```
// Возникло общее исключение при работе с данными,
// его нужно зарегистрировать.
```

```
 catch(DataExceptionex)
{
    litStatus.Text = "The following database error occurred:<br>" +
      ex.Message + "<br>" +
      "Correct the error and click Add to add the contact " +
      "or click Cancel to abort.<br>";
Trace.Warn("Error", "Data exception", ex);
!
```

После внесения указанных изменений в обработчик события *butAdd_Click* любые исключения, не имеющие отношения к данным, останутся необработанными. Чтобы решить задачу практикума, в следующем упражнении вы добавите код для обработки возможных исключений при помощи событий *Error*.

Упражнение 2. Добавление обработчика события Error



Итак, вы организовали обработку любых исключений, связанных с данными, при помощи обработчика события *butAdd_Click* Web-формы Contacts; теперь необходимо организовать обработку и регистрацию непредвиденных исключений при помощи обработчика события *Error* этой Web-формы. Это позволит:

- скрыть от пользователя технические сообщения об ошибках и заменить их более полезной информацией;
- записать сведения о непредвиденных ошибках в журнал трассировки. Данные журнала можно использовать как диагностическое средство, чтобы определить, нужно ли обрабатывать другие исключения,

▶ Обработка непредвиденных исключений с помощью события *Error* формы **AddContact**

1. Добавьте к коду Web-формы обработчик события *Page_Error*, обрабатывающий остальные исключения, которые возникают на странице. Следующий фрагмент кода сохраняет информацию об исключении в журнале трассировки, создает сообщение об ошибке для отображения и выполняет сброс ошибки, чтобы продолжить исполнение приложения:

Visual Basic .NET

```
Private Sub Page_Error(ByVal sender As Object, ByVal e As _
  System.EventArgs) Handles MyBase.Error
    ' Записать сведения об ошибке в журнал трассировки.
    Trace.Warn("Error", "Error event", Server.GetLastError())
    ' Сохранить текст сообщения об ошибке в переменной состояния
    ' Session.
    Session("Error") - "<p>An unexpected error occurred. " & _
        "Try your task again.</p>"
    ' Сбросить ошибку,
    Server.ClearError()
    ' Перезагрузить страницу.
    Response.Redirect("AddContact.aspx")
End Sub
```

Visual C#

```
private void Page_Error(object sender, System.EventArgs e)
{
    // Записать сведения об ошибке в журнал трассировки.
    Trace.Warn("Error", "Error event", Server.GetLastError());
    // Сохранить текст сообщения об ошибке в переменной состояния Session,
    Session["Error"] = "<p>An unexpected error occurred. " +
      "Try your task again.</p>";
    // Сбросить ошибку.
    Server.ClearError();
    // Перезагрузить страницу.
    Response.Redirect("AddContact.aspx");
}
```

2. Модифицируйте обработчик события *Page_Load*, чтобы он отображал сообщение об ошибке, ранее сохраненное в переменной состояния *Session*. Для этого добавьте к нему код, выделенный в следующем примере полужирным шрифтом:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Получить объекты соединения и адаптеров из переменной
    ' состояния Session.
    oledbContactMgmt = Session("oledbContactMgmt")
    adptContactTypes = Session("adptContactTypes")
    adptContacts = Session("adptContacts")
    ' Заполнить набор данных ContactType.
    adptContactTypes.Fill(dsContactTypes)
    ' Заполнить набор данных Contacts.
    adptContacts.Fill(dsContacts)
    ' Связать с данными и заполнить списки drpContactTypes и drpState,
    drpContactTypes.DataBind()
    drpStates.DataBind()
    ' При наличии необработанной ошибки, уведомить о ней.
    If Session("Error") <> "" Then
        litStatus.Text = Session("Error2")
        ' Очистить сообщение об ошибке.
        Session("Error") = Nothing
    End If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Получить объекты соединения и адаптеров из переменной
    // состояния Session.
    oledbContactMgmt = (OleDbConnection)Session["oledbContactMgmt"];
```

```
        adptContactTypes = (OleDbDataAdapter)Session["adptContactTypes"];
        adptContacts = (OleDbDataAdapter)Session["adptContacts"];
        // Заполнить набор данных ContactType.
        adptContactTypes.Fill(dsContactTypes);
        // Заполнить набор данных Contacts,
        adptContacts.Fill(dsContacts);
        // Связать с данными и заполнить списки drpContactTypes и drpState.
        drpContactTypes.DataBind();
        drpStates.DataBind();
        // При наличии необработанной ошибки, уведомить о ней,
        if(Session["Error"] != null)
        {
            litStatus.Text = Session["Error"].ToString();
            // Очистить сообщение об ошибке.
            Session["Error"] = null;
        }
    }
```

Упражнение 3. Создание и просмотр журнала трассировки



Код, добавленный в упражнениях 1 и 2, записывает сообщения об ошибках в журнал трассировки с целью диагностики непредвиденных ошибок, которые возможны во время работы пользователей с приложением. Чтобы получить доступ к просмотру этих сообщений, нужно включить трассировку и добавить к приложению ссылку для отображения журнала трассировки.

► Как обеспечить просмотр журнала трассировки

1. Модифицируйте файл Web.config, чтобы включить трассировку. Элемент *<trace>* включает трассировку с записью результатов в файл Trace.axd, увеличивает число регистрируемых запросов до 20 и разрешает просматривать журнал трассировки с удаленного компьютера:

```
<trace enabled="true" requestLimit="20" pageOutput="false"
    traceMode="SortByTime" localOnly="false" />
```

2. Добавьте на Web-форму SwitchBoard ссылку на журнал трассировки. Эта ссылка видна только при включенной трассировке. Следующий HTML-код создает ссылку *в* виде серверного элемента управления Hyperlink, который можно делать видимым или невидимым во время выполнения, меняя значение элемента *<trace>*:

```
<asp:HyperLink NavigateUrl="Trace.axd" ID="hypTrace" runat="server"
    text="View trace log." Visible="false"></asp:HyperLink>
```

3. Добавьте к обработчику события *Page_Load* Web-формы SwitchBoard код, управляющий отображением ссылки на журнал трассировки:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Если трассировка включена, отобразить ссылку трассировки.
    If Trace.IsEnabled Then
```

```
            hypTrace.Visible = True
        End If
    End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Если трассировка включена, отобразить ссылку трассировки.
    if (Trace.IsEnabled)
        hypTrace.Visible = true;
}
```

Упражнение 4. Организация обработки исключений в других Web-формах



В предыдущих упражнениях этого практикума мы разбирали реализацию обработки исключений шаг за шагом, теперь вам пора сделать это самостоятельно. Вы добавите к Web-формам Calls, DeleteContact и ContactTypes код для обработки исключений, чтобы:

- обрабатывать любые исключения, возникающие при работе с данными, с помощью блоков обработки исключений (см. упражнение 1);
- перехватывать и регистрировать непредвиденные исключения при помощи событий *Error* (см, упражнение 2).



Закончив, сравните свои результаты с приложением Contact Management, которое находится на прилагаемом компакт-диске. Удачи!

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги,

1. Почему так важно наличие обработки исключений в готовом приложении?

2. Назовите два способа обработки исключений в Web-приложениях **ASP.NET.**

3. Расскажите, для чего нужны страницы сообщений об ошибках и как их использовать.

4. Объясните пользу трассировки при обработке исключений.

ГЛАВА 7

Усложненное программирование с применением Web Forms

Занятие 1. Сохранение и извлечение сведений о пользователе 286

Занятие 2. Отправка почты 298

Занятие 3. Применение фреймов 302

Занятие 4. Применение клиентских сценариев 308

Занятие 5. Использование Web-сервисов XML 319

В этой главе

В этой главе рассказано о том, как решать сложные задачи программирования, которые трудно отнести к какой-то определенной категории. В принципе, Web-приложения можно создавать и не зная обсуждаемые здесь предметы. Однако, освоив материал занятий из этой главы, вы научитесь решать ряд основных задач Web-программирования — без этого ваш арсенал навыков программирования останется неполным,

Прежде всего

Для изучения материалов этой главы необходимо:

- уметь создавать Web-приложения с помощью серверных элементов управления и серверного кода;
- знать языки сценариев, такие, как Visual Basic Scripting Edition (VBScript) и JScript. Если у вас нет опыта использования языков сценариев, перед занятием 4 изучи: е разделы «VBScript User's Guide» или «JScript User's Guide» справочной системы Microsoft Visual Studio.

Занятие 1. Сохранение и извлечение сведений о пользователе

Web-приложениям, требующим регистрации пользователей, приходится извлекать регистрационные данные пользователя, когда он снова посещает Web-сайт приложения. Например, Web-сайты электронных магазинов регистрируют сведения о счетах и товарах, отправленных некоторому пользователю, а затем извлекает эту информацию для заполнения полей адреса всякий раз, когда этот пользователь размещает новый заказ.

Изучив материал этого занятия, вы сможете:

- ✓ объяснить, каким образом файлы cookie позволяют идентифицировать пользователей и хранить сведения о них;
- ✓ сохранять простые или сложные данные на пользовательском компьютере при помощи файлов cookie;
- ✓ удалять файлы cookie с пользовательского компьютера;
- ✓ создавать уникальный идентификатор пользователя и хранить его на пользовательском компьютере;
- ✓ сохранять сведения о пользователях в XML-файлах и извлекать их оттуда при помощи созданного ранее уникального идентификатора пользователя.

Продолжительность занятия — около 35 минут.

Идентификация пользователей Web-приложения

Web-приложения могут идентифицировать пользователей по имени и паролю, требуя вводить их при каждом посещении Web-сайта, либо при помощи данных, хранящихся на пользовательских компьютерах в виде файлов cookie. Файлы *cookie* — это небольшие файлы, записываемые Web-приложением на клиентский компьютер.

Преимущество использования файлов cookie заключается в том, что оно делает взаимодействие между клиентом и сервером незаметным для пользователя. При использовании cookie пользователю не обязательно регистрироваться при каждом посещении Web-сайта, его данные просто автоматически появляются, когда они нужны.

Недостаток этого способа в том, что пользователь может запретить браузеру принимать файлы cookie. Некоторым пользователям не нравится, что Web-сайты хранят на их компьютерах данные, по которым можно отследить маршруты их перемещений в Интернете. Поэтому перед использованием файлов cookie обязательно следует проверить, принимает ли их клиент.

Есть два метода хранения и извлечения информации о пользователе с помощью файлов cookie:

- **все сведения о пользователе хранятся на клиентском компьютере в файле cookie** — этот способ удобен для хранения несложных сведений о пользователе, которые не нужны для исполнения других задач на сервере;
- **на клиентском компьютере хранится только идентификатор пользователя, необходимый для извлечения сведений из источника данных на сервере,** — этот вариант оптимален для хранения подробных сведений о пользователе.

Детальному рассмотрению обоих методов посвящены следующие разделы.

Хранение сведений о пользователе на клиентском компьютере

Чтобы сохранить файл cookie на клиентском компьютере, сделайте следующее.

1. Проверьте, поддерживает ли клиентский компьютер использование файлов cookie при помощи свойства *Cookies* объекта *Browser*.
2. Если эта проверка дала положительный результат, выясните, существует ли такой файл cookie при помощи набора *Cookies* объекта *Request*.
3. Если он не существует, создайте новый файл cookie с помощью класса *HttpCookie*.
4. Установите значения свойств *Value* и *Expiration* объекта, представляющего файл cookie.
5. Добавьте объект файла cookie к набору *Cookies* объекта *Response*.

Эти действия иллюстрируются следующим обработчиком события *Page_Load*, который создает файл cookie, регистрирующий время последнего посещения Web-страницы пользователем. Программа проверяет, разрешает ли пользователь принимать файлы cookie, и создает файл cookie, если это первый визит пользователя, либо обновляет файл, записанный ранее, если пользователь посещал эту страницу раньше.

Visual **Basic** **.NET**

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
        ' (1) Проверить, принимает ли браузер файлы cookie.
        If Request.Browser.Cookies Then
            ' (2) Если этот cookie не существует...
            If Request.Cookies("LastVisit") Is Nothing Then
                ' (3) ...создать новый файл cookie,
                Dim cookLastVisit As New HttpCookie("LastVisit", _
                  Now.ToString())
                ' (4) установить срок его действия в одни сутки,
                cookLastVisit.Expires = DateTime.Now.AddDays(1)
                ' (5) добавить его к набору cookies.
                Response.Cookies.Add(cookLastVisit)
                ' Отобразить сообщение.
                Response.Write("This is your first visit. ")
            Else
                ' Получить файл cookie.
                Dim cookLastVisit As HttpCookie = _
                Request.Cookies("LastVisit")
                ' Показать время последнего визита.
                Response.Write("You last visited this page: " & _
                  cookLastVisit.Value)
                ' Обновить cookie на клиентском компьютере.
                Response.Cookies("LastVisit").Value = Now.ToString()
                Response.Cookies("LastVisit").Expires = _
                  DateTime.Now.AddDays(1)
            End If
        Else
            Response.Write("Your browser does not accept cookies.")
        End If
    End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
      // (1) Проверить, принимает ли браузер файлы cookie.
      if (Request.Browser.Cookies)
      {
           // (2) Если этот cookie не существует...
           if (Request.Cookies["LastVisit"] == null)
           {
              // (3) ...создать новый файл cookie,
               HttpCookie cookLastVisit - new HttpCookie("LastVisit",
               DateTime.Now.ToString());
               // (4) установить срок его действия в одни сутки,
               cookLastVisit.Expires = DateTime.Now.AddDays(1);
               // (5) добавить его к набору cookies.
               Response.Cookies.Add(cookLastVisit);
               // Отобразить сообщение.
               Response.Write("This is your first visit.");
           }
           else
           {
               // Получить файл cookie,
               HttpCookie cookLastVisit = Request.Cookies["LastVisit1"];
               // Показать время последнего визита.
               Response.Write("You last visited this page: " +
                 cookLastVisit.Value);
               // Обновить cookie на клиентском компьютере.
               Response.Cookies["LastVisit"].Value =
                 DateTime.Now.ToString();
               Response.Cookies["LastVisit"].Expires =
                 DateTime.Now.AddDays(1);
           }
}
else
{
           Response.Write("Your browser does not accept cookies.");
      }
}
```

Немного о cookie

Имена файлов cookie чувствительны к регистру. Например, *LastVisit* и *Lastvisit* — это разные файлы. Свойство *Expires* определяет срок, по истечении которого клиент может отбросить cookie. По умолчанию период действия файлов cookie истекает по окончании сеанса пользователя. Если установить свойство *Expires* в *DateTime.MaxValue*, он будет длиться вечно.

Можно удалить cookie с клиентского компьютера, присвоив свойству *Expires* значение текущего времени. Например, следующий код удаляет cookie *LastVisit* с клиентскою компьютера:

Visual Basic .NET

```
Private Sub butRemoveCookie_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles butRemoveCookie.Click
      ' Установить свойство Expires так,
      ' чтобы срок действия cookie истек немедленно.
     Response.Cookies("LastVisit").Expires = DateTime.Now
End Sub
```

Visual C#

```
private void butRemoveCookie_Click(object sender, System.EventArgs e)
{
      // Установить свойство Expires так,
      // чтобы срок действия cookie истек немедленно,
      Response.Cookies["LastVisit"].Expires = DateTime.Now;
}
```

Применение ключей с файлами cookie

В одном файле cookie разрешается хранить до 4096 байт данных, идентифицируемых при помощи ключей. Например, следующий код сохраняет в cookie имя и адрес пользователя, связывая их с отдельными ключами:

Visual Basic .NET

```
Private Sub butOK_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butOK.Click
    ' Создать файл cookie,
    Dim cookUserInfo As New HttpCookie("UserInfo")
    ' Заполнить ключи данными из формы,
    cookUserInfo("FirstName") = txtFirstName.Text
    cookUserInfo("LastName") = txtLastName.Text
    cookUserInfo("Street") = txtStreet.Text
    cookUserInfo("City") = txtStreet.Text
    cookUserInfo("State") = drpState.SelectedItem.Value
    cookUserInfo("ZIP") = txtZIP.Text
    ' Установить срок действия cookie,
    cookUserInfo.Expires = DateTime.Now.AddDays(30)
    ' Добавить файл cookie.
    Response.Cookies.Add(cookUserInfo)
End Sub
```

Visual C#

```
private void butOK_Click(object sender, System.EventArgs e)
{
    // Создать файл cookie.
    HttpCookie cookUserInfo - new HttpCookie("UserInfo");
    // Заполнить ключи данными из формы.
    cookUserInfo["FirstName"] = txtFirstName.Text;
    cookUserInfo["LastName"] - txtLastName.Text;
    cookUserInfo["Street"] = txtStreet.Text;
    cookUserInfo["City"] = txtStreet.Text;
    cookUserInfo["State"] = drpState.SelectedItem.Value;
    cookUserInfo["ZIP"] = txtZIP.Text;
    // Установить срок действия cookie.
    cookUserInfo.Expires = DateTime.Now.AddDays(30);
    // Добавить файл cookie.
    Response.Cookies.Add(cookUserInfo);
}
```

Для получения значений из файла cookie достаточно имени ключа, как показано в следующем коде:

Visual Basic .NET

```
Private Sub butGetData_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butGetData.Click
    ' Получить файл cookie.
    Dim cookUserInfo As HttpCookie = Request.Cookies("UserInfo")
    ' Заполнить поля.
    txtFirstName.Text = cookUserInfo("FirstName")
    txtLastName.Text - cookUserInfo("LastName")
    txtStreet.Text = cookUserInfo("Street")
    txtCity.Text = cookUserInfoC'City")
    drpState.SelectedItem.Value = cookUserInfo("State")
    txtZIP.Text = cookUserInfoC'ZIP")
End Sub
```

Visual C#

```
private void butGetData_Click(object sender, System.EventArgs e)
{
    // Получить файл cookie.
    HttpCookie cookUserInfo = Request.Cookies["UserInfo"];
    // Заполнить поля.
    txtFirstName.Text - cookUserInfo["FirstName"];
    txtLastName.Text = cookUserInfo["LastName"];
    txtStreet.Text = cookUserInfo["Street"];
    txtCity.Text = cookUserInfo["City"];
    drpState.SelectedItem.Value = cookUserInfo["State"];
    txtZIP.Text = cookUserInfo["ZIP"];
}
```

Хранение сведений о пользователе на сервере

При хранении сведений о пользователе на сервере вместо клиентского компьютера, файлы cookie служат просто «опознавательным устройством» для пользователя, позволяющим сохранять и извлекать его данные на сервере. Для хранения сведений о пользователях на сервере можно задействовать базы данных, XML-файлы и любые другие хранилища данных.

Чтобы сохранить сведения о пользователе на сервере, выполните следующие действия.

1. Создайте уникальный ключ, идентифицирующий пользователя.
2. Сохраните созданный ключ на клиентском компьютере в виде файла cookie.
3. Создайте на сервере файл для хранения сведений о пользователе.
4. Сохраните сведения о пользователе на сервере, используя созданный уникальный ключ в качестве индекса.

Более подробно эти действия описаны в следующих разделах.

Создание уникальных ключей для идентификации пользователей

В .NET Framework предусмотрено пространство имен *System.Guid* — для создания глобально уникальных идентификаторов, или GUID. *GUID* — это 128-разрядное целое число, уникальное во всех сетях. GUID могут служить уникальными ключами для идентификации любых сущностей, Б том числе пользователей приложения.

Следующий код создает GUID и сохраняет в виде cookie на клиентском компьютере с целью идентификации пользователя в дальнейшем:

Visual Basic .NET

```
Private Sub butOK_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butOK.Click
    ' Получить cookie для запроса.
    Dim cookUserID As HttpCookie - Request.Cookies("UserID")
    ' Если cookie не существует, создать его.
    If cookUserlD Is Nothing Then
        ' Создать новый cookie с новым GUID,
        cookUserlD = New HttpCookie("UserID", _
          System.Guid.NewGuid().ToString())
        cookUserID.Name = "UserID"
    End If
    ' Установить срок действия cookie.
    cookUserID.Expires - DateTime.Now.AddDays(30)
    ' Добавить cookie к отклику.
    Response.Cookies.Add(cookUserID)
    ' Записать сведения о пользователе, взятые из данных формы.
    SetUserInfo(cookUserID.Value)
End Sub
```

Visual C#

```
private void butOK_Click(object sender, System.EventArgs e)
{
      // Получить cookie для запроса.
```

```
    HttpCookie cookUserID  = Request.Cookies["UserID"];
    // Если cookie не существует, создать его.
    if (cookUserID == null)
        // Создать новый cookie с новым GUID.
        cookUserID = new HttpCookie("UserID",
          System.Guid.NewGuid().ToString());
    // Установить срок действия cookie,
    cookUserID.Expires = DateTime.Now.AddDays(30);
    // Добавить cookie к отклику.
    Response.Cookies.Add(cookUserID);
    // Записать сведения о пользователе, взятые из данных
    // формы.
    SetUserInfo(cookUserID.Value);
}
```

Создания файлов для хранения сведений о пользователе

Одна из целей сохранения сведений о пользователе — возможность быстро извлечь их во время следующего визита последнего. Так как у тех, кто ранее уже посещал ваши страницы, уже есть уникальный идентификатор (это GUID, созданный в предыдущем разделе), имеет смысл использовать наборы данных для получения доступа к сведениям об этих пользователях. Наборы данных поддерживают метод *Find,* позволяющий извлекать строки с помощью уникального ключа.

Применение набора данных означает, что создаваемый на сервере файл должен быть файлом базы данных либо XML-файлом. XML позволяет записывать в файлы структурированные данные без издержек и сложностей, неизбежных при обращении к базе данных. Значит, для хранения простых данных, таких, как имя и адрес пользователя, лучше подходит XML-файл.

Чтобы создать XML-файл для хранения сведений о пользователе, выполните следующие действия.

1. В Visual Studio создайте XML-файл, содержащий примерные значения в полях данных, которые предназначены для хранения сведений о пользователе.
2. Сгенерируйте на основе XML-файла схему XML. Схема XML позволяет в наборе данных ссылаться по имени на данные, хранящиеся в XML-файле.
3. Задайте поле ключа в схеме XML, чтобы использовать его с методом *Find* для поиска записей в наборе данных.
4. Прочитайте содержимое схемы XML и XML-файла в набор данных.

Более подробно эти действия описаны в следующих разделах.

Создание XML-файла

Чтобы создать XML-файл в Visual Studio, выберите из меню Project команду Add New Item, затем выберите в списке Templates элемент XML File.

XML-файлы во многом похожи на HTML-файлы, но отличаются чувствительностью элементов и атрибутов XML к регистру и использованием строгого синтаксиса. Элементы данных XML-файла идентифицируются тэгами *<имя_элемента>* и *</имя_элемента>*. Следующий пример демонстрирует XML-файл, хранящий имя и адрес пользователя:

```
<?xml version="1.0" standalone="yes"?>
<USERS xmlns="http://tempuri.org/UserPrefs.xsd">
  <USER>
```

```
    <FIRSTNAME>Joan</FIRSTNAME>
    <LASTNAME>Reddington</LASTNAME>
    <STREET>436 Atlantic Ave.</STREET>
    <CITY>Melbourne Beach</CITY>
    <STATE>FL</STATE>
    <ZIP>32401</ZIP>
      <ID>79844302-6d86-452Q-ac64-c8c3240e21a9</ID>
  </USER>
</USERS>
```

Создание схемы XML

Для применения XML-файла с набором данных необходимо создать *схему XML* (XML schema) — описание элементов данных, содержащихся в XML-файле. Схема предоставляет имена и типы элементов, указывает, является ли поле полем ключа, и содержит ряд других сведений.

Чтобы сгенерировать схему XML на основе XML-файла, активируйте окно Design с загруженным XML-файлом и выберите в меню XML пункт Create Schema — Visual Studio создаст файл со схемой, описывающей этот XML-файл (рис. 7-1).

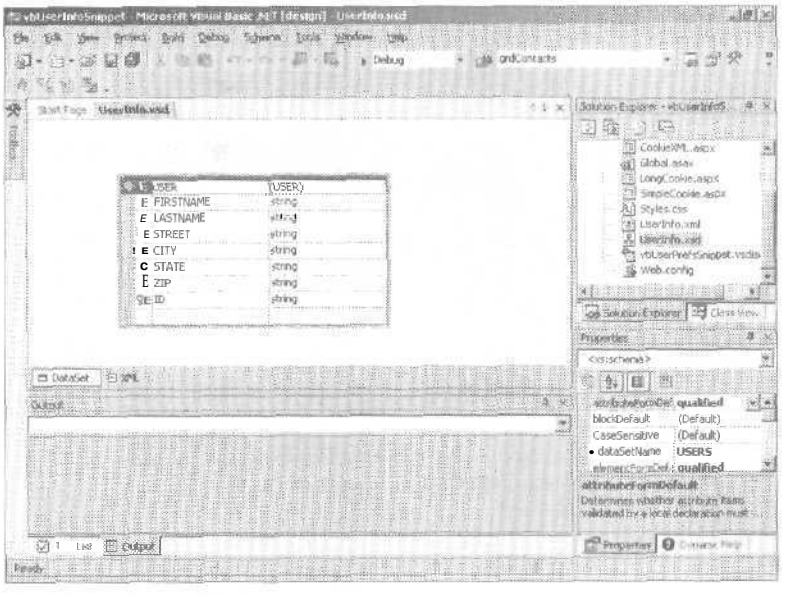


Рис. 7-1. Схема XML

Определение поля ключа

Чтобы предоставить возможность поиска для данных в формате XML, необходимо определить в схеме XML первичный ключ. В схеме, показанной на рис. 7-1, есть поле ключа (UserID), позволяющее искать в наборе данных сведения о нужных пользователях.

Чтобы добавить к схеме XML поле ключа, выполните следующие действия.

1. Щелкните правой кнопкой элемент, который хотите сделать ключом, и выберите из контекстного меню команду Add\New Key — Visual Studio откроет диалоговое окно Edit Key (рис.7-2).

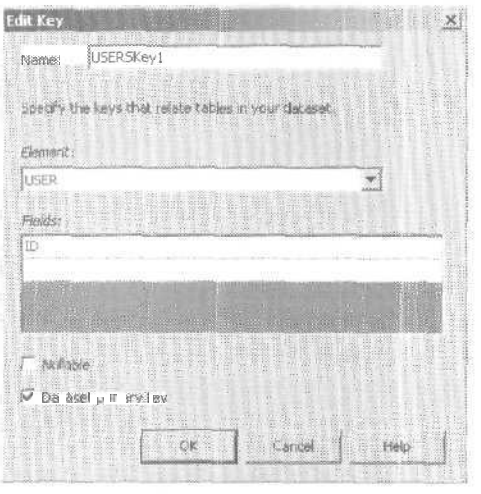


Рис. 7-2. **Диалоговое окно** Edit Key

2. Установите флажок Dataset Primary Key и щелкните OK, чтобы сделать выбранный элемент первичным ключом набора данных. Теперь этот элемент можно использовать с методом *Find* объекта *Dataset* для извлечения из набора данных нужных строк.

Заполнение набора данных с помощью XML-файла

После создания XML-файла и схемы XML можно прочитать хранящиеся в них данные и заполнить ими набор данных. Для этого применяют методы *ReadXmlSchema* и *ReadXml* соответственно. Следующий код демонстрирует вспомогательную функцию, возвращающую набор данных, который создан на основе XML-файла UserInfo:

Visual Basic .NET

```
Function GetUserData() As DataSet
    ' Задать путь к XML-файлу и схеме XML.
    Dim strPath As String = Server.MapPath(Request.ApplicationPath)
    ' Объявить набор данных.
    Dim dsUsers As New DataSet()
    ' Применить схему XML к набору данных.
    dsUsers.ReadXmlSchema(strPath & "\UserInfo.xsd")
    ' Прочитать содержимое XML и заполнить им набор данных.
    dsUsers.ReadXml(strPath & "\UserInfo.xml")
    Return dsUsers
End Function
```

Visual C#

```
private DataSet GetUserData()
{
      // Задать путь к XML-файлу и схеме XML.
      string strPath  = Server.MapPath(Request.ApplicationPath);
      // Объявить набор данных.
      DataSet dsUsers = new DataSet();
```

```
        // Применить схему XML к набору данных.
        dsUsers.ReadXmlSchema(strPath + "\\UserInfo.xsd");
        // Прочитать содержимое XML и заполнить им набор данных.
        dsUsers.ReadXml(strPath + "\\UserInfo.xml");
        return dsUsers;
}
```

Сохранение сведений о пользователе на сервере

Для сохранения сведений о пользователе на сервере в виде XML-файла применяется метод *SaveXML* объекта набора данных. Следующая процедура *SetUserInfo* вызывает процедуру *GetUserData*, описанную в предыдущем разделе, для получения набора данных из XML-файла, а затем при помощи метода *Find* проверяет наличие заданного UserID в полученном наборе данных. Если UserID не найден, *SetUserInfo* добавляет в набор данных строку, в которой будут храниться сведения о пользователе, а затем записывает в нее значения, веденные в серверные элементы управления на Web-форме. В завершение *SetUserInfo* записывает набор данных обратно на сервер.

Visual Basic .NET

```
Sub SetUserInfo(ByVal UserID As String)
    ' Задать путь к XML-файлу и схеме XML.
    Dim strPath As String = Server.MapPath(Request.ApplicationPath())
    ' Получить набор данных Users.
    Dim dsUsers As DataSet = GetUserData()
    ' Найти строку в наборе данных.
    Dim rowUser As DataRow = dsUsers.Tables("User").Rows.Find(UserID)
    ' Если строка не найдена, создать новую строку.
    If rowUser Is Nothing Then
        rowUser = dsUsers.Tables("User").NewRow
        dsUsers.Tables("User").Rows.Add(rowUser)
    End If
    ' Записать в строку данные из полей форма.
    rowUser("FirstName") = txtFirstName.Text
    rowUser("LastName") = txtLastName.Text
    rowUser("Street") = txtStreet.Text
    rowUser("City") = txtCity.Text
    rowUser("State") = drpState.SelectedItem.Text
    rowUser("ZIP") = txtZIP.Text
    rowUser("ID") = UserID
    ' Записать содержимое набора данных в XML-файл.
    dsUsers.WriteXml(strPath & "\UserInfo.xml")
End Sub
```

Visual C#

```
private void SetUserInfo(string UserID)
{
    // Задать путь к XML-файлу и схеме XML.
```

```
    string strPath  = Server.MapPath(Request.ApplicationPath);
    // Получить набор данных Users.
    DataSet dsUsers  = GetUserData();
    // Найти строку в наборе данных,
    DataRow rowUser - dsUsers.Tables["User"].Rows.Find(UserID);
    // Если строка не найдена, создать новую строку,
    if (rowUser == null)
    {
        rowUser = dsUsers.Tables["User"].NewRow();
        dsUsers.Tables["User"].Rows.Add(rowUser);
    }
    // Записать в строку данные из полей формы.
    rowUser["FirstName"] = txtFirstName.Text;
    rowUser["LastName"] = txtLastName.Text;
    rowUser["Street"] = txtStreet.Text;
    rowUser["City"] = txtCity.Text;
    rowUser["State"] = drpState.SelectedItem.Text;
    rowUser["ZIP"] = txtZIP.Text;
    rowUser["ID"] = UserID;
    // Записать содержимое набора данных в XML-файл.
    dsUsers.WriteXml(strPath + "\\UserInfo.xml");
}
```

Извлечение сведений о пользователе из набора данных

Извлечение сведений о пользователе из XML-файла производится так. Сначала при помощи процедуры *GetUserData* следует создать набор данных на основе XML-файла, затем нужно вызвать метод *Find* для извлечения строки с заданным значением UserID. Ниже показана процедура *GetUserInfo*, которая извлекает из набора данных сведения о пользователе и записывает их в серверные элементы управления Web-формы.

Visual Basic .NET

```
Sub GetUserInfo(ByVal UserID As String)
    ' Получить набор данных Users,
    Dim dsUsers As Data.DataSet = GetUserData()
    ' Найти строку в наборе данных.
    Dim rowUser As Data.DataRow - dsUsers.Tables("User").Rows.Find(UserID)
    ' Если заданный пользователь не найден, выйти из процедуры.
    If rowUser Is Nothing Then Exit Sub
    ' Записать содержимое строки в поля формы.
    txtFirstName.Text = rowUser.Item("FirstName")
    txtLastName.Text  = rowUser.Item("LastName")
    txtStreet.Text - rowUser.Item("Street")'
    txtCity.Text  - rowUser.Item("City")
    drpState.SelectedItem.Text = rowUser.Item("State")
    txtZIP.Text  = rowUser.Item("ZIP")
End Sub
```

Visual C#

```
void GetUserInfo(string UserID)
{
    // Получить набор данных Users.
    DataSet dsUsers = GetUserData();
    // Найти строку в наборе данных.
    DataRow rowUser = dsUsers.Tables["User"].Rows.Find(UserID);
    // Если пользователь не найден, выйти из процедуры.
    if (rowUser == null) return;
    // Записать содержимое строки в поля формы.
    txtFirstName.Text = rowUser["FirstName"].ToString();
    txtLastName.Text = rowUser["LastName"].ToString();
    txtStreet.Text = rowUser["Street"].ToString();
    txtCity.Text = rowUser["City"].ToString();
    drpState.SelectedItem.Text = rowUser["State"].ToString();
    txtZIP.Text = rowUser["ZIP"].ToString();
}
```

Занятие 2. Отправка почты

Есть два способа отправки почты из Web-приложений:

- **с помощью протокола mailto:** — применяется для отправки почты с клиента и использует установленную на нем почтовую программу. Этот способ удобен, когда требуется собрать отзывы о Web-сайте, предоставить техническую поддержку и решить другие задачи, требующие ответа на присланные пользователями сообщения;
- **с помощью классов *MailMessage* и *SmtpMail*** — применяется для отправки почты с сервера. Этот метод использует серверную почтовую систему и позволяет автоматически уведомлять об ошибках, рассылать анонимные сообщения и решать другие задачи, не требующие знания адреса отправителя (клиента).

Освоив материал этого занятия, вы научитесь отправлять почту обоими способами.

Изучив материал этого занятия, вы сможете:

- ✓ выбирать способ отправки почты в соответствии с поставленной задачей;
- ✓ создавать гиперссылки для запуска пользовательской почтовой программы;
- ✓ создавать почтовые сообщения с помощью пользовательской и серверной почтовых программ;
- ✓ отправлять почтовые сообщения с сервера.

Продолжительность занятия — около 20 минут.

Отправка почты с клиентского компьютера

Чтобы отправить почту с помощью клиентской почтовой системы, необходимо создать гиперссылку, использующую протокол mailto. Ниже показан пример HTML-кода, который отображает гиперссылку, создающую новое сообщение с заданной темой и коротким текстом:

```
<A href="mailto:someone@microsoft.com?SUBJECT=Sending from
a client&BODY=Some message text.">
```

Когда пользователь щелкает эту гиперссылку, браузер клиента запускает почтовую программу, установленную на клиентском компьютере, и создает сообщение, показанное на рис. 7-3.

Протокол mailto вы можете использовать в серверных или HTML-элементах управления, как обычную строку URL. Например, следующий HTML-код определяет серверный элемент управления Hyperlink, отправляющий сообщение:

```
<asp:HyperLink ID="hypMail"
NavigateUrl="mailto:someone@microsoft.com?
SUBJECT=Mailing a Webform&amp;BODY=Some message text."
Runat="server">Send mail.</asp:HyperLink>
```

Протокол mailto также используют в коде как аргумент метода *Redirect,* однако при этом возможен нежелательный побочный эффект: при создании сообщения в браузере отображается пустая страница.

Текст различных частей сообщения задают посредством ключей, поддерживаемых протоколом mailto. Эти ключи составляют строку запроса и разделяются знаками «&» (см. пример ниже и таблицу 7-1).

```
mailto:почтовый_адрес[&ключ=значение][&ключ=значение]...
```

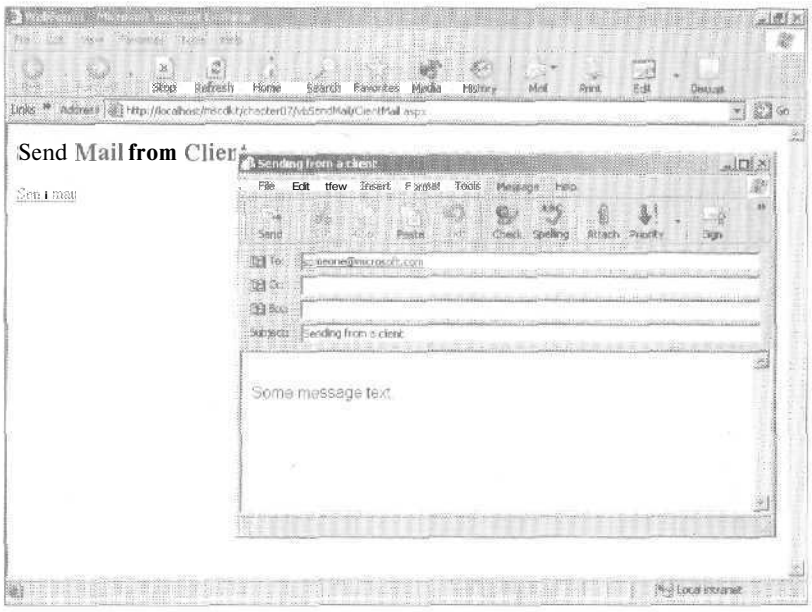


Рис. 7-3. Создание почтового сообщения при помощи гиперссылки

Таблица 7-1. Ключи протокола mailto

Ключ	Назначение
SUBJECT	Задает текст темы сообщения
CC	Определяет список адресов (разделитель — знак «;»), по которым будут отправлены копии сообщения
BCC	Задает список адресов (разделитель — знак «;»), по которым будут отправлены скрытые копии
BODY	Определяет текст сообщения

Протокол mailto не поддерживает ключ для прикрепления файлов к сообщению, но пользователь может это сделать вручную в окне сообщения. Если нужно, чтобы пользователь прикрепил к сообщению файл (например, журнал ошибок или какой-либо другой отчет), включите соответствующие инструкции в тело сообщения.

Отправка почты с сервера

При отправке почты с сервера ASP.NET не запускает клиентскую почтовую систему, чтобы пользователь смог создать сообщение, прикрепить к нему файлы и проверить адрес. Вместо этого сообщение создается и отправляется программно одним из следующих способов:

- для отправки простого сообщения применяют метод *Send* класса *SmtpMail*;
- более сложное сообщение можно создать, используя класс *MailMessage*, а затем отправить его посредством метода *Send* класса *SmtpMail*.

Оба класса — *SmtpMail* и *MailMessage*— определены в пространстве имен .NET Framework *System.Web.Mail*, поэтому следует поместить в начале модуля следующие операторы, чтобы использовать в коде более простые ссылки на эти классы:

Visual Basic .NET

```
Imports System.Web.Mail
```

Visual C#

```
using System.Web.Mail;
```

Чтобы воспользоваться классом *SmtpMail*, не обязательно создавать его экземпляр. Можно отправить простое сообщение, просто вызвав метод *Send,* как показано ниже:

Visual Basic .NET

```
SmtpMail.Send ("someone@microsoft.com", "jesse@contoso.com", _
               "Subject line", _
               "Messagetext.")
```

Visual C#

```
SmtpMail.Send("someone@microsoft.com", "jesse@contoso.com", "Subject line",
"Message text.");
```

Предыдущая строка кода немедленно отправляет сообщение от someone@microsoft.com на адрес jesse@contoso.com. Адреса отправителя и получателя обязательны, но метод *Send* не проверяет их. По умолчанию класс *SmtpMail* применяется для отправки почты локальный SMTP-сервер. Чтобы использовать другой сервер, установите свойство *SmtpServer* этого класса.

Метод *Send* подходит для рассылки простых сообщений, которые содержат только текст, но для отправки сообщения с форматированным текстом или с прикрепленным файлом придется создать объект класса *MailMessage*, а затем вызвать метод *Send* класса *SmtpMail* для отправки сообщения:

Visual Basic .NET

```
Private Sub butMail_Click(ByVal sender As System.Object, +
      ByVal e As System.EventArgs) Handles butMail.Click
    ' Создать сообщение.
    Dim mailNew As New MailMessage()
    ' Установить свойства сообщения.
    mailNew.From = "someone@microsoft.com"
    mailNew.To = "jesse@contoso.com"
    mailNew.Subject = "This is the subject text."
    mailNew.Body = "This is the message text,"
    ' Создать вложение.
    Dim atcItem As New _
    MailAttachment(Server.MapPath(Request.ApplicationPath) & "\joey.jpg")
    ' Вложить его в сообщение.
    mailNew.Attachments.Add(atcItem)
    ' Отправить сообщение.
    SmtpMail.Send(mailNew)
End Sub
```

Visual C#

```
private void butSendMail_Click(object sender, System.EventArgs e)
{
    // Создать сообщение.
    MailMessage mailNew = new MailMessage();
    // Установить свойства сообщения.
    mailNew.From = "someone@microsoft.com";
    mailNew.To = "jesse@contoso.com";
    mailNew.Subject = "This is the subject text.";
    mailNew.Body = "This is the message text.";
    // Создать вложение.
    MailAttachment atcItem = new MailAttachment
     (Server.MapPath(Request.ApplicationPath) +
     "\\joey.jpg");
    // Вложить его в сообщение.
    mailNew.Attachments.Add(atcItem);
    // Отправить сообщение.
    SmtpMail.Send(mailNew);
}
```

Этот код создает сообщение, устанавливает его свойства, прикрепляет к нему файл и отправляет сообщение. И в этом случае при использовании метода *Send* необходимо установить свойства *From* и *To,* но их значения не проверяются.

Занятие 3. **Применение фреймов**

Фреймы (frames) — это области Web-страницы, в которых можно отображать другие Web-страницы. Фреймы позволяют отображать несколько областей, которые прокручиваются и функционируют независимо друг от друга. Из этого занятия вы узнаете, как создавать фреймы в Visual Studio, и научитесь управлять одним фреймом из другого.

Изучив материал этого занятия, вы сможете:

- ✓ создавать наборы фреймов, определяющих отдельные области для отображения содержимого Web-приложения;
- ✓ проверять, поддерживает ли браузер пользователя фреймы;
- ✓ отображать Web-формы и HTML-страницы в областях, определяемых набором фреймов;
- ✓ управлять главным фреймом из фрейма, отображающего информационное наполнение, при помощи гиперссылок и клиентских сценариев.

Продолжительность занятия — около 15 минут.

Создание фреймов в Visual Studio

Чтобы создать фреймы в Visual Studio, выполните следующие действия.

1. Создайте набор фреймов.
2. Создайте Web-страницы, которые будут отображаться во фреймах.
3. Назначьте для каждого фрейма из набора Web-страницу.

Более подробно эти действия описаны в следующих разделах.

Создание набора фреймов

Чтобы создать набор фреймов в Visual Studio, выполните следующие действия.

1. Из меню Project выберите команду Add New Item — Visual Studio откроет диалоговое окно Add New Item, показанное на рис. 7-4.

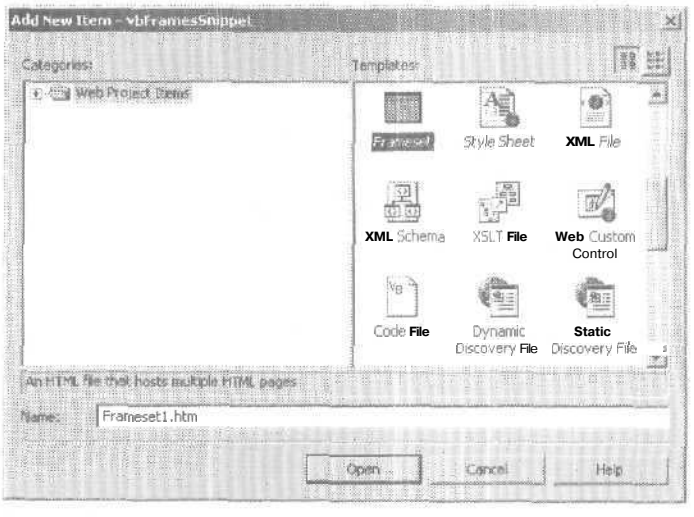


Рис. 7-4. Диалоговое окно Add New Item

2. В диалоговом окне Add New Item выберите из списка Categories элемент U1 или Web, затем из списка Templates выберите Frameset; в текстовом поле Name введите имя создаваемой Web-страницы и щелкните Open. В результате Visual Studio откроет диалоговое окно Select A Frameset Template, показанное на рис. 7-5.

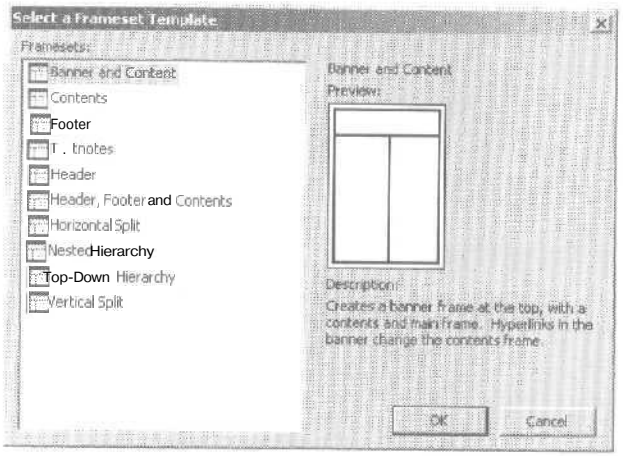


Рис. 7-5. Диалоговое **окно Select A Frameset Template**

Выберите тип фреймов из списка Framesets и шелкните OK. Visual Studio создаст Web-страницу в виде .htm-файла и откроет ее в окне Designer (рис. 7-6).

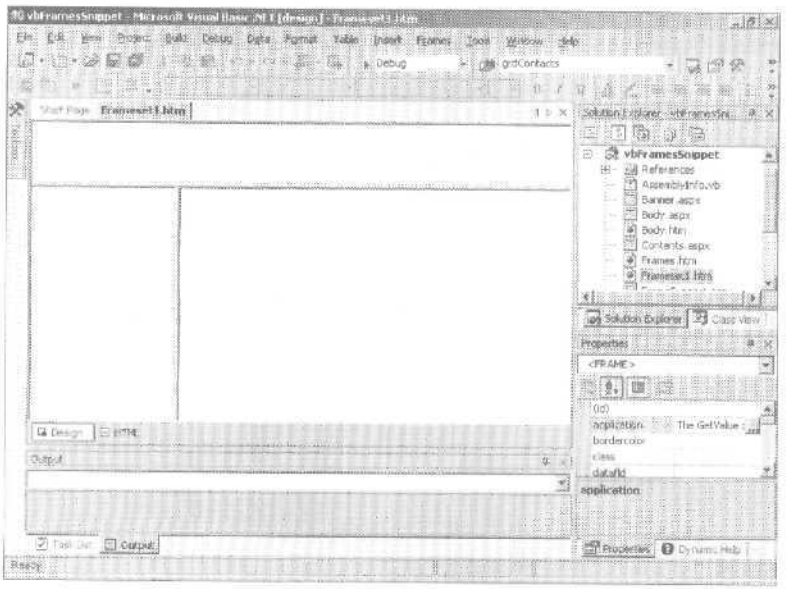


Рис. 7-6. Новый набор пустых фреймов

Добавление страниц к набору фреймов

В каждом фрейме созданного набора фреймов можно отображать HTML-страницу или Web-форму. Сам набор фреймов представляет собой HTML-файл, но отображаемые во фреймах страницы могут быть как HTML-файлами, так и Web-формами.

Создав страницы, которые должны отображаться во фреймах, нужно назначить каждому фрейму HTML-страницу или Web-форму. Это делается следующим образом.

1. Щелкните правой кнопкой фрейм, для которого следует назначить страницу, и выберите из контекстного меню команду Set Page For Frame. Visual Studio откроет диалоговое окно Select Page, показанное на рис. 7-7.

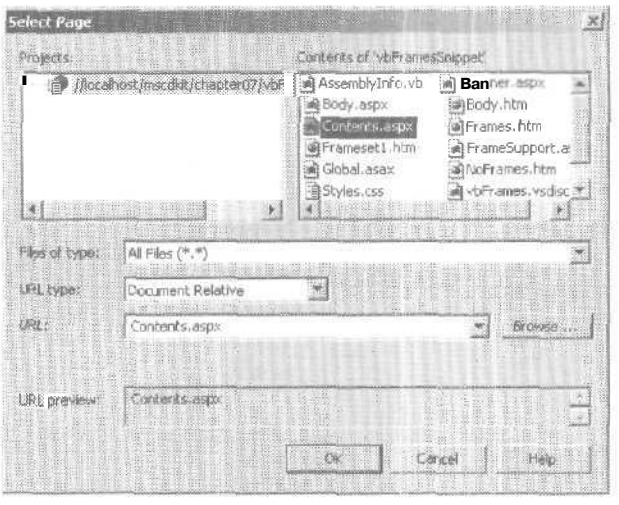


Рис. 7-7. **Диалоговое окно Select Page**

2. Выберите из списка Contents страницу, которая должна отображаться в этом фрейме, и щелкните OK — Visual Studio добавит выбранную страницу к фрейму.
3. Повторите пункты 1 и 2 для каждого фрейма из набора. После этого набор фреймов должен выглядеть, как показано на рис. 7-8.

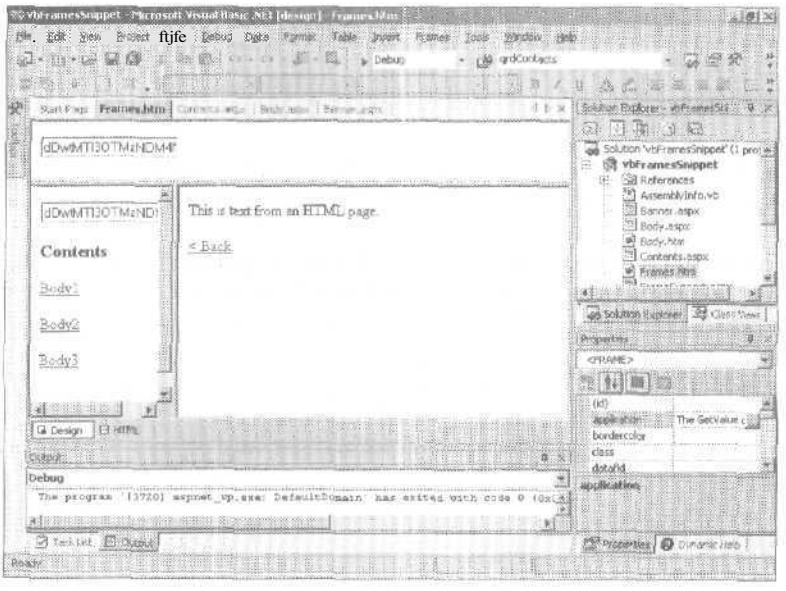


Рис. 7-8. Вид **готового набора фреймов в режиме Design**

В режиме Design Visual Studio не способна корректно отображать Web-формы в наборе фреймов — вместо них появляются сообщения об ошибках либо искаженные элементы управления (рис. 7-8), но во время выполнения набор фреймов отображается правильно.

Чтобы увидеть, как отображается набор фреймов во время выполнения, можно:

- сделать набор фреймов начальной страницей Web-приложения и запустить приложение;
- перейти на страницу набора фреймов во время выполнения по ссылке или вызвав метод *Redirect* в коде.

Во время выполнения набор фреймов примерно выглядит, как на рис. 7-9.

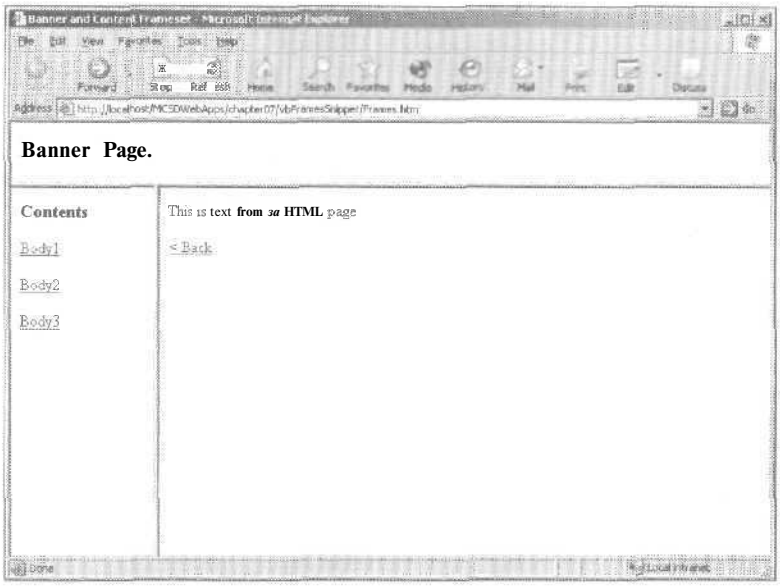


Рис. 7-9. Вид готового набора фреймов во время выполнения

Проверка наличия поддержки фреймов в браузере

Прежде чем решиться на широкое применение фреймов в приложении, знайте, что ранние версии некоторых браузеров их не поддерживают. Если клиент, работающий с таким браузером, запросит страницу с фреймами, то отобразится пустая страница или программа вовсе зависнет.

Чтобы перед применением набора фреймов выяснить, поддерживает ли браузер фреймы, воспользуйтесь свойством *Frames* объекта *Browser*. Например, следующий код отображает набор фреймов или обычную HTML-страницу в зависимости от того, поддерживает ли фреймы клиентский браузер.

Visual Basic .NET

```
Private Sub butDisplayPage_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles butDisplayPage.Click
    ' Если клиентский браузер поддерживает фреймы, отобразить
    ' набор фреймов...
    If Request.Browser.Frames Then
        Server.Transfer("frames.htm")
```

```
' ...в противном случае отобразить обычную HTML-страницу.
    Else
        Server.Transfer("noframes.htm")
    End If
End Sub
```

Visual C#

```
private void butDisplayPage_Click(object sender, System.EventArgs e)
{
      // Если клиентский браузер поддерживает фреймы, отобразить
      // набор фреймов...
      if (Request.Browser.Frames)
            Server.Transfer("frames.htm");
      // ...в противном случае отобразить обычную HTML-страницу.
      else
            Server.Transfer("noframes.htm");
}
```

Кроме того, в HTML-коде страницы с фреймами можно определять альтернативное содержимое, основываясь на поддержке фреймов — посредством элемента *<NOFRAMES>*. Этот элемент поддерживается в Internet Explorer версии 3.0 и выше, а также входит в спецификацию HTML 4.0.

При создании набора фреймов Visual Studio автоматически генерирует в HTML-тексте элемент *<NOFRAMES>*,показанный в следующем примере:

```
<frameset rows="64,*">
      <frame name="banner" src="Banner.aspx" scrolling="no"
      noresize>
      <frameset cols="150,*">
            <frame name="contents" src="Contents.aspx">
            <frame name="main" src="Body.aspx">
      </frameset>
      <noframes>
            <P>
            This HTML frameset displays multiple Web pages. To view this
            frameset, use a Web browser that supports HTML 4.0 and later.
            </p>
      </noframes>
</frameset>
```

Можно модифицировать этот текст, поместив в него альтернативное содержимое или ссылку на аналогичную страницу без фреймов.

Использование ссылок для перехода между фреймами

Каждый фрейм в наборе фреймов функционирует как отдельное окно браузера. Любые страницы, на которые ведут ссылки со страницы, отображенной в данном фрейме, будут отображаться в том же фрейме.

Чтобы создать ссылку для перехода между страницами, отображаемыми в разных фреймах, добавьте к ссылке атрибут *TARGET*. Например, следующий HTML-код показывает список ссылок, ведущих на главный фрейм набора фреймов.

```
<h3>Contents</h3>
<PXA href="Body1.aspx" target="main">Body1</A></P>
<P><A href="Body2.aspx" target="main">Body2</A></P>
<P><A href="Body3.aspx" target="main">Body3</A></P>
```

Если щелкнуть одну из этих ссылок, содержимое фрейма с именем «main» изменится, как показано на рис. 7-10.

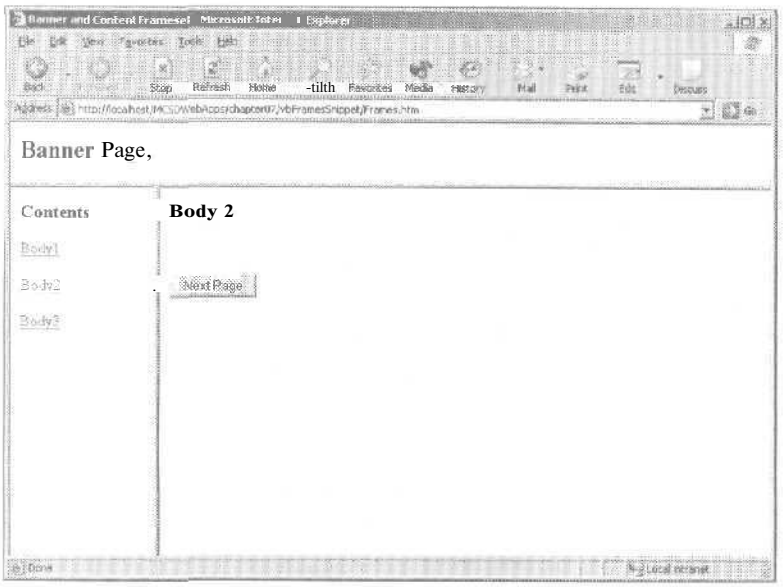


Рис. 7-Ю. Переход между страницами по ссылке

Нельзя использовать методы *Redirect*, *Transfer* и *Execute* для перехода между фреймами в коде, для этой цели годятся только гиперссылки или серверные элементы управления Hyperlink.

Занятие 4. Применение клиентских сценариев

Если ASP.NET предоставляет полный набор инструментов программирования для создания серверных Web-приложений, зачем нужны клиентские сценарии? Поскольку сценарии обладают прямым доступом к клиентскому браузеру, они позволяют делать многое из того, что невозможно осуществить из серверного кода, например:

- **управлять окном браузера.** Поскольку делать это из серверного кода нельзя, для открытия нового окна браузера, передачи фокуса в окне, перехода между фреймами и страницами из списка History, а также для решения ряда других задач, связанных с использованием браузера, требуются клиентские сценарии;
- **немедленно реагировать на события,** генерируемые **при перемещении указателя мыши над** элементом **страницы.** Серверный код способен реагировать на события только после возврата страницы на сервер, тогда как клиентский код немедленно реагирует на событие, генерированное страницей;
- **запускать клиентскую почтовую систему.** Подробно об этом — в занятии 2 этой главы.

На этом занятии вы научитесь добавлять сценарии к Web-формам и HTML-страницам в Visual Studio, чтобы создать код, работающий на клиентском компьютере.

Изучив материал этого занятия, вы сможете:

- ✓ проверять наличие поддержки сценариев в браузере;
- ✓ выбирать язык сценариев для решения поставленной задачи;
- ✓ управлять окном браузера и загруженными в него документами при помощи объектов Document Object Model (DOM);
- ✓ создавать сценарии, встроенные в страницу и исполняемые во время интерпретации страницы браузером;
- ✓ создавать сценарии, реагирующие на события, генерируемые браузером в ответ на щелчки и перемещение указателя мыши;
- ✓ принимать информацию от пользователя и отображать отклик при помощи сценариев.

Продолжительность занятия — около 30 минут.

Что нужно учесть перед применением сценариев

Применяя клиентские сценарии, следует знать, что:

- не все браузеры поддерживают сценарии;
- VBScript поддерживает только Internet Explorer;
- использование сценариев таит в себе угрозу безопасности.

Поскольку не все браузеры поддерживают сценарии, перед отображением страниц со сценариями на клиенте следует проверить возможности его браузера. Ниже показан обработчик события *Page_Load*, который проверяет, поддерживает ли браузер сценарии и, если нет, направляет пользователя на страницу, рекомендующую ему обновить свой браузер.

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Hardies MyBase.Load
    If Request.Browser.VBScript Then
        Response.Redirect("VBScripts.htm")
```

```
        Exit Sub
    Elseif Request.Browser.JavaScript Then
        Response.Redirect ("JScripts.htm")
    Else
        Response.Redirect("NoScripts.htm")
    End If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    if (Request.Browser.VBScript)
        Response.Redirect("VBScripts.htm");
    else if (Request.Browser.JavaScript)
        Response.Redirect("JScripts.htm");
    else
        Response.Redirect("NoScripts.htm");
}
```

При работе с клиентскими сценариями также нельзя забывать о безопасности. У этой проблемы два аспекта. Во-первых, уязвимы сами сценарии — их может просматривать и копировать любой, кто работает с приложением. Во-вторых, существует потенциальная опасность распространения вирусов в сценариях, поэтому некоторые пользователи отключают работу сценариев в параметрах безопасности браузера.

Однако свойства *VBScript* и *JavaScript* объекта *Browser* возвращают *True,* даже если сценарии отключены, поэтому придется проявить немного сообразительности, чтобы проверить, используются ли сценарии в браузере. Следующий HTML-код проверяет, включены ли сценарии, пытаясь запустить небольшой сценарий. Если их работа разрешена, пользователь будет направлен на другую страницу и не увидит сообщение, следующее за строкой, которая запускает сценарий:

```
<html>
   <script>
      window.navigate("scripts.aspx")
   </script>
   <!- Если исполнение сценариев разрешено, пользователь
        никогда не увидит следующее сообщение ->
   <head>
   <title>Scripts</title>
   </head>
   <body MS_POSITIONING="FlowLayout">
   <h2>Scripting is turned off.</h2>
   <p>Your Internet security options specify that your browser will not run
   scripts, therefore you cannot view the page you requested. To turn
   scripting on, reset your browser's Internet security options to Medium, or
   enable active scripting in the custom s ecurity settings.</p>
   <p><a href="CheckBrowser.htm">Click here</a> to try again.</p>
   </body>
</html>
```

Выбор языка сценариев

Клиентский сценарий пишется на VBScript либо на JScript. Языки сценариев отличаются от компилируемых языков, таких, как Visual Basic .NET или Visual C#, следующим:

- тексты на языках сценариев не компилируются, а интерпретируются браузером во время выполнения;
- ключевые слова языков сценариев — это «безопасные варианты» своих аналогов из родительского языка. Сценарии не позволяют создавать и удалять файлы с пользовательского диска, а также выполнять действия, чреватые крахом клиентского компьютера.

Возможности языков VBScript и JScript эквивалентны — они различаются лишь синтаксисом и ключевыми словами. Как правило, программирующие на Visual C# предпочитают JScript, поскольку этот язык следует соглашениям C, а тем, кто пишет на Visual Basic, больше нравится VBScript, поскольку он очень похож на Visual Basic. Однако VBScript поддерживает только Internet Explorer, поэтому, если приложение должно работать в других браузерах, следует использовать JScript.

Если вы программируете на Visual Basic, возможно, вам не понравится, что JScript чувствителен к регистру. Для нормальной работы сценариев, написанных на JScript, для всех ключевых слов, переменных, ссылок на объекты, методов и свойств необходимо задать правильный регистр.

Подробно о языках сценариев рассказано в следующих разделах справочной системы Visual Studio:

- «JScript User's Guide»;
- «JScript Language Reference»;
- «VBScript User's Guide»;
- «VBScript Language Reference».

Знакомство с DOM

Document Object Model (DOM) состоит из объектов, поддерживаемых браузером для работы в сценариях. Эти объекты позволяют управлять окном браузера, страницей (или документом), загруженной в него в данное время, а также объектами страницы. На рис. 7-11 показаны объекты, поддерживаемые DOM.

Обсуждение всех объектов DOM не входит в задачи этой книги, достаточно подробно о них рассказано в разделе «DHTML References» справочной системы Visual Studio. Этот раздел, а также источники, на которые он ссылается, входят в состав справочной системы Web Workshop,

Добавление встроенных сценариев

В Web-страницах зачастую встроены сценарии, которые могут быть частью текста самой страницы либо обработчика одного из ее событий. Встроенные сценарии исполняются, когда браузер обнаруживает их. В предыдущем разделе показан пример встроенного сценария, направляющего пользователя на другую страницу, если браузер разрешает исполнение сценариев:

```
<html>
<script>
window.navigate("scripts.aspx")
</script>
   <!- Если исполнение сценариев разрешено, пользователь
        никогда не увидит следующее сообщение ->
```

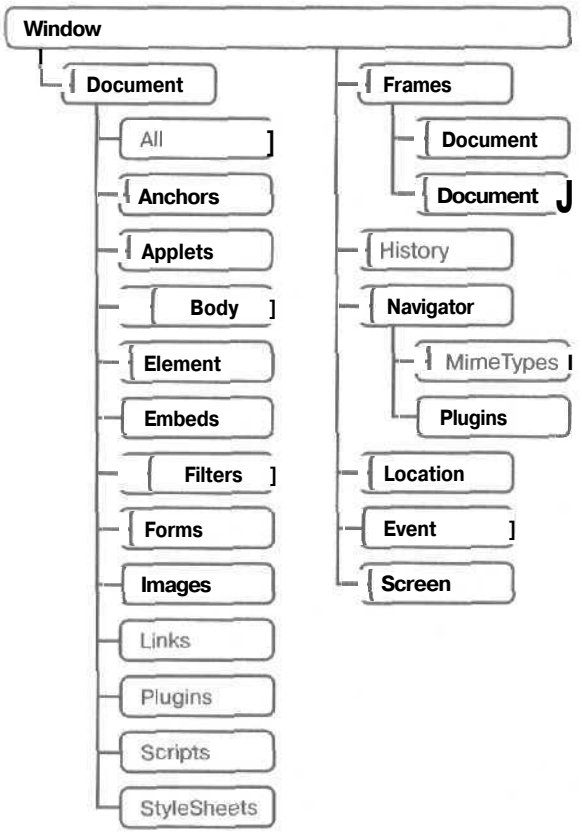


Рис. 7-11. Иерархия объектов DOM

```
    <head>
        <title>Scripts</title>
    </head>
    <body MS_POSITIONING="FlowLayout" onunload="return window_onunload()">
        <h2>Scripting is turned off.</h2>
        <p>Your Internet security options specify that your browser will
        not run scripts, therefore you cannot view the page you requested.
        To turn scripting on, reset your browser's Internet security
        options to Medium, or enable active scripting in the custom security
        settings.</p>
        <pxa href="CheckBrowser.htm">Click here</a> to try again.</p>
    </body>
</html>
```

Если сценарий будет исполнен, пользователь никогда не увидит сообщение о том, что поддержка сценариев отключена; сценарии обрабатываются прежде, чем любое другое содержимое страницы. Любые элементы страницы, на которые ссылается встроенный сценарий, следует располагать перед этим сценарием. Например, следующий HTML-код отображает два текстовых поля и при помощи сценария перемешает курсор во второе ноле:

VBScript

```
<HTML>
   <body language="vbscript">
         <form id="Form1" method="post" runat="server">
               <P>
               <asp:TextBox id="txtFirstName" Runat="server">
               </asp:TextBox>
               </P>
               <P>
               <asp:TextBox id="txtLastName" Runat="server">
               </asp:TextBox>
               </P>
               <script>window.document.all("txtLastName").focus()
               </script>
         </form>
   </body>
</HTML>
```

JScript

```
<HTML>
   <body language="javascript">
         <form id="Form1" method="post" runat="server">
               <P>
               <asp:TextBox id="txtFirstName" Runat="server">
               </asp:TextBox>
               </P>
               <P>
               <asp:TextBox id="txtLastName" Runat="server">
               </asp:TextBox>
               </P>
               <script>window.document.all["txtLastName"].focus()
               </script>
         </form>
   </body>
</HTML>
```

Встроенные сценарии можно применять и в качестве атрибутов элементов HTML, выполняющих различные действия, как это делается в случае HTML-элемента управления Button. Вот пример HTML-кода кнопки, открывающей справочную страницу в новом окне:

```
<button id="butHelp" onclick="window.open('help.aspx', 'help',
'height=200,width=300')">Help</button>
```

Заметьте, что этот встроенный сценарий не зависит от используемого языка Другими словами, он работает независимо от того, установлен атрибут *language* элемента *<body>* в VBScript или в JScript. Чтобы использовать язык сценариев, отличный от заданного элемента *<body>*, следует определить атрибут *language* элемента *<script>*. Вот пример сценария, отображающего список заголовков различных уровней:

VBScript

```
<HTML>
   <script language="vbscript">
   For i = 1 to 6
         document.write("<h" & i & ">")
         document.write("H" & i)
         document.write("</h" & i & "><br>")
   Next
   </script>
</HTML>
```

JScript

```
<HTML>
   <script language="javascript">
   for (i = 1; i <= 6; i++)
   {
         document.write("<h" + i + ">");
         document.write("H" + i);
         document.write("</h" + i + "><br>");
   }
   </script>
</HTML>
```

Этот код отображает шесть уровней заголовков HTML (рис. 7-12).

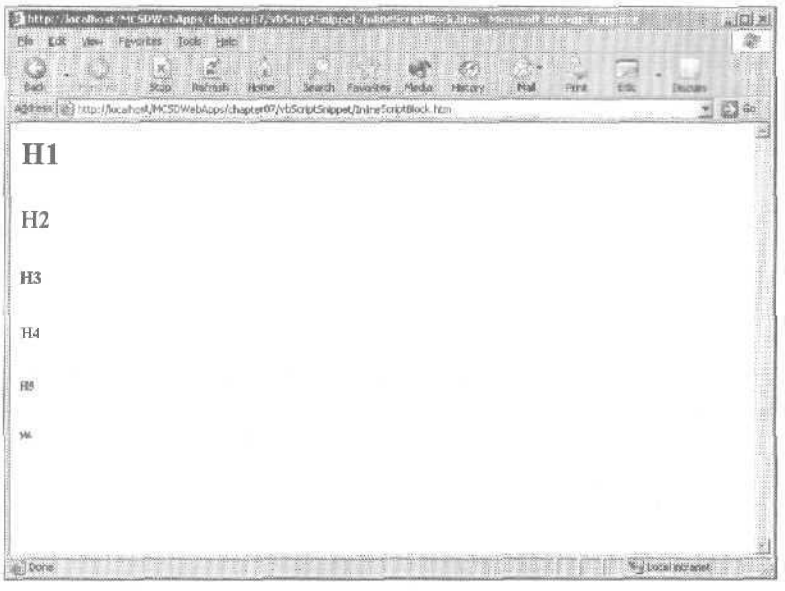


Рис. 7-12. Результат исполнения встроенного сценария

Обработка событий с помощью сценариев

При помощи сценариев также можно определять обработчики событий, генерируемых на странице, таких, как загрузка документа в окно, щелчки кнопок и перемещение указателя мыши над объектами.

Чтобы создать обработчик события в виде сценария, **выполните** следующие действия.

1. **Создайте или откройте в** Visual Studio страницу, **к которой требуется** добавить клиентские **сценарии. Это может быть как** Web-форма (.aspx-файл), **так и HTML-страница** (.htm-файл).
2. **Переключитесь в** режим **HTML.**
3. Выберите в раскрывающемся списке, расположенном вверху слева (рис. 7-13), объект, к которому нужно добавить обработчик события.

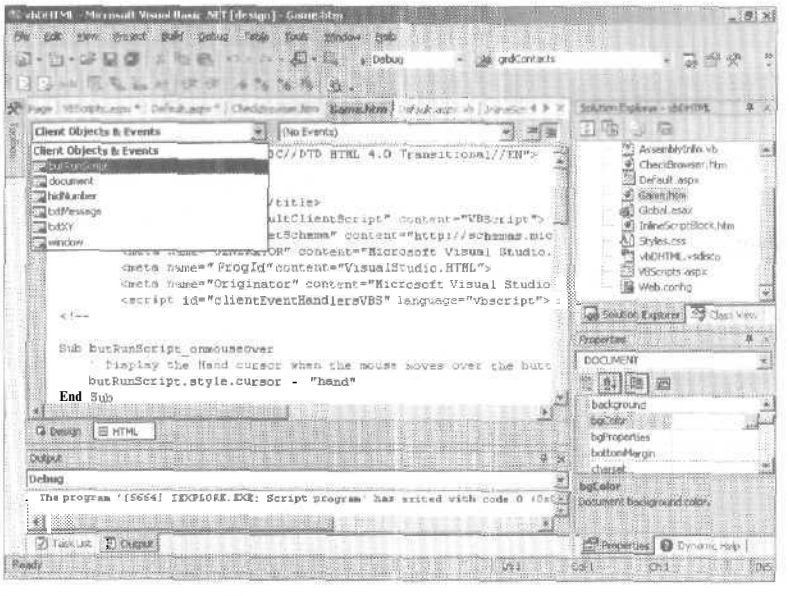


Рис. 7-13. Выбор объекта

4. **Выберите в раскрывающемся списке, расположенном вверху** справа (рис. 7-14), **обрабатываемое событие.**

Visual **Studio создаст блок сценария с пустым обработчиком, например,** таким:

VBScript

```
<script id=clientEventHandlersVBS language=vbscript>
<!-

Sub butRunScript_onmouseover

End Sub

->
</script>
```

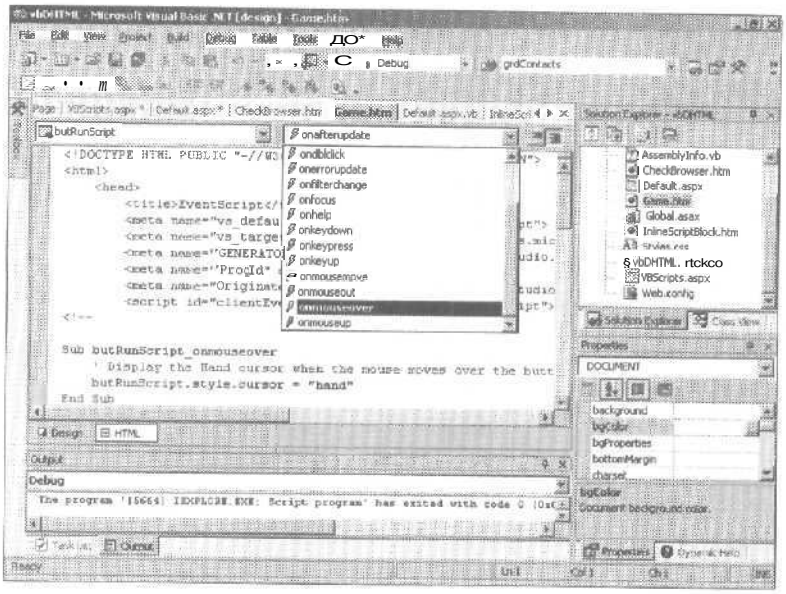


Рис. 7-14. Выбор события

JScript

```
<script id=clientEventHandlersJS language=javascript>
<!-

function butRunScript_onmouseover() {

}

//->
</script>
```

Когда страница сгенерирует это событие, будет исполнен любой код, добавленный к его обработчику. Например, следующий код изменяет вид указателя мыши, когда он находится над кнопкой, превращая его из стрелки в значок в виде ладони:

VBScript

```
Sub butRunScript_onmouseover
   butRunScript.style.cursor = "hand"
End Sub
```

JScript

```
function butRunScript_onmouseover() {
   // Отображает указатель мыши в форме ладони при перемещении
   // его над кнопкой.
   butRunScript.style.cursor = "hand";
}
```

Обработчики событий, исполняемые на клиентской стороне, способны реагировать на целый ряд событий — от нажатия клавиш, перемещений и щелчков мыши до загрузки и выгрузки страниц. В отличие от серверных событий, аргументы клиентских событий удается получить только посредством метода *event* объекта *window*. Например, следующий обработчик события *mousemove* отображает в строке состояния браузера координаты указателя мыши:

VBScript

```
Sub document_onmousemove
    ' Отобразить координаты указателя мыши при перемещении его в
    ' окне браузера.
   window.status = window.event.clientX & ", " & window.event.clientY
End Sub
```

JScript

```
function window_onmousemove() {
   // Отобразить координаты указателя мыши при перемещении его в окне браузера.
   window.status = window.event.clientX + ", " + window.event.clientY;
}
```

Код обработчика, исполняемого на стороне клиента, способен получать и устанавливать значения серверных и HTML-элементов управления, размещенных на странице. Это означает, что можно принимать значения от пользователя и реагировать на них, не возвращая страницу на сервер. Вот пример HTML-кода, который определяет страницу, которая позволяет играть в простую игру, не возвращая страницу на сервер.

VBScript

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
   <head>
   <title>EventScript</title>
         <meta name="vs_defaultClientScript" content="VBScript">
         <meta name="vs_targetSchema"
           content="http://schemas.microsoft.com/intellisense/ie5">
         <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
         <meta name="ProgId" content="VisualStudio.HTML">
         <meta name="Originator" content="Microsoft Visual Studio.NET 7.0">
<script id="clientEventHandlersVBS" language="vbscript">

Sub butRunScript_onclick
    ' Отобразить сообщение в области вывода текста.
   sMessage = "I am thinking of a number between 0 and 9. "
   sMessage = sMessage & "Press a number key to take a guess."
   txtMessage.value = sMessage
    ' Инициализировать генератор случайных чисел.
   Randomize
    ' Получить случайное число и сохранить его в скрытом поле.
   hidNumber.value = Int(9 * Rnd)
```

```
End Sub

Sub document_onkeypress
   ' Если нажатая цифра совпадает с генерированным числом,
   ' уведомить пользователя об успехе. . .
   if (window.event.keyCode - 48) = CInt(hidNumber.value) then
         txtMessage.value = "You guessed it!"
   ' . . .в противном случае показать правильный ответ.
   else
         txtMessage.value = "You didn't guess it. It was: " & hidNumber.value
   end if
End Sub

</script>
   </head>
   <body MS_POSITIONING="FlowLayout">
   <P>
   <textarea id="txtMessage"></textarea>
   </p>
   <P>
   <INPUT type="button" id="butRunScript" value="Run Script"
     NAME="butRunScript">
   </P>
   <INPUT id="hidNumber" type="hidden" name="Hidden1">
</body>
</html>
```

JScript

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
   <HEAD>
         <META NAME="GENERATOR" Content="Microsoft Visual Studio 7.0">
         <TITLE></TITLE>
         <meta name="vs_defaultClientScript" content="JavaScript">
         <meta name="vs_targetSchema"
            content="http://schemas.microsoft.com/intellisense/ie5">
         <meta name="GENERATOR" content="Microsoft Visual Studio. NET 7.0">
         <meta name="ProgId" content="VisualStudio.HTML">
         <meta name="Originator" content="Microsoft Visual Studio. NET 7.0">
   <script id=clientEventHandlersJS language=javascript>
<!-

function window_onkeypress() {
   // Если нажатая цифра совпадает с генерированным числом,
   /,/ уведомить пользователя об успехе. . .
   if ((window.event.keyCode - 48) == parseInt(hidNumber.value))
         txtMessage.value - "You guessed it!";
```

```
    // ...в противном случае показать правильный ответ.
    else
        txtMessage.value = "You didn't guess it. It was: "
        + hidNumber.value;
}

function butRunScript_onclick() {
    // Отобразить сообщение в области вывода текста,
    sMessage = "I am thinking of a number between 0 and 9. "
      + "Press a number key to take a guess.";
    txtMessage.value = sMessage;
    // Получить случайное целое число.
    hidNumber.value = Math.round(9 * Math.random());
}

//->
</script>
</HEAD>
<BODY language=javascript onmousemove="return window_onmousemove()"
  onkeypress="return window_onkeypress()">
        <p>
        <textarea id="txtMessage" style="WIDTH: 200px; HEIGHT: 88px"
        rows="5" cols="22" NAME="txtMessage"></textarea>
        </p>
        <P>
        <INPUT type="button" id="butRunScript" value="Run Script"
          NAME="butRunScript" language=javascript onmouseover="return
          butRunScript_onmouseover()" onclick="return
          butRunScript_onclick()">
        </P>
        <P><INPUT id="hidNumber" type="hidden" name="Hidden1"></P>
    </BODY>
</HTML>
```

Во время выполнения этот код отобразит страницу с предложением угадать число от О до 9 (рис. 7-15).

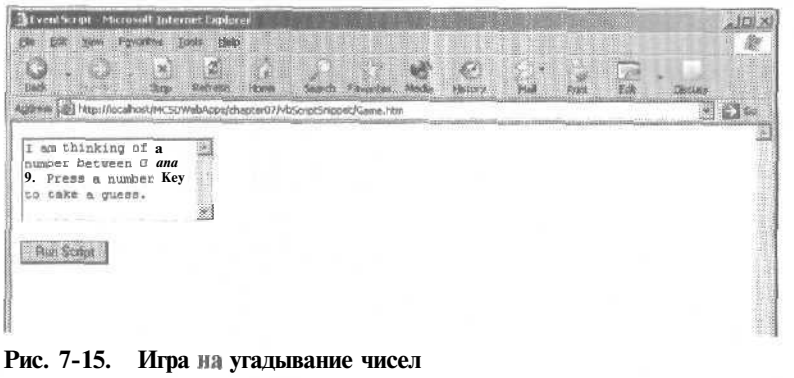


Рис. 7-15. Игра на угадывание чисел

Занятие 5. Использование Web-сервисов XML

Web-сервисы XML — это компоненты бизнес-логики, к которым можно получить доступ через Интернет и задействовать их в своем Web-приложении. В этом занятии рассказано, как искать, применять и ссылаться на Web-сервисы XML в Web-приложениях.

Изучив материал этого занятия, вы сможете:

- ✓ осуществлять поиск Web-сервисов XML по имени его провайдера;
- ✓ добавлять ссылки на Web-сервисы XML к проектам Visual Studio .NET;
- ✓ использовать Web-сервисы XML в Web-приложениях;
- ✓ применять Web-сервисы XML из сценариев на стороне клиента.

Продолжительность занятия — около 10 минут.

Поиск Web-сервисов XML

Web-сервисы XML публикуют в Web с помощью реестра UDDI. В настоящее время для поиска Web-сервисов XML доступны два узла UDDI, совместно поддерживаемых Microsoft и IBM. На этих узлах различные компании регистрируют свои Web-сервисы, чтобы клиенты (в том числе и вы) смогли их найти.

Способов поиска Web-сервисов XML в Интернете великое множество, но проще всего сделать это при помощи стартовой страницы Visual Studio .NET (рис. 7-16).

Раздел «XML Web services» стартовой страницы Visual Studio .NET позволяет искать сервисы по категории (сервисы даты и времени, финансовые, вычислительные или сервисы прогноза погоды). Переключатель Search In, расположенный над полем Category, задает поиск среди рабочих Web-сервисов XML (предполагается, что они прошли отладку и тестирование) или сервисов, которые все еще находятся на стадии разработки.

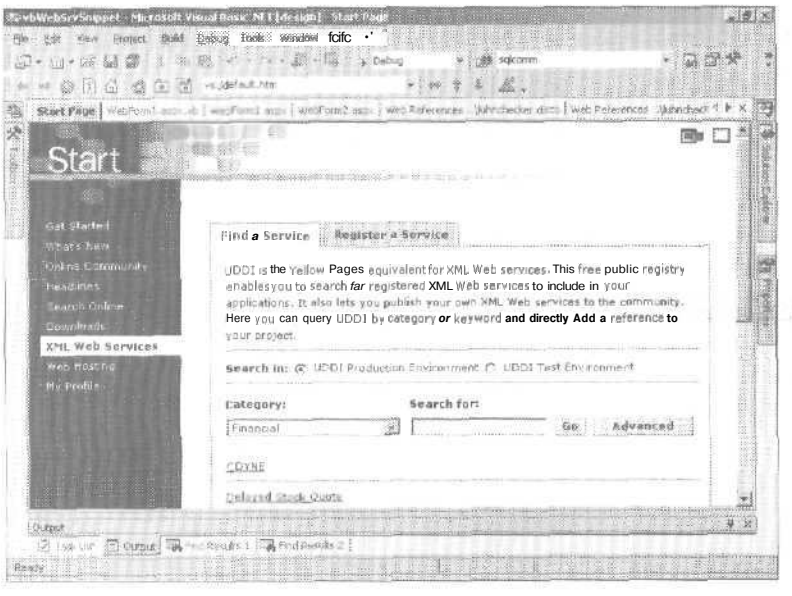


Рис. 7-16. Поиск Web-сервисов XML

Использование Web-сервисов XML

Применение Web-сервисов XML во многом напоминает использование компонентов .NET или COM: вы определяете ссылку на класс, создаете экземпляр этого класса и вызываете свойства и методы созданного в коде. Существует несколько способов определения ссылок на Web-сервисы XML в Visual Studio .NET. Наверное, самый простой из них — воспользоваться стартовой страницей Visual Studio .NET.

Чтобы создать ссылку на Web-сервис XML при помощи стартовой страницы Visual Studio .NET, выполните следующие действия.

1. Найдите нужный Web-сервис XML, щелкнув ссылку XML Web services на стартовой странице Visual Studio .NET.
2. Щелкните гиперссылку Add As Web Reference To Your Current Project, расположенную под описанием Web-сервиса XML — Visual Studio добавит соответствующую Web-ссылку к разделу Web References в Solution Explorer (рис. 7-17).

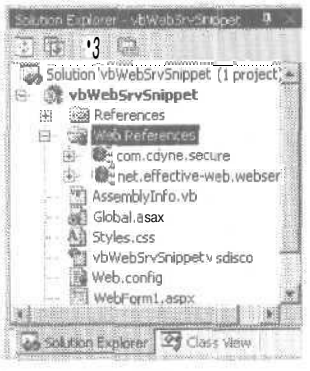


Рис. 7-17. Web-ссылки

Чтобы использовать созданные ссылки в коде, выполните следующие действия.

1. Создайте новый объект класса Web-сервиса XML, имя этого класса отобразится в папке Web References в окне Solution Explorer.
2. Задействуйте в коде свойства и методы объекта, представляющего Web-сервис XML. Классы Web-сервисов XML поддерживают функцию autocomplete и возможности Object Browser, поэтому использование их свойств и методов не отличается от использования свойств и методов любых других объектов.

Примечание Работа следующего примера зависит от доступности Web-сайта сторонней компании. Этот сайт был доступен на момент написания этой книги, но мы не можем гарантировать, что так будет и впредь.

Ниже показан пример программы, использующей Web-сервис CDYNE Credit Card Checker для проверки номера кредитной карты, введенного в текстовом поле:

Visual Basic .NET

```
Private Sub butCheck_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butCheck.Click
        Dim wsCheck As New com.cdyne.secure.LUHNChecker()
        Din wsValid As com.cdyne.secure.ReturnIndicator
        wsValid = wsCheck.CheckCC(txtNumber.Text)
```

```
        lblMsg.Text = wsValid.CardType & " " & wsValid.CardValid.ToString()
End Sub
```

Visual C#

```
private void butCheck_Click(object sender, System.EventArgs e)
{
   com.cdyne.secure.LUHNChecker wsCheck = new con.cdyne.secure.LUHNChecker();
   com.cdyne.secure.ReturnIndicator wsValid;
   wsValid = wsCheck.CheckCC(txtNumber.Text);
   lblMsg.Text - wsValid.CardType + " " + wsValid.CardValid.ToString();
}
```

Использование Web-сервисов XML в клиентских сценариях

В некоторых случаях имеет смысл вызвать Web-сервисы XML из клиентских сценариев, а не из серверного кода. Возможно, ждать отклика Web-сервиса XML придется достаточно долго, поэтому бессмысленно заставлять сервер ждать отклика сервиса, ведь сервер в любом случае переласт его клиенту.

Чтобы задействовать Web-сервис XML в клиентском сценарии, выполните следующие действия.

1. Создайте класс стиля для атрибута WebService (Webservice.htc).
2. Добавьте элемент HTML к странице, использующей класс, созданный на 1 этапе.
3. Напишите сценарий, инициализирующий и вызывающий методы Web-сервиса XML при помощи динамического элемента стиля, описывающего Web-сервис XML.

Следующий HTML-код создает класс стиля *webservice*, *а* также инициализирует и вызывает Web-сервис для отображения ежедневных котировок в ответ на щелчок кнопки Show Quote:

VBScript

```
<HTML>
   <HEAD>
         <title>WebForm2</title>
         <style>
               .webservice { BEHAVIOR:url(webservice.htc) }
         </style>
         <script language="VBScript">
Dim iCallID

Sub init(control, wsAddress, name)
   control.useService wsAddress, name
End Sub

Function getResult()
    if window.event.result.error And (iCallID = window.event.result.id) Then
      Dim xfaultcode, xfaultstring, xfaultsoap
      xfaultcode - window.event.result.errorDetail.code
```

```
        xfaultstring = window.event.result.errorDetail.string
        xfaultsoap = window.event.result.errorDetail.raw
        ' Отобразить сведения об ошибке.
        alert("Error " & xfaultcode & " | " & xfaultstring & " | "
& xfaultsoap)
    Else
        getResult = window.event.result.value
    End If
End Function

Sub getQuote()
    ' Инициализировать выбранный элемент управления при помощи Web-сервиса.
    init ws, "http://webservice.effective-web.net/globalself/
globalselfDailyThought.WSDL", "DailyQuote"
    ' Вызвать метод Web-сервиса.
    iCallID = ws.DailyQuote.callService("getTodaysQuote")
    ' Результат будет отображен в элементе div с помощью onresult.
End Sub
        </script>
    </HEAD>
    <body>
        <form id="WebForm2" method="post" runat="server">
            <h2>Using Web Services from Client-Side Code</h2>
            <div id="ws" class="webservice"
            onresult="ws.innerText = getResult()"></div>
            <br>
            <input type="button" onclick="getQuote()"
            value="Get Quote">
        </form>
        </DIV>
    </body>
</HTML>
```

JScript

```
<HTML>
    <HEAD>
        <title>WebForm2</title>
        <style>
            .webservice { BEHAVIOR:url(webservice.htc) }
        </style>
        <script language="JScript">
var iCallID;

function init(control, wsAddress, name)
{
    control.useService(wsAddress, name);
```

```
}

function getResult()
{
    if((event.result.error)&&(iCallID==event.result.id))
   {
      var xfaultcode - event.result.errorDetail.code;
      var xfaultstring = event.result.errorDetail.string;
      var xfaultsoap - event.result.errorDetail.raw;
      // Отобразить сведения об ошибке,
      alert("Error " + xfautlcode + "     + xfaultstring + " | "
      + xfaultSoap);
   }
   else
   {
      return event.result.value;
   }
}

function getQuote()
{
   // Инициализировать выбранный элемент управления при помощи
   // Web-сервиса.
   init(ws, "http://webservice.effective-web.net/globalself/
globalselfGlobalSelf.WSDL", "DailyQuote");
   // Вызвать метод Web-сервиса.
   iCallID - ws.DailyQuote.callService("getTodaysQuote");
   // Результат будет отображен в элементе div с помощью
   // onresult.
}
        </script>
   </HEAD>
   <body>
        <form id="WebForm2" method="post" runat="server">
             <h2>Using Web Services from Client-Side Code</h2>
             <div id="ws" class="webservice"
             onresult="ws.innerText - getResult();"></div>
             <br>
             <input type="button" onclick="getQuote()"
             value="Get Quote">
        </form>
        </DIV>
   </body>
</HTML>
```

Чтобы получить **дополнительные сведения об использовании Web-сервисов** XML **из** клиентских сценариев, выполните **поиск в справочной системе Visual Studio,** задав как критерий поиска строку «WebService Behavior».

Резюме

- Файлы cookie позволяют незаметно сохранять сведения о пользователях. Можно хранить на компьютере пользователя все сведения о нем либо записывать в cookie только уникальный идентификатор, необходимый для извлечения сведений о пользователе из набора данных, хранящегося на сервере.
- Прежде чем пытаться записать cookie на клиентском компьютере, проверьте, принимает ли его браузер файлы cookie. Это делается при помощи свойства *Cookies* объекта *Browser*.
- Свойство *Cookies* объекта *Request* служит для извлечения файлов cookie с клиентского компьютера; набор *Cookies* объекта *Response* позволяет создавать и модифицировать файлы cookie на компьютере пользователя.
- Чтобы удалить файл cookie с компьютера пользователя, установите значение свойства *Expires* объекта *Cookie* в *Now*.
- Чтобы создать сообщение, которое будет отправлено с помощью клиентской почтовой системы, используйте протокол *mailto:* в составе гиперссылки.
- Для создания и отправки сообщений с помощью серверной почтовой системы применяются классы *MailMessage* и *SmtpMail*.
- С помощью фреймов удается отображать несколько страниц в одном окне браузера, при этом отображение и прокрутка содержимого в каждом из фреймов управляются независимо.
- Прежде чем отобразить страницу с фреймами, выясните, поддерживает ли их браузер, воспользовавшись свойством *Frames* объекта *Browser*.
- Гиперссылка с атрибутом *target*, размещенная в одном фрейме, позволяет отображать в нем содержимое другого фрейма. Для управления фреймами требуются только гиперссылки и клиентские сценарии.
- Клиентские сценарии позволяют управлять окном браузера, немедленно реагировать на события, не вызывающие возврат страницы на сервер, а также выполнять другие действия, которые нельзя реализовать средствами серверного кода.
- При помощи свойства *VBScript* или *JavaScript* объекта *Browser* удается проверить, может ли браузер исполнять сценарии. Чтобы узнать, разрешают ли параметры безопасности пользовательского браузера исполнение сценариев, попробуйте исполнить встроенный сценарий, направляющий пользователя на другую страницу.
- Чтобы реагировать на события, генерируемые на клиентской (а не на серверной) стороне, добавьте к HTML-коду страницы обработчик этого события в виде сценария.

Практикум 7. Создание приложения Advanced Features

Цель этого практикума — создать приложение Advanced Features, которое будет отображать несколько страниц с помощью фреймов; оно позволит выбирать цвет фона для страниц, создавать и отправлять почту с сервера, а также играть в простую игру. На примере этого приложения мы хотим продемонстрировать следующие ключевые возможности;

- проверку пользовательского браузера на предмет наличия в нем поддержки фреймов, файлов cookie и сценариев;
- запись и извлечение файлов cookie с пользовательского компьютера — необходимо для установки цвета фона страниц приложения;
- отправку почты с сервера с помощью объектов *MailMessage* и *SmtpMail*;
- **отображение страниц в фреймах** и управление содержимым этих **фреймов со страницы** Contents.

Готовое приложение Advanced Features выглядит, как на рис. 7-18.

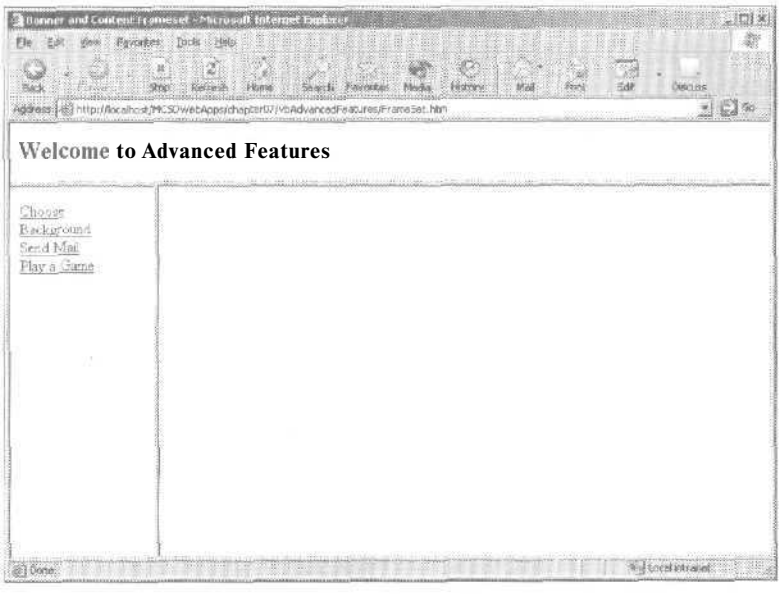


Рис. 7-18. Готовое приложение Advanced Features

Продолжительность практикума — около 45 минут.

Упражнение 1. Проверка наличия поддержки дополнительных возможностей

Сейчас вы создадите для приложения страницу по умолчанию, которая проверяет, включена ли в пользовательском браузере поддержка файлов cookie, фреймов и сценариев. Если какой-либо из этих параметров окажется недоступным, страница отобразит соответствующее сообщение.

Страница по умолчанию (default page) отображается IIS, когда пользователь переходит в каталог Web-приложения, не указав определенную страницу для просмотра. Страницами по умолчанию в IIS считаются страницы с именами Default.htm и Default.aspx, если только для приложения не были назначены другие страницы по умолчанию с помощью IIS.

▶ **Создание страницы по умолчанию, проверяющей наличие в браузере поддержки дополнительных возможностей**

1. Добавьте к приложению Web-форму с именем Default.aspx и сделайте ее начальной страницей приложения.
2. Добавьте к созданной Web-форме следующий обработчик события *Page_Load*:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles MyBase.Load
    ' Создать флаг, следящий за отключенными дополнительными
    ' возможностями.
    Dim bProceed As Boolean = True
    ' Проверить, принимает ли браузер файлы cookie.
    If Request.Browser.Cookies = False Then
        Response.Write("Your browser does not accept cookies.<br>" & _
         "Change your browser's security settings allow cookies." & _
         "<br><br>")
        bProceed = False
    End If
    ' Проверить, поддерживает ли браузер сценарии.
    If Request.Browser.JavaScript = False Then
        Response.Write("Your browser does not support scripts.<br>" & _
        "Update your browser software to the latest version.<br><br>")
        bProceed = False
    End If
    ' Проверить, поддерживает ли браузер VBScript.
    If Request.Browser.VBScript = False Then
        Response.Write("Your browser does not support VBScript.<br>" & _
          "Install the latest version of Internet Explorer.<br><br>")
        bProceed = False
    End If
    ' Если предыдущие проверки браузера прошли успешно,
    ' проверить поддержку сценариев, исполнив сценарий.
    If bProceed Then
        ' Если поддержка сценариев отключена, игнорировать
        ' следующую строку,
        ' в противном случае отобразить набор фреймов.
        Response.Write("<script>window.navigate('frameset.htm')
</script>")
        Response.Write("Scripting is disabled due to your browser " & _
         "security settings.<br> Change your browser's security " & _
         "settings allow scripting.<br><br>")
```

```
        End If
    End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
      // Создать флаг, следящий за отключенными дополнительными
      // возможностями.
      bool bProceed = true;
      // Проверить, принимает ли браузер файлы cookie.
      if (!Request.Browser.Cookies)
      {
           Response.Write("Your browser does not accept cookies.<br>" +
       "Change your browser's security settings allow cookies. "+
            "<br><br>");
           bProceed = false;
      }
      // Проверить, поддерживает ли браузер сценарии.
      if (!Request.Browser.JavaScript)
      {
            Response.Write("Your browser does not support scripts.<br>"+
             "Update your browser software to the latest " +
             "version.<br><br>");
            bProceed = false;
      }
      // Проверить, поддерживает ли браузер VBScript.
      if (!Request.Browser.VBScript)
      {
            Response.Write("Your browser does not support VBScript.<br>"+
            "Install the latest version of Internet Explorer.<br><br>");
            bProceed = false;
      }
      // Если предыдущие проверки браузера прошли успешно,
      // проверить поддержку сценариев, исполнив сценарий.
      if (bProceed)
      {
            // Если поддержка сценариев отключена, игнорировать
            // следующую строку,
            // в противном случае отобразить набор фреймов,
            Response.Write("<script>window.navigate('frameset.htm')"+
             "</script>");
            Response.Write("Scripting is disabled due to your browser " +
              "security settings.<br> Change your browser's security " +
             "settings allow scripting.<br><br>");
      }
}
```

3. Добавьте к Web-форме следующий HTML-код:

```
<p>This Web application uses features that are either not available in
your browser or are disabled through your browser's security settings.
You cannot run this application until you update your browser as described
above. Then click <a href="default.aspx">here</a> to try again.</p>
<a href="default.aspx">here</a>.
```

Во время загрузки страница по умолчанию, созданная в этом упражнении, проверяет, поддерживает ли браузер каждую из дополнительных возможностей. Если хоть одна из них не поддерживается, на странице появляется предупреждение (рис. 7-19).

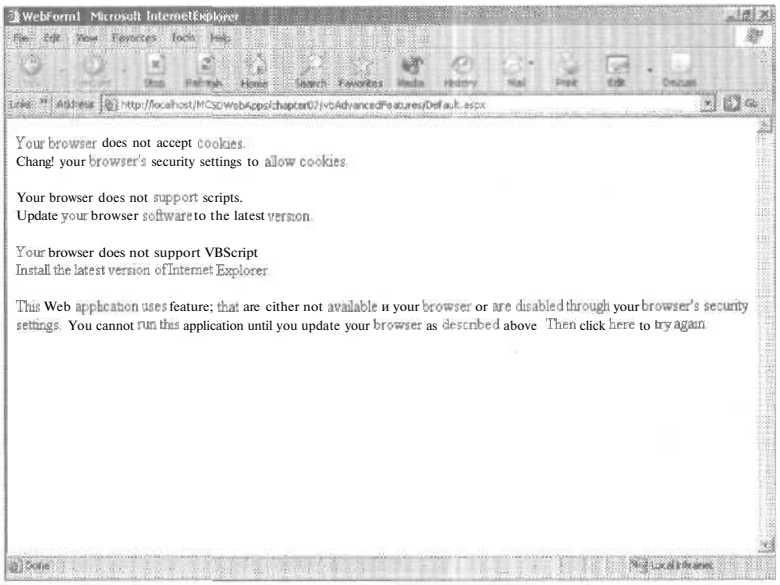


Рис. 7-19. Предупреждение

Если все проверки закончатся успешно, пользователь не увидит страницу по умолчанию, поскольку будет отображен набор фреймов, отображающих другие страницы приложения.

Упражнение 2. Сохранение сведений о пользователе



Сейчас вы создадите Web-форму для выбора цвета фона. После завершения сеанса форма «запоминает» цвет фона, выбранный пользователем, записывая эти данные в файл cookie на компьютере пользователя. Готовая Web-форма показана на рис. 7-20.

► Создание Web-формы Background.aspx

1. Создайте новую Web-форму и назовите ее Background.aspx.
2. Поместите на Web-форму серверный элемент управления DropDownList, элементы списка которого задают различные цвета фона. Проще всего для этого использовать режим HTML (а не Design), поскольку в нем удается быстро создавать элементы списка путем копирования-вставки соответствующего HTML-кода. Вот HTML-код, определяющий DropDownList и элементы его списка:

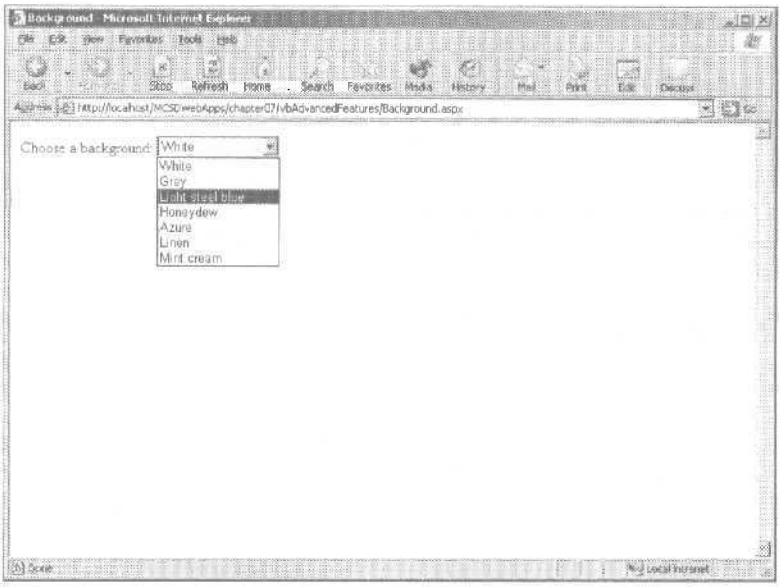


Рис. 7-20. **Web-форма Background.aspx**

```
<asp:dropdownlist id="drpBackground" AutoPostBack="True" Runat="server"
Width="128px">
   <asp:ListItem Value="White">White</asp:ListItem>
   <asp:ListItem Value="Silver">Grey</asp:ListItem>
   <asp:ListItem Value="LightSteelBlue">Light steel blue
   </asp:ListItem>
   <asp:ListItem Value="Honeydew">Honeydew</asp:ListItem>
   <asp:ListItem Value="Azure">Azure</asp:ListItem>
   <asp:ListItem Value="Linen">Linen</asp:ListItem>
   <asp:ListItem Value="MintCream">Mint cream</asp:ListItem>
</asp:dropdownlist>
```

3. Измените элемент *<body>* Web-формы так, чтобы он задавал цвет фона с помощью значений элементов списка DropDownList, используя привязку данных. Вот HTML-код модифицированного элемента *<body>*:

```
<body bgColor="<%# drpBackground.SelectedItem.Value %>">
```

4. Добавьте к обработчику события *Page_Load* код, проверяющий наличие файла cookie и создающий его, если он не существует. Если cookie существует, этот код задаст цвет фона на основе хранящихся в нем данных. Обработчик события *Page_Load* также использует привязку данных, чтобы обновить цвета фона.

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Если страница загружается впервые...
    If IsPostBack = False Then
        ' ...проверить, существует ли файл cookie, и если
```

```
        ' нет, создать его.
        If Request.Cookies("BackgroundColor") Is Nothing Then
            ' Создать файл cookie, определяющий цвет фона по
            'умолчанию.
            Dim cookBackground As_
            New HttpCookie("BackgroundColor", "0")
            ' Установить срок действия файла cookie в одни
            ' сутки.
            cookBackground.Expires = DateTime.Now.AddDays(1)
            ' Добавить cookie к объекту response.
            Response.Cookies.Add(cookBackground)
        Else
            ' Получить из файла cookie цвет фона, выбранный
            ' пользователем.
            drpBackground.SelectedIndex = _
              CInt(Request.Cookies("BackgroundColor").Value)
        End If
    End If
    ' Обновить связанное поле, чтобы задать цвета фона.
    Page.DataBind()
End Sub
```

Visual C#

```
private void Page_Load(object sender. System.EventArgs e)
{
    // Если страница загружается впервые...
    if (!IsPostBack)
    {
        // ...проверить, существует ли файл cookie, и
        // если нет. создать его.
        if (Request.Cookies["BackgroundColor"] == null)
        {
        // Создать файл cookie, определяющий цвет
        // фона по умолчанию.
            HttpCookie cookBackground - new
            HttpCookie("BackgroundColor", "0");
        // Установить срок действия файла cookie в
        // одни сутки.
            cookBackground.Expires - DateTime.Now.AddDays(1);
            // Добавить cookie к объекту response.
            Response.Cookies.Add(cookBackground);
        }
        else
        {
            // Получить из файла cookie цвет фона,
            // выбранный пользователем.
```

```
                drpBackground.SelectedIndex =
                 Convert.ToInt16(Request.Cookies["BackgroundColor"]
                 .Value);
            }
        }
        // Обновить связанное поле, чтобы задать цвета фона.
        Page.DataBind();
    }
```

5. Добавьте следующий код; он будет обновлять файл cookie через объект *Response* при выборе пользователем цвета фона из списка DropDownList:

Visual Basic .NET

```
Private Sub drpBackground_SelectedIndexChanged(ByVal sender As _
  System.Object, ByVal e As System.EventArgs) Handles _
  drpBackground.SelectedIndexChanged
    ' Записать информацию о выбранном цвете в файл cookie.
    Dim cookBackground As New HttpCookie("BackgroundColor")
    ' Задать значение файла cookie.
    cookBackground.Value - drpBackground.SelectedIndex
    ' Установить строк действия файла cookie.
    cookBackground.Expires = DateTime.Now.AddDays(1)
    ' Добавить cookie к объекту Response.
    Response.Cookies.Add(cookBackground)
End Sub
```

Visual C#

```
private void drpBackground_SelectedIndexChanged(object sender,
  System.EventArgs e)
{
      // Записать информацию о выбранном цвете в файл cookie.
      HttpCookie cookBackground - new HttpCookie("BackgroundColor");
      /7 Задать значение файла cookie.
      cookBackground.Value = drpBackground.SelectedIndex.ToString();
      // Установить строк действия файла cookie.
      cookBackground.Expires = DateTime.Now.AddDays(1);
      // Добавить cookie к объекту Response.
      Response.Cookies.Add(cookBackground);

}
```

Упражнение 3. Создание Web-формы Mail



Сейчас вы сконструируете Web-форму для создания и отправки почтовых сообщений с сервера. Законченная Web-форма Mail выглядит, как показано на рис. 7-21.

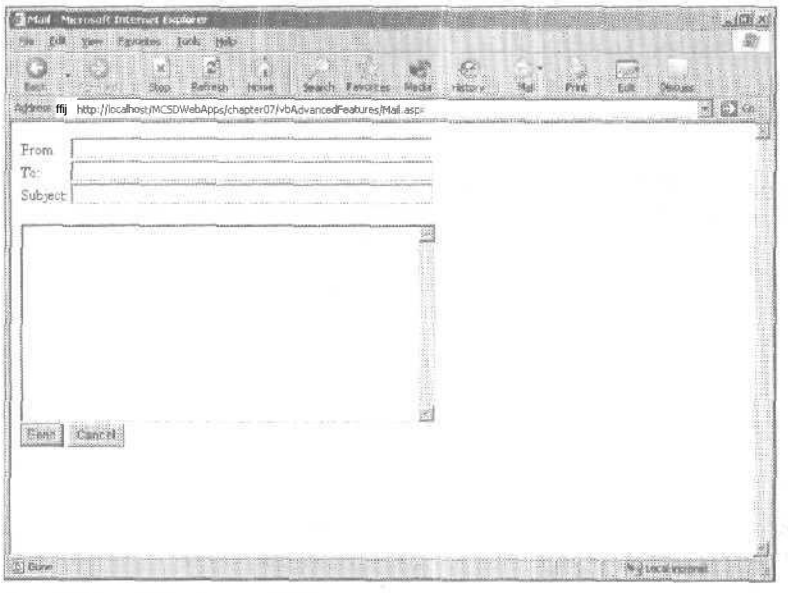


Рис. 7-21. Web-форма Mail.aspx

▶ **Создание Web-формы Mail**

1. Создайте новую Web-форму и назовите ее Mail.aspx.
2. Добавьте к Web-форме текст и серверные элементы управления, показанные в следующем HTML-коде:

```
<P>
From:   
<asp:TextBox ID="txtFrom" Runat="server" Width="376px">
</asp:TextBox>
<asp:RequiredFieldValidator ID="vldtxtFrom" ControlToValidate="txtFrom"
Runat="server" ErrorMessage="Required field.">
</asp:RequiredFieldValidator>
<br>
To:       
<asp:TextBox ID="txtTo" Runat="server" Width="377px">
</asp:TextBox>
<asp:RequiredFieldValidator ID="vldTo" ControlToValidate="txtTo"
Runat="server" ErrorMessage="Required field.">
</asp:RequiredFieldValidator>
<br>
Subject:
<asp:TextBox ID="txtSubject" Runat="server" Width="376px">
</asp:TextBox>
</P>
<asp:TextBox ID="txtMessage" Runat="server" TextMode="MultiLine"
Width="432px" Height="208px"></ asp:TextBox>
<br>
```

```
<asp:Button ID="butSend" Runat="server" Text="Send"></asp:Button>
<input type="reset" id="butReset" value="Cancel">
<br>
<asp:literal id="litStatus" runat="server"></asp:literal>
```

Элементы управления RequiredFieldValidator необходимы здесь для проверки значений, присваиваемых свойствам *From* и *To* объекта *MailMessage*, которые должны быть определены до отправки сообщения.

3. Чтобы применять сокращенные имена в ссылках на члены пространства имен *System. Web. Mail*, поместите в начало модуля Web-формы следующие операторы:

Visual Basic .NET

```
Imports System.Web.Mail
```

Visual C#

```
using System.Web.Mail;
```

4. Добавьте к обработчику события *butSend_Click* код для создания объекта *MailMessage* и отправки сообщения с сервера:

Visual Basic .NET

```
Private Sub butSend_Click(ByVal sender As System.Object, +
  ByVal e As System.EventArgs) Handles butSend.Click
    ' Создать сообщение...
    Dim msgMail As New MailMessage()
    ' ...установить его свойства...
    msgMail.From = txtFrom.Text
    msgMail.To = txtTo.Text
    msgMail.Subject = txtSubject.Text
    msgMail.Body = txtMessage.Text
    ' ...и отправить сообщение.
    SmtpMail.Send(msgMail)
    ' Очистить поля To, Subject и Message.
    txtTo.Text - ""
    txtSubject.Text = ""
    txtMessage.Text = ""
    ' Уведомить об успешной отправке.
    litStatus.Text = "<p>Message sent.</p>"
End Sub
```

Visual C#

```
private void butSend_Click(object sender, System.EventArgs e)
{
      // Создать сообщение...
      MailMessage msgMail - new MailMessage0;
      // ...установить его свойства...
      msgMail.From - txtFrom.Text;
      msgMail.To = txtTo.Text;
```

```
    msgMail.Subject = txtSubject.Text;
    msgMail.Body = txtMessage.Text;
    // ...и отправить сообщение.
    SmtpMail.Send(msgMail);
    // Очистить поля To. Subject и Message.
    txtTo.Text = "";
    txtSubject.Text = "";
    txtMessage.Text - "";
    // Уведомить об успешной отправке.
    litStatus.Text = "<p>Message sent.</p>";
}
```

5. Добавьте к обработчику события *Page_Load* код, который очищает текст уведомления, если пользователь отменил отправку сообщения после успешной отправки предыдущего сообщения:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Очистить текст уведомления,
    litStatus.Text = ""
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Очистить текст уведомления.
    litStatus.Text = "";
}
```

Упражнение 4. Создание пользовательского интерфейса на основе фреймов



Сейчас вы создадите набор фреймов для отображения оглавления, баннера и страниц приложения. Ссылки оглавления будут отображать выбранные страницы приложения в главном фрейме и обновлять текст фрейма баннера с помощью сценария. Готовый набор фреймов показан на рис. 7-22.

▶ Создание пользовательского интерфейса на основе фреймов

1. Создайте новую HTML-страницу и назовите ее Contents.htm.
2. Добавьте к странице Contents следующие гиперссылки:

```
<a href="Background.aspx" target="main"
onclick="ShowBanner('Choose Background')">Choose Background
</a>
<br>
<a href="Mail.aspx" target="main"
onclick="ShowBanner('Send Mail From Server')">Send Mail</a>
```

```
<br>
<a href="Game.htm" target="main" onclick="ShowBanner('Play Game')">Play a
Game</a>
<br>
```

Эти гиперссылки позволяют переходить между страницами приложения, отображая их в главном фрейме из набора фреймов, который вы создадите далее.

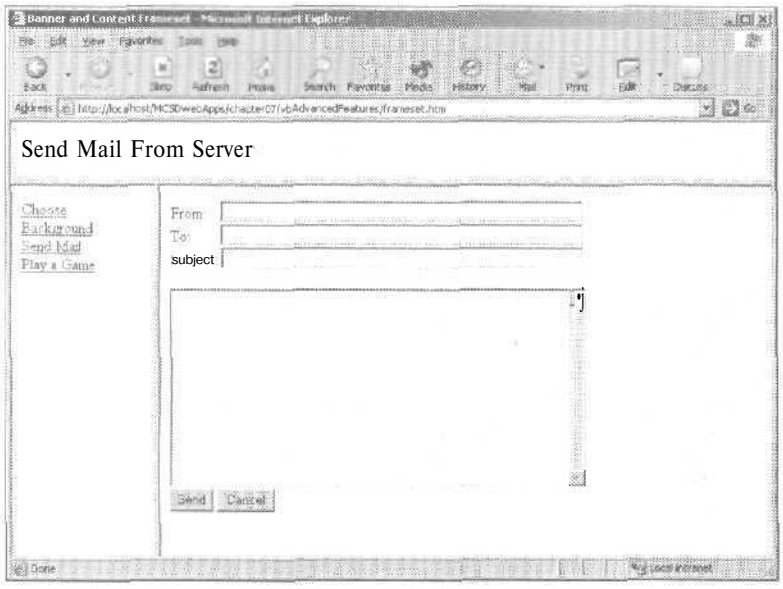


Рис. 7-22. **Вид готового набора фреймов во время выполнения**

3. Добавьте к странице Contents следующий сценарий:

VBScript

```
<script language="vbscript">
' Эта функция обновляет текст фрейма баннера.
Sub ShowBanner(Message)
    ' Записать текст во фрейм баннера.
    parent.frames("banner").document.write("<h2>" & Message & "</h2>")
    ' Закрыть документ, чтобы в следующий раз записать текст
    ' поверх текущего.
    parent.frames("banner").document.close
End Sub
</script>
```

JScript

```
<script language="javascript">
// Эта функция обновляет текст фрейма баннера.
function ShowBanner(Message)
{
```

```
    // Записать текст во фрейм баннера.
    parent.frames["banner"].document.write("<h2>" + Message +
    "</h2>");
    // Закрыть документ, чтобы в следующий раз
    // записать текст поверх текущего.
    parent.frames["banner"].document.close();
}
</script>
```

Сценарий ShowBanner отображает во фрейме баннера текст из гиперссылок, созданных на предыдущем этапе.

4. Создайте набор фреймов для отображения страниц проекта. Для этого из меню Project выберите команду Add New Item, затем из списка Templates выберите Frameset и присвойте новому файлу имя Frameset.htm. Щелкните Open — Visual Studio откроет диалоговое окно Select A Frameset Template,
5. Выберите для набора фреймов шаблон Banner And Content и шелкните OK — Visual Studio откроет пустой набор фреймов в окне Design.
6. Щелкните правой кнопкой крайний слева фрейм и выберите из контекстного меню команду Set Page For Frame — Visual Studio откроет диалоговое окно Select Page.
7. В диалоговом окне Select Page укажите файл Contents.htm и шелкните OK, чтобы назначить страницу Contents для отображения в этом фрейме.
8. Назначьте страницу с набором фреймов начальной страницей приложения. Для этого в окне Solution Explorer шелкните правой кнопкой файл Frameset.htm и выберите из контекстного меню команду Set As Start Page.

Упражнение 5. Дополнительные упражнения по применению сценариев и файлов cookie



Выполнив упражнения 1–4, вы научились применять на практике знания, полученные при изучении этой главы. До сих пор мы давали подробные указания, помогающие вам решать поставленные задачи, а теперь попробуйте сделать это самостоятельно. Ваша цель — создать недостающие компоненты приложения Advanced Features, для чего необходимо:

- добавить к странице Contents сценарий *Window_On Load*, отображающий приветственное сообщение во фрейме баннера при первой загрузке страницы;
- добавить к приложению Advanced Features Web-форму Guess A Number, созданную в конце занятия 4;
- создать для Web-формы Mail файл cookie, в который будет записываться вводимый пользователем адрес отправителя;
- назначить цвет фона для всех страниц, отображаемых в наборе фреймов, в соответствии с цветом, заданным в Web-форме Background Color.



Закончив, сравните свои результаты с приложением Advanced Features, которое хранится на прилагаемом к книге компакт-диске. Удачи!

Закрепление материала

Приведенные ниже вопросы помогут Вам лучше усвоить основные темы данной главы. Если Вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. Напишите HTML-код для гиперссылки, которая будет отправлять сообщение по щелчку пользователя.

2. Напишите код, записывающий на пользовательский компьютер файл cookie с именем пользователя (Rob Young) и текущей датой, который хранится на компьютере пользователя в течение 30 дней.

3. Почему нельзя открыть новое окно браузера из серверного кода? Как открыть страницу в новом окне из клиентского сценария?

4. Как отобразить страницу в одном фрейме, шелкнув гиперссылку в другом фрейме?

ГЛАВА 8

Обеспечение безопасности

Занятие 1. Аутентификация и авторизация пользователей 340

Занятие 2. Аутентификация Windows 345

Занятие 3. Аутентификация Forms 352

Занятие 4. Аутентификация Passport 363

Занятие 5. Защита данных при пересылке 368

В этой **главе**

Изучив эту главу, вы научитесь управлять доступом к Web-приложениям, применяя различные методики ASP.NET для идентификации пользователей и предоставления им разрешений на доступ к приложениям. Вы также научитесь зашишать пересылаемые через Интернет данные от доступа посторонних.

Данная глава не претендует на звание исчерпывающего руководства по защите сервера или сети — этот без преувеличения огромный раздел включает массу сведений как концептуального характера, так и относящихся к защите конкретных систем. Однако эту главу можно считать практическим руководством по реализации защиты Web-приложений средствами ASP.NET.

Прежде всего

Для изучения материалов этой главы вам необходимы:

- два компьютера, подключенных к сети на основе Windows. Для этого занятия подойдет простая сеть на основе рабочих групп (например, домашняя), но лучше использовать компьютеры, подключенные к домену;
- привилегии администратора на сервере сети.

Большинство задач из этой главы можно выполнить, даже не имея доступа к сети, а исполняя приложения локально, как в режиме Debug в Visual Studio, но в этом случае будет сложно протестировать аутентификацию пользователей Web-приложения.

Занятие 1. Аутентификация и авторизация пользователей

Аутентификацией (authentication) называется процесс идентификации пользователей. *Авторизация* (authorization) — это процесс предоставления доступа пользователям на основе их идентификационных данных. Аутентификация наряду с авторизацией представляют собой средства зашиты Web-приложений от несанкционированного доступа.

На этом занятии рассказывается, как **ASP.NET** работает с анонимными пользователями, кроме того, вы познакомитесь с различными способами идентификации и авторизации пользователей. Это пригодится вам при выборе стратегии аутентификации в зависимости от типа создаваемого приложения.

Изучив материал этого занятия, вы сможете:

- ✓ рассказать, как ASP.NET предоставляет доступ анонимным пользователям и как сведения о таких пользователях отображаются инструментами системы безопасности Windows;
- ✓ выбирать методику аутентификации в зависимости от потребностей вашего приложения;
- ✓ добавлять HTML-страницы к списку типов файлов, для запроса которых необходимо пройти аутентификацию.

Продолжительность занятия — около 10 минут.

Анонимный доступ

Большинство открытых Web-сайтов работают в режиме анонимного доступа. Эти сайты содержат общедоступную информацию, которую могут просматривать все желающие, поэтому они не проводят аутентификацию пользователей. Web-приложения ASP.NET предоставляют анонимны и доступ к серверным ресурсам посредством олицетворения. В данном контексте *олицетворение* (impersonation) — это назначение учетной записи неизвестному пользователю.

По умолчанию учетная запись для анонимного доступа имеет имя в виде IUSER_*имя_компьютера* и используется для управления доступом анонимных пользователей к серверным ресурсам.

Чтобы просмотреть или изменить привилегии учетной записи для анонимного доступа с помощью оснастки Computer Management (Управление компьютером), выполните следующие действия.

1. Войдите на сервер как администратор.
2. Выберите из меню Start (Пуск) пункт Administrative Tools\Computer Management (Администрирование\Управление компьютером), чтобы запустить консоль Computer Management (рис. 8-1).
3. Выберите в списке слева элемент Local Users And Groups (Локальные пользователи и группы), затем папку Users, чтобы открыть список авторизованных пользователей для этого компьютера.
4. В списке справа дважды щелкните левой кнопкой анонимную учетную запись с именем в форме IUSR_*имя_компьтера* ~ оснастка Computer Management откроет окно свойств учетной записи, показанное на рис. 8-2.

   Для просмотра групп, к которым принадлежит эта учетная запись, перейдите на вкладку Member Of (Членство в группах). По умолчанию анонимные пользователи являются членами группы Guests, обладающей ограниченными привилегиями.

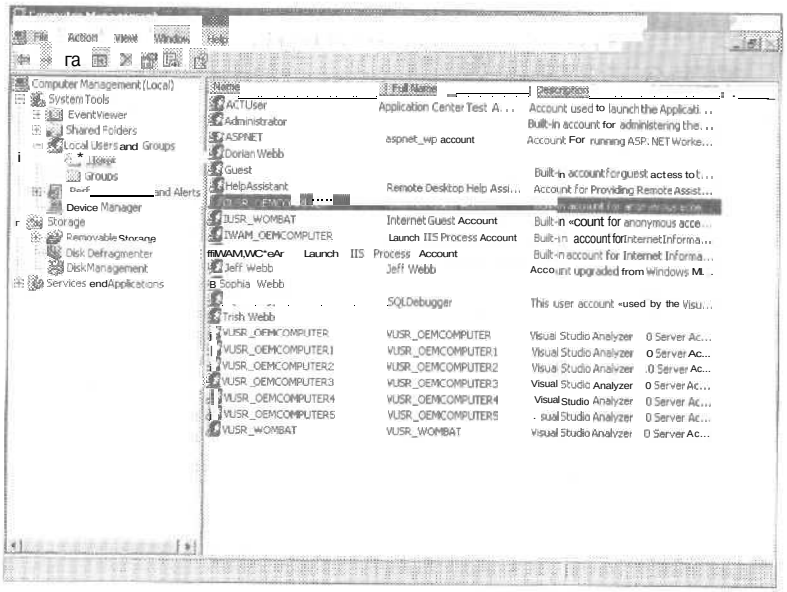


Рис. 8-1. Просмотр пользователей

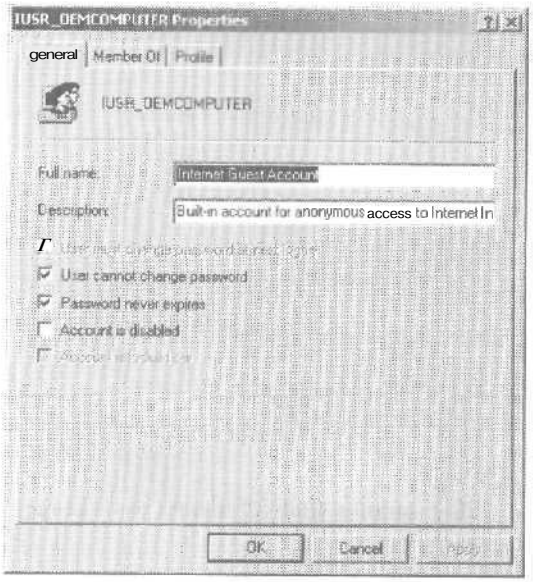


Рис. 8-2. Свойства учетной записи для анонимного доступа

По умолчанию ASP.NET исполняет Web-приложения под учетной записью ASPNET. Это означает, что при попытке приложения выполнить задачу, не предусмотренную привилегиями учетной записи ASPNET, генерируется исключение безопасности и приложение получает отказ в доступе. Кроме того, имя учетной записи заносится а журнал безопасности, который можно просмотреть в той же оснастке (рис. 8-3).

Права доступа для анонимных пользователей можно ограничивать, устанавливая разрешения Windows на доступ к файлам. Для более надежной защиты сервера следует ис-

пользовать файловую систему NT (NTFS), так как более ранние файловые системы, такие, как FAT и FAT32, не обеспечивают безопасность на файловом уровне. Подробнее о файловых разрешениях Windows рассказано в разделах справочной системы Windows, посвященных безопасности.

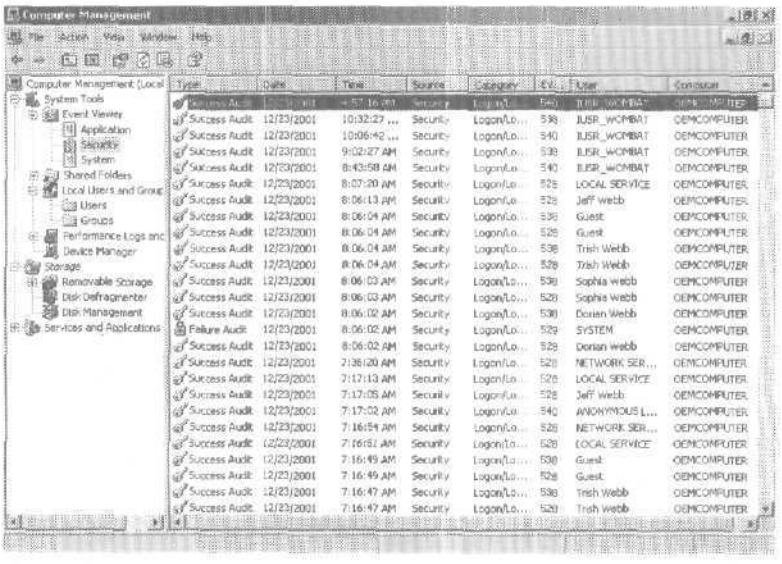


Рис. 8-3. Просмотр журнала событий системы безопасности в консоли Computer Management

Доступ аутентифицированных пользователей

Как говорилось ранее, анонимный доступ подходит для работы с открытой информацией, но если приложение содержит конфиденциальные сведения или работает с ними, например, размещает заказы, все пользователи должны проходить аутентификацию и авторизацию.

Существует три основных способа аутентификации и авторизации пользователей Web-приложений ASP.NET:

- **аутентификация Windows** — применяет для идентификации и авторизации список пользователей сервера. Доступ к серверным ресурсам предоставляется либо запрещается в зависимости от привилегий учетной записи пользователя. Работает аналогично обычным механизмам сетевой безопасности Windows;
- **аутентификация Forms** — направляет пользователя на входную Web-форму, где он вводит свое имя и пароль, после чего форма идентифицирует его с помощью списка пользователей или базы данных, поддерживаемой приложением;
- **аутентификация Passport** — направляет пользователей на сайт Microsoft, где они при помощи единого имени и пароля получают доступ к нескольким Web-сайтам. Зарегистрированный пользователь должен ввести имя и пароль службы Passport, которые приложение затем сверяет со списком пользователей службы Passport.

У каждого способа, включая анонимный доступ, есть свои преимущества и недостатки. В таблице 8-1 перечислены методы аутентификации, оптимальные для различных типов Web-приложений.

Таблица 8-1. Рекомендуемые методы аутентификации **пользователей Web-приложений**

Тип приложения	**Рекомендуемый метод аутентификации**	**Описание**
Открытые Web-приложения, работающие в Интернете	Анонимный доступ	Это обычный метод доступа к Web-сайтам, не требующий регистрации. Безопасность закрытых ресурсов обеспечивается с помощью файловых разрешений NTFS
Web-приложения, работающие в интрасети	Аутентификация средствами Windows	Аутентификация пользователей сети выполняется контроллером домена. Пользователи сети получают доступ к ресурсам Web-приложения согласно своим привилегиям на сервере
Закрытые корпоративные Web-приложения	Аутентификация средствами Windows	Корпоративные пользователи получают доступ к Web-приложению при помощи учетных данных пользователя корпоративной сети. Администрирование пользовательских учетных записей производится с помощью инструментов сетевой безопасности Windows
Коммерческие Web-приложения	Аутентификация с помощью входной формы	В приложениях, собирающих сведения для выставления счетов и отправки товаров, следует применять аутентификацию с помощью форм, которые заодно позволят получить и сохранить сведения о клиенте
Единый доступ к нескольким коммерческим Web-приложениям	Аутентификация с помощью службы Passport	Для аутентификации с помощью службы Passport пользователю достаточно однажды зарегистрироваться в центре выдачи паспортов. После этого идентификационные данные пользователя становятся доступными любому приложению, использующему Passport SDK. Сведения о клиенте хранятся в профиле службы Passport, а не в локальной базе данных

Аутентификация и авторизация пользователей с применением каждого из трех указанных методов более подробно описана в следующих разделах.

Аутентификация запросов HTML-страниц

Описанные методы аутентификации ASP.NET применяются для файлов, являющихся частью Web-приложения. К ним относятся Web-формы (.aspx-файлы), модули (.asax-файлы) и другие ресурсы, обрабатываемые исполняемыми файлами Web-приложения. HTML-страницы (.htm- или .html-файлы) не включаются в число этих файлов автоматически. Такие страницы обрабатываются IIS, а не ASP.NET. Если нужно аутентифицировать пользователей, обращающихся к HTML-страницам Web-приложения, средствами Windows либо с помощью форм или службы Passport, необходимо зарегистрировать эти файлы для обработки исполняемыми компонентами ASP.NET.

Чтобы зарегистрировать .htm- и .html-файлы для обработки исполняемыми файлами ASP.NET, откройте оснастку IIS и выполните следующие действия.

1. В оснастке IIS выберите папку с Web-приложением и выберите из меню Action команду Properties — IIS откроет диалоговое окно Properties.

2. Перейдите на вкладку Directory и щелкните кнопку Configuration, IIS откроет диалоговое окно Application Configuration, показанное на рис. 8-4.

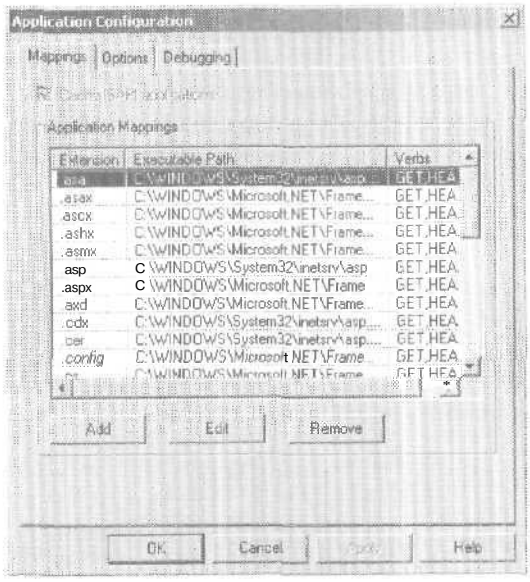


Рис. 8-4, Диалоговое окно Application Configuration

3. Щелкните Add — IIS откроет диалоговое окно Add/Edit Application Extension Mapping, показанное на рис. 8-5.

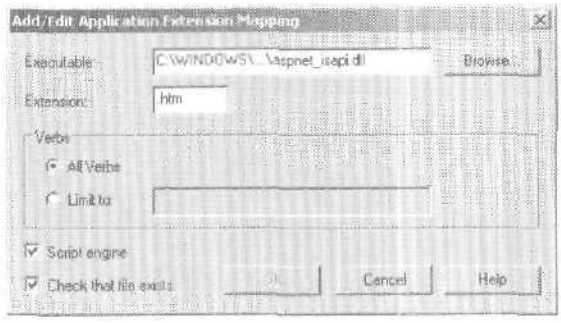


Рис. 8-5. Диалоговое окно Add/Edit Application Extension Mapping

4. Щелкните кнопку Browse и выберите файл aspnet_isapi.dll. Этот файл находится в каталоге Windows Microsoft .Net Framework, путь к нему выглядит примерно так: C:\Windows\Microsoft.NET\Framework\versionnumber\aspnet_isapi.dll.
5. Введите в текстовое поле File Extension **.htm** и щелкните OK.
6. Повторите пункты 3-5, чтобы зарегистрировать расширение .html. Закончив, щелкните OK, чтобы закрыть диалоговое окно US.

Занятие 2. Аутентификация Windows

Аутентификация Windows использует встроенные механизмы системы безопасности операционных систем Windows NT и Windows **XP.** Преимущество аутентификации средствами Windows заключается в возможности использования Web-приложением схемы безопасности, принятой в корпоративной сети, то есть применения одних и тех же имен, паролей и разрешений для получения доступа к ресурсам корпоративной сети и ресурсам Web-приложений.

Изучив материал этого занятия, вы сможете:

- ✓ включать использование аутентификации Windows в Web-приложении;
- ✓ разрешать или запрещать пользователю доступ к приложению в зависимости от его имени или принадлежности к роли;
- ✓ идентифицировать в приложении пользователей, зарегистрировавшихся в системе;
- ✓ применять другие формы аутентификации при помощи параметров **US.**

Продолжительность занятия — около 20 минут.

Включение аутентификации Windows

В новом Web-приложении аутентификация Windows используется по умолчанию.

Чтобы увидеть, как работает аутентификация Windows, выполните следующие действия.

1. Создайте новый проект Web-приложения. Если проект использует Visual Basic .NET, измените элемент, определяющий авторизацию следующим образом (см. строку, выделенную полужирным шрифтом в HTML-коде), а если Visual C# — то следующий элемент необходимо добавить целиком:

```
<authorization>
    <deny users="?"  />
</authorization>
```

2. Добавьте к коду начальной Web-формы проекта следующее HTML-определение таблицы:

```
<TABLE id="tblUser" >
    <TD>Authenticated</TD>
    <TD><span runat="server" id="spnAuthenticated"></span></TD>
    <TD>User name</TD>
    <TD><span runat="server" id="spnUserName"></span></TD>
    <TD>Authentication type</TD>
    <TDx$pan runat="server" id="spnAuthenticationType">
</span></TD>
</TABLE>
```

3. Переключите окно формы в режим Design и добавьте к объекту кода начальной Web-формы следующие строки;

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
```

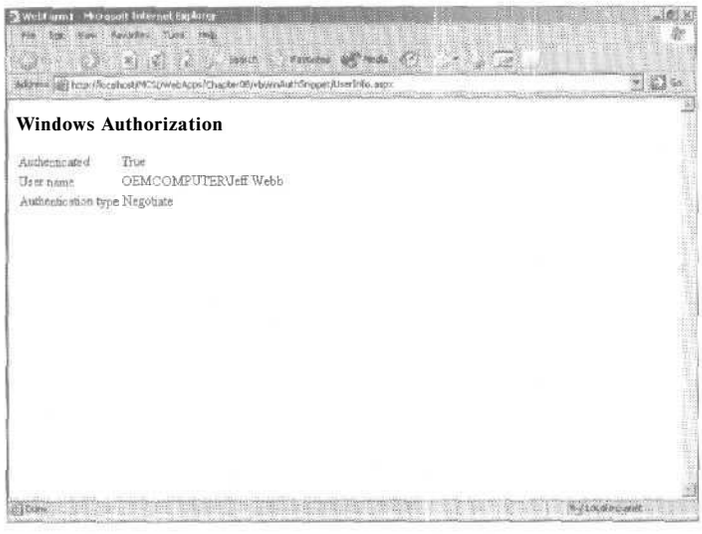
```
    spnAuthenticated.InnerText = User.Identity.IsAuthenticated
    spnUserName.InnerText = User.Identity.Name
    spnAuthenticationType.InnerText - User.Identity.AuthenticationType
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    spnAuthenticated.InnerText = User.Identity.IsAuthenticated.ToString();
    spnUserName.InnerText = User.Identity.Name;
    spnAuthenticationType.InnerText = User.Identity.AuthenticationType;
}
```

4. Запустите проект.

Если проект запущен локально, ASP.NET аутентифицирует с помощью имени, указанного при входе в Windows (рис. 8-6):



Рис, 8-6. Аутентификация средствами Windows, выполненная локально

При удаленном исполнении проекта (например, через Интернет) ASP.NET откроет в браузере диалоговое окно для ввода имени пользователя и пароля, показанное на рис. 8-7.

Если имя и пароль, введенные пользователем, совпадают с заданными в учетной записи, авторизованной в данном домене, ASP.NET идентифицирует пользователя и предоставляет ему доступ к приложению. Приложение исполняется с разрешениями, назначенными учетной записи этого пользователя.

Для авторизованного пользователя ASP.NET создает сертификат авторизации в виде файла cookie, хранящегося в течение сеанса этого пользователя. Сеанс пользователя завершается при закрытии браузера либо при наступлении тайм-аута сеанса.

Лучше всего аутентификация средствами Windows работает в сети, организованной в виде домена. В сетях, где вместо доменов задействованы рабочие группы, возможности применения сетевой безопасности ограничены. Идентификация и авторизация пользователей доменов выполняется контроллером домена, как показано на рис. 8-8.

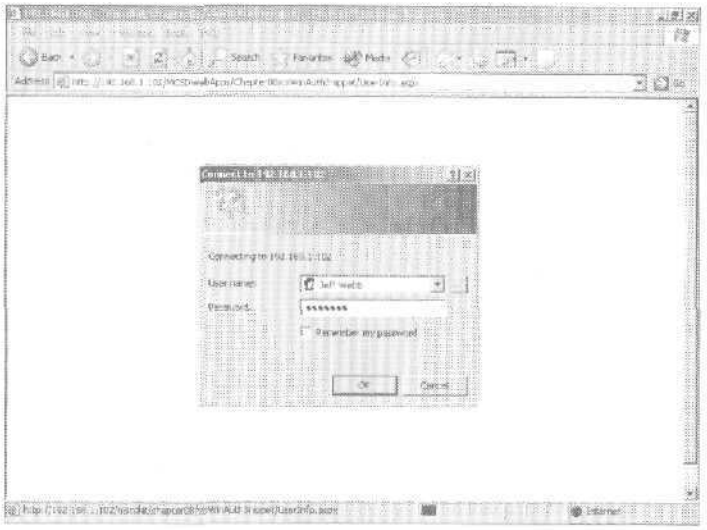


Рис. 8-7. Аутентификация средствами Windows на удаленном компьютере

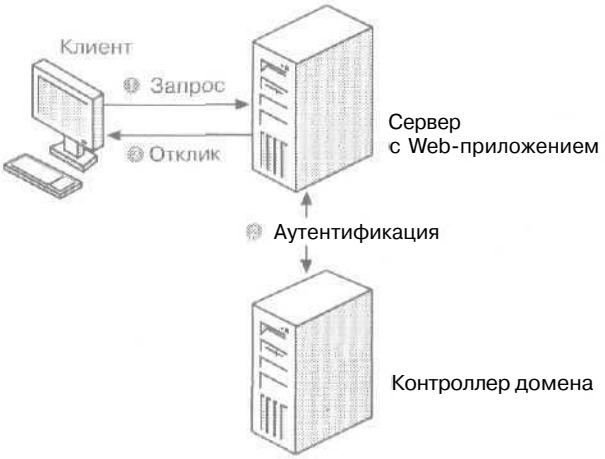


Рис. 8-8. Аутентификация в домене

Одно из ключевых преимуществ аутентификации средствами Windows состоит в том, что она освобождает пользователей, зарегистрировавшихся в сети, от повторной регистрации для доступа к Web-приложениям. Дополнительное преимущество — предоставляемая корпоративным пользователям возможность применять одни и те же сетевые имена и пароли для удаленного доступа к корпоративному Web-сайту из дома и в командировках.

Аутентификация средствами Windows также позволяет организовать дополнительную линию защиты, предоставляя или запрещая доступ отдельным пользователям или группам. Кроме того, аутентификация средствами Windows дублирует ряд функций IIS. Более подробно эти темы рассматриваются в следующих разделах.

Назначение прав доступа отдельным пользователям

Если приложение применяет аутентификацию средствами Windows, **ASP.NET** проверяет список авторизации в файле Web.config проекта, чтобы определить тех, кому разрешен доступ к приложению. Символы «*» и «?» имеют особое значение в списке авторизации: звездочка обозначает всех пользователей, а знак вопроса — всех неавторизированных пользователей. Например, если в файле Web.config задан следующий список авторизации, всем пользователям придется пройти аутентификацию, чтобы получить доступ к Web-приложению:

```
<authorization>
<!-- Доступ неавторизированным пользователям запрещен ->
      <deny users="?"  />
</authorization>
```

Чтобы определить ограниченный круг пользователей, которым вы хотите разрешить доступ, укажите их имена (через запятую) в элементе *<allow>*. Проверяя список авторизации, ASP.NET принимает первое найденное соответствие. Обязательно поместите в список авторизации элемент *<deny>*, чтобы запретить доступ остальным пользователям:

```
<authorization>
<!-- Разрешить доступ двум пользователям...->
<allow users="Deanna Meyer, Michael Emanuel" />
<!--...и запретить его всем остальным. ->
<deny users="*"  />
</authorization>
```

Этот список авторизации разрешает доступ к Web-приложению двум пользователям — Deanna Meyer и Michael Emanuel (у них также должны быть учетные записи в этом домене) и запрещает его всем остальным.

Применение авторизации, основанной на ролях

При использовании авторизации на основе ролей можно назначать разрешения целым группам пользователей, которые выделяются на основе их функций в организации. Роли соответствуют группам Windows NT и Windows XP. В Windows предусмотрено несколько встроенных групп, например Administrators, Users и Guests. Для просмотра, модификации или создания групп служит консоль Computer Management (рис. 8-9).

Чтобы разрешить или запретить доступ определенным группам, добавьте элемент *<roles>* к списку авторизации в файле Web.config Web-приложения. Например, следующий список авторизации разрешает доступ только пользователям из группы системных администраторов:

```
<authorization>
<!-- Разрешить доступ только членам группы Administrators... ->
      <allow roles="Administrators" />
<!- ...и запретить его всем остальным. ->
    <deny users="*"  />
</authorization>
```

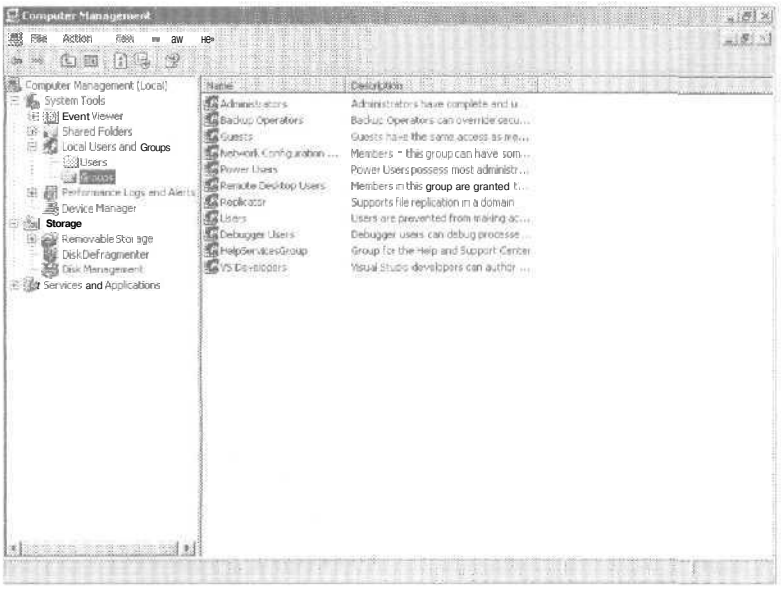


Рис. 8-9. Просмотр групп

Извлечение идентификационных данных пользователя

Приложения извлекают идентификационные данные пользователей, прошедших аутентификацию, с помощью свойства *Identity* объекта *User*. Как показано в следующем фрагменте кода, свойство *Identify* возвращает объект, содержащий имя пользователя и роль, к которой он принадлежит:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles MyBase.Load
     spnAuthenticated.InnerText - User.Identity.IsAuthenticated
     spnUserName.InnerText - User.Identity.Name
     spnAuthenticationType.InnerText - User.Identity.AuthenticationType
End Sub
```

Visual C#

```
private void Page_Load(object sender. System.EventArgs e)
\
    spnAuthenticated.InnerText -
      User.Identity.IsAuthenticated.ToString();
    spnUserName.InnerText - User.Identity.Name;
    spnAuthenticationType.InnerText - User.Identity.AuthenticationType;
}
```

Кроме того, объект *User* поддерживает метод *IsInRole*, позволяющий определить роль текущего пользователя, например, так:

Visual Basic .NET

```
If User.IsInRole("Administrators") Then
    ' Выполнить некоторые действия.
End If
```

Visual C#

```
if(User.IsInRole("Administrators"))
    // Выполнить некоторые действия.
```

Эти методы и свойства вместе с событием *AuthorizeRequest*, определенным в модуле Global, позволяют задать метод авторизации пользователя. Например, при помощи события *Authorize Request* сверяется имя пользователя с именем, не заданным в списке файла Web.config.

Параметры IIS и аутентификация средствами Windows

Параметры авторизации, задаваемые в файле Web.config, дублируют ряд параметров IIS. Если параметры авторизации заданы как файлом Web.config, так и параметрами IIS, первыми проверяются именно они, а лишь затем — файл Web.config. В результате, как правило, используются параметры, задающие минимальные права доступа.

Для просмотра параметров авторизации, заданных в IIS, выполните следующие действия.

1. В оснастке IIS щелкните правой кнопкой мыши папку Web-приложения и выберите из контекстного меню элемент Properties — IIS откроет диалоговое окно свойств папки. Перейдите на вкладку Directory Security и в группе Anonymous Access And Authentication Control щелкните Edit. IIS откроет диалоговое окно Authentication Methods, показанное на рис. 8-10.

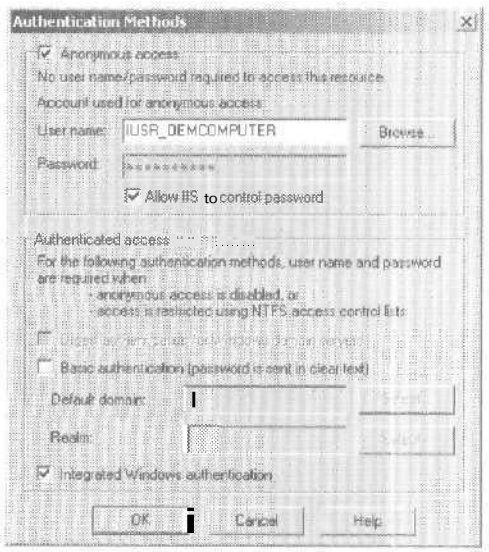


Рис. 8-10, Настройки аутентификации в IIS

Первая группа параметров в этом диалоговом окне управляет анонимным доступом пользователей, не прошедших аутентификацию. Снятие соответствующего флажка рав-

носительно добавлению элемента <deny users="?"> в файл Web.config. Настоятельно рекомендуется позволить IIS проверять пароль учетной записи для анонимного доступа, но если приложение развернуто на нескольких серверах, возможно, потребуется изменить пароль.

Флажки во второй секции диалогового окна позволяют приложению использовать, помимо механизмов аутентификации Windows, аутентификацию Basic или Digest. Эти методы менее безопасны, чем аутентификация с помощью встроенных механизмов Windows, форм или службы Passport и реализуются с участием IIS, а не ASP.NET.

Как видно из рис. 8-10, IIS позволяет задействовать несколько методов аутентификации. Если включено несколько методов аутентификации, метод *Authentication* объекта *Identity* позволяет определить в коде, какой из этих методов применен для аутентификации пользователя- Это делается следующим образом:

Visual Basic .NET

```
Response.Write(User.Identity.AuthenticationType)
```

Visual C#

```
Response.Write(User.Identity.AuthenticationType);
```

Дополнительные сведения об аутентификации Basic и Digest см. в справочной системе IIS.

Занятие 3. Аутентификация Forms

При применении этого метода аутентификации автоматически отображается особая Web-форма, предназначенная для сбора сведений о пользователе. Код, связанный с этой формой, идентифицирует пользователей и предоставляет им доступ, сверяя введенные ими данные со списком пользователей, хранящимся в файле Web.config либо в специальной базе данных (БД).

Достоинство аутентификации с помощью форм заключается в том, что для получения доступа к приложению пользователь не обязательно должен быть членом домена. Кроме того, этот метод аутентификации упрощает создание Web-приложений для электронной коммерции. Эти приложения обрабатывают заказы, размещаемые клиентами, поэтому им просто *необходим* доступ к сведениям о пользователе.

Изучив материал этого занятия, вы сможете:

- ✓ включать аутентификацию с помощью форм в Web-приложении;
- ✓ создавать Web-формы для ввода имени пользователя и пароля;
- ✓ идентифицировать пользователей при помощи списка из файла Web.config приложения;
- ✓ добавлять новых пользователей и идентифицировать зарегистрированных с помощью БД.

Продолжительность занятия — около 20 минут.

Как задействовать аутентификацию с помощью форм

Этот метод аутентификации позволяет создать собственную базу данных со сведениями о пользователях и применять ее для проверки идентификационных данных пользователей, посещающих ваш Web-сайт.

Чтобы применять для идентификации и авторизации пользователей аутентификацию с помощью форм, выполните следующие действия.

1. В файле Web.config установите режим аутентификации в «Forms».
2. Создайте Web-форму для сбора учетных данных.
3. Создайте файл или БД для хранения имен и паролей пользователей.
4. Напишите код, добавляющий сведения о новых пользователях в файл или БД.
5. Напишите код, выполняющий аутентификацию пользователей с применением файла или БД со сведениями о пользователях.

При обращении к Web-приложению, использующему аутентификацию Forms, ASP.NET отображает входную Web-форму, заданную в файле Web.config. Для пользователя, прошедшего авторизацию, ASP.NET генерирует сертификат авторизации в виде файла cookie и хранит его в течение времени, заданного параметрами файла Web.config.

Различие между аутентификацией средствами Windows и с помощью форм заключается в том, что во втором случае все задачи по аутентификации и авторизации пользователей приложение выполняет самостоятельно. Вы должны создать Web-формы и написать код, которые будут принимать имя и пароль пользователя и сверять их со списком авторизованных пользователей.

Для реализации аутентификации с помощью форм применяются классы из пространства имен *System.Web.Security*. Чтобы сделать эти классы доступными коду, необходимо поместить в начало каждого модуля, выполняющего аутентификацию, следующие операторы:

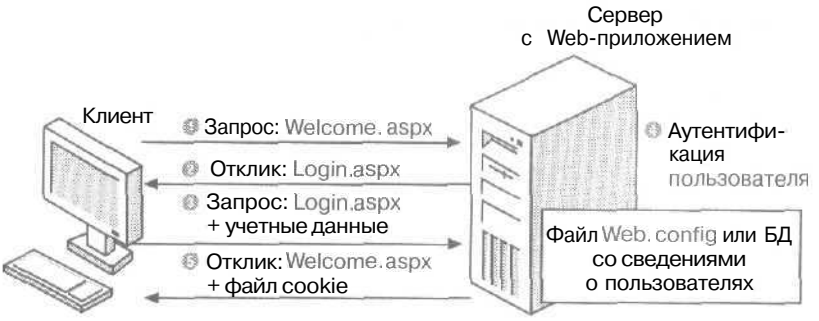


Рис. 8-11. Аутентификация при помощи форм

Visual Basic .NET

```
Imports System.Web.Security
```

Visual C#

```
using System.Web.Security
```

Далее речь пойдет о реализации аутентификации с помощью форм в Web-приложении, при этом мы поясним перечисленные здесь общие действия.

Выбор метода аутентификации

Используемый приложением метод аутентификации устанавливают в элементе *<authorization>* файла Web.config. Чтобы настроить приложение для использования аутентификации с помощью форм, внесите в файл Web.config следующие изменения:

```
<authentication mode="Forms" >                <!- Режим
                                аутентификации ->
  <forms loginUrl="LogIn.aspx" >              <!- Входная форма ->
      <credentials passwordFormat="Clear"> <!- Список
                                             пользователей ->
            <user name="June" password="JuneBug"/>
            <user name="Walter" password="Halifax"/>
      </credentials>
  </forms>
</authentication>

<authorization>
<!-- Запретить доступ любому, кто не прошел аутентификацию ->
  <deny users="?" />
</authorization>
```

Показанный фрагмент файла Web.config демонстрирует упрощенный пример аутентификации с помощью форм, в котором для большинства параметров оставлены значения по умолчанию, а список пользователей включен в файл Web.config. В таблице 8-2 перечислены все возможные атрибуты для элементов, определяющих аутентификацию с помощью форм.

Таблица 8-2. Параметры файла Web.config, определяющие аутентификацию с помощью форм

Элемент	Атрибут	Описание
<authentication >	*mode*	Для использования аутентификации с помощью форм необходимо установить этот атрибут в *Forms*
<forms>	*name*	Определяет имя файла cookie, в котором хранятся учетные данные пользователя, по умолчанию это файл с расширением .authaspx. Если несколько приложений на сервере используют аутентификацию с помощью форм, для каждого из них необходимо указать уникальное имя файла cookie
	login Url	Служит для указания Web-формы, которая отображается, пока пользователь не прошел аутентификацию; по умолчанию задана форма Default.aspx
	protection	Определяет метод, используемый ASP.NET для зашиты файла cookie с идентификационными данными, хранящийся на пользовательском компьютере. По умолчанию задано значение *All*, определяющее защиту шифрованием и проверку подлинности данных. Также может иметь значения *Encryption*, *Validation* и *None*
	timeout	Устанавливает время хранения файла cookie на пользовательском компьютере (в минутах). По умолчанию имеет значение 30 (минут). При получении запроса от пользователя по прошествии более половины установленного времени ASP.NET автоматически обновляет файл cookie
	path	Определяет путь к файлу cookie на пользовательской машине. По умолчанию равен «\»
<credentials>	*passwordFormat*	Указывает алгоритм шифрования пароля пользователя. По умолчанию это *SHA1*, другие возможные значения — *MD5* и *Clear* (запрет шифрования)
<users>	*name*	Задает имя пользователя
	password	Задает пароль пользователя

Элемент *<credentials>* позволяет хранить в файле Web.config список пользователей. Это удобно при простой аутентификации, когда администратор сам добавляет новых пользователей и назначает им пароли, но лучше разрешить клиентам самостоятельно создавать свои учетные записи и выбирать себе пароли.

Для этого следует создать файл или БД для хранения имен пользователей к зашифрованных паролей. БД дает дополнительное преимущество, поскольку в ней можно хранить дополнительные сведения о пользователях, например адрес доставки и список заказов,

В следующем разделе показано, как проводить аутентификацию пользователей с помощью учетных данных, хранящихся в файле Web.config; далее речь пойдет о более совершенном (и более сложном) способе аутентификации — когда требуется БД со сведениями о пользователе.

Создание входной Web-формы

Для аутентификации пользователей с помощью форм необходима прежде всего входная Web-форма, которая позволит зарегистрированным пользователям войти на сайт (а но-

вым **пользователям — зарегистрироваться). Имя этой формы определяется элементом** *<forms>* **файла** Web.config.

В качестве входной формы годится простая Web-форма, содержащая **лишь пару текстовых полей и кнопку. Другой вариант — включить входную форму в** страницу **с не**защищенным **содержимым. Например, можно поместить поля для ввода учетных данных на начальную** страницу **приложения (рис. 8-12).**

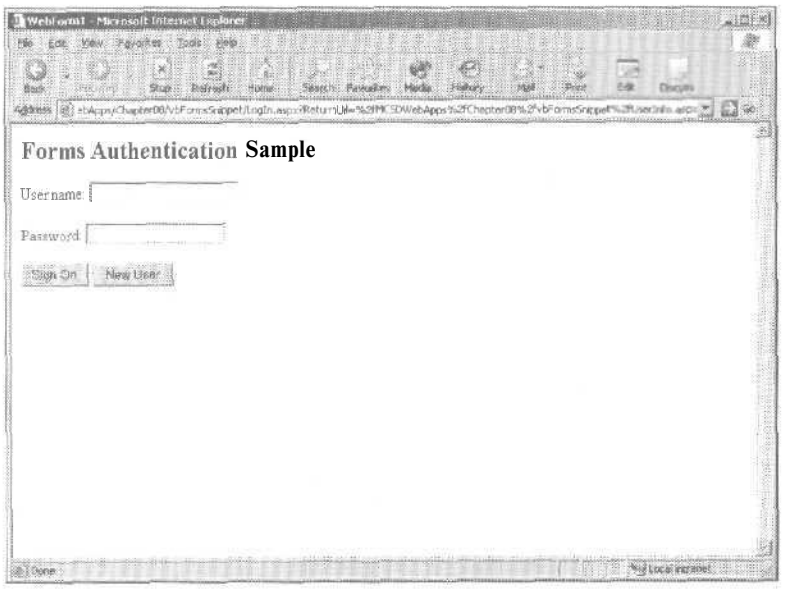


Рис. 8-12. Поля для ввода учетных данных на начальной странице приложения

Когда пользователь щелкает кнопку Sign On, приложение проверяет его имя и пароль, после чего генерирует сертификат аутентификации **и предоставляет доступ к** остальной **части приложения. Вот как выглядит код,** выполняющий **эти действия:**

Visual Basic .NET

```
' Эта строка должна быть в начале модуля.
Imports System.Web.Security

Private Sub butSignOn_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butSignOn.Click
    ' Сверить введенные имя и пароль пользователя
    ' с таковыми из элемента <credentials>.
    If FormsAuthentication.Authenticate(txtUserName.Text, txtPassword.Text) _
        Then
        ' Если они найдены в списке, отобразить начальную страницу приложения,
        FormsAuthentication.RedirectFromLoginPage(txtUserName.Text, True)
    Else
        ' в противном случае очистить поле для ввода пароля.
        txtPassword.Text = ""
        ' После третьей неудачной попытки отобразить страницу "Access Denied",
        If CInt(ViewState("Tries")) > 3 Then
```

```
            Response.Redirect("Denied.htm")
        Else
            в противном случае увеличить число использованных попыток,
            ViewState("Tries") = CInt(ViewState("Tries")) + 1
        End If
    End If
End Sub
```

Visual C#

```
// Эта строка должна быть в начале модуля,
   using System.Web.Security;

private void butSignOn_Click(object sender, System.EventArgs e)
{
      // Сверить введенные имя и пароль пользователя
      // с таковыми из элемента <credentials>.
      if (FormsAuthentication.Authenticate(txtUserName.Text,
        txtPassword.Text))
            // Если они найдены в списке, отобразить начальную
            // страницу приложения,
            FormsAuthentication.RedirectFromLoginPage(txtUserName.Text,
              true);
      else
      {
            // в противном случае очистить поле для ввода пароля,
            txtPassword.Text = "";
            // После третьей неудачной попытки отобразить страницу
            // "Access Denied",
            if (System.Convert.ToInt32(ViewState["Tries"]) > 3)
            Response.Redirect("Denied.htm");
            else
            // в противном случае увеличить число использованных попыток,
            ViewState["Tries"] = System.Convert.ToInt32(ViewState["Tries"])
              + 1;
      }
}
```

Следует обратить внимание на рад моментов:

- класс *FormsAuthentication* определен в пространстве имен *System.Web.Security*, поэтому необходимо импортировать это пространство имен с помощью оператора *Imports* (в Visual Basic .NET, в Visual C# — *using)* либо ссылаться на этот класс с помощью полного имени;
- метод *Authenticate* класса *FormsAuthentication* сверяет имя и пароль пользователя со списком, заданным элементом *<credentials>* файла Web.config;
- метод *RedirectFromLoginPage* класса *FormsAuthentication* отображает начальную страницу приложения. Если поля для ввода учетных данных размещены на начальной странице приложения, после входа пользователя необходимо отключить их или как-то указать, что вход выполнен успешно;

- если пользователь ввел неверное имя или пароль, этот код дает ему еще две попытки, после чего отображает страницу Access Denied. Это HTML-страница, а не Web-форма, поскольку без аутентификации доступ к любым Web-формам приложения запрещен. Если вы направляете пользователей на другую страницу, убедитесь, что она находится вне области видимости приложения.

Класс *FormsAuthentication* позволяет пользователю выйти из приложения, когда он завершил работу, а также удалить файл cookie с паспортом аутентификации с его компьютера. Например, следующий код завершает пользовательский доступ к приложению и при попытке вновь получить его просит ввести имя и пароль.

Visual Basic .NET

```
Imports System.Web.Security

Private Sub butSignOut_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles butSignOut.Click
    ' Удалить файл cookie с паспортом аутентификации.
    FormsAuthentication.SignOut()
    ' Вернуть пользователя на эту (входную) страницу,
    Response.Redirect("UserInfo.aspx")
End Sub
```

Visual C#

```
using System.Web.Security;

private void butSignOut_Click(object sender, System.EventArgs e)
{
   // Удалить файл cookie с паспортом аутентификации.
   FormsAuthentication.SignOut();
   // Вернуть пользователя на эту (входную) страницу,
   Response.Redirect("UserInfo.aspx");
}
```

Аутентификация с помощью базы данных

Предыдущие разделы демонстрируют аутентификацию при помощи списка пользователей из файла Web.config — метод *Authenticate* класса *FormsAuthentication* читает этот файл по умолчанию. Это годится, когда имена и пароли пользователей создаются и обслуживаются системным администратором, но если пользователям разрешено самим выбирать себе имена и пароли, лучше хранить эту информацию в другом месте. Дело в том, что после модификации Web.config необходимо перезапустить Web-приложение, что приведет к сбросу переменных состояния *Application* и *Session,* используемых приложением.

Имена и пароли пользователей можно хранить в любых файлах, но БД дает ряд существенных преимуществ:

- имя пользователя можно применять как первичный ключ для хранения других сведений о пользователе;
- БД обеспечивает высокое быстродействие при извлечении пользовательских имен и паролей;
- SQL позволяет стандартизировать добавление, модификацию и извлечение записей.

Имена и пароли пользователей можно хранить в файле или в БД в зашифрованном виде, для шифрования используется метод *HashPasswordForStoringInConfigFile* класса *FormsAuthentication,* применяющий алгоритмы SHA1 или MD5:

Visual Basic .NET

```
' Зашифровать пароль.
Password = FormsAuthentication.HashPasswordForStoringInConfigFile(Password, "SHA1")
```

Visual C#

```
Password = FormsAuthentication.HashPasswordForStoringInConfigFile(Password, "SHA1");
```

Далее речь пойдет о том, как добавить новые имена и пароли пользователей к простой БД и как применять эту БД для аутентификации пользователей.

Добавление пользователей в базу данных

Чтобы добавить пользователей в БД, следует получить его имя и пароль из соответствующих текстовых полей, а также создать обработчик события, который добавляет пользователя в БД и уведомляет об успешном или неудачном добавлении пользователя. Следующий обработчик события вызывает вспомогательную функцию *AddUser* для добавления учетных данных пользователя в БД:

Visual Basic .NET

```
Private Sub butNewUser_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butNewUser.Click
    If AddUser(txtUserName.Text, txtPassword.Text) Then
        spnNote.InnerText = "User added."
    Else
        spnNote.InnerText = "User exists. Choose a different user name. '
    End If
End Sub
```

Visual C#

```
private void butNewUser_Click(object sender, System.EventArgs e)
{
   if (AddUser(txtUserName.Text, txtPassword.Text))
        spnNote.InnerText = "User added.";
   else
        spnNote.InnerText = "User exists. Choose a different user name.";
}
```

Вспомогательная функция *AddUser,* показанная в следующем коде, шифрует пароль перед записью учетных данных в БД, осуществляемой командой SQL INSERT. Если добавляемое имя пользователя уже есть в БД, блок обработки исключений перехватывает ошибку и возвращает значение *False,* свидетельствующее о неудачном добавлении пользователя.

Visual Basic .NET

```
Private Function AddUser(ByVal UserName As String, ByVal Password _
As String) As Boolean
```

```
    ' Объявить переменную для отслеживания состояния операции.
    Dim bSuccess As Boolean
    ' Зашифровать пароль.
    Password = _
      FormsAuthentication.HashPasswordForStoringInConfigFile(Password, "SHA1")
    ' Создать команду для вставки имени и пароля пользователя.
    Dim oleCommand As New OleDbCommand("INSERT INTO Users" + _
      " VALUES('" + UserName + "', '" + Password + "')", oledbUsers)
    ' Если такая запись уже существует, перехватить ошибку.
    Try
        ' Открыть соединение с БД,
        oledbUsers.Open()
        ' Если запись добавлена, установить переменную состояния в true...
        If oleCommand.ExecuteNonQuery() Then bSuccess = True
        ' ...и закрыть соединение,
        oledbUsers.Close()
    Catch
        ' если нет, считать операцию неудачной,
        bSuccess = False
        ' Закрыть соединение.
        oledbUsers.Close()
    End Try
    ' Вернуть уведомление об успешном или неудачном завершении.
    Return bSuccess
End Function
```

Visual C#

```
private bool AddUser(string UserName, string Password)
{
   /7 Объявить переменную для отслеживания состояния операции.
   bool bSuccess = false;
   // Зашифровать пароль.
   Password = FormsAuthentication.HashPasswordForStoringInConfigFile
                (Password, "SHA1");
   // Создать команду для вставки имени и пароля пользователя.
   OleDbCommand oleCommand = new OleDbCommand("INSERT INTO Users" +
     " VALUES('" + UserNarre + "', '" + Password + "')", oledbUsers);
   // Если такая запись уже существует, перехватить ошибку.
   try
   {
        // Открыть соединение с БД.
        oledbUsers.Open();
        // Если запись добавлена, установить переменную состояния в true...
        if (oleCommand.ExecuteNonQuery() != 0)
        {
             bSuccess = true;
```

```
            // ...и закрыть соединение.
            oledbUsers.Close();
        }
    }
    catch
    {
        // если нет. считать операцию неудачной.
        bSuccess = false;
        // Закрыть соединение.
        oledbUsers.Close();
    }
    // Вернуть уведомление об успешном или неудачном завершении.
    return bSuccess;
}
```

Проверка пользователей с помощью базы данных

Как вам уже известно, для аутентификации пользователей с помощью файла Web.config необходим метод *Authenticate*. Если же вы применяете аутентификацию с помощью БД, вам придется самостоятельно написать код для поиска учетных данных в БД и сравнения их с данными, введенными пользователем на форме. Прежде чем аутентифицировать клиента и предоставить ему доступ к приложению, следующий обработчик события вызывает вспомогательную функцию *Check Password*', чтобы проверить значения полей для ввода имени и пароля пользователя:

Visual Basic .NET

```
Private Sub butSignOn_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butSignOn.Click
    ' Если имя и пароль пользователя найдены, авторизировать пользователя
    ' и показать начальную страницу приложения.
    If CheckPassword(txtUserName.Text, txtPassword.Text) Then
        FormsAuthentication.RedirectFromLoginPage(txtUserName.Text, True)
    Else
        ' Отобразить сообщение.
        spnNote.InnerText = "User name or password not found. Try again."
        ' Дать пользователю три попытки входа.
        ViewState("tries") = ViewState("tries") + 1
        If ViewState("tries") > 3 Then
            Response.Redirect("Denied.htm")
        End If
    End If
End Sub
```

Visual C#

```
private void butSignOn_Click(object sender, System.EventArgs e)
{
    // Проверить имя и пароль пользователя,
    if (CheckPassword(txtUserName.Text, txtPassword.Text))
```

```
        // если они найдены, отобразить начальную страницу приложения,
        FormsAuthentication.RedirectFromLoginPage(txtUserName.Text, true);
    else
    {
        // в противном случае очистить поле для ввода пароля.
        txtPassword.Text = "";
        // Отобразить сообщение.
        spnNote.InnerText = "User name or password not found. Try again.";
        // Если это третья неудачная попытка, отобразить
        // страницу "Access Denied",
        if (System.Convert.ToInt32(ViewState["Tries"]) > 3)
            Response.Redirect("Denied.htm");
        else
        {
      // в противном случае увеличить число попыток.
      ViewState["Tries"] =
        System.Convert.ToInt32(ViewState["Tries"]) + 1;
      if (System.Convert.ToInt32(ViewState["Tries"]) > 3)
          Response.Redirect("Denied.htm");
        }
    }
}
```

Вспомогательная функция *CheckPassword* (см. код ниже) шифрует введенный пароль, ищет в БД запись с таким же именем пользователя и сравнивает зашифрованный пароль с паролем из найденной в БД записи. Обращение к БД производится в блоке обработки исключений, чтобы пользователь не увидел сообщения об ошибках, возникающих при конфликтах из-за блокировки БД.

Visual Baste .NET

```
Private Function CheckPassword(ByVal UserName As String, _
  ByVal Password As String) As Boolean
    ' Объявить переменную, отслеживающую состояние операции.
    Dim bSuccess As Boolean
    ' Зашифровать пароль.
    Password = _
     FormsAuthentication.HashPasswordForStoringInConfigFile(Password, "SHA1")
    ' Создать команду для извлечения строки с заданным значением UserName.
    Dim oleCommand As New OleDbCommand("SELECT * FROM Users" + _
      " WHERE UserName='" + txtUserName.Text + "'", oledbUsers)
    ' Следить за ошибками при обращении к БД,
    Try
        ' Открыть соединение с БД.
        oledbUsers.Open()
        ' Найти запись.
        Dim rdrUsers As OleDbDataReader = oleCommand.ExecuteReader()
        While rdrUsers.Read()
            If Password = rdrUsers.Item("Password") Then bSuccess = True
```

```
        End While
        ' Закрыть соединение.
        oledbUsers.Close()
    Catch
        ' Если запись не найдена, считать операцию неудачной.
        bSuccess = False
        ' Закрыть соединение.
        oledbUsers.Close()
    End Try
    Return bSuccess
End Function
```

Visual C#

```
private bool CheckPassword(string UserName, string Password)
{
   // Объявить переменную, отслеживающую состояние операции,
   bool bSuccess = false;
   // Зашифровать пароль.
   Password = FormsAuthentication.HashPasswordForStoringInConfigFile(Password,
     "SHA1");
   // Создать команду для извлечения строки с заданным значением UserName.
   OleDbCommand oleCommand - new OleDbCommand("SELECT * FROM Users" +
     " WHERE UserName='" + txtUserName.Text +"'", oledbUsers);
   // Следить за ошибками при обращении к БД.
   try
   {
         // Открыть соединение с БД.
         oledbUsers.Open();
         // Найти запись,
         OleDbDataReader rdrUsers = oleCommand.ExecuteReader();
         while (rdrUsers.Read())
         {
               if (Password -- rdrUsers["Password"].ToString())
               bSuccess = true;
         }
         // Закрыть соединение.
         oledbUsers.Close();
   }
   catch
   {
         // Если запись не найдена, считать операцию неудачной.
         bSuccess = false;
         // Закрыть соединение.
         oledbUsers.Close();
   }
   return bSuccess;
}
```

Занятие 4. Аутентификация Passport

При аутентификации Passport идентификация пользователей выполняется с помощью службы единого входа Microsoft Passport. Цель службы Passport — позволить пользователям Интернета входить на разные Web-сайты, требующие аутентификации, при помощи единых удостоверений. При этом Web-приложения могут извлекать сведения о пользователе из профиля, хранимого Microsoft.

Преимущество аутентификации Passport в том, что она освобождает клиента от необходимости запоминать отдельные учетные данные для каждого Web-сайта и позволяет ему централизованно хранить свой профиль. Аутентификация Passport позволяет получить доступ и к другим сервисам Microsoft, таким, как Passport Express Purchase.

Изучив материал этого занятия, вы сможете:

- ✓ устанавливать пакет Passport SDK;
- ✓ включать использование аутентификации Passport в Web-приложении;
- ✓ извлекать данные из пользовательского профиля службы Passport;
- ✓ завершать работу пользователя с приложением, удаляя файл cookie службы Passport.

Продолжительность занятия — около 20 минут.

Как задействовать аутентификацию Passport

Аутентификация Passport идентифицирует пользователей с помощью централизованной службы аутентификации Microsoft. Passport позволяет пользователям применять единые удостоверения для получения доступа к нескольким Web-приложениям. Чтобы включить аутентификацию Passport в Web-приложении, необходимо установить пакет Passport **SDK,** который находится в разделе Downloads на сайте MSDN (см. страницу по адресу *http://msdn.microsofi.com/downloads).*

Passport SDK можно бесплатно использовать для разработки и тестирования, но для использования этого пакета на открытом сайте необходимо приобрести у Microsoft годовую лицензию. Более подробные сведения о лицензировании Passport SDK опубликованы на сайте *http://www.microsoft.com/myservices/passport/.*

Когда пользователь обращается к приложению, применяющему аутентификацию Passport, ASP.NET проверяет наличие на клиентском компьютере действующего файла cookie службы Passport. Если такой cookie не найден, ASP.NET направляет пользователя на входную страницу службы Passport. После входа пользователя служба Passport аутентифицирует его, записывая на пользовательский компьютер файл cookie с идентификационными данными, и направляет пользователя на запрошенную Web-страницу. Процесс аутентификации Passport иллюстрируется на рис. 8-13.

Чтобы задействовать аутентификацию Passport, выполните следующие действия.

1. Установите пакет Passport SDK. Хотя этот пакет не входит в Visual Studio, в .NET Framework есть классы, позволяющие работать с Passport SDK после установки этого пакета.
2. Установите режим аутентификации в файле Web.config в *Passport* и запретите доступ всем пользователям, не прошедшим аутентификацию.
3. Посредством события *PassportAuthentication_OnAuthenticate*получите доступ к пользовательскому профилю службы Passport и авторизации пользователя,

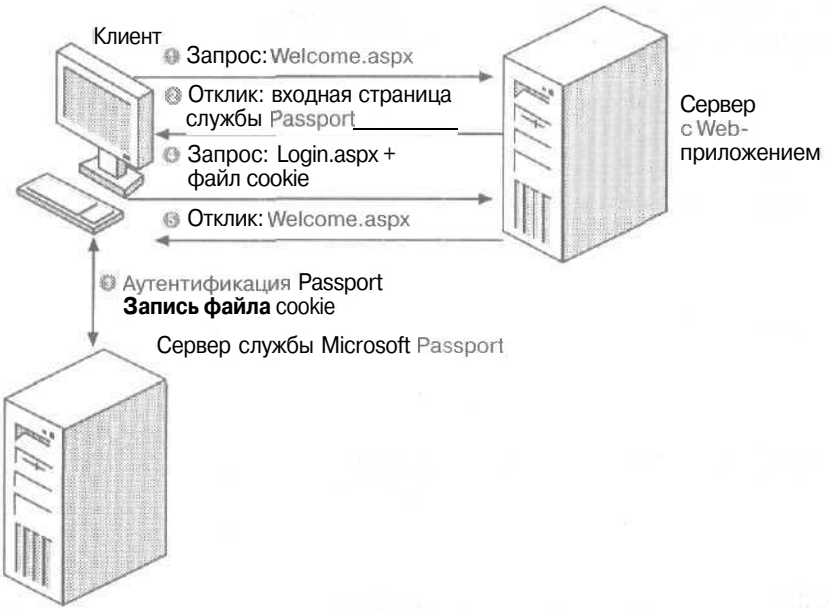


Рис. 8-13. Схема аутентификации с помощью службы Passport

4. Напишите кол процедуры для завершения работы пользователя, удаляющий файлы cookie службы Passport с пользовательского компьютера.

Например, следующие параметры файла Web.config включают аутентификацию Passport и заставляют всех пользователей проходить ее:

```
<authentication mode="Passport" />

<authorization>
    <deny users="?" /> <!- Запретить доступ пользователям,
                           не прошедшим аутентификацию ->
</authorization>
```

Если запустить приложение с такими параметрами Web.config на локальном компьютере, оно автоматически направит пользователя на входную страницу службы Passport. Как видно из рис. 8-14, тестовая (нелицензионная) и лицензионная версии пакета Passport SDK отображают входные страницы разного вида.

После входа пользователя Passport записывает на его компьютер файл cookie, подтверждающий авторизацию, и направляет пользователя на исходно запрошенную им страницу. Сведения о пользователе хранятся в профиле службы Passport. Доступ к этому профилю получают с помощью события *PassportAuthentication_OnAuthenticate*, определенного в модуле Global.asax, например, так:

Visual Basic .NET

```
' Эта строка должна быть в начале модуля.
Imports System.Web.Security

Private Sub PassportAuthentication_OnAuthenticate(ByVal sender As Object, _
  ByVal e As PassportAuthenticationEventArgs)
```

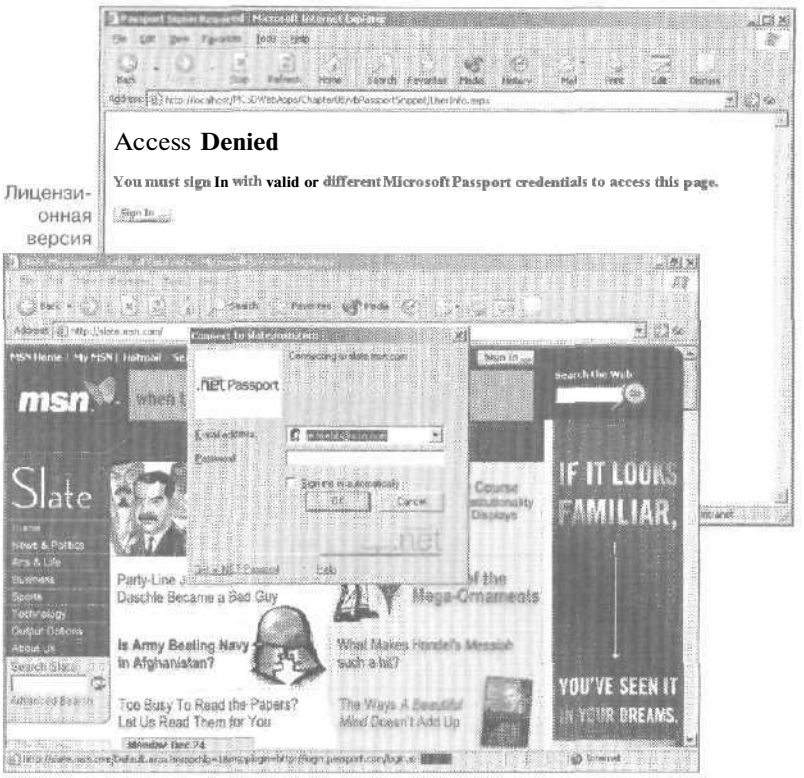


Рис. 8-14. Входные страницы службы Passport

```
    ' Если аутентификация успешна,
    ' получить идентификационные данные Passport для сеанса,
    If e.Identity.IsAuthenticated Then
        Response.Write("Name: " & e.Identity.Item("FirstName") & _
          " " & e.Identity.Item("LastName") & "<br>")
        Response.Write("Address: " & e.Identity.Item("City") & _
          "     " & e.Identity.Item("PostalCode") & "<br>")
        Response.Write("Email: " _
        & e.Identity.Item("PreferredEmail") & _
          "<br>")
        Response.Write("Passport ID: " & e.Identity.Name & "<br>")
    End If
End Sub
```

Visual C#

```
// Эта строка должна быть в начале модуля,
   using System.Web.Security;

protected void PassportAuthentication_OnAuthenticate(Object sender,
PassportAuthenticationEventArgs e)
```

```
{
    // Если аутентификация успешна.
    // получить идентификационные данные Passport для сеанса.
    if (e.Identity.IsAuthenticated)
    {
        Response.Write("Name: " + e.Identity["FirstName"] +
          " " + e.Identity["LastName"] + "<br>");
        Response.Write("Address: " + e.Identity["City"] +
          "    " + e.Identity["PostalCode"] + "<br>");
        Response.Write("Email: " + e.Identity["PreferredEmail"] +
          "<br>");
        Response.Write("Passport ID: " + e.Identity.Name + "<br>");
    }
}
```

Этот код отображает имя и адрес пользователя, а также идентификационные данные из его профиля службы Passport. Если применяется нелицензионная версия Passport SDK, то отображаются данные из тестовой, а не из настоящей учетной записи (рис. 8-15).

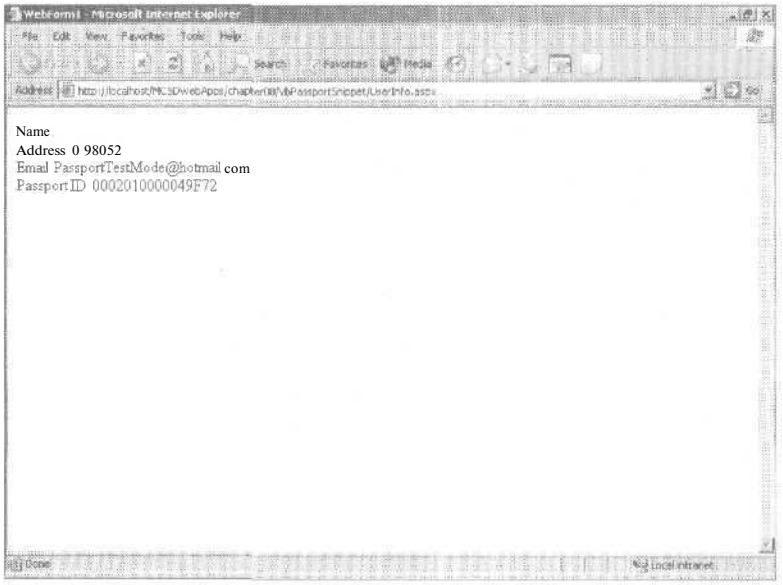


Рис. 8-15. Идентификационные данные тестовой учетной записи Passport

Подтверждение авторизации Passport и данные профиля хранятся в пяти отдельных файлах cookie на компьютере пользователя. Passport SDK требует удалять эти cookie по завершении работы пользователя с Web-приложением. Как это делается, показано на примере следующего обработчика события:

Visual Basic .NET

```
Private Sub butSignOut_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butSignOut.Click
    ' Завершить работу, удалив файлы cookie службы Passport.
```

```
    Response.Cookies("MSPProf").Expires - Now
    Response.Cookies("MSPAuth").Expires - Now
    Response.Cookies("MSPSecAuth").Expires = Now
    Response.Cookies("MSPProfC").Expires = Now
    Response.Cookies("MSPConsent").Expires - Now
    ' Направить пользователя на входную страницу,
    Response.Redirect("UserInfo.aspx")
End Sub
```

Visual C#

```
private void butSignOut_Click(object sender, System.EventArgs e)
{
   // Завершить работу, удалив файлы cookie службы Passport.
   Response.Cookies["MSPProf"].Expires = DateTime.Now;
   Response.Cookies["MSPAuth"].Expires - DateTime.Now;
   Response.Cookies["MSPSecAuth"].Expires - DateTime.Now;
   Response.Cookies["MSPProfC"].Expires - DateTime.Now;
   Response.Cookies["MSPConsent"].Expires = DateTime.Now;
   // Направить пользователя на входную страницу.
   Response.Redirect("UserInfo.aspx");
}
```

Аутентификация с помощью службы Passport предоставляет ряд дополнительных коммерческих возможностей, в том числе и защиту детей от нежелательной информации. Эти возможности описаны в документации на Passport SDK.

Занятие 5. Защита данных при пересылке

До сих пор мы обсуждали такие меры безопасности, как идентификация пользователей и защита Web-приложений от несанкционированного доступа, однако не менее важно обеспечить защиту конфиденциальных данных от доступа посторонних при пересылке через Интернет.

Для защиты взаимодействия, осуществляемого через Интернет, IIS поддерживает стандартные средства шифрования запросов и откликов. Эти криптографические средства требуют наличия на сервере шифровального ключа, который называется *сертификатом сервера* и выдается независимой сторонней компанией, или *центром сертификации* (Certificate Authority).

Изучив материал этого занятия, вы сможете:

- ✓ понять принцип использования протокола Secure Sockets Layer (SSL) в IIS;
- ✓ получать сертификаты сервера для защиты взаимодействия между Web-приложением и пользователями путем шифрования данных;
- ✓ инициировать защищенное взаимодействие между пользователем и Web-приложением;
- ✓ завершать защищенное взаимодействие;
- ✓ требовать защищенное соединение для предоставления доступа к Web-форме,

Продолжительность занятия — около 20 минут.

Защита взаимодействия с помощью SSL

Протокол *Secure Sockets Layer (SSL)* — это стандартное средство зашиты данных, пересылаемых через Интернет, от доступа посторонних лиц. Когда пользователь запрашивает защищенную Web-страницу, сервер генерирует криптографический ключ и шифрует им данные страницы перед отправкой пользователю. На стороне клиента браузер средствами того же самого ключа расшифровывает полученную Web-страницу и шифрует новые запросы, генерируемые защищенной страницей. Схема этого процесса показана на рис. 8-16.

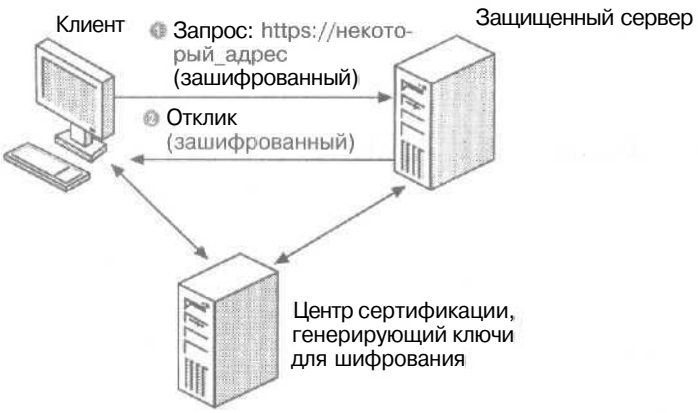


Рис. 8-16. Схема взаимодействия, защищенного протоколом SSL

Для использования SSL в приложении необходимо особое разрешение действующего центра сертификации (Certificate Authority, CA). Оно выдается в форме сертификата сер-

вера, который устанавливается в IIS и идентифицирует сервер. Центр сертификации (ЦС) лицензирует серверные сертификаты (за определенную плату) и гарантирует подлинность сервера в Интернете, подтверждая его идентификационные данные.

Устанавливая соединение через защищенный канал, браузер запрашивает сертификат сервера и проверяет его наличие в списке доверенных сайтов, предоставленном ЦС. Если сервер — держатель сертификата — не найден в списке сайтов, которым пользователь доверяет, фактический адрес сервера не совпадает с зарегистрированным, либо, если при проверке сертификата возникли другие ошибки, браузер отображает предупреждение (рис 8-17).

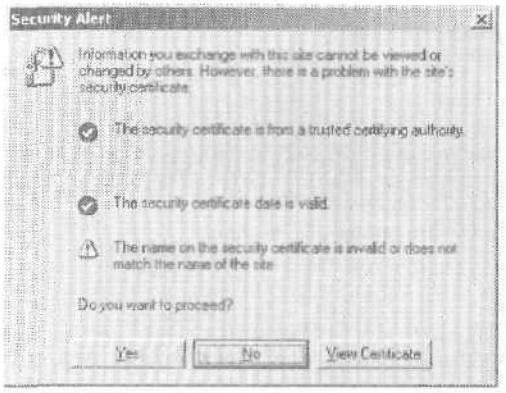


Рис. 8-17. Уведомление о проблемах при проверке сертификата сервера

Таким образом, ЦС не только предоставляет шифровальный ключ для защиты данных при передаче, но и подтверждает подлинность сайта, на который обратился пользователь. Крупнейшим ЦС считается VeriSign. На момент написания этой книги помимо платных услуг эта компания предлагала бесплатные пробные сертификаты (см. сайт *http://www.verisign.com)*.

Чтобы применить SSL в Web-приложении, выполните следующие действия.

1. Сгенерируйте запрос на выдачу сертификата от IIS.
2. Запросите сертификат у ЦС.
3. Установите сертификат на сервере с помощью IIS.
4. Установите сертификат в браузерах, если вы используете тестовый сертификат.
5. Применяйте для обращения к защищенным страницам приложения протокол HTTPS.

Более подробно эти действия описаны в следующих разделах.

Генерация запроса на выдачу сертификата

Чтобы запросить серверный сертификат у ЦС, нужно сгенерировать запрос от IIS. Запрос сертификата содержит зашифрованные сведения о вашем сервере, которые позволят ЦС идентифицировать этот сервер в Интернете.

Чтобы сгенерировать запрос сертификата с помощью оснастки IIS, выполните следующие действия.

1. Выберите Default Web Site из списка папок в левой части окна IIS, затем из меню Action выберите команду Properties — IIS откроет диалоговое окно Default Web Site Properties.
2. В этом окне перейдите на вкладку Directory Security и щелкните Server Certificate. IIS запустит мастер Web Server Certificate (рис. 8-18),
3. Следуйте указаниям мастера, которые довольно просты, и щелкайте кнопку Next, чтобы перейти на следующую страницу мастера.

4. Когда вы щелкнете кнопку Finish на последней странице, мастер создаст зашифрованный текстовый файл с расширением .cer. Это и есть запрос сертификата, который вы отправите в UC.

IIS требует, чтобы перед созданием защищенного соединения или подчиненных сайтов в корневом каталоге сервера был создан сертификат. Вот почему на этапе 1 требовалось выбрать Default Web Site (или корневой Web-сайт, если вы его переименовали). Установив сертификат сервера в корневой каталог, можно повторить эту процедуру для подчиненных сайтов, если для них требуются отдельные сертификаты.



Рис. 8-18, **Мастер Web** Server **Certificate**

Запрос сертификата

Процедура запроса сертификата зависит от выбранного ЦС. Как отмечалось ранее, VeriSign предоставляет бесплатные тестовые сертификаты, которые прекрасно подходят для учебных заданий из этой книги.

Чтобы запросить сертификат, выполните следующие действия.

1. Посетите Web-сайт ЦС и запросите сертификат для использования SSL. От уровня защиты и объема поддержки, предоставляемой ЦС, обычно зависит размер ежегодных выплат.
2. Выберите нужный тип серверного сертификата и введите регистрационные данные для подачи запроса.
3. Скопируйте или отправьте в ЦС по электронной почте файл запроса, созданный в предыдущем разделе.

Приняв решение о выдаче сертификата серверу, ЦС присылает сертификат, обычно по электронной почте. Сохраните сертификат в текстовом файле с расширением .cer. Для этого нужно вырезать его из тела сообщения и вставить в документ Notepad или другого редактора, как показано на рис. 8-19.

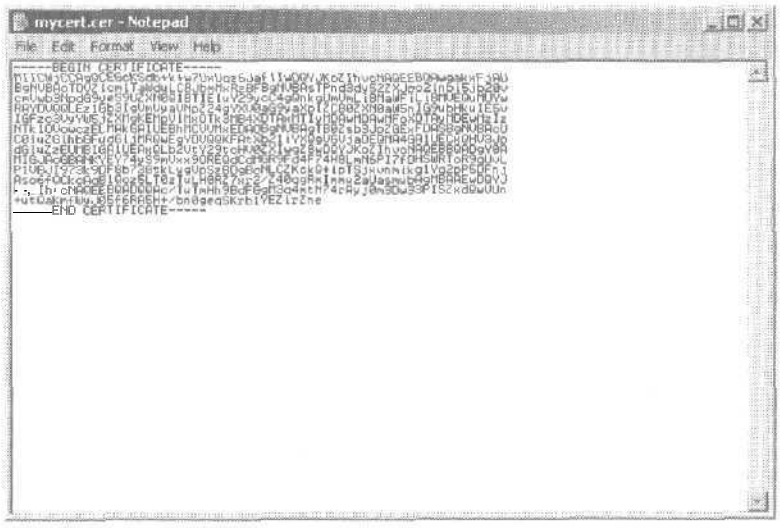


Рис. 8-19. Сохранение сертификата сервера

Установка сертификата

Чтобы получить возможность использования в Web-приложениях протокола SSL, сохраненный сертификат сервера следует установить в IIS.

Чтобы установить сертификат сервера в IIS, выполните следующие действия.

1. Выберите Default Web Site из списка папок в левой части окна IIS, затем из меню Action выберите команду Properties. IIS откроет диалоговое окно Default Web Site Properties.
2. В этом окне перейдите на вкладку Directory Security и щелкните Server Certificate — IIS запустит мастер Web Server Certificate (рис. 8-18).
3. Щелкните Next и выберите Process The Pending Request And Install The Certificate (рис. 8-20).
4. Щелкните Next и введите имя файла сертификата, как показано на рис. 8-21. Файл сертификата — это файл, созданный в предыдущем разделе.
5. Щелкните Next, затем Finish, чтобы завершить установку.

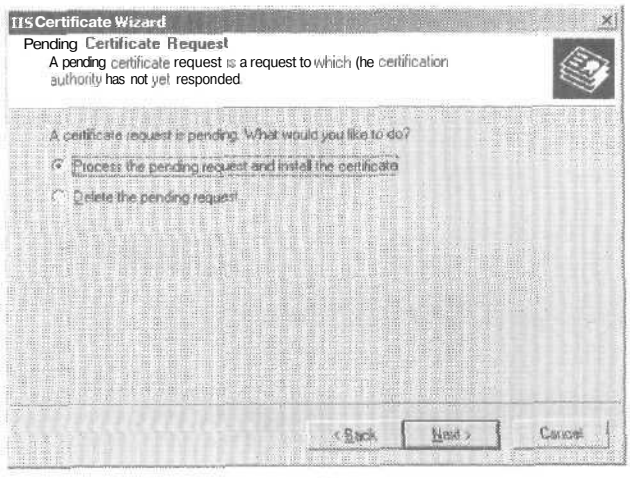


Рис. 8-20. Установка сертификата сервера в IIS с помощью мастера Server Certificate

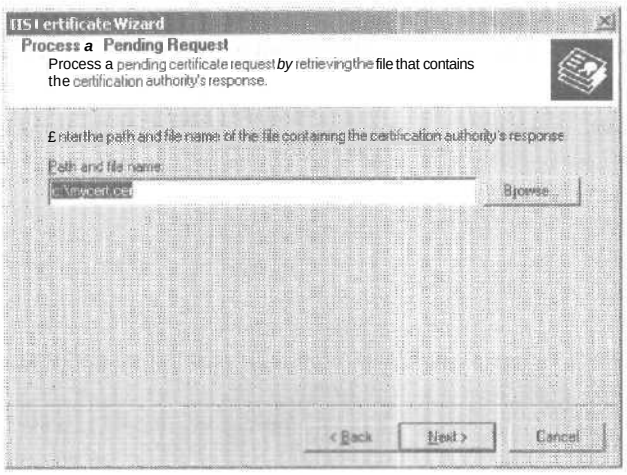


Рис. 8-21. Ввод имени файла сертификата сервера

Использование защищенного подключения

После установки сертификата сервера можно использовать защищенное подключение для просмотра любых Web-страниц, запрошенных через протокол HTTPS (Secure Hypertext Transfer Protocol). Например, следующая гиперссылка отображает защищенную Web-страницу, показанную на рис. 8-22:

```
<a href="https://www.contoso.com/mscdkit/Secure.aspx">Go to secure page.</a>
```

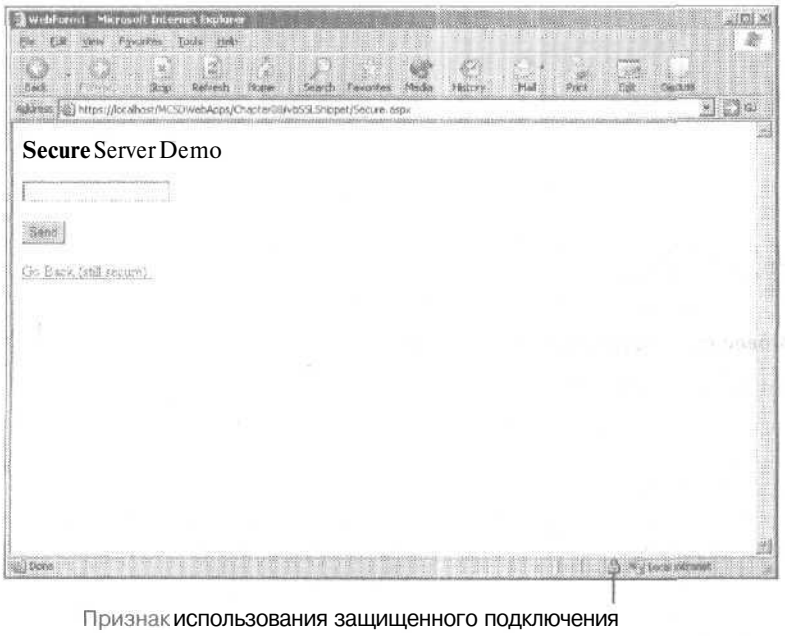


Рис. 8-22. Защищенная Web-страница

Использование защищенного подключения начинается при переходе на защищенную страницу (по протоколу HTTPS:) и продолжается до перехода на незащищенный сайт.

Например, щелчок ссылки Go Back на странице, показанной на рис. 8-22, не прерывает защищенное подключение, так как эта ссылка использует следующий относительный адрес:

```
<a href="Default.aspx">Go Back (still secure).</a>
```

Чтобы завершить защищенное подключение, перейдите на страницу, использующую протокол HTTP, например, по следующей ссылке:

```
<a href="http://www.contoso.com/mscdkit/Default.aspx">
Go Back (not secure).</a>
```

IIS позволяет требовать защищенное подключение для просмотра отдельных папок и файлов вашего Web-приложения, что позволит предотвратить случайный (или преднамеренный) просмотр защищенных страниц через уязвимый протокол HTTP.

Чтобы при помощи IIS сделать наличие защищенного подключения необходимым для просмотра Web-страницы, выполните следующие действия.

1. Выберите папку или файл, которые необходимо просматривать только через защищенное подключение, и выберите из меню Action команду Properties — IIS откроет диалоговое окно свойств этой папки или файла.
2. Перейдите на вкладку Directory Security и щелкните кнопку Edit в секции Secure Communications. IIS откроет диалоговое окно Secure Communications, показанное на рис. 8-23.

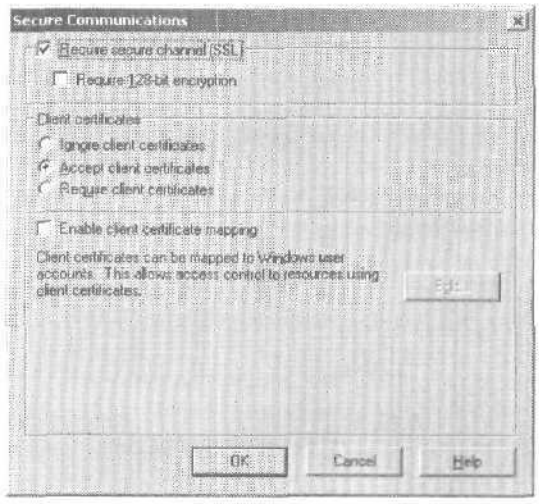


Рис. 8-23. Диалоговое окно Secure Communications

3. Установите флажок Require Secure Channel (SSL) и щелкните **OK.**

Web-страницы, требующие защищенного подключения, нельзя просматривать через протокол HTTP. Пользователь должен ввести URL или щелкнуть ссылку, использующую протокол HTTPS; в противном случае ему будет отказано в доступе (рис. 8-24).

Поскольку HTTP и HTTPS отличаются всего лишь одной буквой, многие пользователи упускают это различие из виду. Поэтому важно, чтоб в приложении была точка входа, доступная через протокол HTTP, направляющая на защищенные (HTTPS) страницы. Точкой входа на защищенный сайт может стать, например, незащищенная страница по умолчанию, которая сразу же направляет пользователя на защищенные страницы сайта. Требование защищенного подключения для просмотра всех без исключения страниц Web-сайта — верный способ снизить трафик до нуля!

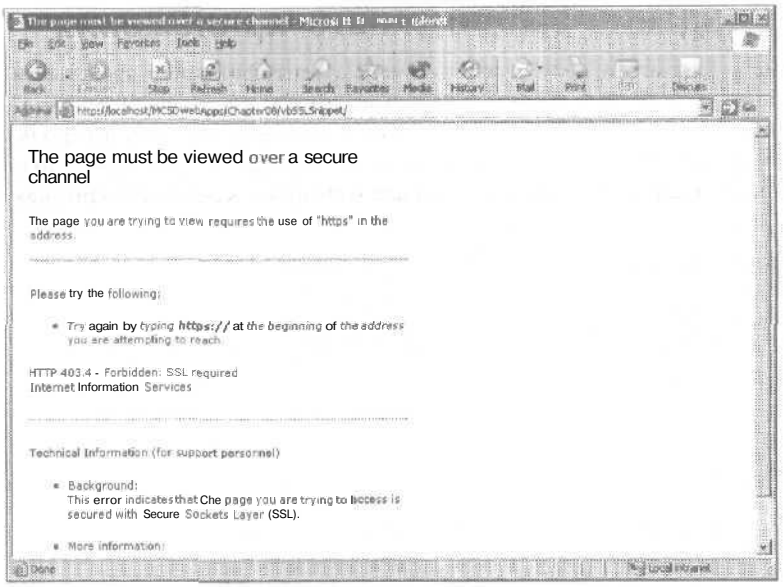


Рис. 8-24. Отказ в доступе через незащищенное соединение

Резюме

- Аутентификация — это процесс идентификации пользователей, а авторизация — процесс предоставления им определенных прав доступа.
- Анонимный доступ считается методом доступа к Web-приложению по умолчанию. Анонимные пользователи работают с Web-приложением под учетной записью Windows с именем вида IUSER_*имя_компьютера*.
- Посредством пространства имен *System.Web.Security* ASP.NET поддерживает три режима аутентификации: средствами Windows *(Windows),* при помощи форм *(Forms)* и службы Passport *(Passport).*
- ASP.NET не проводит аутентификацию запросов HTML-страниц автоматически. Чтобы заставить пользователей пройти аутентификацию перед просмотром таких страниц, при помоши IIS зарегистрируйте файлы с расширениями .htm и .html для обработки исполняемыми компонентами ASP.NET.
- Чтобы запретить анонимный доступ и сделать аутентификацию обязательной, добавьте в секцию авторизации файла Web.config элемент <*deny users*=»*?*» >. Этот элемент действует независимо от режима аутентификации.
- Элемент <*authorization* > позволяет разрешить или запретить доступ некоторым пользователям при аутентификации Windows.
- При помощи элемента <*credentials*> или внешней БД с учетными данными можно управлять доступом пользователей при аутентификации с помощью форм.
- Изменение файла Web.config вызывает перезапуск Web-приложения, поэтому не следует добавлять пользователей к элементу <*credentials*> файла Web.config во время выполнения. Вместо Web.config следует применять для хранения пользовательских имен и паролей внешний файл или **БД.**
- Чтобы задействовать классы для работы со службой Passport из пространства имен *System.Web.Security* установите пакет Passport SDK.
- Чтобы зашитить взаимодействие через Интернет шифрованием, установите сертификат сервера и примените протокол HTTPS.

Практикум 8. Создание защищенного приложения

Цель этого практикума — создать простое приложение с аутентификацией средствами форм для идентификации пользователей и предоставления им прав доступа. Это приложение должно позволять новым клиентам регистрироваться, выбирая себе имя и пароль, а уже зарегистрированным — входить на сайт. После входа пользователя приложении должно отображать его идентификационные данные. Готовое Web-приложение выглядит, как на рис. 8-25.

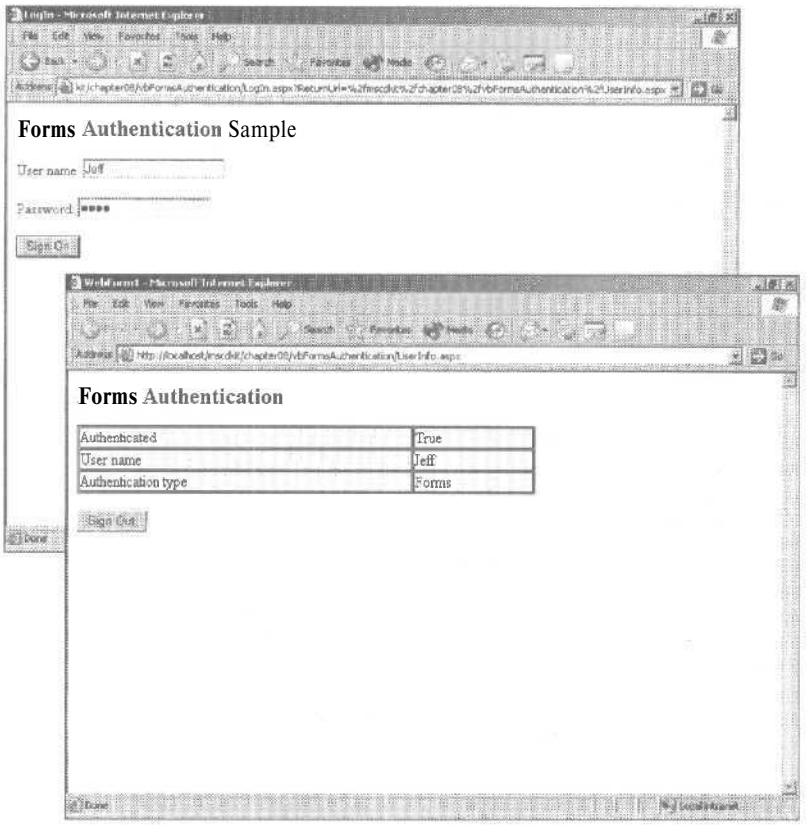


Рис. 8-25. Web-формы Login и UserInfo

Продолжительность практикума — около 30 минут.

Упражнение 1. Установка режима аутентификации

Чтобы задать аутентификацию с помощью форм, создайте новое приложение Web Forms, откройте файл Web.config, сгенерированный для нового проекта, и внесите следующие изменения в его элемент <*authentication*>:

```
<authentication mode="Forms">
    <forms loginUrl="LogIn.aspx">
        <credentials passwordFormat="SHA1">
    </credentials>
```

```
    </forms>
</authentication>
```

Эти параметры задают аутентификацию с помощью форм, направляют пользователей, не прошедших аутентификацию, на Web-форму LogIn.aspx и активизируют шифрование паролей по алгоритму **SHA1.** Чтобы сделать аутентификацию с применением этих параметров обязательной, добавьте к файлу Web.config следующий элемент *<authorization >*:

```
<authorization>
    <deny users="?" /> <!- Запретить доступ пользователям,
                           не прошедшим аутентификацию –>
</authorization>
```

Показанные выше параметры запрещают доступ неидентифицированным пользователям, вынуждая их проходить аутентификацию. После внесения указанных изменений в файл Web.config можно приступать к созданию Web-формы LogIn.aspx, которая позволит новым пользователям создать себе учетные записи, а зарегистрированным — войти на сайт.

Упражнение 2. Создание Web-формы Login



Сейчас вы создадите Web-форму Logln.aspx и файл данных Users.xml, которые потребуются для аутентификации пользователей. Поскольку это всего лишь упрощенный пример, для неизвестного пользователя Logln.aspx просто создает новую учетную запись.

В реальном же приложении незарегистрированных пользователей следует направлять на специальную Web-страницу для создания новых учетных записей либо обязать администратора создавать учетные записи для пользователей и отправлять им пароли по электронной почте. В любом случае при создании входной формы приложения действуют те же основные принципы.

Чтобы создать файл данных Users.xml, в редакторе Visual Studio или другом текстовом редакторе создайте текстовый файл со следующей строкой:

```
<users name="" password="" />
```

Наше приложение-пример применяет этот файл как шаблон для сохранения пользовательских имен и паролей. Так как пароли будут шифроваться, в первой строке мы оставим поля пустыми. В дальнейшем, когда к файлу добавятся учетные данные, эту строку можно удалить.

Создайте Web-форму LogIn.aspx и добавьте на нее текстовые поля, верификаторы и другие элементы управления, перечисленные ниже, и задайте для них свойства, как указано в таблице:

Тип элемента управления	Свойство	Значение
Text Box	ID	TxtUserName
RequiredFieldValidator	И)	VldUserName
	ControlToValidate	TxtUserName
	ErrorMessage	Name is required.
Text Box	[D	TxtPassword
	Text Mode	Password
RequiredFieldValidator	II>	VldPassword
	ControlTo Validate	Txt Password
	ErrorMessage	Password is required.

(окончание)

Тип элемента управления	Свойство	Значение
Button	ID	ButSignOn
	Text	Sign On
Button	ID	ButAddUser
	Text	Add User
	Visible	False
Label	ID	LblStatus

Теперь можно писать код для LogIn.aspx. Начните модуль кода этой формы со следующей строки:

Visual Basic .NET

```
Imports System.Web.Security
```

Visual C#

```
using System.Web.Security;
```

Добавьте к обработчику события *Click* элемента управления butSignOn код, который будет проверять имя и пароль пользователя, и, если имя пользователя не найдено, предлагать ему создать учетную запись (см. пример ниже). Обработчик события применяет две вспомогательные функции: *UserExists*, которая проверяет, существует ли заданное имя пользователя в Users.xml, и *PasswordValid*, проверяющую введенный пароль.

Visual Basic .NET

```
Private Sub butSignOn_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles butSignOn.Click
    ' Проверить имя пользователя.
    If UserExists(txtUserName.Text) Then
        ' Проверить пароль (в зашифрованном виде).
        If PasswordValid(txtUserName.Text, txtPassword.Text) Then
            ' Позволить пользователю войти.
            FormsAuthentication.RedirectFromLoginPage(txtUserName.Text, True)
        Else
            ' Отобразить сообщение о вводе неверного пароля.
            lblStatus.Text = "Password does not match. Try again."
        End If
    Else
        ' Если имя пользователя не найдено, предложить добавить его.
        lblStatus.Text = _
        "User name not found. Click Add User to add it now."
        ' Показать кнопку Add User.
        butAddUser.Visible = True
    End If
    ' Следить за числом попыток,
    Session("Tries") = CInt(Session("Tries")) + 1
     If Session("Tries") > 2 Then
```

```
        ' После третьей неудачной попытки отказать а доступе.
        Response.Redirect( "Denied.htm")
    End If
End Sub
```

Visual C#

```
private void butSignOn_Click(object sender, System.EventArgs e)
{
   // Проверить имя пользователя.
   if (UserExists(txtUserName.Text))
         // Проверить пароль (в зашифрованном виде).
         if (PasswordValid(txtUserName.Text, txtPassword.Text))
               // Позволить пользователю войти.
               FormsAuthentication.RedirectFromLoginPage(txtUserName.Text, true);
         else
               // Отобразить сообщение о вводе неверного пароля.
               lblStatus.Text = "Password does not match. Try again.";
   else
   {
         // Если имя пользователя не найдено, предложить добавить его,
         lblStatus.Text = "User not found. Click Add User to add it now.";
         // Показать кнопку Add User.
         butAddUser.Visible = true;
   }
   // Следить за числом попыток.
   ViewState["Tries"] = System.Convert.ToInt32(ViewState["Tries"]) + 1;
   if (System.Convert.ToInt32(ViewState["Tries"]) > 3)
         Response.Redirect("Denied.htm");
}
```

Функции *UserExists*и *PasswordValid* загружают содержимое файла Users.xml в набор данных и ищут в нем заданные имя и пароль. Если таковые найдены, эти функции возвращают *True*, в противном случае — *False*.

Visual Basic .NET

```
Private Function UserExists(ByVal UserName As String) As Boolean
    ' Создать набор данных для загрузки XML-файла.
    Dim dsUsers As New DataSet()
    ' Обработать ошибку, возникающую в случае отсутствия файла.
    Try
        ' Определить путь к файлу Users.xml.
        Dim strXMLFile As String = Server.MapPath(Request.ApplicationPath) &_
          "\Users.xml"
        ' Прочитать файл.
        dsUsers.ReadXml(strXMLFile, XmlReadMode.InferSchema)
        ' Для каждой строки в таблице Users...
        Dim rowUser As DataRow
        For Each rowUser In dsUsers.Tables("Users").Rows
```

```
            ' ...проверить, совпадает ли имя с заданным.
            If rowUser("name") = UserName Then
                Return True
                Ex   For
            End If
        Next
    Catch
        ' В случае ошибки вернуть false.
        Return False
    End Try
End Function

Private Function PasswordValid(ByVal UserName As String, _
  ByVal Password As String)
    ' Создать набор данных для загрузки XML-файла,
    Dim dsUsers As New DataSet()
    ' Обработать ошибку, возникающую в случае отсутствия файла,
    Try
        ' Определить путь к файлу Users.xml.
        Dim strXMLFile As String = Server.MapPath(Request.ApplicationPath) &_
          "\Users.xml"
        ' Прочитать файл
        dsUsers.ReadXml(strXMLFile, XmlReadMode.InferSchema)
        ' Для каждой строки в таблице Users...
        Dim rowUser As DataRow
        For Each rowUser In dsUsers.Tables("Users").Rows
            ' ...проверить, совпадает ли имя с заданным.
            If rowUser("name") = UserName Then
                If rowUser("password") = _
                  FormsAuthentication.HashPasswordForStoringInConfigFile _
                  (Password, "SHA1") Then
                    Return True
                    Exit For
                End If
            End If
        Next
    Catch
        ' В случае ошибки вернуть false,
        Return False
    End Try
End Function
```

Visual C#

```
private bool UserExists(string UserName)
{
    // Создать набор данных для загрузки XML-файла,
   DataSet dsUsers = new DataSetO;
```

```
    // Обработать ошибку, возникающую в случае отсутствия файла,
    try
    {
          // Определить путь к файлу Users.xml.
          string strXMLFile - Server.MapPath(Request.ApplicationPath) +
            "\\Users.xml";
          // Прочитать файл.
          dsUsers.ReadXml(strXMLFile, XmlReadMode.InferSchema);
          // Для каждой строки в таблице Users...
          foreach (DataRow rowUser in dsUsers.Tables["Users"].Rows)
                // ...проверить, совпадает ли имя с заданным.
                if (rowUser["name"].ToString() -- UserName)
                      return true;
    }
    catch
    {
          // В случае ошибки, вернуть false.
          return false;
    }
    // Если имя не найдено, вернуть false,
    return false;
}

private bool PasswordValid(string UserName, string Password)
{
    // Создать набор данных для загрузки XML-файла.
    DataSet dsUsers = new DataSet();
    // Обработать ошибку, возникающую в случае отсутствия файла.
    try
    {
          // Определить путь к файлу Users.xml.
          string strXMLFile = Server.MapPath(Request.ApplicationPath) +
            "\\Users.xml";
          // Прочитать файл.
          dsUsers.ReadXml(strXMLFile, XmlReadMode.InferSchema);
          // Для каждой строки в таблице Users...
          foreach (DataRow rowUser in dsUsers.Tables["Users"].Rows)
                // ...проверить, совпадает ли имя с заданным.
                if (rowUser["name"].ToString() == UserName)
                      if (rowUser["password"].ToString() ==
                        FormsAuthentication.HashPasswordForStoringInConfigFile
                        (Password, "SHA1"))
                            return true;
    }
    catch
    {
          // В случае ошибки вернуть false.
```

```
            return false;
        }
        // В случае ошибки вернуть false,
        return false;
}
```

Функция *PasswordValid*, показанная выше, **использует** метод *HashPasswordForStoringInConfigFile*, поскольку **пароли хранятся в зашифрованном виде. Тот же метод** используется функцией *AddUser* для **сохранения пароля в** файле Users.xml, **как показано далее.**

Visual Basic .NET

```
Private Sub butAddUser_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butAddUser.Click
    If AddUser(txtUserName.Text, txtPassword.Text) Then
        ' Уведомить об успешном завершении.
        lblStatus.Text = "User added. Click Sign On to continue."
        ' Скрыть кнопку,
        butAddUser.Visible - False
    Else
        ' Уведомить о неудаче.
        lblStatus.Text = "User could not be added. Choose a different name."
    End If
End Sub

Private Function AddUser(ByVal UserName As String, ByVal Password As String)
    ' Если такой пользователь уже существует, вернуть False и выйти из функции.
    If UserExists(UserName) Then
        Return False
        Exit Function
    End If
    ' в противном случае добавить сведения о пользователе в XML-файл.
    Din dsUsers As New DataSet()
    ' Обработать ошибку, возникающую в случае отсутствия файла.
    Try
        ' Определить путь к файлу Users.xml.
        Dim strXMLFile As String = Server.MapPath(Request.ApplicationPath) &_
          "\Users.xml"
        ' Прочитать файл.
        dsUsers.ReadXml(strXMLFile, XmlReadMode.InferSchema)
        ' Добавить строку.
        Dim rowUser As DataRow - dsUsers.Tables("users").NewRow()
        ' Записать имя пользователя.
        rowUser("name") = UserName
        ' Записать пароль(зашифрованный).
        rowUser("password") = _
          FormsAuthentication.HashPasswordForStoringInConfigFile _
         (Password, "SHA1")
        ' Добавить строку
```

```
        dsUsers.Tables("users").Rows.Add(rowUser)
        ' Записать набор данных.
        dsUsers.WriteXml(strXMLFile)
        Return True
    Catch
        ' В случае ошибки вернуть False.
        Return False
    End Try
End Function
```

Visual C#

```
private void butAddUser_Click(object sender, System.EventArgs e)
{
   if (AddUser(txtUserName.Text, txtPassword.Text))
   {
        // Уведомить об успешном завершении.
        lblStatus.Text = "User added. Click Sign On to continue.";
        // Скрыть кнопку.
        butAddUser.Visible = false;
   }
   else
        // Уведомить о неудаче.
        lblStatus.Text = "User could not be added. Choose a different name.";
}

private bool AddUser(string UserName, string Password)
{
   // Если такой пользователь уже существует, вернуть False и выйти из функции.
   if (UserExists(UserName))
   {
        return false;
   }
   // в противном случае добавить сведения о пользователе а XML-файл,
   DataSet dsUsers = new DataSet();
   // Обработать ошибку, возникающую в случае отсутствия файла.
   try
   {
        // Определить путь к файлу Users.xml.
        string strXMLFile = Server.MapPath(Request.ApplicationPath) +
          "\\Users.xml";
        // Прочитать файл.
        dsUsers.ReadXml(strXMLFile, XmlReadMode.InferSchema);
        // Добавить строку.
        DataRow rowUser = dsUsers.Tables["users"].NewRow();
        // Записать имя пользователя.
        rowUser["name"] = UserName;
```

```
            // Записать пароль(зашифрованный).
            rowUser["password"] =
                  FormsAuthentication.HashPasswordForStoringInConfigFile
                   (Password, "SHA1");
            // Добавить строку.
            dsUsers.Tables["users"].Rows.Add(rowUser);
            // Записать набор данных.
            dsUsers.WriteXml(strXMLFile);
            return true;
      }
      catch
      {
            // В случае ошибки вернуть False.
            return false;
      }
}
```

Упражнение 3. Отображение сведений о пользователе



Сейчас вы создадите Web-форму Userlnfo.aspx, отображающую имя и сведения об аутентификации пользователя при помощи объекта *Identity* класса *User*. Кроме того, вы создадите HTML-страницу, отображаемую при отказе в доступе. Поскольку по умолчанию запросы HTML-страниц не аутентифицируются Web-приложением, эта страница будет доступна даже пользователям, не прошедшим аутентификацию.

Создайте Web-форму Userlnfo.aspx и сделайте ее начальной страницей приложения. Пользователи, не прошедшие аутентификацию, направляются на Web-форму LogIn aspx, поскольку она задана в файле Web.config как входная страница, После аутентификации пользователи направляются на начальную страницу приложения вызовом метода *RedirectFromLoginPage*.

На странице Userlnfo.aspx находится таблица, в которой отображаются имя пользователя, сведения об его аутентификации, а также кнопка для выхода из приложения. Определение таблицы (см. пример HTML далее) содержит элементы **, выступающие в роли полей для подстановки реальных сведений о пользователях:

```
<H2>Forms Authorization</H2>
<TABLE id="tblUser" borderColor="black" cellSpacing="1" cellPadding="1"
width="500" bgColor="aliceblue" border="1">
 <TR>
     <TD>Authenticated</TD>
     <TD><SPAN id="spnAuthenticated" runat="server"></SPAN></TD>
 </TR>
 <TR>
     <TD>User name</TD>
     <TD><SPAN id="spnUserName"runat="server"></SPAN></TD>
 </TR>
 <TR>
     <TD>Authentication type</TD>
```

```
    <TD><SPAN id="spnAuthenticationType" runat="server"></SPAN></TD>
  </TR>
</TABLE>
<P>
<asp:Button id="butSignOut" runat="server" Text="Sign Out"></asp:Button>
</P>
```

Обработчик страничного события *Page_Load* извлекает сведения о пользователе и отображает их, как показано далее.

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
        ' Отобразить сведения об аутентификации пользователя.
        spnAuthenticated.InnerText = User.Identity.IsAuthenticated()
        spnUserName.InnerText = User.Identity.Name()
        spnAuthenticationType.InnerText = User.Identity.AuthenticationType()
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
   // Отобразить сведения об аутентификации пользователя.
   spnAuthenticated.InnerText = User.Identity.IsAuthenticated.ToString();
   spnUserName.InnerText = User.Identity.Name;
   spnAuthenticationType.InnerText = User.Identity.AuthenticationType;
}
```

Обработчик события *butSignOut_Click* завершает работу пользователя с приложением с помощью метода *SignOut*, удаляя с пользовательского компьютера файл cookie с подтверждением аутентификации. После это пользователю придется снова пройти аутентификацию, чтобы получить доступ к Web-приложению,

Visual Basic .NET

```
Private Sub butSignOut_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butSignOut.Click
        ' Удалить файл cookie, подтверждающий аутентификацию.
        FormsAuthentication.SignOut()
        ' Направить пользователя на входную страницу.
        Response.Redirect("UserInfo.aspx")
End Sub
```

Visual C#

```
private void butSignOut_Click(object sender, System.EventArgs e)
{
    // Удалить файл cookie, подтверждающий аутентификацию.
   FormsAuthentication.SignOut();
   // Направить пользователя на входную страницу.
```

```
    Response.Redirect("UserInfo.aspx");
}
```

Создав Web-форму UserInfo.aspx, добавьте к приложению HTML-страницу Denied.htm — на эту страницу Web-форма LogIn направит пользователя после третьей неудачной попытки ввода имени и пароля. Страница Denied.htm отображает сообщение, показанное в следующем HTML-коде:

```
<h2>Access Denied</h2>
<p>You must enter a valid user name and password to use this application,
</p>
```

Упражнение 4. Усложненные задания



В предыдущих упражнениях мы разбирали реализацию аутентификации при помощи Forms шаг за шагом. Чтобы изучить другие методы аутентификации, самостоятельно поэкспериментируйте с параметрами файла Web.config (см. пример в упражнении 1), определяющими режим аутентификации приложения.

- Чтобы включить аутентификацию средствами Windows, закомментируйте существующий элемент <*authentication*> и добавьте к файлу Web.config следующий элемент:

  ```
  <authentication mode="Windows" />
  ```

- Чтобы назначить аутентификацию с помошью службы Passport, установите пакет Passport SDK и модифицируйте элемент <*authentication*> файла Web.config, как показано ниже:

  ```
  <authentication mode="Passport" />
  ```

В завершение, чтобы набраться опыта, попробуйте получить серверный сертификат от какого-нибудь ЦС (например, от VeriSign) и использовать его для защиты данных приложения при пересылке через Интернет. Для создания защищенных подключений используйте протокол HTTPS (см. занятие 5).

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. Какой режим аутентификации ASP.NET лучше всего подходит для идентификации и авторизации пользователей корпоративной сети?

2. Чем отличаются списки со сведениями о пользователях из файла Web.config, применяемые для аутентификации средствами Windows и аутентификации с помощью форм?

3. Как заставить всех пользователей проходить аутентификацию с помощью файла Web.config, независимо от режима аутентификации ASP.NET?

4. Каким образом использование протокола SSL обеспечивает безопасность Web-приложения?

5. Как инициировать и завершить взаимодействие, защищенное протоколом SSL?

ГЛАВА 9

Компоновка и развертывание Web-приложений

Занятие 1. Компоновка Web-приложения 388

Занятие 2. Развертывание Web-приложения 393

Занятие 3. Сопровождение приложения после развертывания 400

Занятие 4. Развертывание приложения на нескольких серверах 416

В этой главе

В этой главе рассказано, как публиковать готовые Web-приложения на Web-сервере, а также как наблюдать за работой приложения и поддерживать его производительность на уровне, соответствующем текущим потребностям пользователей.

Прежде всего

Для изучения материалов этой главы необходимы:

- доступ к Web-серверу. Это может быть сервер поставщика услуг Web-хостинга либо сервер, работающий в вашей локальной сети;
- хотя бы одно готовое Web-приложение, созданное в одном из практикумов;
- умение работать с Microsoft Internet Information Services (IIS) и Microsoft Management Console (MMC);
- Microsoft Visual Studio .NET Professional, Microsoft Visual Studio .NET Enterprise Developer или Microsoft Visual Studio .NET (только для последнего занятия этой главы).

Занятие 1. Компоновка Web-приложения

В простейшем случае Visual Studio .NET выполняет компоновку приложения автоматически. В действительности это происходит каждый раз, когда вы выбираете в меню команду Run в процессе разработки или отладки приложения.

Прежде чем приступать к компоновке приложения, готового к развертыванию, следует:

1. задать параметры компоновки;
2. определить данные, идентифицирующие приложение;
3. сконфигурировать выбранное приложение.

На этом занятии вы научитесь выполнять все эти действия и подготавливать Web-приложение к развертыванию.

Изучив материал этого занятия, вы сможете:

- ✓ скомпоновать окончательную версию Web-приложения;
- ✓ добавлять к сборке приложения имя автора, название компании, описание, а также сведения об авторских правах и версии приложения;
- ✓ устанавливать параметры конфигурации приложения в файле Web.config.

Продолжительность занятия — около 10 минут.

Установка параметров компоновки

Параметр компоновки *debug* позволяет получить отладочную версию Web-приложения, а параметр *release* — окончательную версию.

Чтобы изменить параметры компоновки, выполните следующие действия.

1. Из пункта меню Build выберите команду Configuration Manager — Visual Studio .NET откроет диалоговое окно Configuration Manager, показанное на рис. 9-1.

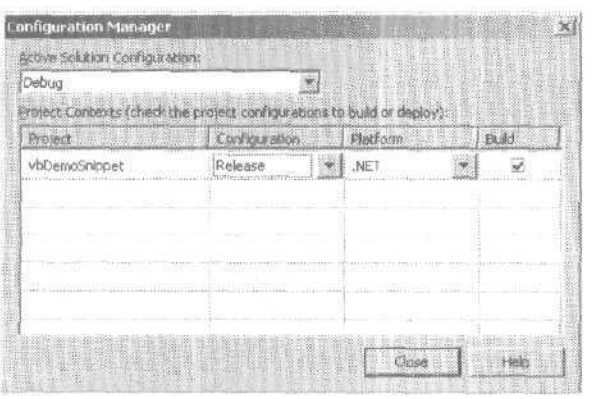


Рис. 9-1. Диалоговое окно Configuration Manager

2. Задайте параметры компоновки и щелкните OK.
3. Откройте файл Web.config проекта и задайте для элемента *compilation* значение, соответствующее параметрам компоновки, выбранным в пункте 2. Например, следующее значение элемента *compilation* используется для компоновки окончательной версии приложения:

```
<compilation defaultLanguage="vb" debug="false" ,/>
```

Если задана компоновка отладочной версии приложения (debug), генерируется файл базы данных программы (.pdb-файл) со сведениями о символах, используемых в приложении при компиляции проекта. При помощи этой базы данных Visual Studio .NET отслеживает значения переменных, устанавливает точки прерывания и вычисляет значения членов класса *Debug*.

Если выбрана компоновка *окончательной* (release) версии, файл базы данных не генерируется. В результате члены класса *Debug* оказываются недоступными, а точки прерывания игнорируются. Окончательная версия приложения компонуется быстрее, чем отладочная, поскольку при этом не требуется мониторинг дополнительных данных.

Параметры файла Web.config должны совпадать с параметрами компоновки, заданными для приложения. Хотя бессмысленно назначать в одном месте компоновку отладочной версии приложения, а в другом — окончательной, Visual Studio .NET не согласует эти параметры автоматически.

Ввод сведений о приложении

Перед развертыванием приложения рекомендуется снабдить его описанием, состоящим из названия приложения, имени автора и номера версии. Windows будет отображать эти сведения при просмотре свойств сборки (.dll-файла) приложения (рис. 9-2).

Чтобы задать сведения, идентифицирующие приложение, откройте файл AssemblyInfo и определите значения для соответствующих атрибутов сборки. На рис. 9-2 показаны атрибуты, определяющие название продукта и сведения о его версии:

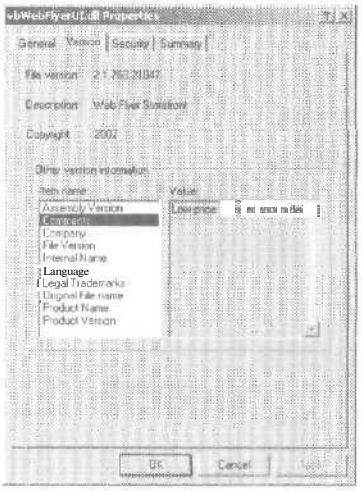


Рис. 9-2. Свойства сборки

Visual Basic .NET

```
Imports System.Reflection
Imports System.Runtime.InteropServices

' Общие сведения о сборке определяются при помощи следующих  атрибутов.
' Чтобы изменить сведения о сборке, измените
' значения соответствующих атрибутов,
```

```
' Обратите внимание на следующие атрибуты сборки:

<Assembly: AssemblyTitle("Web Flyer Storefront")>
<Assembly: AssemblyDescription("Low-price reservation finder.")>
<Assembly: AssemblyCompany("Contoso Ltd. ")>
<Assembly: AssemblyProduct("Web Flyer")>
<Assembly: AssemblyCopyright("2002")>
<Assembly: AssemblyTrademark("Web Flyer is a trademark of Contoso Ltd.")>
<Assembly: CLSCompliant(True)>

' Следующий GUID является идентификатором библиотеки типов;
' он необходим, если этот проект должен быть доступен для COM.

<Assembly: Guid("CE84F243-D7D9-4B7E-B43B-520A0D6B9B30")>

' Сведения о версии сборки состоят из следующих четырех значений:
'
'       Старшего номера версии
'       Младшего номера версии
'       Номера компоновки
'       Номера ревизии
'
' Можно определить все эти значения, либо использовать для ' номеров компоновки
и ревизии значения по умолчанию, заменив ' их знаком '*':

<Assembly: AssemblyVersion("2.1.*")>
```

Visual C#

```
using System.Reflection;
using System.Runtime.CompilerServices;

// Общие сведения о сборке определяются при помощи следующих
//  атрибутов.
// Чтобы изменить сведения о сборке, измените
// значения соответствующих атрибутов.

[assembly: AssemblyTitle("Web Flyer Storefront")]
[assembly: AssemblyDescription("Low-price reservation finder.")]
[assembly: AssemblyConfiguration("")]
[assembly:AssemblyCompanyC'ContosoLtd.")]
[assembly: AssemblyProduct("Web Flyer")]
[assembly: AssemblyCopyright("2002")]
[assembly: AssemblyTrademark("Web Flyer is a trademark of Contoso Ltd.")]
[assembly: AssemblyCulture("")]
```

```
// Сведения о версии сборки состоят из следующих четырех значений:
//
//      Старшего номера версии
//      Младшего номера версии
//      Номера компоновки
//      Номера ревизии
//
// Можно определить все эти значения, либо использовать для номеров
// компоновки и ревизии значения по умолчанию, заменив их знаком '*':

[assembly: AssemblyVersion("2.1.*")]
```

Следующие фрагменты файла AssemblyInfo генерируются автоматически:

- атрибут *Guid*, который Visual Studio .NET генерирует, чтобы идентифицировать сборку для компонентов COM. Значение этого атрибута следует изменять, только когда из-за удаления или существенной модификации открытых объектов и членов, используемых компонентами COM, сборка оказалась несовместимой со своими прежними версиями;
- атрибут *Assembly Version* автоматически обновляет номер компоновки, если в номере версии сборки он задан звездочкой (*). При необходимости можно добавлять уровни нумерации версий или вовсе удалить номер компоновки,

Конфигурирование приложения

Исполнение Web-приложений определяется при помощи текстовых конфигурационных файлов (эти файлы имеют расширение .config). Конфигурационный файл Web-приложения (Web.config) находится в его корневом каталоге и заменяет некоторые параметры, заданные в следующих файлах:

- **Machine.config** (располагается в каталоге **Windows\Microsoft.NET\Framework*номер_версии*\config).** Определяет базовую конфигурацию для всех сборок .NET, исполняемых на сервере;
- Web.config **(располагается в корневом** каталоге IIS). Определяет базовую конфигурацию для всех Web-приложений. Параметры, заданные в этом файле, приоритетнее тех, что заданы в файле Machine.config;
- все файлы Web.config, **размешенные в родительских каталогах текущего приложения.** Приложение «наследует» параметры всех файлов Web.config, расположенных в каталогах, составляющих путь к приложению.

Для управления Web-приложением в файле Web.config используются элементы XML, описанные в таблице 9-1. Поскольку эти элементы определяются с использованием синтаксиса XML, они чувствительны к регистру, то есть должны вводится именно в том виде, в котором они показаны в таблице.

После внесения изменений в файл Web.config IIS автоматически перезапускает приложение и применяет внесенные изменения. Побочный эффект перезапуска приложения — сброс текущих переменных состояния *Application* и *Session* — может негативно отразиться на пользователях.

По соображениям безопасности доступ из браузера к файлу Web.config запрещен. Пользователь, запросивший с Web-сайта файл Web.config, получает сообщение об отказе в доступе.

Таблица 9-1. Атрибуты Web.config

Элемент	Назначение	Где искать дополнительные сведения
compilation	Устанавливает тип компоновки в *debug* или *release*	Раздел «Установка параметров компоновки» этого занятия
customErrors	Отображает пользовательские страницы с сообщениями об ошибках при получении определенных кодов состояния HTTP	Глава 6
authentication	Устанавливает тип аутентификации	Глава 8
authorization	Содержит список имен пользователей или ролей, обладающих правом на исполнение этого приложения	Глава 8
trace	Включает трассировку для диагностики ошибок или оптимизации приложения	Глава 6
sessionState	Определяет, как ASP.NET хранит переменные состояния *Session*	Занятие 3 этой главы; глава 3
globalization	Устанавливает сведения о культуре для локализованных приложений	Глава 15

Занятие 2. Развертывание Web-приложения

Под *развертыванием* (deploying) Web-приложения понимают его установку на сервер, для работы на котором оно предназначено. Поскольку Web-приложения работают на серверах, обычно им не нужна установочная программа. Этим Web-приложения существенно отличаются от приложений Windows, которые распространяются на компакт-дисках или других носителях вместе с программой установки, копирующей и регистрирующей компоненты приложения на пользовательском компьютере.

В этом занятии вы научитесь развертывать готовые Web-приложения на собственном сервере или на сервере провайдера услуг хостинга. Вы также узнаете, какие параметры IIS и системы безопасности необходимо определить для нормальной работы Web-приложений.

Изучив материал этого занятия, вы сможете:

- ✓ загрузить готовое Web-приложение на сервер службы хостинга;
- ✓ создать виртуальный каталог IIS для развертывания Web-приложения на своем сервере;
- ✓ скопировать Web-приложение в виртуальный каталог собственного сервера;
- ✓ установить компоненты COM и .NET, необходимые Web-приложению;
- ✓ назначить права доступа для учетной записи ASPNET.

Продолжительность занятия - около 15 минут.

Загрузка приложений на сервер провайдера

Если у вас есть учетная запись службы хостинга приложений ASP.NET (регистрация в службе хостинга описана в главе I), вы можете установить свое Web-приложение на сервер службы хостинга прямо из Visual Studio .NET.

Чтобы установить свое приложение на сервере службы хостинга из Visual Studio . NET, выполните следующие действия.

1. Откройте проект, который вы хотите установить на сервер провайдера из Visual Studio .NET.
2. На начальной странице Visual Studio .NET щелкните ссылку Web Hosting.
3. На странице Web Hosting щелкните вкладку Premier Providers или Additional Providers.
4. Найдите на странице со списком провайдеров секцию, в которой указан ваш провайдер, и щелкните Upload Directly To Your Account — Visual Studio .NET откроет загрузочную страницу вашего провайдера (рис. 9-3).
5. Следуйте указанным на странице инструкциям (они могут отличаться в разных службах хостинга).

Развертывание приложения на собственном сервере

Если ваше приложение предназначено для работы в локальной сети или у вас есть свой Web-сервер, можно установить приложение на собственном сервере, выполнив следующие действия.

1. Подготовьте виртуальный каталог приложения при помощи IIS.
2. Скопируйте Web-приложение в созданный виртуальный каталог.
3. Установите в кэш глобальных сборок сервера все разделяемые .NET-компоненты, используемые приложением.
4. Настройте разрешения системы безопасности на сервере, чтобы открыть доступ к необходимым ресурсам для приложения.

В следующих разделах эти действия описаны более подробно.

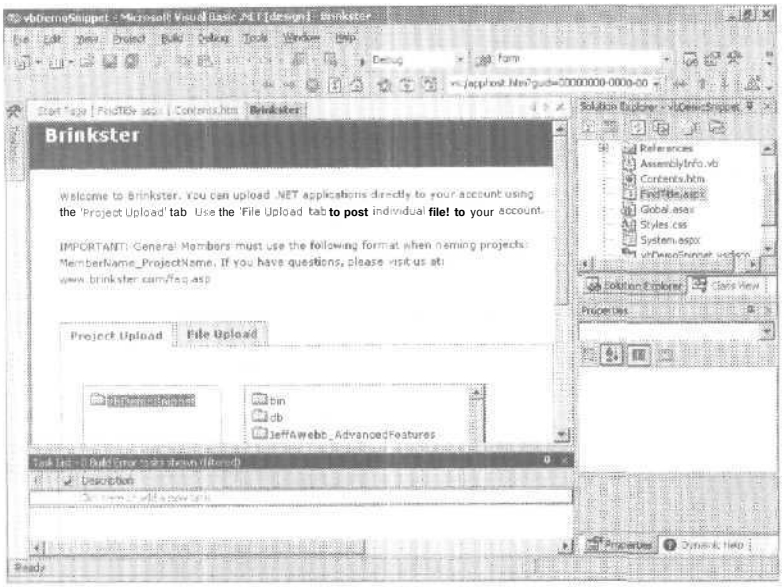


Рис. 9-3. Загрузка Web-приложения

Подготовка виртуального каталога

Web-приложения ASP.NET работают под управлением IIS, поэтому перед развертыванием на сервере Web-приложений необходимо установить на сервер I1S и .NET Framework. Для работы ASP.NET требуются следующие компоненты:

- IIS 5.0 или выше;
- .NET Framework той же версии, что применялась при компиляции сборки Web-приложения.

Необходимые серверные компоненты устанавливают с дистрибутива Visual Studio .NET. Убедившись, что установка выполнена корректно, создайте на сервере виртуальный каталог приложения, выполнив следующие действия.

1. С помощью Проводника создайте на сервере физический каталог для хранения приложения.
2. В IIS запустите мастер Virtual Directory Creation, чтобы создать новый виртуальный каталог на основе физического каталога, созданного на этапе 1 (подробное описание работы с этим мастером см. в главе 2).

Копирование приложения

Границы Web-придожения определяются структурой его каталогов, то есть область видимости приложения начинается в его корневом каталоге и включает в себя все вложенные каталоги корневого каталога. Таким образом, установить Web-приложение не сложнее, чем перетащить содержимое каталога, в котором приложение хранилось на этапе разработки, в физический каталог на сервере, который вы создали только что.

В принципе, структура каталогов Web-приложения, в которых размещаются файлы содержимого, может быть произвольной, но все же следует придерживаться некоторых правил:

- сборку (.dll-файл) приложения следует хранить в каталоге /bin;
- обычно начальная страница приложения называется Default.aspx или Default.htm.

Использование в приложении начальной страницы по умолчанию определяется параметрами IIS. Если страница по умолчанию задана, IIS отобразит ее, когда пользователь перейдет в каталог Web-приложения, не указав в запросе определенную страницу. Например, пользователь может запросить URL, указав лишь имя домена: *http://www.contoso.com*. Если страница по умолчанию определена, IIS вернет клиенту страницу *http://www.contoso. com/defaultaspx*.

Примечание Не путайте страницу по умолчанию IIS и начальную страницу приложения Web Forms в Visual Studio .NET. В Visual Studio .NET начальную страницу требуется задавать для каждого проекта. Это необходимо среде разработки, чтобы определить, какую страницу отображать первой во время отладки. Начальная страница, заданная в Visual Studio .NET, не влияет на страницу по умолчанию в IIS.

Установка разделяемых компонентов

Если ваше приложение использует некоторые компоненты, например нестандартные элементы управления, совместно с другими приложениями, работающими на этом сервере, эти компоненты также необходимо установить. Порядок установки различается в зависимости от типа компонента:

- компоненты COM копируют на сервер и регистрируют с помощью инструментов для регистрации компонентов COM;
- **.NET-компоненты** без строгого имени копируют в каталог /bin приложения;
- **.NET-компоненты** со строгим именем помещают в кэш глобальных сборок сервера.

Обычно для компонентов COM предусмотрена собственная установочная программа, позволяющая устанавливать и удалять их из системы. Если ее нет, можно скопировать компонент на сервер и зарегистрировать с помощью утилиты MFC RegSvr32.exe, например, так:

```
RegSvr32 MyComp.dll
```

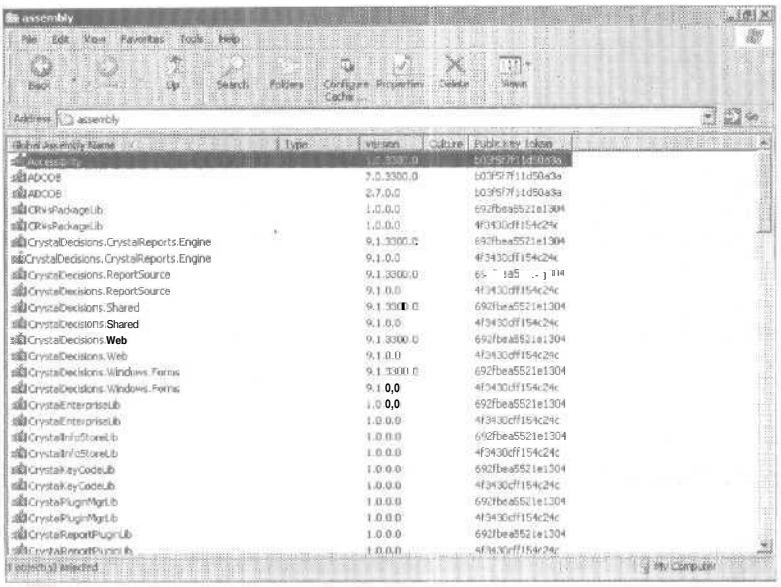
.NET-компоненты различаются по типу имен, которые могут быть строгими или нестрогими в зависимости от способа хранения имен в сборке. Уникальность нестрогих имен не гарантируется, поэтому при совместном использовании компонентов с нестрогими именами возможны конфликты. У строгих имен есть цифровая подпись и открытый ключ, которые делают конфликт невозможным. Кроме того, .NET-компоненты со строгими именами не способны вызывать неуправляемый код (такой, как компоненты COM), что исключает конфликты из-за зависимостей.

.NET-компоненты с нестрогими именами следует скопировать в каталог /bin каждого из Web-приложений, использующих эти компоненты, а .NET-компоненты со строгими именами можно копировать в кэш глобальных сборок сервера (рис. 9-4).

Кэш глобальных сборок (global assembly cache) — это особая папка каталога Windows, в которой хранятся разделяемые .NET-компоненты. Когда открывается эта палка, Explorer запускает расширение оболочки Windows под названием Assembly Cache Viewer (ShFusion.dll), показанное на рис. 9-4.

Чтобы установить .NET-компоненты со строгими именами, можно перетащить их в окно Assembly Cache Viewer или скопировать при помощи инструмента Global Assembly Cache (GacUtil.exe) следующим образом:

```
GacUtil -i MyServeControls.dll
```



Рис, 9-4. Просмотр .NET-компонентов в кэше глобальных сборок

Назначение прав доступа

По умолчанию процесс ASP.NET работает под учетной записью ASPNET, которая создается при установке .NET Framework. Эта учетная запись обладает ограниченными привилегиями, что может привести к ошибкам из-за отказа в доступе, если Web-приложение попытается записать или прочитать файл, расположенный за пределами его границ.

Если вашему приложению нужны дополнительные разрешения, выполните одно из следующих действий:

- предоставьте учетной записи ASPNET доступ к необходимым файлам (это возможно, только если на сервере используется файловая система NTFS);
- переместите учетную запись ASPNET в другую группу;
- при помощи олицетворения назначьте процессу приложения другую учетную запись.

Чтобы предоставить учетной записи ASPNET доступ к файлу или папке, выполните следующие действия.

1. В окне Explorer (Проводника) выберите из меню View (в русскоязычной версии Windows — Сервис) пункт Folder Options (Настройки вида папки) — Windows откроет диалоговое окно свойств папки.
2. Перейдите на вкладку View (Вид) и снимите флажок Use Simple File Sharing (Использовать простой общий доступ к файлам) в конце списка (рис. 9-5). Щелкните OK, чтобы внести изменения и закрыть диалоговое окно.
3. В окне Explorer щелкните правой кнопкой мыши файл или папку, к которым следует предоставить доступ, и выберите из контекстного меню команду Properties (Свойства). Windows откроет диалоговое окно свойств файла или папки.
4. Перейдите на вкладку Security (Доступ) и щелкните кнопку Add (Добавить), чтобы добавить учетную запись ASPNET к списку Group Or User Names (Группы или пользователи), как показано на рис. 9-6.
5. Пометьте флажками нужные разрешения. Щелкните OK, чтобы внести изменения и закрыть диалоговое окно.

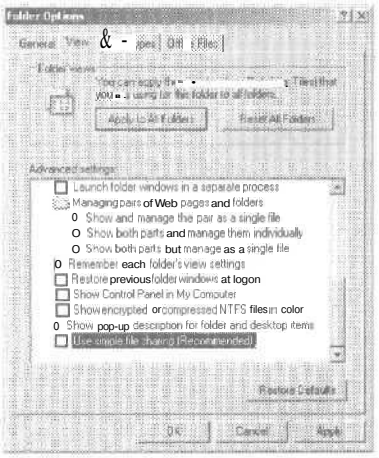


Рис. 9-5. Предоставление полного доступа к файлам

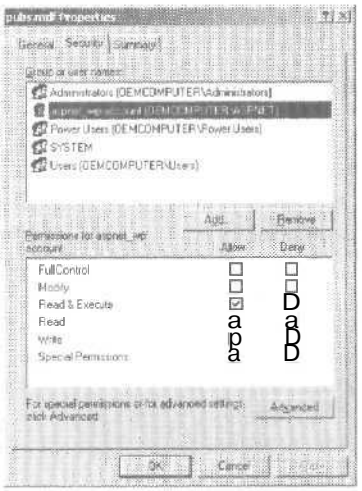


Рис. 9-6. Добавление учетной записи ASPNET

Чтобы поместить пользовательскую учетную запись ASPNET в другую группу, выполните следующие действия.

1. Из пункта меню Start (Пуск) выберите команду Administrative Tools\Computer Management (Администрирование\Управление компьютером). Windows откроет консоль Computer Management (Управление компьютером).
2. Выберите компонент Local Users And Groups (Локальные пользователи и группы), затем в списке слева выберите Users (Пользователи), как показано на рис. 9-7.
3. Щелкните правой кнопкой мыши пользовательскую учетную запись **ASPNET** и выберите из контекстного меню команду Properties (Свойства) — консоль Computer Management откроет окно свойств этой учетной записи (рис. 9-8).
4. Перейдите на вкладку Member Of и щелкните кнопку Add (Добавить), чтобы добавить учетную запись ASPNET к новой группе. Щелкните OK, чтобы внести изменения и закрыть диалоговые окна.

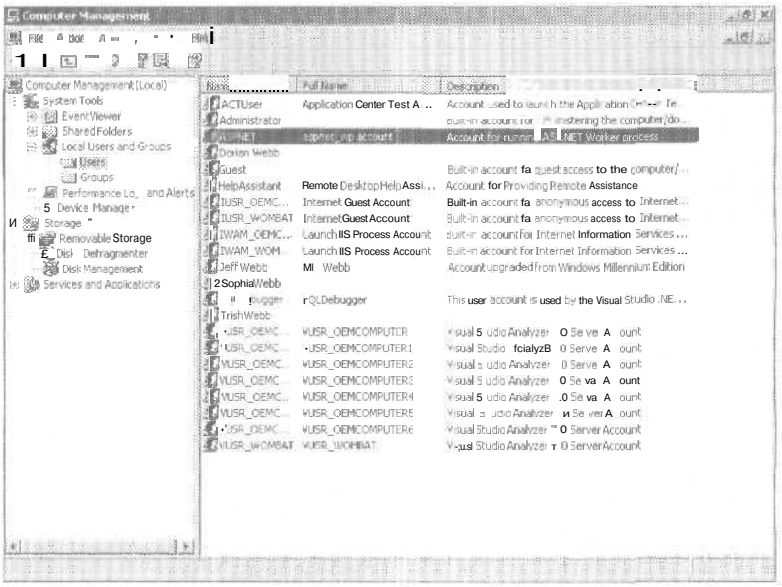


Рис. 9-7. Изменение учетной записи пользователя ASPNET

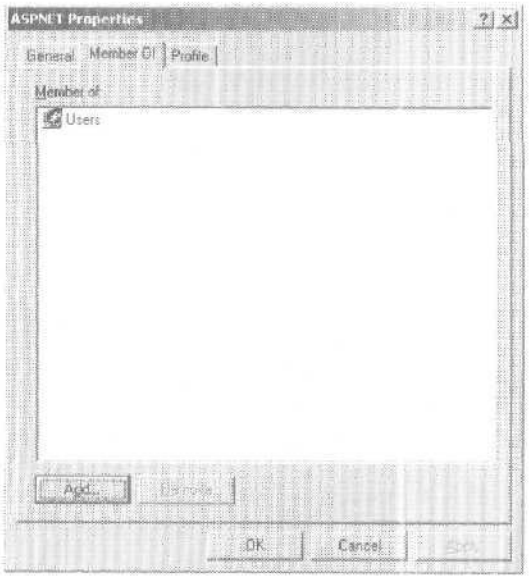


Рис. 9-8. Выбор группы для учетной записи ASPNET

В результате добавления учетной записи ASPNET к группе Administrators Web-приложение получит полный набор привилегий на сервере, но это угрожает безопасности, поскольку в этом случае внешние пользователи смогут сделать Web-приложение инструментом для взлома сервера.

Чтобы при помощи олицетворения назначить процессу ASP.NET другую учетную запись вместо ASPNET, воспользуйтесь атрибутом *impersonation* элемента *identity* из файла

Web.config. Например, следующий элемент назначает для приложения учетную запись пользователя WebFlyr:

```
<identity impersonate="true" name="localhost\WebFlyr" password="hpB14dQi" />
```

Если в приложении используется аутентификация средствами Windows, то при помощи олицетворения можно также передавать на сервер идентификационные данные аутентифицированных пользователей. Например, следующий фрагмент файла Web.config заставляет исполнять Web-приложение с разрешениями пользователя, зарегистрировавшегося для работы с приложением:

```
<!-- Использовать олицетворение. ->
<identity impersonate="true" />

<authorization>
<deny users="?" />                    <!- Сделать аутентификацию
                                        обязательной. ->
</authorization>
```

Дополнительные сведения об аутентификации пользователей см. в главе 8.

Занятие 3. Сопровождение приложения после развертывания

После развертывания Web-приложения необходимо следить за его работой на сервере, так как на этом этапе могут возникать самые разные проблемы, связанные как с числом и непредсказуемым поведением пользователей приложения, так и с атаками злоумышленников.

Таким образом, после развертывания приложение нуждается в сопровождении, что подразумевает:

- наблюдение за производительностЕ,ю, обнаружение неполадок и проблем с безопасностью;
- устранение любых сбоев по мере их обнаружения;
- настройку производительности приложения в соответствии с трафиком, генерируемым пользователями.

На этом занятии вы научитесь решать указанные задачи и познакомитесь с некоторыми методами регулирования производительности приложения.

Изучив материал этого занятия, вы сможете:

- ✓ использовать инструментальные средства Windows для мониторинга сбоев. работы системы безопасности и производительности приложения, работающего на сервере;
- ✓ создавать оповещения для уведомления о возникающих проблемах;
- ✓ отображать производительность приложения на диаграммах и отслеживать его активность с помощью журналов событий;
- ✓ устранять неполадки в развернутом приложении путем замены его сборки или отдельных файлов содержимого;
- ✓ определять способ управления процессами приложений ASP.NET при помощи параметров конфигурации сервера;
- ✓ настраивать приложение для повышения его производительности;
- ✓ оценивать влияние использования различных методик программирования и функций ASP.NET на быстродействие Web-приложений.

Продолжительность занятия — около 30 минут.

Наблюдение за работой сервера

После развертывания приложения необходимо следить за сервером, на котором оно работает. Это поможет обнаружить и устранить проблемы, связанные с безопасностью и быстродействием, а также другие непредвиденные ошибки. Для наблюдения за соответствующими событиями в Windows служат следующие оснастки MMC:

- Event Viewer (Просмотр событий) — содержит списоксобытий, генерированных приложением, подсистемой безопасности и другими системами компьютера в порядке их возникновения. Этот инструмент позволяет просматривать хронику реальных событий на сервере и получать сведения об отдельных событиях;
- Performance **Log and Alert (Журналы и оповещения производительности) —** позволяет создавать новые события, отображаемые оснасткой Просмотр событий, и отслеживать показания счетчиков событий, представляя собранные ими сведения в виде графика или отчета.

Более подробно использование этих инструментов описано в следующих разделах.

Просмотр событий

Оснастка консоли **MMC** под названием Просмотр событий позволяет отслеживать события, генерируемые приложением, подсистемой безопасности и другими компонентами системы по мере их возникновения.

Для вызова оснастки Просмотр событий достаточно выбрать из меню Start (Пуск) команду Administrative Tools\Event Viewer (Администрирование\Просмотр событий). Чтобы в Windows XP Professional добавить в меню пункт Администрирование, вызовите окно свойств кнопки Пуск и на вкладке Start Menu (Меню «Пуск») щелкните кнопку Customize (Настроить). Чтобы добавить в меню пункт Администрирование, установите соответствующий переключатель на вкладке Advanced (Дополнительно) диалогового окна Customize Start Menu (Настройка меню пуск). После этого можно вызвать оснастку Просмотр событий, щелкнув одноименный ярлык. В результате Windows откроет эту оснастку в окне консоли **MMC** (рис. 9-9).

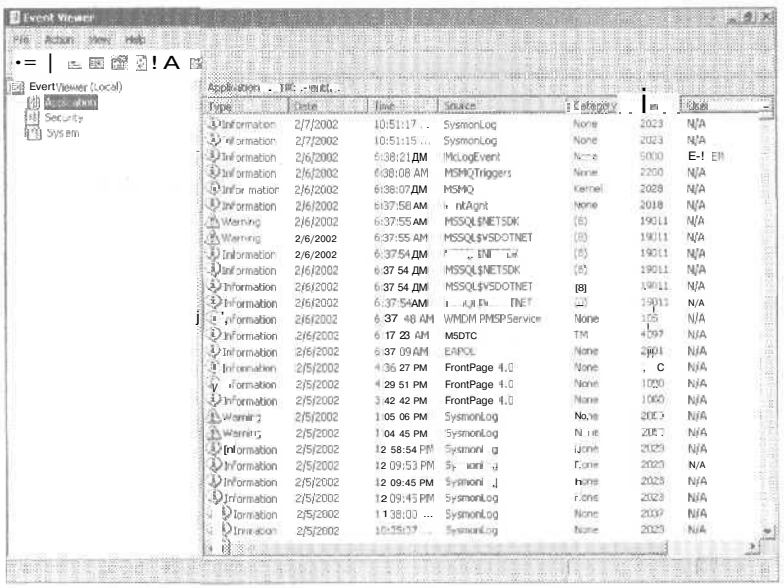


Рис. 9-9. Просмотр событий

Оснастка Event Viewer отображает три типа событий, записанных в журналы приложений, безопасности и системы: уведомления, предупреждения и ошибки.

Для просмотра сведений о некотором событии дважды щелкните его левой кнопкой — оснастка откроет окно свойств события (рис. 9-10).

Оснастка Event Viewer позволяет увидеть как общую картину происходящего на сервере в данный момент, так и подробные сведения об отдельном событии. Для создания новых типов событий, а также для мониторинга событий служит оснастка Performance Logs and Alerts (Журналы и оповещения производительности), о которой пойдет речь в следующем разделе.

Определение новых типов событий

Часто сведений, собираемых при помощи стандартных типов событий, недостаточно. Например, с их помощью вы не узнаете, когда число пользователей, работающих с Web-приложением, превысит 1000 или когда возникнет необработанное исключение Б Web-

приложении. Оснастка Performance Logs and Alerts (Журналы и оповещения производительности) позволяет добавлять подобные события к счетчикам и журналам трассировки.

Чтобы вызвать оснастку, выберите в меню Start (Пуск) команду Administrative Tools\ Performance Logs and Alerts (Администрирование\Журналы и оповещения производительности) — Windows откроет эту оснастку в окне консоли MMC (рис. 9-11).

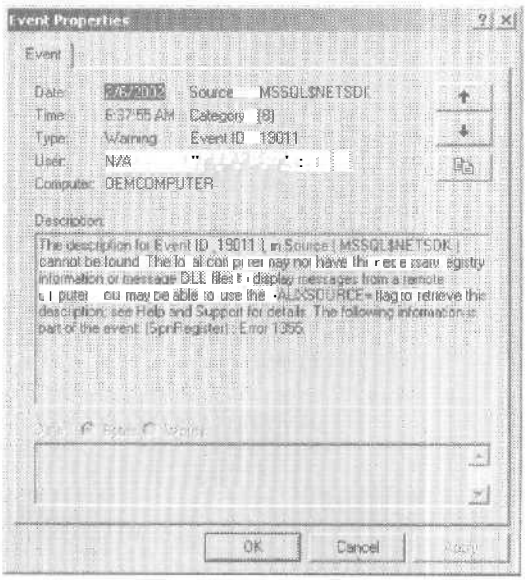


Рис. 9-10. Просмотр сведений о событии

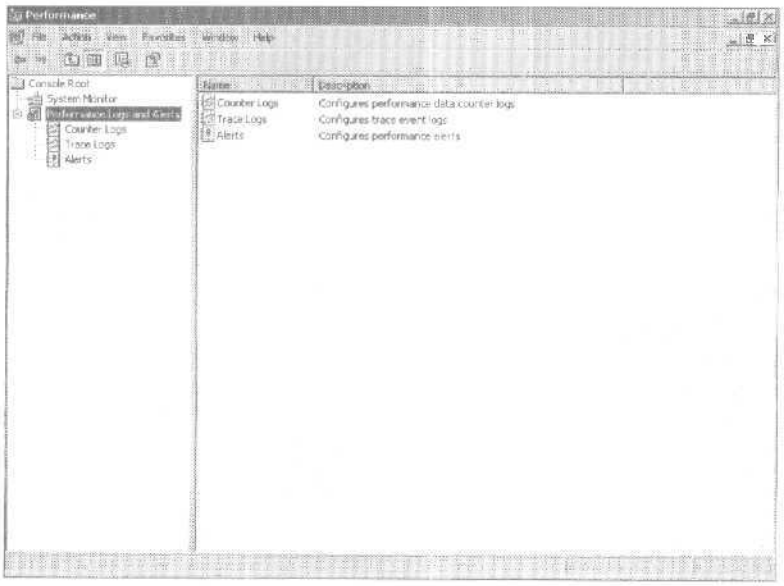


Рис. 9-11. Выбор событий для мониторинга

Оснастка Performance Logs and Alerts позволяет создавать файлы журналов для регистрации активности сервера и создания оповещений, генерирующих события, отображаемые оснасткой Event Viewer. Этот инструмент позволяет собирать самые разные сведения о сервере, исполняемых на нем приложениях и работающих с ними пользователях.

Лучше всего разбираться в принципах работы этой оснастки на практике. Для этого попробуйте самостоятельно определить событие для наблюдения за системой, выполнив следующие действия.

1. В левой панели окна Performance (Производительность) щелкните элемент Alerts (Оповещения) из списка Performance Logs And Alerts (Журналы и оповещения производительности).
2. Выберите из меню Action (Действие) команду New Alert Settings (Новые параметры оповещений) — оснастка откроет одноименное диалоговое окно (рис. 9-12).

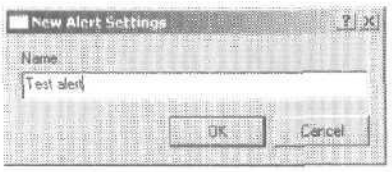


Рис. 9-12. Ввод имени нового оповещения

3. Введите имя оповещения и щелкните OK — оснастка откроет диалоговое окно свойств нового оповещения (рис. 9-13).

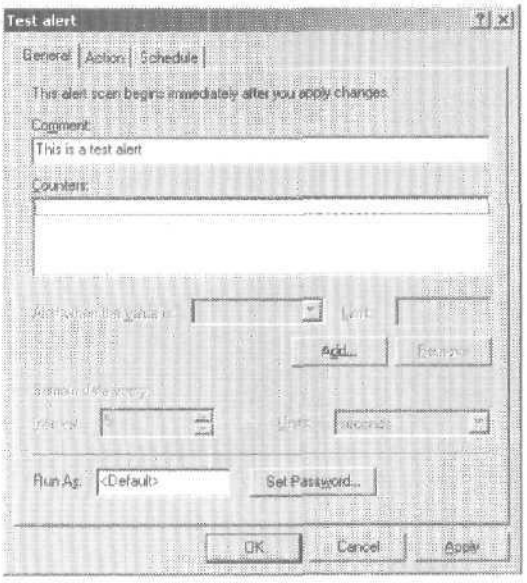


Рис. 9-13. Настройка оповещения

4. Добавьте событие для мониторинга, щелкнув кнопку Add (Добавить). Оснастка откроет диалоговое окно Add Counters (Добавить счетчики), показанное на рис. 9-14.
5. Из раскрывающегося списка Performance Objects (Объект) выберите устройство или приложение, за которым будет вестись наблюдение — оснастка отобразит список счетчиков для мониторинга выбранного объекта. Чтобы добавить выбранный счетчик к оповещению, щелкните кнопку Add. Закончив, щелкните кнопку Close (Закрыть).

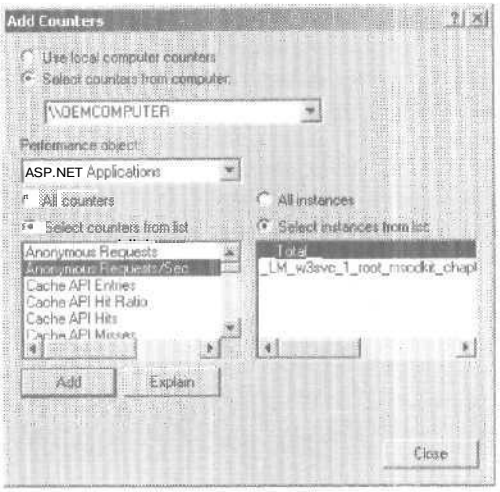


Рис. 9-14, Добавление счетчиков числа пользователей ASP.NET

6. Определите критерий для генерации оповещения. Для этого введите нужное число в поле Limit диалогового окна свойств оповещения.

   Согласно параметрам, показанным на рис. 9-15, оповещение будет сгенерировано, если число пользователей ASP.NET превысит 1000.

7. Перейдите на вкладку Action, чтобы определить действия, которые сервер должен предпринимать при получении оповещения. По умолчанию он регистрирует оповещения в форме событий, доступных через оснастку Event Viewer; кроме этого возможно переслать оповещение по электронной почте, либо записать его в журнал (рис. 9-16).
8. Закончив, щелкните OK — оснастка добавит оповещение к списку Alerts (Оповещения), как показано на рис. 9-17.

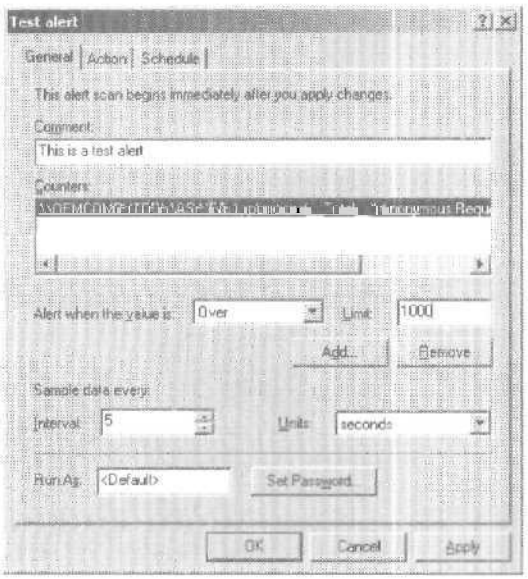


Рис. 9-15. Установка критерия для генерации оповещения

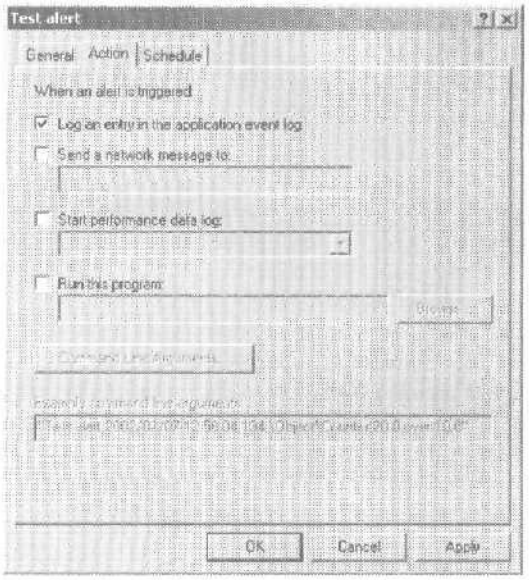


Рис. 9-16. Выбор действия сервера при получении оповещения

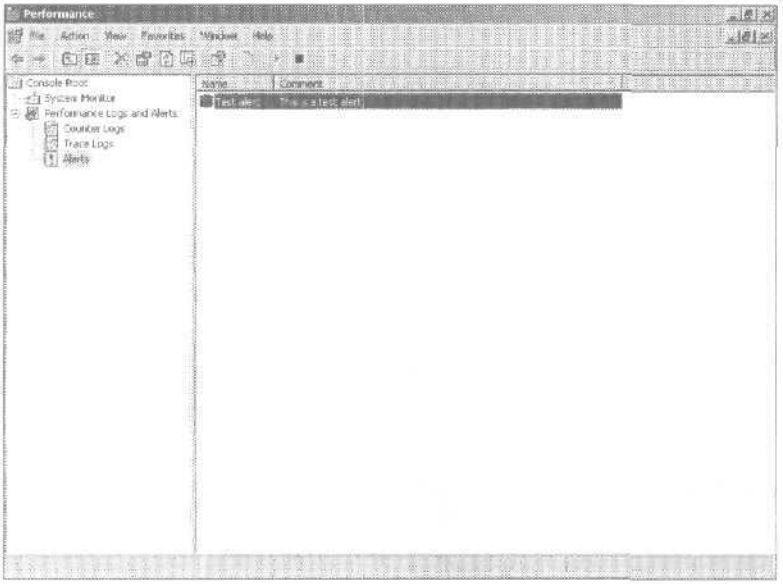


Рис. 9-17. Новое оповещение

По умолчанию оснастка немедленно начинает мониторинг нового оповещения. Чтобы остановить или начать мониторинг оповещения, щелкните его правой кнопкой и выберите из контекстного меню команду Stop или Start соответственно.

Windows генерирует оповещения каждый раз, когда выполняется условие, заданное для него в качестве критерия. Некоторые счетчики, например счетчики ошибок, не сбрасываются автоматически, поэтому значение такого счетчика никогда не падает ниже порога, однажды достигнув его. В результате оповещения, генерируемые на основе такого счетчи-

ка, будут создаваться регулярно, через определенные промежутки времени, поэтому следует быть осторожным при выборе ответных действий для таких оповещений. Если вы назначите отправку почтового сообщения в ответ на оповещение, генерируемое счетчиком ошибок, вы рискуете вызвать вал сообщений.

Создание журналов событий

Определение событий на основе новых оповещений годится для обнаружения отдельных неполадок и выявления снижения производительности, но для мониторинга этих параметров в течение длительного периода необходимо создать журнал событий. Журналы событий позволяют регистрировать самые разные события в течение некоторого времени и отображать собранные сведения в графическом или текстовом виде.

Чтобы создать журнал событий, выполните следующие действия.

1. В левой панели окна Performance (Производительность) выберите элемент Counter Logs (Журналы счетчиков) из списка Performance Logs And Alerts (Журналы и оповещения производительности).
2. Выберите из меню Action (Действие) команду New Log Settings (Новые параметры журнала) — оснастка откроет одноименное диалоговое окно (рис. 9-18).

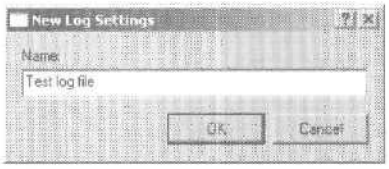


Рис. 9-18. Ввод имени файла журнала

3. Укажите имя журнала и щелкните OK — оснастка откроет диалоговое окно свойств файла журнала, показанное на рис. 9-19

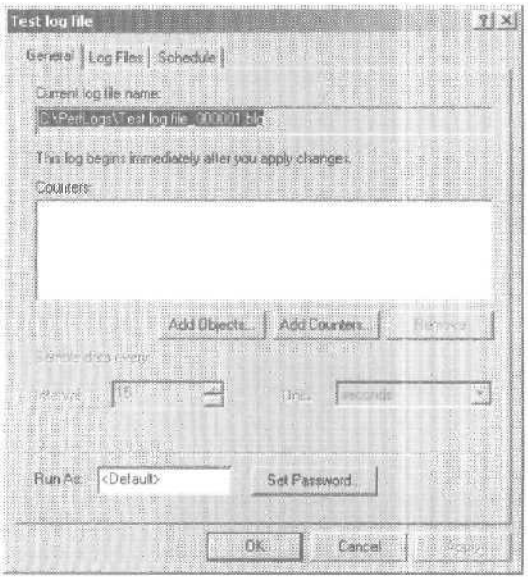


Рис. 9-19. Настройка журнала

4. При помощи кнопок Add Objects (Добавить объекты) или Add Counters (Добавить счетчики) добавьте элементы, которые необходимо отслеживать в журнале. В ответ на щелчок кнопки Add Counters оснастка откроет одноименное диалоговое окно (рис, 9-20). Счетчики, указанные в этом окне, работают так же, как счетчики для оповещений.

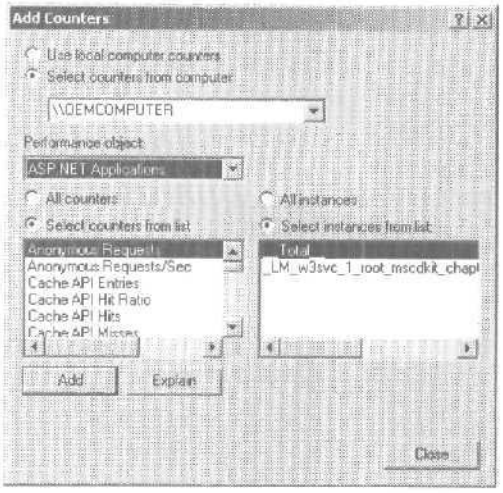


Рис. 9-20. Выбор счетчиков для мониторинга

5. Перейдите на вкладку Log Files (Файлы журнала), чтобы указать формат создаваемого файла журнала (рис. 9-21).
6. Файлы журналов можно сохранять в двоичном или текстовом формате либо в виде базы данных. Назначив формат, щелкните OK — оснастка добавит журнал к списку Counter Logs (рис. 9-22).

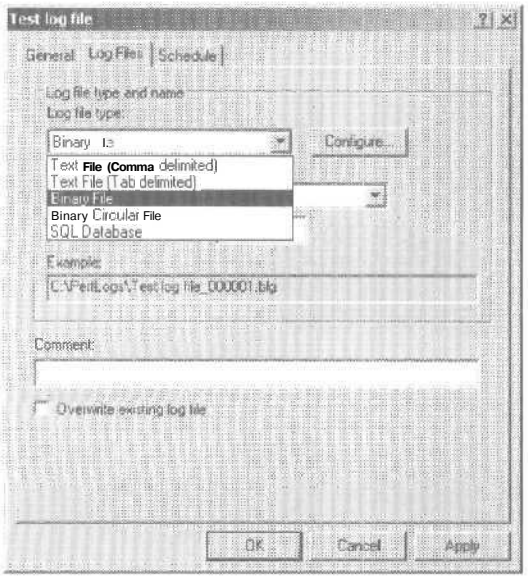


Рис. 9-21. Выбор формата файла **журнала**

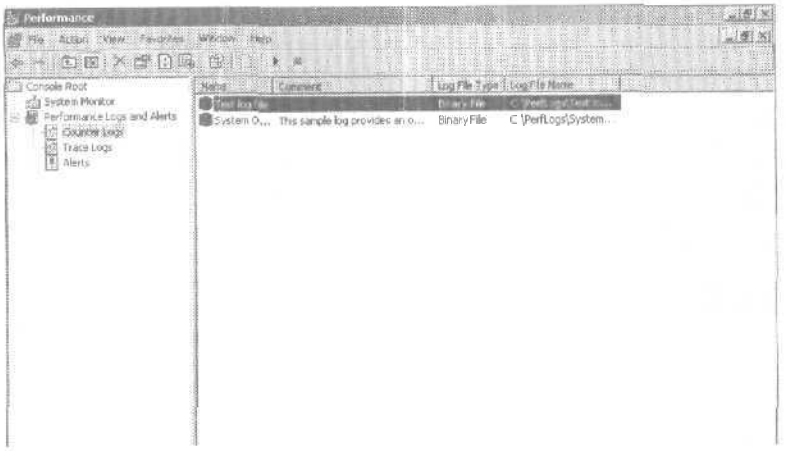


Рис. 9-22. Новый журнал

Как и для оповещений, оснастка начинает вести новый журнал немедленно. Чтобы остановить или начать ведение журнала, щелкните его правой кнопкой и выберите из контекстного меню команду Stop или Start соответственно.

Просмотр журналов событий

По умолчанию оснастка Performance Logs and Alerts сохраняет файлы журналов в каталоге PerfLogs. Для просмотра файла через оснастку выполните следующие действия.

1. Выберите из списка в левой панели окна Performance (Производительность) элемент System Monitor (Системный монитор) — оснастка покажет системный монитор с графиками текущих статистик производительности
2. Щелкните на панели инструментов кнопку View Log Data (Просмотр данных журнала) — оснастка откроет диалоговое окно свойств System Monitor (рис. 9-23).

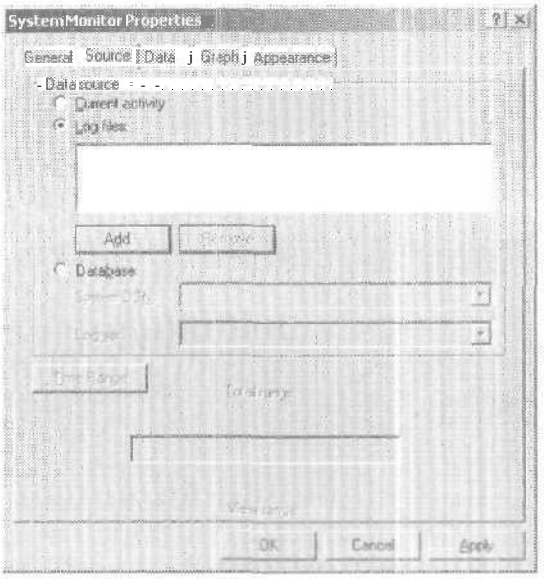


Рис. 9-23. Свойства System **Monitor**

3. На вкладке Source (Источник) установите переключатель Log Files и щелкните кнопку Add (добавить), чтобы выбрать файл, который следует открыть. Оснастка откроет диалоговое окно Select Log File (Выбор файла журнала), показанное на рис. 9-24.

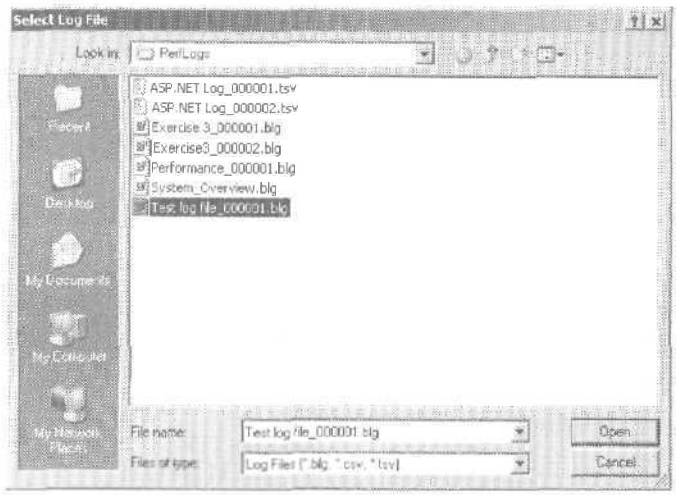


Рис. 9-24. Выбор файла журнала

4. Выберите в лапке PerfLogs нужный файл журнала и щелкните кнопку Open (Открыть), чтобы добавить журнал к System Monitor.
5. В диалоговом окне свойств System Monitor перейдите на вкладку Data, чтобы добавить содержимое журнала на график, как показано на рис. 9-25.

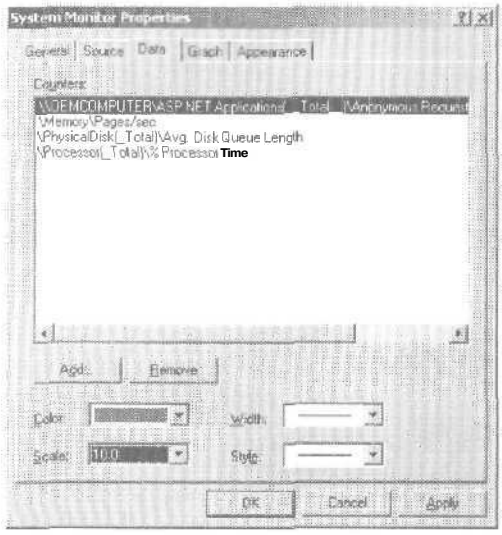


Рис. 9-25. Добавление элементов к графику

6. Щелкните OK, чтобы закрыть диалоговое окно свойств System Monitor — оснастка откроет файл журнала и поместит его содержимое на график (рис. 9-26).
7. Для просмотра данных в текстовом виде щелкните на панели инструментов кнопку View Report (Просмотр отчета) — System Monitor откроет вместо графика текстовый отчет (рис. 9-27).

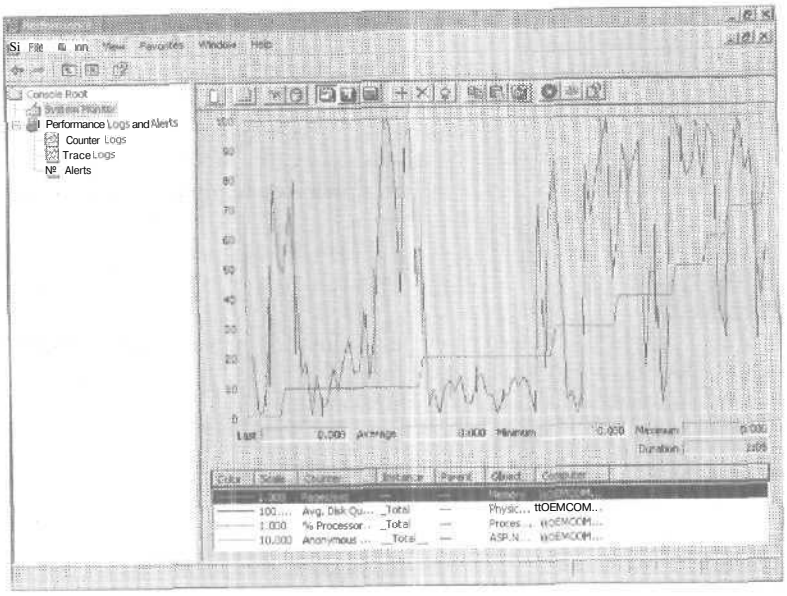


Рис. 9-26. Просмотр журнала в виде графика

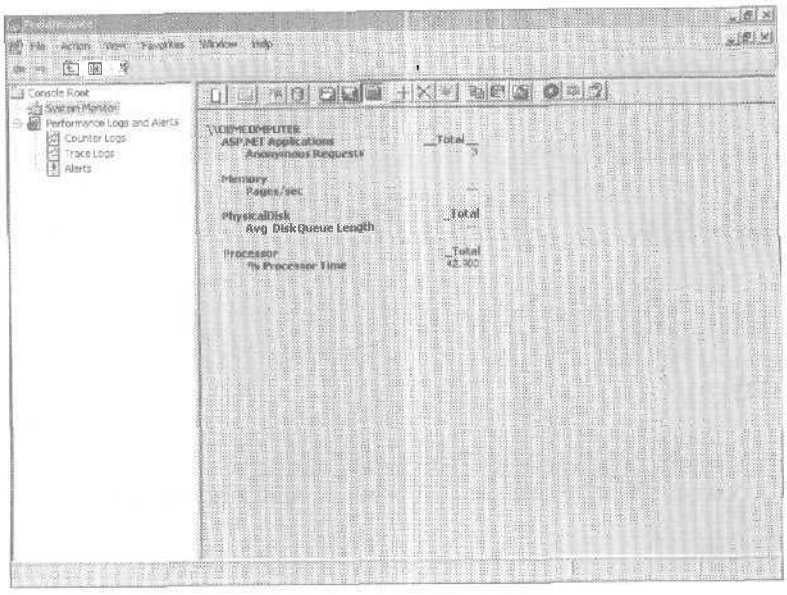


Рис. 9-27. **Просмотр журнала в виде текстового отчета**

Примечание Оснастки Event Viewer и Performance Logs and Alerts позволяют следить за «состоянием здоровья» сервера, обнаруживать ошибки приложений, выявлять конфликты из-за ресурсов и прогнозировать потребность в аппаратных ресурсах. Подробно о решении этих задач — в справочной системе соответствующей оснастки.

Восстановление приложения во время его эксплуатации

После развертывания Web-приложения можно заменять его файлы и устранять неполадки «на месте», то есть без перезапуска сервера или IIS.

Чтобы восстановить Web-приложение, скопируйте новую сборку (.dll-файл) и/или файлы содержимого (.aspx, .ascx и другие) в папку приложения на сервере. Обнаружив замену сборки, ASP.NET автоматически перезапустит приложение, при этом не требуется ни устанавливать, ни регистрировать сборку на сервере.

Web-приложения ASP.NET обладают ограниченной способностью к самовосстановлению после сбоев путем *рециркуляции процессов* (process recycling), то есть принудительного завершения и перезапуска рабочих процессов (aspnet_wp.exe), ставших неактивными либо занимающих слишком много ресурсов. Для управления рециркуляцией процессов ASP.NET служат атрибуты элемента *processModel*, определяемого в файле Machine.config (эти атрибуты выделены **полужирным шрифтом** в следующем примере):

```
<processModel
   enable="true"
   timeout="Infinite"
   idleTimeout="Infinite"
   shutdownTimeout="0:00:05"
   requestLimit="Infinite2"
   requestQueueLimit="5000"
   restartQueueLimit="10"
   memoryLimit="60"
   webGarden="false"
   cpuMask="0xffffffff"
   userName="machine"
   password="AutoGenerate"
   logLevel="Errors"
   clientConnectedCheck="0:00:05"
   comAuthenticationLevel="Connect"
   comImpersonationLevel="Impersonate"
   responseRestartDeadlockInterval="00:09:00"
   responseDeadlockInterval="00:03:00"
   maxWorkerThreads="25"
   maxIoThreads="25"
/>
```

Атрибуты элемента *processModel,* управляющие рециркуляцией процессов, описаны в таблице 9-2.

Таблица 9-2. Атрибуты, управляющие рециркуляцией процессов

Имя	Назначение
Timeout	Определяет период (чч:мм:сс), по истечении которого процесс завершается и перезапускается. В качестве превентивной меры выполняет автоматическую рециркуляцию процессов после обработки ими определенного количества запросов
ShutDownTimeOut	Задает время, отводимое на завершение процессов. Если процесс не завершится по истечении этого времени, это принудительно сделает система

Таблица 9-2. (*окончание*)

Имя	Назначение
RequestLimit	Определяет число запросов из очереди, после исполнения которых процесс завершается и перезапускается. Используется так же, как атрибут *timeout*
RestartQueueLimit	Задает длину очереди (в виде числа запросов), по достижении которой запрос перезапускается
memory Limit	Определяет максимальный объем физической памяти (в процентах), занятой процессом ASP.NET. Если процесс выделит больше памяти, он будет завершен, и вместо него запустится новый процесс. Этот атрибут позволяет предотвратить утечки памяти, ведущие к снижению производительности сервера
response Restart-DeadlocInterval	Задает период ожидания, по истечении которого процессы, заблокировавшие друг друга, перезапускаются. Обычно значение этого атрибута равно нескольким минутам, что предотвращает снижение производительности сервера из-за серьезных сбоев в приложениях
responseDead-lockInterval	Определяет время ожидания, по истечению которого перезапускаются процессы, заблокировавшие друг друга. Процесс перезапускается, если при наличии запросов в очереди ему не удается обработать их в течение заданного времени

Настройка развернутых приложений

У Web-приложений ASP.NET есть ряд параметров, позволяющих регулировать их производительность после развертывания приложений на сервере:

- для управления кэшированием страниц в оперативной памяти сервера используется Microsoft FrontPage Server Extensions;
- уровнем зашиты процессов приложения управляют при помощи соответствующих параметров приложения в IIS;
- атрибуты элемента *processModel,* определяемого в файле Machine.config, управляют числом потоков и тайм-аутами Web-приложений на сервере;
- для управления сохранением данных в переменных состояния *Session* предназначены атрибуты элемента *sessionState*, определяемого в файле Web.config.

Применение всех этих параметров описаны в следующих разделах.

Настройка кэширования

Для ускорения доступа к часто запрашиваемым документам Web-приложения ASP.NET держат их в памяти сервера. Хранение часто запрашиваемых элементов в оперативной памяти сервера называется *кэшированием* (caching).

Чтобы настроить кэширование для Web-приложения, выполните следующие действия.

1. Настройте параметры FrontPage Server Extensions для Web-приложения. Для этого:
   - откройте IIS и выберите Web-приложение. Далее щелкните в меню Action команду All Tasks\Configure FrontPage Server Extensions — IIS запустит мастер New Subweb Wizard;
   - следуйте указаниям мастера, для перехода между его страницами щелкайте кнопку Next.
2. Щелкните правой кнопкой Web-приложение в окне IIS и выберите из контекстного меню команду Properties — IIS откроет диалоговое окно свойств Web-приложения.
3. Перейдите на вкладку Server Extensions и выберите в списке Performance примерное число страниц для вашего Web-приложения (рис. 9-28).

4. FrontPage Server Extensions предоставляют различные наборы предопределенных параметров кэширования для Web-приложений в зависимости от числа составляющих их документов (.aspx- или .htm-файлов). Можно также установить собственные параметры кэширования, выбрав из списка Performance элемент Use Custom Settings, В этом случае IIS откроет диалоговое окно Performance (рис. 9-29). Выставите нужные параметры и щелкните OK, чтобы закрыть диалоговые окна, открытые ранее.

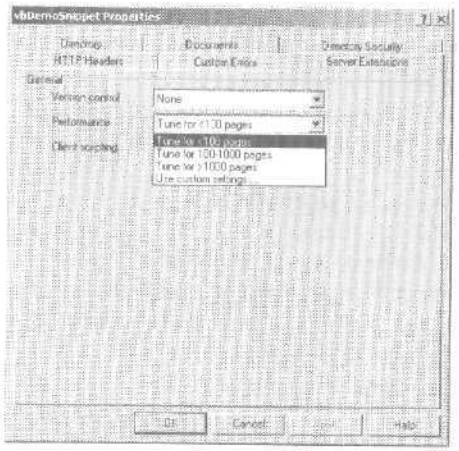


Рис. 9-28. Настройка производительности с помощью **FrontPage** Server Extensions

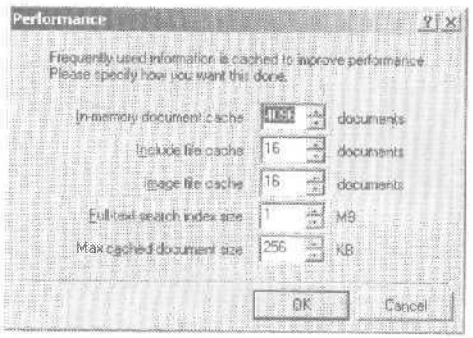


Рис. 9-29. Настройка нестандартных параметров кэширования

При изменении параметров кэширования необходимо соблюдать равновесие: если отвести под кэш слишком много памяти, ее не хватит приложениям, а если слишком мало, то приложению придется чаще обращаться к диску для чтения необходимых документов.

Изменение уровня защиты приложения

Как вы знаете из главы 2, IIS способен исполнять Web-приложения в одном процессе с 1IS, в отдельном процессе вместе с другими Web-приложениями либо изолированно, в собственном процессе. Чем выше уровень изоляции, тем надежнее работает Web-приложение, поскольку изолированные приложения более защищены от влияния ошибок и иных сбоев, возникающих в других процессах.

Однако за повышение уровня изоляции приходится расплачиваться снижением быстродействия Web-приложения, поскольку в изолированном приложении необходимо передавать вызовы процедур через границы процесса. Как и в случае кэширования, здесь не

обойтись без компромисса, так как выигрыш в надежности оборачивается проигрышем в быстродействии.

Чтобы изменить уровень зашиты Web-приложения, выполните следующие действия.

1. Щелкните правой кнопкой Web-приложение в окне IIS и выберите из контекстного меню команду Properties — IIS откроет диалоговое окно свойств Web-приложения.
2. В раскрывающемся списке Application Protection щелкните Low, Medium или High, чтобы назначить уровень зашиты приложения. Щелкните OK.

Настройка атрибутов элемента processModel

Атрибуты элемента *processModel,* определяемого в файле Machine.config на сервере, позволяют управлять определенными параметрами производительности. Они определяют, например, число запросов в очереди, частоту проверки соединения с клиентом и количество потоков, исполняемых каждым процессором. Эти атрибуты выделены **полужирным шрифтом** в следующем примере:

```
<processModel
   enable="true"
   timeout="Infinite"
   idleTimeout="Infinite"
   shutdownTimeout="0:00:05"
   requestLimit="Infinite"
   requestQueueLimit="5000"
   restartQueueLimit="10"
   memoryLimit="60"
   webGarden="false"
   cpuMask="0xffffffff"
   userName="machine"
   password="AutoGenerate"
   logLevel="Errors"
   clientConnectedCheck="0:00:05"
   comAuthenticationLevel="Connect"
   comImpersonationLevel="Impersonate2"
   responseRestartDeadlockInterval="00:09:00"
   responseDeadlockInterval="00:03:00"
   maxWorkerThreads="25"
   maxIoThreads="25"
/>
```

В таблице 9-3 описаны атрибуты элемента *processModel,* связанные с регулировкой производительности.

Таблица 9-3. Атрибуты, управляющие производительностью

Имя	Назначение
requestQueueLimit	Определяет, сколько запросов можно поставить в очередь, прежде чем ASENET станет возвращать отклик с кодом состояния 503 (Server too busy)
clientConnectedCheck	Задает интервал (чч:мм:сс) между проверками соединения с клиентом
maxWorkerThreads	Определяет максимальное число потоков на процессор
maxIOThreads	Задает максимальное число потоков ввода-вывода на процессор

Как правило, чем меньше значения этих параметров, тем быстрее сервер обрабатывает запросы, но при этом он способен обслужить меньшее число клиентов. И наоборот, чем больше эти параметры, тем больше пользователей может обслуживать сервер, но их запросы будут обрабатываться медленнее.

Изменение атрибутов элемента sessionState

Запись сведений о состоянии для каждого сеанса занимает время, поэтому если Web-приложение не использует переменные состояния *Session,* следует отключить их.

Чтобы отключить переменные состояния *Session,* откройте файл приложения Web.config и установите атрибут *mode* элемента *sessionState* в *«Off»,* как показано ниже:

```
<sessionState
        mode="Off"
        stateConnectionString="tcpip=127.0.0.1:42424"
        sqlConnectionString="data source=127.0.0.1;user id=sa;password="
        cookieless="false"
        timeout="20"
/>
```

Если вы отключили переменные состояния *Session,* то не сможете использовать их ни в одном из модулей кода или классов приложения.

Советы по оптимизации

Как правило, *настройкой* (tuning) называют регулирование развернутого приложения без изменения его кода. В отличие от настройки, под *оптимизацией* (optimization) обычно понимают изменение кода таким образом, чтобы он быстрее исполнялся и потреблял меньше ресурсов. Проще говоря, примерами оптимизации могут быть все рациональные приемы программирования, усвоенные при изучении материала этой книги. Самые важные из них кратко описаны далее:

- отключайте **отладочные функции в рабочих приложениях.** Код, скомпилированный как окончательная (Release) версия, работает быстрее кода, скомпилированного как отладочная (Debug) версия;
- **старайтесь минимизировать число обменов данными между клиентом и** сервером. Для обработки серверных событий, генерированных страницей, ASP.NET должна вернуть форму на сервер. Старайтесь конструировать Web-формы так, чтобы пользователь получал возможность отослать данные только после заполнения формы. Для этого перед отправкой данных на сервер следует проверить (при помощи верификаторов), заполнена ли форма;
- **отключайте переменные** состояния Session, **если они не нужны.** В некоторых случаях стоит заменить переменные состояния механизмами, использующими хранение данных на клиенте, такими, как файлы cookie;
- **отключайте состояние отображения для серверных элементов управления, не требующих сохранения значений** между **отображениями;**
- **применяйте хранимые** процедуры **при работе с базами данных.** Хранимые процедуры работают быстрее, чем обычные запросы;
- заменяйте **наборы данных объектами SqIDataReader,** если **можно извлечь данные путем однонаправленного чтения.** Код. использующий объект *SqIDataReader,* работает быстрее и использует меньше памяти, чем код, который создает набор данных.

Эти советы — не догма, руководствуйтесь здравым смыслом, применяя их. Если вам нужно сохранять сеансовые данные для клиента, без колебаний применяйте переменные состояния *Session* и не «изобретайте велосипед», пытаясь написать собственный код для решения этой задачи.

Занятие 4. Развертывание приложения на нескольких серверах

Для Web-приложений, обслуживающих множество клиентов или предоставляющих большие объемы данных, необходима возможность наращивания производительности по мере роста потребностей пользователей, то есть *масштабируемость* (scalability). Web-приложения ASP.NET являются масштабируемыми, поскольку они способны работать в нескольких процессах, исполняемых несколькими процессорами и/или серверами.

Web-приложение, работающее на одном многопроцессорном сервере, называется в документации по ASP.NET *Web-садом* (Webgarden), а исполняемое на нескольких серверах — *Web-фермой* (Web farm). В этом занятии вы научитесь применять серверы с подобной конфигурацией в собственных Web-приложениях.

Изучив материал этого занятия, вы сможете:

- ✓ конфигурировать Web-приложение для исполнения на нескольких или всех процессорах многопроцессорного сервера;
- ✓ исполнять Web-приложение на нескольких серверах;
- ✓ предоставлять нескольким экземплярам Web-приложения доступ к переменным *Session* при помощи сервера состояния;
- ✓ применять базы данных SQL для совместного использования переменных состояния *Session*.

Продолжительность занятия — около 10 минут.

Масштабирование путем наращивания числа процессоров

При использовании многопроцессорного сервера можно задать процессоры, на которых будет исполняться ASP.NET. Это делается посредством атрибута *webGarden* элемента *processModel*, определяемого в файле Machine.config сервера (см. строки, выделенные в примере ниже **полужирным шрифтом):**

```
<processModel
   enable="true"
   timeout="Infinite"
   idleTimeout="Infinite"
   shutdownTimeout="0:00:05"
   requestLimit="Infinite"
   requestQueueLimit="5000"
   restartQueueLimit="10"
   memoryLimit="60"
   webGarden="true"
   cpuMask="0xffffffff"
   userName="machine"
   password="AutoGenerate"
   logLevel="Errors"
   clientConnectedCheck="0:00:05"
   comAuthenticationLevel="Connect"
   comImpersonationLevel="Impersonate"
   responseRestartDeadlockInterval="00:09:00"
```

```
responseDeadlockInterval="00:03:00"
maxWorkerThreads="25"
maxIoThreads="25"
```

В таблице 9-4 описаны атрибуты элемента *processModel*, управляющие Web-садами.

Таблица 9-4. Атрибуты, управляющие Web-садами

Имя	Назначение
webGarden	Для исполнения Web-приложения ASP.NET на нескольких процессорах многопроцессорного сервера этот атрибут должен быть установлен в *true*
cpuMask	Задает процессоры, исполняющие Web-приложения ASP.NET. Значение *0xffffffff* означает, что для исполнения приложения используются все процессоры

Атрибут *cpuMask* представляет собой битовую маску, каждый разряд которой разрешает (если он равен I) или запрещает (если он равен 0) исполнение ASP.NET на определенном процессоре сервера. Например, на четырехпроцессорном сервере двоичное значение 1101 разрешает ASP.NET использовать процессоры 0, 2 и 3, освобождая процессор 1 для других задач. Двоичное значение 1101 соответствует шестнадцатеричному 0x0000000d.

Исполнение приложений ASP.NET на нескольких процессорах требует особого подхода к использованию переменных состояния *Application* и *Session*. Подробно об этом рассказано далее в разделе «Предоставление доступа к сведениям о состоянии»,

Масштабирование путем наращивания числа серверов

Чтобы использовать несколько серверов для обработки запросов одного HTTP-адреса, необходимо установить в сети средства балансировки нагрузки, которые бывают как аппаратными, так и программными.

В Windows 2000 Advanced Server и Windows 2000 Datacenter имеется компонент Network Load Balancing (NLB), распределяющий запросы между серверами. Дополнительные сведения об установке и применении этого инструмента см. в разделе справочной системы Visual Studio .NET, озаглавленном «Network Load Balancing Provider».

Если сеть поддерживает балансировку нагрузки, можно установить Web-приложение на нескольких серверах, при этом каждый запрос автоматически направляется наименее загруженному серверу.

При исполнении приложений ASP.NET на нескольких серверах необходим особый подход к использованию переменных состояния *Application* и *Session*. Подробнее об этом — в следующем разделе.

Предоставление доступа к сведениям о состоянии

При использовании как Web-садов, так и Web-ферм запросы клиентов направляются для обработки наименее загруженному процессу ASP.NET, Это означает, что на протяжении сеанса клиента могут обслуживать разные процессоры или серверы. При этом следует учитывать некоторые моменты, касающиеся переменных состояния *Application* and *Session:*

- переменные состояния приложения уникальны для каждого экземпляраWeb-приложения. Клиенты могут обмениваться данными через переменные состояния *Application* приложений, исполняемых на Web-садах или Web-фермах:
- сеансовые переменные состояния по умолчанию хранятся внутри процесса. Чтобы сделать переменные состояния *Session* доступными всем сеансам, обслуживаемым Web-садом или Web-фермой, необходимо определить механизм доступа к этим переменным состояния.

Более подробно эти проблемы рассматриваются в следующих разделах.

Совместное использование сведений о состоянии на уровне приложения

Чтобы обмениваться данными между сеансами, обслуживаемыми Web-садом или Web-фермой, следует использовать для сохранения и извлечения данных ресурс, доступный всем процессам, например XML-файл, базу данных или любой другой ресурс, поддерживающий стандартные методы доступа к файлам или базам данных.

Совместное использование сведений о состоянии на уровне сеанса

ASP.NET поддерживает два метода для **обмена** данными **переменных состояния *Session* в Web-садах и Web-фермах:**

- **сервер состояния (задается сетевым адресом)** — метод, **простой в реализации и не требующий установки SQL Server;**
- **база данных SQL (задается SQL-соединением)** — обеспечивает максимальную производительность при сохранении и извлечении сведений о состоянии.

Чтобы обмениваться данными переменных состояния ***Session*** **при помощи сервера состояния, выполните** следующие **действия.**

1. **Установите значения для атрибутов** *sessionState* **и** *stateConnectionString* **элемента** *mode,* **определяемого в файле** Web.config **приложения. Ниже показан пример настройки параметров,** определяющих **использование** сервера **состояния с IP-адресом** 192.168.1.102 **через** порт 42:

```
<sessionState
   mode="StateServer"
   stateConnectionString="tcpip=192.168.1.102:42"
   sqlConnectionString="data source=192.168.1.102;user id=sa;password="
   cookieless="false"
   timeout="20"
/>
```

2. **Запустите на сервере состояния** утилиту aspnet_state.exe. При установке Visual **Studio** .NET **Professional** Edition **или** Visual **Studio** .NET Enterprise Architect **Edition эта** утилита записывается в **каталог** \WINDOWS\Microsoft.NET\Framework*номер_версии*

Чтобы обмениваться данными переменных состояния *Session* **через базу** данных, **выполните** *следующие* **действия.**

1. Установите **значения для атрибутов** *sessionState* **и** *sqlConnectionString* элемента ***mode,* определяемого в файле** Web.config приложения. **Например,** следующие **параметры определяют использование сервера SQL с IP-адресом** 192.168.1.102:

```
<sessionState
   mode="SQLServer"
   stateConnectionString="tcpip=192.168.1.102:42"
   sqlConnectionString="data source=192.168.1.102;user id=sa;password="
   cookieless="false"
   timeout="20"
/>
```

2. **Запустите** утилиту InstallSqlState.sql **на сервере, через который процессы** будут **обмениваться данными переменных состояния** *Session.* **При установке Visual Studio** .NET **Professional Edition,** Visual **Studio** .NET Enterprise **Developer Edition или** Visual **Studio** .NET **Enterprise Architect Edition эта** утилита **записывается в каталог** \WINDOWS\Microsoft.NET\Framework*номер_версии*.

Резюме

- При компоновке приложения для развертывания используйте параметр Release и установите атрибут *debug* из файла Web.config в *false*.
- Файлы AssemblyInfo.vb или AssemblyInfo.cs идентифицируют сборку приложения и предоставляют сведения о его версии.
- Для развертывания Web-приложения загрузите его на сервер провайдера при помощи соответствующих инструментов Visual Studio .NET либо просто скопируйте на сервер сборку и файлы содержимого приложения.
- Назначьте пользовательской учетной записи ASPNET соответствующие привилегии, чтобы разрешить Web-приложению записывать файлы и обращаться к другим ресурсам сервера, доступ к которым ограничен.
- Для наблюдения за работой Web-приложения после его развертывания применяйте оснастки MMC Event Viewer (Просмотр событий) и Performance Logs and Alerts (Журналы и оповещения безопасности).
- Web-приложения восстанавливают, заменяя их сборки и файлы содержимого на сервере.
- Для самовосстановления Web-приложений при возникновении утечек памяти и других сбоев используется рециркуляция процессов ASP.NET.
- Для повышения производительности Web-приложений в FrontPage Server Extensions применяется кэширование часто запрашиваемых страниц.
- Атрибуты элемента *processModel* в файле Machine.config сервера позволяют создать Web-сад.
- Для создания Web-ферм необходимы компоненты балансировки нагрузки, распределяющие запросы между серверами.
- Совместное использование переменных состояния *Session* в Web-садах и Web-фермах определяется значением элемента *sessionState* в файле Web.config приложения.

Практикум 9. Компоновка и развертывание Web-приложения

В этом практикуме вы подготовите Web-приложение и выполните его развертывание на сервере. Далее вы займетесь мониторингом и настройкой приложения. Для упражнений этого практикума вам понадобится готовое приложение. Это может быть одно из приложений, созданных в практикумах предыдущих глав, или ваше собственное приложение.

Поскольку методики, предлагаемые в этом практикуме, применимы к любым типам Web-приложений, интересно выбрать приложение, которое вы планируете развертывать в Интернете для других пользователей или для себя.

Продолжительность практикума — около 30 минут.

Упражнение 1. Подготовка приложения



Сейчас вы скомпонуете окончательную версию Web-приложения, пользуясь материалами занятия 1 и следующими указаниями.

1. Введите идентификационные данные приложения и определите номер его версии в файле AssemblyInfo.vb или AssemblyInfo.cs.
2. Откройте файл Web.config приложения и установите атрибут *debug* элемента *compilation* в *false*.
3. Установите параметр компоновки приложения в Release.
4. Выполните компоновку приложения.

Упражнение 2. Развертывание приложения



Способ развертывания приложения зависит от типа доступного сервера. Если у вас есть учетная запись службы хостинга (см. главу 1), воспользуйтесь инструментами Visual Studio .NET для загрузки приложения на сервер провайдера.

Для развертывания приложения на собственном сервере воспользуйтесь материалами занятия 2 и следуйте указаниям, приведенным ниже,

1. С помошью IIS создайте виртуальный каталог для размещения приложения на сервере.
2. Скопируйте Web-приложение в созданный виртуальный каталог.
3. Проверьте параметры безопасности сервера; убедитесь, что разрешений пользовательской учетной записи ASPNET достаточно для выполнения необходимых действий.
4. Если требуется, предоставьте этой учетной записи доступ к ресурсам, используемым вашим приложением.
5. Попробуйте запустить приложение на сервере, чтобы удостовериться, что текущие параметры безопасности не мешают его работе.

Упражнение 3. Отображение сведений о производительности приложения на графике



Сейчас вы при помощи оснастки MMC Performance and Alerts зарегистрируете в журнале число запросов, обработанных ASP.NET, и отобразите собранные сведения на графике вместе с активностью процессора сервера. Примерный вид графика показан на рис. 9-30.

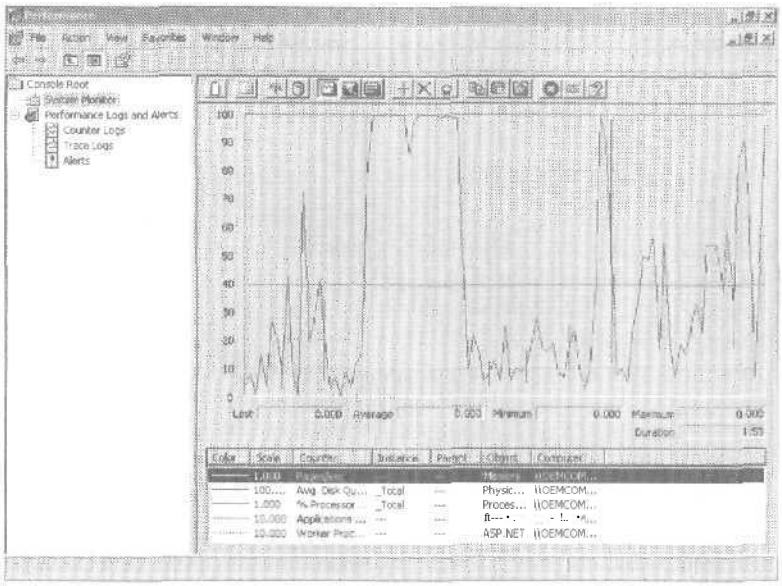


Рис. **9-30. График параметров производительности Web-приложения**

Используя знания, полученные на занятии 3, выполните следующие действия.

1. Вызовите оснастку MMC Performance and Alerts и подключитесь к серверу, на котором работает Web-приложение, развернутое в предыдущем упражнении. Если приложение развернуто на сервере службы Web-хостинга, подключиться к нему не удастся. В этом случае запустите Web-приложение локально и подключитесь к своему локальному компьютеру.
2. Создайте новый журнал счетчика и назовите его «Exercise 3».
3. Добавьте в журнал счетчики, перечисленные в таблице.

Объект	Счетчик
Processor	% Processor time
ASP.NET applications	Anonymous requests
ASP.NET	Applications running
ASP.NET	Worker processes running

Чтобы собрать сведения о сервере, начните регистрацию данных в журнале и остановите ее через несколько минут.

4. Щелкните в оснастке элемент System Monitor (Системный монитор) и измените его свойства, чтобы он показывал файл журнала:
   - щелкните правой кнопкой System Monitor и выберите из контекстного меню команду Properties (Свойства);
   - перейдите на вкладку Source (Источник), выберите LogFiles (Файлы журнала) и щелкните кнопку Add (Добавить), чтобы добавить файлы журналов для отображения;
   - **перейдите** на вкладку **Data (Данные) и** щелкните **кнопку Add, чтобы добавить** счетчики ASP.NET для мониторинга;
   - на вкладке Properties установите для всех счетчиков ASP.NET интервал, равный 10;
   - **шелкните кнопку OK,** чтобы **отобразить содержимое журнала на графике.**

5. Для просмотра данных журнала в текстовом виде щелкните кнопку Report (Просмотр отчета) в окне System Monitor.
6. Поэкспериментируйте с другими счетчиками (например, со счетчиками ошибок приложений ASP.NET, попаданий кэша и т. д.).

Упражнение 4. Настройка приложения



Сейчас вы установите FrontPage Server Extensions и настроите параметры кэширования для Web-приложения. По умолчанию Visual Studio .NET устанавливает FrontPage Server Extensions одновременно с .NET-компонентами. Если в упражнении 2 для развертывания приложения использовался сервер службы Web-хостинга, эти действия выполнить не удастся из-за отсутствия доступа к серверу провайдера. В этом случае, как и в упражнении 3, используйте Web-приложение, запущенное локально.

Используя знания, полученные на занятии 3, выполните следующие действия.

1. Запустите IIS и установите FrontPage Server Extensions в виртуальный каталог сервера, в котором находится Web-приложение.
2. В IIS вызовите окно свойств виртуального каталога и перейдите на вкладку Server Extensions.
3. Выберите в списке Performance элемент Tune For <100 Pages и щелкните OK.
4. Чтобы зарегистрировать изменения на сервере, запустите журнал счетчика, созданного в упражнении 3.

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. Какие разрешения назначаются Web-приложениям по умолчанию?

2. Почему файл Machine.config так важен для развертывания Web-приложений?

3. Как задать параметры кэширования для Web-приложения?

4. Опишите особенности применения переменных состояния *Session* в Web-приложениях, работающих на Web-фермах и Web-садах.

ГЛАВА 10

Тестирование Web-приложений

Занятие 1. Подготовка тестирования 424
Занятие 2. Проведение тестирования 436
Занятие 3. Отладка приложения 450

В этой главе

Цель тестирования программного обеспечения — убедиться в его соответствии предъявляемым требованиям. Из этого простого положения ясно, что нельзя начинать тестирование без:

- полных и четко сформулированных требований к программному продукту;
- грамотно определенных задач тестирования, в противном случае неясно, когда считать тестирование законченным.

Обычно требования к продукту и задачи тестирования формулируются независимо от разработчиков: это делают, как правило, заказчики или администрация компании. Б этой главе рассказано, как планировать тестирование на основе требований к продукту, подготавливать и проводить тесты, а также устранять неполадки, выявленные тестированием.

Прежде всего

Для занятий по материалам этой главы вам потребуются:

- доступ к Web-серверу (через локальную сеть или провайдера услуг хостинга) для тестирования и отладки Web-приложений;
- хотя бы одно готовое приложение, созданное в практикуме одной из предыдущих глав;
- знание языка сценариев Windows (VBScript или JScript);
- установленный Application Center Test (ACT) из Visual Studio .NET Enterprise Architect Edition — он понадобится только в разделе, посвященном нагрузочному тестированию Web-приложений.

Занятие 1. Подготовка тестирования

Любой программный продукт, предназначенный для широкого применения, должен пройти тестирование. Чем сложнее продукт и чем шире круг пользователей, тем более необходимо тестирование для его коммерческого успеха. Без тестирования нельзя быть уверенным, что программа будет работать как надо и не придется краснеть перед пользователями за качество программного продукта.

В отношении программного обеспечения термин *тестирование* почти всегда означает *автоматизированное тестирование* (automated testing). Оно подразумевает применение программ, воспроизводящих записанные действия пользователя или имитирующих работу компонента с системой. Автоматизированные тесты *воспроизводимы,* то есть их можно проводить многократно, а их результаты *измеримы* (позволяют судить, выполнен тест или нет). Эти ключевые характеристики тестирования гарантируют, что прошедший тестирование программный продукт соответствует предъявляемым к нему требованиям.

В этой главе описано, как составить план тестирования, использовать различные типы тестов для всесторонней проверки программного продукта и создавать различные испытательные задания при помощи Visual Studio .NET.

Изучив материал этого занятия, вы сможете:

- ✓ составлять план тестирования программного обеспечения;
- ✓ описывать различные типы тестирования программного обеспечения;
- ✓ писать задания для блочного тестирования компонентов программ;
- ✓ писать задания для комплексного тестирования готовых компонентов программы;
- ✓ объяснять, как обнаружить регресс приложения с помощью библиотеки тестов;
- ✓ создавать задания для нагрузочного тестирования Web-приложений с помощью **ACT;**
- ✓ тестировать Web-приложения на клиентах с различной конфигурацией.

Продолжительность занятия — около 30 минут.

Составление плана тестирования

Первый шаг тестирования — разработка его плана на основе требований к программному продукту. Обычно план тестирования считается официальным документом, основное назначение которого — гарантировать, что продукт:

- **прошел всестороннее тестирование.** Код, не прошедший тестирование, вносит элемент неопределенности в работу приложения и повышает риск сбоев;
- **соответствует предъявляемым требованиям.** Чтобы программный продукт был признан соответствующим требованиям заказчика, он должен работать, как описано в его спецификации, и поддерживать все перечисленные в ней функции. В силу этих обстоятельств спецификация программного продукта должна быть написана четко и ясно;
- **не имеет дефектов.** Качество реализации функций программы должно быть не ниже заданного стандартом, который также необходимо четко описать в плане тестирования.

Наличие плана тестирования поможет избежать ручного тестирования, когда множество разработчиков и тестировщиков независимо друг от друга проверяют работоспособность отдельных частей кода. Результаты такого тестирования обычно трудно воспроизводимы и всегда непредсказуемы. Хороший план тестирования содержит ответы на следующие вопросы:

- **как написаны тестовые задания** — в плане следует описать языки и инструменты, используемые для тестирования;

- **кто отвечает за тестирование** — необходимо указать группы или отдельных специалистов, которые занимаются подготовкой и проведением тестирования;
- когда **проводится тестирование** — график тестирования не должен отставать от графика разработки;
- **где проводится тестирование и как публикуются его результаты** — следует так организовать тесты, чтобы их можно было регулярно повторять;
- **что подвергается тестированию** — выбор измеримых критериев и постановка конкретных задач позволят не пропустить момент, когда тестирование можно считать успешно завершенным,

В зависимости от типа тестов на некоторые из этих вопросов можно отвечать по-разному. Например, за первоначальное тестирование кода обычно отвечают написавшие его разработчики, а за проверку работоспособности кода приложения в целом несет ответственность специальная группа тестировщиков. В следующих разделах описаны типы и методы тестирования, применяемые в Visual Studio .NET.

Типы тестирования

В плане тестирования предусмотрено проведение различных тестов, позволяющих убедиться в соответствии программного продукта требованиям заказчика и отсутствии в нем дефектов. Наиболее популярные типы тестирования описаны в таблице 10-1.

Таблица 10-1. Типы тестирования

Тип	Назначение
Блочное	Независимая проверка отдельных компонентов кода
Комплексное	Проверка совместной работы всех компонентов
Регрессивное	Выявление сбоев в работе уже реализованных функций, вызванных добавлением новых функций
Нагрузочное (стрессовое)	Проверка работы продукта при экстремальных нагрузках
Тестирование на платформе	Проверка работы программного продукта на всех целевых аппаратных и программных платформах

Обычно эти тесты зависят друг от друга и проводятся в порядке, указанном в таблице 10-1. Порядок тестирования отражен на диаграмме, показанной на рис. 10-1.

Схема, показанная на рис. 10-1, оптимальна для тестирования программных продуктов с модульной конструкцией, разрабатываемых в виде набора компонентов, которые пишутся, тестируются, монтируются и сдаются по отдельности. Таков современный подход к разработке и тестированию программного обеспечения. Проектов, использующих монолитную конструкцию, когда интеграция и тестирование программного продукта выполняются только на заключительной стадии цикла разработки, лучше избегать.

Блочное тестирование

Блок — это наименьший фрагмент кода, который можно подвергнуть тестированию. С точки зрения объектно-ориентированного программирования, блоками являются классы, свойства, методы и события. Все блоки должны пройти тестирование, прежде чем их интегрируют в проект.

Обычно задания для блочного тестирования некоторого компонента пишут его авторы, используя тот же язык программирования, на котором написан код, или родственный язык сценариев (например, если код написан на Visual Basic .NET, сценарий для тестиро-



Рис. 10-1. Цикл тестирования программного продукта

вания можно написать на VBScript). Процедура самотестирования блока бывает такой простой, как получение или установка значения некоторого свойства, или довольно сложной, как, например, расчет результата на основе примерных данных с последующим сравнением полученного результата с известным эталоном.

Тестирование компонентов организуется в соответствии со структурой программы, то есть отдельные свойства и методы класса проверяются во время тестирования вызывающей их процедуры, которую в свою очередь вызывает специальное тестирующее приложение. Вот пример тестирующей процедуры, которая создает экземпляр класса *FlashCardClass* (вы написали его при выполнении практикума в одной из предыдущих глав) и тестирует все его свойства и методы:

Visual Basic .NET

```
Function TestFlashCard() As Boolean
    ' Создать объект класса
    Dim FlashCard As New vbFlashCards.FlashCardClass()
    ' Если обьект не существует...
    If IsNothing(FlashCard) Then
        Console.WriteLine("FlashCardClass failed")
        ' ...вернуть False и закончить на этом.
        Return False
    End If
    ' Проверить метод shuffle с помощью максимального и минимального значений.
    FlashCard.Shuffle(Integer.MinValue, Integer.MaxValue)
    ' Протестировать свойства класса.
    Dim bResult As Boolean - True
    Dim intFirst As Integer = FlashCard.FirstNumber
    Dim intSecond As Integer = FlashCard.SecondNumber
    Dim dblAnswer As Double = intFirst + intSecond
```

```
    If dblAnswer o FlashCard.Answer Then
        Console.WriteLine("Error: Numbers don't add up!")
        bResult = False
    End If
    ' Перемешать карточки.
    FlashCard.Shuffle(Integer.MinValue, Integer.MaxValue)
    ' Убедиться, что значения не повторяются.
    If intFirst = FlashCard.FirstNumber Then
        Console.WriteLine("Warning: FirstNumber not unique after shuffle")
    End If
    If intSecond = FlashCard.SecondNumber Then
        Console.WriteLine("Warning: SecondNumber not unique after shuffle")
    End I*
    ' Проверить арифметические действия,
    FlashCard.Operation = "-"
    If FlashCard.FirstNumber - FlashCard.SecondNumber <> FlashCard.Answer Then
        Console.WriteLine("Error: - operator doesn't substract.")
        bResult = False
    End If
    FlashCard.Operation = "x"
   If FlashCard.FirstNumber * FlashCard.SecondNumber <> FlashCard.Answer Then
        Console.WriteLine("Error: x operator doesn't multiply. ")
        bResult - False
    End If
    FlashCard.Operation = "+"
   If FlashCard.FirstNumber + FlashCard.SecondNumber o FlashCard.Answer Then
        Console.WriteLine("Error: + operator doesn't add. ")
        oResult - False
    End If
    ' Вернуть сообщение об успехе или провале теста.
    Return bResult
Ena Function
```

Visual C#

```
static bool TestFlashCardClass()
{
   // Создать объект класса.
   csFlashCards.FlashCardClass FlashCard = new csFlashCards.FlashCardClass();
   // Если объект не существует...
   if (FlashCard -- null)
   {
         Console.WriteLine("FlashCardClass failed");
         // ...вернуть false и закончить на этом,
         return false;
   }
   /I Проверить метод shuffle с помощью максимального и минимального значений.
   FlashCard.Shuffle(int.MinValue, int.MaxValue);
```

```
    // Протестировать свойства класса,
    bool bResult  = true;
    int intFirst - FlashCard.FirstNumber;
    int intSecond = FlashCard.SecondNumber;
    double dblAnswer - intFirst + intSecond;
    if (dblAnswer != (double)FlashCard.Answer())
    {
          Console.WriteLine("Error: Numbers don't add up!");
          bResult = false;
    }
    // Перемешать карточки.
    FlashCard.Shuffle(int.MinValue, int.MaxValue);
    // Убедиться, что значения не повторяются.
    if (intFirst == FlashCard.FirstNumber)
          Console.WriteLine("Warning: FirstNumber not unique after shuffle");
    if (intSecond == FlashCard.SecondNumber)
          Console.WriteLine("Warning: SecondNumber not unique after shuffle");
    // Проверить арифметические действия.
    FlashCard.Operation - "-";
    if (FlashCard. FirstNumber - FlashCard. SecondNumber != FlashCard. Answer())
    {
          Console.WriteLine("Error: - operator doesn't substract.");
          bResult - false;
    }
    FlashCard.Operation = "x";
    if (FlashCard. FirstNumber * FlashCard.SecondNumber != FlashCard.Answer())
    {
          Console.WriteLine("Error: x operator doesn't multiply.");
          bResult = false;
    }
    FlashCard.Operation - "+";
    if (FlashCard. FirstNumber + FlashCard.SecondNumber != FlashCard.Answer())
    {
          Console.WriteLine("Error: + operator doesn't add.");
          bResult = false;
    }
     // Вернуть сообщение об успехе или провале теста.
    return bResult;
}
```

Важнейшей характеристикой блочного тестирования считается его *полнота*. Блочное тестирование должно охватить все фрагменты кода приложения и весь диапазон возможных значений аргументов. **В** предыдущем **примере это требование выполнялось путем** передачи **методу *Shuffle* крайних значений аргументов — *MinValue* и** *MaxValue*.

Тестирующие **приложения должны выводить сообщения об ошибках, генерированные тестируемым элементом, чтобы не** усложнять локализацию **и исправление ошибок. Предыдущий код отображает сообщения в окне Output при помощи метода *Console.*** *WriteLine*. **Сообщения об ошибках также можно записывать в файл журнала тестирования средствами метода *Debug.*** *WriteLin***или генерировать уведомления вызовом** *Debug.Assert*.

Комплексное тестирование

Первый комплексный тест должен дать ответ на вопрос: компилируется ли приложение? На этом этапе ошибка компиляции любого из компонентов проекта задерживает комплексное тестирование. В некоторых группах разработчиков компоновку проекта принято выполнять еженощно. Это дает уверенность, что программный продукт будет компилироваться; если же при компиляции возникает ошибка, наутро группа оперативно устраняет ее.

Основное затруднение при компоновке незаконченного проекта состоит в том, что некоторые компоненты пытаются использовать другие компоненты, которые пока еще не написаны. Эта проблема возникает из-за применения при разработке модульного подхода, когда последовательность создания компонентов изменяется. Чтобы избежать этого, применяют *заглушки* (stubs) — компоненты, не выполняющие никаких функций, за исключением предоставления определения классов, свойств и методов другим компонентам. Заглушка относится к компонентам, необходимым для тестирования кода, их создание будет описано немного позже.

После разрешения всех проблем, возникающих при компоновке программы, комплексное тестирование превращается в подобие блочного тестирования — основное внимание концентрируется на проверке совместной работы отдельных компонентов приложения. При этом можно тестировать компоненты, для взаимодействия которых требуются другие компоненты, пока отсутствующие. Для тестирования подобных компонентов служат *драйверы*. В данном контексте это компоненты, необходимые для тестирования совместной работы двух и более компонентов. На более поздних этапах разработки драйверы заменяют настоящими компонентами, которые будут готовы к этому времени.

Помимо заглушек и драйверов для тестирования приложений используются тестовые интерфейсы, обеспечивающие автоматизацию комплексного тестирования. *Тестовый интерфейс* (testing interface) — это набор открытых свойств и методов, с помощью которых внешняя тестирующая программа управляет тестируемыми компонентами. Вот пример тестового интерфейса, автоматизирующего тестирование Web-формы приложения FlashCard:

Visual Basic .NET

```
#If DEBUG Then
    ' Открытый интерфейс для комплексного тестирования.
    Public FirstNumber As String
    Public SecondNumber As String

    Public Function TestUI(ByVal Answer As String) As Boolean
        FirstNumber = lblFirst.Text
        SecondNumber = lblSecond.Text
  ' Записать Answer в поле txtAnswer.
        txtAnswer.Text = Answer
        ' Исполнить обработчик события Changed.
        txtAnswer_TextChanged(Me, EventArgs.Empty)
        ' Проверить результат.
        If lblFeedback.Text = "Correct!" Then
            Return True
        Else
            Return False
        End If
    End Function
#End If
```

Visual C#

```
#if DEBUG
// Открытый интерфейс для комплексного тестирования.
public string FirstNumber;
public string SecondNumber;

public bool TestUI(string Answer)
{
   FirstNumber - lblFirst.Text;
   SecondNumber = lblSecond.Text;
   txtAnswer.Text = Answer;
   // Исполнить обработчик события Changed.
   txtAnswer_TextChanged(this, EventArgs.Empty);
   // Проверить результат.
   if (lblFeedback.Text == "Correct!")
         return true;
   else
         return false;
}
tfendif
```

Здесь используются директивы препроцессора *#If...#EndIf* (*#if...#endif* в случае C#), чтобы скомпилированная версия тестового интерфейса не оказалась в конечной версии программного продукта, Это важно, поскольку тестовый интерфейс открывает доступ к внутренним механизмам приложения, которые в конечной версии следует скрыть.

Регрессивное тестирование

Блочное и комплексное тестирование составляют основу регрессивного тестирования. Все тестовые задания, созданные разработчиками или тестировщиками, после проведения тестирования передаются в библиотеку тестов, которую используют для регулярного проведения планового тестирования. Если после внесения нового или модификации имеющегося компонента приложение перестало проходить один из блочных или комплексных тестов, ранее успешно выполнявшихся, констатируют *регресс* приложения.

Если имеющиеся тестовые задания написаны корректно, то сообщения об ошибках, генерируемые при автономном и комплексном тестировании, помогут легко выявить причину регресса, в противном случае потребуется усовершенствовать имеющиеся тесты или написать новые.

Ключ к успеху комплексного и регрессивного тестирования — частое проведение полного набора тестов. По возможности это нужно делать каждую ночь после компоновки проекта. Своевременное обнаружение и исправление ошибок поможет избежать ситуаций, когда одни неполадки маскируют другие.

Нагрузочное тестирование

Возможно, после развертывания Web-приложения с ним будут работать не больше двух-трех клиентов одновременно, а может статься, что запросов окажется тысячи. Для проверки работы Web-приложения в условиях резкого колебания нагрузки служит нагрузочное тестирование. Для подготовки и проведения нагрузочных тестов Web-приложений используется Application Center Test (ACT).

Процедура применения ACT такова.

1. Создайте тестовое задание, записав действия пользователя во время сеанса Web-приложения с помощью ACT.
2. Установите свойства теста, такие, как уровень нагрузки и продолжительность тестирования.
3. Проведите тестирование.

Обычно нагрузочное тестирование выполняется в цикле разработки проекта позже и не так часто, как блочное, комплексное или регрессивное тестирование. Подробно методика подготовки и проведения нагрузочного тестирования описана в следующих разделах.

Запись задания для нагрузочного тестирования

Чтобы записать задание для нагрузочного тестирования с помощью ACT, выполните следующие действия.

1. В ACT выберите из меню Action команду New Test — ACT запустит мастер New Test Wizard.
2. Щелкните кнопку Next — мастер откроет страницу Test Source (рис, 10-2).

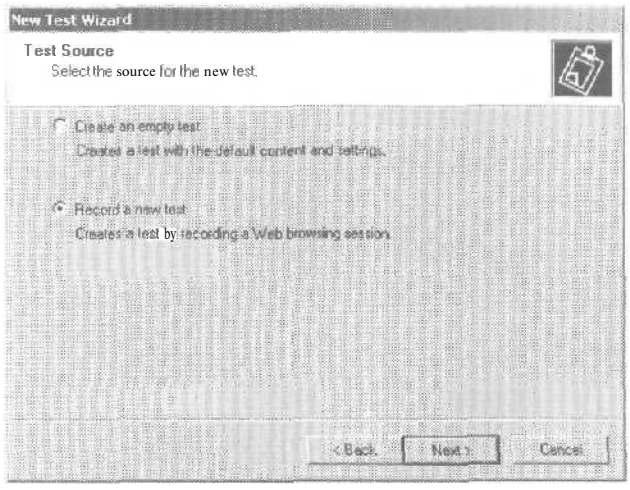


Рис. 10-2. Запись нового нагрузочного теста

3. Установите переключатель в положение Record A New Test и щелкните Next. Мастер откроет страницу Test Type (рис. 10-3).
4. Щелкните Next — мастер откроет страницу Browser Record (рис. 10-4).
5. Щелкните кнопку Start Recording — мастер откроет окно браузера, как показано на рис. 10-5.
6. Выполните действия, которые нужно записать. Можно по очереди открыть все страницы Web-приложения и выполнить на них какие-либо действия.
7. Закончив, щелкните кнопку Stop Recording на странице Browser Record, затем щелкните Next. Мастер откроет страницу Test Properties (рис. 10-6).
8. Введите имя теста и щелкните Next — мастер откроет страницу Completing The New Test. Щелкните Finish, чтобы завершить работу мастера.

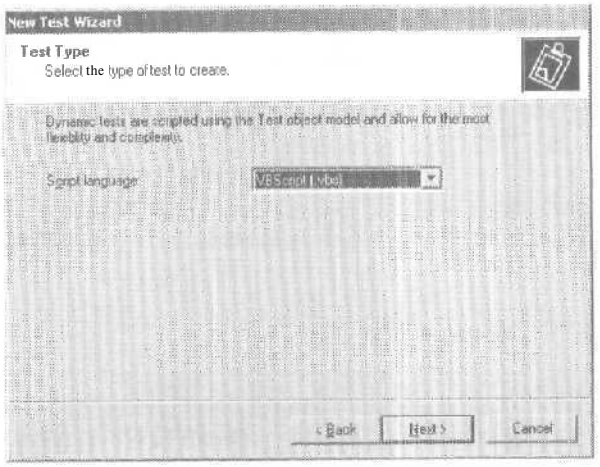


Рис. 10-3. Выбор языка сценариев

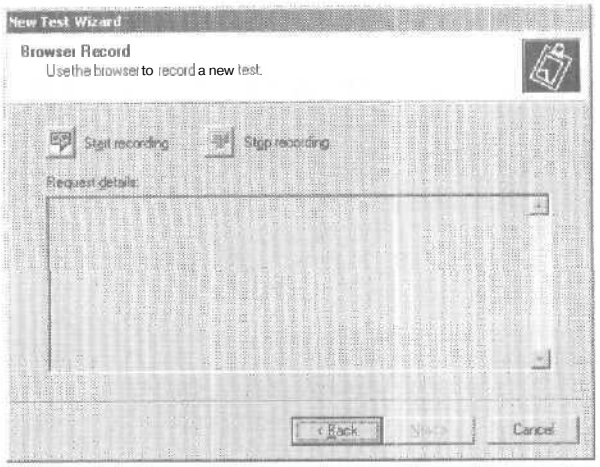


Рис. 10-4. Запуск и остановка записи

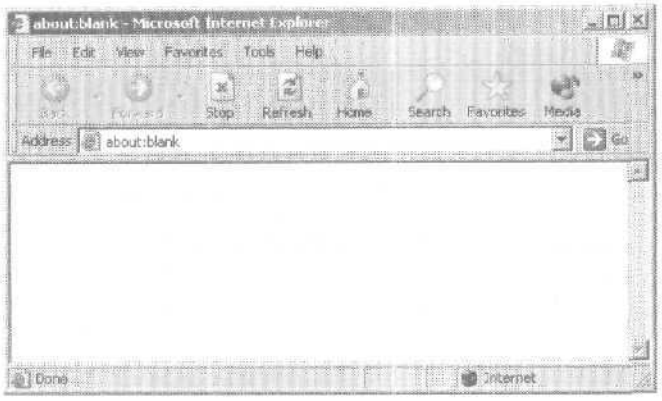


Рис. 10-5. Запись выполняемых действий

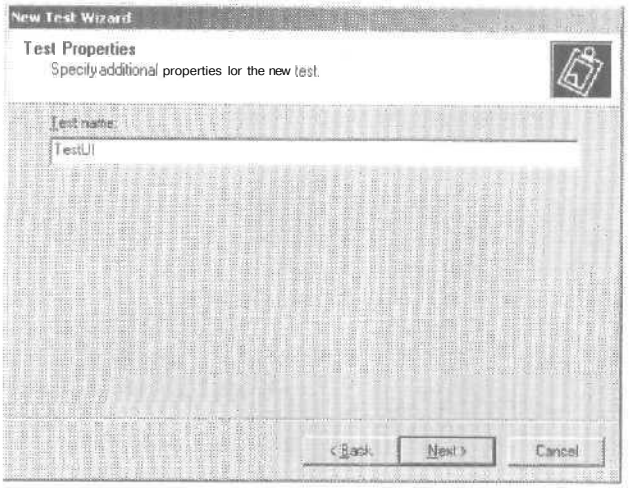


Рис. 10-6. Ввод имени теста

Установка свойств теста

Записанный тест появляется в списке Tests в окне ACT, как показано на рис. 10-7.

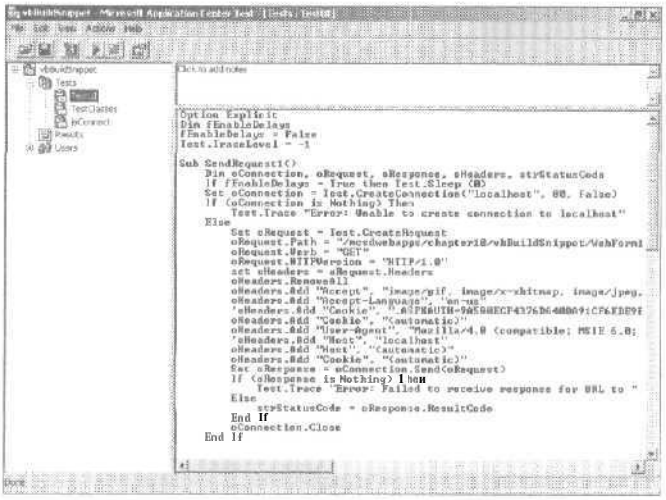


Рис. 10-7. Записанный тест

Тестовые задания ACT можно создавать на языках VBScript или JScript, но сам ACT записывает их только на VBScript. Чтобы установить уровень нагрузки и продолжительность тестирования для записанного теста, выполните следующие действия.

1. В меню Action выберите элемент Properties — ACT откроет окно свойств теста (рис. 10-8).
2. Определите уровень нагрузки, увеличивая число пользователей, одновременно работающих с Web-приложением. Затем установите продолжительность тестирования, задав время исполнения теста либо число исполняемых запросов. Закончив, щелкните OK.

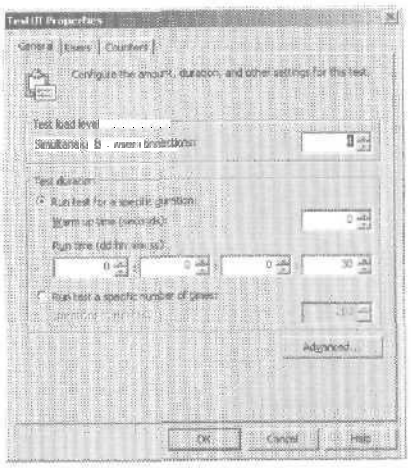


Рис. 10-8. Установка уровня нагрузки и продолжительности тестирования

Исполнение нагрузочного теста

Чтобы исполнить нагрузочный тест в ACT, выберите из меню Action пункт Start Test — ACT откроет диалоговое окно Test Status (рис. 10-9).

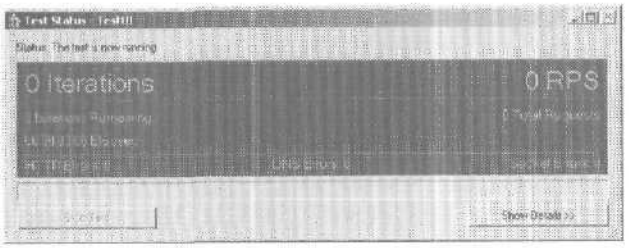


Рис. 10-9. Диалоговое окно Test Status

Исполнение тестов в ACT и анализ их результатов более подробно описаны в занятии 2.

Тестирование на платформе

В случае Web-приложений тестирование на платформе обычно сводится к проверке следующих моментов:

1. Web-формы корректно отображаются во всех поддерживаемых браузерах, включая все их версии;
2. если Web-приложение открывается в браузере, который оно не поддерживает, оно отображает соответствующее сообщение, например инструкции по загрузке нужной версии браузера;
3. при отсутствии на клиентском компьютере необходимых компонентов, например объектов ActiveX или *подключаемых модулей* (plug-in), пользователю предлагается установить их;
4. Web-приложение обладает приемлемой производительностью при применении медленных подключений, например, через модем.

Для проведения тестирования на платформе необходима лаборатория с компьютерами, аппаратное и программное обеспечение которой представляет различные платформы, поддерживаемые вашим приложением (рис. 10-10).

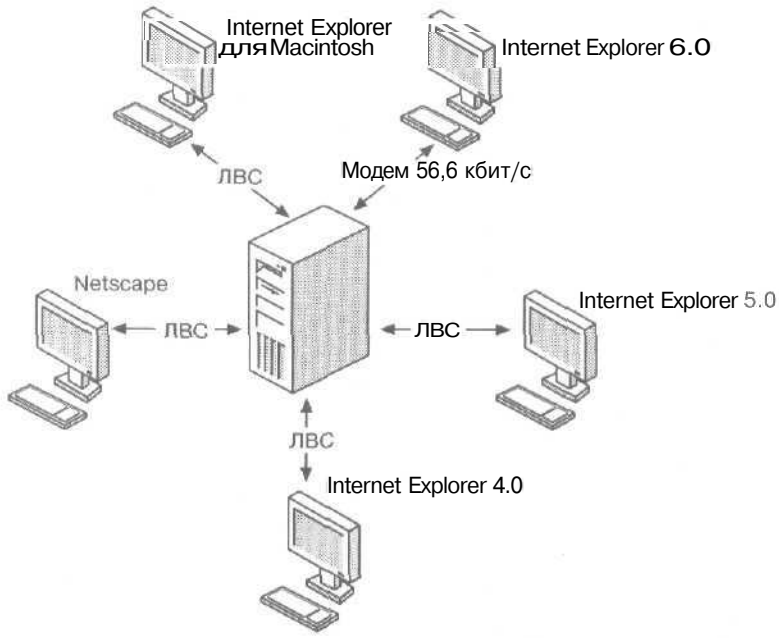


Рис. 10-10. Организация лаборатории для тестирования на платформе

Проверка корректности отображения Web-форм требует участия человека. Поскольку в различных браузерах ASP.NET по-разному прорисовывает серверные элементы управления, на заключительной стадии тестирования следует предусмотреть визуальную проверку Web-приложения.

Занятие 2. Проведение тестирования

На предыдущем занятии вы научились планировать тестирование и создавать различные типы тестовых заданий с помошью Visual Studio .NET, а сейчас узнаете, как проводить различные тесты и как получать их результаты.

Изучив материал этого занятия, вы сможете:

- ✓ проводить нагрузочное тестирование Web-приложений с помощью ACT;
- ✓ записывать результаты и сведения о производительности при исполнении нагрузочного теста;
- ✓ имитировать в ACT разные типы браузеров и проверять наличие определенных строк в откликах браузера при помощи тестовых сценариев;
- ✓ тестировать сборки .NET с помощью сервера сценариев Windows;
- ✓ создавать командные файлы для сборки и тестирования сборок .NET;
- ✓ использовать Планировщик Windows для запуска в заданное время командных файлов, выполняющих автоматическую компоновку и тестирование в ночное время.

Продолжительность занятия — около 30 мин.

Тестирование с помощью ACT

На предшествующем занятии мы кратко продемонстрировали подготовку и исполнение нагрузочных тестов в ACT. В этом разделе подробнее обсуждаются вопросы настройки и исполнения тестов ACT, в том числе:

- регистрация результатов тестирования;
- настройка тестов для одновременного использования нескольких соединений;
- исполнение тестов с различными комбинациями имен и паролей пользователей;
- добавление к тесту счетчиков производительности;
- имитация различных браузеров и типов сетевых подключений.

Регистрация результатов тестирования

По умолчанию для каждого теста ACT создает файл журнала в каталоге /Perflogs, расположенном на том же диске, где установлен ACT.

Чтобы изменить каталог, в котором ACT хранит файлы журнала, выполните следующие действия.

1. Щелкните правой кнопкой элемент в левой панели окна ACT и выберите из контекстного меню команду Properties — ACT откроет окно свойств теста (рис. 10-11).
2. Перейдите на вкладку Debugging и введите в текстовое поле Folder To Save Log In путь к каталогу для хранения файлов журнала этого теста. Закончив, щелкните OK.

В зависимости от значения свойства *TraceLevel*, заданного для теста, в журналы ACT записывается разный объем данных. Чтобы его изменить, установите свойство *TraceLevel* для объекта *Test* в сценарии теста. Например, если включить в сценарий следующие строки, ACT будет регистрировать все сообщения:

VBScript

```
Test.TraceLevel = -1
```

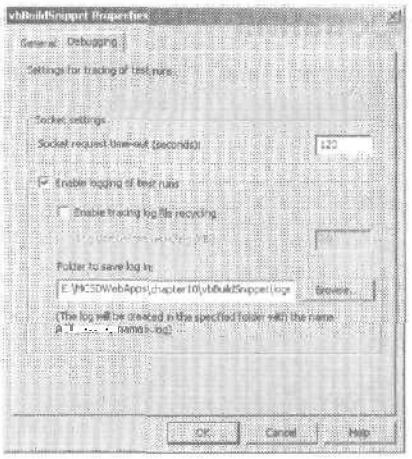


Рис. 10-11. Определение каталога для хранения файлов журнала

JScript

```
Test.TraceLevel = -1;
```

Возможные значения свойства *TraceLevel* перечислены в таблице 10-2.

Таблица 10-2. Значения свойства TraceLevel

Значение	Что регистрируется
-1	Все данные, в том числе запросы и отклики приложения, а также любые сообщения, генерируемые при вызове метода *Trace* из сценария теста
0	Это значение отключает регистрацию данных в журнале
1	Запросы и отклики приложения
2	Сообщения, генерируемые при вызове метода *Trace* из сценария теста (это значение задано по умолчанию)

Вызовом метода *Trace* объекта *Test* из тестового сценария сообщения записываются в журнал. Следующий сценарий записывает в журнал сообщение, если ACT не удается подключиться к серверу:

VBScript

```
Dim g_oConnection

ConnectToServer

Sub ConnectToServer()
Set g_oConnection = Test.CreateConnection
("http://www.contoso.com", 80, false)
    If (g_oConnection is Nothing) Then
        Test.Trace "Error: Unable to create connection to contoso"
    End If
End Sub
```

JScript

```
var g_oConnection

ConnectToServer();

function ConnectToServer()
{
    g_oConnection = Test.CreateConnection
                        ("http://www.contoso.com", 80, false);
    if (g_oConnection == null)
        Test.Trace("Error: Unable to create connection to contoso");
}
```

Использование нескольких подключений

ACT способен моделировать различные уровни загруженности сервера, увеличивая число одновременных подключений. По умолчанию для отправки запросов Web-приложению ACT применяет одно подключение. Создание дополнительных подключений эквивалентно росту числа одновременно работающих пользователей.

Чтобы увеличить количество подключений, используемых при тестировании, выполните следующие действия.

1. В правой панели окна ACT щелкните тест правой кнопкой и выберите из контекстного меню элемент Properties — ACT откроет окно свойств теста (рис. 10-12).

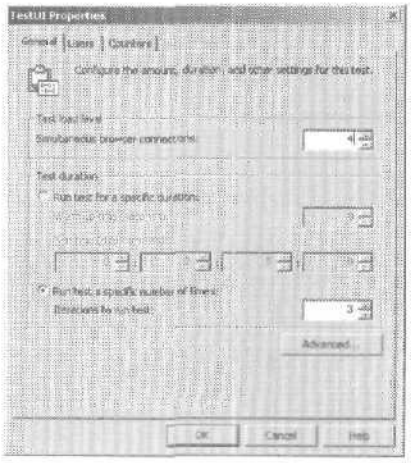


Рис. 10-12. Увеличение числа подключений

2. В секции Test Load Level измените число одновременных подключений и щелкните OK.

Добавление пользователей и паролей

ACT поддерживает назначение имен и паролей пользователей Web-приложения посредством групп, которые хранятся в проекте теста. Эта функция позволяет, кроме прочего, тестировать аутентификацию пользователей Web-приложения и работу приложения с cookies.

Чтобы добавить в группу новых пользователей, выполните следующие действия.

1. Щелкните элемент Users в левой панели окна ACT, чтобы раскрыть список групп пользователей.
2. Выберите группу пользователей и в таблице слева введите имена и пароли новых пользователей. На рис. 10-13 показано добавление пользователей в группу Default Users Group.

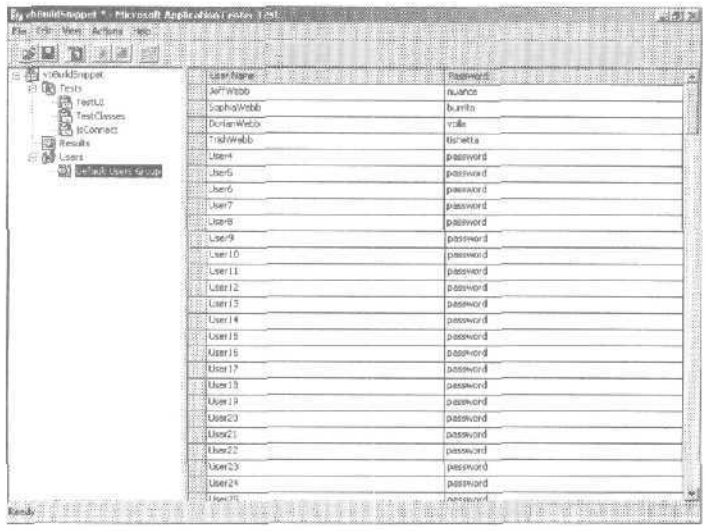


Рис. 10-13. Добавление пользователей к группе

3. В правой панели окна ACT шелкните правой кнопкой тест и выберите из контекстного меню команду Properties — ACT откроет окно свойств теста (рис. 10-14).

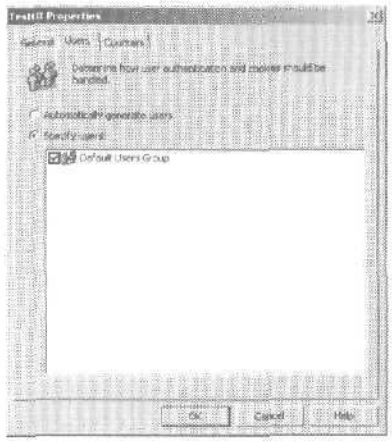


Рис. 10-14. Активация пользователей через свойства теста

4. Перейдите на вкладку Users, установите переключатель Specify Users и щелкните OK.

ACT автоматически выбирает пользователей для каждого прогона теста, а также для каждого нового подключения. Пользователей можно указать и вручную, вызывая из сценария теста метод *GetNextUser* объекта *Test* следующим образом:

VBScript

```
Sub ChangeUserO
   Test.GetNextUser
   Test.Trace("Next user name: " & Test.GetCurrentUser.Name)
End Sub
```

JScript

```
function ChangeUserO
{
   Test.GetNextUser();
   Test.Trace("Next user name: " + Test.GetCurrentUser.Name);
}
```

Добавление счетчиков производительности

Счетчики производительности, добавленные к тесту, позволяют следить за производительностью Web-приложения во время нагрузочного теста. Для этих целей используются те же счетчики, что и для наблюдения за производительностью сервера при помощи оснастки Performance Logs and Alerts консоли MMC (см. главу 9).

Чтобы добавить к нагрузочному тесту счетчики производительности, выполните следующие действия.

1. В правой панели окна ACT щелкните правой кнопкой тест и выберите из контекстного меню команду Properties — ACT откроет окно свойств теста.
2. Перейдите на вкладку Counters и щелкните Add — ACT откроет диалоговое окно Browse Performance Counters, показанное на рис. 10-15.

   Выберите интересующие вас счетчики и щелкните Add, затем Close.
3. Щелкните OK, чтобы закрыть окно свойств теста.

При исполнении теста ACT записывает в XML-файл статистические данные о производительности, которые также отображаются на панели результатов теста в окне ACT (рис 10-16).

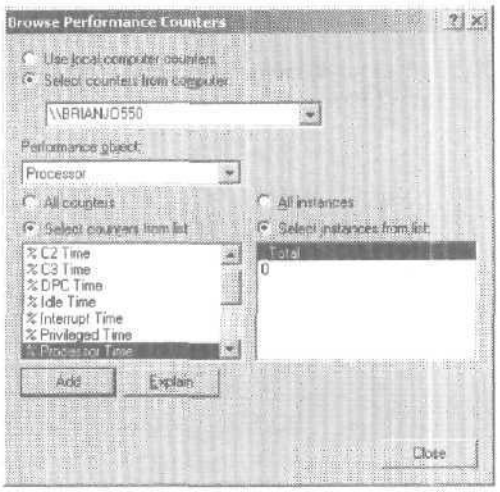


Рис. 10-15. Добавление счетчиков

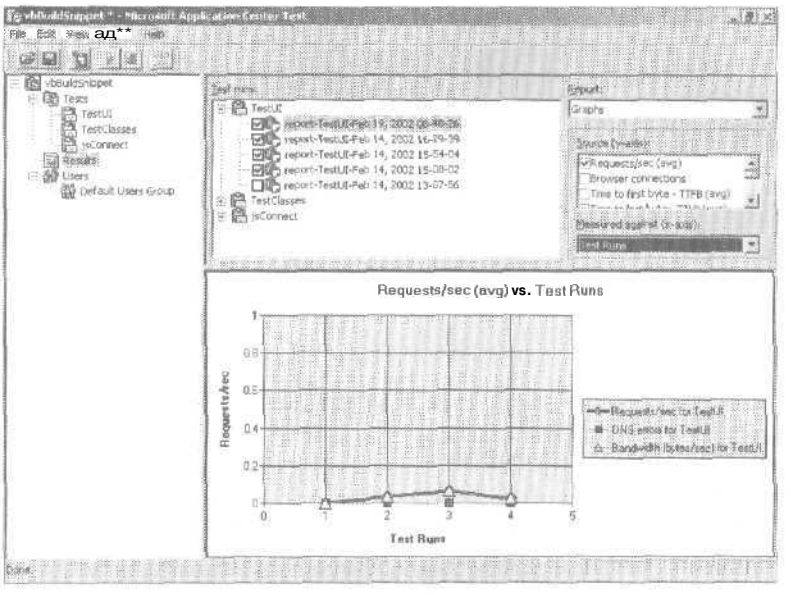


Рис. 10-16. Просмотр сведений о производительности

Имитация различных браузеров

ACT способен имитировать разные типы браузеров, изменяя сведения об агенте пользователя в заголовке посылаемых им запросов. Соответствующий пример вы найдете в тесте Browser Type из проекта ACTSamples, который устанавливается вместе с ACT.

Скопировав тест Browser Type в свой проект теста, используйте его для проверки совместимости приложения с разными типами браузеров, а также корректности его взаимодействия с браузерами, которые оно не поддерживает. Но поскольку ACT не отображает отклики, при помоши этого теста нельзя проверить, корректно ли отображается Web -приложение различными браузерами.

Получение содержимого откликов

ACT возвращает отклик сервера в свойстве *Body* объекта *Response*. Проверить наличие ожидаемого результата в отклике позволяет функция VBScript *Instr*, как показано ниже:

VBScript

```
Set oResponse = oConnection.Send(oRequest)
If Instr(oResponse.Body, "Correct!") Then
    Test.Trace("Correct response.")
Else
    Test.Trace("Incorrect response.")
End If
```

JScript

```
oResponse - oConnection.Send(oRequest);
if (oResponse.Body.search"Correct!"))
```

```
    Test.Trace("Correct response.");
else
    Test.Trace("Incorrect response.");
```

Дополнительный пример использования функции *Instr* для поиска нужных строк в отклике приводится в образце теста Body, поставляемом с ACT.

Примечание Справочная система ACT содержит полное описание объектной модели тестирования, а также концептуальные сведения о нагрузочном тестировании Web-приложений с помощью ACT.

Тестирование .NET-сборок

ACT ориентирован на взаимодействие с Web-приложениями посредством запросов и откликов. Это здорово для тестирования свойств, связанных с пользовательским интерфейсом, но менее удобно для тестирования внутренних классов приложения.

Допустим, нужно выполнить блочное тестирование класса *FlashCardClass*, как показано в занятии 1, но с использованием ACT. Как это сделать? Можно попробовать следующий способ:

VBScript

```
Test.TraceLevel = -1

Dim FlashCard
Set FlashCard = CreateObject("vbFlashCards.FlashCardClass")

If (FlashCard Is Nothing) Then
   Test.Trace("Error: FlashCard not created.")
   Test.StopTest
End If
```

JScript

```
Test.TraceLevel = -1;

var FlashCard;
FlashCard = CreateObject("csFlashCards.FlashCardClass");

if (FlashCard == null)
{
   Test.Trace("Error: FlashCard not created.");
   Test.StopTest();
}
```

Все хорошо, только вот работать это не будет! Поскольку Web-приложения применяют .NET Framework, пространства имен не годятся в качестве аргументов метода *CreateObject*, который ожидает программный идентификатор (progID) объекта COM, а не пространство имен.

Чтобы использовать сценарий для тестирования .NET-сборки, выполните следующие действия.

1. Зарегистрируйте .NET-сборку для использования с COM.
2. Убедитесь, что COM-приложения в состоянии найти .NET-сборку.
3. Создайте .NET-объект с progID, зарегистрированным для .NET-сборки.

Более подробно эти действия описаны в следующих разделах.

Регистрация .NET-сборок

Для регистрации .NET-сборки в системном реестре служит утилита RegAsm.exe, поставляемая с .NET Framework.

Процедура использования утилиты RegAsm.exe такова.

1. Вызовите окно командной строки Visual Studio .NET. Для этого выберите в меню Start элемент All Programs\MicrosoftVisual Studio .NET\Visual Studio .NET Tools\Visual Studio .NET Command Prompt.
2. В командной строке перейдите в каталог, содержащий регистрируемую сборку, и исполните следующую команду:

```
RegAsm.exe assembly.dll
```

где *assembly.dll* — это имя регистрируемой сборки.

RegAsm.exe регистрирует открытые классы из .NET-сборки, назначая им progID, созданные из их имени и пространства имен. Например, после регистрации сборки FlashCards, созданной в главе 2, в реестре появится запись, показанная на рис. 10-17.

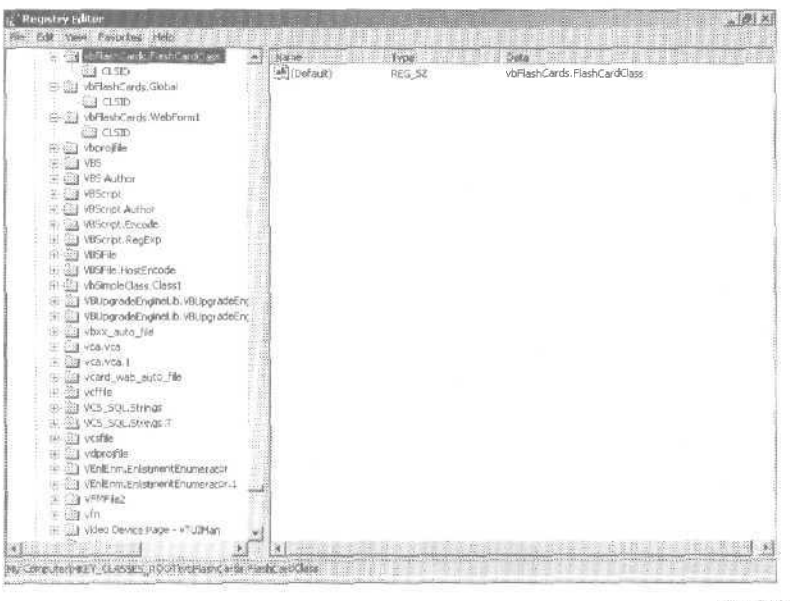


Рис. 10-17. Запись системного реестра со сведениями о сборке FlashCards

Размещение .NET-сборок

Помимо регистрации .NET-сборки компонентам COM необходимо указать путь к каталогу, в котором следует искать сборку. Это можно сделать по-разному, вот три самых простых способа:

1. поместить сборку в GAC (Global Assembly Cache);
2. скопировать сборку в каталог использующего ее компонента COM;
3. скопировать компонент COM в каталог сборки.

Сборки, помещенные в GAC, доступны всем приложениям, установленным на данном компьютере. Перед размещением в GAC сборке необходимо назначить строгое имя. Исчерпывающие инструкции по созданию строгих имен и размещению сборок в GAC вы найдете в справочной системе Visual Studio .NET.

Сборки, используемые только одним приложением, проще всего размещать в одном каталоге с компонентом COM. Этот способ особенно хорош для тестирования, когда сборка определенно не должна быть доступной другим приложениям.

Создание .NET-объектов

Зарегистрировав .NET-сборку и указав компонентам COM путь к ней, можно создать экземпляр .NET-объекта, вызвав один из методов для создания объектов COM, например функцию VBScript *CreateObject*.

Чтобы попробовать применить вышеописанную методику с инструментами для тестирования, такими, как сервер сценариев Windows (Windows Scripting Host), выполните следующие действия.

1. Зарегистрируйте сборку Web-приложения, как показано в предыдущем разделе.
2. Скопируйте файл CScript.exe из каталога \Windows\system32 или \WINNT\system32 в каталог \bin Web-приложения. Эта копия CScript.exe понадобится для исполнения сценариев тестов; она должна быть в одном каталоге с .NET-сборкой, чтобы COM- и .NET-объекты смогли взаимодействовать.
3. Создайте файл сценария VBScript (или JScript) и исполните его при помощи копии CScript.exe из каталога \bin.

Следующий код демонстрирует тестирование класса *FlashCardClass* с помощью сервера сценариев Windows:

VBScript

```
' TestFlash.vbs
' Сценарий для блочного тестирования класса FlashCardClass
Option Explicit

Dim FlashClass

Set FlashClass = CreateObject("vbFlashCards.FlashCardClass")
' Проверить, создан ли объект.
If (FlashClass is Nothing) Then
    WScript.Echo "Error: FlashCardClass not created."
    WScript.Quit
End If
' Проверить метод Shuffle.
FlashClass.Shuffle 1, 1000
Dim X, Y
x = FlashClass.FirstNumber
y = FlashClass.SecondNumber
' Перемешать карточки еще раз.
FlashClass.Shuffle 1, 1000
If (x = FlashClass.FirstNumber) or (y = FlashClass.SecondNumber) Then
    WScript.Echo "Warning: Number not unique."
End If
```

```
' Проверить свойство Answer.
If (FlashClass.FirstNumber + FlashClass.SecondNumber = FlashClass.Answer) Then
    WScript.Echo "Success: Answer adds up!"
Else
    WScript.Echo "Error: Answer incorrect."
End If
```

JScript

```
// TestFlash.js
// Сценарий для блочного тестирования класса FlashCardClass
var FlashClass;

FlashClass = new ActiveXObject("vbFlashCards.FlashCardClass");
// Проверить, создан ли объект.
if (FlashClass == null)
{
    WScript.Echo("Error: FlashCardClass not created.");
    WScript.Quit();
}
// Проверить метод Shuffle.
FlashClass.Shuffle(1, 1000);
var X, Y;
x = FlashClass.FirstNumber;
y = FlashClass.SecondNumber;
// Перемешать карточки еще раз,
FlashClass.Shuffle(1, 1000);
if ((x == FlashClass.FirstNumber) || (y == FlashClass.SecondNumber))
    WScript.Echo("Warning: Number not unique.");
// Проверить свойство Answer.
if (FlashClass.FirstNumber + FlashClass.SecondNumber == FlashClass.Answer())
    WScript.Echo("Success: Answer adds up!");
else
    WScript.Echo("Error: Answer incorrect.");
```

Для запуска этих сценариев **используются** следующие **команды** (исполняемые **в командной строке):**

```
@rem VBScript
bin\cscript TestFlash.vbs > logs\FlashErr.log

@rem JScript
bin\cscript TestFlash.js > logs\FlashErr.log
```

Автоматическая компоновка и тестирование

Регулярное **проведение блочного, комплексного и нагрузочного** тестирования **гарантирует неизменно высокое качество программного продукта и позволяет вовремя** заметить регресс. **Многие группы разработчиков выполняют компоновку и тестирование** каждую ночь. Проведение тестов **в ночное** время **удобно:** удается вовремя **завершать разработку**

компонентов и не нарушать рабочий процесс, освобождая компьютеры группы для других задач. Поскольку тестирование проводится поздно ночью, его обычно выполняют в автоматическом (необслуживаемом) режиме, который освобождает от необходимости запускать гесты вручную и следить за ходом тестирования.

Вот что надо сделать, чтобы запустить автоматическую компоновку и тестирование программы.

1. Создайте командный файл, устанавливающий значения переменных окружения, выполняющий компоновку и запускающий необходимые тесты.
2. При помощи Планировщика Windows (Windows Task Manager) создайте график запуска командного файла; настройте его так, чтобы он выполнялся каждую ночь;
3. Убедитесь, что компоновка и тестирование прошли успешно, изучив журналы, генерированные соответствующими командами.

Более подробно эта процедура описана в следующих разделах.

Создание командного файла

Можно создать командный файл (с расширением .bat), который будет выполнять все необходимые команды. Обычно командный файл состоит из трех секций, отвечающих за:

- установку переменных окружения, таких, как PATH и LIB, которые служат для поиска компонентов, необходимых для сборки;
- запуск среды разработки Visual Studio .NET для компоновки решения;
- запуск инструментальных программ, используемых для тестирования сборки.

Переменные окружения, требуемые Visual Studio .NET, находятся в командном файле VsVars32.bat, который устанавливается в каталог Command\Tools, расположенный в каталоге Visual Studio .NET. При помощи операций копирования-вставки удается перенести строки для установки нужных переменных окружения из VsVars32.bat в свой командный файл.

Для запуска Visual Studio .NET из командной строки используется команда DevEnv.exe с файлом решения (их расширение — .sln) в качестве параметра. Так, следующая команда выполняет компоновку приложения FlashCard в режиме отладки, записывая при этом любые сообщения об ошибках в каталог Logs:

```
DevEnv vbFlashCards.sln /build Debug > Logs\builders.log
```

Все необходимые команды можно собрать в одном файле. Ниже показан пример командного файла, который устанавливает необходимые переменные окружения, выполняет компоновку приложения и запускает сценарии для тестирования полученной сборки:

```
@rem Build.bat
@rem ==========================================================================
@rem Команды из vsvars32.bat, устанавливающие переменные окружения
@rem ==========================================================================
@SET VSINSTALLDIR=E:\Program Files\Microsoft Visual Studio .NET\Common7\IDE
@SET VCINSTALLDIR=E:\Program Files\Microsoft Visual Studio .NET
@SET FrameworkDir=C:\WINDOWS\Microsoft.NET\Framework
@SET FrameworkVersion=v1.0.3705
@SET FrameworkSDKDir=C:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK
@rem Ссылка на каталог с общими файлами Visual Studio.

@rem
@rem Ссылка на каталог с файлами среды разработки Visual Studio.
@rem
```

```
@set DevEnvDir=%VSINSTALLDIR%

@rem
@rem Ссылка на каталог с файлами Visual C++.
@rem
@set MSVCDir=%VCINSTALLDIR%\VC7

@set PATH=%DevEnvDir%;%MSVCDir%\BIN;%VCINSTALLDIR%\Common7\Tools;
%VCINSTALLDIR%\Common7\Tools\bin\prerelease;%VCINSTALLDIR%\Common7\
Tools\bin;%FrameworkS DKDir%\bin;%FrameworkDir%\%FrameworkVersion%;%PATH%;
@set INCLUDE=%MSVCDir%\ATLMFC\INCLUDE;%MSVCDir%\INCLUDE;
%MSVCDir%\PlatformSDK\include\prerelease;%MSVCDir%\PlatformSDK
\include;%FrameworkSDKDir%\include;%INCLUDE%
@set LIB=%MSVCDir%\ATLMFC\LIB;%MSVCDir%\LIB;
%MSVCDir%\PlatformSDK\lib\prerelease;
%MSVCDir%\PlatformSDK\lib;%FrameworkSDKDir%\lib;%LIB%

@rem ========================================================================
@rem Конец секции переменных окружения
@rem ========================================================================

@rem ========================================================================
@rem Вызов среды разработки для компоновки приложения
@rem ========================================================================
devenv vbBuildSnippet.sln /build debug > logs\builderrs.log
@rem ========================================================================
@rem Конец секции компоновки
@rem ========================================================================

@rem ========================================================================
@rem Команды тестирования
@rem ========================================================================
bin\cscript RunTests.vbs > logs\testerrs.log
@rem ========================================================================
@rem Конец секции тестирования
@rem ========================================================================
```

Планирование заданий

Для планирования запуска командного файла в ночное время используется Планировщик Windows.

Чтобы запустить Планировщик, выполните следующие действия.

1. В меню Start выберите элемент Alt Programs\Accessories\System Tools\Scheduled Tasks. (Пуск\Программы\Стандартные\Служебные\Назначенные задания) — Windows откроет папку Scheduled Tasks (Назначенные задания).
2. Дважды щелкните значок Add Scheduled Tasks (Добавить задание), чтобы запустить мастер Scheduled Task Wizard (Мастер планирования заданий). Щелкните Next (Далее), затем Browse (Обзор) и выберите командный файл для запуска. Мастер откроет страницу для выбора графика запуска (рис. 10-18).

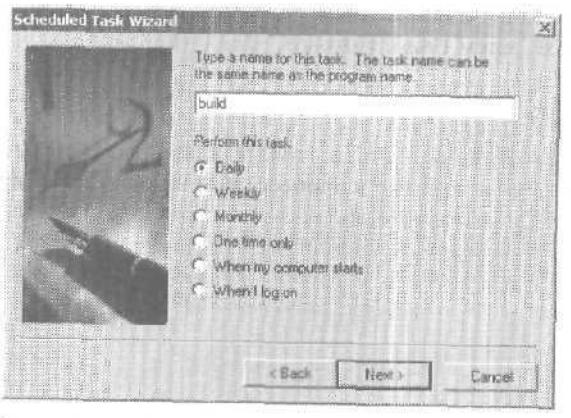


Рис. 10-18. Выбор графика запуска задания

3. Установите переключатель в положение, соответствующее нужной периодичности запуска, и щелкните Next - мастер откроет страницу для выбора времени запуска (рис 10-19)
4. Введите время запуска задания и щелкните Next. Далее откроется страница для ввода пароля (рис. 10-20).

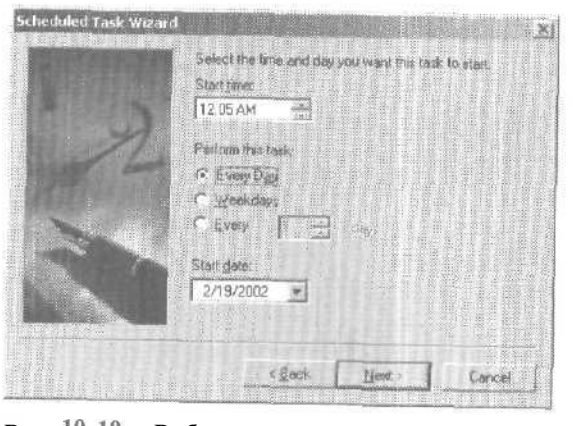


Рис. 10-19. Выбор времени запуска задания

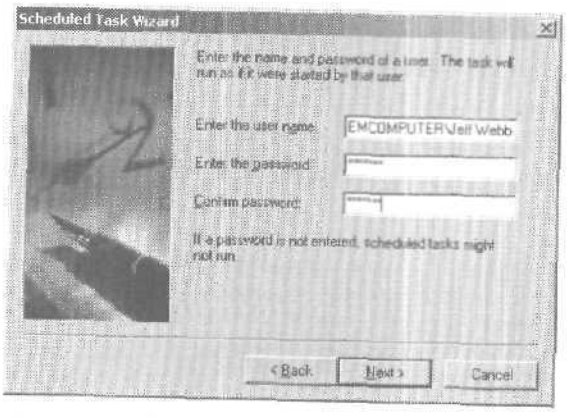


Рис. 10-20. Ввод имени пользователя и пароля

5. Введите имя и пароль пользовательской учетной записи, под которой будет запускаться задание, и щелкните кнопку Next, затем Finish (Готово).

Просмотр файлов журнала

При использовании Планировщика для компоновки и тестирования в необслуживаемом режиме важно изучать файлы журнала, генерируемые в процессе компоновки и тестирования. Показанные ранее команды для компоновки и тестирования используют оператор перенаправления вывода (>) для записи выходной информации в файлы журнала, расположенные в каталоге Log.

Файлы журнала легко просматривать в Блокноте (рис. 10-21).



Рис. 10-21. Просмотр журнала компоновки

Занятие 3. **Отладка приложения**

Тестирование — это одно из немногих дел, в которых даже неудача считается положительным результатом. Из этого занятия вы узнаете, как использовать такие результаты для устранения ошибок в коде (вам понадобятся знания, полученные при изучении отладочных средств Visual Studio .NET в 1 главе). Вы научитесь исправлять некоторые ошибки, выявленные во время тестирования, писать код для диагностики и отлаживать Web-приложения, работающие на удаленном сервере.

Изучив материал этого занятия, вы сможете:

- ✓ **применять результаты блочного** тестирования для **исправления общих** ошибок;
- ✓ **использовать в разработке диагностические** средства, **такие, как** классы **Debug** и Trace, для вывода сообщений и генерации журналов;
- ✓ конфигурировать Web-сервер так, чтобы он разрешал отладку Web-приложений с удаленных рабочих станций;
- ✓ отлаживать Web-приложения, работающие на удаленном сервере.

Продолжительность занятия — около 30 минут.

Применение результатов тестирования

Возможно, запустив сценарий блочного тестирования приложения FlashCard (см. главу 1), вы заметили, что оно не проходит некоторые тесты. Дело в том, что FlashCard — это упрощенная демонстрационная программа, не предназначенная для коммерческого применения и, следовательно, не столь надежная. Это хорошее оправдание, время от времени им можно пользоваться и на работе!

А если серьезно, результаты тестирования представляют собой не что иное, как руководство к действию. Блочное тестирование FlashCard позволяет выявить ряд ошибок. Во-первых, переполнение, возникающее из-за инициализации генератора случайных чисел целочисленными аргументами *MinValue* и *MaxValue*. Результат сложения двух чисел, близких к верхнему пределу значений типа integer, окажется больше допустимого для integer. Вот соответствующий код:

Visual Basic .NET

```
' Тестирующий код
Function TestFlashCard() As Boolean
    'Создать объект
    Dim FlashCard As New vbFlashCards.FlashCardClass()
    'Проверить, существует ли объект.
    If IsNothing(FlashCard) Then
        Console.WriteLine("FlashCardClass failed")
        ' Вернуть false и закончить на этом.
        Return False
    End If
    'Проверить метод shuffle с использованием максимального и
    'минимального значений.
    FlashCard.Shuffle(Integer.MinValue, Integer.MaxValue)
    'Проверить свойства,
```

```
    If (FlashCard.Answer o FlashCard.FirstNumber + FlashCard.SecondNumber) _
        Then
        Console.Write("Error: Answers don't match")
    End If
End Function

' Код FlashCardClass с ошибкой.
' Вычислить ответ с помощью текущего оператора,
Public Function Answer() As Integer
    Select Case mstrOp
        Case "+"
            Return mintFirstNumber + mintSecondNumber ' Переполнение!!!
        Case "x", "*"
            Return mintFirstNumber * mintSecondNumber
        Case "-"
            Return mintFirstNumber - mintSecondNumber
        Case Else
            Return 0
    End Select
End Function
```

Visual C#

```
// Тестирующий код.
static bool TestFlashCardClass()
{
   //Создать объект.
   csFlashCards.FlashCardClass FlashCard = new csFlashCards.FlashCardClass();
   //Проверить, существует ли объект.
   if (FlashCard == null)
   {
         Console.WriteLine("FlashCardClass failed");
         // Вернуть false и закончить на этом.
         return false;
   }
   //Проверить метод shuffle с использованием максимального и
   //минимального значений.
   FlashCard.Shuffle(int.MinValue, int.MaxValue);
   //Проверить свойства.
   int intFirst = FlashCard.FirstNumber;
   int intSecond = FlashCard.SecondNumber;
   double dblAnswer = intFirst + intSecond;
   if (dblAnswer != (double)FlashCard.Answer())
   {
         Console.WriteLine("Error: Numbers don't add up!");
   }
}
```

```
// Код FlashCardClass с ошибкой.
// Вычислить ответ с помощью текущего оператора.
public int Answer()
{
   switch(mstrOp)
   {
         case "+":
               return mintFirstNumber + mintSecondNumber;
//Переполнение!!!
         case "x":
               return mintFirstNumber * mintSecondNumber;
         case "*":
               return mintFirstNumber * mintSecondNumber;
         case "-":
               return mintFirstNumber - mintSecondNumber;
         default :
               return 0;
   }
}
```

Есть два пути решения этой проблемы: ограничить допустимые значения свойства *FirstNumber* и *SecondNumber* либо изменить тип значения, возвращаемого методом *Answer*. С точки зрения пользователя второе решение более понятно, поскольку тип integer часто применяют для входных значений, а выбор для этих целей другого типа с более узким диапазоном значений сложнее обосновать.

Следующий код демонстрирует один из вариантов измененного метода *Answer*, который теперь способен пройти блочное тестирование:

Visual Basic .NET

```
Public Function Answer() As Double
    Select Case mstrOp
        Case "+"
        Return CDbl(mintFirstNumber) + CDbl(mintSecondNumber)
        Case "x", "*"
        Return CDbl(mintFirstNumber) * CDbl(mintSecondNumber)
        Case "-"
        Return CDbl(mintFirstNumber) - CDbl(mintSecondNumber)
        Case "/"
        Return CDbl(mintFirstNumber) / CDbl(mintSecondNumber)
        Case Else
        Return 0
    End Select
End Function
```

Visual C#

```
public double Answer()
{
   switch(mstrOp)
```

```
    {
        case "+":
        return mintFirstNumber + mintSecondNumber;
        case "x":
        return mintFirstNumber * mintSecondNumber;
        case "*":
        return mintFirstNumber * mintSecondNumber;
        case "-":
        return mintFirstNumber - mintSecondNumber;
        default :
        return 0;
    }
}
```

Вот еще одна проблема с FlashCardClass: если присвоить свойству *Operation* значение, отличное от +, -, /, * или *x, Answer* просто вернет 0, что не очень хорошо. Следующие изменения в коде гарантируют использование только допустимых операций:

Visual Basic .NET

```
Public Property Operation() As String
    Get
        Operation = mstrOp
    End Get
    Set(ByVal Value As String)
        If InStr("+-/x*", Value) Then
            mstrOp = Value
        Else
            Throw New Exception("Unrecognized operator")
        End If
    End Set
End Property
```

Visual C#

```
public string Operation
{
    get
    {
        return mstrOp;
    }
    set
    {
        if (" +-/x* ".IndexOf(value, 1) != 0)
            mstrOp - value;
        else
            throw(new Exception("Unrecognized operator"));
    }
}
```

Эти примеры демонстрируют, как важно выполнить блочное тестирование новых классов до их включения в проект. Когда класс станет частью большого проекта, найти причину ошибки будет гораздо труднее.

Использование классов Debug и Trace

В качестве инструментов для поиска ошибок и аномального поведения приложения .NET Framework поддерживает классы *Debug* и *Trace,* определенные в пространстве имен *System.Diagnostics.* Эти классы позволяют отображать *тревожные сообщения* (alerts) или записывать некоторые сообщения в зависимости от событий, происходящих внутри приложения.

Эти классы отличаются способом их обработки в конечной сборке. При компиляции конечной версии сборки все свойства и методы *Debug* автоматически удаляются из кода, а методы и свойства *Trace* остаются. Таким образом, вы получаете два уровня диагностики: один для отладки до выпуска, а второй — для мониторинга производительности и диагностики на местах использования программного продукта.

Поскольку классы *Debug* и *Trace* входят в одно из пространств имен .NET, для их использования необходимо добавить в модули кода следующие операторы:

Visual Basic .NET

```
Imports System.Diagnostics
```

Visual C#

```
using System.Diagnostics;
```

Отображение тревожных и информационных сообщений

Классы *Debug* и *Trace* поддерживают метод *Assert*, который проверяет некоторое значение и отображает тревожное сообщение, если при проверке получилось значение *False.* Метод *Assert* останавливает приложение при получении неожиданного результата. Например, следующий код отображает тревожное сообщение, если при сложении чисел, показанных на форме FlashCards, получился неожиданный результат:

Visual Basic .NET

```
Private Sub txtAnswer_TextChanged(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles txtAnswer.TextChanged
    If txtAnswer.Text = FlashCard.Answer Then
        lblFeedback.Text = "Correct!"
        Debug.Assert(CInt(txtAnswer.Text) = CInt(lblFirst.Text) + _
        CInt(lblSecond.Text), "Form values don't add up!")
        ' Получить другой набор чисел.
        FlashCard.Shuffle()
        ' Отобразить новые числа.
        RefreshDisplay()
        ' Очистить поле, отображающее ответ.
        txtAnswer.Text = ""
    Else
        lblFeedback.Text = "Oops! Try Again."
    End If
End Sub
```

Visual C#

```
private void txtAnswer_TextChanged(object sender, System.EventArgs e)
{
   if(txtAnswer.Text == FlashCard.Answer().ToString())
   {
        lblFeedback.Text = "Correct!";
        Debug.Assert(Convert.ToInt32(txtAnswer.Text) ==
           Convert.ToInt32(lblFirst.Text) +
           Convert.ToInt32(lblSecond.Text),
           "Form values don't add up!");
        // Получить другой набор чисел.
        FlashCard.Shuffle();
        // Отобразить новые числа.
        RefreshDisplay();
        // Очистить поле, отображающее ответ.
        txtAnswer.Text = "";
   }
   else
   {
        lblFeedback.Text = "Oops! Try Again.";
   }
}
```

В случае Web-приложения тревожные сообщения отображаются на сервере. Если сервер Web-приложения работает на локальном компьютере, тревожное сообщение при исполнении этого кода появится на локальном компьютере, как и ожидалось (рис. 10-22). Если Web-приложение работает на удаленном сервере, то тревожное сообщение можно увидеть, лишь подключившись к серверному процессу приложения для удаленной отладки, но об этом чуть дальше.

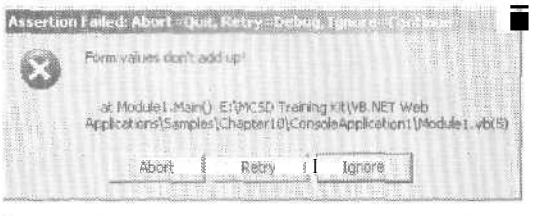


Рис. 10-22. Вид тревожного сообщения (на локальном сервере)

Методы *Write* и *WriteLine* позволяют записывать сообщения без остановки приложения. По умолчанию эти сообщения отображаются во время отладки в окне Output Visual Studio .NET.

Ведение журнала сообщений Debug и Trace

Для регистрации сообщений, генерируемых классами отладки и трассировки после развертывания приложения, создайте класс *TextWriterTracListener* и добавьте его к набору *Listeners* класса *Debug* или *Trace*. Следующий код направляет отладочные сообщения на консоль сервера:

Visual Basic .NET

```
Debug.Listeners.Add(New TextWriterTraceListener(Console.Out))
Debug.WriteLine("Starting tests.")
```

Visual C#

```
Debug.Listeners.Add(new TextWriterTraceListener(Console.Out));
Debug.WriteLine("Starting tests.");
```

Вместо этого можно записывать сообщения в файл. Для этого необходимо указать имя файла и вызвать метод *Flush*. Следующий код записывает сообщения в файл Results.log;

Visual Basic .NET

```
Debug.Listeners.Add(New TextWriterTraceListener("Results.log"))
Debug.WriteLine("Starting tests.")
Debug.Flush()
```

Visual C#

```
Debug.Listeners.Add(new TextWriterTraceListener("Results.log"));
Debug.WriteLine("Starting tests.");
Debug.Flush();
```

Удаленная отладка

Обычно Web-приложения отлаживают на том же компьютере, где велась его разработка, однако локальный компьютер не воспроизводит окружение, в котором приложению придется работать. Для устранения неполадок, возникающих во время работы приложений в реальных условиях, необходима возможность отладки Web-приложения, работающего на удаленном сервере.

Для отладки приложения на удаленном сервере выполните следующие действия.

1. Установите на сервере, обслуживающем Web-приложение, серверные компоненты Visual Studio .NET для удаленной отладки.
2. Определите на сервере права доступа, необходимые для удаленной отладки приложений.
3. Подключитесь к процессу Web-приложения с удаленного компьютера, на котором собираетесь выполнять отладку,

Эти действия более подробно описаны в следующих разделах.

Установка компонентов для удаленной отладки

Мастер установки Visual Studio .NET позволяет установить компоненты для удаленной отладки, Для этого запустите на сервере программу Visual Studio .NET Setup и выберите Full Remote Debugging (рис. 10-23).

Программа Setup установит необходимые компоненты и настроит сервер для удаленной отладки.

Определение прав доступа

Для отладки серверного процесса необходимо иметь учетную запись на этом сервере, причем она должна быть членом группы Debugger Users, которая создается программой Visual Studio .NET Setup во время установки компонентов для удаленной отладки. Следует удостовериться, что в эту группу включены все пользователи, которых следует наделить правами на отладку приложений на этом сервере.

Чтобы добавить пользователей к группе Debugger Users, воспользуйтесь MMC (рис. 10-24).

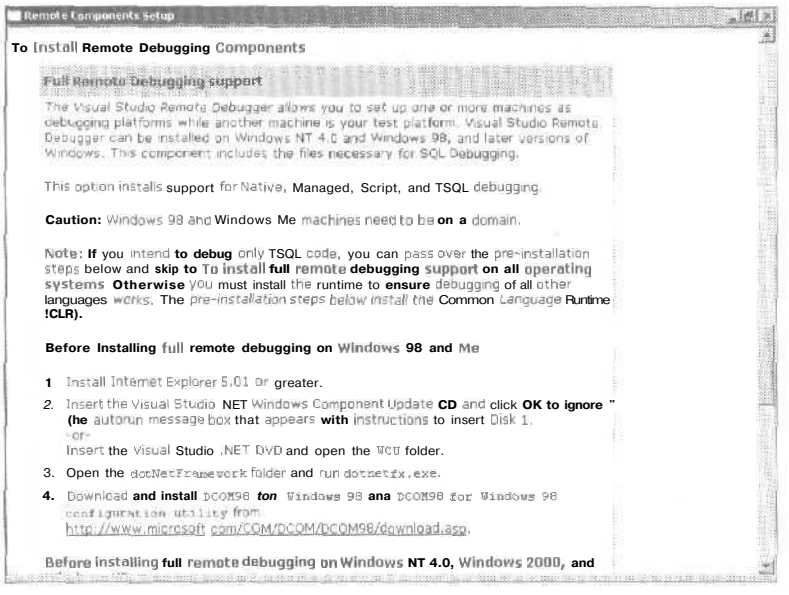


Рис. 10-23. Установка компонентов для удаленной отладки

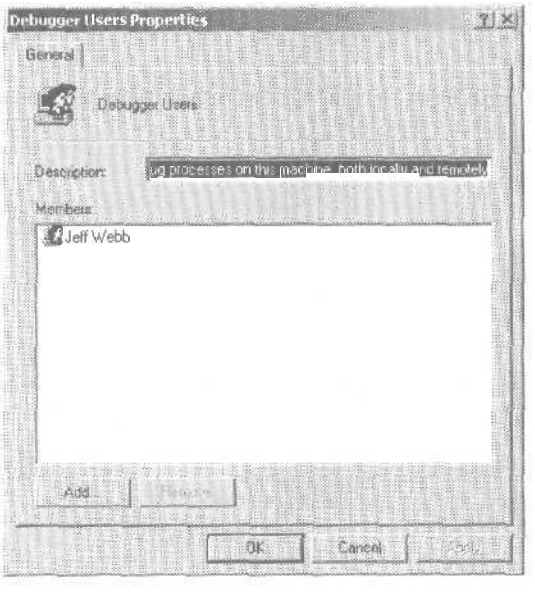


Рис. 10-24. Добавление пользователей, уполномоченных на удаленную отладку

Подключение к удаленному процессу

Подключившись к процессу, работающему на удаленном сервере, Visual Studio .NET может получать сообщения, направляемые в окно Output, отображать тревожные сообщения, генерируемые методом *Assert,* устанавливать точки прерывания и просматривать значения переменных; кроме того, в этом случае доступно пошаговое исполнение кода на удаленном сервере.

Чтобы подключиться к удаленному процессу, выполните следующие действия.

1. Выберите из меню Tools элемент Debug Process — Visual Studio .NET откроет диалоговое окно Process, показанное на рис. 10-25.

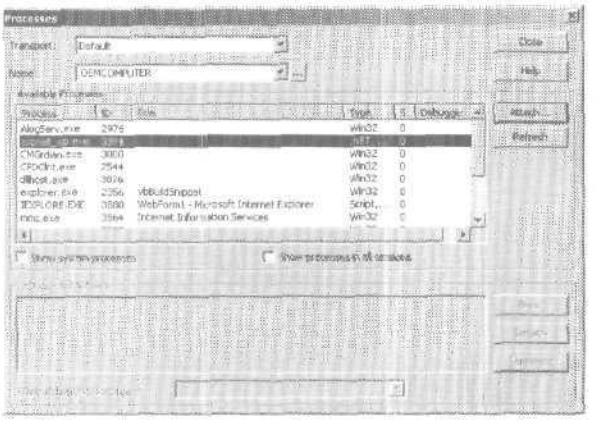


Рис. 10-25. Подключение к процессу

2. По умолчанию Web-приложения работают в процессе aspnet_wp.exe. Выберите удаленный процесс для отладки, щелкните Attach, затем Close.

Подключившись к удаленному процессу, Visual Studio .NET получает возможность вмешиваться в его исполнение посредством установки точек прерывания. Проще всего сначала установить точку прерывания на функции *Page_Load,* это позволит остановить приложение перед обработкой Web-формы, а затем расставить остальные точки прерывания, где необходимо, и понаблюдать за нужными переменными в удаленном коде.

Чтобы установить точку прерывания в удаленном процессе, выполните следующие действия.

1. В Visual Studio .NET щелкните ярлычок Breakpoints (рис. 10-26).

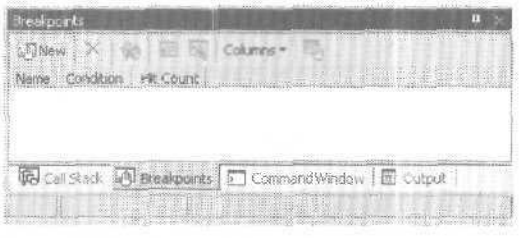


Рис. 10-26. Окно Breakpoints

2. Щелкните на панели инструментов в окне Breakpoints кнопку New — Visual Studio .NET откроет диалоговое окно New Breakpoint (рис. 10-27).
3. В текстовом поле Function введите имя процедуры и щелкните OK. В случае Web-приложения проще всего остановить исполнение перед процедурой Page_Load, а затем расставить остальные точки прерывания, где необходимо.

После останова удаленного процесса на точке прерывания вы имеете право применять любые отладочные инструменты Visual Studio .NET, доступные для отладки локальных процессов: просмотр значений переменных, пошаговое исполнение кода, а также исполнение отдельных команд.

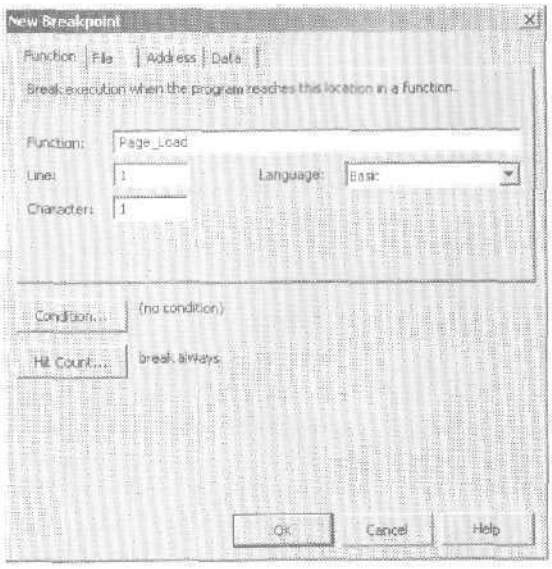


Рис. 10-27. Установка **точки прерывания**

Резюме

- Тестирование позволяет определить степень соответствия программного продукта предъявляемым к нему требованиям.
- Блочное тестирование проверяет работу отдельных фрагментов кода.
- Комплексное тестирование проверяет совместную работу всех компонентов.
- Регрессивные тесты позволяют убедиться, что добавленный новый код не нарушил работы имеющегося кода.
- Нагрузочное тестирование проверяет работу программного продукта в условиях экстремальных нагрузок.
- Тесты на платформе позволяют убедиться, что Web-приложение корректно отображается в разных браузерах и обладает достаточной производительностью при использовании различных видов сетевых подключений.
- ACT позволяет записать действия пользователя при работе с Web-приложением с помощью VBScript и воспроизвести полученный сценарий, чтобы смоделировать работу нескольких пользователей, регистрируя возникающие в приложении ошибки и параметры производительности сервера.
- Чтобы получить возможность тестирования .NET-сборок с помощью Windows Scripting Host, необходимо зарегистрировать сборку для использования с COM и скопировать сервер сценариев (CScript.exe) в каталог сборки.
- Для проведения компоновки и тестирования в необслуживаемом режиме следует создать командный файл с командами для компоновки и запуска сценариев тестов и разработать график запуска этого командного файла с помощью Планировщика.
- Диагностические классы *Debug* и *Trace* служат для отображения тревожных сообщений и регистрации ошибочных ситуаций в процессе разработки.
- По умолчанию при компиляции окончательной версии сборки код, использующий класс *Debug*, удаляется, а код, использующий класс *Trace,* остается.
- Visual Studio .NET может подключиться к процессу Web-приложения, работающего на удаленном сервере, чтобы выполнить его отладку в реальных условиях.

Практикум 10. Тестирование и отладка

Цель этого практикума — выполнить анализ реального плана тестирования, подготовить блочное тестирование приложения-примера и нагрузочное тестирование развернутого приложения. Вы также попрактикуетесь в пошаговом исполнении Web-приложения, работающего на удаленном сервере, при помощи отладочных инструментов Visual Studio .NET. Это позволит вам закрепить навыки, полученные при изучении этой главы.

Продолжительность практикума — около 30 минут.

Упражнение 1. Анализ плана тестирования



В различных компаниях применяются разные стратегии тестирования, отличается и используемая терминология. В этом упражнении вы должны найти понятия, обсуждаемые в занятии 1, в реальном плане тестирования. Для этого можно взять план тестирования компании, в которой вы работаете. Если начальство воспротивится, попробуйте найти документацию по планированию тестирования в Интернете, используя строку «software test plan» как критерий поиска.

Возьмите распечатку плана тестирования и выделите все предложения, где упоминаются следующие понятия:

- блочное тестирование;
- комплексное тестирование;
- ручное тестирование.

Попробуйте найти в плане ответы на вопросы, поставленные в занятии 1.

- Как написаны тестовые занятия?
- Кто отвечает за тестирование?
- Когда проводится тестирование?
- Где хранятся тесты и их результаты?
- Что подвергается тестированию?

Можно ли на основе собранной вами информации утверждать, что проанализированный план является четким руководством по тестированию программного продукта на соответствие предъявленным требованиям?

Упражнение 2. Блочное тестирование



Сейчас вы на VBScript или JScript напишете сценарии для блочного тестирования классов *Circle* и *Sphere,* объявленных на базе абстрактного класса *Shapes* (см. главу 3). Сценарий должен проверять:

1. можно ли создать экземпляры классов *Circle* и *Sphere;*
2. можно ли эти объекты передавать процедурам, принимающим аргументы, объявленные с типом интерфейса *IFigure*;
3. все ли свойства работают в полном диапазоне их допустимых значений;
4. все ли методы возвращают ожидаемые значения при тестировании с двумя или более наборами значений.

▶ **Как написать сценарий блочного тестирования на VBScript или JScript**

1. Зарегистрируйте тестируемую сборку для использования с COM, запустив утилиту RegAsm.exe из командной строки Visual Studio .NET.
2. Скопируйте сервер сценариев Windows (CScript.exe) в каталог тестируемой сборки.

3. Напишите тексты сценариев на VBScript или Jscript при помощи Visual Studio .NET, Блокнота или другого текстового редактора.
4. Исполните тесты в командной строке Visual Studio .NET, используя копию CScript.exe из каталога сборки.



Закончив, сравните свои результаты с образцами, которые находятся на прилагаемом компакт-диске.

Упражнение 3. Нагрузочное тестирование



Если на вашем компьютере установлена Visual Studio .NET Enterprise Architect, попробуйте применить ACT для нагрузочного тестирования Web-приложения, развернутого на тестовом сервере. Выберите любое из Web-приложений, развернутых на сервере, и выполните следующие действия.

1. Запишите по крайней мере пять разных запросов к серверу.
2. Добавьте счетчики производительности, измеряющие доли процессорного времени, занятого сервером, число страниц памяти в секунду и среднюю длину очереди чтения диска.
3. Используйте для записи результатов тестирования каталог Log, расположенный в каталоге Web-приложения.
4. Установите свойства теста так. чтобы он исполнялся в течение 30 секунд, имитируя одновременную работу 4 пользователей.
5. Исполните тест.
6. Изучите данные счетчиков производительности сервера, отображаемые на диаграмме в окне ACT.



Закончив, сравните свои результаты с примерными результатами тестирования приложения FlashCards с использованием ACT (их можно найти на прилагаемом к книге CD).

Упражнение 4. Пошаговое исполнение Web-приложения на удаленном сервере



Сейчас вы воспользуетесь Visual Studio .NET, чтобы приостановить Web-приложение, работающее на удаленном сервере, и попробуете выполнить его пошагово. Для этого необходимо быть членом группы Debugger Users на сервере, кроме того, на сервере следует установить компоненты Visual Studio .NET для удаленной отладки.

▶ **Пошаговое исполнение Web-приложения на удаленном сервере**

1. На рабочей станции откройте Web-приложение в Internet Explorer: это гарантирует, что Web-приложение будет работать на сервере в тот момент, когда вы попытаетесь подключиться к его процессу.
2. В Visual Studio .NET выберите из меню Tools элемент Debug Processes. Укажите сервер, на котором выполняется приложение, затем выберите процесс Web-приложения (его имя — aspnet_wp.exe) и щелкните Attach.
3. Установите точку прерывания на функции *Page_Load*.
4. Переключитесь в Internet Explorer и щелкните Refresh.
5. Вернитесь в Visual Studio .NET. Обратите внимание, что исполнение удаленного приложения остановилось на функции *Page_Load*.
6. Установите дополнительные точки прерывания.
7. В Visual Studio .NET щелкните Continue, чтобы продолжить исполнение Web-приложение до следующей точки прерывания.

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. Опишите связи между блочным, комплексным и регрессивным тестированием?

2. Почему нагрузочное тестирование, как правило, более важно для Web-приложения, чем для автономного Windows-приложения?

3. Чем отличаются **классы *Debug* и** *Trace*?

4. Какие два действия нужно выполнить, чтобы сделать компонент, расположенный в .NET-сборке, доступным компоненту COM?

ГЛАВА 11

Создание нестандартных Web-элементов управления

Занятие 1. Создание пользовательских элементов управления 465
Занятие 2. Создание составных элементов управления 476
Занятие 3. Создание генерируемых элементов управления 498

В этой главе

Нестандартные элементы управления расширяют инструментарий Web-разработчика. Они инкапсулируют ключевые элементы интерфейса и программной логики, благодаря чему их можно многократно использовать не только в одном приложении, но и в других проектах компании. В этой главе рассказано, как создавать и применять с Web-формами нестандартные элементы управления трех типов, поддерживаемых Microsoft Visual Studio .NET:

- **пользовательские элементы управления (user controls)** — создаются с помощью графического интерфейса дизайнера Visual Studio .NET путем комбинирования существующих серверных и HTML-элементов управления; представляют собой функциональные блоки, инкапсулирующие некоторые возможности пользовательского интерфейса. Пользовательский элемент управления хранится в файле, копию которого необходимо включать в каждый проект, использующий этот элемент управления;
- **составные элементы управления (composite controls)** — создаются из существующих серверных и HTML-элементов управления. Похожи на пользовательские, но в отличие от них создаются программно, поэтому могут быть скомпилированы в сборки (dll-файлы). Последние допустимо применять одновременно в нескольких приложениях и добавлять на панель Toolbox в Visual Studio .NET;
- **генерируемые элементы управления (rendered controls)** — генерируются кодом в виде HTML-тэгов «с нуля», а не компонуются из существующих элементов управления. Подобно составным элементам управления, генерируемые элементы управления компилируются и их можно добавлять на панель Toolbox. Однако, создавая генерируемые элементы управления, приходится самостоятельно писать код для решения задач, которые в составных элементах управления автоматизированы.

Прежде всего

Для изучения материалов этой главы необходимо:

- знать жизненный цикл Web-приложений;
- владеть приемами объектно-ориентированного программирования (ООП), в том числе методами работы с пространствами имен и сведениями о состоянии, **обсуждаемыми в** главе 3;
- иметь опыт разработки Web-приложений с применением стандартных серверных и HTML-элементов управления.

Занятие 1. Создание пользовательских элементов управления

Пользовательские элементы управления состоят из серверных или HTML-элементов управления, смонтированных на специальной странице, которую разрешено помещать на Web-формы как отдельный элемент управления. Таким образом, пользовательский элемент управления позволяет создать на основе нескольких элементов управления единый графический компонент, решающий определенную задачу.

Однажды созданный пользовательский элемент управления можно применять на всех Web-формах проекта. Но, поскольку такие элементы управления не компилируются в сборки, им присущ ряд ограничений, не характерных для других типов элементов управления:

- в каждом проекте Web-приложения, где применяется некоторый пользовательский элемент управления, должна быть отдельная копия этого элемента;
- пользовательские элементы управления нельзя добавлять на панель Toolbox в Visual Studio .NET, а только перетаскивать на поверхность Web-форму из окна Solution Explorer;
- код пользовательского элемента управления инициализируется только после загрузки Web-формы, следовательно, значения его свойств не обновляются до срабатывания страничного события *Page_Load*.

На этом занятии вы научитесь создавать пользовательские элементы управления и применять их с Web-формами.

Изучив материал этого занятия, вы сможете:

- ✓ применять пользовательские элементы управления при создании Web-приложений;
- ✓ добавлять свойства, методы и события к пользовательскому элементу управления;
- ✓ интегрировать пользовательские элементы управления в Web-форму путем модификации ее HTML-кода;
- ✓ активировать параметры времени разработки (например, использование сеточной разметки) для пользовательского элемента управления.

Продолжительность занятия — около 30 минут.

Создание и применение пользовательских элементов управления

Чтобы создать пользовательский элемент управления и задействовать его в Web-приложении, следует выполнить пять действий.

1. Добавьте к проекту страницу пользовательского элемента управления (см. ниже).
2. Определите внешний вид пользовательского элемента управления при помощи дизайнера.
3. Напишите код свойств, методов и событий элемента управления.
4. Перетащите пользовательский элемент управления из окна Solution Explorer на поверхность Web-формы,
5. Задействуйте пользовательский элемент управления в коде Web-формы. Для этого объявите его на уровне модуля, а затем вызывайте методы, свойства и события элемента управления, где это необходимо.

Более подробно эти действия описаны в следующих разделах. Кроме того, в последнем разделе этого занятия приводится ряд соображений, которые необходимо учесть при работе с пользовательскими элементами управления.

Создание интерфейса и кода пользовательского элемента управления

Пользовательские элементы управления компонуют из ютовых серверных и HTML-элементов управления, перетаскивая их с панели Toolbox на *страницу пользовательского элемента управления* (Web User Control page). Пользовательские элементы управления хранятся в файлах с расширением .ascx, которые представляют собой файлы проекта Web-приложения, подобно Web-формам или HTML-страницам.

Чтобы создать пользовательский элемент управления и добавить его к Web-приложению, выполните следующие действия.

1. В меню Project выберите команду Add Web User Control — Visual Studio .NET откроет диалоговое окно Add New Item.
2. Введите в текстовом поле Name имя пользовательского элемента управления и шелкните OK. Visual Studio .NET создаст пустую страницу пользовательского элемента управления и добавит ее к проекту, как показано на рис. 11-1.

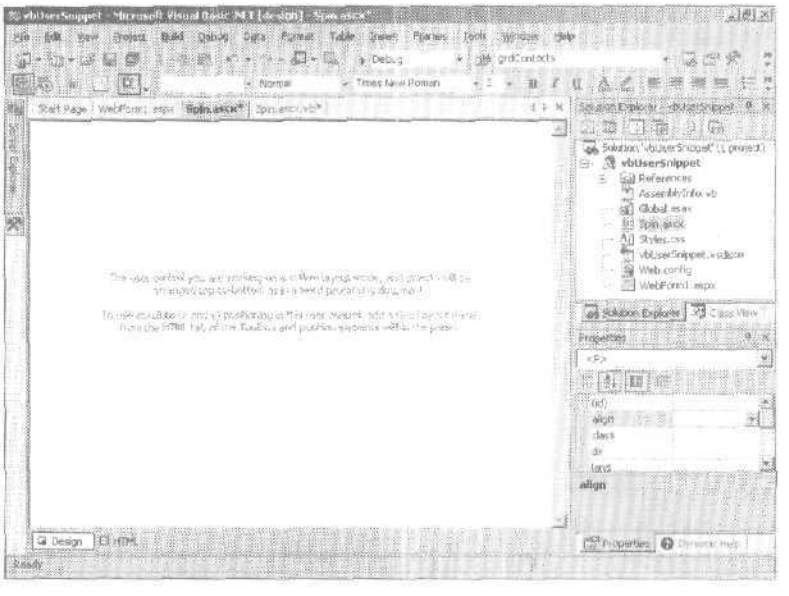


Рис. 11-1. Новая страница пользовательского элемента управления

Добавив к проекту страницу пользовательского элемента управления, определите его вид, поместив на страницу нужные серверные и HTML-элементы управления. Пользовательские элементы управления поддерживают только *потоковую разметку* (flow layout). Чтобы применить *сеточную разметку* (grid layout), следует добавить HTML-элемент управления Grid Layout Panel — он выполняет роль контейнера элементов управления, для которых требуется фиксированное положение на Web-форме. На рис. 11-2 показан простой пользовательский элемент управления, созданный из двух серверных элементов управления Button, размещенных на поверхности Grid Layout Panel.

Элемент управления, показанный на рис. 11-2, — Web-эквивалент элемента управления Spin: щелкая его стрелки, можно увеличивать или уменьшать некоторое значение. Ниже показан HTML-код этого примера, он пригодится нам в следующих разделах:

```
<DIV id="pnlGrid" runat="server" style="WIDTH: 20px; POSITION: relative;
HEIGHT: 48px" ms_positioning="GridLayout">
   <asp:Button id="butUp" Text="^" runat="server" />
   <asp:Button id="butDown " Text="v" runat="server" />
</DIV>
```

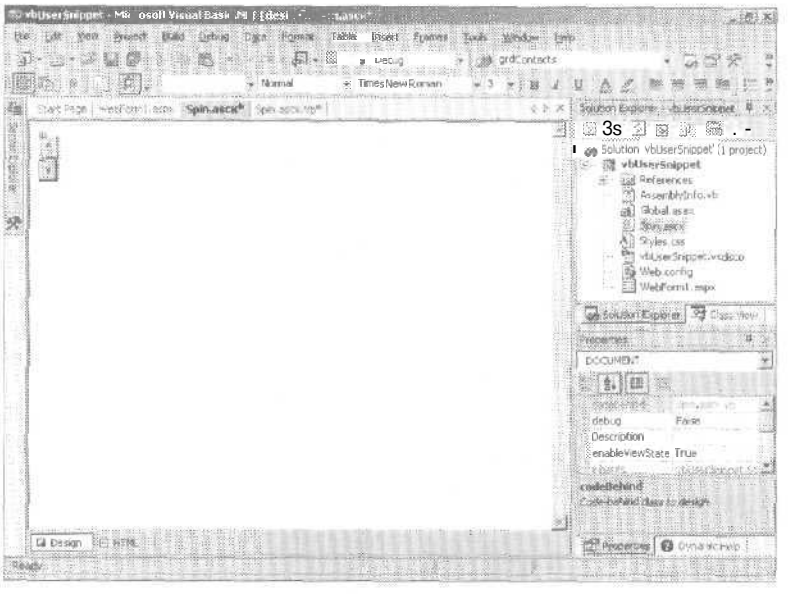


Рис. 11-2. Пример простого пользовательского элемента управления

Свойства, методы и события пользовательского элемента управления

Пользовательским элементом управления управляют из его модуля кода. Это позволяет скрыть внутреннее устройство пользовательского элемента управления, оставив доступными лишь свойства, методы и события, необходимые для выполнения этим элементом управления своих функций.

Дважды щелкните пользовательский элемент управления, чтобы отредактировать его модуль кода. Во время создания нового пользовательского элемента управления Visual Studio .NET автоматически генерирует для него шаблон кода (рис. 11-3).

Хотя код, показанный на рис. 11-3, похож на обычный код Web-формы, у него есть существенные отличия:

- **классы пользовательских элементов управления объявляются с ключевым** словом ***MustInherit*** **(в Visual Basic .NET) или *abstract*** (в C#), поэтому они годятся только для объявления других классов. В данном случае его потомки — это классы пользовательских элементов управления, размещаемых на Web-форме;
- все классы **пользовательских элементов управления являются потомками** класса ***System.Web.UI.UserControl*** наследуя от него базовый набор свойств и методов, которые необходимы для создания этого элемента управления и получения значений параметров времени разработки из его HTML-атрибутов;
- **событие *Page_Load* пользовательского элемента управления** срабатывает, когда **он загружается содержащей его Web-формой,** то есть только после страничного события *Page_Load* — помните об этом, применяя пользовательские элементы управления с Web-формами.

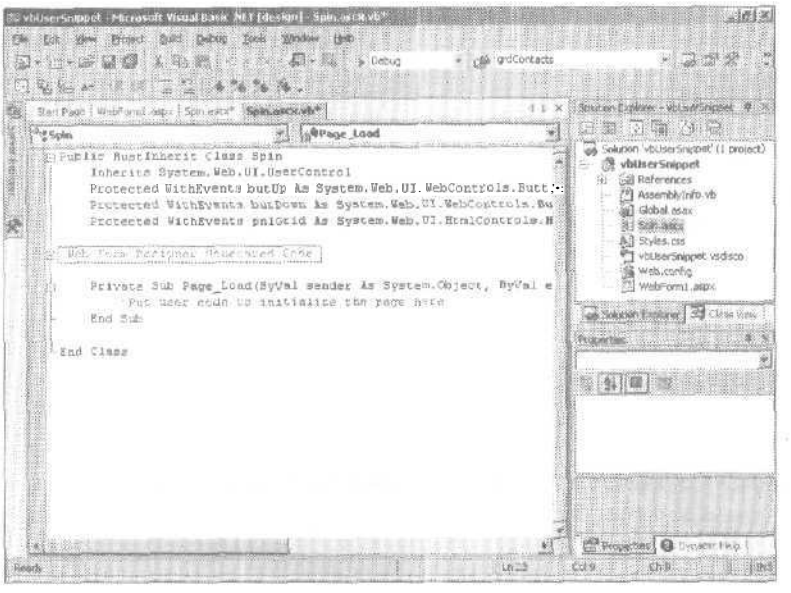


Рис. 11-3. Шаблон кода пользовательского элемента управления (Visual Basic .NET)

Чтобы создать свойства и методы пользовательского элемента управления, которые можно вызывать из кода Web-формы, выполните следующие действия.

1. Объявите открытое свойство или метод, который хотите сделать доступным в коде Web-формы, содержащей этот элемент управления.
2. Напишите код обработчиков событий для элементов управления, составляющих пользовательский элемент управления, — эти обработчики выполняют львиную долю работы пользовательского элемента управления.
3. Если свойство или метод требует сохранения некоторых параметров от одного отображения страницы до другого, добавьте к нему код для записи и восстановления нужных значений из состояния отображения (поля *ViewState*) элемента управления.

Следующий код демонстрирует свойство *Value*, которое возвращает значение элемента управления Spin, созданного в предыдущем разделе:

Visual Basic .NET

```
Public Property Value() As Integer
    Get
        ' Вернуть значение свойства.
        Return ViewState("Value")
    End Get
    Set(ByVal Value As Integer)
        ' Установить значение свойства.
        ViewState("Value") = Value
    End Set
End Property
```

Visual C#

```
public int Value
{
    get
{
        // Вернуть значение свойства.
        return Convert.ToInt32(ViewState["Value"]);
    }
    set
    {
        // Установить значение свойства.
        ViewState["Value"] = value;
    }
}
```

Пользователь изменяет значение элемента управления Spin, щелкая его кнопки. Следующий код обрабатывает щелчки, увеличивая или уменьшая значение свойства Value в зависимости от того, какую кнопку щелкнул пользователь:

Visual Basic .NET

```
Private Sub butDown_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles butDown.Click
    ' Уменьшить значение.
    Me.Value -= 1
End Sub

Private Sub butUp_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles butUp.Click
    ' Увеличить значение.
    Me.Value += 1
End Sub
```

Visual C#

```
private void butDown_Click(object sender, System.EventArgs e)
{
      // Уменьшить значение,
      this.Value -= 1;
}

private void butUp_Click(object sender, System.EventArgs e)
{
      // Увеличить значение.
      this.Value += 1;
}
```

Добавление пользовательского элемента управления к Web-форме

Пользовательские элементы управления можно перетащить на поверхность Web-формы прямо из окна Solution Explorer. При этом Visual Studio .NET генерирует директиву *Register* и HTML-теги, необходимые для создания этого элемента управления на Web-форме.

Вот пример HTML-кода, сгенерированного при размещении на Web-форме пользовательского элемента управления Spin:

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
Inherits="vbUserSnippe t.WebForm1"%>
<%@ Register TagPrefix="uc1" TagName="Spin" Src="Spin.ascx" %>
<HTML>
   <body>
         <form id="Form1" method="post" runat="server">
               <uc1:Spin id="Spin1" runat="server"></uc1:Spin>
         </form>
   </body>
</HTML>
```

Пользовательские элементы управления способны не только соседствовать с другими элементами управления на Web-форме, но и взаимодействовать с ними. Следующий HTML-код получился при добавлении Spin сразу после элемента управления TextBox:

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
Inherits="vbUserSnippe t.WebForm1"%>
<%@ Register TagPrefix="uc1" TagName="Spin" Src="Spin.ascx" %>
<HTML>
   <body>
         <form id="Form1" method="post" runat="server">
               <asp:TextBox id="TextBox1" runat="server">
               </asp:TextBox>
               <uc1:Spin id="Spin1" runat="server" Value="5">
               </uc1:Spin>
         </form>
   </body>
</HTML>
```

Заметьте, что у пользовательского элемента управления имеется атрибут для хранения свойства *Value*. Во время загрузки элемента управления Spin ASP.NET устанавливает его свойства согласно значениям HTML-атрибутов.

В режиме Design этот HTML выглядит, как на рис 11-4.

Как видно из рис. 11-4, в дизайнере Web-формы пользовательский элемент управления выглядит совсем не так, как во время выполнения: вместо кнопок со стрелками дизайнер отображает какой-нибудь универсальный элемент управления. Увы. таково ограничение Visual Studio .NET: здесь в режиме Design пользовательские элементы управления не отображаются.

Возможно, вы также заметили, что элемент управления Spin поддерживает только потоковую разметку, но это уже ограничение, присущее самому элементу управления. Чтобы организовать поддержку сеточной разметки, позволяющей задавать абсолютное положение элемента управления на Web-форме, придется добавить код для поддержки атрибутов стиля — этот прием мы продемонстрируем далее.

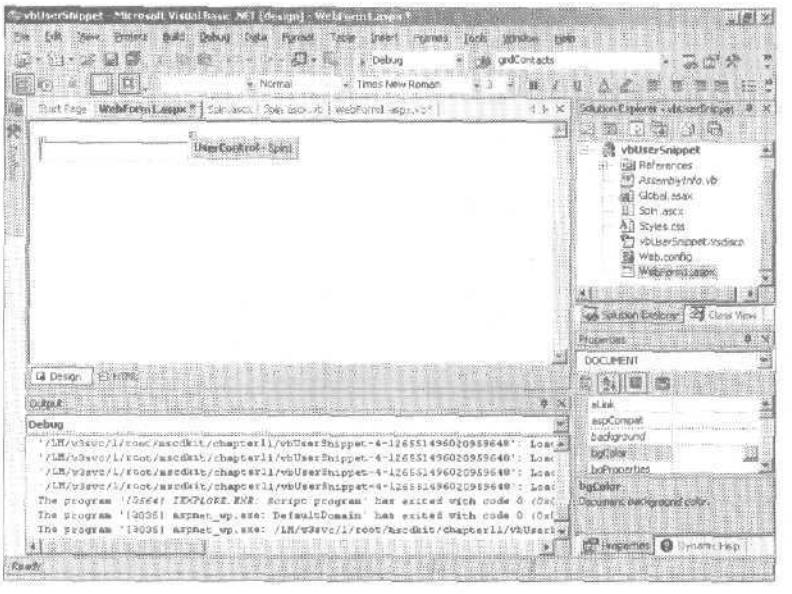


Рис. 11-4. Вид пользовательского элемента управления Spin на Web-форме

Использование элементов управления в коде

Создав пользовательский элемент управления и добавив его к Web-форме, можно задействовать его в коде. Для этого выполните следующие действия.

1. Объявите пользовательский элемент управления на уровне модуля. Следующая строка кода объявляет пользовательский элемент управления Spin, добавленный к Web-форме в предшествующем разделе:

Visual Basic .NET

```
Protected WithEvents Spin1 As Spin
```

Visual C#

```
protected Spin Spin1;
```

2. Теперь можно вызывать свойства, методы и события этого элемента, как у обычных элементов управления. Вот пример обработчика события, отображающего значение элемента управления Spin в текстовом поле:

Visual Basic .NET

```
Private Sub Page_PreRender(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles MyBase.PreRender
    ' Отобразить значение Spin.
    TextBox1.Text = Spin1.Value
End Sub
```

Visual C#

```
private void Page_PreRender(object sender, System.EventArgs e)
{
      // Отобразить значение Spin,
      TextBox1.Text = Spin1.Value.ToString();
}
```

Здесь нужно отметить одну очень важную деталь: в предыдущем коде вместо страничного события *Page_Load* используется событие *Page_PreRender* Web-формы. Иначе значение текстового поля пользовательского элемента управления Spin не обновится значением HTML-атрибута *Value*, пока вы не щелкнете одну из его кнопок (проверьте сами!). Дело в том, что код пользовательского элемента управления исполняется после обработки страничного события *Page_Load*. Никакие изменения, сохраненные в состоянии отображения пользовательского элемента управления, не загружаются, пока не сработает его событие *Page_Load*.

Объявление событий пользовательского элемента управления

Помимо свойств и методов, пользовательские элементы управления поддерживают события, позволяющие ему реагировать на действия пользователя, работающего с Web-формой.

Чтобы добавить к пользовательскому элементу управления событие, выполните следующие действия.

1. Объявите событие в модуле кода пользовательского элемента управления. Вот пример кода, объявляющего событие *Click* пользовательского элемента управления Spin:

Visual Basic .NET

```
' Объявить событие элемента управления Spin.
Public Event Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
```

Visual C#

```
// Объявить событие.
public event EventHandler Click;
```

2. Напишите метод, генерирующий это событие. В дальнейшем это облегчит создание производных классов, поскольку те смогут переопределять этот метод-

Visual Basic .NET

```
' Метод, генерирующий событие.
Protected Overridable Sub OnClick(ByVal e As EventArgs)
    RaiseEvent Click(Me, e)
End Sub
```

Visual C#

```
// Метод, генерирующий событие.
protected virtual void OnClick(EventArgs e)
{
```

```
        if (Click != null)
        {
            Click(this, e);
        }
}
```

3. **Поместите вызовы этого метода в код пользовательского элемента** управления. **Ниже** показан **пример кода,** генерирующего событие ***Click* в ответ на щелчок любой из кнопок** пользовательского элемента **управления Spin:**

Visual Basic .NET

```
Private Sub butDown_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles butDown.Click
    ' Уменьшить значение Value.
    Me.Value -= 1
    ' Вызвать метод OnClick,
    OnClick(e)
End Sub

Private Sub butUp_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles butUp.Click
    ' Увеличить значение Value.
    Me.Value += 1
    ' Вызвать метод OnClick.
    OnClick(e)
End Sub
```

Visual C#

```
private void butDown_Click(object sender, System.EventArgs e)
{
    // Уменьшить значение Value.
    this.Value -- 1;
    // Вызвать метод OnClick.
    OnClick(e);
}

private void butUp_Click(object sender, System.EventArgs e)
{
    // Увеличить значение Value.
    this.Value += 1;
    // Вызвать метод OnClick.
}
```

Чтобы задействовать событие пользовательского элемента управления в Web-форме, добавьте этот элемент управления к Web-форме, как показано ранее, и напишите обработчик для его события. Вот пример обработчика, обновляющего текстовое поле it ответ на щелчок элемента управления Spin:

Visual Basic .NET

```
Private Sub Spin1_Click(ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles Spin1.Click
    ' Обновить информацию при помощи обработчика события
    ' пользовательского элемента управления.
    TextBox1.Text = Spin1.Value
End Sub
```

Visual C#

```
private void Spin1_Click(object sender, System.EventArgs e)
{
      // Отобразить значение элемента управления Spin.
      TextBox1.Text = Spin1.Value.ToString();
}
```

Включение сеточной разметки

Как вы знаете, у пользовательских элементов управления нет встроенной поддержки абсолютного позиционирования на Web-форме, использующей сеточную разметку. В ASP.NET элементы управления поддерживают абсолютное позиционирование при помощи атрибута *style*. Чтобы включить поддержку абсолютного позиционирования во время разработки, достаточно добавить атрибут *style* к HTML-коду, объявляющему пользовательский элемент управления в Web-форме. К примеру, следующий HTML определяет пользовательский элемент управления Spin, который можно перетаскивать по поверхности Web-формы в режиме Design:

```
<UserControl:Spin id="Spin1" runat="server" Value="5"
  style="Z-INDEX: 101; LEFT: 248px. POSITION: absolute; TOP: 88px">
  </UserControl:Spin>
```

Однако, запустив Web-приложение, вы тут же столкнетесь с проблемой: во время выполнения ASP.NET игнорирует атрибут *style* и отображает пользовательский элемент управления Spin с потоковой разметкой.

Чтобы заставить атрибут *style* работать, потребуется добавить к пользовательскому элементу управления свойство *Style*, позволяющее задавать положение и размер элемента управления через значения атрибутов. Если элемент управления применяет контейнер (такой, как Grid Layout Panel в случае Spin), можно просто передать значение атрибута *style* контейнеру, как показано ниже:

Visual Basic .NET

```
Public Property Style() As String
    Get
        ' Вернуть значение атрибута style <онтейнера,
        Return pnlGrid.Attributes("style")
    End Get
    Set(ByVal Value As String)
        ' Установить значение атрибута style контейнера.
        pnlGrid.Attributes("style") = Value
    End Set
End Property
```

Visual C#

```
public string Style
{
    get
    {
        // Вернуть значение атрибута style контейнера
        return pnlGrid.Attributes["style"];
    }
    set
    {
        // Установить значение атрибута style контейнера.
        pnlGrid.Attributes["style"] = value;
    }
}
```

Эта методика программирования, которую иногда называют *делегированием* (delegation), годится и для реализации других стандартных атрибутов элементов управления, таких, как *Visible, Enabled* или *BackColor.* Для этого достаточно добавить к пользовательскому элементу управления свойство, которое передает контейнеру значение атрибута через свойство *Attributes*, как показано ранее.

Занятие 2. Создание составных элементов управления

Составные элементы управления создаются путем объединения нескольких серверных или HTML-элементов управления в одном классе, который вместе с другими классами элементов управления компилируется в сборку (.dll-файл) библиотеки нестандартных элементов управления. Однажды созданную библиотеку нестандартных элементов управления можно загрузить в Visual Studio .NET и задействовать, подобно стандартным серверным и HTML-элементам управления.

По своим функциям составные элементы управления аналогичны пользовательским, но хранятся в отдельных сборках. Это освобождает от необходимости копировать составные элементы управления в каждый проект, где они *нужны,* что приходится делать в случае пользовательских элементов управления. Однако составные элементы труднее создавать, поскольку их нельзя скомпоновать с помощью графического интерфейса дизайнера Visual Studio .NET. Но зато составные элементы управления позволяют в существенно большей степени контролировать их поведение, что, впрочем, делает их более сложными по сравнению с пользовательскими.

Изучив материал этого занятия, вы сможете:

- ✓ создавать решение, состоящее из двух проектов, для разработки и отладки составных элементов управления;
- ✓ вручную добавлять составные элементы управления к Web-форме тестового проекта при помощи директивы *Register* и HTML-тегов;
- ✓ формировать вид составного элемента управления, добавляя к нему элементы управления посредством метода *CreateChildControls*;
- ✓ создавать свойства, методы и события составного элемента управления;
- ✓ обрабатывать изменения элемента управления, вносимые во время разработки, например изменение размеров и положения на Web-форме с сеточной разметкой;
- ✓ создавать новые элементы управления, объявляя их на основе серверных элементов управления.

Продолжительность занятия — около 35 минут.

Создание и применение составных элементов управления

Процедура создания составного элемента управления и его применения в Web-приложении состоит из шести этапов.

1. Создайте решение с проектом нестандартного элемента управления.
2. Добавьте к созданному решению проект Web-приложения и сделайте его стартовым проектом. Проект Web-приложения нужен для тестирования нестандартного элемента управления во время разработки.
3. Добавьте к проекту Web-приложения ссылку на проект нестандартного элемента управления, а к Web-форме — HTML-элемент и директиву *Register,* необходимые для работы этого элемента управления.
4. Определите графический интерфейс составного элемента, добавляя к нему имеющиеся элементы управления с помощью его метода *CreateChildControls*.
5. Создайте свойства, методы и события, которые будет поддерживать составной элемент управления.

6. Скомпилируйте составной элемент управления и протестируйте его. Хотя этот этап указан последним в списке, следует выполнять его в конце каждого из предшествующих этапов.

В следующих разделах эти действия описаны более подробно.

Создание проекта нестандартного элемента управления

Составные и генерируемые элементы управления — это типы нестандартных элементов управления, представляющие собой классы, при компиляции которых получаются сборки. Большинство поведенческой логики эти классы наследуют от класса *WebControl*. В зависимости от типа создаваемых элементов управления, в них также реализуются различные интерфейсы. В этом и следующем занятии при обсуждении нестандартных элементов управления имеются в виду именно составные и генерируемые элементы управления.

Проще всего начать разработку такого элемента управления с помощью шаблона проекта Web Control Library из Visual Studio .NET.

Чтобы создать элемент управления на основе шаблона, выполните следующие действия.

1. Выберите из меню File элемент New\Project — Visual Studio .NET откроет диалоговое окно New Project, показанное на рис. 11-5.

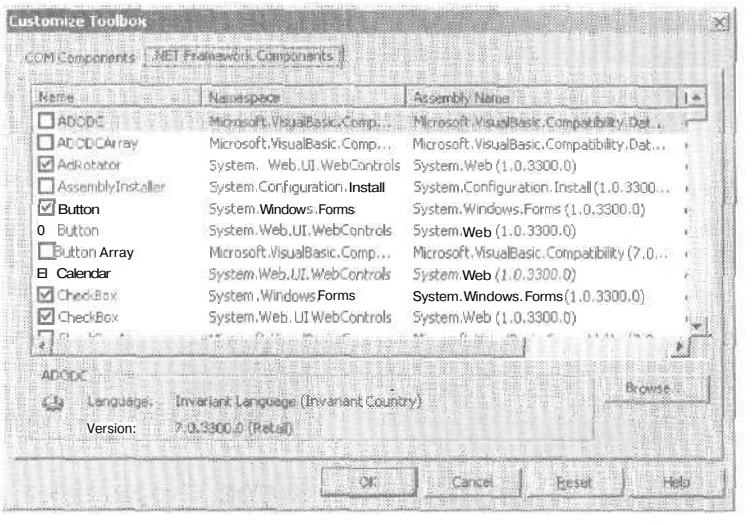


Рис. 11-5. Создание проекта нестандартного элемента управления

2. На панели Templates щелкните значок Web Control Library, введите в текстовое поле Name имя проекта и щелкните OK. В результате Visual Studio .NET создаст проект, содержащий код для единственного нестандартного элемента управления с именем WebControl1 (рис. 11-6).

Кол. показанный на рис. 11-6, представляет собой заготовку нестандартного элемента управления. В нем объявлен класс *WebControl1* с единственным свойством *Text:*

Visual Basic .NET

```
Imports System.ComponentModel
Imports System.Web.UI
```

```
<DefaultProperty("Text"), ToolboxData("<{0}:WebCustomControl1 _
runat=server></{0}:WebCustomControl1>")>
Public Class WebCustomControl1
    Inherits System.Web.UI.WebControls.WebControl

    Dim _text As String

    <Bindable(True), Category("Appearance"), DefaultValue("")> _
      Property [Text]() As String
        Get
            Return _text
        End Get

        Set(ByVal Value As String)
            _text - Value
        End Set
    End Property

     Protected Overrides Sub Render(ByVal output As _
       System.Web.UI.HtmlTextWriter)
        output.Write([Text])
    End Sub

End Class
```

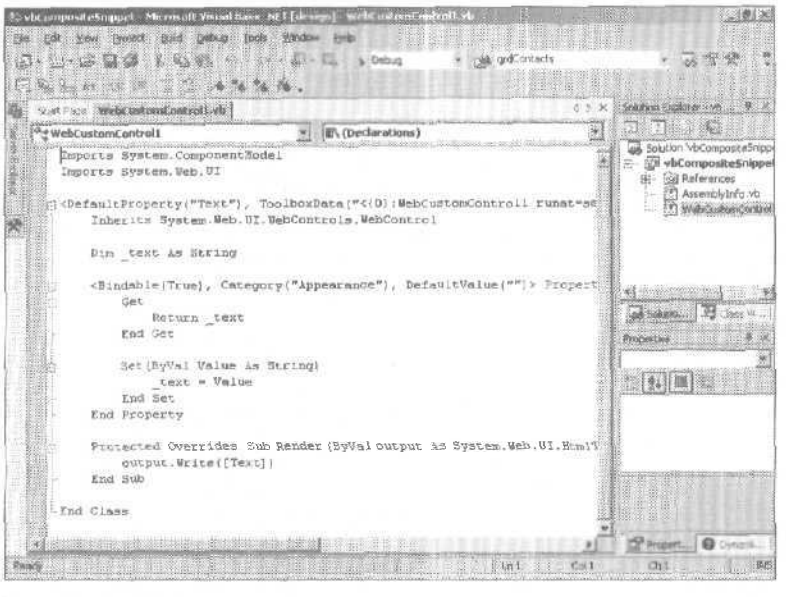


Рис. 11-6. Шаблон кода нестандартного элемента управления (на Visual Basic .NET)

Visual C#

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.ComponentModel;

namespace csCompositeSnippet
{
   /// <summary>
   /// Summary description for WebCustomControl1.
   /// </summary>
[DefaultProperty("Text"),
  ToolboxData("<{0}:WebCustomControl1
   runat=server></{0}:WebCustomControl1>")]
   public class WebCustomControl1 : System.Web.UI.WebControls.WebControl
   {
         private string text;

          [Bindable(true),
               Category("Appearance"),
               DefaultValue("")]
         public string Text
         {
               get
               {
                     return text;
               }

               set
               {
                     text = value;
               }
         }

         /// <summary>
         ///Render this control to the output parameter specified
         /// </summary>
         /// <param name="output"> The HTML writer to write out to </param>
         protected override void Render(HtmlTextWriter output)
         {
               output.Write(Text);
         }
   }
}
```

Возможно, в этом **коде вы** встретите ряд незнакомых элементов, **некоторые из них** описаны в таблице 11-1.

Таблица 11-1. Структура шаблона нестандартного элемента управления

Фрагмент шаблона	Название	Описание
`<DefaultProperty("Text"), ToolboxData("<{0}: WebCustomControl1 runat=server></{0}: WebCustomControl1>")>` `[DefaultProperty("Text"), ToolboxData("<{0}: WebCustomControl1 runat=server></{0}: WebCustomControl1>")]`	**Атрибуты** класса	Определяют параметры времени разработки для элемента управления, то есть содержимое окна свойств этого нестандартного элемента управления, отображаемое Visual Studio .NET
`Inherits System.Web.UI.WebControls.WebControl` `: System.Web.UI.WebControls.WebControl`	Базовый класс	Любой нестандартный элемент управления является потомком одного из базовых классов элементов управления
`<Bindable(True), Category("Appearance"), DefaultValue("")>` `[Bindable(true), Category("Appearance"), DefaultValue("")]`	Атрибуты свойства	Подобно атрибутам класса, определяют параметры времени разработки для свойства
`Property [Text]() As String` `public string Text`	**Определение** свойства	Устанавливает поведение свойства во время выполнения. Квадратные скобки свидетельствуют о том, что в качестве имени свойств разрешается применять ключевые слова (хотя в данном примере это не так)
`Protected Overrides Sub Render(ByVal output As System.Web.UI.HtmlTextWriter)` `protected override void Render(HtmlTextWriter output)`	Метод *Render*	Выполняет рендеринг нестандартного элемента управления. Это переопределенный метод *Render* базового класса

В следующих разделах разъясняется, как добавлять и изменять эти части шаблона при создании составного элемента управления. Подробно об атрибутах рассказано в разделе «Design-Time Attributes for Components» справочной системы Visual Studio .NET.

Создание тестового проекта

Поскольку для проекта нестандартного элемента управления, созданного в предыдущем разделе, задан выходной тип «Class Library», его можно только вызывать из другого приложения и нельзя исполнять как самостоятельное приложение. Для исполнения и отладки нестандартного элемента управления в Visual Studio .NET необходимо добавить к текущему решению еще один проект.

Чтобы создать и добавить к решению проект для тестирования нестандартного элемента управления, выполните следующие действия.

1. Откройте проект нестандартного элемента управления и выберите из меню File элемент Add Project\New Project — Visual Studio .NET откроет диалоговое окно New Project.

2. На панели Templates выберите шаблон ASP.NET Web Application, введите в текстовом поле Location имя проекта и шелкните OK. В результате Visual Studio .NET создаст новый проект Web-приложения и добавит его к текущему решению.
3. В окне Solution Explorer шелкните правой кнопкой проект Web-приложения и выберите из контекстного меню элемент Set As Startup Project. Visual Studio .NET сделает проект Web-приложения стартовым и выделит его имя в окне Solution Explorer полужирным шрифтом.
4. В окне Solution Explorer шелкните правой кнопкой элемент References проекта Web-приложения и выберите из контекстного меню элемент Add Reference — Visual Studio .NET откроет диалоговое окно Add Reference (рис.11-7).

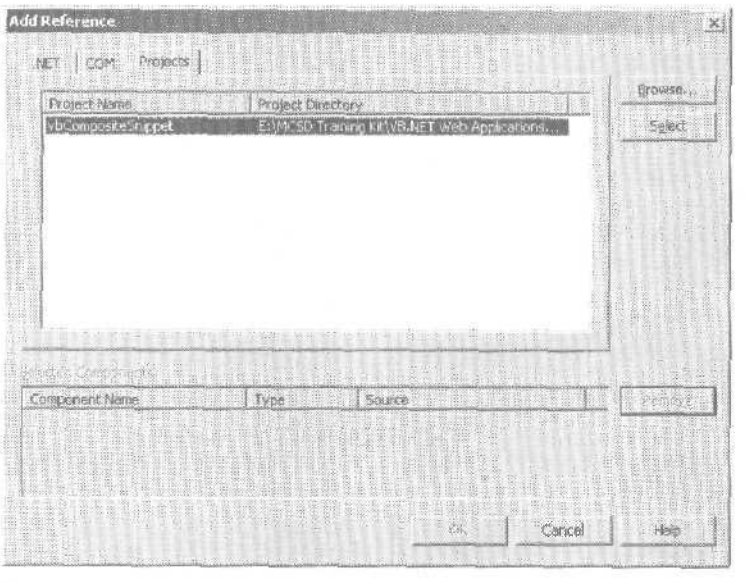


Рис. 11-7. Добавление ссылки на проект

5. Перейдите на вкладку Projects и шелкните кнопку Select, чтобы добавить к проекту Web-приложения ссылку на проект нестандартного элемента управления, и шелкните OK. Visual Studio .NET добавит ссылку в окно Solution Explorer (рис. 11-8).

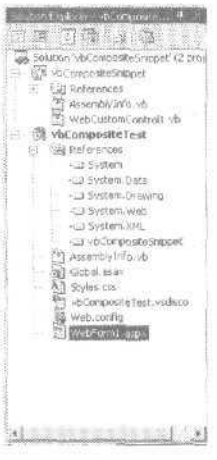


Рис. 11-8. Решение **в окне Solution Explorer**

В результате добавления в проект указанной ссылки сборка (.dll-файл) с нестандартным элементом управления копируется в каталог /bin Web-приложения, и этот элемент управления становится доступным Web-приложению. При внесении любых изменений в сборку нестандартного элемента управления ее копия в каталоге Web-приложения обновляется автоматически.

Разрабатывая нестандартный элемент управления, помните: тестовое Web-приложение не «увидит» никаких изменений, пока вы не перекомпилируете сборку с элементом управления. Перекомпиляция автоматически выполняется при запуске Web-приложения, но не при переключении между окнами элемента управления и тестирующего Web-приложения в режиме Design. Чтобы увидеть изменения, внесенные в элемент управления в режиме Design, сначала его следует перекомпилировать и только потом переключаться на тестирующее приложение.

Добавление нестандартного элемента управления к тестовому проекту

Для тестирования и отладки нестандартного элемента управления необходимо добавить его экземпляр к Web-форме тестового Web-приложении, созданного в предыдущем разделе. Это делается следующим образом.

1. Откройте Web-форму тестового Web-приложения и переключитесь в режим HTML, щелкнув ярлычок HTML в левом нижнем углу окна Design.
2. Поместите в начало Web-формы директиву *Register*. Вот пример директивы, регистрирующей сборку нестандартного элемента управления с именем *CompositeControls*:

```
<%@ Register TagPrefix="Custom" Namespace="vbCompositeSnippet"
Assembly="vbCompositeSnippet" %>
```

3. Создайте экземпляр нестандартного элемента управления на Web-форме с помощью HTML-тегов. Ниже показан пример HTML-тега, который создает экземпляр элемента управления WebCustomControl1, написанного в предыдущем разделе:

```
<Custom:WebCustomControl1 id="custTest" runat="server" />
```

В случае нестандартных элементов управления у директивы *Register* возможны три атрибута (таблица 11-2).

Таблица 11-2. Атрибуты директивы *Register*

Имя	Значение
TagPrefix	Группа пользовательского элемента управления. Например, префикс «asp» определяет принадлежность к серверным элементам управления ASP.NET. Служит для создания правил именования, упорядочивающих элементы управления
Namespace	Имя проекта и пространство имен сборки нестандартного элемента управления, в котором находится регистрируемый элемент управления. Visual Basic .NET неявно использует имя проекта в качестве пространства имен, поэтому для элементов управления, написанных на Visual Basic .NET, в качестве значения этого атрибута следует указывать имя проекта
Assembly	Имя сборки (.dll-файла), содержащей нестандартный элемент управления. В Web-приложении должна быть ссылка на сборку элемента управления. Как говорилось ранее, добавление ссылки на сборку означает ее копирование в каталог /bin Web-приложения

Префикс, заданный атрибутом *TagPrefix* директивы *Register*, составляет первую часть HTML-элемента, определяющего нестандартный элемент управления. Вторую его часть составляют имена проекта и класса элемента управления в сборке. Таким образом, если *TagPrefix=»Custom»* и имя класса — WebCustomControl1, то для создания элемента управления потребуется HTML-элемент <Custom:WebCustomControl1 .../>.

Нестандартные элементы управления способны не только сосуществовать на Web-формах с другими элементами управления, но и взаимодействовать с ними. Например, следующий HTML-код определяет нестандартный элемент управления WebControl1, добавленный сразу после элемента управления Button:

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="Default.aspx.vb"
Inherits="vbCompositeSnippet.WebForm1"%>
<%@ Register TagPrefix="Custom" NameSpace="vbCompositeSnippet"
Assembly="CompositeSnippet" %>
<HTML>
<body>
<form id="Form1" method="post" runat="server">
   <asp:button id="butCount" runat="server" />
   <Custom:WebCustomControl1 id="custTest" runat="server"
      Text="This is a custom control" />
</form>
</body>
</HTML>
```

Заметьте, что у нестандартных элементов управления имеется атрибут Text для хранения одноименного свойства. Благодаря HTML-атрибутам, значения свойств нестандартных элементов управления можно хранить в HTML-коле. В отличие от пользовательских элементов управления, нестандартные элементы управления корректно отображаются дизайнером Visual Studio .NET (рис. 11-9).

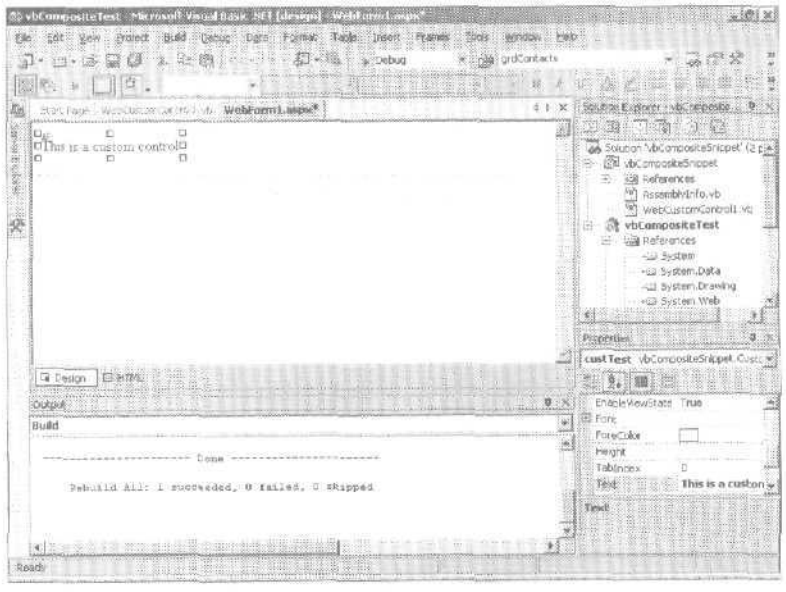


Рис. 11-9. Нестандартный элемент управления WebControll на поверхности Web-формы (во время разработки)

Следует отметить еще один момент: если выбрать нестандартный элемент управления в окне дизайнера, Visual Studio .NET покажет его свойства в окне Properties. В этом смысле нестандартные элементы также полностью поддерживаются Visual Studio .NET — в отличие от пользовательских элементов управления,

Чтобы применить нестандартный элемент управления в коде, достаточно ссылаться на него, как на обычный элемент управления. Следующий обработчик события *Click* кнопки увеличивает значение нестандартного элемента управления WebCustomControl1:

Visual Basic .NET

```
Private Sub butAdd_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles butAdd.Click
    ' Увеличить значение свойства Text нестандартного элемента
    ' управления.
    custTest.Text = Val(custTest.Text) + 1
End Sub
```

Visual C#

```
private void butAdd_Click(object sender, System.EventArgs e)
{
      // Преобразовать строку в число.
      int intText = 0;
      try
      {
            intText = int.Parse(custTest.Text);
      }
      catch
      {
            intText = 0;
      }
      // Увеличить полученное число,
      intText ++;
      // Отобразить результат в элементе управления.
     custTest.Text = intText.ToString();
}
```

При исполнении этого кода (в составе тестового Web-приложения) ASP.NET отобразит на Web-форме нестандартный элемент управления и серверный элемент управления Button (рис. 11-10).

Даже если несколько раз щелкнуть кнопку, созданную предыдущим кодом, отображаемое формой значение никогда не превысит 1. В чем же дело? А в том, что в шаблоне нестандартного элемента управления, генерированном Visual Studio .NET, отсутствует код для сохранения значений свойств между отображениями формы. Такой код придется добавлять самостоятельно, но об этом — в следующих разделах.

Определение вида составного элемента управления

Графический интерфейс составного элемента управления создается путем добавления имеющихся серверных или HTML-элементов управления к его набору *Controls* при помощи метода *CreateChildControls*, который является переопределенным методом базового класса.

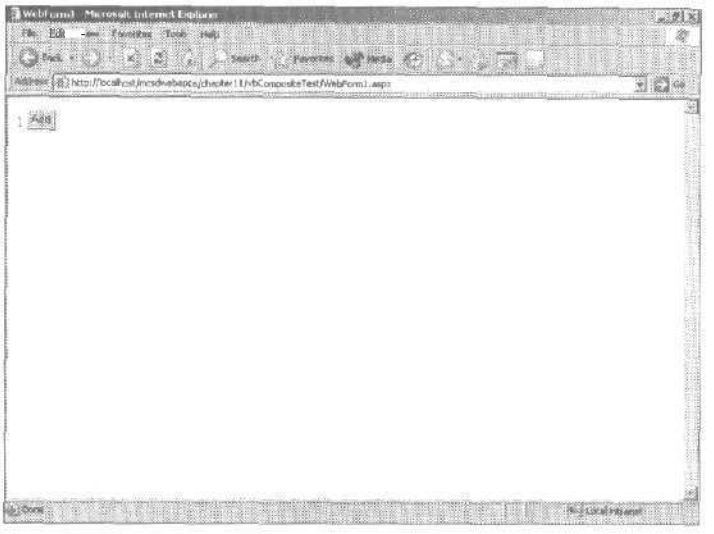


Рис. 11-10. Нестандартный элемент управления WebControl1 на поверхности Web-формы (во время выполнения)

Определение облика составного элемента управления показано на примере следующего кода. Этот код создает составной элемент управления MathBox. Он состоит из элементов управления TextBox, Button и Label, объявленных на уровне класса и, следовательно, доступных всем процедурам класса. Процедура *CreateChildControls* определяет внешний вид составного элемента управления, добавляя к его набору *Controls* указанные элементы управления, а также несколько элементов управления Literal.

Visual Basic .NET

```
Public Class MathBox
        Inherits System.Web.UI.WebControls.WebControl

        Dim txtMath As New TextBox()
        Dim butSum As New Button()
        Dim lblResult As New Label()

        ' Этот код определяет вид составного элемента управления.
        Protected Overrides Sub CreateChildControls()
            ' Добавить дочерние элементы управления.
            ' Задать свойство TextMode для элемента управления
            ' TextBox и добавить его в набор.
            txtMath.TextMode = TextBoxMode.MultiLine
            Controls.Add(txtMath)
            ' Начать новую строку.
            Controls.Add(New LiteralControl("<br>"))
            ' Задать свойство Text для элемента управления Button
            ' и добавить его в набор.
            butSum.Text = "Sum"
            Controls.Add(butSum)
```

```
            ' Добавить элементы управления Label и Literal для
            ' отображения результата.
            Controls.Add(New LiteralControl(" &nbspResult: <b>"))
            Controls.Add(lblResult)
            Controls.Add(New LiteralControl("</b>"))
        End Sub
End Class
```

Visual C#

```
public class MathBox : System.Web.UI.WebControls.WebControl
   {
        TextBox txtMath = new TextBox();
        Button butSum = new Button();
        Label lblResult = new Label();

        private string text;

        protected override void CreateChildControls()
        {
            // Добавить дочерние элементы управления.
            // Задать свойство TextMode для элемента управления
            // TextBox и добавить его в набор,
            txtMath.TextMode = TextBoxMode.MultiLine;
            Controls.Add(txtMath);
            // Начать новую строку
            Controls.Add(new LiteralControl("<br>"));
            // Задать свойство Text для элемента управления Button
            // и добавить его в набор.
            butSum.Text = "Sum";
            Controls.Add(butSum);
            // Добавить элементы управления Label и Literal
            // для отображения результата.
            Controls.Add(new LiteralControl(" &nbspResult: <b>"));
            Controls.Add(lblResult);
            Controls.Add(new LiteralControl("</b>"));

        }
   }
```

Далее на этом занятии мы будем работать с классом *MathBox*. Добавьте код *MathBox* к проекту составного элемента управления, созданного в предыдущих разделах, чтобы затем добавить этот элемент к тестовому Web-приложению. Директива *Register*, использованная ранее для добавления WebCustomControl1, поможет решить эту задачу, а также создать экземпляр *MathBox*. Вот необходимый HTML-код:

```
<Custom:MathBox id="mathTest" runat="server" />
```

Если добавить нестандартный элемент управления на Web-форму в его нынешнем состоянии, то во время разработки он выглядит как небольшой зеленый прямоугольник, но во время исполнения его вид кардинально изменится (рис. 11-11).

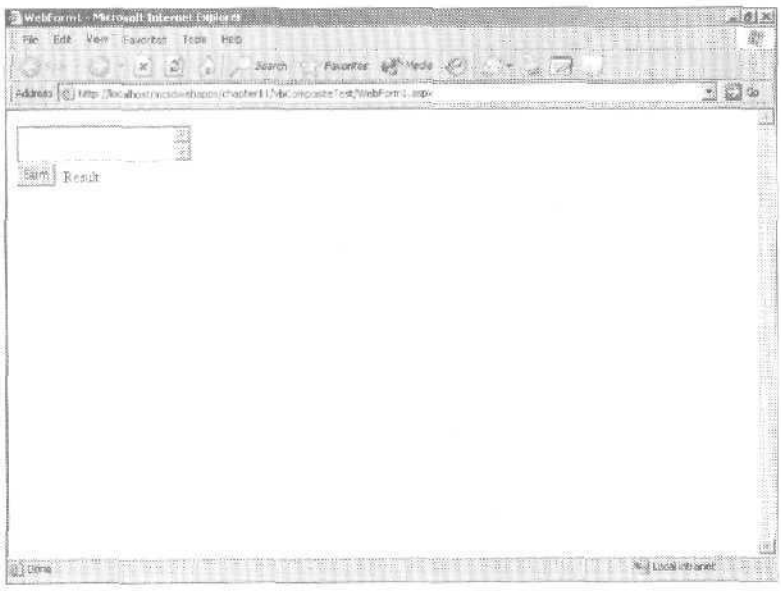


Рис. 11-11. **Вид** составного элемента **управления** MathBox **во время** выполнения

Как видите, пока MathBox не отображается должным образом во время разработки, поскольку у него нет ни одного свойства времени разработки. Мы решим эту проблему в следующем разделе.

При создании составных элементов управления не нужно переопределять метод *Render*. Несмотря на это, в шаблоне кода нестандартного элемента управления, генерируемом Visual Studio .NET, метод *Render* переопределяется. Поэтому при использовании шаблона необходимо удалить из него метод *Render,* сгенерированный Visual Studio .NET, либо изменить его так, чтобы он делегировал рендеринг обратно базовому классу (вызывая метод из базового класса), как показано здесь:

Visual Basic .NET

```
' Поскольку з составном элементе управления метод Render
' не переопределяют, делегировать рендеринг базовому классу.
    Protected Overrides Sub Render(ByVal writer As HtmlTextWriter)
        MyBase.Render(writer)
    Eno Sub
```

Visual C#

```
// Поскольку в составном элементе управления метод Render
// не переопределяют, делегировать рендеринг базовому классу.
protected override void Render(HtmlTextWriter output)
{
        base.Render(output);
}
```

Создание свойств и методов

Нестандартный элемент управления Math Box берет числа, введенные в его текстовое поле, складывает их и возвращает результат. Для решения этой задачи ему понадобятся два свойства: *Values*— для хранения массива с введенными числами и *Result* — для хранения суммы этих чисел. Кроме того, потребуется метод *Sum* для выполнения вычислений.

Следующий код демонстрирует реализацию указанного метода и свойств *Values* и *Result*. В нем также имеется свойство *Text* — оно облегчает сопряжение составного элемента управления с серверными элементами управления ASP.NET и упрощает установку текста по умолчанию при помощи атрибутов HTML.

Visual Basic .NET

```
' Реализация свойств и методов класса MathBox,
        <DefaultValue("")> Property Text() As String
            Get
                ' Проверить, существуют ли дочерние элементы управления,
                EnsureChildControls()
                ' Вернуть текст, введенный в TextBox.
                Return txtMath.Text
            End Get
            Set(ByVal Value As String)
                ' Проверить, существует ли дочерний элемент управления,
                EnsureChildControls()
                ' Записать текст в TextBox.
                txtMath.Text = Value
            End Set
        End Property

        Property Values() As String()
            Get
                EnsureChildControls0
                ' Вернуть массив, заполненный значениями из TextBox,
                Return txtMath.Text.Split(Chr(13))
            End Get
            Set(ByVal Value() As String)
                EnsureChildControls0
                ' Записать в TextBox содержимое массива строк.
                txtMath.Text = String.Join(Chr(13), Value)
            End Set
        End Property

        Readonly Property ResuIt0 As String
            Get
                EnsureChildcontrols()
                ' Вернуть результат из Label.
                Return lblResult.Text
            End Get
        End Property
```

```
    Sub Sum()
        EnsureChildcontrols()
        ' Если в TextBox есть текст. . .
        If txtMath.Text.Length Then
            ' ...разбить его на строки и записать результаты в массив.
            Dim arrNums As String()
            arrNums = txtMath.Text.Split(Chr(13))
            Dim strCount As String, dblSum As Double
            ' Сложить элементы массива.
            For Each strCount In arrNums
                ' Игнорировать все. кроме чисел, с помощью
                ' блока для обработки ошибок.
                Try
                    dblSum += Convert.ToDouble(strCount)
                Catch
                End Try
            Next
            ' Отобразить результат в Label.
            lblResult.Text - dblSum.ToString
        Else
            IblResult.Text = "0"
        End If
    End Sub
```

Visual C#

```
// Реализация свойств и методов класса MathBox.
    [DefaultValue("0")]
    public string Text
    {
        get
        {
            // Проверить, существуют ли дочерние элементы управления,
            EnsureChildControls();
            // Вернуть текст, введенный в TextBox.
            return txtMath.Text;
        }

        set
        {
            // Проверить, существуют ли дочерние элементы управления
            EnsureChildControls();
            // Записать текст в TextBox.
            txtMath.Text = value;
        }
    }

    char[] strSep = {'\r'};
```

```
public string[] Values
{
      get
      {
       EnsureChildControls();
       // Вернуть массив, заполненный значениями из TextBox.
            return txtMath.Text.Split(strSep);
      }
      set
      {
       EnsureChildControlsO;
       // Записать в TextBox содержимое массива строк.
       txtMath.Text = String.Join(" ", value);
      }
}

public string Result
{
      get
      {
       EnsureChildControls();
       // Вернуть результат из Label.
       return lblResult.Text;
      }
}

public void Sum()
{
   EnsureChildControls();
   // Если в TextBox есть текст...
   if (txtMath.Text.Length != 0)
       {
       // ...разбить его на строки и записать результаты в массив.
       string[] arrNums;
       arrNums = txtMath.Text.Split(strSep);
            double dblSum = 0;
       // Сложить элементы массива.
            foreach (string strCount in arrNums)
            {
                  // Игнорировать все, кроме чисел,
                  // с помощью блока для обработки ошибок.
                  try
                  {
                     dblSum += Convert.ToDouble(strCount);
                  }
                  catch
                  {
```

```
            }
        }
        // Отобразить результат в Label,
        lblResult.Text = dblSum.ToString();
    }
    else
        lblResult.Text = "0";
}
```

В этом коде применяется оператор *EnsureChildControls*, который проверяет, созданы ли экземпляры дочерних элементов управления. Обязательно вызывайте его, прежде чем обращаться к любому из дочерних элементов управления составного элемента управления.

Добавив свойства и методы к составному элементу управления, перекомпилируйте его, чтобы обновить сборку. После этого вид составного элемента управления в режиме Design станет таким же, как во время выполнения (рис. 11-12).

Этот элемент управления не виден во время разработки, но вы можете отобразить его, установив одно из его свойств следующим образом, как показано ниже:

```
<Custom:MathBox id="mthTest" runat="server" Text="0" />
```

Поскольку составные элементы управления вызывают метол *Render* из базового класса, они автоматически поддерживают абсолютное позиционирование с использованием сеточной разметки. Иными словами, можно перетащить составной элемент управления в любое место Web-формы, и он останется там во время выполнения.

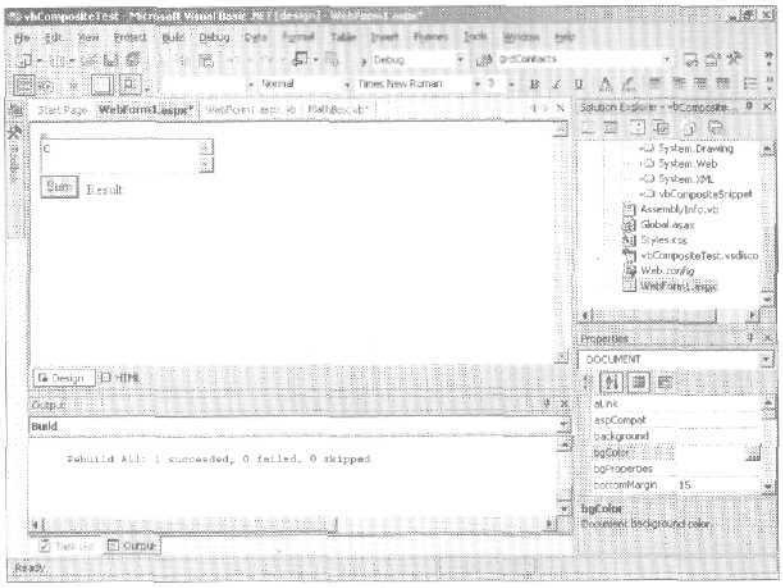


Рис. 11-12. Вид нестандартного элемента управления MathBox во время разработки

Обработка событий

Итак, теперь MathBox способен получать значения, выполнять вычисления и отображать результат, но не реагирует на щелчки кнопки Sum.

Чтобы кнопка Sum заработала, сделайте следующее.

1. Добавьте к процедуре *CreateChildControls* обработчик события *Click* кнопки, воспользовавшись методом *AddHandler*.
2. Напишите обработчик для этого события. Его аргументы и тип возвращаемого результата должны совпадать с таковыми события *Click*.

Для этого необходимо добавить к классу *MathBox* код, выделенный в показанном ниже примере **полужирным шрифтом.** Заметьте, что обработчик события просто вызывает созданный ранее метод *Sum*, выполняющий расчет и отображение результата.

Visual Basic .NET

```
Protected Overrides Sub CreateChildControls()
    ' Добавить дочерние элементы управления.
    ' Задать свойство TextMode для элемента управления TextBox
    ' и добавить его в набор.
    txtMath.TextMode = TextBoxMode.MultiLine
    Controls.Add(txtMath)
    ' Начать новую строку
    Controls.Add(New LiteralControl("<br>"))
    ' Задать свойство Text для Button и добавить Button
    ' и добавить его в набор.
    butSum.Text = "Sum"
    Controls.Add(butSum)
    ' Добавить элементы управления Label и Literal для отображения
    ' результата.
    Controls.Add(New LiteralControl(" &nbspResult: <b>"))
    Controls.Add(lblResult)
    Controls.Add(New LiteralControl("</b>"))
    ' Добавить обработчик события.
    AddHandler butSum.Click, AddressOf Me.butSumClicked
End Sub

' Это обработчик события Click кнопки
Sub butSumClicked(ByVal source As Object, ByVal e As EventArgs)
    ' Вызвать метод Sum.
    Sum()
End Sub
```

Visual C#

```
protected override void CreateChildControls()
{
   // Добавить дочерние элементы управления.
   // Задать свойство TextMode для элемента управления TextBox
   // и добавить его в набор.
   txtMath.TextMode = TextBoxMode.MultiLine;
   Controls.Add(txtMath);
   // Начать новую строку
```

```
    Controls.Add(new LiteralControl("<br>"));
    // Задать свойство Text для Button и добавить его в набор.
    butSum.Text = "Sum";
    Controls.Add(butSum);
    // Добавить элементы управления Label и Literal для отображения
    // результата.
    Controls.Add(new LiteralControl(" &nbspResult : <b>"));
    Controls.Add(lblResult);
    Controls.Add(new LiteralControl("</b>"));
    // Добавить обработчик события Click кнопки.
    butSum.Click += new EventHandler(butSumClicked);
}

void butSumClicked(object sender, EventArgs e)
{
      // Вызвать метод Sum.
      Sum();
}
```

Генерация событий

Помимо событий *Click* кнопки Sum вам придется генерировать события, которые будет обрабатывать Web-форма, содержащая нестандартный элемент управления.

Чтобы наделить нестандартный элемент управления способностью генерировать события, выполните следующие действия.

1. Объявите в классе нестандартного элемента управления открытое событие.
2. Чтобы генерировать событие, вызовите в коде этого элемента управления соответствующий метод.

Вот код, демонстрирующий объявление и генерацию событий в классе *MathBox*.

Visual Basic .NET

```
' Объявить событие.
      Event Click(ByVal sender As Object, ByVal e As EventArgs)

      ' Это обработчик события Click кнопки.
      Sub butSumClicked(ByVal source As Object, ByVal e As EventArgs)
          ' Вызвать метод Sum.
          Sum()
          ' Вызвать метод, генерирующий событие,
          OnClick(EventArgs.Empty)
      End Sub

      Protected Overridable Sub OnClick(ByVal e As EventArgs)
          ' Генерировать событие.
          RaiseEvent Click(Me, e)
      End Sub
```

Visual C#

```
// Объявить событие,
        public event EventHandler Click;

        void butSumClicked(object sender, EventArgs e)
        {
              // Вызвать метод Sum.
              Sum();
              // Вызвать метод, генерирующий событие.
              OnClick(EventArgs. Empty);
        }

        protected virtual void OrClick(EventArgs e)
        {
              // Генерировать событие.
              Click(this, e);
        }
```

Чтобы сделать событие *Click* событием по умолчанию для элемента управления, добавьте к объявлению класса атрибут *DefaultEvent*.Событие по умолчанию — это событие, которое Visual Studio .NET автоматически генерирует по двойному щелчку элемента управления на Web-форме. Ниже показан пример объявления события по умолчанию (см. код, выделенный **полужирным шрифтом**):

Visual Basic .NET

```
<DefaultEvent("Click")> Public Class MathBox
```

Visual C#

```
[DefaultEvent("Click")]
   public class MathBox : System.Web.UI.WebControls.WebControl
```

Чтобы задействовать событие элемента управления MathBox в Web-форме, дважды щелкните этот элемент управления, размещенный на тестовой Web-форме, и добавьте следующий обработчик события:

Visual Basic .NET

```
Private Sub mathTest_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles mathTest.Click
    Response.Write(mathTest.Result)
End Sub
```

Visual C#

```
private void mthTest Click(object sender, System.EventArgs e)
{
      Response.WriteCmthTest.Result.ToString());
}
```

Изменение размеров элемента управления

Многие параметры отображения во время разработки и во время выполнения автоматически обрабатываются составными элементами управления путем вызова метода *Render* из базового класса. Однако они не способны автоматически изменять размеры дочерних элементов управления, поскольку нуждаются в указаниях о том, как именно это следует делать.

Чтобы организовать обработку изменения размеров составного элемента управления, выполните следующие действия.

1. Переопределите метод базового класса *Render*.
2. Добавьте к нему код, должным образом изменяющий размеры дочерних элементов управления.
3. Вызовите метод *Render* из базового класса, чтобы отобразить составной элемент управления.

Следующий код изменяет размеры дочернего элемента управления TextBox в зависимости от высоты и ширины элемента управления MathBox:

Visual Basic .NET

```
Protected Overrides Sub Render(ByVal writer As _
  System.Web.UI.HtmlTextWriter)
    EnsureChildcontrols()
    ' Изменить ширину текстового поля в соответствии с шириной MathBox.
       txtMath.Width = Me.Width
       ' Изменить высоту текстового поля в соответствии с высотой MathBox.
       txtMath.Height = Unit.Parse(Me.Height.Value butSum.Height.Value)
       ' Выполнить рендеринг элемента управления,
       MyBase.Render(writer)
    End Sub
```

Visual C#

```
protected override void Render(HtmlTextWriter output)
{
   EnsureChildcontrols0;
   // Изменить ширину текстового поля в соответствии с шириной MathBox,
   txtMath.Width = this.Width;
   // Изменить высоту текстового поля в соответствии с высотой MathBox,
      couble dHeight = this.Height.Value - butSum.Height.Value;
   txtMath.Height = Unit.Parse(dHeight.ToString());
   // Выполнить рендеринг элемента управления,
      base.Render(output);
}
```

Создание производных элементов управления

В предыдущих разделах говорилось о создании составных элементов управления путем комбинирования их из нескольких существующих элементов управления. Те же методы позволяют создавать новые элементы управления на основе единственного существующего элемента управления, добавлять к нему свойства, методы и события, а также изменять его поведение.

Поскольку нестандартные элементы управления такого типа создаются на основе единственного элемента управления, можно порождать их классы непосредственно от класса базового элемента управления, а не от общего класса *WebControl*. Такие элементы управления не являются настоящими составными элементами управления, поэтому их часто называют *производными* (superclassed controls).

Следующий пример класса *SuperText* демонстрирует объявление нестандартного элемента управления на основе серверного элемента управления TextBox и добавление метода, сортирующего содержимое текстового поля;

Visual Basic .NET

```
Public Class SuperText
    Inherits System.Web.UI.WebControls.TextBox

    ' Объявить новый метод для TextBox.
    Public Sub Sort()
        ' Создать массив.
        Dim arrText As String()
        ' Поместить слова из текстового поля в массив.
        arrText = Me.Text.Split(" ")
        ' Отсортировать массив,
        Array.Sort(arrText)
        ' Собрать строку из элементов массива и поместить ее
        ' обратно в TextBox.
        Me.Text = String.Join(" ", arrText)
    End Sub

End Class
```

Visual C#

```
public class SuperText : System.Web.UI.WebControls.TextBox
{

    public void Sort()
    {
        // Создать массив.
        string[] arrText;
        // Поместить слова из текстового поля в массив,
        char[] strSep = {' '};
        arrText = this.Text.Split(strSep);
        // Отсортировать массив.
        Array.Sort(arrText);
        // Собрать строку из элементов массива и поместить ее
        // обратно в TextBox.
        this.Text = String.Join(" ", arrText);
    }
}
```

Чтобы использовать этот элемент управления с Web-формой, зарегистрируйте его сборку, как показано выше, и добавьте к Web-форме следующий HTML:

```
<Custom:SuperText id="superTest" runat="server" />
<asp:Button ID="butSort" Runat="server" Text="Sort" />
```

Для тестирования метода *Son* добавьте к модулю кода Web-формы следующий обработчик:

Visual Basic .NET

```
Private Sub butSort_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles butSort.Click
    superTest.Sort()
End Sub
```

Visual C#

```
private void butSort_Click(object sender, System.EventArgs e)
{
      superTest.Sort();
}
```

И во время выполнения, и во время разработки элемент управления SuperText ведет себя, как обычное текстовое поле (рис. 11-13).

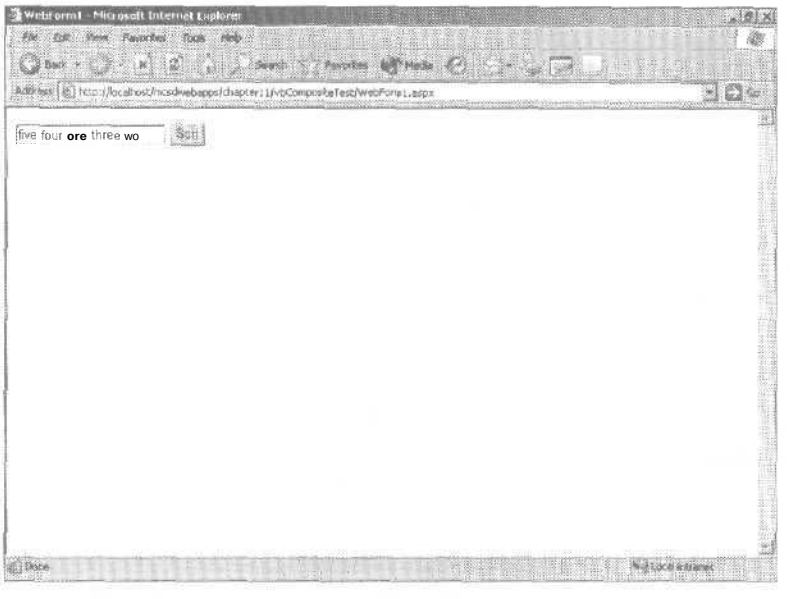


Рис. 11-13. Элемент управления SuperText в действии

Занятие 3. Создание генерируемых элементов управления

Генерируемые нестандартные элементы управления создаются практически «с нуля», поэтому разработчик полностью контролирует их облик и функции. Но, поскольку у них нет дочерних элементов управления, придется самостоятельно писать код для решения задач, которые составные элементы управления обычно делегируют дочерним элементам.

Поскольку составные и генерируемые элементы управления тесно связаны, на этом занятии вы большей частью упрочите навыки, полученные на занятии 2. Материалы некоторых разделов в равной степени применимы к составным и к генерируемым элементам управления, так что это занятие можно считать посвященным углубленному изучению разработки нестандартных элементов управления вообще. Тем не менее в главе 11 данная тема раскрыта далеко не полностью, в качестве руководства по разработке нестандартных элементов управления мы рекомендуем книгу Nikhil Kothari и Vandana Datye *«Developing ASP.NET Server Controls and Components»*, вышедшую в издательстве Microsoft Press в 2002 году.

Изучив материал этого занятия, вы сможете:

- ✓ формировать облик нестандартного элемента управления с помощью методов *Render* и служебных методов объекта *HtmTextWriter,*
- ✓ сохранять значения свойств в состоянии отображения элемента управления и восстанавливать их оттуда;
- ✓ получать текст и элементы управления, расположенные между открывающим и замыкающим HTML-тегами нестандартного элемента управления;
- ✓ реагировать на действия, выполненные пользователем над элементом управления, генерацией кэшируемых событий, а также событий postback;
- ✓ принимать данные от пользователя;
- ✓ добавлять нестандартные элементы управления на панель Toolbox в Visual Studio .NET;
- ✓ изменять значок на панели Toolbox и значение HTML-тэга *TagPrefix* для элемента управления в Visual Studio .NET.

Продолжительность занятия — около 40 минут.

Создание и использование генерируемых элементов управления

После выполнения всех действий, описанных в занятии 2, вы должны получить генерируемый элемент управления. Фактически заготовка, которую создает Visual Studio .NET для проекта Web Control Library, уже является генерируемым элементом управления, правда, очень простым и не слишком полезным. Компонуя готовые элементы управления, удается создавать более интересные элементы управления и делать это быстрее, поэтому в данной главе составные элементы управления рассматриваются первыми.

Поскольку генерируемые и составные элементы управления очень похожи, при создании первых годятся все приемы, о которых шла речь в предыдущем занятии, кроме одного: их интерфейс формируется только элементами HTML (а не другими элементами управления). Поэтому в вашем распоряжении не будет готовых свойств, методов и событий для получения и отображения данных — все это придется создавать самостоятельно.

Первые три этапа создания генерируемых и составных элементов управления совпадают (см. занятие 2).

1. Создайте решение с проектом нестандартного элемента управления.
2. Добавьте к созданному решению проект Web-приложения и сделайте его стартовым проектом. Web-приложение понадобится для тестирования нестандартного элемента управления в ходе разработки.
3. Добавьте к проекту Web-приложения ссылку на проект элемента управления, а к Web-форме — директиву *Register* и элемент HTML, необходимые для использования нестандартного элемента управления на Web-форме.
4. Создайте интерфейс нестандартного элемента управления, переопределив метод *Render* базового класса.
5. Напишите необходимый код, сохраняющий значения свойств, реагирующий на действия пользователя и принимающий от него данные.

Последние этапы подробно описаны в следующих разделах. Заключительные разделы занятия посвящены добавлению готовых нестандартных элементов управления на панель Toolbox, а также выбору значка и тегов, используемых Visual Studio .NET для элемента управления. Изложенный в них материал в равной степени относится и к составным элементам управления.

Определение облика генерируемого элемента управления

Облик генерируемого элемента управления формируют, переопределяя метод *Render* базового класса и записывая в аргумент output этого метода элементы HTML с помощью служебных методов объекта *HtmlTextWriter* (таблица 11-3),

Таблица 11-3. Служебные методы HtmlTextWriter

Метод	Описание
AddAttribute	Добавляет атрибуты к следующему элементу HTML, который будет сформирован
RenderBeginTag	Формирует открывающий тег HTML-элемента, который затем будет записан с помощью метода *WriteLine*
RenderEndTag	Формирует замыкающий тег HTML-элемента и записывает этот элемент, а также любые атрибуты, формирование которых еще не закончено. После вызова *RenderEndTag* все атрибуты считаются сформированными
Write	Немедленно записывает строку
WriteAttribute	Немедленно записывает HTML-атрибут
WriteBeginTag	Немедленно записывает открывающий тег HTML-элемента
WriteEndTag	Немедленно записывает замыкающий тег HTML-элемента
WriteFullBeginTag	Немедленно записывает начальный тег вместе с замыкающей скобкой (>)
WriteLine	Немедленно записывает строку содержимого. Эквивалент метода *Write*, отличающийся от него лишь добавлением к концу строки символа перехода на новую строку

Как видно из таблицы 11-3, существует два способа записи элементов HTML в аргумент output. Наиболее прямолинейный — запись HTML непосредственно в объект *Html-*

TextWriter с помощью методов *Write*. Вот пример кода, создающего кнопку, которая в ответ на щелчок показывает сообщение:

Visual Basic .NET

```
Imports System.ComponentModel
Imports System.Web.UI

Public Class AlertBJtton
    Inherits System.Web.UI.WebControls.WebControl

    Protected Overrides Sub Render(ByVal output As _
      System.Web.UI.HtmlTextWriter)
        ' Записать заголовок
       output.Write("<h3>Rendered Control</h3>")
       ' Открыть HTML-тег INPUT (вставить "<INPUT").
       output.WriteBeginTag("INPUT")
       ' Записать некоторые атрибуты.
       output.WriteAttribute("value", "Custom Button")
       output.WriteAttribute("type", "button")
       output.WriteAttribute("onclick", "javascript:alert('Howdy!')")
       ' Закрыть HTML-тег INPUT (вставить ">").
       output.WriteEndTag("INPUT")
    End Sub

End Class
```

Visual C#

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.ComponentModel;

namespace csRenderedSnippet
{
   public class AlertButton : System.Web.UI.WebControls.WebControl
   {
         protected override void Render(HtmlTextWriter output)
         {
              // Записать заголовок.
              output.Write("<h3>Rendered Control</h3>");
              // Открыть HTML-тег INPUT (вставить "<INPUT").
              output.WriteBeginTag("INPUT");
              // Записать некоторые атрибуты.
              output.WriteAttribute("value", "Custom Button");
              output.WriteAttribute("type", "button");
              output.WriteAttribute("onclick", "javascript:alert('Howdy!')");
              // Закрыть HTML-тег INPUT (вставить ">").
```

```
            output.WriteEndTag("INPUT");
        }
    }
}
```

Можно добавить этот код к проекту нестандартного элемента управления и скомпилировать его, чтобы создать сборку с элементом управления. Если добавить полученный элемент управления к Web-форме, то во время выполнения он примет вид, как на рис. 11-14.

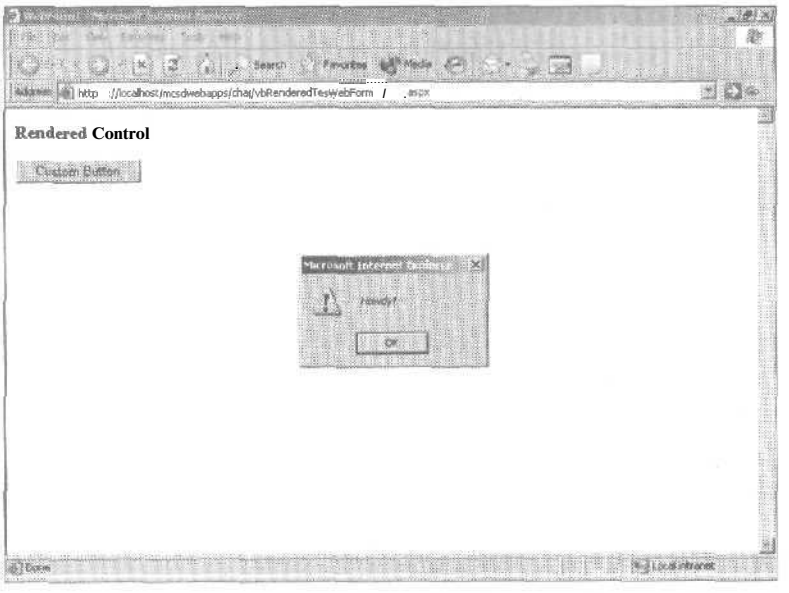


Рис. 11-14. Генерируемый элемент управления AlertButton в действии

Альтернативный подход — заранее сформировать необходимые теги и атрибуты, а затем записать их все вместе вызовом метода *RenderEndTag*. Следующий код демонстрирует использование этого подхода на примере реализации метода *Render* нестандартного элемента управления AlertButton;

Visual Basic .NET

```
Protected Overrides Sub Render(ByVal output As _
      System.Web.UI.HtmlTextWriter)
        ' Записать заголовок.
        output.Write("<h3>Rendered Control</h3>")
        ' Записать некоторые атрибуты.
        output.AddAttribute("value", "Custom Button")
        output.AddAttribute("type", "button")
        output.AddAttribute("onclick", "javascript:alert('Howdy!')")
        ' Открыть HTML-тег INPUT (вставить "<INPUT").
        output.RenderBeginTag("INPUT")
        ' Закрыть HTML-тег INPUT (вставить ">").
        output.RenderEndTag()
    End Sub
```

Visual C#

```
protected override void Render(HtmlTextWriter output)
{
   // Записать заголовок.
   output.Write("<h3>Rendered Control</h3>");
   // Записать некоторые атрибуты.
   output.AddAttribute("value", "Custom Button");
   output.AddAttribute("type", "button");
   output.AddAttribute("onclick", "javascript:alert('Howdy!')");
   // Открыть HTML-тег INPUT (вставить "<INPUT").
   output.RenderBeginTag("INPUT");
   // Закрыть HTML-тег INPUT (вставить ">").
   output.RenderEndTag();
}
```

Преимущество этого способа заключается в том, что он позволяет создавать **HTML**-атрибуты в любом порядке.

Сохранение значений свойств

Свойства составного элемента управления, показанного в главе 2, сохранялись в свойствах его дочерних элементов управления. Поскольку у генерируемого элемента управления нет дочерних элементов управления, значения его свойств между отображениями страницы приходится сохранять в его состоянии отображения (*ViewState*).

Следующий код отображает значение свойства Text, сохраняемого между отображениями страницы:

Visual Basic .NET

```
Property Text() As String
    Get
        ' Если знамение свойства установлено...
        If Not IsNothing(ViewState("Text")) Then
            ' ... вернуть его,
            Return ViewState("Text")
        Else
            ' иначе вернуть пустую строку,
            Return ""
        End If
    End Get
    SetCByVal Value As String)
        ' Сохранить значение свойства.
        ViewStateC'Text") = Value
    End Set
End Property
```

Visual C#

```
public string Text
{
    G
```

```
        {
        // Если значение свойства установлено...
         if (ViewState["Text"] != null)
             // ...вернуть его,
             return ViewState["Text"].ToString();
         else
             // иначе вернуть пустую строку.
             return null;
        }

        set
        {
             // Сохранить значение свойства.
             ViewState["Text"] = value;
        }
}
```

При сохранении в состоянии отображения данные сериализуются **и записываются в скрытое поле** *ViewState***Web-формы. После возврата страницы на сервер выполняется восстановление значений из состояния отображения, при этом данные извлекаются из скрытого поля и** десериализуются.

Состояние отображения оптимизировано для хранения данных типа *String*, *Array List* и *HashTable*, но в нем разрешается хранить любые типы данных, которые могут быть сериализованы или поддерживают метод *TypeConverter*.Для остальных типов данных придется самостоятельно написать перегруженные версии методов *SaveViewState* и *LoadViewState*, сохраняющие и восстанавливающие значения из состояния отображения.

В HTML-элементах часто присутствует текст, так или иначе модифицируемый ими. Вот пример HTML-тега, отображающего текст, выделенный полужирным шрифтом:

```
<b>Bold me</b>
```

Чтобы получить текст из нестандартного элемента управления, выполните следующие **действия.**

1. Добавьте к определению класса элемента управления атрибут *ParseChildren*.
2. Реализуйте в классе нестандартного элемента управления интерфейс *INamingContainer*.
3. Для извлечения текста используйте набор *Controls*.

Ниже показан пример класса нестандартного элемента управления, который извлекает текст, заключенный между открывающим и замыкающим тегами нестандартного элемента управления, и отображает его красным цветом:

Visual Basic .NET

```
' Атрибут ParseChildren нужен для создания дочерних элементов управления,
<ParseChildren(False)> Public Class Red
    Inherits WebControls.WebControl
    ' Этот интерфейс позволяет получить массив Controls.
    Implements INamingContainer

    Protected Overrides Sub Render(ByVal output As _
      System.Web.UI.HtmlTextWriter)
        ' Вернуть текст, заключенный между тэгами, в значении LiteralControl.
```

```
        Dim litText As LiteralControl
        ' Получить текст из набора Controls.
        litText = Controls(0)
        ' Сделать текст красным с помощью тега <font>.
        output.Write("<font color='red'>" & litText.Text & "
        </font>")
    End Sub

End Class
```

Visual C#

```
[ParseChildren(false)]
    public class Red : System.Web.UI.WebControls.WebControl, INamingContainer
    !
    // Интерфейс INamingContainer позволяет получить массив Controls,

        protected override void Render(HtmlTextWriter output)
        {
            // Вернуть текст, заключенный между тэгами,
            // в значении LiteralControl.
            LiteralControl litText;
            // Получить текст из набора Controls.
            litText = (LiteralControl)Controls[0];
            // Сделать текст красным с помощью тега <font>.
            output.Write("<font color='red'>" + litText.Text +
            "</font>");
        }
    }
```

Следующий HTML позволит использовать этот нестандартный элемент управления на Web-форме:

```
<custom:red id="Red1" runat="server">Some red text</custom:red>
```

Набор *Controls* позволяет получать любые элементы, заключенные между открывающим и замыкающим тегами нестандартного элемента управления, включая элементы управления. Например, следующий класс нестандартного элемента управления группирует элементы управления с помощью панели, располагает их в центре страницы и отображает на красном фоне:

Visual Basic .NET

```
' Атрибут ParseChildren нужен для создания дочерних элементов управления.
    <ParseChildren(False)> Public Class Center
        Inherits WebControls.WebControl
        ' Этот интерфейс позволяет получить массив Controls.
        Implements INamingContainer

        Protected Overrides Sub Render(ByVal output As _
          System.Web.UI.HtmlTextWriter)
```

```
            ' Добавить атрибуты панели.
            output.AddAttribute("align", "center")
            output.AddStyleAttribute("BACKGROUND-COLOR", "red")
            ' Начать панель.
            output.RenderBeginTag("div")
            ' Определить переменные для хранения дочерних элементов управления,
            Dim ctrItem As Control
            ' Для каждого дочернего элемента управления:
            For Each ctrltem In Controls
                ' ...выполнить рендеринг в HTML.
                ctrItem.RenderControl(output)
            Next
            output.RenderEndTag()
        End Sub

    End Class
```

Visual C#

```
[ParseChildren(false)]
   public class Center : System.Web.UI.WebControls.WebControl,
     INamingContainer
   {
   // Интерфейс INamingContainer делает возможным получение массива Controls.

        protected override void Render(HtmlTextWriter output)
        {
              // Добавить атрибуты панели.
              output.AddAttribute("align", "center");
              output.AddStyleAttribute("BACKGROUND-COLOR", "red");
              // Начать панель.
              output.RenderBeginTag("div");
              // Для каждого дочернего элемента управления:
              foreach (Control ctrltem in Controls)
              {
                    // ... выполнить рендеринг в HTML.
                    ctrItem.RenderControl(output);
              }
              output.RenderEndTagO;
        }
   }
```

Обработка действий пользователя

Составные элементы управления применяют для обработки действий пользователя возможности своих дочерних элементов управления (см. занятие 2). В случае генерируемых элементов управления обработкой действий пользователя придется заниматься особо. Способ обработки действий пользователя зависит от типа события:

1. кэшируемые события допустимо генерировать в коде нестандартного элемента управления;
2. события, возвращающие форму на сервер (события postback), необходимо генерировать на стороне клиента с помощью сценария, который формируется нестандартным элементом управления.

Способы генерации этих событий описаны в следующем разделе.

Генерация кэшируемых событий

Кэшируемые события обрабатываются серверным кодом после возврата Web-формы на сервер по завершении обработки события *Page_Load* и события, вызвавшего возврат формы. Чтобы генерировать кэшируемое событие, следует выполнить два действия:

1. объявить это событие;
2. генерировать его, вызвав в коде нестандартного элемента управления соответствующий метод.

Ниже показан пример класса нестандартного элемента управления с событием *OnChange*, которое срабатывает при изменении свойства *Text:*

Visual Basic .NET

```
<DefaultEvent("OnChange")>Public Class RenderText
        Inherits WebControls.WebControl

        Event Change(ByVal sender As Object. ByVai e As EventArgs)

        Property Text() As String
            Get
                ' Если значение свойства Text установлено...
                If Not IsNothing(ViewState("Text")) Then
                    ' ...вернуть его,
                    Return ViewState("Text")
                Else
                    ' иначе вернуть пустую строку.
                    Return ""
                End If
            End Get
            Set(ByVal Value As String)
                ' Сохранить значение свойства,
                ViewState("Text") = Value
                ' Вызвать метод, генерирующий событие.
                OnChange(EventArgs.Empty)
            End Set
        End Property

        ' Метод, генерирующий событие.
        Protected Overridable Sub OnChange(ByVal e As EventArgs)
            ' Генерировать событие.
            RaiseEvent Change(Me, EventArgs.Empty)
        End Sub
```

```
        Protected Overrides Sub Render(ByVal output As _
          System.Web.UI.HtmlTextWriter)
            ' Установить атрибуты элемента Input.
            output.AddAttribute("value", Me.Text)
            ' Создать элемент Input.
            output.RenderBeginTag("Input")
            ' Закрыть элемент  Input  (вставить />).
            output.RenderEndTag()
        End Sub

    End Class
```

Visual C#

```
[DefaultEvent("Change")]
   public class RenderText : System.Web.UI.WebControls.WebControl
   {
         // Объявить событие.
         public event EventHandler Change;

         public string Text
         {
               get
               {
                // Если значение свойства Text установлено. . .
                if (ViewState["Text"] != null)
                     // . . .вернуть его,
                     return ViewState["Text"].ToString();
                else
                    // иначе вернуть пустую строку.
                    return null;
               }
               set
               {
                     // Сохранить значение свойства.
                     ViewState["Text"] = value;
                     // Вызвать метод, генерирующий событие.
                     OnChange(EventArgs.Empty);
               }
         }

         protected virtual void OnChange(EventArgs e)
         {
               if (Change != null)
                      // Генерировать событие.
                      Change(this, e);
         }
```

```
        protected override void Render(HtmlTextWriter output)
        {
            // Установить атрибуты элемента Input.
            output.AddAttribute("value", this.Text);
            // Создать элемент Input .
            output.RenderBeginTag("Input");
            // Закрыть элемент  Input  (вставить />).
            output.RenderEndTag();
        }
    }
```

Заметьте, что в этом определении класса событие *Change* является событием по умолчанию для элемента управления, представленного классом. Следовательно, в ответ на двойной щелчок этого элемента управления в окне дизайнера Visual Studio .NET сгенерирует обработчик для события *Change*.

Добавьте к Web-форме следующий код для обработки события Change:

Visual Basic .NET

```
Private Sub txtTest_OnChange(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles txtTest.OnChange
        Response.Write("Text changed! ")
    End Sub
```

Visual C#

```
private void txtTest_Change(object sender, System.EventArgs e)
        {
            Response.Write("Text changed! ");
        }
```

Генерация событий postback

События postback инициируют возврат Web-формы на сервер, где она немедленно обрабатывается. Это событие обрабатывается сразу после страничного события *Page_Load*. Чтобы генерировать такое событие из нестандартного элемента управления, выполните следующие действия.

1. Реализуйте в нестандартном элементе управления интерфейс *IPostBackEventHandler*.
2. Объявите событие, которое нужно генерировать.
3. Сгенерируйте элемент HTML, способный реагировать на клиентские события, такие, как *onclick*.
4. Добавьте к этому элементу HTML атрибут со сценарием, возвращающим страницу на сервер.
5. Генерируйте событие, вызвав метод . *Raise Post Back Event,* объявленный в интерфейсе *IPostBackEvent Handler*.

Следующий код демонстрирует генерацию postback-события *Click* из кода нестандартного элемента управления:

Visual Basic .NET

```
' Генерация события postback.
<DefaultEvent("Click")> Public Class AlertButton
```

```
    Inherits System.Web.UI.WebControls.WebControl
    ' (1) Интерфейс, необходимый для обработки событий postback.
    Implements IPostBackEventHandler

    ' (2) Объявить событие postback.
    Event Click(ByVal sender As Object, ByVal e As EventArgs)


    ' (3) Генерировать элемент HTML, реагирующий на события,
    '     инициируемые пользователем.
    Protected Overrides Sub Render(ByVal output As _
      System.Web.UI.HtmlTextWriter)
        ' Добавить атрибуты.
        output.AddAttribute("value", "Postback Button")
        output.AddAttribute("type", "button")
        ' (4) Добавить атрибуты, необходимые для генерации
        '     события postback на стороне клиента.
        output.AddAttribute("onclick", "javascript:alert('Howdy!');" & _
          Page.GetPostBackEventReference(Me))
        ' Открыть HTML-тег Input (вставить "<INPUT").
        output.RenderBeginTag("INPUT")
        ' Закрыть HTML-тег Input (вставить ">" и записать атрибуты).
        output.RenderEndTag()
    End Sub

    ' Это метод, объявленный в интерфейсе IPostBackEventHandler.
    ' Его реализацию вы должны написать самостоятельно,
    Public Sub RaisePostBackEvent(ByVal eventArgument As String) _
      Implements System.Web.UI.IPostBackEventHandler.RaisePostBackEvent
        ' (5) Вызвать метод, генерирующий событие.
        OnClick(EventArgs.Empty)
    End Sub

    Protected Overridable Sub OnClick(ByVal e As EventArgs)
        ' Генерировать событие.
        RaiseEvent Click(Me, EventArgs.Empty)
    End Sub

End Class
```

Visual C#

```
// (1) Интерфейс IpostBackEventHandler необходим для обработки событий postback,
[DefaultEvent("Click")] public class AlertButtonThree :
  System.Web.UI.WebControls.WebControl, IPostBackEventHandler
{
    // (2) Объявить событие postback.
    public event EventHandler Click;
```

```
// (3) Генерировать элемент HTML, реагирующий на события.
// инициируемые пользователем.
protected override void Render(HtmlTextWriter output)
{
      // Записать заголовок
      output.Write("<h3>Rendered Control</h3>");
      // Добавить атрибуты.
      output.AddAttribute("value", "Custom Button");
      output.AddAttribute("type", "button");
      // (4) Добавить атрибуты, необходимые для генерации
      //     события postback на стороне клиента.
      output.AddAttribute("onclick", "javascript:alert('Howdy!');" +
            Page.GetPostBackEventReference(this));
      // Открыть HTML-тег Input (вставить "<INPUT").
      output.RenderBeginTag("INPUT");
      // Закрыть HTML-тег Input(вставить ">").
      output.RenderEndTag();
}

// Это метод, объявленный в интерфейсе
IPostBackEventHandler.
// Его реализацию вы должны написать самостоятельно.
public void RaisePostBackEvent(string eventArgument)
{
      // (5) Вызвать метод, генерирующий событие.
      OnClick(EventArgs.Empty);
}

protected virtual void OnClick(EventArgs e)
{
      // Генерировать событие.
      if (Click != null)
            Click(this, e);
}

}
```

Если пользователь во время выполнения щелкнет этот элемент управления, в результате события *onclick* кнопки исполняется следующий сценарий:

```
alert('Howdy!');              //Показать сообщение,
__doPostBack('winTest','');  // Generated by
                              // Page.GetPostBackEventReference(Me);
```

Метод _ *doPostBack()* возвращает страницу на сервер, где оно перехватывается методом *RaisePostBackEvent* интерфейса *IPostBackEventHandler,* который и генерирует событие *Click* из кода элемента управления.

Добавьте к Web-форме следующий код для обработки события *Click:*

Visual Basic .NET

```
Private Sub altTest_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles altTest.Click
        Response.Write("Button clicked!")
    End Sub
```

Visual C#

```
private void altTest(object sender, System.EventArgs e)
        {
            Response.Write("Button clicked!");
        }
```

Прием данных от пользователя

Генерируемые элементы управления не способны автоматически сохранять данные, введенные пользователем. В этом можно убедиться, создав генерируемый элемент управления TextBox на основе HTML-элемента *Input.* Введите в *Input* какой-нибудь текст, но как только страница вернется на сервер, все, что вы напечатали, исчезнет.

Чтобы получить данные, введенные пользователем, выполните следующие действия.

1. Реализуйте интерфейс *I Post BackData Handler.*
2. Добавьте атрибут *name,* уникально идентифицирующий элемент HTML, из которого берутся данные.
3. Переопределите метод *Load Post BackData* из интерфейса *IpostBackDataHandler* — этот метод понадобится для получения данных от пользователя.
4. Переопределите метод *RaisePostDataChanged 'Event* из интерфейса *IpostBackDataHandler.* Хотя писать реализацию этого метода не обязательно, переопределить его необходимо, так как он является частью интерфейса.
5. Генерируйте событие для уведомления об изменении данных. Это тоже не обязательно: если вы не хотите, чтобы у элемента управления было событие *change,* данный этап можно опустить.

Эта процедура иллюстрируется следующим кодом нестандартного элемента управления, который сохраняет данные, введенные пользователем в нестандартный элемент управления TextBox, генерируемый путем рендеринга HTML-элемента *Input:*

Visual Basic .NET

```
<DefaultEvent("Change")> Public Class RenderText
        Inherits WebControls.WebControl
        ' (1) Интерфейс IPostBackDataHandler позволяет
        '     получить данные при возврате страницы.
        Implements IPostBackDataHandler

        Event Change(ByVal sender As Object, ByVal e As EventArgs)

        Property Text.() As String
            Get
                ' Если значение свойства Text задано...
                If Not IsNothing(ViewState("Text")) Then
                    ' ...вернуть его,
```

```
            Return ViewState("Text")
        Else
            ' иначе вернуть пустую строку.
            Return ""
        End If
    End Get
    Set(ByVal Value As String)
        ' Сохранить значение свойства,
        ViewState("Text") = Value
        ' Вызвать метод, генерирующий событие.
        OnChange(EventArgs.Empty)
    End Set
End Property

Protected Overrides Sub Render(ByVal output As _
  System.Web.UI.HtmlTextWriter)
    ' Установить атрибуты элемента Input.
    output.AddAttribute("value", Me.Text)
    ' (2) Этот атрибут обязателен для получения данных
    ' при возврате страницы.
    output.AddAttribute("name", Me.UniqueID)
    ' Создать элемент Input с этими атрибутами.
    output.RenderBeginTag("Input")
    ' Закрыть элемент Input.
    output.RenderEndTag()
    ' Записать созданный элемент в HtmlTextWriter.
    output.WriteLine()
End Sub

' (3) Получить данные от пользователя. Это необходимый метод.
Public Function LoadPostData(ByVal postDataKey As String, _
ByVal postCollection As _
System.Collections.Specialized.NameValueCollection) As Boolean _
Implements System.Web.UI.IPostBackDataHandler.LoadPostData
    ' Если пользователь изменил значение Input,
    ' обновить значение свойства соответствующим образом,
    If Me.Text <> postCollection(postDataKey) Then
        Me.Text = postCollection(postDataKey)
        ' Если возвращено значение True,
        ' ниже вызывается метод RaisePostDataChangedEvent.
        Return True
    Else
        ' Если возвращено значение False,
        ' метод RaisePostDataChangedEvent не вызывается.
        Return False
    End If
End Function
```

```
        ' (4) Перегруженный метод RaisePostDataChangedEvent.
        '     Это необходимый метод.
        Public Sub RaisePostDataChangedEvent() Implements _
          System.Web.UI.IPostBackDataHandler.RaisePostDataChangedEvent
            ' (5) Вызвать метод, генерирующий событие (не обязательно),
            OnChange(EventArgs.Empty)
        End Sub

        ' Метод, генерирующий событие.
        Protected Gverridable Sub OnChange(ByVal e As EventArgs)
            ' Генерировать событие
            RaiseEvent Change(Me, EventArgs.Empty)
        End Sub
    End Class
```

Visual C#

```
// (1) Интерфейс IPostBackDataHandler позволяет
//     получить данные при возврате страницы.
  [DefaultProperty("Text"), DefaultEvent("Change")] public class RenderText :
     System.Web.UI.WebControls.WebControl, IPostBackDataHandler
   {
        // Объявить событие.
        public event EventHandler Change;

         [DefaultValue("")]
        public string Text
        <
              get
              {
               // Если значение свойства Text задано...
               if (ViewState["Text"] != null)
                   // ...вернуть его,
                   returnViewState["Text"].ToString();
               else
                   // иначе вернуть пустую строку.
                   return null;
              }

              : et
              {
                    // Сохранить значение свойства.
                    ViewState["Text"] = value:
                    // Вызвать метод, генерирующий событие.
                    OnChange(EventArgs.Empty);
              }
        }
```

```
protected override void Render(HtmlTextWriter output)
{
   // Установить атрибуты элемента Input.
   output.AddAttribute("value", this.Text);
   // (2) Этот атрибут обязателен для получения данных
   // при возврате страницы.
   output.AddAttribute("name", this.UniqueID);
   // Создать элемент Input с этими атрибутами.
   output.RenderBeginTag("Input");
   // Закрыть элемент Input (вставить />).
   output.RenderEndTag();
}

// (3) Получить данные от пользователя.
//     Это необходимый метод.
public bool LoadPostOata(string postDataKey,
  System.Collections.Specialized.NameValueCollection postCollection)
{
   // Если пользователь изменил значение Input,
   // обновить значение свойства соответствующим образом.
      if (this.Text != postCollection[postDataKey])
      {
            this.Text = postCollection[postDataKey];
            // Если возвращено значение True, ниже вызывается
            // метод RaisePostDataChangedEvent.
            return true;
      }
   else
       // Если возвращено значение False,
       // метод RaisePostDataChangedEvent не вызывается,
       return false;
}

// (4) Перегруженный метод RaisePostDataChangedEvent.
//     Это необходимый метод.
public void RaisePostDataChangedEvent()
{
   // (5) Вызвать метод, генерирующий событие (не обязательно).
   OnChange(EventArgs.Empty);
}

protected virtual void OnChange(EventArgs e)
{
      if (Change != null)
            // Генерировать событие.
            Change(this, e);
}
```

Добавление нестандартных элементов управления на панель Toolbox

Созданные и отлаженные нестандартные элементы управления можно добавить на панель Toolbox Visual Studio .NET. Это делается следующим образом.

1. Выберите в меню Tools команду Customize Toolbox — Visual Studio .NET откроет диалоговое окно Customize Toolbox (рис. 11-15).
2. Перейдите на вкладку .NET Framework Components и щелкните кнопку Browse — Visual Studio .NET откроет диалоговое окно Open.
3. В окне Open найдите сборку (.dll-файл) с нестандартным элементом управления, который нужно добавить, и выберите ее, щелкнув кнопку Open. Visual Studio .NET откроет сборку и добавит содержащиеся в ней элементы управления к списку в диалоговом окне Customize Toolbox.
4. Щелкните OK, чтобы закрыть диалоговое окно Customize Toolbox.

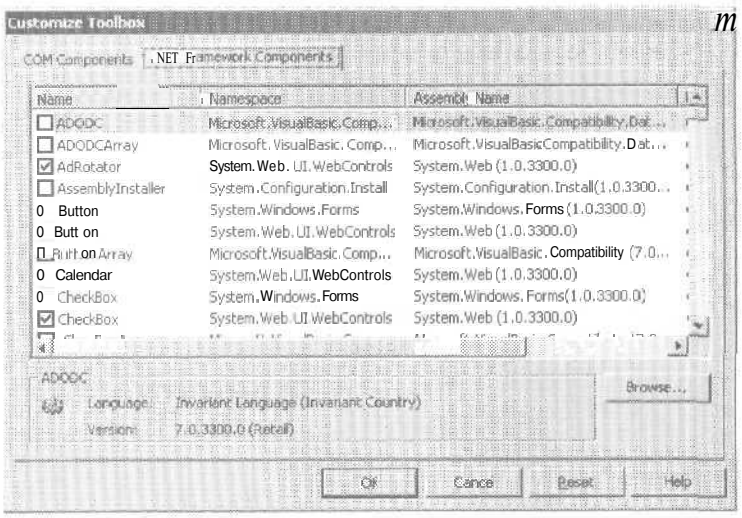


Рис. 11-15. Добавление элементов на панель Toolbox

В режиме Design Visual Studio .NET показывает нестандартные элементы управления, добавленные на панель Tolbox, в категории General (рис. 11-16).

Теперь достаточно дважды щелкнуть нестандартный элемент управления или перетащить его с Toolbox на поверхность формы, чтобы поместить его на Web-форму. При этом Visual Studio .NET автоматически добавит к Web-форме директиву *Register* и необходимый HTML-код (он выделен полужирным шрифтом в показанном ниже примере):

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
Inherits="vbRenderedTest.WebForm1"%>
<%@ Register TagPrefix="cc1" Namespace="vbRenderedSnippet"
Assembly="vbRenderedSnippet" %>
<HTML>
        <form id="Form1" method="post" runat="server">
            <cc1:Chart id="Chart1" runat="server"></cc1:Chart>
        </form>
    </body>
</HTML>
```



Рис. 11-16. Нестандартные элементы управления на панели Toolbox

Атрибуты сборки нестандартного элемента управления позволяют изменять значок, представляющий этот элемент на панели Toolbox, и элемент *TagPrefix*, используемый для ссылки на этот элемент управления в HTML-коде. Об этом — в следующем разделе.

Определение значка Toolbox и значения *TagPrefix*

Вы можете снабдить свой элемент управления уникальным значком, представляющим его на панели инструментов, и задать для него значение *TagPrefix*, используемое в HTML-коде. Если вы успели добавить элементы управления на панель Toolbox, придется удалить их оттуда и заново добавить, чтобы внесенные изменения вступили в силу.

Чтобы назначить нестандартному элементу управления значок для отображения на панели Toolbox, выполните следующие действия.

1. Создайте файл растрового изображения значка размером 16 на 16 пикселов Сохраните файл в каталоге /bin проекта элемента управления.

   Добавьте в начало модуля класса элемента управления следующую строку:

   Visual Basic .NET

   ```
   Imports System.Drawing
   ```

   Visual C#

   ```
   using System.Drawing;
   ```

2. Добавьте к определению класса элемента управления атрибут *ToolboxBitmap*. Например, следующая строка задает для нестандартного элемента управления Red файл значка с именем Red:

   Visual Basic .NET

   ```
   <ParseChildren(False), ToolboxBitmap("Red")> Public Class Red
   ```

   Visual C#

   ```
   [ParseChildren(false), ToolboxBitmap("Red")]
   ```

3. Перекомпилируйте сборку элемента управления.

Чтобы изменить значение TagPrefix, используемого в HTML для ссылки на элемент управления, выполните следующие действия.

1. Добавьте в файл AssemblyInfo из проекта нестандартного элемента управления следующие строки:

Visual Basic .NET

```
Imports System.Web.UI

<Assembly: TagPrefix("project.namespace", "prefix")>
```

Visual C#

```
using System.Web.UI

[assembly: TagPrefix("namespace", "prefix")]
```

2. Перекомпилируйте сборку элемента управления,

Здесь *project.namespace* — соответственно имя проекта и пространство имен нестандартного элемента управления, а *prefix* — префикс для объявления этого элемента управления из пространства имен *namespace*.

Резюме

- Пользовательские элементы управления создаются посредством размещения серверных и HTML-элементов управления на странице пользовательского элемента управления (такие страницы хранятся в файлах с расширением .ascx).
- Чтобы поместить пользовательский элемент управления на Web-форму, его нужно перетащить на поверхность формы из окна Solution Explorer, а не с панели Toolbox.
- В проекте, где применяется пользовательский элемент управления, должна быть копия этого элемента. Пользовательские элементы управления инициализируются только после обработки события *Page_Load* Web-формы, на которой они размещены,
- Классы составных элементов управления объявляют на основе класса *WebControl*.
- Составные элементы управления состоят из серверных и HTML-элементов управления, добавляемых с помощью метода *CreateChildControls*.
- Чтобы расширить возможности существующего серверного элемента управления, объявите на основе его класса новый класс нестандартного элемента управления.
- Генерируемые элементы управления позволяют в наибольшей степени контролировать свой облик и поведение, но требуют писать код для решения задач, которые составные элементы управления перекладывают на свои дочерние элементы.
- Облик генерируемых элементов управления формируют с помощью метода *Render* и методов объекта *HtmlTextWriter*.
- Интерфейс *IPostBackEventHandler* позволяет инициировать события postback в генерируемом элементе управления.
- Интерфейс *IPostBackDataHandler* позволяет генерируемому элементу управления получать данные, вводимые в него пользователем во время выполнения.

Практикум 11. Создание нестандартных элементов управления



В этом практикуме вы создадите три нестандартных элемента управления, составляющие проект тестового Web-приложения. Законченное приложение со всеми элементами управления показано на рис. 11-17. Оно также есть на прилагаемом к книге компакт-диске (в зависимости от языка оно называется vbUserControlSample или csUserControlSample).

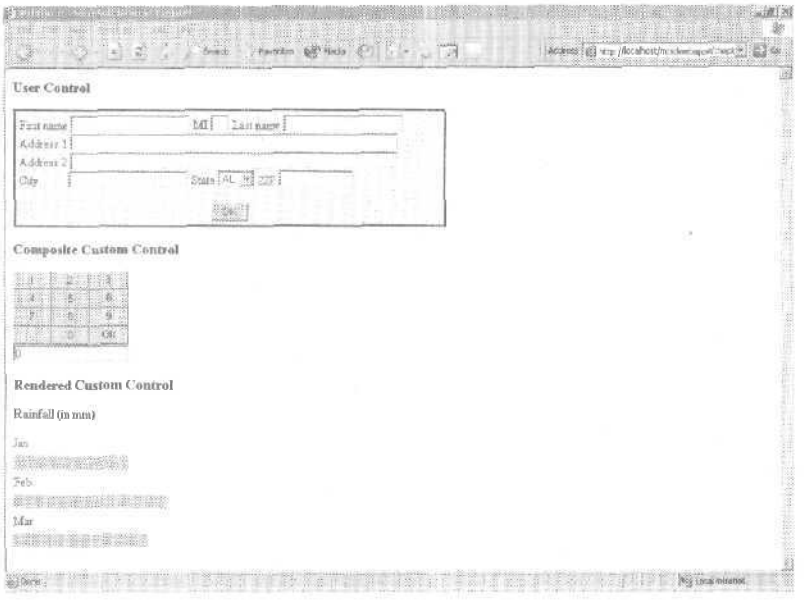


Рис. 11-17. Готовое тестовое приложение с нестандартными элементами управления

Продолжительность практикума — около 60 минут.

Упражнение 1. Создание пользовательского элемента управления AddressBox



Сейчас вы создадите Web-приложение, содержащее пользовательский элемент управления для ввода имени и адреса. Применив навыки, полученные в занятии 1, вы построите и протестируете пользовательский элемент управления. При этом придерживайтесь следующей примерной схемы.

1. Создайте проект Web-приложения.
2. Добавьте к проекту пользовательский элемент управления (.ascx-файл) с текстом и элементами управления, необходимыми для ввода фамилии и адреса (состоящего из города, штата и почтового индекса).
3. Для проверки наличия данных в обязательных полях добавьте к пользовательскому элементу управления верификаторы.
4. Для форматирования и позиционирования текста, а также компонентов пользовательского элемента управления возьмите HTML-элемент управления Grid Layout Panel.

5. Создайте событие *Click*, генерируемое в коле пользовательского элемента управления сразу после проверки введенных данных.
6. Определите открытое неизменяемое свойство для получения сведений об адресе из пользовательского элемента управления.
7. Создайте открытое изменяемое свойство *Style* для поддержки абсолютного позиционирования с использованием сеточной разметки.
8. Протестируйте созданный вами элемент управления. Для этого поместите его на Web-форму и вызывайте его событие и свойства из кода.



Закончив проект, сравните свои результаты с примером пользовательского элемента управления AddressBox, который записан на прилагаемом к книге компакт-диске.

Упражнение 2. Создание составного элемента управления DigitPad



Ваша задача — создать на основе шаблона Web Control Library библиотеку Web-элементов управления с нестандартным элементом управления, который позволит пользователю вводить числа, щелкая кнопки. Применяя навыки, полученные в занятии 2, вы создадите и протестируете составной элемент управления, Вот примерный план решения этой задачи.

1. Добавьте к решению, созданному в предыдущем упражнении, проект Web Control Library.
2. Добавьте к проекту пользовательского элемента управления ссылку на проект Web Control Library, чтобы получить возможность тестирования составного элемента управления с помощью уже созданного Web-приложения.
3. Создайте составной элемент управления, использующий HTML-элементы управления InputButton для ввода чисел.
4. Создайте событие postback, генерируемое из кода составного элемента управления по завершении ввода чисел.
5. Определите открытое неизменяемое свойство, возвращающее числа, введенные при помощи этого элемента управления.
6. Протестируйте созданный элемент управления. Для этого поместите этот элемент на Web-форму и вызывайте его событие и свойства из кода.



Закончив, сравните свои результаты с примером составного элемента управления DigitPad. который находится на прилагаемом компакт-диске.

Упражнение 3. Создание генерируемого элемента управления BarChart



Сейчас вы добавите к библиотеке, созданной в упражнении 2, генерируемый элемент управления. Он должен строить простую столбчатую диаграмму на основе чисел, введенных между открывающим и замыкающим тегами этого элемента управления. Применяя навыки, полученные на занятии 3, и придерживаясь примерного плана решения задачи, вы создадите и протестируете генерируемый элемент управления.

1. Добавьте к созданному ранее проекту Web Control Library новый класс, созданный на основе шаблона Web Custom Control Class.

2. **Создайте свойство** *Value,* **которое получает массив с текстовыми** элементами, введенными между открывающим **и** замыкающим **тэгами генерируемого** элемента **управления** (его вы **создадите в этом упражнении).**
3. Создайте свойство *ChartTitle*, позволяющее устанавливать и получать текст заголовка диаграммы.
4. Переопределите метод *Render* этого элемента управления, поместив в него код для создания диаграммы на основе значения свойства *Value.* Проще всего построить диаграмму с помощью HTML-элементов управления Horizontal Rule (<HR>).
5. Протестируйте генерируемый элемент управления. Для этого поместите его на Web-форму и вызывайте его события и свойства из кода.



Закончив, сравните свои результаты с примером генерируемого элемента управления BarChart, который находится на прилагаемом компакт-диске.

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. Кратко опишите основные области применения пользовательских, составных и генерируемых элементов управления.

2. Почему наследование так важно для создания нестандартных Web-элементов управления?

3. **Назовите главный метод, переопределяемый при** создании **составных элементов** управления.

4. Назовите главный метод, переопределяемый при создании генерируемых элементов управления.

5. Чем отличается инициирование событий postback в составных и генерируемых элементах управления?

ГЛАВА 12

Работа с мультимедиа

Занятие 1. Воспроизведение звука 522

Занятие 2. Воспроизведение видео 532

Занятие 3. Анимация на Web-страницах 538

В этой главе

Разнообразие видов предоставляемой информации играет не последнюю роль в популярности Интернета, На страницах Web-сайтов текст и изображения соседствуют с музыкой, видео и анимацией. Из этой главы вы узнаете, как добавлять к Web-приложениям ASP.NET компоненты для работы с мультимедиа, научитесь управлять в коде их отображением, а также воспроизведением звука и видео.

Прежде всего

Для занятий по материалам этой главы необходимо:

- понимать жизненный цикл Web-форм **ASP.NET;**
- уметь редактировать Web-формы в режиме HTML;
- знать, как создаются клиентские сценарии на JScript и VBScript в коде Web-страниц.

Занятие 1. Воспроизведение звука

При перезагрузке Web-страницы с фоновым звуком браузер перезапускает его воспроизведение. Это происходит из-за особенностей методики, используемой ASP.NET для серверной обработки событий, генерированных на клиенте. Для обработки событий страница возвращается на сервер. При отправке страницы ASP.NET останавливает воспроизведение звука, а после получения отклика сервера начинает его снова. В силу этих обстоятельств лучше управлять воспроизведением звука и другого мультимедийного содержимого с помощью клиентских сценариев, а не серверного кода.

В главе 7 говорилось, как добавлять к Web-формам сценарии, которые реагируют на события, генерированные на клиенте, и управляют браузером. Из этого занятия вы узнаете, как с помощью элементов HTML обогатить Web-форму или HTML-страницу звуками, и научитесь управлять воспроизведением звука при помощи клиентских сценариев.

Изучив материал этого занятия, вы сможете:

- ✓ описать преимущества и недостатки различных способов воспроизведения звука на Web-страницах;
- ✓ воспроизводить фоновый звук и вставлять фоновые звуки из серверного кода;
- ✓ воспроизводить звук на Web-странице с использованием пользовательского проигрывателя по умолчанию;
- ✓ применять объекты ActiveX и подключаемые модули с Web-страницами;
- ✓ воспроизводить звук с помощью объектов ActiveX, представляющих проигрыватели Media Player и RealPlayer;
- ✓ организовывать поддержку браузера Netscape Navigator, добавляя объекты ActiveX в виде *внедренных модулей* (embedded plug-ins).

Продолжительность занятия — около 30 минут.

Способы воспроизведения звука

Существуют три основных способа воспроизведения звука на Web-формах ASP.NET:

- в фоне — с помощью HTML-элемента *bgsound*;
- в виде **внедренного звукового файла** — с помощью HTML-элемента *embed*, вставляющего в Web-страницу ссылку на пользовательский проигрыватель мультимедийных файлов;
- через **объект ActiveX** — с использованием HTML-элемента *object* для вставки на страницу объекта ActiveX, представляющего проигрыватель.

Первая методика поддерживается почти всеми браузерами и платформами, но не позволяет управлять воспроизведением мультимедийного содержимого, что возможно при использовании двух других методик. Более подробно все эти методики описаны далее.

Воспроизведение фоновых звуков

HTML-элемент *bgsound* проигрывает звук во время отображения Web-страницы. Этот элемент не доступен в виде HTML-элемента управления, поэтому вы не найдете его на панели Toolbox в Visual Studio .NET.

Чтобы добавить к Web-форме фоновый звук, выполните следующие действия.

1. В Visual Studio вызовите окно Web-формы и переключитесь в режим HTML.
2. Вставьте в HTML-код страницы элемент bgsound, например, так:

```
<bgsound src="test.wav" id="bgTest" loop="infinite">
```

Сразу же можно заметить, что воспроизведение фонового звука перезапускается при каждом отображении страницы. Это не проблема в случае обычных HTML-страниц, но Web-формы ASP.NET перезагружаются при каждом событии, возвращающем форму на сервер: примером может быть событие *Click* серверного элемента управления Button.

Если воспроизведение звука должно начинаться только при первом отображении страницы, проверьте страничное свойство *IsPostBack*, чтобы определить, нужно ли добавлять фоновый звук к отклику. Вот пример кода, который проигрывает фоновую мелодию только при первом отображении страницы:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
  Этот код проигрывает мелодию только при первом отображении страницы.
 If Not (IsPost.Back) Then
 Response.Write("<bgsound src='test.wav' id='bgsTest' loop='1'>")
 Enc If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
 // Этот код проигрывает мелодию только при первом отображении страницы.
 if (!IsPostBack)
 Response.Write(""<bgsound src='test.wav' id='bgsTest' loop='1'>"[G1]);
}
```

Аналогичным приемом можно воспользоваться и для воспроизведения звука в ответ на другие события, генерируемые на странице, такие, как щелчки кнопок. Воспроизведение фоновых звуков *аддитивно,* то есть звуки, воспроизводимые одновременно, накладываются друг на друга.

Внедрение звуковых файлов

Одно из ограничений фонового звучания заключается в том, что им нельзя управлять после начала воспроизведения. Воспроизведение начинается во время загрузки страницы и не прекращается, пока не кончится звуковой файл либо не откроется другая страница. Чтобы пользователь имел возможность запускать и останавливать воспроизведение, а также регулировать громкость звука, возьмите HTML-элемент *embed.* Следующий HTML-код внедряет звуковой файл и отображает на странице интерфейс подключаемого модуля пользовательского проигрывателя по умолчанию:

```
<embed src="test.wav" id="wavTest" hidden="false" height="20" width="150"
autostart="false" type="audio/ wav" loop="true">
</embed>
```

Подключаемый модуль, отображаемый этим HTML-кодом, зависит от проигрывателя, установленного на пользовательском компьютере. На рис. 12-1 показан подключаемый модуль QuickTime, который позволяет пользователю запускать и останавливать воспроизведение, а также «перематывать» запись и регулировать громкость.

Внедрение звука предоставляет пользователю определенные возможности по управлению воспроизведением, но не очень большие: разрешено скрывать и показывать модуль

проигрывателя, изменять его размеры и расположение, заставлять его проигрывать звук «по кругу» — и только. Далее показан пример HTML-кнопок, скрывающих и вновь отображающих модуль проигрывателя:

```
<input type="button" value="Hide" id="butHide" onclick="wavTest.hidden='true'">
<input type="button" value="Show" id="butShow"
onclick="wavTest.hidden='false'">
```

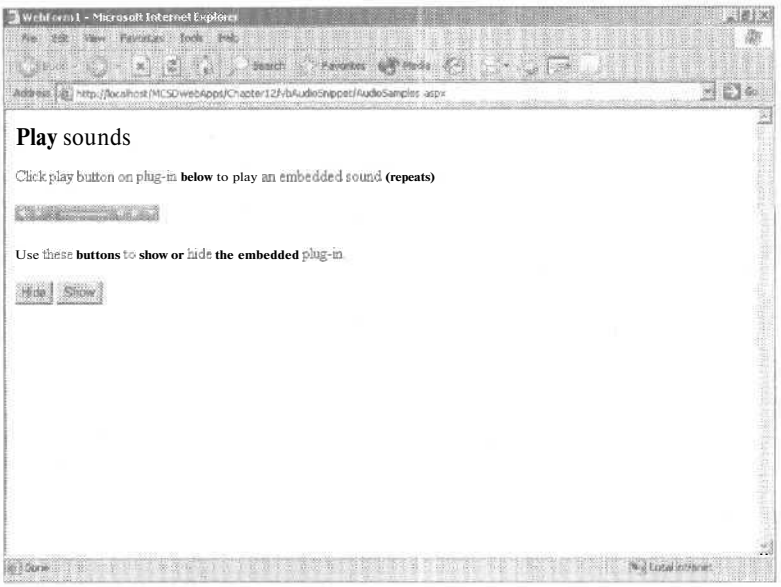


Рис. 12-1. Подключаемый модуль проигрывателя QuickTime

Чтобы предоставить возможность запуска, остановки или приостановки воспроизведения, а также отключения звука из клиентского сценария, необходимо добавить в Web-форму звук в виде объекта, но это — предмет следующего раздела.

Добавление звуков в виде объектов ActiveX

Чтобы включить в Web-форму звук в виде объекта ActiveX, следует добавить к HTML-коду элемент *object* с идентификатором класса (*clsid*) проигрывателя, используемого для воспроизведения звука. Вставка звука в виде объекта позволяет управлять воспроизведением из клиентских сценариев, ссылаясь на этот элемент как на объект.

Чтобы добавить к Web-форме звук в виде объекта, выполните следующие действия.

1. Добавьте к Web-форме ActiveX-объект проигрывателя.
2. Установите свойства этого объекта для воспроизведения звука.
3. Добавьте код, который будет проверять, установлен ли в пользовательской системе нужный объект ActiveX, и, если нет, выполнять соответствующие действия.

Более подробно эти действия описаны в следующем разделе.

Добавление объекта ActiveX к Web-форме

Некоторые объекты ActiveX способны воспроизводить из браузера различное мультимедийное содержимое, в том числе аудиоданные. Наиболее распространены два из них —

объекты Windows Media Player и RealPlayer. Любой из них можно добавить к Web-форме, для чего следует выполнить следующие действия.

1. Откройте Web-форму в режиме Design, перейдите на вкладку HTML на панели Toolbox, щелкните Toolbox правой кнопкой и выберите из контекстного меню команду Customize Toolbox.
2. Перейдите на вкладку COM Components. Прокручивая список установленных компонентов, найдите и пометьте в нем компоненты, которые нужно добавить на Toolbox. Объект RealPlayer называется RealPlayer G2 Control, а объект Media Player — Windows Media Player.
3. Закройте диалоговое окно, щелкнув OK, — Visual Studio загрузит выбранные элементы управления в Toolbox (рис.12-2).

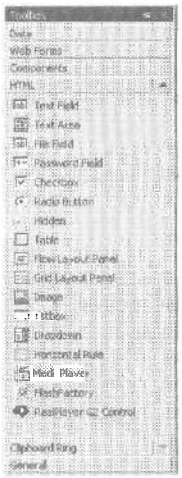


Рис. 12-2. Объекты ActiveX в Toolbox

4. Перетащите объект проигрывателя из Toolbox на Web-форму. В результате Visual Studio добавит к Web-форме HTML-элемент *object*, содержащий идентификатор класса проигрывателя и значения его параметров по умолчанию. Вот пример HTML-элемента *object*

```
<p>Here's the sound as a Windows Media Player object:</p>
<OBJECT id="objWMPlay" classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95"
VIEWASTEXT>
 <PARAM NAME="AudioStream" VALUE="-1">
 <PARAM NAME="AutoSize" VALUE="-1">
 <PARAM NAME="AutoStart" VALUE="0">
 <PARAM NAME="AnimationAtStart" VALUE="-1">
 <PARAM NAME="AllowScan" VALUE="-1">
 <PARAM NAME="AllowChangeDisplaySize" VALUE="-1">
 <PARAM NAME="AutoRewind" VALUE="0">
 <PARAM NAME="Balance" VALUE="0">
 <PARAM NAME="BaseURL" VALUE="">
 <PARAM NAME="BufferingTime" VALUE="5">
 <PARAM NAME="CaptioningID" VALUE="">
 <PARAM NAME="ClickToPlay" VALUE="-1">
 <PARAM NAME="CursorType" VALUE="0">
```

```
<PARAM NAME="CurrentPosition" VALUE="-1">
<PARAM NAME="CurrentMarker" VALUE="0">
<PARAM NAME="DefaultFrame" VALUE="">
<PARAM NAME="DisplayBackColor" VALUE="0">
<PARAM NAME="DisplayForeColor" VALUE="16777215">
<PARAM NAME="DisplayMode" VALUE="0">
<PARAM NAME="DisplaySize" VALUE="0">
<PARAM NAME="Enabled" VALUE="-1">
<PARAM NAME="EnableContextMenu" VALUE="-1">
<PARAM NAME="EnablePositionControls" VALUE="-1">
<PARAM NAME="EnableFullScreenControls" VALUE="0">
<PARAM NAME="EnableTracker" VALUE="-1">
<PARAM NAME="Filename" VALUE="test.wav">
<PARAM NAME="InvokeURLs" VALUE="-1">
<PARAM NAME="Language" VALUE="-1">
<PARAM NAME="Mute" VALUE="0">
<PARAM NAME="PlayCount" VALUE="1">
<PARAM NAME="PreviewMode" VALUE="0">
<PARAM NAME="Rate" VALUE="1">
<PARAM NAME="SAMILang" VALUE="">
<PARAM NAME="SAMIStyle" VALUE="">
<PARAM NAME="SAMIFileName" VALUE="">
<PARAM NAME="SelectionStart" VALUE="-1">
<PARAM NAME="SelectionEnd" VALUE="-1">
<PARAM NAME="SendOpenStateChangeEvents" VALUE="-1">
<PARAM NAME="SendWarningEvents" VALUE="-1">
<PARAM NAME="SendErrorEvents" VALUE="-1">
<PARAM NAME="SendKeyboardEvents" VALUE="0">
<PARAM NAME="SendMouseClickEvents" VALUE="0">
<PARAM NAME="SendMouseMoveEvents" VALUE="0">
<PARAM NAME="SendPlayStateChangeEvents" VALUE="-1">
<PARAM NAME="ShowCaptioning" VALUE="0">
<PARAM NAME="ShowControls" VALUE="-1">
<PARAM NAME="ShowAudioControls" VALUE="-1">
<PARAM NAME="ShowDisplay" VALUE="0">
<PARAM NAME="ShowGotoBar" VALUE="0">
<PARAM NAME="ShowPositionControls" VALUE="-1">
<PARAM NAME="ShowStatusBar" VALUE="0">
<PARAM NAME="ShowTracker" VALUE="-1">
<PARAM NAME="TransparentAtStart" VALUE="0">
<PARAM NAME="VideoBorderWidth" VALUE="0">
<PARAM NAME="VideoBorderColor" VALUE="0">
<PARAM NAME="VideoBorder3D" VALUE="0">
<PARAM NAME="Volume" VALUE="0">
<PARAM NAME="WindowlessVideo" VALUE="0">
</OBJECT>
```

```
<p>Here's the sound as a RealPlayer object:</p>
<OBJECT id="objRPlay" classid="clsid:CFCDAA03-8BE4-11CF-B84B-0020AFBBCCFA"
VIEWASTEXT>
 <PARAM NAME="_ExtentX" VALUE="7938">
 <PARAM NAME="_ExtentY" VALUE="2646">
 <PARAM NAME="AUTOSTART" VALUE="0">
 <PARAM NAME="SHUFFLE" VALUE="0">
 <PARAM NAME="PREFETCH" VALUE="0">
 <PARAM NAME="NOLABELS" VALUE="0">
 <PARAM NAME="LOOP" VALUE="0">
 <PARAM NAME="NUMLOOP" VALUE="0">
 <PARAM NAME="CENTER" VALUE="0">
 <PARAM NAME="MAINTAINASPECT" VALUE="0">
 <PARAM NAME="BACKGROUNDCOLOR" VALUE="#000000">
</OBJECT>
```

Как видно из этого HTML-кода, объекты разных проигрывателей обладают различными наборами управляющих параметров. Для разработчика гораздо проще выбрать определенный проигрыватель и требовать, чтобы пользователь установил именно его, чем писать код, поддерживающий разные проигрыватели. Многие пользователи устанавливают у себя на компьютере несколько проигрывателей, чтобы иметь возможность выбирать нужный без риска возникновения проблем с совместимостью.

Установка свойств и методов объекта проигрывателя

Элементы *param*, включенные в HTML-текст объекта ActiveX, представляют свойства этого объекта. Эти свойства наряду с методами и событиями используются для управления проигрывателем из клиентских сценариев.

Найти достаточно полную справочную информацию об объектах ActiveX сложно. Проще всего получить дополнительные сведения о свойствах, методах и событиях объекта, добавив к текущему проекту ссылку на этот объект и просматривая его члены в окне Object Browser, показанном на рис. 12-3.

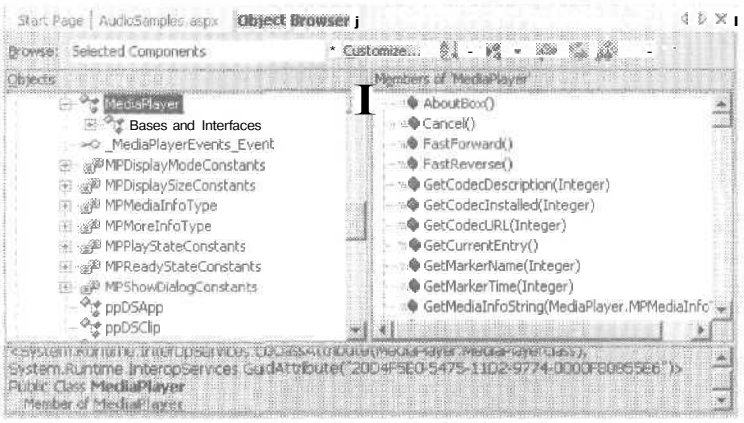


Рис. 12-3. Просмотр свойств, методов и событий объекта Media Player

Для вызова свойств и методов объекта ActiveX из сценариев предназначены HTML-элементы. Вот пример сценариев, которые в ответ на щелчки соответствующих кнопок

запускают и останавливают воспроизведение звука при помощи объекта Windows Media Player:

```
<input type="button" value="Start" id="butStart"
onclick="objWMPlay.play()">

<input type="button" value="Stop" id="butStop"
onclick="objWMPlay.stop()">
```

Имена методов объекта RealPlayer, запускающих и останавливающих воспроизведение, несколько отличаются:

```
<input type="button" value="Start" id="butStartPlay"
onclick="objRPlay.object.source='test.wav';objRPlay.doplay()">

<input type="button" value="Stop" id="butStopPlay"
onclick="objRPlay.costop()">
```

Чтобы получить возможность реагировать на события, генерируемые объектом на клиентском компьютере, необходимо написать обработчик события в виде сценария. Следующий сценарий отображает сообщение по завершении воспроизведения звука через Media Player:

JScript

```
<SCRIPT FOR="objWMPlay" EVENT="EndOfStream(lResult)"
LANGUAGE="Jscript">
 alert("End of clip");
</SCRIPT>
```

VBScript

```
<script language="vbscript">
 Sub objWMPlay_EndOfStream(lResult)
 alert("End of clip.")
 End Sub
</script>
```

Как видно из следующего кода, имена событий объекта RealPlayer также отличаются:

Jscript

```
<SCRIPT FOR="objRPlay" EVENT="OnClipClosed()"
LANGUAGE="Jscript">
 alert("End of clip.");
</SCRIPT>
```

VBScript

```
<script language="vbscript">
 Sub objRPlay_OnClipClosed()
 alert("End of clip.")
 End Sub
</script>
```

Свойства, методы и события объекта ActiveX можно вызывать и из серверного кода, но это приведет к перезагрузке страницы и, следовательно, к перезапуску воспроизведения звука, что обычно нежелательно.

Установка объектов ActiveX на клиентский компьютер

Если на пользовательском компьютере нет нужного объекта ActiveX, при помощи атрибута *codebase* следует задать каталог, из которого пользователь установит отсутствующий объект. Атрибут *codebase* принимает два аргумента: первый задает каталог для загрузки, а второй — номер версии загружаемого элемента управления. Вот пример элемента *object,* который загружает ActiveX-объект Windows Media Player, если тот отсутствует на пользовательском компьютере либо если его версия меньше, чем 6.4:

```
<OBJECT ID="objWMPlayer" classid="CLSID:22D6F312-B0F6-11D0-94AB-0080C74C7E95"
 CODEBASE="http://activex.microsoft.com/activex/controls/mplayer/en/
 nsmp2inf.cab#Version=6,4.5,715" VIEWASTEXT>
 <PARAM NAME="AutoStart" VALUE="True">
 <PARAM NAME="FileName" VALUE="test.avi">
</OBJECT>
```

Когда пользователь вызывает страницу с таким элементом *object,* браузер проверяет наличие заданной версии объекта на машине пользователя. Если нужная версия не найдена, браузер отображает перед загрузкой и установкой объекта диалоговое окно, показанное на рис. 12-4.

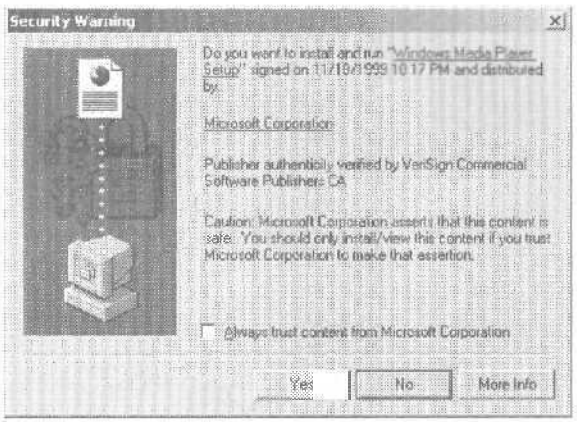


Рис. 12-4, Установка объекта

После установки объекта страница загружается и начинается воспроизведение звука.

Работа с объектами ActiveX в Netscape Navigator

Netscape Navigator не поддерживает ни объекты ActiveX, ни HTML-элемент *object.* Добавлять звуки на страницы, предназначенные для Netscape Navigator, следует как *внедренные модули* (embedded plug-ins).

Netscape Navigator игнорирует элемент *object,* поэтому для получения страниц, поддерживающих и Microsoft Internet Explorer, и Netscape Navigator, элемент *embed* можно вкладывать прямо в элемент *object.* Вот пример HTML, отображающего объект Windows Media Player в любом из этих браузеров:

```
<OBJECT ID="objWMPlayer" classid="CLSID:22D6F312-B0F6-11D0-94AB-0080C74C7E95"
CODEBASE="http://activex.microsoft.com/activex/
controls/mplayer/en/nsmp2inf
.cab#Version=6,4,5,715" type="application/x-oleobject" VIEWASTEXT>
```

```
 <PARAM NAME="AutoStart" VALUE="False">
 <PARAM NAME="FileName" VALUE="test.wav">
 <EMBED type="application/x-mplayer2"
 pluginspage="http://www.microsoft.com/Windows/MediaPlayer/"
 SRC="test.wav"
 name="objWMPlayer"
 autostart="0"
 showcontrols="1"
 </EMBED>
</OBJECT>
```

Если подключаемый модуль Media Player не установлен в Netscape Navigator, предыдущий код направит пользователя на страницу, которая поможет ему загрузить и установить нужный модуль. Чтобы пользователю не пришлось перезапускать свой браузер после установки подключаемого модуля, добавьте после элемента *object* следующий сценарий:

```
<!- Этот сценарий должен быть написан на JavaScript, поскольку Navigator не
поддерживает VBScript ->
<script language="JavaScript">
 // Если боаузер - Netscape Navigator...
 if ( navigator.appName == "Netscape" )
 !
 // ...обновить список установленных подключаемых модулей.
 navigator.plugins.refresh();
 }
</script>
```

Подключаемыми модулями Netscape управляют с помощью сценариев, но для этого нужны другие сценарии, чем для управления объектами ActiveX. Вот некоторые отличия этих сценариев:

- доступ к свойствам осуществляется через методы *Set* и *Get, a* не прямым присваиванием значений;
- события подключаемого модуля перехватываются Java-апплетом, играющим роль прокси.

Указанные трудности при программировании наряду с проблемой стабильности браузера и сложностью отладки разнородного кода, предназначенного для различных платформ, ограничивают целесообразность применения в Netscape Navigator объектов ActiveX, управляемых сценариями.

Если приложение, использующее объекты ActiveX, должно поддерживать Netscape Navigator, удобнее добавлять объекты как подключаемые модули (см. пример выше) и не пытаться использовать сценарии для управления объектами. Ниже показан пример HTML-кода, который проверяет браузер пользователя. Если это Navigator, код деактивирует кнопки Input, управляющие Media Player, и отображает вместо них встроенный интерфейс модуля Media Player.

```
<html>
 <head>
 <title>ActiveX Objects Cross Browser</title>
 </head>
 <body>
 <h2>Using ActiveX Objects Across Different Browsers</h2>
 <OBJECT ID="objWMPlayer"
```

```
classid="CLSID:22D6F312-B0F6-11D0-94AB-0080C74C7E95"
CODEBASE="http://activex.microsoft.com/activex/controls/mplayer/en/ns
mp2inf.cab#Version=6,4,5,715" type="application/x-oleobject"
VIEWASTEXT>
<PARAM NAME="AutoStart" VALUE="False">
<PARAM NAME="FileName" VALUE="test.wav">
<!-Скрыть интерфейс Media Player, если браузер - Internet Explorer -->
<param name="ShowControls" value="False">
<EMBED type="application/x-mplayer2"
pluginspage="http://www.microsoft.com/Windows/MediaPlayer/"
SRC="test.wav"
name="objWMPlayer"
autostart="0"
showcontrols="1"> <!- Если браузер - Navigator, показать интерфейс
                  Media Player ->
</EMBED>
</OBJECT>
<!- Управляющие кнопки Input, формирующие нестандартный интерфейс Media Player
для использования с Internet Explorer ->
<div id="pnlControls">
<!- Этот сценарий должен быть написан на JavaScript, поскольку Navigator не
поддерживает VBScript ->
<scrlpt language="JavaScript">
// Если браузер - Netscape Navigator...
if ( navigator.appName -- "Netscape" )
{
// ...обновить список установленных подключаемых модулей,
navigator.plugins.refresh();
// Закончить страницу здесь (не отображать кнопки Input).
alert("You are using Netscape Navigator. Some ActiveX
       object features have been disabled.");
document.all["pnlControls"].setAttribute("disabled","1");
}
</script>
<br>
<p>Use these buttons to control the Media Player object.</p>
<input type="button" value="Start" id="butStart"
onclick="objWMPlayer.Play();" >
<input type="button" value="Stop" id="butStop"
onclick="objWMPlayer.stop()" >
<input type="button" value="Repeat" id="butRepeat"
onclick="objWMPlayer.PlayCount='2'" >
<input type="button" value="Mute" id="butMute"
onclick="objWMPlayer.Mute=!(objWMPlayer.Mute) " >
</div>
</body>
</html>
```

Занятие 2. Воспроизведение видео

Многие из методов управления воспроизведением видео идентичны или очень похожи на методы управления воспроизведением звука, с которыми вы познакомились на предыдущем занятии. Как и в случае звука, при перезагрузке Web-страницы воспроизведение видео перезапускается, поэтому лучше всего управлять воспроизведением видеоданных и реагировать на события, связанные с видео, при помощи клиентских сценариев, а не серверного кода.

На этом занятии вы узнаете, как добавить к Web-форме или HTML-странице видеоданные с помощью элементов HTML и объектов ActiveX, а также научитесь применять клиентские сценарии, написанные на JScript и VBScript, для управления воспроизведением видео.

Изучив материал этого занятия, вы сможете:

- ✓ описать преимущества и недостатки различных способов воспроизведения видео на Web-страницах;
- ✓ отображать видеоданные на Web-страницах с помощью HTML-элемента *img;*
- ✓ воспроизводить видео с помощью клиентского проигрывателя по умолчанию;
- ✓ использовать ActiveX-объект Windows Media Player для воспроизведения видео;
- ✓ управлять воспроизведением видео с помощью Media Player и клиентских сценариев;
- ✓ применять различные версии Media Player.

Продолжительность занятия — около 20 минут.

Способы воспроизведения видео

Существует три основных способа добавления видеоданных к Web-формам:

- **с помощью HTML-элемента *img* с атрибутом *dynsrc;***
- **внедрение с** помощью **HTML-элемента *embed,*** вставляющего на страницу ссылку на пользовательский проигрыватель по умолчанию;
- **внедрение в виде объекта ActiveX** при помощи HTML-элемента *object*, вставляющего на страницу ActiveX-объект проигрывателя.

У каждого способа есть свои преимущества и недостатки, о них пойдет речь в следующих разделах.

Использование элемента img

Проще всего отображать на Web-формах видеоданные, добавив их с помощью HTML-элемента *img.* У этого элемента имеется атрибут *dynsrc,* определяющий файл с отображаемыми видеоданными. Вот пример HTML-элемента *img с* атрибутом *dynsrc*:

```
<img dynsrc="clock.avi">
```

Серверные элементы управления Image и ImageButton, а также одноименные HTML-элементы управления поддерживают атрибут *dynsrc.* Ниже показан пример HTML-кода, отображающего видео с помощью элемента управления ImageButton:

```
<asp:ImageButton id="ImageButton1" dynsrc="clock.avi"
runat="server">
</asp:ImageButton>
```

Однако применение атрибута *dynsrc* имеет ряд ограничений:

- его не поддерживает Netscape Navigator;
- он принимает только файлы в формате AVI (Audio Video Interleaved);
- не позволяет управлять воспроизведением клиентских сценариев, что возможно при добавлении видеоданных в форме объектов ActiveX.

Внедрение видеоданных в страницы

HTML-элемент *embed* позволяет воспроизводить видео на Web-формах с помощью пользовательского проигрывателя по умолчанию. Например, следующий HTML-код внедряет в странипу файл с видеоданными:

```
<embed src="clock.avi" id="aviTest" hidden="false" height="200" width="200"
autostart="false" type="video/avi" loop="true">
</embed>
```

Внедренные видеоданные отображаются подключаемым модулем, зарегистрированным для воспроизведения этого типа данных. На рис. 12-5 показано, как выглядит в браузере предыдущий HTML-код, воспроизводящий видео через подключаемый модуль Windows Media Player.

Использование HTML-элемента *embed* для отображения видео дает следующие преимущества:

- этот элемент поддерживает большинство браузеров, в том числе Internet Explorer и Netscape Navigator;
- позволяет управлять воспроизведением с помощью интерфейса внедренного проигрывателя;
- позволяет отображать видеоданные в разных форматах.

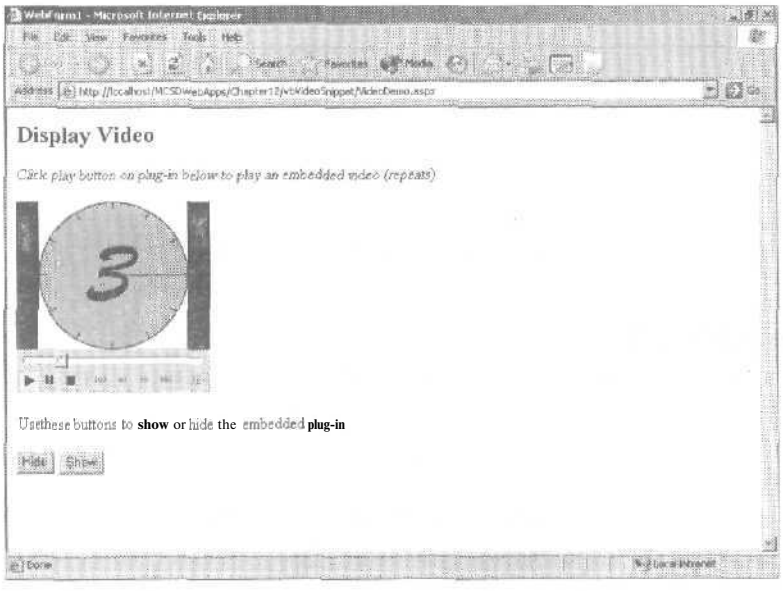


Рис. 12-5. Воспроизведение внедренного видео с помощью подключаемого модуля

Недостаток этого HTML-элемента заключается в ограниченной по сравнению с объектами ActiveX поддержке использования сценариев.

Внедрение видео в форме объектов ActiveX

Как и звук, видеоданные также можно добавлять к Web-формам в виде объектов ActiveX, Для этого следует вставить в HTML-код формы элемент *object* с идентификатором класса (*clsid*) проигрывателя, используемого для воспроизведения внедряемых видеоданных. Воспроизведением видео, внедренного в форме объекта, можно управлять из клиентских сценариев, ссылаясь на HTML-элемент *object,* как на объект.

Процедуры воспроизведения видео, внедренного как объект ActiveX и внедренного аналогичным образом звука одинаковы, только параметр FileName указывает на файл с видеоданными, а не звуковой файл. Вот пример HTML-кода, который с помощью Media Player воспроизводит видео, внедренное как объект ActiveX:

```
<OBJECT id="objWMPlay"
classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95"
CODEBASE="http://activex.microsoft.com/activex/controls/
mplayer/en/ns
mp2inf.cab#Version=6,4,5,715"
type="application/x-oleobject" VIEWASTEXT>
 <PARAM NAME="AutoStart" VALUE="0">
 <PARAM NAME="EnableTracker" VALUE="-1">
 <PARAM NAME="Filename" VALUE="clock.avi">
 <PARAM NAME="Mute" VALUE="0">
 <PARAM NAME="ShowControls" VALUE="-1">
 <PARAM NAME="ShowAudioControls" VALUE="-1">
 <PARAM NAME="ShowDisplay" VALUE="0">
 <PARAM NAME="ShowGotoBar" VALUE="0">
 <PARAM NAME="ShowPositionControls" VALUE="-1">
 <PARAM NAME="ShowStatusBar" VALUE="0">
 <PARAM NAME="ShowTracker" VALUE="-1">
 <PARAM NAME="VideoBorderWidth" VALUE="0">
 <PARAM NAME="VideoBorderColor" VALUE="0">
 <PARAM NAME="VideoBorder3D" VALUE="0">
 <PARAM NAME="Volume" VALUE="0">
 <PARAM NAME="WindowlessVideo" VALUE="0">
</OBJECT>
```

Для управления воспроизведением видео при помощи ActiveX-объекта Windows Media Player используются те же свойства и методы, что и в случае воспроизведения звука; эти свойства и методы описаны в предыдущем занятии. Например, следующие кнопки позволяют запускать, останавливать и приостанавливать воспроизведение видео, а также отключать звук во время воспроизведения:

```
<input onclick="objWMPlay.object.play()" type="button" value="Start">
<input onclick="objWMPlay.stop()" type="button" value="Stop">
<input onclick="objWMPlay.PlayCount='2'" type="button" value="Repeat">
<input onclick="objWMPlay.mute=!objWMPlay.mute" type="button" value="Mute">
<input onclick="objWMPlay.showcontrols='false'" type="button"
 value="Hide Controls">
```

На рис. 12-6 показан вид ActiveX-объекта Windows Media Player во время выполнения.

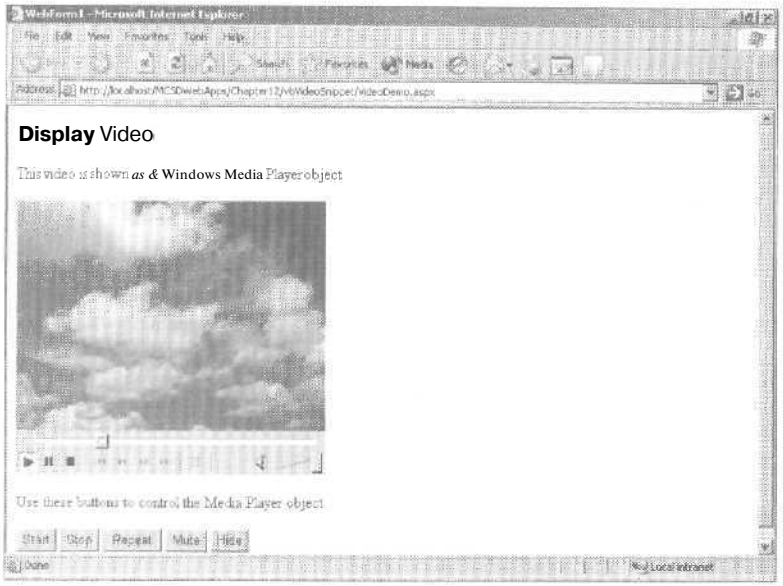


Рис. 12-6. ActiveX-объект Windows Media Player, воспроизводящий видео

Как и при воспроизведении звука, для обработки событий, генерируемых объектом Media Player при воспроизведении видео, применяются клиентские сценарии. Вот пример такого сценария, отображающего сообщение, когда пользователь приостанавливает воспроизведение видео:

Jscript

```
<SCRIPT language="Jscript"event="PlayStateChange(iState) " for="objWMPlay">
 if  (iState==2)
 alert("Video paused.");
</SCRIPT>
```

VBScript

```
<script language="vbscript">
Sub objWMPlay_PlayStateChange(iState)
 if iState=2 then alert("Video paused.")
End Sub
</script>
```

Использование объектов **ActiveX с Netscape** Navigator

Как говорилось в предыдущем занятии, Netscape Navigator не поддерживает ни объекты ActiveX, ни HTML-элемент *object*. Чтобы Netscape Navigator получил возможность воспроизводить видео, необходимо добавлять его в форме внедренных подключаемых модулей. Для этого применяется методика, описанная в предыдущем занятии. Согласно ей, аудиоданные добавляются в форме объектов ActiveX, благодаря чему их удается воспроизводить в Netscape Navigator.

Работа с разными версиями Media Player

Все предыдущие **примеры** использовали для **воспроизведения звука и видео** Media Player **6.4, хотя в настоящее время доступна версия 8.0. Версия 6.4 выбрана в силу по** крайней **мере** двух причин:

- в Netscape Navigator **есть подключаемый** модуль **с эквивалентными возможностями;**
- более поздние версии, начиная с 7.0, нельзя устанавливать в прежних версиях операционных систем (ОС) от Microsoft, таких, как Windows 95 или Windows NT 4.

Если совместимость с прежними версиями ОС не требуется, можно использовать последнюю версию Media Player. Для этого добавьте к Web-форме следующий HTML-элемент *object:*

```
<OBJECT id="Player" height="240" width="320"
classid="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6" VIEWASTEXT>
 <PARAM NAME="URL" VALUE="clouds.avi">
 <PARAM NAME="rate" VALUE="1">
 <PARAM NAME="balance" VALUE="0">
 <PARAM NAME="currentPosition" VALUE="0">
 <PARAM NAME="defaultFrame" VALUE="">
 <PARAM NAME="playCount" VALUE="1">
 <PARAM NAME="autoStart" VALUE="-1">
 <PARAM NAME="currentMarker" VALUE="0">
 <PARAM NAME="invokeURLs" VALUE="-1">
 <PARAM NAME="baseURL" VALUE="">
 <PARAM NAME="volume" VALUE="50">
 <PARAM NAME="mute" VALUE="0">
 <PARAM NAME="uiMode" VALUE="full">
 <PARAM NAME="stretchToFit" VALUE="0">
 <PARAM NAME="windowlessVideo" VALUE="0">
 <PARAM NAME="enabled" VALUE="-1">
 <PARAM NAME="enableContextMenu" VALUE="-1">
 <PARAM NAME="fullScreen" VALUE="0">
 <PARAM NAME="SAMIStyle" VALUE="">
 <PARAM NAME="SAMILang" VALUE="">
 <PARAM NAME="SAMIFilename" VALUE="">
 <PARAM NAME="captioningID" VALUE="">
</OBJECT>
```

Media Player 7 отличается от версии 6.4 в частности идентификатором класса и иной объектной моделью. Например, теперь вместо свойства *Filename* файл с видеоданными задают с помощью свойства *URL.*

У последней версии также нет установочного .cab-файла, на который ссылался атрибут *codebase.* Вместо использования этого атрибута необходимо проверить, установлена ли нужная версия проигрывателя, и, если нет, направить пользователя на страницу, которая поможет установить ее. Ниже показан пример кода, который проверяет наличие Media Player версии 7 или выше и при отсутствии таковой направляет пользователя на сайт Microsoft, с которого можно загрузить нужную версию проигрывателя:

Jscript

```
<script language="JavaScript">
// Используя блок обработки исключений, проверить, можно ли
// создать объект проигрывателя.
try
{
 if(document.objWMPlayer.object.enabled)
 ; // Если объект успешно создан, закончить на этом,
}
catch(ex)
{
 // Если возникла ошибка, предложить пользователю загрузить Media Player
 document.writeln(<p>You don't have the current version of Media
 Player. ");
 document.writeln(
 "<a href='http://www.microsoft.com/windows/windowsmedia/download/'>
 Click here</a> to install it.</p>");
}
</script>
```

VBScript

```
<script language="vbscript">
' Используя блок обработки исключений, проверить, можно ли
' создать объект проигрывателя.
On Error Resume Next
' Если объект успешно создан, закончить на этом,
If Document.objWMPlayer.enabled then x=0
' Если возникла ошибка, предложить пользователю загрузить Media Player,
If Err Then
 document.writeln("<p>You don't have the current version of Media Player. ")
 document.writeln( _
 "<a href='http://www.microsoft.com/windows/windowsmedia/download/'>" & _
 "Click here</a> to install it.</p>")
End If
On Error Goto 0
</script>
```

В справочной системе Visual Studio подробно описана версия 7 Media Player, которая открывается разделом «Windows Media Player 7.1 SDK».

Занятие 3. Анимация на Web-страницах

Анимированные тексты и изображения часто применяются на Web-страницах в рекламе, для акцентирования какого-либо утверждения или просто для развлечения. Спектр методов, при помощи которых создают анимированное содержимое для Web, чрезвычайно широк. На этом занятии вы познакомитесь с основными методами создания анимированного содержимого, сравните их преимущества и недостатки, а также научитесь пользоваться ими.

Изучив материал этого занятия, вы сможете:

- ✓ описать преимущества и недостатки различных способов анимации;
- ✓ анимировать текст с помощью HTML-элемента *marquee*;
- ✓ создавать и применять анимированные GIF-файлы;
- ✓ анимировать элементы Web-страниц при помощи клиентских сценариев;
- ✓ синхронизировать события на Web-странице;
- ✓ синхронно изменять атрибуты элементов HTML, такие, как цвет или положение на странице;
- ✓ синхронизировать воспроизведение звука, видео и других мультимедийных данных.

Продолжительность занятия — около 35 минут.

Способы отображения анимированного содержимого

Способов анимации элементов Web-форм множество. Преимущества и недостатки наиболее распространенных из них сравниваются в таблице 12-1.

Таблица 12-1. Методики анимации содержимого

Методика	Преимущества	Недостатки
Отображение анимированного текста и графических объектов с помощью HTML-элемента *marquee*	• Используются простые HTML-теги и атрибуты • Не требуются сценарии	• Доступно только в Internet Explorer • Позволяет создавать ограниченный набор эффектов
Внедрение в страницу объектов ActiveX или подключаемых модулей	• Позволяет создавать сложные анимированные презентации • Позволяют начинать отображение анимации, не дожидаясь окончания загрузки (поддерживают потоковые данные) • Имеются широко доступные инструменты от сторонних разработчиков, позволяющие создавать и отображать анимированное содержимое такого рода • Поддерживают интерактивность • Объектами ActiveX и некоторыми подключаемыми модулями можно управлять из клиентских сценариев	• Требует установки объектов ActiveX и подключаемых модулей. Для использования инструментов в некоторых случаях требуется лицензия, сложные инструменты приходится долго осваивать

Таблица 12-1. (*окончание*)

Методика	Преимущества	Недостатки
Использование анимированных GIF-файлов	• Можно использовать с любым HTML-элементом, поддерживающим графику • Не требует написания кода	• С возрастанием сложности анимации возрастает размер файлов • Перед отображением необходимо загрузить файл целиком • Не позволяют управлять воспроизведением
Управление HTML-элементами страницы с помощью клиентских сценариев	• Позволяет управлять любым элементом страницы • Сценарии, написанные на JScript, поддерживаются большинством браузеров	• Число эффектов, которые можно генерировать только программными средствами, ограничено • Код может стать настолько сложным, что его будет трудно сопровождать • Трудно и технически сложно синхронизировать эффекты
Управление элементами страницы с помощью HTML+TIME	• Использует стандартный язык на основе XML • Позволяет управлять любым элементом на Web-странице посредством синхронизированной анимации • Может совмещаться с клиентскими сценариями для создания интерактивных эффектов	• Не поддерживается в Netscape Navigator • Нельзя просматривать эффекты в режиме Design, их разрешается редактировать только в виде HTML

Более подробно эти методики описаны в следующих разделах.

Анимация текста

HTML-элемент *marquee* анимирует любые вложенные в него элементы, перемешая их вверх, вниз, вправо или влево. Этот элемент часто применяют для отображения броских баннеров. Следующий пример HTML-кода отображает контрастный текст «Notice Me», перемещающийся справа налево.

```
<marquee bgcolor="red" scrollamount="10" behavior="alternate"
direction="right" loop="true">
 <h1 style="COLOR:Yellow">Notice Me</h1>
</marquee>
```

Кроме того, с помощью элемента *marquee* удается отображать текст, «уплывающий» за пределы окна просмотра, подобно титрам в кинофильмах. Вот пример HTML-кода, прокручивающего список участников проекта. Строки списка появляются внизу, движутся вверх по экрану и исчезают за верхним краем элемента *marquee*.

```
<marquee scrollamount="1" behavior="scroll" direction="up" loop="false">
<h2>Credits</h2>
<p>Thanks to the following participants:</p>
<p>Denise Smith - trusted friend and advisor.</p>
<p>Alan Steiner - original party animal.</p>
```

```
<p>Will Shakespeare - lead grammarian.</p>
<p>Gary Yukish - thanks for the spell check. </p>
<p>Karin Zimprich - who invented the four-point seat belt.</p>
</marquee>
```

Вложенные элементы *marquee* позволяют получать различные эффекты. Вот пример HTML-текста, который отображает шарик, прыгающий по рамке:

```
<marquee scrollamount="10" behavior="alternate" direction="up" loop="true">
 <marquee scrollamount="5" behavior="alternate" direction="right"
 loop="true">
 <img src="circles.gif">
 </marquee>
</marquee>
```

У элемента *marquee* есть событие *onbounce*, позволяющее создавать дополнительные эффекты. Ниже показаны вложенные элементы *marquee,* отображающие шарик, который катится, постепенно замедляя движение, пока не останавливается:

```
<marquee id="mrqVert" scrollamount="10" behavior="alternate" direction="down"
loop="true" height="400" onbounce="OnBounce()">
 <marquee id="mrqHoriz" scrollamount="5" behavior="alternate"
 direction="right" loop="true">
 <img src="circles.gif">
 </marquee>
</marquee>
```

VBScript

```
<script language=vbscript>
bOdd=0
Sub OnBounce()
 ' Получить высоту документа.
 x = document.all("mrqVert").height
 ' Переключить флаг.
 bOdd = Not bOdd
 if x > 100 then
 ' Если это четный прыжок (от нижней границы окна)...
 if bOdd then
 ' ...уменьшить высоту отскока при каждом прыжке.
 document.all("mrqVert").height = x - 20
 end if
 else
 ' Остановить шарик.
 document.all("mrqVert").scrollamount=0
 document.all("mrqHoriz").scrollamount=0
 end if
End Sub
</script>
```

J script

```
<script language=JavaScript>
var bOdd=0;
function OnBounce()
{
 // Получить высоту документа.
 x = document.all["mrqVert"].height;
 // Переключить флаг.
 bOdd = !bOdd;
 if (x > 100)
 // Если это четный прыжок (от нижней границы окна)...
 if (bOdd)
 // ...уменьшить высоту отскока при каждом прыжке,
 document.all["mrqVert"].height = x - 20;
 else
 {
 // Остановить шарик.
 document.all["mrqVert"].scrollamount=0;
 document.all["mrqHoriz"].scrollamount=0;
 }
}
</script>
```

Анимация простых графических элементов

Простейшей формой анимированного Web-содержимого считаются анимированные файлы формата GIF (Graphic Interchange Format). GIF-файлы могут хранить несколько изображений. Используя эту особенность, с помощью инструментов, таких, как Microsoft PhotoDraw, удается собрать анимированное изображение из нескольких картинок (рис. 12-7).

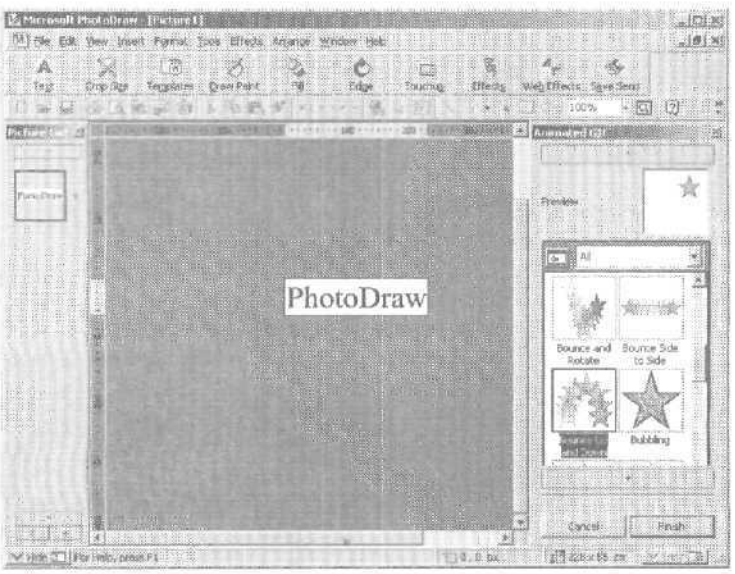


Рис. 12-7. Анимированный GIF-файл в PhotoDraw

PhotoDraw или другой инструмент позволяют добавлять к анимированному ряду кадры, задавая для каждого кадра время отображения и другие атрибуты. Готовый GIF-файл допустимо использовать с любым HTML-элементом, принимающим графику, в том числе с серверным элементом управления Image button.

Применение анимированных GIF-файлов можно сочетать с другими методиками анимации, например с элементом *marquee*. Следующий **HTML** отображает в окне браузера летающий самолетик:

```
<marquee behavior="scroll" direction="down" height="800" width="800">
 <marquee behavior="alternate" direction="right">
 <img src="airplane.gif">
 </marquee>
</marquee>
```

Анимация с помощью объектов и подключаемых модулей

Существует ряд объектов ActiveX и подключаемых модулей браузера, позволяющих создавать анимированные презентации для Web. Самые распространенные из них — Macromedia ShockWave и Windows Media Player.

Принцип отображения анимированного содержимого у объектов ActiveX и подключаемых модулей отличается, на каждого из них есть документация, где обсуждается их использование для создания презентаций. Более подробные сведения о добавлении объектов ActiveX и подключаемых модулей к Web-страницам см, в занятии 1.

Анимация с помощью клиентских сценариев

Клиентские сценарии способны управлять любыми элементами Web-форм, поэтому их тоже применяют для анимации различных объектов путем их перемешения или сокрытия с последующим отображением. Ниже показаны HTML-код и клиентский сценарий, которые создают изображение, перемещаемое в окне браузера вслед за указателем мыши.

```
<html>
<body MS_POSITIONING="GridLayout" onmousemove="OnMove()">
 <img id="imgPlane" src="circles.gif" height="20" width="20"
 style="Z-INDEX: 101; LEFT: 1px; POSITION: absolute; TOP: 1px">
```

VBScript

```
<script language="vbscript">
' Обьявить глобальные переменные.
iCount = 0
x = 0
y = 0
xImg - 0
yImg - 0
Sub OnMove()
 if iCount > 0 then exit sub
  ' Получить координаты X и Y указателя мыши.
 xMouse = window.event.clientx
 yMouse = window.event.clienty
  ' Получить координаты X и Y изображения.
```

```
 xlmg = document.all("imgPlane").style.getAttribute("LEFT", 0)
 yImg = document.all("imgPlane").style.getAttribute("TOP",0)
  ' Преобразовать координаты из строк в числа.
 xImg = cint(left(xImg, len(xImg) -2))
 yImg = cint(left(yImg, len(yImg) -2))
  ' Сократить расстояние между точками, определяемыми двумя наборами координат.
 x - (xMouse - xImg) / 4
 y - (yMouse - yImg) / 4
  ' Исполнять этот код через каждые 100 миллисекунд.
 window.setInterval "MoveImg", 100, "VBScript"
 window.setInterval "Movelmg", 200, "VBScript"
 window.setInterval "Movelmg", 300, "VBScript"
 window.setInterval "Movelmg", 400, "VBScript"
 window.setInterval "MoveImg", 500, "VBScript"
End Sub

Sub MoveImg()
 if iCount < 5 then iCount = iCount + 1 else iCount - 0
  ' Переместить элемент IMG вслед за указателем мыши.
 document.all("imgPlane").style.setAttribute "LEFT", xImg + x, 0
 document.all("imgPlane").style.setAttribute "TOP", yImg + y, O

End Sub
</script>
```

Jscript

```
<script language="JScript">
// Объявить глобальные переменные.
var iCount = 0;
var x = 0;
var y = 0;
var xImg = 0;
var yImg - O;

function OnMove()
{
 if (iCount > 0 )
 return;
 // Получить координаты X и Y указателя мыши,
 xMouse = parseInt(window.event.clientX);
 yMouse = parseInt(window.event.clientY);
 // Получить координаты X и Y изображения.
 xImg = document.all["imgPlane"].style.getAttribute("LEFT",0);
 yImg - document.all["imgPlane"].style.getAttribute("TOP",0);
 // Преобразовать координаты из строк в числа.
 xImg = parseInt(xImg);
 ylrng = parseInt(yImg);
```

```
 // Сократить расстояние между точками, определяемыми двумя наборами координат,
 x = (xMouse - xImg) / 4;
 y = (yMouse - yImg) / 4;
 // Исполнять этот код через каждые 100 миллисекунд.
 window.setInterval ("MoveImg()", 100, "JavaScript") ;
 window.setInterval ("MovelmgO", 200, "JavaScript") ;
 window.setInterval ("MoveImg()", 300, "JavaScript") ;
 window.setInterval ("MoveImg()", 400, "JavaScript") ;
 window.setInterval ("MoveImg()", 500, "JavaScript") ;
}

function MovelmgO
{
 if (iCount < 4)
 iCount ++;
 else
 iCount = 0;
 // Переместить элемент IMG вслед за указателем мыши.
 document.all["imgPlane"].style.setAttribute ("LEFT", xImg + x, 0);
 document.all["imgPlane"].style.setAttribute ("TOP", yImg + y, 0);
}
</script>
```

Показаный код одновременно демонстрирует возможности и сложности анимации HTML-элементов с помощью клиентских сценариев: с любым элементом разрешается делать что угодно, но синхронизировать полученные эффекты очень трудно.

В предыдушем примере анимация координируется методом *setInterval*, передвигающим изображение на четверть дистанции каждые 10 миллисекунд. Прием интересный, но изображение двигается заметными рывками. Чтобы решить эту проблему, необходимо найти способ синхронизация событий исключительно средствами HTML. В следующем разделе разъясняется, как это сделать с помощью расширения стандарта **HTML.**

Анимация с помощью HTML+TIME

До сих пор мы, рассказывая о звуке, видео и анимации, не затрагивали такой важный элемент, как синхронизация. В исходной версии стандарта HTML возможность синхронизации событий на странице не предусмотрена, однако она имеет ключевое значение для создания и воспроизведения синхронизируемого мультимедийного содержимого, такого, как звук, видео и анимация.

Чтобы решить эту проблему, организация World Wide Web Consortium (W3C) разработала стандарт Synchronized Multimedia Integration Language (SMIL, произносится как «смайл»), реализованный в Internet Explorer под названием HTML+TIME.

HTML+TIME позволяет синхронизировать события на странице при создании мультимедийных презентаций. Синхронизация осуществляется с помощью *временной шкалы* (timeline), которая может быть последовательной, параллельной или монопольной. Для использования HTML+TIME требуется Internet Explorer 5.5 или более поздней версии. Применение HTML+TIME с Web-формами и HTML-страницами описано в следующих разделах.

Использование HTML+TIME

Чтобы использовать HTML+TIME на Web-форме или HTML-странице, выполните следующие действия.

1. Добавьте к HTML-элементу страницы атрибут пространства имен XML (XMLNS). Вот пример HTML, объявляющего пространство имен HTML+TIME на странице:

   ```
   <HTML XMLNS:t ="urn:schemas-microsoft-com:time">
   ```

2. Добавьте стиль, определяющий динамический атрибут *time* для HTML-элементов страницы. Следующее определение стиля устанавливает для класса *time* атрибут по умолчанию (он попеременно показывает и скрывает управляемый им элемент):

   ```
   <style>
     .time {behavior: url(#default#time2);}
   </style>
   ```

3. Добавьте директиву IMPORT, импортирующую в страницу пространство имен XML, объявленное выше. Вот пример HTML для импорта пространства имен HTML+TIME:

   ```
   <?IMPORT namespace="t" implementation="#default#time2">
   ```

4. Создайте временную шкалу для элементов, которые предполагаете синхронизировать с помощью HTML+TIME. Следующий HTML устанавливает последовательную временную шкалу для элементов, расположенных ниже его на странице:

   ```
   <t:SEQ>
   ```

5. Установите синхронизирующие атрибуты для HTML-элементов, управляемых с помощью HTML+TIME. Этот пример HTML отображает текстовую строку в течение 2 секунд:

   ```
   <DIV CLASS="time" DUR="2">These words are presented</DIV>
   ```

Ниже показан HTML-код, демонстрирующий все описанные выше действия на примере отображения строки текста, меняющегося каждые две секунды. Полужирным **шрифтом** выделены элементы, которые нужно добавить к обычной Web-форме или HTML-странице для использования HTML+TIME.

```
<html XMLNS:t ="urn:schemas-microsoft-com:time">
 <head>
 <title>HTMLTIME</title>
 <meta name="GENERATOR" content="Microsoft Visual Studio. MET 7.0">
 <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
 <meta name="vs_defaultClientScript" content="JavaScript">
 <meta name="vs_targetSchema"
 content="http://schemas.microsoft.com/intellisense/ie5">
 <style>
 .time {behavior: url(#default#time2);}
 </style>
 </head>
 <body MS_POSITIONING="GridLayout">
 <form id="Form1" method="post" runat="server">
 <?IMPORT namespace="t" implementation="#default#time2">
 <t:SEQ>
 <DIV CLASS="time" DUR="2">These words are presented</DIV>
 <DIV CLASS="time" DUR="2">sequentially, one line at a time
</DIV>
```

```
 <DIV CLASS="time" DUR="2">to make their point more clearly,
</DIV>
 <DIV CLASS="time">Can you remember what I just said?</DIV>
 </t:SEQ>
 </form>
 </body>
</html>
```

Дизайнер Visual Studio не поддерживает директиву *IMPORT,* поэтому необходимо запустить приложение, чтобы увидеть элементы HTML+TIME в Visual Studio. Чтобы получить возможность создания и редактирования элементов страницы в окне Design, временно закомментируйте директиву *IMPORT* перед переключением в режим Design;

```
<!- <?IMPORT namespace="t" implementation="#default#time2"> ->
```

В этом случае элементы управления HTML+TIME не отображаются, как и синхронизирующие атрибуты обычных HTML-элементов, зато можно добавлять и изменять другие элементы страницы.

Создание временных шкал

В предыдущем разделе с помощью элемента HTML+TIME *t:SEQ* вы создали последовательную временную шкалу. Как видно из названия, в последовательной временной шкале события происходят поочередно, длительность события определяется значением атрибута *DVR.* К примеру, следующий фрагмент HTML поочередно показывает строки, отображая каждую в течение 2 секунд:

```
<t:SEQ>
 <DIV CLASS="time" DUR="2">These words are presented</DIV>
 <DIV CLASS="time" DUR="2">sequentially, one line at a time
</DIV>
 <DIV CLASS="time" DUR="2">to make their point more clearly,
</DIV>
 <DIV CLASS="time" DUR="2">Can you remember what I just said?
</DIV>
</t:SEQ>
```

Помимо последовательных шкал, элементы управления HTML+TIME позволяют создавать монопольные (*t:EXCL*) и параллельные (*t:PAR*) временные шкалы. Монопольные временные шкалы отображают не более одного элемента одновременно и делают это в моменты, заданные атрибутом *BEGIN.* Если отображение следующего элемента начнется до окончания отображения предыдущего, то следующий элемент заместит предыдущий. Ниже показан пример HTML, который отображает несколько строк, каждую в течение двух секунд. При этом атрибут *DUR* игнорируется во всех случаях, кроме последней строки, которая действительно отображается в течение трех секунд:

```
<t:EXCL>
 <DIV CLASS="time" BEGIN="0" DUR="3">These words are presented
</DIV>
 <DIV CLASS="time" BEGIN="2" DUR="3">using an exclusive timeline,
</DIV>
 <DIV CLASS="time" BEGIN="4" DUR="3">Each line has a start time.</DIV>
 <DIV CLASS="time" BEGIN="6" DUR="3">Can you remember what I said?</DIV>
</t:EXCL>
```

Параллельная временная шкала позволяет отображать несколько элементов одновременно. Начало и продолжительность отображения каждого элемента определяется соответственно атрибутами *BEGIN* и *DUR*. Например, следующий фрагмент HTML отображает каждую строку в течение трех секунд, причем в отдельные моменты видны сразу две строки:

```
<t:PAR>
 <DIV CLASS="time" BEGIN="0" DUR="3">These words are presented
</DIV>
 <DIV CLASS="time" BEGIN="2" DUR="3">using a parallel timeline.
</DIV>
 <DIV CLASS="time" BEGIN="4" DUR="3">More than one is displayed
</DIV>
 <DIV CLASS="time" BEGIN="6" DUR="3">at a time.</DIV>
</t:PAR>
```

Последовательная временная шкала отличается также тем, что отображаемые с ее помощью элементы заменяют друг друга. Первый из двух предыдущих примеров отображает все строки в одной и той же области страницы, в то время как примеры, использующие монопольную и параллельную временные шкалы, показывают каждую строку на новом месте.

Временные шкалы могут быть вложенными, при этом *главная временная шкала* (master timeline) управляет подчиненными. Например, следующий HTML показывает текст с помощью трех разных временных шкал:

```
<h1>Timeline Demos</h1>
<t:SEQ>
 <h2 CLASS="time" DUR="2">Sequential Timeline</h2>
 <t:SEQ>
 <DIV CLASS="time" DUR="2">These words are presented</DIV>
 <DIV CLASS="time" DUR="2">sequentially, one line at a time</DIV>
 <DIV CLASS="time" DUR="2">to make their point more clearly.
</DIV>
 <DIV CLASS="time" DUR="2">Can you remember what I just said?</DIV>
 </t:SEQ>
 <h2 CLASS="time" DUR="2">Exclusive Timeline</h2>
 <t:EXCL>
 <DIV CLASS="time" BEGIN="0" DUR="3">These words are presented</DIV>
 <DIV CLASS="time" BEGIN="2" DUR="3">using an exclusive timeline.
</DIV>
 <DIV CLASS="time" BEGIN="4" DUR="3">Each line has a start time.</DIV>
 <DIV CLASS="time" BEGIN="6" DUR="3">Can you remember what I just said?</DIV>
 </t:EXCL>
 <h2 CLASS="time" DUR="2">Parallel Timeline</h2>
 <t:PAR>
 <DIV CLASS="time" BEGIN="0" DUR="3">These words are presented
</DIV>
 <DIV CLASS="time" BEGIN="2" DUR="3">using a parallel timeline.</DIV>
 <DIV CLASS="time" BEGIN="4" DUR="3">More than one is displayed</DIV>
 <DIV CLASS="time" BEGIN="6" DUR="3">at a time.</DIV>
```

```
 </t:PAR>
 <h2 CLASS="time" DUR="4">That's it!</h2>
</t:SEQ>
```

Использование атрибута *TIMECONTAINER*

Показанные ранее элементы *t:SEQ*, *t:EXCL* и *t:PAR* определяют временную шкалу для элементов, расположенных между их открывающими и замыкающими тегами. Кроме того, с помошью атрибута *TIMECONTAINER* можно создавать собственные временные шкалы для отдельных HTML-элементов. Вот HTML, эквивалентный примеру с последовательной временной шкалой, показанному в предыдущем разделе:

```
<DIV class="time" timeContainer="seq">
 <DIV CLASS="time" DUR="2">These words are presented</DIV>
 <DIV CLASS="time" DUR="2">sequentially, one line at a time
</DIV>
 <DIV CLASS="time" DUR="2">to make their point more clearly.
</DIV>
 <DIV CLASS="time" DUR="2">Can you remember what I just said?
</DIV>
</DIV>
```

Атрибут *TIMECONTAINER* немного облегчает создание временных шкал, позволяя просто добавлять их в существующие HTML-контейнеры, такие, как элементы *div*, *form* или *body*. Они также облегчают создание отдельных временных шкал для отдельных элементов страницы. Вот пример HTML, использующего отдельную временную шкалу отображения последней строки при помощи элемента *marquee*:

```
<marquee CLASS="time" TIMECONTAINER="seq" REPEATCOUNT="indefinite"
behavior=alternate>
 <h2 CLASS="time" DUR="2">These words are presented</h2>
 <h2 CLASS="time" DUR="2">sequentially, one line at a time</h2>
 <h2 CLASS="time" DUR="2">to make their point more clearly.</h2>
 <h2 TIMECONTAINER="seq" CLASS="time" DUR="2">
 <div class="time" dur="1">Can you remember what I just said?
</div>
 <div class="time" dur="1">Cause I've already forgotten.</div>
 </h2>
</marquee>
```

Анимация с помощью атрибутов

По умолчанию временная шкала просто показывает и скрывает элементы, однако иногда требуются и другие эффекты. Для этого предназначен элемент *t:ANIMATE* изменяющий атрибуты элемента управления в заданный момент времени. Например, следующий HTML переключает атрибут *span* текста с «end» на «beginning» каждые 250 миллисекунд:

```
<t:ANIMATE targetelement="sEnd" attributename="innerHTML" from="end"
to="beginning" dur=".25" repeatcount="indefinite" />
<h2>This is the <span id="sEnd">end</span>.</h2>
```

С помощью элемента *t:ANIMATECOLOR* можно создавать переходы цветов. Вот пример HTML, изменяющего цвет заголовка с желтого на красный за пять секунд:

```
<t:ANIMATECOLOR targetElement="hEnd" attributeName="color" from="yellow"
to="red" start="2" dur="5" fill="hold">
<h2 id="hEnd" style="color:black">This is the <span id="sEnd">end</span>.
</h2>
```

Элемент *t:ANIMATEMOTION* позволяет перемещать элементы по странице. Так, этот HTML перемещает заголовок из точки сетки с координатами (400, 400) в точку с координатами (1, 1):

```
<t:ANIMATEMOTION targetElement="hEnd" from="400,400" to="1,1" start="2" dur="5"
fill="hold">
<h2 id="hEnd" style="position:absolute;color:black">This is the
<span id="sEnd">end</span>.</h2>
```

t:ANIMATEMOTION поддерживает и относительные перемещения элементов, когда расстояния задаются относительно исходного положения элемента. Для этого вместо атрибута *TO* следует использовать атрибут *BY*. Другой вариант — определить путь перемещения элемента средствами атрибута *VALUES* или *PATH*. Атрибут *VALUES* принимает строку из координатных пар. Например, следующий HTM L перемешает заголовок по контуру квадрата размером 200 на 200:

```
<t:ANIMATEMOTION targetElement="hEnd" values="1,200;200,200;200,1;1,1;1,200"
start="2" dur="5" fill="hold">
<h2 id="hEnd" style="position:absolute;color:black">This is the <span
id="sEnd">end</span>.</h2>
```

Атрибут PATH принимает в качестве аргумента строку с командами Scalable Vector Graphics (SVG), которые задают перемещение элемента в абсолютных и относительных координатах. Следующий HTML перемещает заголовок по контуру треугольника:

```
<t:ANIMATEMOTION targetElement="hEnd" path="M 0 200 h 200 v -200 z" start="2"
dur="10" repeatcount="indefinite">
<h2 id="hEnd"style="position:absolute;color:black">This is the <span
id="sEnd">end</span>.</h2>
```

Управление различными видами мультимедиа с помощью HTML+TIME

HTML+TIME также поддерживает элементы для управления воспроизведением звука, видео, отображения графики и других *синхронизированных мультимедийных данных* (generic timed media). Однако пока эти элементы, в сущности, не более чем эквиваленты *t: ANIMATION*. Можно применять тэги, специфичные для определенного вида мультимедиа, если вы считаете, что они делают код понятнее, или надеетесь получить выгоду после выхода новых версий HTML+TIME.

Элементы *t:A UDIO*, *t:VIDEO*, *t:MEDIA*, *t:IMG* и *t:REF* используются так же, как *t:ANIMATION*. Например, показанный ниже HTML подает звуковой сигнал каждые две секунды:

```
<t:AUDIO src="beep.wav" hasaudio="True" dur="2" repeatcount="indefinite" />
```

Аналогичным образом следующий HTML поочередно показывает две картинки, меняя их каждые 5 секунд:

```
<t:par REPEATCOUNT="indefinite" >
 <t:IMG src="gondola.jpg" style="position:absolute; " begin="0" end="5" >
 <t:IMG SRC="bridge.jpg" style="postion:absolute; " begin="5" end="10" >
</t:par>
```

В предыдущем примере использование элемента *t:IMG* или его эквивалента *t:ANIMATE* не обязательно, поскольку тег *<img class=»time»>* делает то же самое. Однако элементы HTML+TIME позволяют добавлять переходы, недоступные при применении элемента *img*. Следующий HTML меняет картинки с использованием переходов типа «затемнение» (fade) и «вытеснение шторкой» (wipe):

```
<t:seq REPEATCOUNT="indefinite" >
 <t:IMG src="gondola.jpg" style="position:absolute; " DUR="5"
 FILL="transition" timecontainer="par" >
 <t:TRANSITIONFILTER TYPE="fade" mode="in" begin="0" end="1"/>
 <t:TRANSITIONFILTER TYPE="fade" mode="out" begin="4" end="5" />
 </t:IMG>
 <t:IMG SRC="bridge.jpg" style="postion:absolute; " DUR="5" FILL="transition"
 timecontainer="par" >
 <t;TRANSITIONFILTER TYPE="clockwipe" mode="in" begin="0" end="1" />
 <t:TRANSITIONFILTER TYPE="clockwipe" mode="out" begin="4" end="5" />
 </t:IMG>
</t:SEQ>
```

Примечание Элементы HTML+TIME позволяют определять скорость воспроизведения анимации, переходы между кадрами и множество других параметров анимации, не вошедших в это занятие. Подробнее об этом рассказано в разделе «HTML+TIME» в справочной системе Visual **Studio** .NET.

Резюме

- Для управления воспроизведением мультимедиа применяются клиентские события и сценарии, поскольку при использовании серверного кода неизбежна перезагрузка страницы и, как следствие, перезапуск воспроизведения.
- Для воспроизведения фоновых звуков используется HTML-элемент *bgsound*.
- Атрибут *dynsrc* HTML-элемента *img* позволяет отображать видео, но не дает возможности управлять воспроизведением,
- HTML-элемент *embed* воспроизводит звук и отображает видео с помощью пользовательского проигрывателя по умолчанию.
- HTML-элемент *object* позволяет добавлять звук и видео в форме объектов ActiveX, при этом воспроизведением можно управлять из клиентского сценария.
- Чтобы получить верный идентификатор класса и значения атрибутов по умолчанию для объекта ActiveX, добавьте его на панель Toolbox в Visual Studio .NET и перетащите с Toolbox на поверхность Web-формы.
- Атрибут *codebase* HTML-элемента *object* позволяет автоматически проверить, установлена ли на клиентском компьютере требуемая версия объекта ActiveX, и, если нет, установить ее.
- Простейшими методами анимации считаются применение анимированных GIF-файлов и HTML-элемента *marquee*.
- Клиентские сценарии позволяют анимировать любые элементы Web-формы и реагировать на события, инициируемые пользователем, но при этом возникают проблемы с синхронизацией событий.
- Для синхронизации событий мультимедиа в Internet Explorer служат элементы HTML+TIME, которые представляют собой вариант стандарта SMIL, реализованный Microsoft.
- Элементы HTML+TIME позволяют:
  - создавать на Web-странице различные типы временных шкал;
  - задавать порядок, время начала и длительность событий в данной временной шкале;
  - управлять атрибутами, цветом и положением элементов при помощи синхронизируемых событий временной шкалы.

Практикум 12. Создание мультимедийного приложения

В этом практикуме вы создадите Web-приложение с двумя Web-формами (или HTML-страницами). Первая страница демонстрирует использование звука и подвижных элементов, а вторая — использование элементов HTML+TIME на примере текстового слайд-шоу.

Для этого практикума мы специально подобрали простые примеры, показывающие недюжинные возможности методик работы с мультимедиа при их правильном выборе и применении. Выполнив предложенные задания, не останавливайтесь на достигнутом, попробуйте самостоятельно дополнить созданное приложение привлекательными эффектами.

Продолжительность практикума — около 15 минут.

Упражнение 1. Использование звука и движущихся объектов

Сейчас вы создадите Web-форму или HTML-страницу, отображающую анимированную картинку в фомате GIF (надпись прыгает в окне браузера и издает звук при каждом прыжке). Примерный вид готовой страницы показан на рис.12-8.

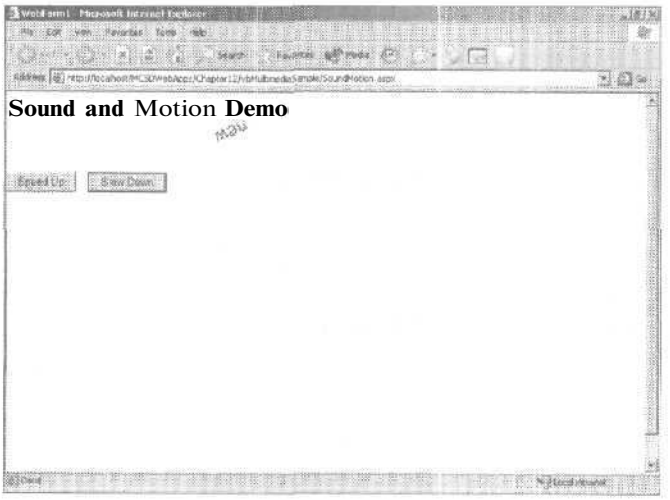


Рис. 12-8. **Страница, демонстрирующая использование звука и движущихся объектов**

Используя навыки, полученные в трех занятиях из этой главы, создайте Web-форму или HTML-страницу, способную:

1. отображать анимированный .gif-файл с помощью HTML-элемента *img*;
2. перемещать отображаемый .gif-файл по экрану с использованием вложенных HTML-элементов *marquee*;
3. предоставлять кнопки для управления скоростью движения изображения;
4. использовать ActiveX-объект Windows Media Player для воспроизведения звука в ответ на событие *bounce*;
5. проверять браузер пользователя и, если это не Internet Explorer, уведомлять пользователя о недоступности некоторых возможностей страницы.

Закончив, сравните свои результаты с образцом, который находится на прилагаемом компакт-диске.

Упражнение 2. Создание слайд-шоу

Сейчас вы создадите Web-форму или HTML-страницу, использующую элемент HTML+TIME для демонстрации анимированных слайдов. Готовая страница показана на рис. 12-9.

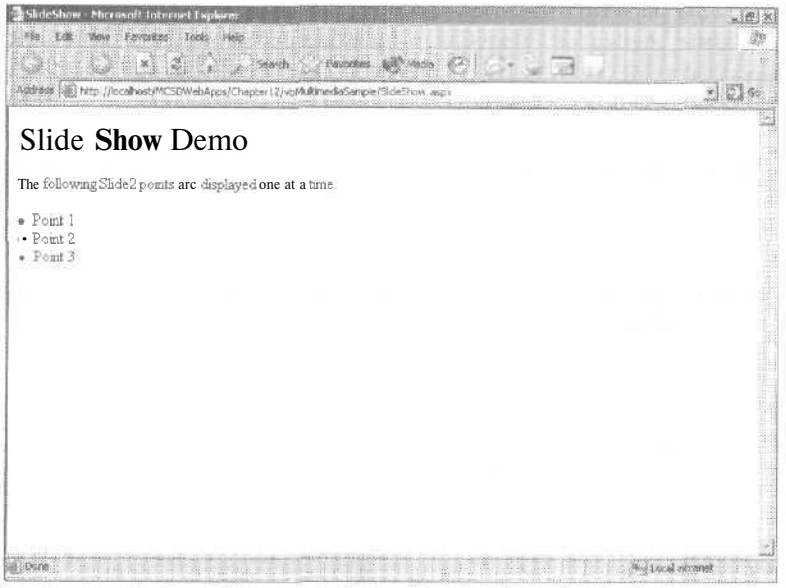


Рис.12-9. Демонстрация слайдов

Используя навыки, полученные в трех занятиях из этой главы, и следующие указания, создайте вторую Web-форму или HTML-страницу приложения.

1. Добавьте к странице пространство имен, стиль и директиву IMPORT, необходимые для применения элементов HTML+TIME.
2. Определите временную шкалу, последовательно отображающую три текстовых слайда.
3. В первых двух слайдах определите параллельные временные шкалы, отображающие три элемента списка.
4. В последнем слайде определите последовательную временную шкалу, демонстрирующую отсчет от 1 до 5.
5. Добавьте переход, направляющий пользователя на другую страницу после завершения последней временной шкалы.

Закончив, сравните свои результаты с образцом, который находится на прилагаемом компакт-диске.

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. Почему для управления мультимедиа, как правило, используют клиентские сценарии, а не серверный код?

2. Перечислите HTML-элементы для работы со звуком и видео, поддерживаемые большинством браузеров.

3. Что нужно, чтобы организовать поддержку браузера Netscape Navigator в HTML-элементе объекта ActiveX?

4. Назовите ключевую возможность стандарта HTML+TIME, не доступную при использовании других методов анимации.

5. Опишите два способа создания временной шкалы при использовании HTML+TIME.

ГЛАВА 13

Форматирование данных, генерируемых Web-приложением

Занятие 1. Применение каскадных таблиц стилей 556

Занятие 2. Использование XSL-преобразований 570

В этой главе

ASP.NET предоставляет два взаимодополняющих способа форматирования выходных данныхWeb-приложений:

- управление **видом элементов Web-форм с помощью каскадных таблиц стилей,** позволяющих установить цвет, размер, шрифт и особенности поведения HTML-элементов Web-страницы;
- **применение XSL-преобразований для генерации HTML-документов на основе XML-файлов и отображения полученных документов на Web-формах.** XSL-преобразование переносит данные их XML-файла в HTML-элементы и форматирует их с помощью заданных стилей.

В этой главе вы научитесь выбирать подходящую методику форматирования и применять ее в Web-приложениях.

Прежде всего

Для занятий по материалам этой главы вам необходимо:

- уметь создавать Web-формы в режиме HTML на Visual Studio .NET;
- знать HTML-элементы, применяемые для создания Web-форм и HTML-страниц.

Занятие 1. Применение каскадных таблиц стилей

Каскадные таблицы стилей (Cascading Style Sheets, CSS) — это средство хранения и упорядочения сведений о форматировании HTML-элементов, составляющих Web-формы. Благодаря централизованному хранению этих сведений использование таблиц стилей облегчает изменение облика Web-приложений.

Visual Studio .NET содержит инструменты для редактирования стилей. При создании нового проекта Web-приложения для него генерируется таблица стилей по умолчанию. В этом занятии вы научитесь модифицировать таблицу стилей по умолчанию средствами инструментов Visual Studio .NET, подключать таблицы стилей к Web-формам и менять таблицы стилей во время выполнения.

Изучив материал этого занятия, вы сможете:

- ✓ объяснить, как ASP.NET применяет в Web-формах стили, определенные на разных уровнях;
- ✓ подключать таблицы стилей к Web-формам;
- ✓ модифицировать стили в таблице стилей;
- ✓ создавать новые стили и добавлять их в таблицу стилей;
- ✓ создавать классы стилей, которые можно применять к различным типам HTML-элементов;
- ✓ переключать таблицы стилей в период выполнения с целью генерации версии Web-формы с крупным шрифтом или форматирования содержимого Web-формы для печати;
- ✓ применять динамические элементы стиля для управления элементами Web-форм.

Продолжительность занятия — около 35 минут.

Как работают стили

Хотя во введении к этому занятию сказано, что таблицы стилей хранят сведения о формате в едином месте (в CSS-файле), в Web-приложениях форматирование применяется на трех уровнях. Прежде чем начинать работу с таблицами стилей, необходимо усвоить, как работают стили, определенные на разных уровнях. В таблице 13-1 дается описание уровней, на которых определяются стили, а также разъясняется, как такие стили применять к элементам Web-формы.

Таблица 13-1. Уровни определения стилей

Уровень	Где определяется	На что действует
Глобальный (global)	В файле таблицы стилей	Все страницы, ссылающиеся на эту таблицу стилей
Страничный (page)	В страничном элементе *head*	Все элементы текущей страницы
Локальный (inline)	Внутри HTML-элемента	Только текущий элемент

Иерархия этих уровней напоминает иерархию уровней области видимости, с которой вы как программист наверняка знакомы. Формат, определенный внутри элемента (локально), имеет приоритет перед форматом, заданным на уровне страницы, а тот — перед форматом, определенным глобально (в таблице стилей). Эти правила и дали имя таблицам стилей, которые называются *каскадными*. На рис. 13-1 показано, как стили, определенные на разных уровнях, применяются к элементу, содержащему текст.

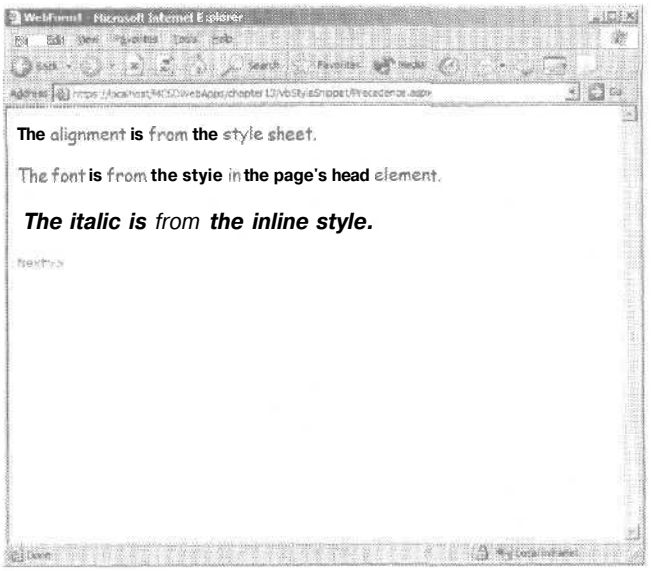


Рис. 13-1. Иерархия стилей

Далее показан HTML с определениями стилей, использованными для создания страницы на рис. 13-1.

```
<HTML>
   <HEAD>
         <title>WebForm1</title>
         <!- (1) Ссылка на таблицу стилей. ->
         <LINK REL="stylesheet" TYPE="text/css" HREF="Styles.css">
         <!- (2) Определение стиля на уровне страницы, ->
         <style>
         P {
         font-family: 'Comic Sans MS', Lucida Sans, sans-serif;
         font-size: medium:
         }
         </style>
   </HEAD>
   <body>
         <p>The alignment is from the style sheet.</p>
         <p>The font is from the style in the page's head element,
         </p>
         <!- (3) Определение стиля, встроенное в элемент ->
         <p style="FONT-SIZE: large; FONT-STYLE: italic">The italic is from
         the inline style.</p>
   </body>
</HTML>

<!- (1) Выдержка из таблицы стилей Styles.css , на которую ссылается элемент
HEAD. ->
```

```
p
{
   font-size: small;
   text-align: center;
}
```

Этот пример демонстрирует применение к одному и тому же элементу атрибутов стиля, определенных на разных уровнях. Хотя атрибут *font-size* определен на всех трех уровнях, видно действие только атрибутов, определенных на уровнях страницы и элемента. Это так, поскольку стиль, заданный на уровне страницы, имеет приоритет перед стилем, определенным в таблице стилей, Поскольку атрибуты *text-align*, *font-family* и *font-style* определяются только на одном из уровней, действует каждый из них.

Хранение определений стилей в таблице стилей (.css-файле) имеет ряд преимуществ перед хранением стилей в Web-формах или встраиванием их в HTML-элементы:

- благодаря централизованному хранению сведений о формате достаточно модифицировать стиль в одном месте, чтобы изменить вид всего приложения;
- отдельные таблицы стилей позволяют параллельно хранить несколько наборов правил, форматирующих выходную информацию для различных устройств и ситуаций. Например, у приложения могут быть таблицы стилей, генерирующие страницы крупным шрифтом, или страницы, оптимизированные для печати. Пользователь имеет возможность выбрать нужную таблицу стилей во время выполнения.

Совет Не следует определять стили на уровне страницы или внутри элементов, если можно обойтись глобальными стилями: частое использование локальных стилей затрудняет форматирование страниц Web-приложения.

Применение таблиц стилей с Web-формами

Для каждого нового Web-приложения, использующего язык Visual Basic .NET, Visual Studio .NET автоматически генерирует таблицу стилей Styles.css. Однако она не применяется автоматически ни к одной из Web-форм или HTML-страниц приложения.

Чтобы ее активизировать, необходимо добавить к страничному элементу *head* элемент *link* (см. строку, выделенную в примере ниже полужирным шрифтом):

```
<HEAD>
<title>WebForm1</title>
<meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
<meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
<meta name="vs_defaultClientScript" content="JavaScript">
<meta name="vs_targetSchema" content="http://schemas.microsoft.com/
intellisense/ie5">
<LINK REL="stylesheet" TYPE="text/css" HREF="Styles.css">
</HEAD>
```

Внимание! Таблица стилей по умолчанию автоматически генерируется только для новых проектов на языке Visual Basic .NET. Для проекта на Visual C# таблицу стилей придется создать «с нуля» либо использовать как заготовку копию таблицы стилей по умолчанию из любого проекта на Visual Basic .NET.

Модификация **стилей**

Мастер Style Builder позволяет изменить любой из стилей, составляющих таблицу стилей. Коррективы, внесенные в таблицу стилей, действуют на все Web-формы, ссылающиеся на эту таблицу стилей.

Чтобы изменить стиль, выполните следующие действия.

1. В Visual Studio откройте таблицу стилей. Определения стилей Visual Studio .NET отображает в окне Document, а структуру таблицы стилей — в окне Tool (рис. 13-2).

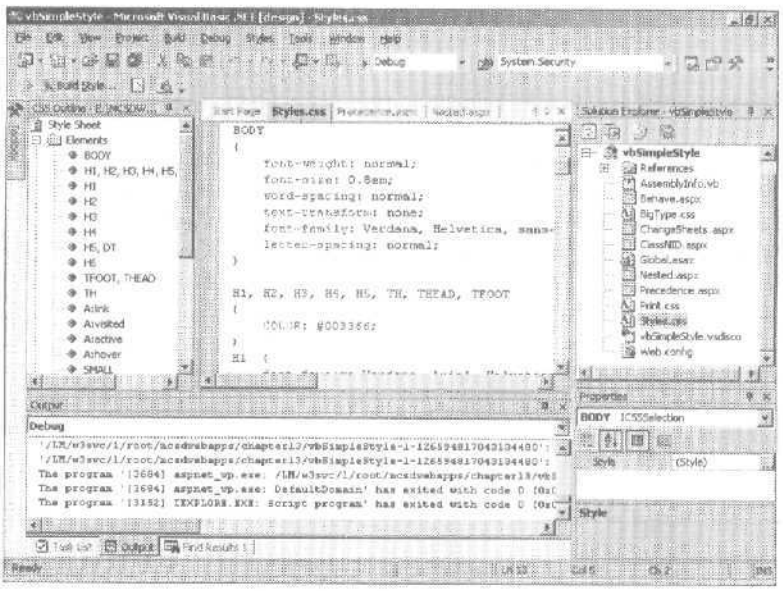


Рис. 13-2. Отображение таблицы стилей в окнах Document и Tool

2. Выберите в окне Tool стиль, который нужно изменить, — Visual Studio .NET покажет его определение в окне Document.
3. Щелкните правой кнопкой определение стиля (или его имя в окне Tool) и выберите из контекстного меню команду Build Style — Visual Studio .NET запустит мастер Style Builder (рис.13-3).

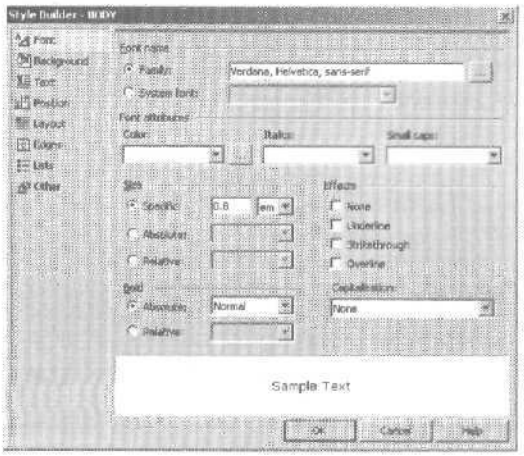


Рис. 13-3. Мастер Style Builder

4. В окне Style Builder установите все параметры форматирования для добавляемого или модифицируемого стиля. Закончив, щелкните **OK.**

   После этого вы увидите, что мастер Style Builder добавил к определению стиля новые или изменил существующие атрибуты.

Создание стилей

До сих пор мы создавали стили для форматирования HTML-элементов, указывая имена этих элементов; при этом поиск параметров форматирования по умолчанию для любых HTML-элементов в CSS осуществляется по имени элемента. Но при создании стилей можно также указывать имена классов стилей или идентификаторы элементов.

Использование имен классов позволяет оформлять различные элементы единым стилем, равно как и применять разные стили к одинаковым элементам в зависимости от того, как используются элементы. Ссылаясь на некоторый элемент по идентификатору, можно оформить его единым стилем везде, где он встречается в приложении.

Чтобы добавить стиль, выполните следующие действия.

1. Откройте таблицу стилей, щелкните правой кнопкой ее окно Document или Tool и выберите из контекстного меню команду Add Style Rule — Visual Studio .NET запустит мастер Add Style Rule (рис. 13-4).

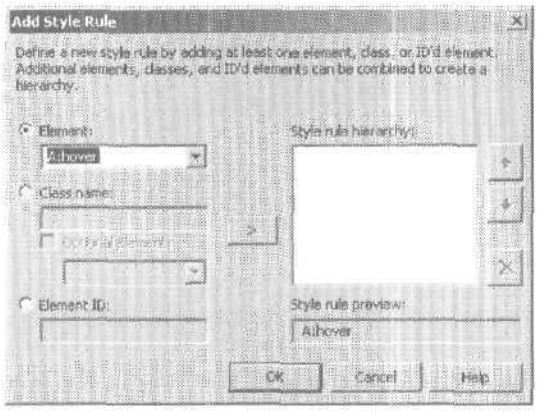


Рис. 13-4. Добавление стиля

2. Выберите тип создаваемого стиля. Форматируемые элементы можно задавать при помощи их имен, классов или идентификаторов. Чтобы создать стиль, введите имя элемента и щелкните кнопку «>» (добавить) — элемент будет добавлен к правилу стиля.
3. Повторите пункт *1* для всех элементов, которые следует отформатировать с помощью данного стиля.

Применение классов стилей

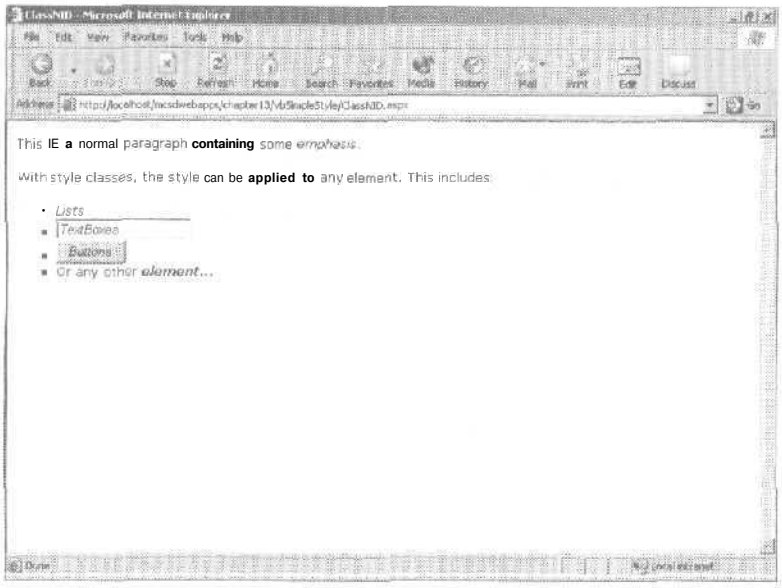
Использование классов стилей позволяет назначать одинаковый формат разным **HTML**-элементам Web-формы. При создании правила стиля для класса Visual Studio .NET добавляет к таблице стилей определение стиля с идентификатором вида *.имя_класса*, например:

```
.emphasis
{
    font-style: italic;
}
```

Этот класс стиля *.emphasis* задает курсивное начертание для всех использующих его элементов (рис. 13-5).

Для применения классов стилей к HTML-элементам служит атрибут *class, а к* серверным элементам управления — атрибут *CssClass (см.* атрибуты, выделенные в следующем примере HTML **полужирным шрифтом):**

```
<p>This is a paragraph containing <span class="emphasis">emphasis</span>.</p>
<asp:TextBox ID="Text1" Runat="server" CssClass="emphasis">Some text
</asp:TextBox>
```



Рис, 13-5. Результат применения класса стиля

Создание стилей для заданных элементов

Существует еще один способ определения стилей, когда форматируемый элемент Web-формы задают при помощи идентификатора. Идентификатор любого элемента должен быть уникальным в пределах страницы, поэтому использование идентификатора позволяет назначать форматирование не более чем для одного элемента страницы. При создании правила стиля для форматирования элемента с заданным идентификатором Visual Studio .NET добавляет к таблице стилей определение, содержащее идентификатор *#elementID*:

```
#inserted
{
      text-decoration: underline;
}

fldeleted
{
      text-decoration: line-through;
}
```

Это определение стиля задает для шрифта элементов с идентификатором *inserted* подчеркнутое начертание, а для элементов с идентификатором *deleted* — перечеркнутое (рис. 13-6).

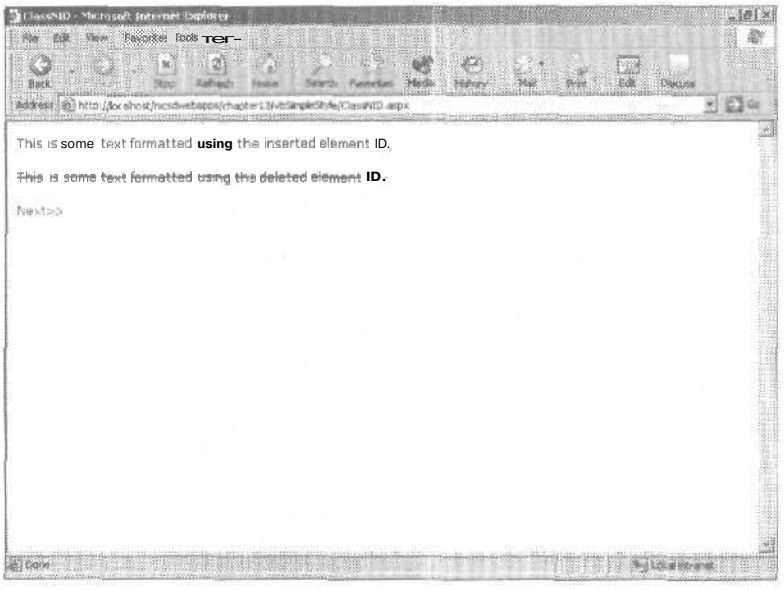


Рис. 13-6. Применение стилей при помощи идентификаторов элементов

Результат назначения стиля в Visual Studio .NET не всегда проявляется немедленно, прямо в окне дизайнера Web-формы. Если *форматирование*, назначенное через применение стиля, не заметно, попробуйте следующее:

- убедитесь, что таблица стилей сохранена;
- переключите Web-форму из режима HTML в Design и обратно;
- щелкните Web-форму правой кнопкой и выберите из контекстного меню команду View In Browser.

Создание вложенных стилей

Нетрудно заметить, что определения некоторых стилей из Styles.css (таблицы стилей по умолчанию) содержат списки из нескольких элементов. Такие определения задают форматирование для вложенных элементов. Например, следующие стили определяют разные маркеры для вложенных списков:

```
UL LI {
   list-style-type: square ;
   }

UL LI LI {
   list-style-type: disc;
   }

UL LI LI LI {
   list-style-type: circle;
   }
```

Результат применения такого стиля показан на рис.13-7.

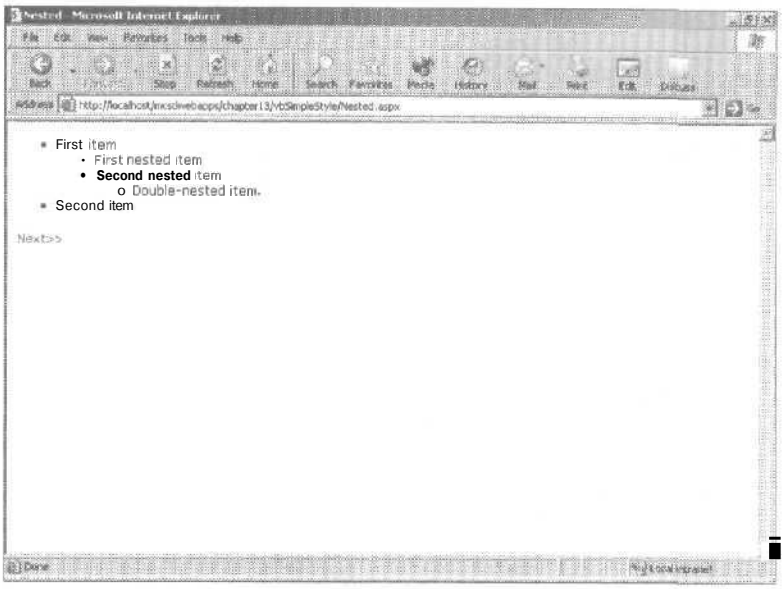


Рис.13-7. Форматирование вложенного списка

Чтобы создать собственный вложенный стиль, добавьте несколько элементов к списку Style rule hierarchy в диалоговом окне Add Style Rule (рис. 13-8).

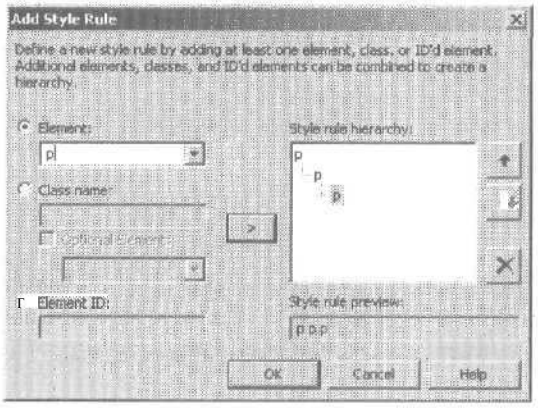


Рис. 13-8. Создание вложенных стилей

Переключение таблиц стилей во время выполнения

Как говорилось ранее, можно создать несколько таблиц стилей и разрешить пользователю выбирать набор стилей во время выполнения. Например, одна таблица стилей будет форматировать Web-форму для печати, а другая — назначать для нее крупный шрифт.

Для автоматического переключения таблиц стилей при печати Web-формы необходимо задать отдельные таблицы стилей для печати и отображения на экране. Это делается при помощи атрибута *media* элемента *link* таблицы стилей. Например, следующие элементы *link* определяют использование таблицы стилей по умолчанию для отображения Web-формы на экране, а при выводе на печать применяют к форме таблицу стилей Print.css:

```
<LINK REL="stylesheet" TYPE="text/css" HREF="Styles.css" media="screen">
<LINK REL="stylesheet" TYPE="text/css" HREF="Print.css" media="print">
```

Чтобы переключать таблицы стилей во время выполнения, необходимо создать клиентский сценарий, изменяющий элемент *href* в ссылке на таблицу стилей. Ниже показан пример HTML-кода Web-формы со ссылкой, щелкнув которую, пользователь может поменять таблицу стилей Styles.css на BigType.css (и обратно):

VBScript

```
<HTML>
<HEAD id="myhead">
<title>ChangeSheets</title>
<LINK ID="ScreenStyle" REL="stylesheet" TYPE="text/css" HREF="Styles.css">
<script language="vbscript">
' Сценарий на VBScript, переключающий таблицу стилей по умолчанию
' и таблицу, задающую крупный шрифт.
Sub SwitchSheets()
   If document.all("ScreenStyle").GetAttribute("HREF") = "Styles.css" then
         document.all("ScreenStyle").SetAttribute "HREF", "BigType.css", 0
   else
         document.all("ScreenStyle").SetAttribute "HREF", "Styles.css", 0
   end if
End Sub
</script>
</HEAD>
   <body>
         <form id="Form1" method="post" runat="server">
               <h2>Changing Style Sheets at Run-Time</h2>
               <p>This page first appears using the default style sheet,
               </p>
               <p><a onclick="SwitchSheets" href="#">Click here
</a> to switch between the standard and the enlarged-type version.</p>
         </form>
   </body>
</HTML>
```

JScript

```
<HTML>
   <HEAD>
         <titie>ChangeSheets</title>
         <LINK ID="ScreenStyle" REL="stylesheet" TYPE="text/css"
         HREF="Styles.css" media="screen">
         <LINK ID="PrintStyle" REL="stylesheet" TYPE="text/css"
         HREF="Print.css" media="print">
         <script id=clientEventHandlersJS language=jscript>
         function SwitchSheets()
         {
               // Переключение таблицы стилей по умолчанию и
```

```
            // таблицы, задающей крупный шрифт.
        if (document.all["ScreenStyle"].getAttribute("HREF") == "Styles.css")
            document.all["ScreenStyle"].setAttribute ("HREF",
            "BigType.css", 0);
        else
            document.all["ScreenStyle"].setAttribute ("HREF",
            "Styles.css", 0);
        }
        </script>
        </HEAD>
    <body MS_POSITIONING="FlowLayout">
        <FORM id="Form1" method="post" runat="server">
            <H2>Changing Style Sheets at Run-Time</H2>
            <P>This page first appears using the default style sheet,
            </P>
            <P><A onclick="SwitchSheets();" href="#">Click here</A>
            to switch between the standard and
            the enlarged-type version.</P>
            <A href="Behave.aspx">Next&gt;&gt;</A>
        </FORM>
    </body>
</HTML>
```

На рис. 13-9 показано, как меняется Web-форма в период выполнения, когда пользователь щелчком кнопки назначает другую таблицу стилей.

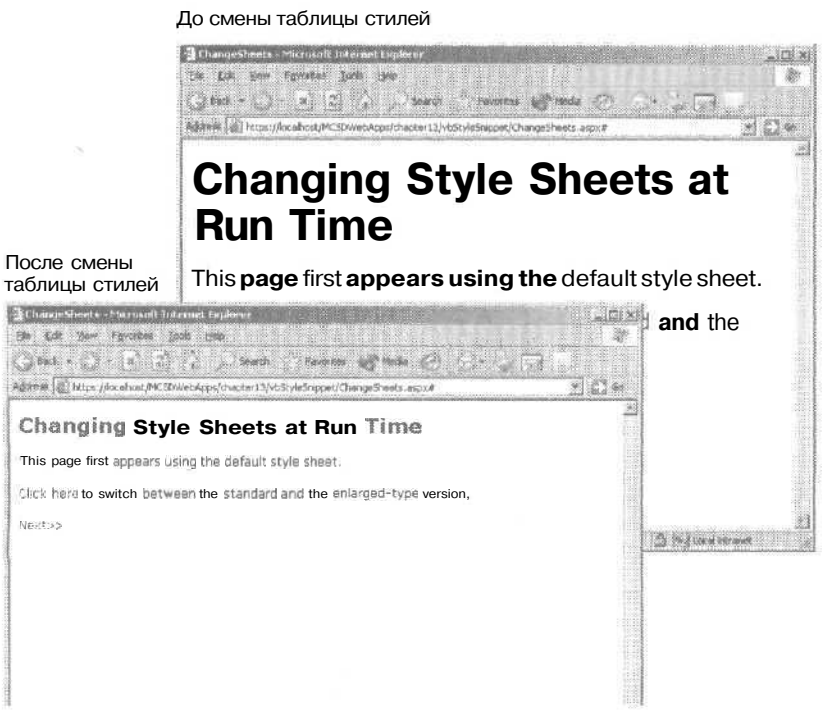


Рис. 13-9. Смена таблицы стилей во время выполнения

Использование динамических элементов стиля

Помимо внешнего вида, стили определяют поведение HTML-элементов во время выполнения при помощи *динамических элементов стиля* (behaviors), представляющих своего рода динамическое форматирование. Поскольку Web-содержимое не всегда статично, динамические элементы стиля используются в таблицах стилей для описания динамических изменений.

Краткое знакомство с динамическими элементами стиля состоялось в главе 12, где для анимации HTML-элементов при помощи HTML+TIME вы применяли класс стиля *time:*

```
<style>
   .time    {behavior: url(#default#time2); }
</style>
```

Свойство *behavior* импортирует файлы динамических элементов стиля с кодом на JScript или VBScript, определяющим действия этого элемента стиля. В разделе Behaviors Library справочной системы Visual Studio .NET приводятся примеры динамических элементов стиля, которые можно использовать в собственном коде.

Например, следующий HTML импортирует файл-пример динамического элемента стиля Movable.htc и применяет его для форматирования абзаца. В результате получается абзац, который во время выполнения можно перетаскивать по поверхности Web-формы:

```
<HEAD>
   <title>Behave</title>
   <style>
         .movable {behavior: url(behaviors/
   movable.htc); CURSOR: move;
         mv-boundary:  0  800  600  0}
   </style>
</HEAD>
<body>
   <form id="Form1" method="post" runat="server">
         <p class="movable">You can drag this text around the screen.
        </p>
   </form>
</body>
</HTML>
```

Это определение стиля для класса *movable* содержит URL файла динамического элемента стиля; кроме того, здесь заданы стиль указателя мыши и границы действия. Стиль назначается посредством атрибута *class* элемента *p*. Отформатированный этим стилем абзац пользователь может перетаскивать во время выполнения (рис. 13-10).

Динамический элемент стиля *movable* применяется с любыми элементами Web-формы, включая серверные элементы управления. Однако некоторые динамические элементы стиля не совместимы с отдельными типами серверных элементов управления. Например, динамический элемент стиля *ColorPick* (см. пример кода ниже) применяется только с HTML-элементами управления «текстовое поле» (эти элементы служат для ввода данных типа *text*):

```
<HTML xmlns:control>
<HEAD>
   <title>Behave</title>
```

```
    <style>
          .color { BEHAVIOR: url(behaviors/ColorPick.htc) }
    </style>
</HEAD>
<body>
    <form id="Form1" method="post" runat="server">
          <p>Use the color picker below to change the background color:
          </p>
          <input ID="ColorPic" type="text" class="color" value="#fFFf00"
          onchange="document.bgColor=ColorPic.value; " style="cp-grid-size:2">
    </form>
</body>
</HTML>
```

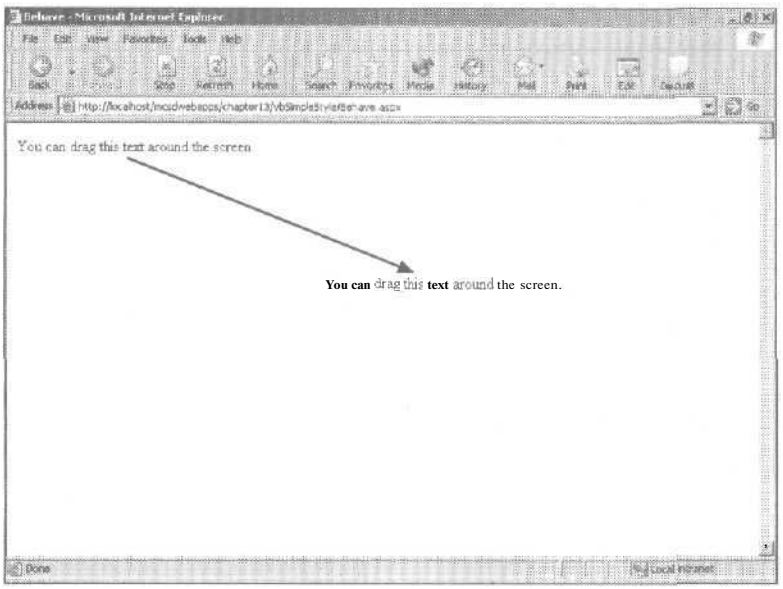


Рис. 13-10. Динамический элемент **стиля** *movable*

Показанный HTML-код отображает палитру, щелкнув которую, пользователь может выбрать цвет фона во время выполнения (рис. 13-11),

Наконец, некоторые динамические элементы стиля создают собственные типы элементов, которые можно применять с Web-формами. Например, пример динамического элемента Slider создает элемент slider, позволяющий принимать данные от пользователя. Чтобы использовать элемент управления Slider. прежде необходимо добавить к элементу *html* Web-формы атрибут *xmlns*, объявляющий имя элемента управления:

```
<HTML xmlns:control>
    <HEAD>
          <title>Behave</title>
          <style>
                control\:slider { BEHAVIOR: url(behaviors/
                slider.htc) }
```

```
        </style>
    </HEAD>
    <body>
        <form id="Form1" method="post" runat="server">
            <control:slider ID="Slider"
            onchange="alert(Slider.value) "
            STYLE="sl-orientation:horizontal">
            </control:slider>
        </form>
    </body>
</HTML>
```

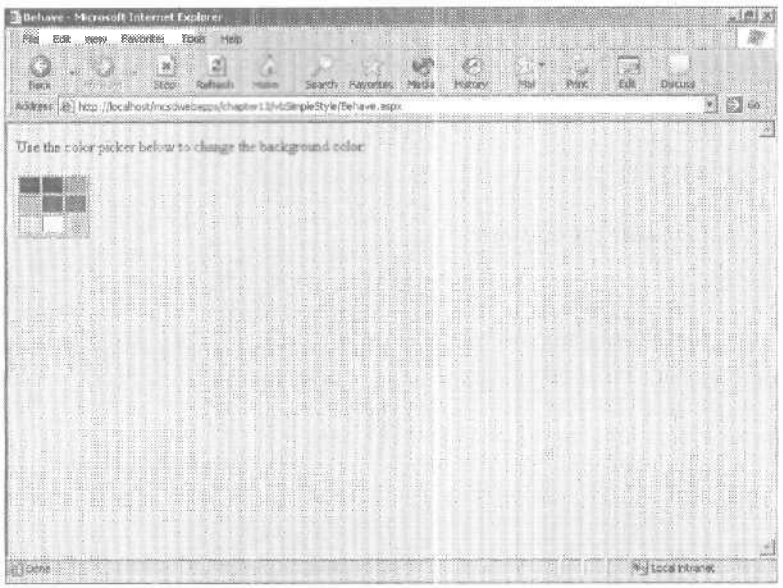


Рис. 13-11. Динамический элемент стиля *ColorPick*

Этот HTML-код создает элемент управления slider, отображающий свое значение при перемещении ползунка на Web-форме (рис. 13-12).

Внимание! Применять динамические элементы стиля, создающие собственные элементы управления на Web-формах, допустимо, но решать задачи, которые не удается осуществить при помощи стандартных серверных и HTML-элементов управления, все же лучше средствами нестандартных элементов управления. Элементы управления, созданные динамическими элементами стиля, некорректно отображаются дизайнером Visual Studio .NET и не поддерживают свойства и методы, легко доступные при использовании серверного кода, а нестандартные элементы лишены обоих этих недостатков.

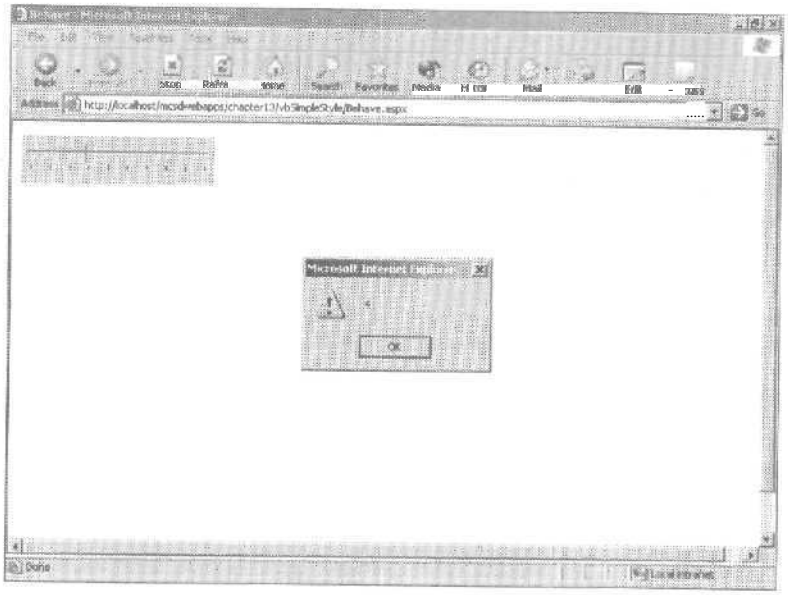


Рис. 13-12. Динамический элемент стиля ***Slider***

Занятие 2. Использование XSL-преобразований

На этом занятии вы познакомитесь с применением XSL для преобразования XML-файлов с помощью выражений на языке XML Path Language (XPath).

Изучив материал этого занятия, вы сможете:

- ✓ применять элемент управления XML для преобразования данных XML-файлов в форматированный HTML-текст для отображения на Web-форме;
- ✓ рассказать, чем отличаются языки XML и HTML;
- ✓ создавать XML-файлы;
- ✓ создавать шаблоны XSL, управляющие преобразованием XML-файлов в HTML-документы;
- ✓ исполнять повторяющиеся и условные действия, а также вставлять HTML-элементы и сортировать выходные данные с помощью XSL-элементов;
- ✓ вычислять значения XPath-выражений.

Продолжительность занятия — около 40 минут.

Как работают XSL-преобразования

При помощи серверного элемента управления XML XSL-преобразования (XSL Transformations, XSLT) генерируют из входного XML-файла форматированные данные, применяя набор правил, хранящихся в файле описания XSL. Этот XSL-файл напоминает каскадные таблицы стилей, с которыми вы познакомились в предыдущем занятии. Подобно им, XML-файл предоставляет правила для форматирования выходных данных, отображаемых на Web-форме. Однако в отличие от CSS-файлов XSL-файлы способны задавать произвольное размещение элементов на Web-форме, а также выполнять логические операции, например организовывать циклы и проверять условия.

Можно рассматривать XSL-файлы как уровень интеллектуального форматирования, который скорее дополняет, чем заменяет CSS. XSL обычно применяют для размещения структурированных данных на Web-формах, а CSS — для определения внешнего вида этих данных.

Чтобы выполнить XSL-преобразование на Web-форме, сделайте следующее.

1. Добавьте на Web-форму серверный элемент управления XML.
2. Занесите в его свойство *Document Source* имя XML-файла, подлежащего форматированию.
3. В свойство *Document Source* этого элемента управления поместите имя XSL-файла, определяющего формат выходных данных.

Во время выполнения серверный элемент управления XML обрабатывает данные входного XML-файла, применяя к нему описание XSL и генерируя форматированные данные (рис. 13-13).

Создание XML-файла

Язык XML — это способ описания структурированных данных для представления их в текстовом формате. В XML элементы данных идентифицируются тегами вида *<имя_элемента>...</имя_элемента>*— этим XML похож на HTML. Но, в отличие от HTML, в XML действуют строгие правила:

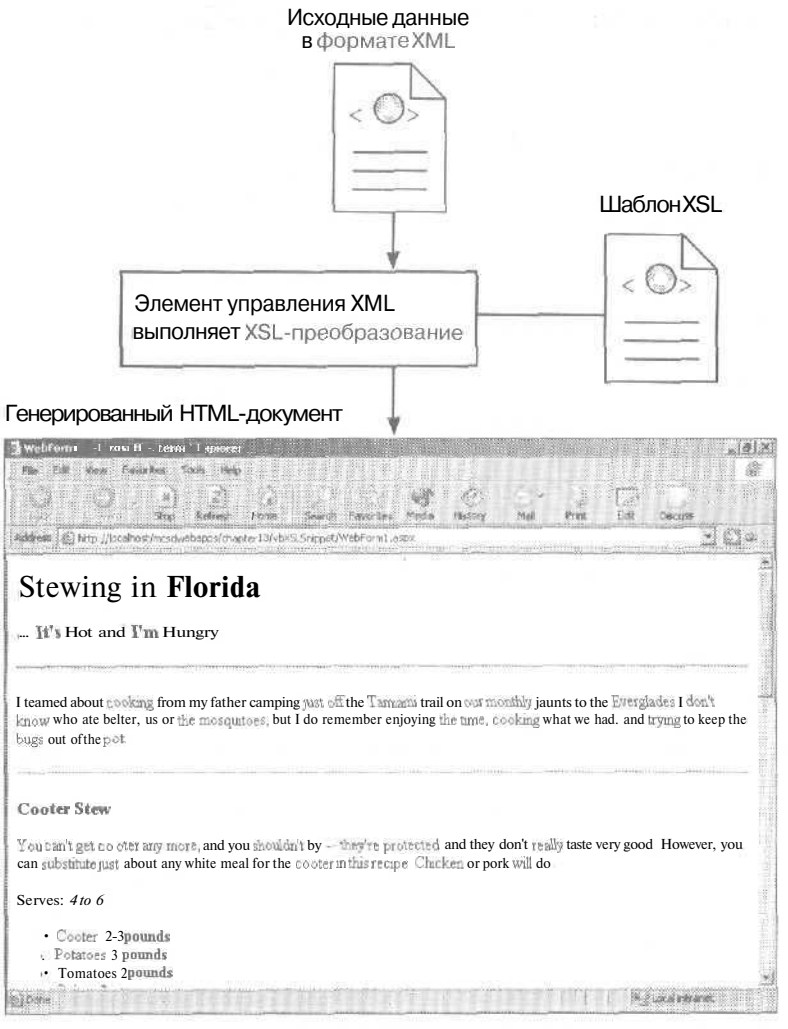


Рис. 13-13. Схема XSL-преобразования

- **у каждого элемента должны быть открывающий и замыкающий тэги.** В HTML-документах у тегов *<li>*, *
* и некоторых других может не быть замыкающих тегов (соответственно *
* и *</br>*), что не допускается в XML-документах;
- **имена тегов, определяющих элементы, чувствительны к** регистру, В HTML теги *<P>* и *<p>* эквивалентны, а в XML — нет;
- **значения атрибутов всегда заключаются в двойные кавычки.** При определении числовых атрибутов в HTML разрешается опускать двойные кавычки, например, так: *<table cols=3>*, в XML же кавычки должны быть всегда: *<table cols="3">*;
- **вложенный (дочерний)** элемент **необходимо закрыть до того, как будет закрыт его элемент-контейнер (родительский элемент).** Например, в HTML допустимо следующее определение: *Bold textitalic boldregular italic*, но в XML оно должно быть записано только так: *Bold textitalic boldregular italic*.

Благодаря этим строгим правилам XML обладает одной важной возможностью, отсутствующей в HTML: в XML разрешено определять собственные теги для разметки любых данных.

Чтобы создать XML-файл в Visual Studio .NET, выберите из меню Project команду Add New Item, в списке Templates укажите элемент XML file и щелкните Open.

Перед созданием нового XML-файла полезно выписать на бумаге все теги и нарисовать структуру документа, Это поможет избежать неприятностей, когда из-за пропущенного тэга приходится переделывать все элементы. Например, следующий XML-документ описывает данные книги рецептов, которая и будет примером для этого занятия:

```
<cookbook>
   <title></title>
   <subtitle></subtitle>
   <description></description>
   <recipe>
         <title></title>
         <subtitle></subtitle>
         <description></description>
         <servings></servings>
         <ingredients>
               <ingredient>
                     <name></name>
                     <quantity></quantity>
               </ingredient>
               <ingredient>
                     <name></name>
                     <quantity></quantity>
               </ingredient>
         </ingredients>
         <instructions>
               <introduction></introduction>
               <stepx/step>
               <step></step>
               <step></step>
               <summary></summary>
         </instructions>
   </recipe>
</cookbook>
```

Данные, составляющие каждый из элементов, заключаются между открывающим и замыкающим тэгами. Здесь важно усвоить, что структура XML-файла и имена элементов описывают *содержимое, а* не *форматирование* элементов. Содержимое, размеченное подобным образом, можно отформатировать при помощи XSL-файлов.

Структура XML строго иерархична и состоит из элементов, называемых *узлами* (nodes). Узлы связаны отношениями типа «родитель — потомок»; для описания связей между узлами применяют язык XML Path или XPath (рис. 13-14).

Язык XML хорошо подходит для описания структурированных данных, например биржевых котировок, сведений о работниках, инвентарных списков, информации о состоянии заказа и других данных, которые обычно обрабатываются в упорядоченном виде, но мало приспособлены для описания неструктурированных данных и результатов творческой деятельности.



Рис. 13-14. **Узлы XML**

Создание XSL-файла

XSL-файл определяет разметку и логику для преобразования содержимого XML-файла в данные, отображаемые Web-формой для пользователя. Чтобы создать XSL-файл в Visual Studio, выберите из меню Project команду Add New Item, укажите в списке Templates элемент XSL file и щелкните Open.

XSL-файл содержит выражения, определяющие шаблон для выборки элементов из XML-файла. Каждый шаблон состоит из стандартных HTML-элементов, определяющих формат элементов, импортированных из XML-файла. Ниже показан шаблон XSL, который преобразует элементы заголовка, подзаголовка и текста введения книги рецептов, определенные в XML-файле, в HTML-элементы *h1*, *h2* и *p* соответственно.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/cookbook">
      <xsl:apply-templates select="cookbook" />
      <h1><xsl:value-of select="title" /></h1>
      <h3><xsl:value-of select="subtitle" /></h3>
      <hr />
      <p><xsl:value-of select="introduction" /></p>
      <hr />
      <h2><xsl:text>Recipes</xsl:text></h2>
      <xsl:apply-templates select="recipe" />
   </xsl:template>
</xsl:stylesheet>
```

Сводка XSL-элементов и атрибутов, использованных в этом примере, дана в таблице 13-2,

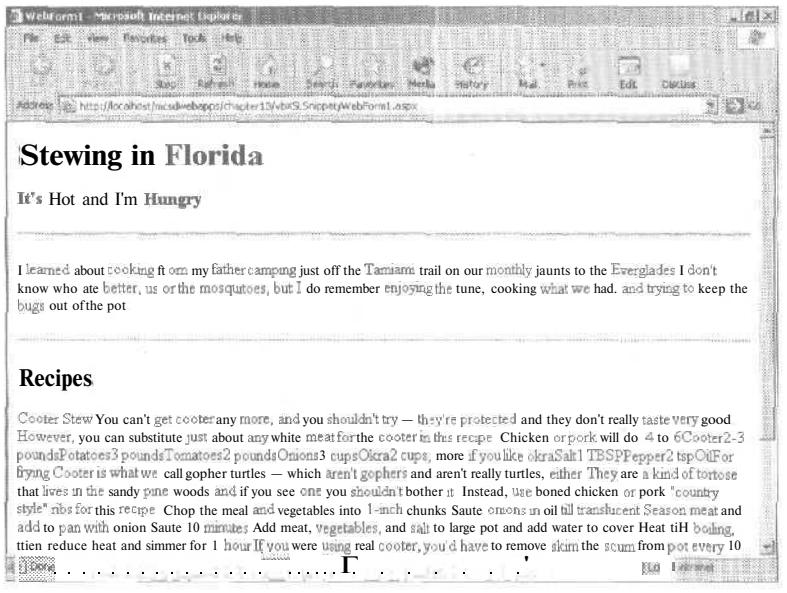
Таблица 13-2. Базовые XSL-элементы

Имя	Атрибуты	Назначение
xsl:stylesheet	*version*	Определяет используемую версию XSL; на время написания этой книги была доступна версия 1.0
	xmlns:xsl	Определяет префиксы элементов XSL-файла
xsl:template	*match*	Определяет шаблон узла XML
xsl:apply-templates	*select*	Применяет шаблон к выбранному узлу

Таблица 13-2. *(окончание)*

Имя	Атрибуты	Назначение
xsl:value-of	*select*	Извлекает значение узла XML или вычисляет значение выражения Xpath
xsl:text		Включает в выходные данные текст (в виде литералов) и знаки пробела

На рис. 13-15 показано, как эти элементы преобразуют XML-данные книги рецептов при помощи серверного элемента управления XML.



Рис, 13-15. Базовые XSL-элементы в действии

XSL-элементы, результат применения которых показан на рис. 13-15, весьма скудно отображают данные с помощью XSL: определяют шаблон, импортируют в него данные, отображают значения XML-элементов и вставляют текст. Наверное, такие простые действия легко выполнить с помощью обычных HTML-тэгов и таблиц стилей, но XSL дает некоторые преимущества, позволяя:

- корректировать положение элементов (разметку страницы), не изменяя сами данные;
- выполнять логические операции над данными, в том числе организацию циклов и условную обработку.

Эти преимущества обсуждаются в следующих разделах.

Изменение разметки

Чтобы изменить разметку страницы, в XSL-файле задают перемещения элементов. Ниже показан пример XSL-кода, который отображает XML-данные книги рецептов в две колонки и создает подзаголовок:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/cookbook">
```

```
        <xsl:apply-templates select="cookbook" />
          <table>
                <tr>
                      <td width="300">
                            <h3><xsl:value-
                            of select="subtitle" />
                            <xsl:text>...</xsl:text></h3>
                            <h1><xsl:value-of select="title"/>
                            </h1>
                      </td>
                      <td width="500">
                         <xsl:text>recipe text goes here..
                         </xsl:text>
                      </td>
                </tr>
                <tr>
                      <td width="300">
                            <hr />
                            <xsl:value-of select="introduction" />
                            <hr />
                      </td>
                      <td width="500">
                      </td>
                 </tr>
          </table>
      </xsl:template>
  </xsl:stylesheet>
```

Результат нового XSL-преобразования показан на рис. 13-16.

Создание дополнительных шаблонов

В XSL-файле может быть несколько шаблонов. В принципе, собственный шаблон XSL допустимо назначать для каждого узла XML, требующего уникального форматирования. К примеру, для XML-файла книги рецептов, показанного ранее, можно определить четыре шаблона:

- основной, форматирующий заголовок и описание книги. Он считается отправной точкой XSL-преобразования. Обработка XSL-файла начинается *с* этого шаблона, поскольку он задает форматирование корневого узла (match="/cookbook");
- шаблон рецептов, определяющий форматирование рецептов книги;
- шаблон ингредиентов, форматирующий список ингредиентов;
- шаблон для форматирования указаний по приготовлению блюд — они следуют за списком ингредиентов.

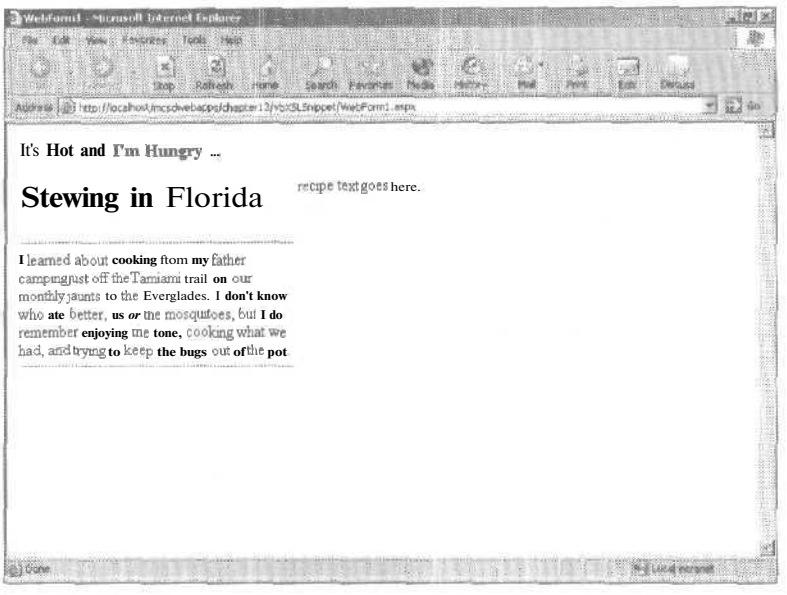


Рис. 13-16. Изменение разметки страницы при помощи XSL

Элемент *xsl:apply-templates* назначает шаблон для узла. Ниже показан пример XSL, применяющий шаблоны для форматирования списка ингредиентов и инструкций, взятые из шаблона рецепта, который, в свою очередь, берется из основного шаблона. Ссылки между шаблонами выделены в примере **полужирным шрифтом:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

   <xsl:template match="/cookbook">
      <xsl:apply-templates select="cookbook" />
      <h1><xsl:value-of select="title" /></h1>
      <h3><xsl:text>...</xsl:text>
   <xsl:value-of select="subtitle" /> </h3>
      <hr />
      <p><xsl:value-of select="introduction" /></p>
      <hr />
      <xsl:apply-templates select="recipe" />
   </xsl:template>

   <xsl:template match="recipe">
         <h3><xsl:value-of select="title" /></h3>
         <p><xsl:value-of select="description" /></p>
         <p><b><xsl:text>Serves: </xsl:text></b>
         <i><xsl:value-of select="servings" /></i></p>
         <xsl:apply-templates select="ingredients" />
         <xsl:apply-templates select="instructions" />
   </xsl:template>
```

```
    <xsl:template match="ingredients">
    <ul>
          <li><xsl:value-of select="ingredient/name" />
          <xsl:text>     </xsl:text>
          <b><xsl:value-of select="ingredient/quantity" /></b>
</li>
    </ul>
    </xsl:template>

    <xsl:template match="instructions">
    <p><xsl:value-of select="introduction" /></p>
    <ol>
          <li><xsl:value-of select="step" /></li>
    </ol>
    <p><xsl:value-of select="summary" /></p>
    </xsl:template>

</xsl:stylesheet>
```

Во время выполнения этот XSL генерирует HTML-страницу, показанную на рис. 13-17.

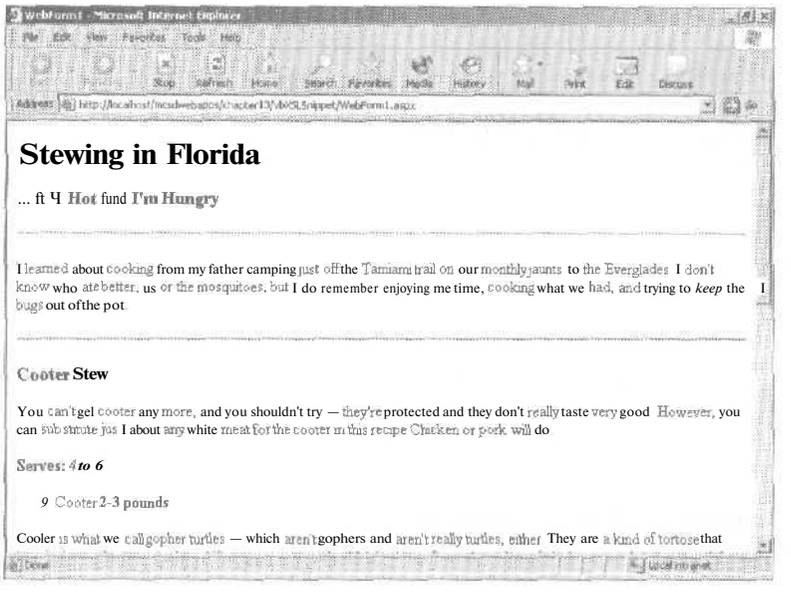


Рис. 13-17. Совместное использование нескольких шаблонов

Но погодите-ка: на странице указан только один ингредиент и дана одна инструкция, а где остальные? Дело в том, что все созданные нами шаблоны обрабатывают лишь по одному узлу. Чтобы отформатировать при помощи одного шаблона несколько узлов (то есть применить этот шаблон несколько раз), необходимо воспользоваться XSL-элементами, определяющими циклы, но об этом — в следующем разделе.

Выполнение повторяющихся действий

Для форматирования повторяющихся элементов XML-файла, таких, как элементы списков и строки таблиц, применяется XSL-элемент *xsl:for-each*. Этот элемент распространяет заданный формат на все экземпляры повторяющегося узла (см. пример ниже).

```
<xsl:for-each select="ingredient">
   <xsl;value-of select="name" /><br />
</xsl:for-each>
```

Этот XSL генерирует имя для каждого узла ingredient. Для ссылки на повторяющийся узел применяется XPath-идентификатор «.». Например, следующий XSL генерирует текст для каждого узла step, присутствующего в XML-файле:

```
<xsl:for-each select="step">
   <xsl:value-of select="." /xbr />
</xsl:for-each>
```

Следующий XSL использует элемент *xsl:for-each* (см. строки, выделенные в приведенном ниже примере **полужирным шрифтом)** для отображения всех элементов, представляющих рецепты, ингредиенты и инструкции книги рецептов:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

   <xsl:template match="/cookbook">
      <xsl:apply-templates select="cookbook" />
      <h1><xsl:value-of select="title" /></h1>
      <h3><xsl:text>...</xsl:text><xsl:value-of select="subtitle" />
</h3>
      <hr />
      <p><xsl:value-of select="introduction" /></p>
      <nr />
      <xsl:for-each select="recipe">
<xsl:apply-templates select="." />
</xsl:for-each>
   </xsl:template>

   <xsl:template match="recipe">
<h3><xsl:value-of select="title" /></h3>
<p><xsl:value-of select="description" /></p>
<p><b><xsl:text>Serves: </xsl :textx/b>
<i><xsl :value-of select="servings" /x/ix/p>
<xsl:apply-templates select="ingredients" />
<xsl:apply-templates select="instructions" />
   </xsl:template>

   <xsl:template match="ingredients">
   <ul>
<xsl:for-each select="ingredient">
<li><xsl:value-of select="name" />
```

```
<xsl:text> </xsl:text>
<b><xsl:value-of select="quantity" /></b></li>
</xsl:for-each>
   </ul>
   </xsl:template>

   <xsl:template match="instructions">
   <p><xsl:value-of select="introduction" /></p>
   <ol>
<xsl:for-each select="step">
<li><xsl:value-of select="." /></li>
</xsl:for-each>
   </ol>
   <p><xsl:value-of select="summary" /></p>
   </xsl:template>

</xsl:stylesheet>
```

Во время выполнения этот XSL форматирует XML книги рецептов, генерируя следующую страницу (рис. 13-18).

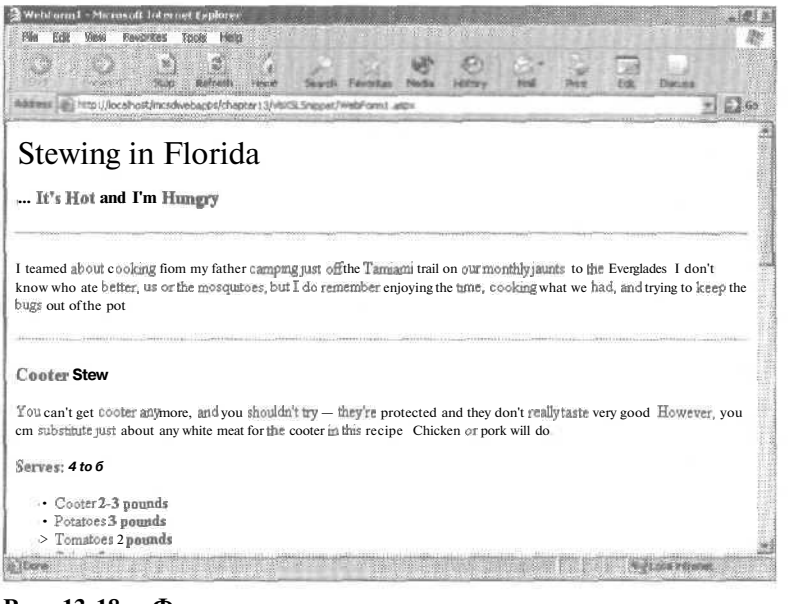


Рис. 13-18. Форматирование повторяющихся элементов

Добавление гиперссылок и других HTML-элементов

Элемент *xsl;element* позволяет составлять HTML-элементы с различными атрибутами, такие, как гиперссылки, а также серверные или HTML-элементы управления. Например, следующий XSL генерирует гиперссылку, возвращающую пользователя к началу страницы:

```
<xsl:element name="a">
   <xsl:attribute name="name">
```

```
      <xsl:text>#top</xsl:text>
   </xsl:attribute>
   <xsl:text>Return to top.</xsl:text>
</xsl:element>
```

Во время выполнения XSL, показанный выше, генерирует такой HTML-код:

```
<a href="#top">Return to top.</a>
```

Совместив этот XSL с Xpath-функцией *generate-id*, можно автоматически сгенерировать оглавление документа со ссылками на элементы документа (см. строки, выделенные полужирным шрифтом в показанном ниже примере):

```
<xsl:template match="/cookbook">
   <h1><xsl:value-of select="title" /></h1>
   <h3><xsl:text>...</xsl:text><xsl:value-of select="subtitle" />      </
h3>
   <hr />
   <p><xsl:value-of select="introduction" /></p>
   <hr />
   <!- Вывести оглавление ->
<h4>Contents</h4>
<xsl:apply-templates mode="contents" select="recipe" />
   <xsl:for-each select="recipe">
      <xsl:apply-templates select="." />
   </xsl:for-each>
</xsl:template>

<xsl:template match="recipe">
      <!- Создать ссылку на элемент документа ->
      <xsl:element name="a">
            <xsl:attribute name="name">
                  <xsl:value-of select="generate-id(title)" />
            </xsl:attribute>
            <h3><xsl:value-of select="title" /></h3>
      </xsl:element>
      <p><xsl:value-of select="description" /></p>
      <p><b><xsl:text>Serves: </xsl:textx/b>
      <i><xsl:value-of  select="servings"  /x/ix/p>
      <xsl:apply-templates select="ingredients" />
      <xsl:apply-templates select="instructions" />
</xsl:template>

<xsl:template  mode="contents" match="recipe">
      <!- Создать ссылки оглавления ->
      <xsl:element name="a">
            <xsl:attribute name="href">
                  <xsl:text>#</xsl:text>
                  <xsl:value-of select="generate-id(title)" />
            </xsl;attribute>
```

```
            <xsl:value-of select="title" />
        </xsl:element>
        <br />
    </xsl:template>
```

Во время выполнения этот XSL генерирует страницу, показанную на рис. 13-19.

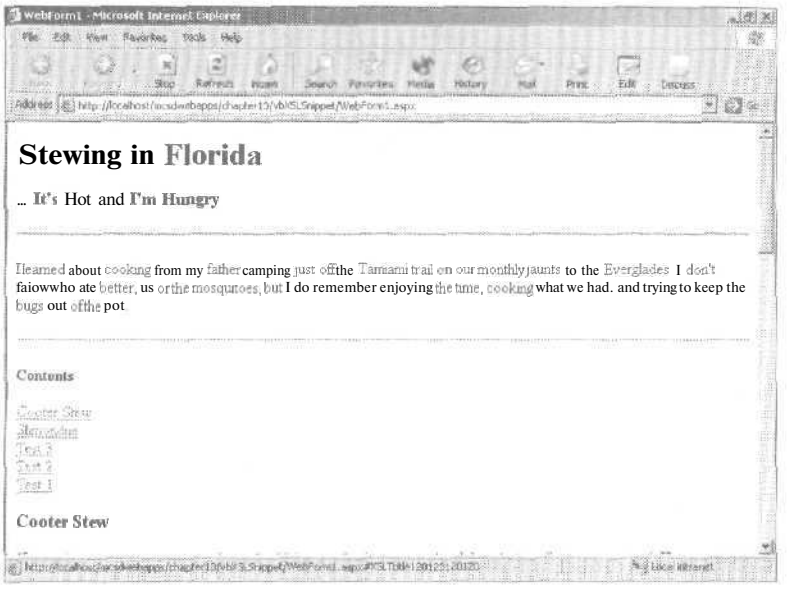


Рис. 13-19, Генерация оглавления

Xpath-функция *generate-id* генерирует для узла XML, переданного ему в качестве аргумента, уникальный идентификатор вида *XSLTимя_узла#*, где # — 12-разрядное целое число; например XSLTtitle120123120120.

Сортировка элементов

Для сортировки элементов, вложенных в другой элемент XSL, применяется элемент *xsl:sort*. Например, следующий XSL сортирует оглавление, созданное в предыдущем разделе:

```
<h4>Contents</h4>
<xsl:apply-templates mode="contents" select="recipe">
   <xsl:sort select="title" order="ascending" />
</xsl:apply-templates>
```

Элемент *xsl:sort* можно включать в элементы *xsl:apply-templates* (см. предыдущий код) или *xsl:for-each*.

Выполнение условных действий

В XSL есть два набора элементов, выполняющих некоторые действия в зависимости от условия. Элемент *xsl:if* применяется, когда нужно проверить одно условие, а *xsl:choose* с вложенными элементами *xsl:when* и *xsl:otherwise* — когда необходимо проверить несколько условий. Эти элементы перечислены в таблице 13-3.

Таблица 13-3. XSL-элементы, выполняющие действия по условию

Имя	Атрибуты	Действие
xsl:if	*test*	Проверяет булево выражение; если результат — *true,* обрабатывает вложенные элементы
xsl:choose		Проверяет несколько условий с помощью вложенных элементов *xsl:when* и *xsl:otherwise*; эквивалент операторов Select Case (в Visual Basic .NET) или switch (в Visual C#)
xsi.when	*test*	Проверяет одно булево выражение, вложенное в элемент *xsl:choose*; если результат — *true,* обрабатывает содержимое этого элемента и выходит из него; эквивалент оператора Case (в Visual Basic .NET) или case (в Visual C#)
xsl:otherwise		Обрабатывает вложенный элемент, если ни один из элементов *xsl:when*, вложенных в элемент *xsi.choose* не дал *true;* эквивалент операторов Case Else (в Visual Basic .NET) или *default* (в Visual C#)

Атрибут условного элемента *test* вычисляет значение выражения Xpath, чтобы определить, следует ли обрабатывать содержимое этого элемента. Например, следующий XSL создает оглавление, если в книге содержится более 4 рецептов:

```
<xsl:if test="count(recipe) > 4">
   <h4>Contents</h4>
   <xsl:apply-templates mode="contents" select="recipe" />
</xsl:if>
```

Этот XSL использует XPath-функцию *count*, чтобы выяснить, сколько узлов recipe содержит документ. Если таких узлов больше 4, условный элемент возвращает имя заголовочного стиля *Contents.* В результате обрабатывается шаблон *contents*, в противном случае оглавление не создается.

Аналогичным образом можно применять XPath-функции внутри элементов *xsi.choose* для генерации различных HTML-тэгов в зависимости от нескольких условий. Например, следующий XSL вставляет разрыв строки после каждого четвертого элемента оглавления, остальные элементы он разделяет символами табуляции;

```
<xsl:template mode="contents" match="recipe">
      <xsl:element name="a">
            <xsl:attribute name="href">
                  <xsl:text>#</xsl:text>
                  <xsl:value-of select="generate-id(title)" />
            </xsl:attribute>
            <xsl:value-of select="title" />
      </xsl:element>
      <xsl:choose>
            <xsl:when test="position() mod 4 - 0">
                  <br />
            </xsl:when>
            <xsl:otherwise>
                  <xsl:text>  </xsl:text>
            </xsl:otherwise>
      </xsl:choose>
</xsl:template>
```

Этот XSL применяет Xpath-функцию *position,* которая возвращает порядковый номер текущего узла в наборе его родительского узла, а также Xpath-оператор *mod,* позволяющий проверить, делится ли номер текущей позиции на четыре без остатка,

Примечание Вы правы, если полагаете, что здесь мы лишь слегка коснулись возможностей XSL; XML, XSL и XPath. Предыдущее занятие дает достаточно сведений, чтобы начать изучать эти языки самостоятельно. Подробно они описаны в документации на Microsoft XML SDK, которая входит в справочную систему Visual Studio .NET. Также рекомендуем прочесть следующие книги издательства Microsoft Press: Michael J. Young, «*XML Step by Step*» (второе издание); R. Allen Wyke, Sultan Rehman, Brad Leupen, «*XML Programming (Core Reference)*».

Резюме

- Иерархия и приоритет стилей определяются уровнями, на которых они применяются. Стиль может быть локальным (применяется к элементу), страничным (применяется к страничному элементу *head*) и глобальным (применяется посредством таблицы стилей).
- Элемент *link*, вложенный в страничный элемент *head,* служит для подключения таблицы стилей.
- Для модификации существующих стилей применяют мастер Style Builder из Visual Studio .NET.
- Чтобы назначить одинаковый формат разным типам элементов, создайте класс стиля.
- Атрибут *CssStyle* позволяет применять классы стиля к серверным элементам управления; а атрибут *class* — к HTML-элементам.
- XSL-преобразование генерирует HTML-текст на основе данных XML-файла при помощи серверного элемента управления XML.
- Язык XML применяется для упорядочения данных в виде иерархического набора узлов.
- Для форматирования узлов XML-файла служат XSL-шаблоны.
- Элемент *xsl:value-of* позволяет извлекать данные из узла XML.
- Элемент *xsl:text* позволяет включать текст в виде литералов, а также знаки пробела в данные, генерируемые XSL-преобразованием.
- Элемент *xsl:for-each* позволяет выполнять повторяющиеся действия для форматирования списков, таблиц и других повторяющихся узлов XML-файлов,
- Элементы *xsl:if*, *xsl:choose*, *xsl:when* и *xsl:otherwise* служат для условной обработки XSL-элементов.

Практикум 13. Создание книги рецептов

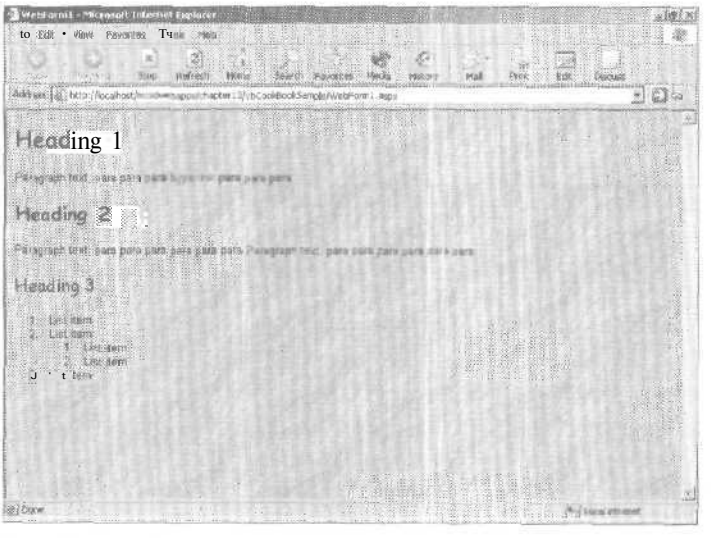
Задача этого практикума — создать Web-форму, использующую две таблицы стилей. Альтернативная таблица стилей применяется для создания версии этой формы с крупным шрифтом, предназначенной для пользователей с ослабленным зрением. Содержимое этой формы будет генерироваться из XML-файла при помощи XSL-преобразования. Кроме того, рекомендуем применить в этом практикуме какие-нибудь дополнительные данные — это поможет вам закрепить полученные знания и навыки.

Продолжительность практикума — около 30 минут.

Упражнение 1. Создание и подключение таблицы стилей



Используя навыки, полученные в занятии 1, создайте собственную таблицу стилей на основе таблицы стилей по умолчанию, генерированной Visual Studio .NET. Поскольку выбор форматирования (и стилей) — дело вкуса, вы вольны, изменяя предложенное здесь форматирование, создать собственную уникальную книгу. Примерный вид готовой формы показан на рис. 13-20.



Рис, 13-20. Тестовая Web-форма, оформленная с помощью различных стилей

▶ **Как создать пользовательскую таблицу стилей и подключить ее к Web-форме**

1. Внесите в таблицу стилей по умолчанию (Styles.css) изменения, предложенные ниже, либо откорректируйте стили по собственному вкусу:
   - измените шрифт, заданный по умолчанию для всех заголовков, на Comic Sans MS;
   - измените шрифт, заданный по умолчанию для основного текста, на Lucida Sans-Serif;
   - добавьте к странице фоновое изображение (через таблицу стилей);
   - задайте для текста заголовков и линеек такой цвет, чтобы они выделялись на странице.

2. Закончив модификацию таблицы стилей, подключите ее к Web-форме, добавив элемент *link* к элементу *head* Web-формы (см. строку, выделенную полужирным шрифтом в показанном ниже примере):

```
<HEAD>
   <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
   <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
   <meta name="vs_defaultClientScript" content="JavaScript">
   <meta name="vs_targetSchema"
         content="http://schemas.microsoft.com/intellisense/ie5">
   <LINK REL="stylesheet" TYPE="text/css" HREF="Styles.css">
</HEAD>
```

3. Добавьте к Web-форме текст и заголовки из примера, чтобы проверить работу созданной вами таблицы стилей.
4. Для просмотра Web-формы откройте ее в браузере. Если вид Web-формы не устроит вас, скорректируйте таблицу стилей,

Упражнение 2. Создание альтернативных таблиц стилей



В этом упражнении вы должны создать альтернативную таблицу стилей, отображающую заголовки и текст крупным шрифтом, чтобы пользователям с ослабленным зрением было удобно работать с Web-формой. Вы также добавите ссылку с клиентским сценарием, чтобы пользователи могли переключать таблицы стилей во время выполнения. Примерный вид результата показан на рис. 13-21.

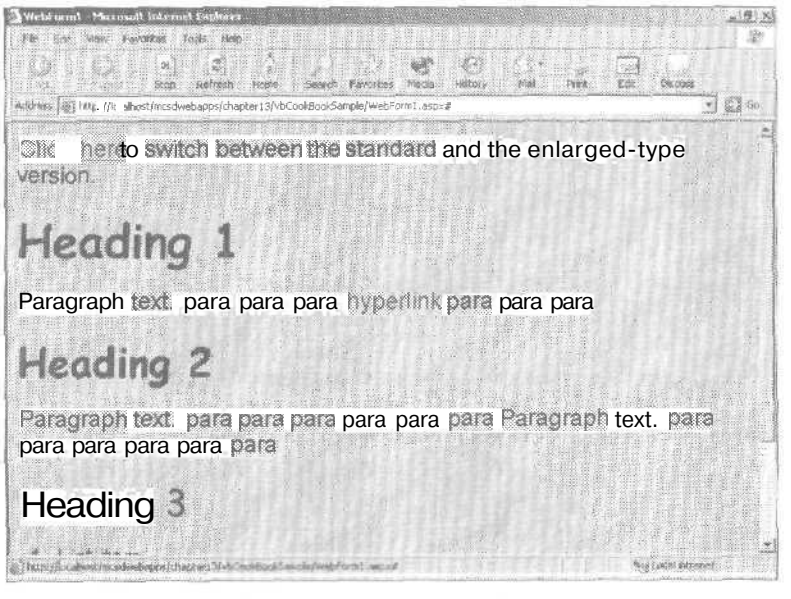


Рис. 13-21. Пример формы с крупным шрифтом

▶ **Как создать таблицу стилей, назначающую элементам формы крупный шрифт**

1. Добавьте к проекту новую пустую таблицу стилей и назовите ее BigType.css.

2. Путем копирования-вставки перенесите в BigType.css содержимое таблицы стилей по умолчанию (Styles.css). Если вы выбрали для проекта язык Visual C#, придется позаимствовать таблицу стилей по умолчанию из какого-нибудь проекта, использующего язык Visual Basic .NET.
3. В таблице стилей BigType.css задайте крупный шрифт для элемента *body*.
4. Добавьте атрибуты *ID* и *media* к элементу *link,* подключающему таблицу стилей по умолчанию к Web-форме, созданной в упражнении 1 (см. атрибуты, выделенные полужирным шрифтом в примере ниже):

```
<LINK ID="ScreenStyle" REL="stylesheet" TYPE="text/css" HREF="Styles.css"
media="screen">
```

5. Добавьте еще один элемент link. Он нужен, чтобы задействовать таблицу стилей по умолчанию при форматировании формы для печати:

```
<LINK ID="PrintStyle" REL="stylesheet" TYPE="text/css" HREF="Styles.css"
media="print">
```

6. Добавьте к Web-форме, созданной в упражнении 1, гиперссылку с событием *onclick* — она будет переключать таблицу стилей по умолчанию и BigType.css:

```
<a onclick="SwitchSheets()" href="#">Click here
to switch between the standard and the enlarged-type version.</a>
```

7. Добавьте следующий клиентский сценарий, переключающий таблицы стилей:

VBScript

```
<scrlpt language="vbscript"> ' Переключение между таблицей
                             ' стилей по умолчанию
                             ' и таблицей стилей, задающей
                             ' крупный шрифт.
Sub SwitchSheets()
   If document.all("ScreenStyle").GetAttribute("HREF", 0) =
"Styles.css" then
         document.all("ScreenStyle").SetAttribute "HREF",
   "BigType.css", 0
   else
         document.all("ScreenStyle").SetAttribute "HREF",
   "Styles.css", 0
   end if
End Sub
</script>
```

JScript

```
<script id="clientEventHandlersJS" language="jscript">
function SwitchSheets()
{
   // Переключение таблиц стилей.
if (document.all["ScreenStyle"].getAttribute("HREF") == "Styles.css")
   document.all["ScreenStyle"].setAttribute ("HREF", "BigType.css", 0);
else
   document.all["ScreenStyle"].setAttribute ("HREF", "Styles.css", 0);
```

```
}
</script>
```

8. Откройте Web-форму в браузере и попробуйте сменить таблицу стилей. При необходимости откорректируйте таблицу стилей BigType.css.

Упражнение 3. Исполнение XSL-преобразования



Сейчас вы добавите к Web-форме, созданной в упражнении 1 и модифицированной в упражнении 2, серверный элемент управления XML. В свойства *DocumentSource* и *TransformSource* добавленного элемента управления необходимо записать соответственно имена XML- и XSL-файлов, созданных в занятии 2 (они есть в примерах к этой книге на прилагаемом компакт-диске). Результат показан на рис. 13-22.

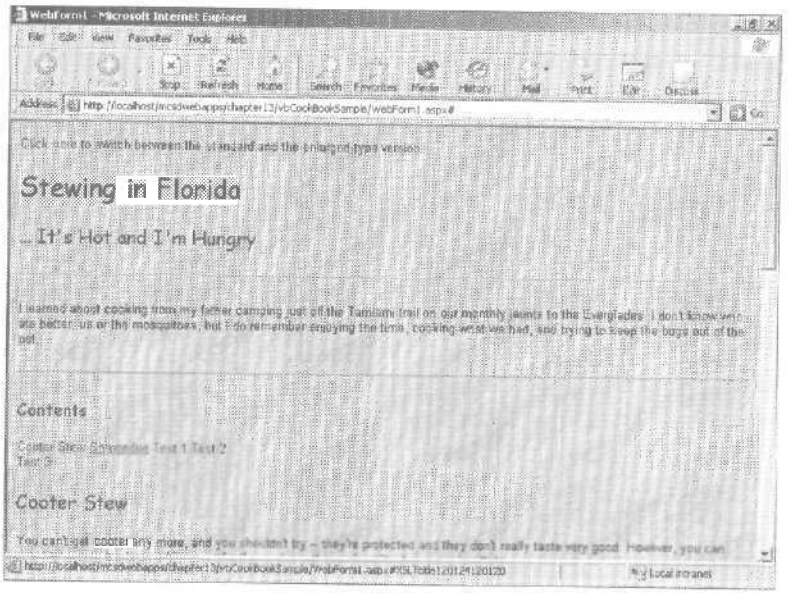


Рис. 13-22. Пример страницы книги рецептов

► **Исполнение XSL-преобразования**

1. Удалите из Web-формы, с которой вы работали в предыдущих упражнениях, текст примера, применявшийся для проверки таблиц стилей; оставьте только гиперссылки и клиентский сценарий, созданные в упражнении 2.
2. После гиперссылки добавьте серверный элемент управления XML следующим образом:

```
<a onclick="SwitchSheets" href="#">Click here to switch between the
standard and the enlarged-type version.</a>
<asp:Xml id="XmlTransform" runat="server" DocumentSource=""
TransformSource="" />
```

3. Скопируйте в каталог проекта файлы cook.xml и cook.xsl.
4. Запишите в свойство *DocumentSource* элемента управления XML строку **cook.xml**, а в свойство *TransformSource* — **cook6.xsl.**
5. Откройте Web-форму в браузере, чтобы проверить работу XML-преобразования.

Упражнение 4. Самостоятельная работа



Подумайте, как применить XML с XSL-преобразованием к данным, для представления которых на вашем Web-сайте в настоящее время используется HTML. Применение XML для разметки содержимого — более абстрактная задача, чем форматирование при помощи HTML-тэгов, поэтому оно требует навыка. Лучший способ научиться писать разметку XML — поработать самостоятельно. Попрактикуйтесь в применении XML на каких-нибудь данных, взятых из реального приложения или с настоящего Web-сайта. Вот примерный план решения этой задачи.

1. Выберите простой проект, например форму для размещения заказа или ввода адреса, и создайте на ее основе XML-файл.
2. Создайте XSL-файл с шаблонами для преобразования XML-файла в HTML.
3. Создайте Web-форму и добавьте к ней элемент управления XML, необходимый для исполнения преобразования.
4. Выполняя упражнение, продумайте ряд вопросов.
   - Как определить иерархию узлов?
   - В чем преимущества иерархических наборов узлов, таких, как этот:

```
<items>
      <item>Item 1</item>
      <item>Item 2</item>
</items>
```

   - Как применить навыки программирования баз данных и написания объектно-ориентированного кода при создании XML-файлов?

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. В чем преимущество использования каскадных таблиц стилей перед локальными стилями при форматировании Web-приложений?
2. Почему лучше создавать стили для форматирования классов, а не отдельных HTML-элементов?
3. Опишите взаимосвязь CSS и XSL при форматировании Web-приложений,
4. Чем отличаются языки HTML и XML?

ГЛАВА 14

Создание справочной системы

Занятие 1. Создание справочной системы в Web-приложениях 590
Занятие 2. Применение HTML Help Workshop 595
Занятие 3. Подключение справочной системы к Web-формам 607

В этой главе

В этой главе рассказано о различных способах отображения справочной информации в Web-приложении, применении HTML Help Workshop для создания компилированных справочных файлов и о том, как сделать эти файлы доступными пользователю Web-приложения.

Прежде всего

Для занятий по материалам этой главы необходимо:

- знание языка HTML;
- навыки создания Web-форм в режиме HTML.

Занятие 1. Создание справочной системы в Web-приложениях

Существует несколько способов отображения справочной информации в Web-приложении: в виде всплывающих подсказок (у элементов управления), на отдельных Web-формах и HTML-страницах, а также при помощи компилированных справочных файлов в формате HTML (HTML Help). На этом занятии вы научитесь применять вышеперечисленные методы с Web-формами.

Изучив материал этого занятия, вы сможете:

- ✓ добавлять всплывающие подсказки к серверным и HTML-элементам управления;
- ✓ отображать справочную информацию на Web-формах и HTML-страницах в браузере;
- ✓ управлять размерами окна браузера, отображающего справочные сведения, а также передачей фокуса в этом окне;
- ✓ отображать справочные сведения при помощи HTML Help Viewer.

Продолжительность занятия — около 15 минут.

Отображение всплывающих подсказок

Всплывающая подсказка (ToolTip) — краткое описание элемента управления, которое появляется, если на секунду задержать над ним указатель мыши. Всплывающие подсказки широко используются в приложениях Windows для отображения полезных сведений о кнопках инструментальных панелей и других графических элементах управления, назначение которых трудно угадать. В некотором смысле всплывающие подсказки выполняют роль пояснительных подписей для не слишком понятных значков (рис. 14-1).

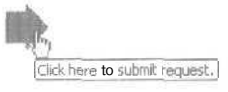


Рис. 14-1. Всплывающая подсказка

Большинство серверных элементов управления поддерживают свойство *ToolTip*, задающее текст всплывающей подсказки. Во время выполнения ASP.NET помещает значение этого свойства в HTML-атрибут *title*, сгенерированный для элемента управления, после чего Internet Explorer отображает содержимое этого атрибута в виде всплывающей подсказки. Но только Internet Explorer интерпретирует значения атрибутов *title* как всплывающие подсказки, другие браузеры их не отображают.

У HTML-элементов управления нет свойства *ToolTip*, поэтому атрибут *title* придется добавлять к ним вручную. Следующий HTML-код демонстрирует, чем отличается определение подсказок для серверных и HTML-элементов управления:

```
<asp:Button id="butOK" runat="server" Text="OK"
  ToolTip="Submits purchase."></asp:Button>
<INPUT type="reset" value="Cancel" title="Cancels purchase.">
```

Однако не у всех элементов управления имеется свойство *ToolTip* или атрибут *title*, например, серверные элементы управления DropDownList и ListBox не поддерживают их.

Отображение справочных сведений на отдельных Web-формах и HTML-страницах

Наверное, проще всего обеспечить пользователя справочной информацией, организовав ее в виде Web-форм или HTML-страниц, размещенных в каталоге Web-приложения либо в каталоге, вложенном в него.

Для отображения справочных сведений в браузере применяются гиперссылки и другие элементы управления. В этом страницы справочной системы не отличаются от обычных страниц, однако одно важное отличие все же есть: как правило, справочные сведения выводят в новом окне браузера или в отдельном фрейме, чтобы отделить их от основного содержимого Web-приложения (рис. 14-2).

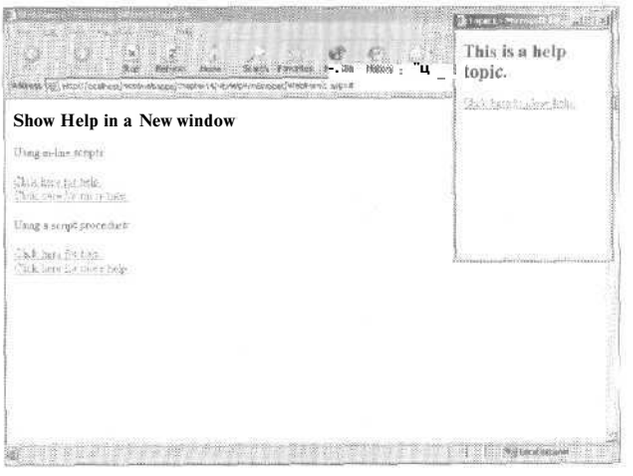


Рис. 14-2. Отображение справочного окна в браузере

Отображение справочных сведений в отдельном окне позволяет пользователю остаться на текущей странице Web-приложения и непосредственно сопоставлять справочные сведения с задачей, над которой он работает в данный момент.

Для отображения справочных сведений в отдельном окне используется метод *open* клиентского объекта *window,* например, так:

```
<a href="#" onclick="window.open('topic1.aspx', 'helpwin').focus()">Click here
for help.</a>
```

Эта гиперссылка открывает раздел справочной системы в новом окне браузера, расположенном поверх других окон Web-приложения, при помощи события *onclick.* Установив атрибут *href* в «#», вы получите ссылку на текущую страницу Web-приложения; такая ссылка не будет направлять пользователя в другое место. Первый аргумент метода *open* задает отображаемый раздел справочной системы, а второй — имя нового окна: это нужно, чтобы открывать в том же окне все разделы справочной системы, которые будут вызваны в дальнейшем. Наконец, метод *focus* отвечает за размещение окна справочной системы поверх остальных окон Web-приложения.

Использование аналогичных сценариев, обрабатывающих событие *onclick,* позволяет гиперссылкам, определенным на разных страницах Web-приложения, открывать другие разделы справочной системы в том же самом окне. Если в приложении есть формы, содержащие несколько ссылок на страницы справочной системы, разумно поместить такой сценарий в процедуру и вызывать ее из обработчика события *onclick:*

VBScript

```
<HTML>
   <HEAD>
   <title>WebForm1</title>
<script language="vbscript">
' Открыть Web-форму или HTML-страницу в небольшом окне браузера
' и передать этому окну фокус.
Sub ShowHelp(topicName)
   Dim HeipWindow
   ' Отобразить раздел в именованном окне.
   Set HeipWindow = window.open(topicName,
   "helpwin","left=600,height=300,width=200")
   ' Расположить это окно поверх других окон.
   HelpWindow.focus
End Sub
</script>
   </HEAD>
   <body>
         <h2>Show Help in a New window</h2>
         <A onclick="ShowHelp('topic1.aspx')" href="#">Click here
         for help.</A>
         <BR>
         <A onclick="ShowHelp('topic2.aspx')" href="#">Click here
         for more help.</A>
   </body>
</HTML>
```

JScript

```
<HTML>
   <HEAD>
<script language=jscript>
// Открыть Web-форму или HTML-страницу в небольшом окне браузера
// и передать этому окну фокус.
function ShowHelp(topicName)
{
   var HeipWindow;
   // Отобразить раздел в именованном окне.
   HeipWindow = window.open(topicName,
     "helpwin","left=600,height=300,width=200");
   // Отобразить поверх других окон.
   HelpWindow.focus();
}
</script>
   </HEAD>
   <body>
         <h2>Show Help in a New window</h2>
```

```
        <a href="#" onclick="ShowHelp('topic1.aspx')">Click here for
        help.</a>
        <br>
        <a href="#" onclick="ShowHelp('topic2.aspx')">Click here for
        more help.</a>
    </body>
</HTML>
```

Метод *dose* объекта *window* позволяет создать в окне, отображающем раздел справочной системы, ссылку, которая будет закрывать это окно:

```
<a href="#" onclick="window.close()">Click here to close help.</a>
```

Есть и другие способы отображения разделов справочной системы в именованных окнах, например при помощи атрибута *target* в гиперссылке, но при этом труднее располагать окно справочной системы поверх остальных окон Web-приложения.

Отображение справочной информации в формате HTML Help

Метод *showHelp* объекта *window* позволяет отображать справочные файлы в формате HTML (HTML Help), а также обычные HTML-страницы и Web-формы при помощи утилиты HTML Help Viewer (hh.exe). Файл HTML Help состоит из нескольких файлов разделов, написанных на языке HTML и скомпилированных в единый сжатый файл, поддерживающий ряд дополнительных возможностей, о которых мы поговорим далее.

Метод *showHelp* отображает содержимое справочной системы в окне программы HTML Help Viewer. По умолчанию оно располагается поверх окна браузера, отображающего Web-приложение (рис. 14-3).

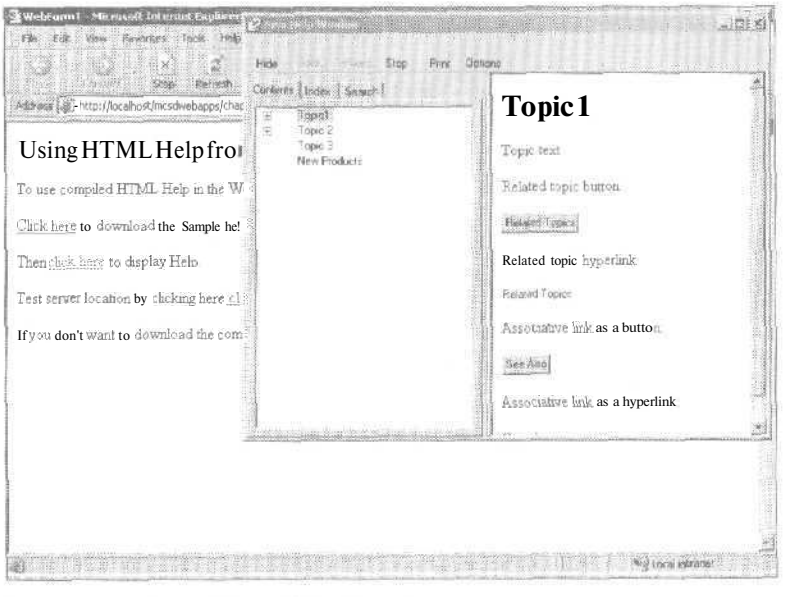


Рис. 14-3. Окно HTML Help Viewer

Чтобы открыть Web-форму или HTML-страницу при помощи метода *showHelp* при вызове передайте ему в первом аргументе имя файла этой формы (страницы):

```
<a href="#" onclick="window.showHelp('Topic1.aspx')">Show Help</a>
```

Этот HTML-код отображает Web-форму Topic1.aspx по щелчку гиперссылки Show Help. Файл Topic1.aspx не является компилированным справочным файлом, это обычный раздел справочной системы, написанный при помощи ASP.NET. Применение метода *showHelp* гарантирует отображение окна справочной системы поверх окна Web-приложения, даже если последнее обладает фокусом ввода.

Метод *showHelp* также позволяет отображать разделы компилированных справочных файлов. Такие файлы (с расширением .chm) состоят из нескольких HTML-страниц, скомпилированных в единый файл при помощи HTML Help Workshop. При работе с компилированными справочными файлами метод *showHelp* определяет отображаемый раздел при помощи идентификатора контекста, который сопоставляется с именем файла раздела во время компиляции справочного файла.

Чтобы отобразить раздел компилированного справочного файла в формате HTML, передайте методу *showHelp* в качестве первого аргумента имя компилированного файла и файла раздела, как показано ниже:

```
<a href="#" onclick="window.showHelp('c:\\Help\\HelpSample.chm')">
Show Help</a><br>
```

Этот HTML-код открывает компилированный файл HelpSample.chm. Обратите внимание, что перед использованием компилированный справочный файл необходимо загрузить на компьютер пользователя и сохранить там. Применение компилированных файлов имеет ряд преимуществ по сравнению с использованием для хранения и отображения справочной информации отдельных Web-форм или HTML-страниц:

- меньший **размер** справочных **файлов.** При компиляции выполняется сжатие разделов, поэтому размер компилированного файла много меньше суммарного размера его исходных файлов;
- **наличие** инструментов для **создания оглавления, указателя и средств поиска.** HTML Help Workshop включает инструменты для реализации указанных возможностей в справочной системе приложения;
- **внедрение справочных файлов с целью их отображения.** При помощи Java-апплета и ActiveX-элемента управления HTML Help можно отображать содержимое справочной системы в одном окне с приложением, а не в отдельном окне браузера;
- **простота локализации.** Поскольку HTML Help сопоставляет имена файлов разделов идентификаторам контекста, в приложении может быть несколько компилированных справочных файлов на разных языках (русском, английском и др.), использующих общий набор идентификаторов контекста.

Однако у компилированных справочных файлов есть и ограничения:

- **исходные файлы должны быть в формате HTML** — например, Web-формы нельзя скомпилировать в .chm-файлы;
- **компилированный** файл **необходимо загружать на пользовательский компьютер целиком.** Если справочный файл велик, а пользователь подключен через модем, то при первом вызове справочного файла ему придется долго ждать его загрузки, зато потом этот файл будет открываться намного быстрее.

Создание компилированных справочных файлов в формате HTML и их применение с Web-приложениями более подробно описано в следующем занятии.

Занятие 2. Применение HTML Help Workshop

Утилита HTML Help Workshop входит в состав Microsoft Visual Studio .NET и устанавливается на компьютере разработчика в отдельный каталог. Освоив материал этого занятия, вы научитесь применять HTML Help Workshop для создания компилированных справочных файлов с оглавлением, предметным указателем и возможностью полнотекстового поиска.

Изучив материал этого занятия, вы сможете:

- ✓ создать проект справочной системы;
- ✓ **добавить к этому** проекту разделы **и скомпилировать** их в единый справочный файл;
- ✓ автоматически генерировать таблицу оглавления во время компиляции;
- ✓ модифицировать автоматически собранное оглавление;
- ✓ определять в разделах справочного файла ключевые слова, индексируемые в предметном указателе;
- ✓ создавать различные типы перекрестных ссылок между разделами справочного файла;
- ✓ дополнять разделы справочных файлов специальными возможностями, такими, как ассоциативные ссылки и ссылки на смежные разделы (Related Topics);
- ✓ реализовать полнотекстовый поиск в справочном файле;
- ✓ компилировать справочный файл и открывать его для предварительного просмотра при помощи интерфейса HTML Help Workshop или утилиты командной строки.

Продолжительность занятия — около 30 минут.

Создание справочной системы в формате HTML Help

Справочная система HTML Help состоит из отдельных HTML-страниц, представляющих разделы справочной системы. В любом разделе справочной системы могут быть ключевые слова, индексируемые в предметном указателе, а также перекрестные ссылки на другие разделы. Проект HTML Help включают файлы, описанные в таблице 14-1.

HTML Help Workshop предоставляет инструменты для упорядочивания, редактирования и компиляции файлов, составляющих проект справочной системы. По умолчанию Visual Studio .NET записывает HTML Help Workshop в каталог Program Files\HTML Help Workshop.

Таблица 14-1. Файлы проекта справочной системы HTML Help

Тип файла	Назначение
Файл раздела (.htm)	Эти файлы составляют содержимое справочной системы
Файл проекта (.hhp)	Определяет, как и из каких разделов компилируется проект, а также файлы оглавления и предметного указателя, которые необходимо включить в проект
Файл оглавления (.hhc)	Создает оглавление справочного файла
Предметный указатель (.hhk)	Создает предметный указатель справочного файла
Список исключенных слов (.stp)	Содержит слова, исключаемые из полнотекстового поиска
Компилированный справочный файл (.chm)	Служит для развертывания готовой справочной системы

Чтобы создать проект справочной системы HTML Help, выполните следующие действия.

1. В окне HTML Help Workshop выберите в меню File команду New, запускающую мастер New Project Wizard.
2. Этот мастер позволят создать справочную систему, преобразовав существующий проект WinHelp, либо создать новый проект справочной системы HTML Help. Чтобы создать новый проект, щелкните Next — мастер откроет страницу Destination.
3. Введите имя файла проекта справочной системы, который вы собираетесь создать, и определите каталог, где он будет размещаться, затем щелкните Next — мастер откроет страницу Existing Files.
4. Пометьте флажками все имеющиеся файлы оглавления, предметного указателя и разделов, которые следует включить в новый справочный файл, и щелкните Next. Если на этом этапе вы выбрали какие-либо файлы, мастер попросит указать размещение файлов каждого типа, и противном случае он сразу создаст пустой проект справочной системы (рис. 14-4).

Интерфейс HTML Help Workshop довольно прост: вкладки Project, Contents и Index в левой панели окна позволяют редактировать соответственно содержимое проекта, файлы оглавления и предметного указателя. Кнопки, расположенные наверху панели, создают новые и открывают существующие файлы, компилируют проект, отображают скомпилированный проект и вызывают справочную систему HTML Workshop.

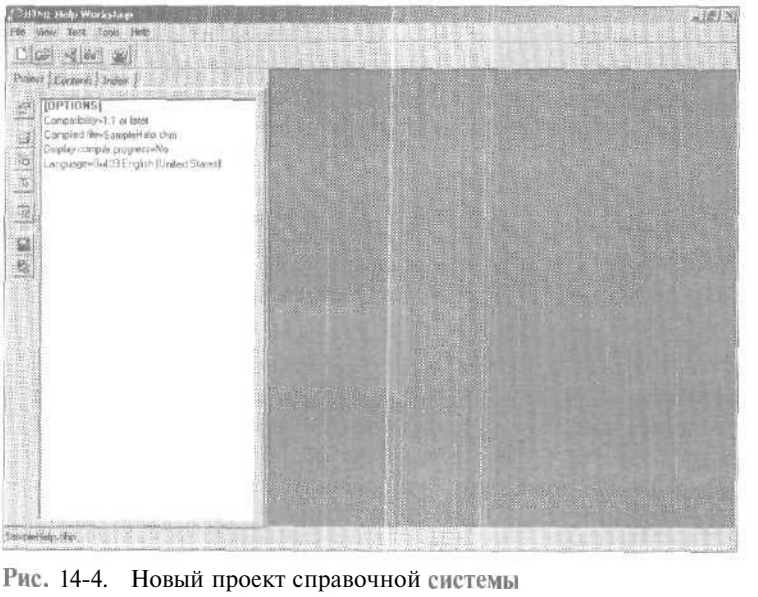


Рис. 14-4. Новый проект справочной системы

Чтобы добавить к проекту справочной системы файлы разделов, выполните следующие действия.

1. Выберите в меню New элемент HTML File и введите имя создаваемого раздела — Workshop создаст новый файл раздела справочной системы (рис, 14-5).
2. Сохраните HTML-файл и щелкните Add/Remove Topic Files на вкладке Project, чтобы открыть диалоговое окно Topic Files.
3. Щелкните кнопку Add, выберите файлы, которые нужно добавить к проекту, и щелкните OK.

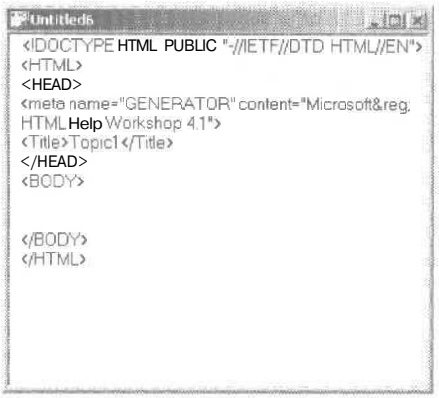


Рис. 14-5. Создание файла разделов

4. Снова щелкните OK, чтобы закрыть диалоговое окно Topic Files и добавить выбранные файлы к проекту.

HTML Help Workshop поддерживает лишь базовые возможности редактирования текстов, поэтому лучше создавать файлы разделов в полнофункциональном HTML-редакторе, например в Microsoft FrontPage, а HTML Help Workshop применять только для создания файла проекта и компиляции разделов.

Установка параметров проекта

Параметры проекта справочной системы HTML Help Workshop отображает на вкладке Project. Эти параметры хранятся в файле проекта (в .hhp-файле) — обычном текстовом файле, который разрешается редактировать в HTML Help Workshop или любом текстовом редакторе (например, в Блокноте).

Чтобы задать параметры проекта в HTML Help Workshop, щелкните Project Options на вкладке Project — откроется диалоговое окно Project Options, показанное на рис. 14-6.

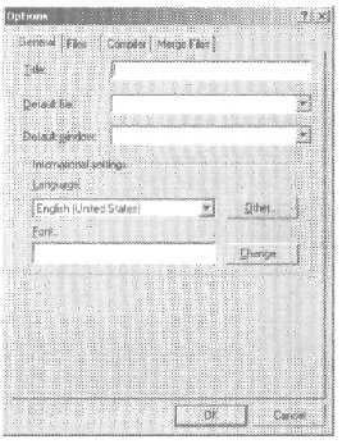


Рис. 14-6. Установка параметров проекта

В нем можно задать название и региональные параметры проекта, выбрать способ создания файлов предметного указателя и оглавления, а также указать, нужно ли поддерживать поиск и ряд других возможностей. Соответствующие элементы располагаются в сек-

ции OPTIONS файла проекта. К примеру, следующие параметры проекта задают автоматическое создание файлов предметного указателя и оглавления во время компиляции справочного файла HelpSample.chm, отображающего в окне справочной системы строку «HTML Help Sample»:

```
[OPTIONS]
Auto Index=Yes
Auto TOC=9
Compatibility=1.1 or later
Compiled file=HelpSample.chm
Contents file= Contents.hhc
Default topic=topic1.htm
Display compile progress=No
Full text search stop list file=Search.stp
Index file=Index.hhk
Language=0x409 English (United States)
Title=HTML Help Sample
```

Файл проекта также содержит секцию FILES со списком компилируемых файлов разделов. Если в секции OPTIONS задан параметр *Auto TOC*, то порядок файлов в этой секции будет определять порядок разделов в файле оглавления. Например, перечисленные ниже файлы разделов представлены в оглавлении в следующем порядке:

```
[FILES]
topic1.htm
topic1a.htm
topic1b.htm
topic2.htm
topic3.htm
```

Создание оглавления

При автоматической генерации оглавления HTML Help Workshop руководствуется уровнями заголовков разделов и порядком файлов разделов в секции FILES файла проекта.

Заголовки, помеченные тэгами *<h1>*, помещаются на верхний уровень; заголовки, помеченные тэгами *<h2>*, — уровнем ниже и так далее, пока вновь не встретится заголовок с тэгами *<h1>*. Если в разделах определены заголовки нескольких уровней, HTML Help Workshop создаст подчиненные элементы оглавления для каждого из них, даже если это вызывает дублирование элементов. Поэтому при автоматической генерации оглавления лучше использовать не более одного HTML-элемента заголовка на раздел.

Следующая секция FILES содержит файлы разделов с заголовками трех уровней, о чем свидетельствуют их имена файлов:

```
[FILES]
topic1.htm
topida.htm
topic1b.htm
topic2.htm
topic2a.htm
topic2b.htm
topic2bi.htm
```

```
topic2bii.htm
topic3.htm
```

Файл оглавления, сгенерированный на основе этих файлов, во время выполнения принимает следующий вид (рис. 14-7).

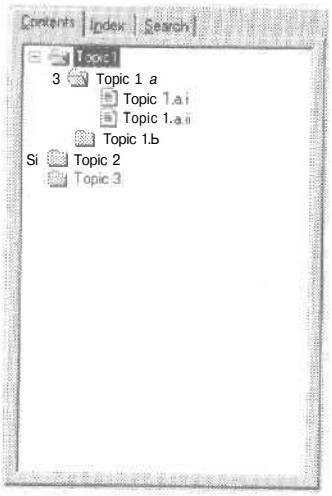


Рис. 14-7. Автоматически собранное оглавление

HTML Help Workshop сохраняет сгенерированное оглавление в HTML-файле. Можно создать оглавление по своему вкусу, для этого возьмите автоматически собранное оглавление, отключите параметр Auto TOC в файле проекта и отредактируйте полученный файл.

Чтобы отредактировать файл оглавления в HTML Help Workshop, шелкните вкладку Contents — откроется оглавление (рис. 14-8).

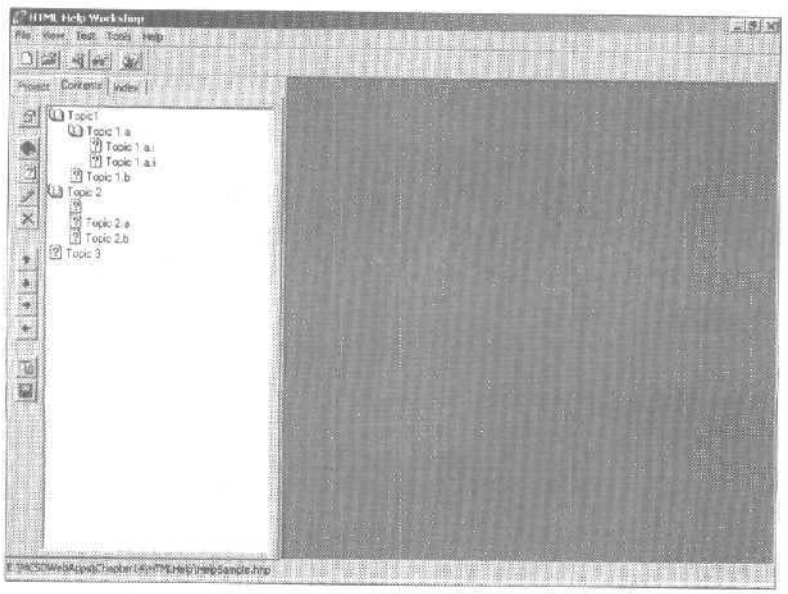


Рис. 14-8. Редактирование файла оглавления

Нетрудно заметить, что в автоматически сгенерированном оглавлении, показанном на рис. 14-8, используются разные значки для разделов верхнего уровня: вместо значка в виде книги, обозначающего заголовок первого уровня, третьему разделу назначен значок в виде страницы.

Чтобы изменить значок раздела, выполните следующие действия.

1. Выделите в оглавлении нужный раздел и щелкните Edit Selection, затем перейдите на вкладку Advanced — откроется диалоговое окно Table Of Contents Entry (рис. 14-9).
2. Выберите из списка Image Index изображение, которое следует использовать в качестве значка для данного элемента оглавления, и щелкните OK.

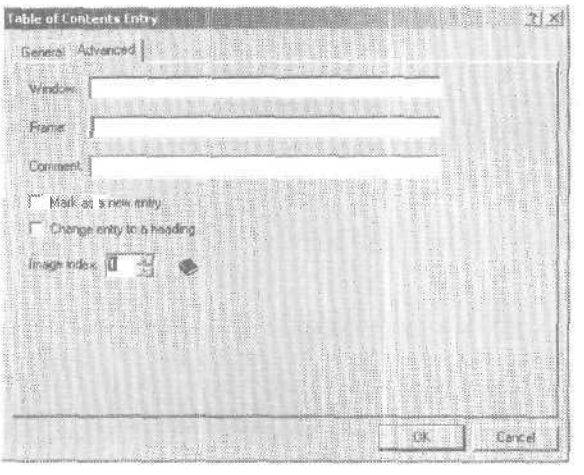


Рис. 14-9. Изменение значка элемента оглавления

Файл оглавления хранится в формате HTML, элементы оглавления в нем заданы элементами OBJECT. Вот пример HTML-файла оглавления:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft&reg; HTML Help Workshop 4.1">
<!- Sitemap 1.0 ->
</HEAD><BODY>
<UL>
   <LI> <OBJECTtype="text/sitemap">
          <param name="Name" value="Topic1">
          <param name="Local"value="topic1.htm">
          </OBJECT>
   <UL>
          <LI> <OBJECT type="text/sitemap">
               <param name="Name" value="Topic 1.a">
               <param name="Local" value="topic1a.htm">
               </OBJECT>
          <UL>
               <LI> <OBJECT type="text/sitemap">
                    <param name="Name" value="Topic 1.a.i">
                    <param name="Local" value="topic1ai.htm">
```

```
                    </OBJECT>
                <LI> <08JECT type="text/sitemap">
                    <param name="Name" value="Topic 1.a.ii">
                    <param name="Local" value="topic1aii.htm">
                    </OBJECT>
        </UL>
        <LI> <OBJECT type="text/sitemap">
            <param name="Name" value="Topic 1.b">
            <param name="Local" value="topic1b.htm">
            <param name="ImageNumber" value="1">
            </OBJECT>
    </UL>
    <LI> <OBJECT type="text/sitemap">
        <param name="Name" value="Topic 2">
        <param name="Local" value="topic2.htm">
        </OBJECT>
    <UL>
        <LI> <OBJECTtype="text/sitemap">
            <param name="Name" value="">
            <param name="Local" value="topic2.htm">
            </OBJECT>
        <LI> <OBJECTtype="text/sitemap">
            <param name="Name" value="Topic 2.a">
            <param name="Local" value="topic2a.htm">
            </OBJECT>
         <LI> <OBJECT type="text/sitemap">
            <param name="Name" value="Topic 2.b">
            <param name="Local" value="topic2b.htm">
            </OBJECT>
    </UL>
    <LI> <OBJECT type="text/sitemap">
        <param name="Name" value="Topic 3">
        <param name="Local" value="topic3.htm">
        <param name="ImageNumber" value="1">
        </OBJECT>
</UL>
</BODY></HTML>
```

Иерархия элементов оглавления определяется при помощи HTML-элементов списков, а элементы OBJECT задают заголовок, значок, адрес и другие сведения, Например, следующий элемент оглавления включает ссылку на страницу компании New Products и по щелчку пользователя открывает эту страницу в новом окне:

```
<OBJECT type="text/sitemap">
    <param name="Name" value="New Products">
    <param name="Local" value="http://www.mycompany.com/newproducts.htm">
    <param name="WindowName" value="NewWindow">
    <param name="Comment" value="Sample of text/sitemap params">
    <param name="New" value="1">
```

```
   <param name="ImageNumber" value="7">
</OBJECT>
```

Добавление элементов указателя

HTML Help Workshop может автоматически генерировать файл предметного указателя на основе элементов **OBJECT,** определенных в файлах разделов. Элементы указателя в файле раздела создаются в виде ActiveX-объектов с заданным *classID*, например:

```
<OBJECT type="application/x-oleobject" classid="clsid:1e2a7bd0-dab9-11d0-b93a-
00c04fc99f9e">
   <param name="Keyword" value="Index Item 1">
   <param name="Keyword" value="Index Item 1, Subitem a">
   <param name="Keyword" value="Index Item 1, Subitem b">
</OBJECT>
```

Этот HTML-код создает три элемента указателя для ссылки на файл раздела (рис. 14-10).



Рис. 14-10. Элементы указателя, автоматически сгенерированные во время выполнения

HTML Help Workshop не записывает автоматически в файл указателя элементы указателя, определенные в файлах разделов. Однако вы можете вручную определять элементы в файле указателя. Это позволяет упорядочить оглавление или создать дополнительные элементы, помимо генерируемых автоматически. Во время генерации предметного указателя содержимое .hhk-файла и элементы указателя, определенные в разделах, объединяются. К примеру, в результате объединения следующего файла указателя с элементами, генерируемыми автоматически, получится указатель, показанный на рис. 14-11.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft&reg; HTML Help Workshop 4.1">
<!- Sitemap 1.0 ->
</HEAD><BODY>
<UL>
   <LI> <OBJECT type="text/sitemap">
         <param name="Name" value="Index Item 1">
         <param name="Name" value="Topic 1.b">
```

```
        <param name="Local" value="topic1b.htm">
        </OBJECT>
    <LI> <OBJECT type="text/sitemap">
        <param name="Name" value="Index Item 2">
        <param name="Name" value="Topic 2.a">
        <param name="Local" value="topic2a.htm">
        </OBJECT>
</UL>
</BODY></HTML>
```

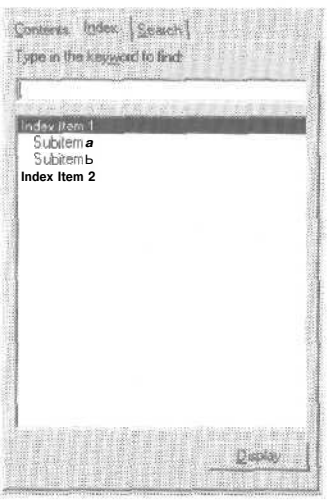


Рис. 14-11. Указатель, полученный в результате объединения элементов

Добавление перекрестных ссылок

Существует три способа добавления перекрестных ссылок на разделы **справочной** системы **HTML Help:**

- при помощи стандартных гиперссылок HTML — они создают связи типа «один к одному»;
- с помощью команды Related Topics — эта команда позволяет создать ссылку на список разделов;
- **с применением ассоциативных ссылок** (*ALink*) между **разделами на основе определенных в них ключевых слов.**

Как пользоваться гиперссылками, **вы уже знаете, поэтом в следующих разделах** описаны два других **метода.**

Добавление ссылок на смежные разделы

Ссылки на смежные разделы добавляют при помощи перекрестных ссылок либо гиперссылок.

- Чтобы поместить перекрестную ссылку Related Topics в раздел справочной системы, добавьте к его файлу ActiveX-элемент управления HTML Help. Например, следующий элемент OBJECT отображает кнопку, всплывающее меню которой содержит ссылки на два смежных раздела:

```
<OBJECT id=hhctrl type="application/x-oleobject"
        classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11"
        codebase="hhctrl.ocx#Version=4,74,9273,0"
        width=100
        height=100>
    <PARAM name="Command" value="Related Topics, MENU">
    <PARAM name="Button" value="Text:Related Topics">
    <PARAM name="Item1" value="Topic 2;Topic2.htm">
    <PARAM name="Item2" value="Topic 3;Topic3.htm">
</OBJECT>
```

Вот как выглядит эта кнопка во время выполнения (рис. 14-12).



Рис. 14-12. Ссылки на смежные разделы

- Чтобы ссылка Related Topics отображалась как гиперссылка, а не элемент управления Button, замените параметр *Button* в элементе OBJECT параметром *Text*. Ниже показан пример HTML-текста, отображающий такую же ссылку на смежные разделы, но в виде гиперссылки:

```
<OBJECT id- hhctrl type="application/x-oleobject"
        classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11"
        codebase="hhctrl.ocx#Version=4,74,9273,0"
        width=100
        height=100>
    <PARAM name="Command" value="Related Topics, MENU">
    <PARAM name="Text" value="Text:Related Topics">
    <PARAM name="Item1" value="Topic 2;Topic2.htm">
    <PARAM name="Item2" value="Topic 3;Topic3.htm">
</OBJECT>
```

Создание ассоциативных ссылок

Ассоциативные ссылки связывают ключевые слова, а не файлы разделов. Эта особенность делает их удобными для создания перекрестных ссылок между файлами справочной системы HTML Help, поскольку для создания таких ссылок не обязательно знать названия разделов компилированного файла, достаточно помнить нужные ключевые слова и их размещение. Ассоциативные ссылки удобно применять и внутри справочных файлов: их имена меняются в ходе разработки, поэтому проще оперировать списком ключевых слов, нежели постоянно редактировать имена файлов в перекрестных ссылках.

Ассоциативная ссылка состоит из двух частей:

- ключевого слова, заданного в файлах целевых разделов и определяющего разделы, связанные перекрестной ссылкой;
- ссылки, определенной в файле ссылающегося раздела, который будет источником ссылки.

Чтобы определить ключевые слова для ассоциативной ссылки, необходимо вставить в целевой раздел элемент OBJECT. Следующий элемент OBJECT создает ключевое слово «Link1»:

```
<Object type="application/x-oleobject" classid="clsid:1e2a7bd0-dab9-11d0-b93a-
00c04fc99f9e">
   <param name="ALink Name" value="Link1">
</OBJECT>
```

Ассоциативные ссылки весьма напоминают ключевые слова, используемые при составлении указателя для справочной системы в формате **HTML,** но никогда не появляются в предметном указателе — это позволяет независимо пользоваться предметным указателем и перекрестными ссылками.

Чтобы создать ссылку на раздел с некоторым ключевым словом, необходимо вставить ActiveX-объект элемента управления HTML Help в файл раздела, который будет источником ссылки. Следующий элемент OBJECT в ответ на щелчок отображает всплывающее меню со списком ссылок:

```
<OBJECT id=hhctrl type="application/x-oleobject"
        classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11"
        codebase="hhctrl.ocx#Version=4,74,9273,0"
        width=100
        height=100>
    <PARAM name="Command" value="ALink, menu">
    <PARAM name="Button" value="Text:See Also">
    <PARAM name="Flags" value=",,1">
    <PARAM name="Item1" value="">
    <PARAM name="Item2" value="Link1">
</OBJECT>
```

Во время выполнения этот HTML-код будет отображать список разделов с ключевым словом «Link1» (рис. 14-13),



Рис. 14-13. Ассоциативные ссылки

Чтобы ассоциативные ссылки отображались в виде гиперссылок, а не кнопок, следует опустить параметр *Button в* определении элемента *object* и вставить в гиперссылку вызов события *Click* объекта. Например, следующий HTML-код отображает список ссылок по щелчку гиперссылки See Also:

```
<OBJECT id=hhctrl type="application/x-oleobject"
        classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11"
        codebase="hhctrl.ocx#Version=4,74,9273,0"
        width=100
        height=100>
    <PARAM name="Command" value="ALink, menu">
    <PARAM name="Flags" value=",,1">
    <PARAM name="Item1" value="">
    <PARAM name="Item2" value="Link1">
</08JECT>

<a href="#" onclick="hhctrl.Click()">See also</a>
```

Организация поддержки поиска

Чтобы организовать поддержку полнотекстового поиска в справочном файле, перейдите на вкладку Compiler диалогового окна Project Options и щелкните Compile Full-Text Search Information.

При этом HTML Help Workshop построит таблицу конкорданса для всех слов, встречающихся в файлах разделов, и добавит ее к компилированному справочному файлу. Подобная таблица содержит ссылки на все слова раздела, поэтому она может быть довольно велика. Можно контролировать размеры таблицы, опуская слова, заведомо не нужные для поиска, такие, как артикли, союзы, личные местоимения и числительные.

Чтобы исключить некоторые слова из полнотекстового поиска, выполните следующие действия.

1. Создайте текстовый файл со списком слов, которые следует исключить из поиска.
2. На вкладке Project в окне HTML Help Workshop щелкните Change Project Options.
3. В диалоговом окне Project Options на вкладке Compiler введите в текстовое поле Full Text Search Stop List имя созданного файла с исключаемыми словами.

Если пользователь попытается выполнить поиск по слову из этого списка, справочная система сообщит, что поиск по данному слову невозможен.

Компиляция и предварительный просмотр справочной системы

Скомпилировать проект справочной системы HTML Help можно двумя способами:

1. в HTML Help Workshop перейдите на вкладку Project и щелкните Save All Files And Compile;
2. воспользуйтесь утилитой командной строки HTML Help Compiler (hhc.exe). Например, скомпилировать проект HelpSample.hhp в командной строке можно так:

```
hhc.exe HelpSample.hhp
```

Для просмотра скомпилированного проекта также предусмотрены два способа:

1. щелкните кнопку View Compiled File, расположенную наверху окна HTML Help Workshop, и укажите имя компилированного файла для просмотра;
2. в Windows дважды щелкните значок компилированного файла справочной системы (.chm-файла).

Внимание! Пока компилированный справочный файл открыт для просмотра, его нельзя перекомпилировать. Чтобы сделать это, прежде необходимо закрыть файл.

Занятие 3. Подключение справочной системы к Web-формам

Из этого занятия вы узнаете, как вызывать справочную систему из Web-форм. Здесь материал занятия 1 дополнен демонстрацией различных способов отображения справочных сведений в формате HTML Help, а также описанием различий между компилированными и некомпилированными справочными файлами. В этом занятии также разъясняется, как заставить справочную систему реагировать на действия пользователя в Web-приложении.

Изучив материал этого занятия, вы сможете:

- ✓ отображать компилированные и некомпилированные справочные файлы при помощи HTML Help Viewer;
- V открывать разделы HTML Help в обычном окне браузера;
- ✓ отображать различные разделы справочной системы в зависимости от элемента управления, обладающего фокусом.

Продолжительность занятия — около 10 минут.

Применение HTML Help Viewer

Как говорилось в занятии I, метод *showHelp* позволяет отображать справочные файлы в HTML Help Viewer. Процедура вызова метода *showHelp* зависит от того, компилирован ли справочный файл:

- если файл не компилирован, его можно открыть через Интернет, указав адрес раздела:

```
<a href="#" onclick="window.showHelp('Topic1.htm')">Help!</a>
```

- перед вызовом метода *showHelp* для отображения разделов компилированного справочного файла необходимо целиком загрузить этот файл на локальный компьютер:

```
<a href="#"
onclick="window.showHelp('c:\\Help\\HelpSample.chm')">Help!</a>
```

HTML Help Viewer по-разному отображает компилированные и некомпилированные справочные файлы (рис. 14-14).

Как показано на рис. 14-14, для компилированного справочного файла отображаются вкладки с оглавлением, предметным указателем и поиском, а для некомпилированного справочного файла — только запрошенный раздел. При применении компилированных справочных файлов пользователь получает максимум возможностей справочной системы, но вынужден сначала целиком загрузить справочный файл на свой компьютер.

Чтобы загрузить справочный файл на пользовательский компьютер создайте гиперссылку, указывающую на компилированный файл, и предоставьте пользователю инструкции по сохранению этого файла. Например, следующий HTML создает гиперссылку, которая загружает файл HelpSample.chm и поясняет, в каком каталоге его сохранить:

```
<з href="HelpSample.chm">Click hereto download the sample Help file and
then save the file to the C:\Help folder on your computer.</a>
```

В ответ на щелчок этой гиперссылки браузер спросит у пользователя, следует ли открыть справочный файл или нужно сохранить его на диске. После сохранения файла в *указанном* каталоге метод *showHelp* сможет отображать разделы этого файла.

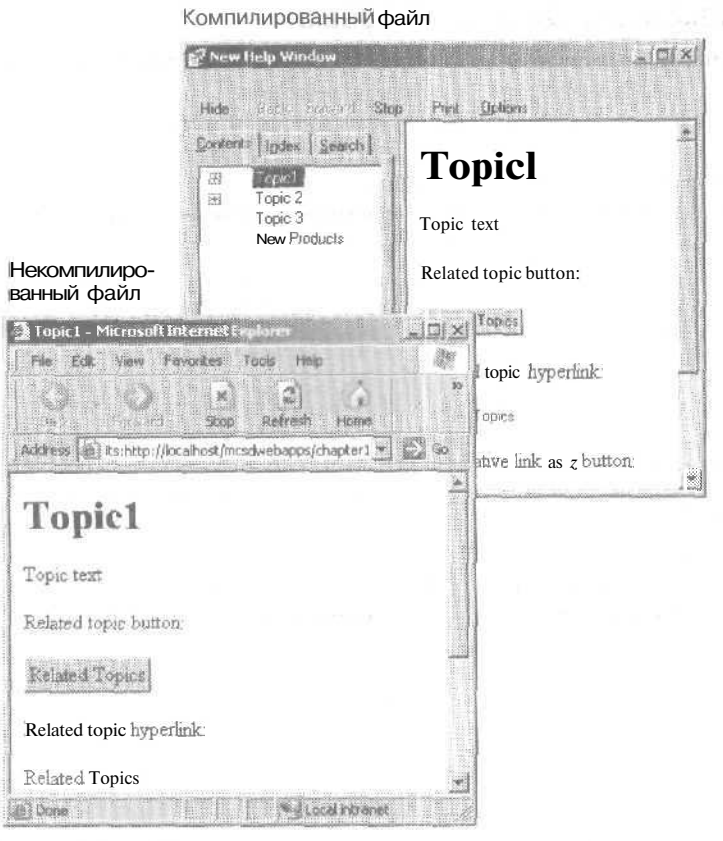


Рис. 14-14. Компилированный и некомпилированный справочные файлы

Отображение справочных файлов в браузере

Метод *showHelp* — не единственный для отображения компилированных справочных файлов; разделы справочного файла можно отображать в браузере, вызывая их из самого справочного файла посредством URL. Естественно, браузер не отобразит оглавление, указатель или страницу поиска, но зато он откроет раздел компилированного файла через Интернет, а у метода *showHelp* это не получалось!

Чтобы отобразить компилированный файл в браузере, создайте гиперссылку на URL раздела в компилированном файле. Например, следующая гиперссылка откроет Topic1.htm в новом окне браузера:

```
<a href="ms-its:http://www.mycompany.com/Help/HelpSample.chm::/topic1.htm"
target="HelpWin">Show Help</a>
```

Для отображения раздела компилированного справочного файла, хранимого в Web, эта гиперссылка применяет подключаемый протокол *ms-its:* и синтаксис указания пути к разделу справочной системы. Протокол *ms-its:* поддерживается Internet Explorer, начиная с версии 4.0. Синтаксис указания пути к разделу справочного файла HTML Help имеет следующий вид: *compiledfile::/topicfile*где *compiledfile*— URL компилированного справочного файла, а *topicfile*— имя файла раздела из этого файла справки,

Отображение контекстно-зависимой справки

При отображении компилированных справочных файлов при помощи HTML Help Viewer идентификаторы контекста применяются для обозначения разделов справочного файла и сопоставления их задачам в Web-приложении. Такой тип справочной информации называется *контекстно-зависимой*, поскольку в зависимости от текущего контекста (или задачи) отображаются различные разделы справочного файла.

Отображение контекстно-зависимой справочной информации сопряжено с рядом трудностей:

- нажатие клавиши, вызывающей справочную систему (F1), можно перехватывать при помощи события *onkeydown* и свойства *window.event.keyCode*, но нельзя отменить действие, выполняемое Internet Explorer по умолчанию — вызов его собственной справочной системы по нажатию FI;
- аргумент *contextID* метода *showHelp* не работает при вызове этого метода из Web-формы или HTML-страницы.

Чтобы обойти эти трудности, придется выбрать другой механизм вызова справочной системы в Web-приложении. Кроме того, для вызова необходимых разделов справочной системы следует применять синтаксис указания пути к разделу справочной системы на основе HTML. Ниже показан пример HTML-кода, использующего событие *onfocus* для отображения различных разделов справочной системы в зависимости от элемента управления Web-формы, с которым работает пользователь в данный момент.

VBScript

```
<HTML>
   <HEAD>
   <title>Context Help</title>
<script language="vbscript">
Dim LastTopic

Sub ShowContextHelp(Topic)
   If (document.all("chkHelp").checked And LastTopic <> Topic) Then
         ' Отобразить справочные сведения для этого контекста.
         window.showHelp "c:\help\HelpSample.chm::/" & Topic
         ' Сохранить имя раздела.
         LastTopic = Topic
   End If
End Sub
</script>
   </HEAD>
   <body onfocus="ShowContextHelp('Topic1.htm')>
         <form id="WebForm2" method="post" runat="server">
               <H2>Context Sensitive Help</H2>
               <INPUT onfocus="ShowContextHelp('Topic2.htm')"
               type="text">
               <BR>
               <INPUT onfocus="ShowContextHelp('Topic3.htm')"
               type="button"
               value="Button">
```

```
            <BR>
            <INPUT id="chkHelp" type="checkbox" name="chkHelp">Show user
            assistance.
        </form>
    </body>
</HTML>
```

JScript

```
<HTML>
    <HEAD>
        <title>Context Help</title>
    <script language="jscript">
    var LastTopic;

    function ShowContextHelp(Topic)
    {
        if ((document.all("chkHelp").checked) && (LastTopic != Topic))
        {
            // Отобразить справочные сведения для этого
            // контекста.
            window.showHelp("c:\\help\\HelpSample.chm::/"
            + Topic);
            // Сохранить имя раздела.
            LastTopic = Topic;
        }
    }
    </script>
    </HEAD>
    <body onfocus="ShowContextHelp('Topic1.htm')">
        <form id="WebForm2" method="post" runat="server">
            <h2>Context Sensitive Help</h2>
            <INPUT type="text" onfocus="ShowContextHelp('Topic2.htm')">
            <br>
            <INPUT type="button" value="Button"
            onfocus="ShowContextHelp('Topic3.htm')">
            <br>
            <INPUT id="chkHelp" type="checkbox">Show user assistance.
        </form>
    </body>
</HTML>
```

Элемент управления CheckBox позволяет пользователю включить отображение контекстно-зависимой справочной информации. Для каждого элемента страницы процедура *ShowContextHelp* отображает связанный с ним раздел справочной системы, когда тот получает фокус (рис. 14-15).

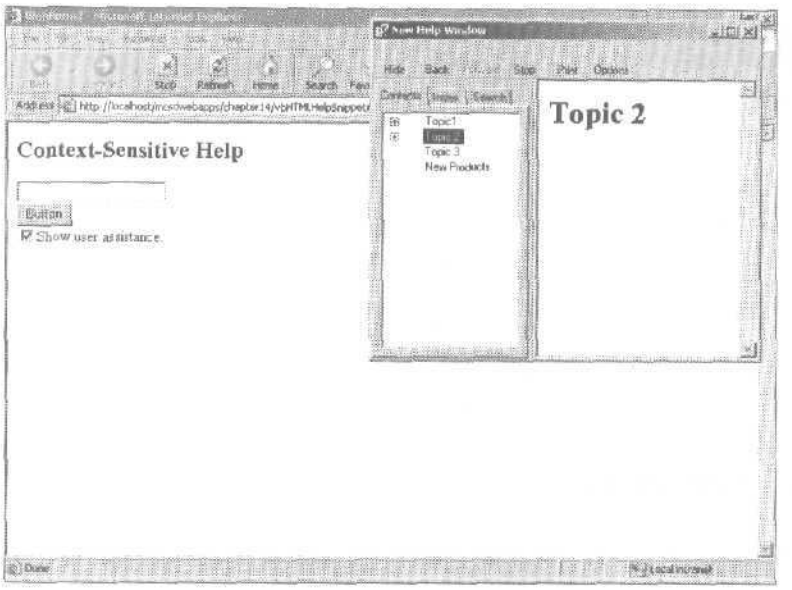


Рис. 14-15. Контекстно-зависимая справка

Резюме

- В Web-приложении можно отображать справочную информацию для пользователя в виде всплывающих подсказок, Web-форм и HTML-страниц, а также в виде компилированных файлов в формате HTML Help.
- Справочная система HTML Help поддерживает встроенное оглавление, указатель и средства поиска.
- Для отображения справочной информации при помощи HTML Help Viewer используется метод *showHelp* объекта *window,* вызываемый из клиентских сценариев.
- Для создания и компиляции проекта справочной системы в HTML Help применяют HTML Help Workshop.
- ActiveX-объект HTML Help применяют для добавления к справочной системе ключевых слов предметного указателя, а также ссылок на смежные разделы и ассоциативных ссылок.
- Прежде чем вызывать компилированный справочный файл из Web-приложения, необходимо целиком загрузить этот файл на пользовательский компьютер.

Практикум 14. Создание и применение справочных файлов HTML Help

Цель этого практикума — создать компилированный файл в формате HTML Help с несколькими разделами, оглавлением, указателем и возможностью поиска и подключить его к Web-приложению.

Продолжительность практикума — около 30 минут.

Упражнение 1. Создание проекта справочной системы



Сейчас вы, применяя знания, полученные в занятии 2, сгенерируете новый проект HTML Help Workshop, создадите пять файлов разделов и включите их в проект. Чтобы не усложнять задание сверх меры, в разделы мы поместили минимум информации, но при желании вы можете добавить в них какие-нибудь свои элементы.

▶ Создание нового проекта HTML Help

1. Запустите HTML Help Workshop. В меню File выберите команду New, в диалоговом окне New выберите Project и следуйте указаниям мастера New Project, чтобы создать пустой проект в каталоге HelpDemo вашего компьютера.
2. Создайте пять HTML-файлов, определите в них заголовки и сохраните, как показано в следующей таблице.

Название	Заголовок	Имя файла
`<title>Topic1</title>`	`<h1>Topic1</h1>`	Topic1.htm
`<title>Topic2</title>`	`<h1>Topic2</h1>`	Topic2.htm
`<title>Topic3</title>`	`<h1>Topic3</h1>`	Topic3.htm
`<title>Topic4</title>`	`<h1>Topic4</h1>`	Topic4.htm
`<title>Topic5</title>`	`<h1>Topic5</h1>`	Topic5.htm

3. Включите эти файлы в проект справочной системы; готовый проект HTML Help Workshop показан на рис. 14-16.

Упражнение 2. Создание оглавления, указателя и поддержки поиска



Сейчас вы автоматически сгенерируете оглавление, указатель и поддержку поиска при компиляции проекта справочной системы. Это позволит вам проверить, насколько полно вы усвоили знания, полученные на занятии 2.

▶ Создание оглавления, указателя и поддержки поиска

1. В диалоговом окне Options проекта перейдите на вкладку Files, установите флажки, необходимые для автоматической генерации файла оглавления и включения ключевых слов в указатель, затем введите имена файлов оглавления и указателя, как показано на рис. 14-17,
2. В окне параметров проекта перейдите на вкладку Compiler и установите флажок Compile Full-Text Search Information (рис. 14-18). Щелкните OK, чтобы закрыть диалоговое окно.

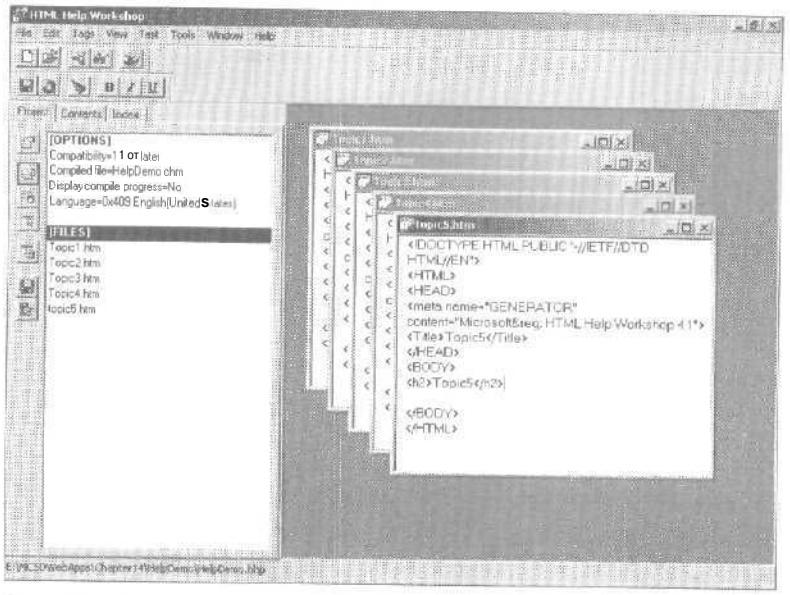


Рис. 14-16. Создание проекта справочной системы

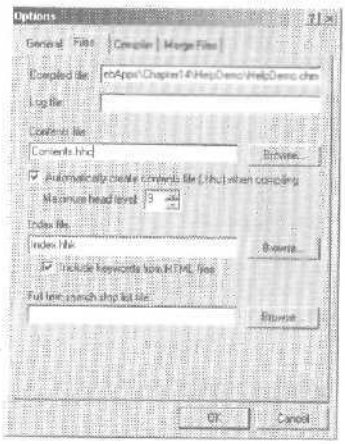


Рис. 14-17. Автоматическая генерация файлов оглавления и указателя

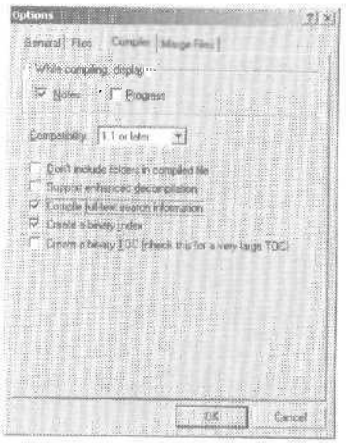


Рис. 14-18. Добавление поддержки поиска

3. В HTML Help Workshop перейдите на вкладку Project и щелкните Save All Files And Compile. В результате Workshop сгенерирует оглавление, указатель и данные для поиска, после чего скомпилирует справочный файл, отражая ход компиляции в журнале (рис. 14-19).
4. На инструментальной панели HTML Help Workshop щелкните кнопку View Compiled File, в диалоговом окне View Compiled File введите имя и путь файла HelpDemo.chm, затем щелкните View — Workshop откроет компилированный справочный файл (рис. 14-20).

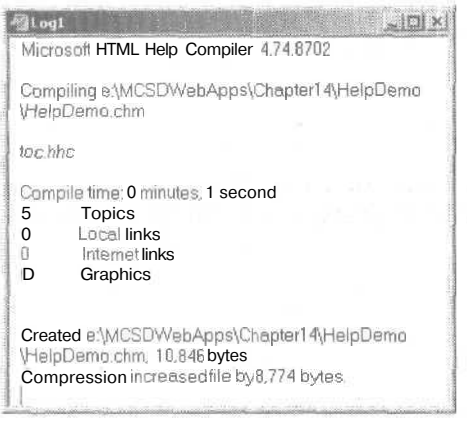


Рис. 14-19. Журнал компилятора

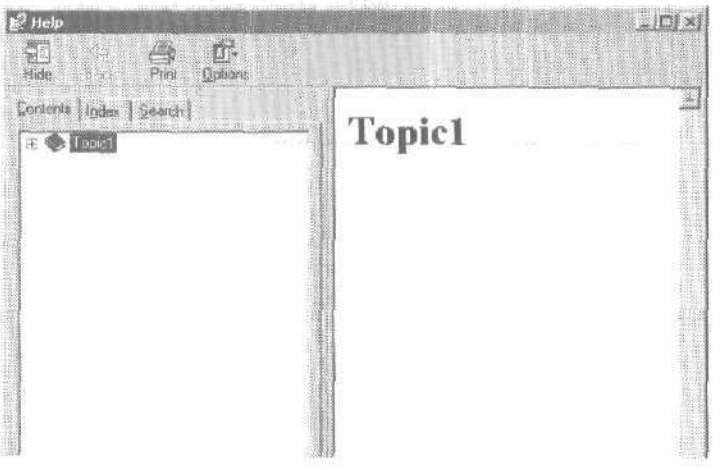


Рис. 14-20. Компилированный справочный файл

Упражнение 3. Добавление ключевых слов к указателю



Сейчас вы добавите к файлам разделов ключевые слова, подлежащие индексированию в указателе, и перекомпилируете справочный файл, чтобы автоматически генерировать элементы указателя.

▶ **Как добавить к файлу раздела ключевые слова для указателя**

1. В HTML Help Workshop откройте файл раздела Topic2.htm и вставьте новую строку между тегами *<body>*и *</body>*.
2. В меню Edit выберите элемент Compiler Information — Workshop откроет одноименное диалоговое окно.
3. Щелкните Add и введите ***Item1*** — имя ключевого слова указателя, которое нужно добавить к разделу, затем щелкните OK. Щелкните OK еще раз, чтобы закрыть диалоговое окно Compiler Information и добавить элемент указателя к файлу Topic2.htm. В результате следующий HTML будет сгенерирован и добавлен к файлу Topic2.htm:

```
<0bject type="application/x-oleobject"
classid="clsid:1e2a7bd0-dab9-11d0-b93a-00c04fc99f9e">
   <param name="Keyword" value="Item1">
</OBJECT>
```

4. Скопируйте этот фрагмент HTML в остальные файлы разделов. Чтобы создать собственные ключевые слова, добавьте новые или измените сгенерированные параметры. Например, следующий HTML-код добавляет к файлу раздела три ключевых слова указателя:

```
<Object type="application/x-oleobject"
classid="clsid:1e2a7bd0-dab9-11d0-b93a-00c04fc99f9e">
   <param name="Keyword" value="Item1">
   <param name="Keyword" value="new">
   <param name="Keyword" value="improved">
</OBJECT>
```

5. Перекомпилируйте проект и просмотрите полученный указатель. Примерный вид предметного указателя показан на рис. 14-21.

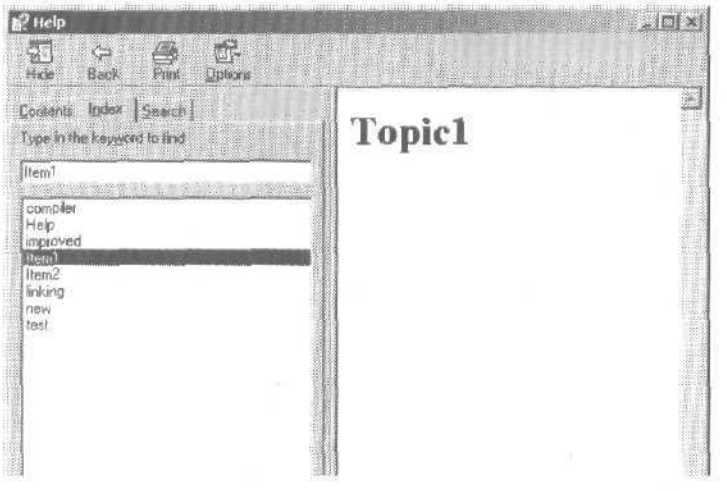


Рис. 14-21. Вид ключевых слов в указателе

Упражнение 4. Подключение справочной системы к Web-форме



Для выполнения этого упражнения обратитесь к материалам занятия 3. Вы должны подключить компилированный справочный файл, созданный ранее, к простейшей Web-форме.

► **Подключение компилированного справочного файла к Web-форме**

1. Создайте новый проект Web-приложения и назовите его vbHelpDemo (или csHelpDemo).
2. **Добавьте к форме Webforml следующий HTML-код (см. ниже). Он содержит инструкции по загрузке справочного файла и позволяет вызывать его из Web-формы:**

```
<p>To use Help with this Web form:</p>
<ol>
   <li><a href="HelpDemo.chm">Click here</a> to download Help.</li>
```

```
    <li>Save the file to the c:/help folder on your computer.</li>
    <li>Then use the buttons below to display Help topics.</li>
</ol>
<INPUT type="button" value="Show Help"
onclick="window.showHelp('c:\\help\\HelpDemo.chm')">
<br>
<INPUT type="button" value="Show Topic3"
onclick="window.showHelp('c:\\help\\HelpDemo.chm::/Topic3.htm')">
<br>
<INPUT type="button" value="Show Topic5"
onclick="window.showHelp('c:\\help\\HelpDemo.chm::/Topic5.htm')">
```

3. Скопируйте компилированный справочный файл, созданный в предыдущих упражнениях, в каталог этого Web-приложения.
4. Запустите Web-приложение. После загрузки можно вызывать разделы справочного файла, щелкая кнопки (рис. 14-22).

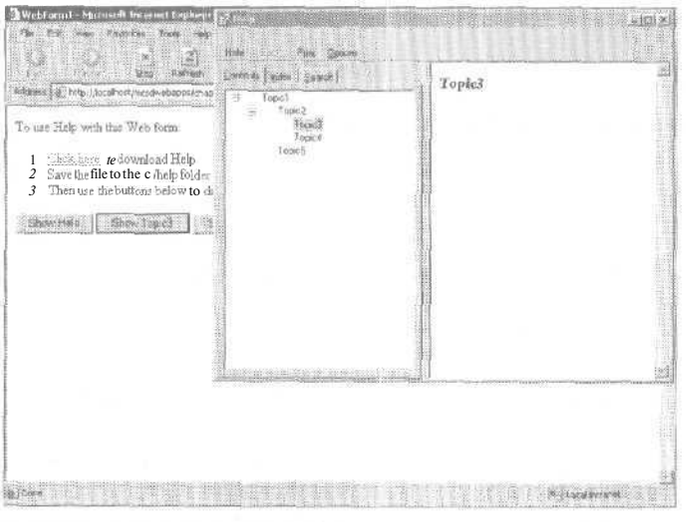


Рис. 14-22. Вызов справочной системы из Web-формы

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. В какой HTML-атрибут, генерируемый ASP.NET, во время выполнения записывается значение свойства *Tooltip* серверного элемента управления?

2. Чем отличается отображение справочных файлов в формате HTML с помощью метода *showHelp* объекта *window* и в окне браузера?

3. Как определить в разделе справочной системы ключевые слова для предметного указателя?

ГЛАВА 15

Глобализация Web-приложений

Занятие 1. Выбор способа 618

Занятие 2. Создание и применение сопутствующих сборок 630

Занятие 3. Другие аспекты глобализации 639

В этой главе

В контексте этой главы *глобализацией* (globalization) называется процесс создания приложения, отвечающего потребностям пользователей из разных стран мира, а значит, принадлежащих к различным мировым культурам. Глобализация приложения — это больше, чем просто перевод элементов пользовательского интерфейса, это выбор и применение соответствующих форматов денежных единиц, даты и времени, а также календаря, направления письма, правил сортировки — список можно продолжать. Приспособление приложения к различиям культурных традиций называется *локализацией* (localization).

К счастью, применение инфраструктуры .NET Framework существенно упрощает задачу локализации приложений, поскольку ее классы для форматирования, сортировки, а также для работы с датами и временем поддерживают концепцию *культур* (cultures). Если в приложении используются классы из пространства имен *System.Globalization*, достаточно задать текущую культуру — и львиная доля работы по локализации будет проделана автоматически!

Из этой главы вы узнаете, как определять культуру, к которой принадлежит пользователь, и в зависимости от результата генерировать соответствующие отклики, направлять пользователя к локализованным версиям Web-приложения либо обрабатывать различия между культурами в коде единственного приложения. В результате выполнения этих операций вы получите локализованный пользовательский интерфейс.

Прежде всего

Для занятий по материалам этой главы необходимы:

- знание языка HTML;
- твердые навыки создания Web-форм в режиме HTML.

Занятие 1. Выбор способа

Прежде чем приступать к локализации, необходимо разобраться в преимуществах и недостатках различных подходов к глобализации Web-приложений. На этом занятии вы познакомитесь с тремя способами организации Web-приложений для их подготовки к глобализации. Мы рассмотрим их от простого к сложному, третий способ подробно описан в занятии 2.

Изучив материал этого занятия, вы сможете:

- ✓ выбрать способ глобализации, оптимальный для вашего приложения;
- ✓ определять культуру, к которой принадлежит пользователь, и генерировать отклик в зависимости от результата;
- ✓ создавать версии Web-приложения, ориентированные на отдельную культуру, при помощи параметров файла Web.config;
- ✓ назначать Web-приложению другую культуру, которую он будет использовать при форматировании дат, денежных сумм и решения других задач;
- ✓ определять текущую культуру приложения;
- ✓ объяснить, чем отличаются свойства *CurrentCulture* и *CurrentUICulture*.

Продолжительность занятия — около 30 минут.

Способы глобализации Web-приложений

Предусмотрено несколько способов создания Web-приложения, поддерживающего различные культуры. Каждый из них предполагает использование особых инструментов из арсенала .NET Framework. Выбор отдельного способа или их комбинации зависит от назначения приложения (таблица 15-1).

Каждый из этих способов предполагает определение культуры, к которой принадлежит пользователь, во время выполнения и строит дальнейшие действия на результатах этой операции. Методика определения культуры пользователя и обработки его запроса подробно описана в следующих разделах; там же сравниваются преимущества и недостатки каждого из этих способов.

Таблица 15-1. Различные подходы к глобализации Web-приложений

Способ	Описание	Оптимальный объект применения
Определение культуры и перенаправление пользователя к соответствующей версии приложения	Для каждой из поддерживаемых культур создается отдельная версия приложения. При получении запроса определяется культура пользователя, и запрос направляется соответствующей версии приложения	Приложения, в которых много текста, требующего перевода, и мало компонентов, выполняющих вычисления
Приспособление к различиям между культурами во время выполнения	Создается единственная версия приложения, которая определяет культуру пользователя и во время выполнения адаптирует генерируемые данные при помощи соответствующих спецификаторов формата и иных инструментов	Простые приложения, отображающие небольшие объемы данных

Таблица 15-1. *(окончание)*

Способ	Описание	Оптимальный объект применения
Использование сопутствующих сборок	Создается единственная версия приложения, хранящая зависимый от культуры текст в файлах ресурсов, компилируемых в сопутствующие сборки. Во время выполнения приложение определяет культуру, к которой принадлежит пользователь, и загружает строки из соответствующей сборки	Приложения, генерирующие содержимое во время выполнения либо по большей части состоящие из вычислительных компонентов

Определение культуры, к которой принадлежит пользователь

Компоненты .NET Framework, обеспечивающие поддержку различных языков и культур, включая классы *CultureInfo*, *Calendar,* а также классы, выполняющие сравнение, которые применяются по ходу этой главы, располагаются в пространстве имен *System.Globalization*. Импортировав это пространство имен, как показано ниже, можно ссылаться на его классы при помощи их сокращенных имен:

Visual Basic .NET

```
Imports System.Globalization
```

Visual C#

```
using System.Globalization;
```

Для получения списка предпочитаемых языков пользователя в ASP.NET используется свойство *UserLanguages* объекта *Request.* Первый элемент массива, возвращаемого свойством *UserLanguages*, представляет текущий язык пользователя. На основе этого знамения можно создать экземпляр класса *CultureInfo*, который и будет представлять текущую культуру пользователя.

Чтобы во время выполнения определить культуру пользователя, выполните следующие действия.

1. Получите значение свойства *UserLanguages* объекта *Request.*
2. Используя полученное значение, создайте экземпляр класса *Culturelnfo,* отражающий текущую культуру пользователя.

Ниже показан пример кода, при первой загрузке страницы определяющий культуру, к которой принадлежит пользователь, и отображающий надпись с ее полным и сокращенным названиями (на английском языке):

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' Этот код выполняется при первом отображении страницы.
    If Not IsPostBack Then
       Dim sLang As String
       ' Определить язык, предпочитаемый пользователем.
       sLang = Request.UserLanguages(0)
```

```
        ' Создать на основе результата объект CultureInfo.
        Dim CurrentCulture As New CultureInfo(sLang)
        lblCulture.Text = CurrentCulture.EnglishName & ":" & _
          CurrentCulture.Name
    End If
End Sub
```

Visual C#

```
private void Page_Load(object sender. System.EventArgs e)
{
    // Этот код выполняется при первом отображении страницы.
    if (!IsPostBack)
    {
        // Определить язык, предпочитаемый пользователем.
        string sLang - Request.UserLanguages[0];
        // Создать на основе результата объект CultureInfo,
        CultureInfo CurrentCulture - new CultureInfo(sLang);
        lblCulture.Text - CurrentCulture.EnglishName + ":" +
          CurrentCulture.Name;
    }
}
```

Внимание! Свойство *Name* класса CultureInfo и массив *UserLanguages* объекта *Request* по-разному используют регистр символов при хранении текстовых значений, поэтому, если нужно сравнить их значения, сначала преобразуйте значение *Name* в нижний регистр.

Использование нескольких версий Web-приложения

С концептуальной точки зрения, проще всего поддерживать несколько культур, направляя пользователя к версии приложения, соответствующей его культуре. При обращении пользователя к начальной странице приложения, определяется культура, к которой принадлежит пользователь, после чего он направляется к подходящей версии Web-приложения (рис. 15-1).

Помимо простоты, этот способ отличается рядом и других достоинств:

- содержимое отдельных версий приложения поддерживается независимо, что позволяет им при необходимости отображать совершенно разную информацию;
- можно автоматически направлять пользователя на сайт, расположенный ближе к нему географически и, вероятно, лучше соответствующий его потребностям;
- можно писать файлы содержимого (например, Web-формы или HTML-страницы) на родном языке пользователя, избегая сложностей с поддержкой локализованных ресурсов.

Автономное сопровождение содержимого, предназначенного для пользователей разных культур, лучше всего подходит для Web-приложений, отображающих значительные объемы информации, которую необходимо перевести или как-то иначе адаптировать. Данный способ предполагает компиляцию и развертывание собственных исполняемых компонентов Web-приложения для каждой версии сайта, поддерживающей определенную культуру. Естественно, это потребует дополнительных усилий для поддержания согласованной работы и сопровождения всех версий Web-сайта.

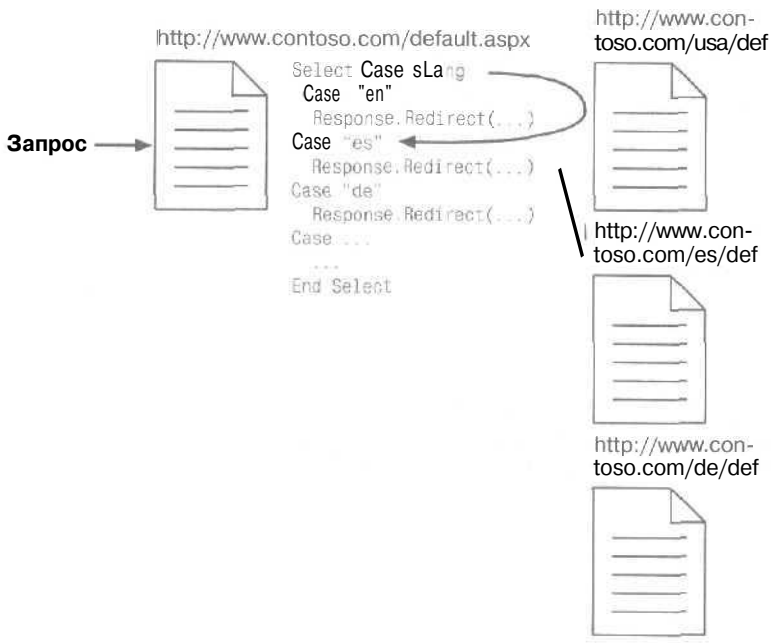


Рис. 15-1. Определение культуры, к которой принадлежит пользователь, и направление его к соответствующей версии приложения

Выбор версии приложения на основе языка пользователя

Массив *UserLanguages* объекта *Request,* как и свойство *Name* класса *CultureInfo* возвращают сведения о культуре в виде значения, состоящего из двух частей. Первые две буквы обозначают код языка, а две последние — код региона. Например, в американской (кодируемой значением «en-US») и британской (en-GB) культурах используется общий язык — английский, но разные форматы денежных сумм и дат.

Чтобы выбрать версию приложения, к которой следует направить пользователи, исключительно на основе языка, выполните следующие действия.

1. Получите сведения о предпочтительном языке пользователя, как показано выше.
2. Получите первые две буквы возвращенного значения.
3. Сравните эти буквы со списком кодов языков.

Например, следующий код определяет язык пользователя и направляет его на соответствующую версию Web-сайта:

Visual **Basic .NET**

```
Private Sub Page_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
   Dim sLang As String
   ' Получить язык, предпочитаемый пользователем.
   sLang = Request.UserLanguages(0)
   ' Взять две первые буквы полученного значения.
   sLang = sLang.Substring(0, 2)
   ' Направить пользователя на соответствующий сайт,
   Select Case sLang
```

```
        Case "en"
            ' Сайт на английском (американская версия).
            Response.Redirect("http://www.contoso.com/usa")
        Case "es"
            ' Сайт на испанском.
            Response.Redirect("http://www.contoso.com/es")
        Case "de"
            ' Сайт на немецком.
            Response.Redirect("http://www.contoso.com/de")
        Case "zh"
            ' Сайт на китайском.
            Response.Redirect("http://www.contoso.com/zh")
        Case Else
            ' Сайт на английском (американская версия),
            Response.Redirect("http://www.contoso.com/usa")
    End Select
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Получить язык, предпочитаемый пользователем.
    string sLang = Request.UserLanguages[0];
    // Взять две первые буквы полученного значения.
    sLang = sLang.Substring(0, 2);
    // Направить пользователя на соответствующий сайт.
    switch (sLang)
    {
        case "en":
            // Сайт на английском (американская версия).
            Response.Redirect("http://www.contoso.com/usa")
            break;
        case "es":
            // Сайт на испанском.
            Response.Redirect("http://www.contoso.com/es");
            break;
        case "de":
            // Сайт на немецком,
            Response.Redirect("http://www.contoso.com/de");
            break;
        case "zh":
            // Сайт на китайском.
            Response.Redirect("http://www.contoso.com/zh");
            break;
        default:
            // Сайт на английском (американская версия).
            Response.Redirect("http://www.contoso.com/usa");
```

```
            break;
        }
    }
```

Список кодов языков и культур вы найдете в разделе справочной системы Visual Studio .NET, посвященном классу *CultureInfo*.

Назначение культуры через параметры Web.config

Элемент *globalization*, определяемый в файле Web.config, позволяет создать Web-приложение, использующее одну определенную культуру. Атрибут *culture* элемента *globalization* задает способ обработки Web-приложением данных, зависимых от культуры, например форматирования дат, денежных сумм и чисел.

Поскольку у Web-приложения может быть только один файл Web.config, установка атрибута *culture* делает приложение специфичным для культуры. Все запросы и отклики это приложение будет обрабатывать согласно правилам единственной культуры, заданной атрибутом *culture*.

Например, если поместить в Web.config следующий элемент, приложение будет обрабатывать запросы и отклики с использованием правил, принятых Б Саудовской Аравии (код соответствующей культуры — ar-SA):

```
<globalization requestEncoding="utf-8" responseEncoding="utf-8"
 culture="ar-SA" />
```

Во время выполнения Web-приложение форматирует отображаемые им даты, денежные суммы и числа согласно значению атрибута *culture* элемента *globalization,* заданного в файле Web.config (рис. 15-2).



Рис. 15-2. Отклик, специфичный для культуры

Как показано на рис. 15-2, значения дат, времени, а также числа и денежные суммы отображаются согласно правилам, принятым в арабской культуре, точнее, в культуре Саудовской Аравии. Тем не менее содержимое страницы не отображается слева направо, а

не наоборот, как принято в арабской письменности. Чтобы страница отображалась корректно, определите HTML-атрибут *dir* для элемента *body* следующим образом:

```
<body dir="rtl">
```

Атрибут *dir* также можно применять для отдельных панелей, текстовых полей и иных элементов управления, Если определить этот атрибут для элемента *body,* все содержимое страницы будет отображаться справа налево (рис. 15-3).



Рис. 15-3. Страница, содержимое **которой отображается** справа **налево**

Приспособление к текущей культуре во время выполнения

По умолчанию Web-приложения работают на сервере с использованием нейтральной культуры. *Нейтральная культура* представляет язык межнационального общения, такой, как английский или испанский. Определив культуру при помощи атрибута *culture* в файле Web.config приложения, вы тем самым заставляете ASP.NET назначить эту культуру всем потокам этого Web-приложения. *Поток* (thread) — это базовая единица исполнения приложения, которой сервер выделяет процессорное время. ASP.NET поддерживает несколько потоков для Web-приложения, исполняя их в рабочем процессе aspnet_wp.exe.

Определение культуры в файле Web.config создает статическую связь между приложением и некоторой культурой. Впрочем, культуру можно назначить и динамически, во время выполнения. Это делается при помощи свойства *CurrentCulture* класса *Thread* следующим образом:

Visual Basic .NET

```
Imports System.Globalization
Imports System.Threading

Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
   Dim sLang As String
```

```
    ' Получить язык, предпочитаемый пользователем.
    slang = Request.UserLanguages(0)
    ' Назначить потоку культуру в соответствии с культурой
    ' пользователя.
    Thread.CurrentThread.CurrentCulture = New CultureInfo(sLang)
End Sub
```

Visual C#

```
using System.Globalization;
using System.Threading;

private void Page_Load(object sender, System.EventArgs e)
{
    // Получить язык, предпочитаемый пользователем.
    sLang = Request.UserLanguages[0];
    // Назначить потоку культуру в соответствии с культурой
    // пользователя.
    Thread.CurrentThread.CurrentCulture = new CultureInfo(sLang);
}
```

Этот код определяет культуру пользователя и назначает текущему потоку соответствующую культуру, которую ASP.NET будет использовать для форматирования дат, денежных сумм и чисел.

Динамическое назначение культуры на уровне потоков обладает рядом преимуществ по сравнению с созданием отдельных версий Web-приложений для поддержки различных культур:

- можно организовать поддержку различных культур в коде единственного экземпляра приложения. Это освобождает от необходимости компиляции и развертывания исполняемых компонентов приложения для каждой из версий, специфичных для культуры;
- приложение располагается по единому Web-адресу — стало быть, можно ненаправлять пользователей к другим Web-сайтам;
- пользователю предоставлено право выбора любой из доступных культур.

Динамическое назначение культуры идеально подходит для простых Web-приложений, не содержащих больших объемов текста, которые необходимо переводить. В ситуациях, когда Web-приложение должно отображать значительный объем иноязычного содержимого, используется методика, описанная далее я разделе «Применение сопутствующих сборок».

Назначение текущей культуры

Код, показанный ниже, иллюстрирует основные преимущества динамического назначения культуры на уровне потока. При первой загрузке страницы этот код генерирует список доступных культур и сохраняет его в элементах раскрывающегося списка. Когда пользователь выбирает культуру из этого списка, код назначает выбранную им культуру текущему потоку, и ASP.NET отображает Web-форму с использованием заданной культуры.

Visual Basic .NET

```
Imports System.Globalization
Imports System.Threading
```

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    ' При первом отображении страницы необходимо создать список
    ' доступных культур.
   If Not IsPostBack Then
      ' Создать массив со списком культур,
      Dim arrCultures() As CultureInfo = _
        CultureInfo.GetCultures(CultureTypes.SpecificCultures)
      Dim item As CultureInfo
      ' Для каждой культуры...
      For Each item In arrCultures
            ' ...поместить название культуры в элемент списка.
           drpCulture.Items.Add(New ListItem(item.EnglishName, item.Name))
            ' Если это текущая культура потока, выбрать ее.
          If item.Name = Thread.CurrentThread.CurrentCulture.Name Then
                ' Выделить этот элемент списка
            drpCulture.SelectedIndex = drpCulture.Items.Count - 1
           End If
      Next
  End If
End Sub

Private Sub drpCulture_SelectedIndexChanged(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles drpCulture.SelectedIndexChanged
    ' Изменить культуру текущего потока в соответствии
    ' с выбранным элементом списка.
    Thread.CurrentThread.CurrentCulture = New _
      CultureInfo(drpCulture.SelectedItem.Value)
    ' Получить выбранную культуру из списка.
    Dim SelectedCulture As New CultureInfo(drpCulture.SelectedItem.Value)
End Sub

Private Sub Page_PreRender(ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles MyBase.PreRender
    ' Отобразить в заголовке название культуры на ее языке.
    head1.InnerHtml = Thread.CurrentThread.CurrentCulture.NativeName
    ' Отобразить дату, денежную сумму и число в формате,
    ' определяемом заданной культурой,
    Label1.Text = DateTime.Now.ToString("F")
    Label2.Text = 1234567890.ToString("C")
    Label3.Text = 1234567890.ToString("N")
End Sub
```

Visual C#

```
using System.Globalization;
using System.Threading;
```

```
private void Page_Load(object sender, System.EventArgs e)
{
   // При первом отображении страницы необходимо создать список
   // доступных культур.
   if (!IsPostBack)
   {
         // Создать массив со списком культур.
         CultureInfo[] arrCultures =
           CultureInfo.GetCultures(CultureTypes.SpecificCultures);
         // Для каждой культуры...
         foreach (CultureInfo item in arrCultures)
         {
               // ... поместить название культуры в элемент
               // списка.
               drpCulture.Items.Add(new ListItem(item.EnglishName,
                 item.Name));
               // Если это текущая культура потока, выбрать ее,
               if (item.Name == Thread.CurrentThread.CurrentCulture.Name)
                     // Выделить этот элемент списка
               drpCulture.SelectedIndex = drpCulture.Items.Count - 1;
         }
   }
}

private void drpCulture_SelectedIndexChanged(object sender,
  System.EventArgs e)
{
   // Изменить культуру текущего потока в соответствии с
   // выбранным элементом списка.
   Thread.CurrentThread.CurrentCulture = new
     CultureInfo(drpCulture.SelectedItem.Value);
   // Получить выбранную культуру из списка.
   CultureInfo SelectedCulture = new
     CultureInfo(drpCulture.SelectedItem.Value);
}

private void Page_PreRender(object sender, System.EventArgs e)
{
   // Отобразить в заголовке название культуры на ее языке,
   head1.InnerHtml = Thread.CurrentThread.CurrentCulture.NativeName;
   // Отобразить дату, денежную сумму и число в формате,
   // определяемом заданной культурой.
   Label1.Text = DateTime.Now.ToString("F");
   Label2.Text = 1234567890.ToString("C");
   Label3.Text = 1234567890.ToString("N");
}
```

Этот код формирует текст заголовка и надписей в обработчике события *PreRender*, чтобы назначение культуры вступило в силу до записи строк в соответствующие элементы управления. Метод ToString применяет соответствующие спецификаторы для форматирования строк согласно правилам, принятым в назначенной культуре. Подробно о спецификаторах формата рассказано в разделе «Formatting Types» справочной системы Visual Studio .NET. Результат исполнения предыдущего кода показан на рис. 15-4.



Рис. 15-4. Результат назначения культуры на уровне потока

Если пользователь выберет в списке новую культуру (рис. 15-4), то текст заголовка, календарь, даты, денежные суммы и числа автоматически приводятся в соответствие с ней.

Определение культуры потока

Свойство *CurrentCulture* потока позволяет определить культуру, используемую Web-приложением во время выполнения. Например, если текущая культура является арабской или иудейской, следующий код отображает на Web-форме панель, отформатированную справа налево:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    ' Определить культуру текущего потока.
    Dim CurrentCulture As CultureInfo = Thread.CurrentThread.CurrentCulture
    ' Определить язык.
    Dim sLang As String = CurrentCulture.Name.Substring(0, 2)
    ' Если это арабский или иврит, отображать содержимое справа
    ' налево.
    If sLang = "ar" or sLang = "he" Then
        panel1.Attributes.Add("dir", "rtl")
```

```
    End If
End Sub
```

Visual C#

```
private voia Page_Load(object sender, System.EventArgs e)
{
        // Определить культуру текущего потока.
        CultureInfo CurrentCulture - Thread.CurrentThread.CurrentCulture;
        // Если это арабская или иудейская культура, отображать
        // содержимое справа налево.
        string sLang  - CurrentCulture.Name.Substring(0, 2);
        if ((sLang == "ar") ||(sLang == "he"))
         {
              Panel1.Attributes.Add("dir", "rtl");
         }
}
```

Применение сопутствующих сборок

В предыдущем разделе говорилось о поддержке различных культур в пределах одного Web-приложения, но ни слова об одной важной детали: как обрабатывать содержимое, требующее перевода? Эту задачу решают при помощи сопутствующих сборок.

При применении этого способа строки, переведенные на язык каждой из поддерживаемых культур, хранятся в отдельных файлах сборок, каждый из которых содержит только ресурсы для некоторой культуры. Подобные сборки можно выбирать и загружать автоматически в зависимости от значения свойства *Current UICulture*. Помимо преимуществ, которые дает приспособление к культуре во время выполнения, использование сопутствующих сборок еще больше упрощает отображение иноязычного содержимого из нескольких источников, содержащих переведенные ресурсы. Подробнее о создании и применении сопутствующих сборок — в следующем занятии.

Занятие 2. Создание и применение сопутствующих сборок

Сопутствующие сборки (satellite assemblies) — это файлы сборок (.dll-файлы) с локализованными ресурсами приложения. Каждый такой файл содержит ресурсы для какой-либо одной культуры. У приложения может быть несколько сопутствующих сборок, в зависимости от числа поддерживаемых им культур.

На этом занятии вы научитесь готовить приложение к применению сопутствующих сборок, помещать локализованные элементы пользовательского интерфейса в сопутствующие сборки, а также загружать и отображать эти элементы во время выполнения.

Изучив материал этого занятия, вы сможете:

- ✓ создавать **локализованные ресурсы и помещать** их **в** сопутствующие **сборки;**
- ✓ создавать в приложении поддержку отображения в HTML-элементах локализованных строк из сопутствующих сборок;
- ✓ загружать ресурсы в объект *Resource Manager,*
- ✓ объяснить, как ASP.NET выбирает загружаемые ресурсы на основе свойства *Current UICulture*;
- ✓ получать строки из объекта *Resource Manager* и отображать их в элементах пользовательского интерфейса.

Продолжительность занятия — около 30 минут.

Как работают сопутствующие сборки

В проектах Web-приложений сопутствующие сборки применяют для хранения строк, графических элементов и других элементов пользовательского интерфейса, зависящих от культуры и переведенных на язык этой культуры. Сопутствующие сборки создают с помощью инструмента Resource Manager. Во время выполнения Web-приложение загружает переведенные строки в Web-форму, руководствуясь значением свойства *CurrentUICulture* текущего потока (рис. 15-5).

Чтобы задействовать сопутствующие сборки, предусмотрена следующая процедура.

1. Определите атрибуты *id* и *runat* для всех элементов пользовательского интерфейса приложения, требующих перевода. У серверных элементов управления эти атрибуты имеются по умолчанию, а к HTML-элементам (таким, как *heading, paragraph* и *span*) их придется добавить самостоятельно. Это позволит загружать переведенные строки в эти элементы управления.
2. Создайте файл ресурсов для использования по умолчанию. Эти ресурсы будут загружаться, когда культура не задана либо не удалось ее определить. Имя этого файла указывается в виде *имя_файла*.resx, например strings.resx.
3. Создайте файлы ресурсов для каждого из языков межнационального общения, поддерживаемых приложением. Для этих файлов следует назначать имена в виде *имя_файла.код_языка*.resx, например strings.es.resx.
4. Если требуется, создайте файлы ресурсов для каждого из языков региональных культур, поддерживаемых приложением. Этим файлам назначают имена вида *имя_файла.-код_языка-код_региона*.resx, например strings.ex-MX.resx.
5. Напишите код для загрузки этих ресурсов в Web-формы через объект *ResourceManager*

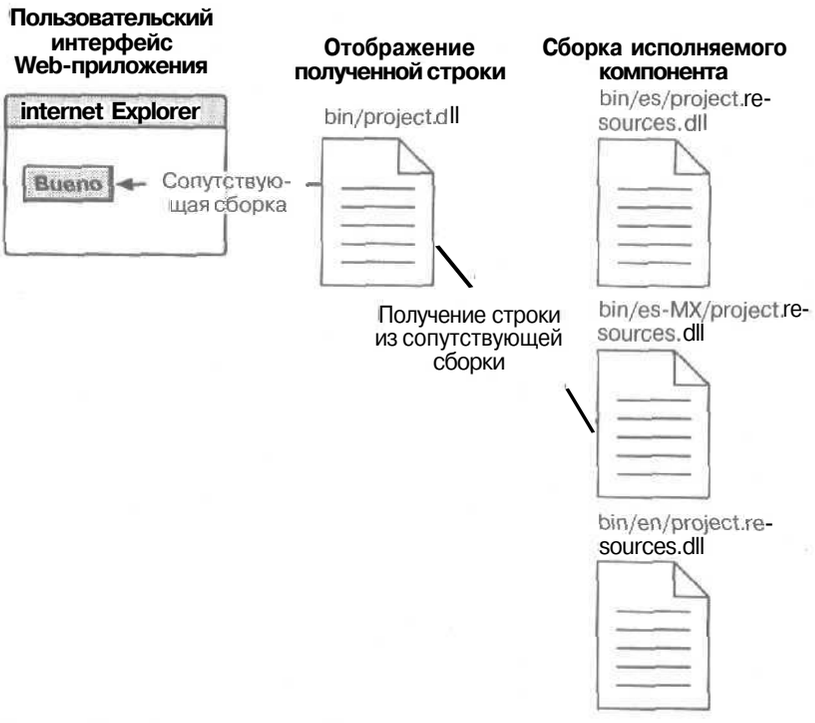


Рис. 15-5. Сопутствующие сборки в действии

6. Напишите кол, определяющий культуру пользователя и соответствующим образом устанавливающий свойства *CurrentUICulture* и *CurrentCulture* класса *Thread*. Свойство *CurrentCulture* определяет формат, используемый ASP.NET для дат, денежных суVIM и чисел, а свойство *CurrentUICulture*— сопутствующую сборку, из которой будут загружены переведенные строки для элементов интерфейса.
7. Напишите код, получающий строки из файлов ресурсов и отображающий их в элементах Web-формы.

Более подробно эти действия разбираются в следующих разделах.

Создание поддержки ресурсов в HTML-элементах

Загрузка ресурсов из сопутствующих сборок в элементы Web-формы выполняется на сервере, поэтому необходимо удостовериться, что у всех элементов Web-формы определены атрибуты *id* и *runat*, иначе они окажутся недоступными серверному коду. По умолчанию у серверных элементов управления эти атрибуты имеются, а у HTML-элементов управления — нет.

Вот пример HTML-кода простой Web-формы, содержащей заголовок, небольшой текст и несколько элементов управления:

```
<HTML>
   <HEAD>
         <title>WebForm1</title>
   </HEAD>
   <body>
         <form id="Form1" method="post" runat="server">
```

```
            <h1>Sample heading</h1>
            <p>Some introductory text.</p>
            <span>Enter a number: </span>
            <asp:TextBox ID="txtCurrency" Runat="server" />
            <br>
            <span>Amount in local currency: </span>
            <asp:Label ID="lblCurrency" Runat="server" />
            <br>
            <br>
            <asp:Button ID="butOK" Text="OK" Runat="server" />
            <input type="button" onclick="txtCurrency.value=''"
              value="Cancel">
        </form>
    </body>
</HTML>
```

Чтобы наделить эту Web-форму способностью использовать ресурсы, необходимо внести в нее ряд изменений (см. HTML-элементы, выделенные в примере ниже **полужирным шрифтом**):

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
Inherits="vbSatelliteSnippet.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
    <HEAD>
        <title>WebForm1</title>
    </HEAD>
    <body>
        <form id="Form1" method="post" runat="server">
            <h1 id="head1" runat="server">Sample heading</h1>
            <p id="p1" runat="server">Some introductory text.</p>
            <span id="sp1" runat="server">Enter a number: </span>
            <asp:TextBox ID="txtCurrency" Runat="server" />
            <br>
            <span id="sp2" runat="server">
            Amount in local currency: </span>
            <asp:Label ID="lblCurrency" Runat="server" />
            <br>
            <br>
            <asp:Button ID="butOK" Text="OK" Runat="server" />
            <input id="butCancel" runat="server" type="button"
              onclick="txtCurrency.value=''" value="Cancel">
        </form>
    </body>
</HTML>
```

Наличие атрибутов *id* и *runat y* HTML-элементов заставляет Visual Studio .NET обрабатывать их как серверные элементы управления. При переключении из режима **HTML** в режим Design, вы можете заметить, что Visual Studio .NET добавила в файл кода Web-формы следующие объявления:

Visual Basic .NET

```
Protected WithEvents head1 As System.Web.UI.HtmlControls.HtmlGenericControl
Protected WithEvents p1 As System.Web.UI.HtmlControls.HtmlGenericControl
Protected WithEvents sp1 As System.Web.UI.HtmlControls.HtmlGenericControl
Protected WithEvents sp2 As System.Web.UI.HtmlControls.HtmlGenericControl
Protected WithEvents butCancel As System.Web.UI.HtmlControls.HtmlInputButton
```

Visual C#

```
protected System.Web.UI.HtmlControls.HtmlGenericControl head1;
protected System.Web.UI.HtmlControls.HtmlGenericControl p1;
protected System.Web.UI.HtmlControls.HtmlGenericControl sp1;
protected System.Web.UI.HtmlControls.HtmlGenericControl sp2;
protected System.Web.UI.HtmlControls.HtmlInputButton butCancel;
```

Эти объявления позволяют присваивать значения и устанавливать атрибуты *innerHTML* этих HTML-элементов управления во время выполнения.

Создание файлов ресурсов

При использовании в Web-приложении сопутствующих сборок для хранения строк на нескольких языках применяются три типа файлов ресурсов, описанных в таблице 15-2.

Таблица 15-2. Типы файлов ресурсов, применяемых с сопутствующими сборками

Тип	Способ именования	Как компилируется	Назначение
Файл ресурсов по умолчанию	*имя_файла*.resx, например strings.resx	Встраивается в сборку исполняемого компонента	Используется, если культура не задана или не распознана Web-приложением
Файл ресурсов для конкретного языка	*Имя_файла.код_языка*.resx, например strings.es.resx	В отдельную сборку, которая содержит только ресурсы и хранится в каталоге, названном по коду языка и вложенному в каталог bin, например bin\es\	Хранит строки на языке, используемом в нескольких культурах
Файл ресурсов для конкретной культуры	*имя_файла.код_языка-код_региона*.resx, например strings.es-MX.resx	В отдельную сборку, которая содержит только ресурсы и хранится в каталоге, названном по коду языка и коду региона, вложенном в каталог bin, например bin\es-MX\	Хранит строки на диалекте некоторого языка, используемом в какой-либо одной культуре

Список кодов культур, идентифицирующих языки и файлы ресурсов для конкретной культуры, вы найдете в разделе справочной системы, посвященному классу *CultureInfo*. Эти коды также используются в свойстве *Name* класса *CultureInfo* и массиве *UserLanguages* объекта *Request*.

Чтобы в Visual Studio .NET добавить файл ресурсов к проекту Web-приложения, выполните следующие действия.

1. В меню Project выберите команду Add New Item, затем в списке Templates щелкните элемент Assembly Resource File.
2. Введите имя для создаваемого файла ресурсов и щелкните Open — Visual Studio .NET создаст новый файл ресурсов и добавит его к проекту (рис. 15-6).

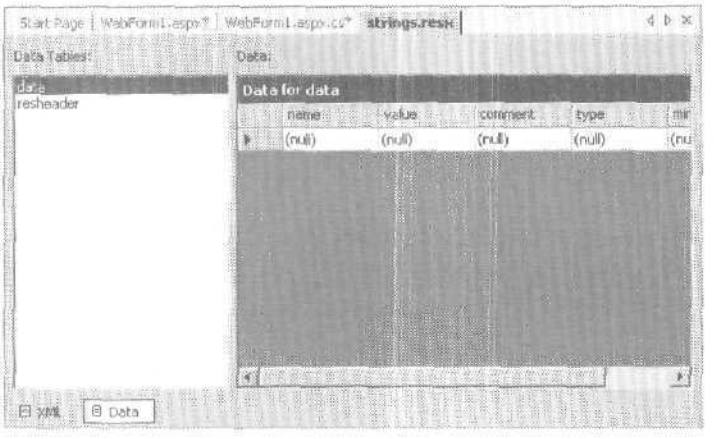


Рис. 15-6. Новый файл ресурсов

Разместить переведенные строки пользовательского интерфейса Web-формы в файле ресурсов можно следующим образом.

1. Откройте файл ресурсов и введите в поле Name идентификатор элемента Web-формы. Этот идентификатор должен быть уникальным в пределах приложения, поэтому удобно использовать имена в виде *имя_формы.имя_элемента*, например webform1.butok.
2. Введите в поле Value строку, которую этот элемент управления должен отображать в иноязычном варианте интерфейса.

На рис. 15-7 показаны иноязычные варианты строк для серверного и HTML-элемента управления, которые были созданы в предыдущем разделе.

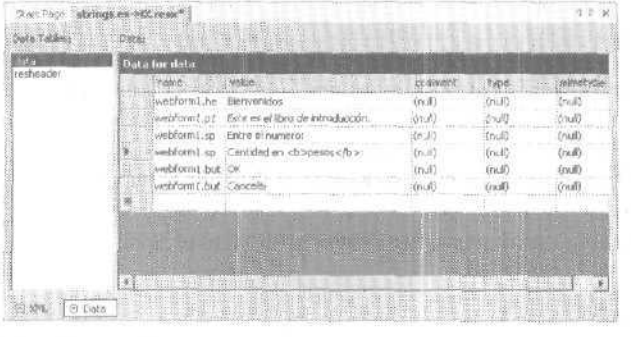


Рис. 15-7. Иноязычные варианты строк в файле ресурсов

Совет По умолчанию поле name файла ресурсов чувствительно к регистру символов, поэтому во избежание ошибок при поиске ресурсов во время выполнения, необходимо применять согласованные схемы использования заглавных и строчных букв при именовании ресурсов. Например, в этом занятии принято соглашение, допускающее применение в именах ресурсов только строчных букв.

Загрузка файлов ресурсов

В .NET Framework классы для управления ресурсами находятся в пространстве имен *System.Resources*. Чтобы вызывать классы из этого пространства имен при помощи сокращенных имен, поместите в начало Web-формы следующий оператор:

Visual Basic .NET

```
Imports System.Resources
```

Visual C#

```
using System.Resources;
```

Чтобы загрузить ресурсы в Web-форму, создайте экземпляр класса *ResourceManager*. Например, следующие строки кода загружают файл ресурсов в Web-приложение:

Visual Basic .NET

```
' Загрузить ресурсы.
Protected gStrings As New ResourceManager("vbSatelliteSnippet.strings", _
  GetType(WebForm1).Assembly)
```

Visual C#

```
// Загрузить ресурсы.
protected ResourceManager gStrings = new
  ResourceManager("csSatelliteSnippet.strings", typeof(WebForm1).Assembly);
```

Этот пример загружает ресурсы из сборки Strings, связанной с родительской сборкой Webform1, и сохраняет возвращаемый объект в переменной *gStrings*. Аргументы конструктора класса *ResourceManager* описаны в таблице 15-3.

Таблица 15-3. Аргументы конструктора объекта *ResourceManager*

Имя	Описание
baseName	Представляет пространство имен для ресурсов данного проекта. В проектах на Visual Basic .NET используется схема именования *«имя_проекта.имя_файла_ресурсов»*, а в проектах на Visual C# — *«корневое_пространство_имен.имя_файла_ресурсов»*. Во вторую часть имени код культуры не входит, поэтому пространство имен MyProj.Strings включает содержимое файлов Strings.resx, Strings.en.resx, Strings.es.rex и т. п.
assembly	Объект, представляющий текущую сборку исполняемых компонентов приложения. Значение этого аргумента можно получить из свойства *Assembly* класса *Type*, например GetType("Webform1").Assembly

Получение и установка культуры пользовательского интерфейса

Как показано в предыдущем занятии, для определения культуры, к которой принадлежит пользователь, применяют массив *User Languages* объекта *Request*, а для назначения культуры текущему потоку — свойство *CurrentCulture* класса *Thread*. При использовании сопутствующих сборок для этого потребуется одно дополнительное действие — установка свой-

ства *CurrentUICulture*. Это свойство определяет сопутствующую сборку, строки из которой загружаются в объект *Resource Manager*.

Чтобы задать культуру для пользовательского интерфейса, необходимо установить свойство *CurrentUICulture* объекта *Thread*, Например, следующий код определяет язык, применяемый пользователем, и назначает соответствующую культуру для текущего потока и пользовательского интерфейса этого потока:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
   If Not IsPostBack Then
      ' Определить язык, предпочитаемый пользователем,
      Dim sLang As String = Request.UserLanguages(0)
      ' Назначить потоку культуру, определяющую форматирование,
      ' правила сравнения и т.п.
      Thread.CurrentThread.CurrentCulture = _
         CultureInfo.CreateSpecificCulture(slang)
      ' Установить значение свойства CurrentUICulture, чтобы задать
      ' сопутствующую сборку, из которой будут загружаться ресурсы,
      Thread.CurrentThread.CurrentUICulture - New CultureInfo(slang)
   End If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
   if (!IsPostBack)
   {
         // Определить язык, предпочитаемый пользователем,
         string sLang = Request.UserLanguages[0];
         // Назначить потоку культуру, определяющую форматирование,
         // правила сравнения и т.п.
         Thread.CurrentThread.CurrentCulture =
           CultureInfo.CreateSpecificCulture(sLang);
         // Установить значение свойства CurrentUICulture, чтобы задать
         // сопутствующую сборку, из которой будут загружаться ресурсы.
         Thread.CurrentThread.CurrentUICulture = new CultureInfo(sLang);
   }
}
```

В этом коде значения свойств *CurrentCulture* и *CurrentUICulture* устанавливаются по отдельности. Для установки свойства *CurrentCulture* вызывается метод *CreateSpecificCulture*, который никогда не возвращает объект, представляющий нейтральную культуру. *Нейтральные культуры* (neutral culture) представляют языки межнационального общения (используемые в нескольких культурах), а не специфичные языки и регионы. Нейтральная культура не может быть значением свойства *CurrentCulture*, поскольку среди прочего это свойство применяет региональные стандарты для форматирования значений денежных сумм и т. п.

После установки свойства *CurrentCulture* потока ASP.NET автоматически выбирает подходящие ресурсы по следующему алгоритму:

1. если найдена сопутствующая сборка, культура которой точно соответствует заданной, загружаются ресурсы из этой сборки;
2. **если найдена сопутствующая** сборка с **нейтральной** культурой, **соответствующей значению** *CurrentUICulture*, используются ресурсы из **этой** сборки;
3. если не найдено ни одной сопутствующей сборки, соответствующей значению *CurrentUICulture*, загружаются ресурсы из сборки исполняемых компонентов приложения.

Отображение строк из файлов ресурсов

После выполнения всех предыдущих действий отображение на Web-форме строк из файла ресурсов не должно вызывать затруднений.

Для отображения строк из файла ресурсов на Web-форме применяется метод *GetStrings* объекта *Resource Manager*. Этот метод извлекает строки из файла ресурсов и записывает их в свойства *Text, value* или *innerHTML* различных элементов Web-формы.

Следующий код собирает воедино все вышеперечисленные действия. Он загружает файл ресурсов, определяет культуру пользователя, назначает ее текущему потоку и отображает строки из файла ресурсов в серверных элементах управления и HTML-элементах Web-формы:

Visual Basic .NET

```
Imports System.Resources
Imports System.Globalization
Imports System.Threading

' Загрузить ресурсы.
Protected gStrings As New ResourceManager("vbSatelliteSnippet.strings", _
  GetType(WebForm1).Assembly)

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
If Not IsPostBack Then
    ' Определить язык, предпочитаемый пользователем.
    Dim sLang As String = Request.UserLanguages(0)
    ' Назначить потоку культуру, определяющую форматирование,
    ' правила сравнения и т.п.
    Thread.CurrentThread.CurrentCulture = _
  CultureInfo.CreateSpecificCulture(sLang)
    ' Установить значение свойства CurrentUICulture, чтобы задать
    ' сопутствующую сборку, из которой будут загружаться ресурсы.
    Thread.CurrentThread.CurrentUICulture = New CultureInfo(sLang)
    ' Получить строки из файла ресурсов.
    head1.InnerHtml = gStrings.GetString("webform1.head1")
    p1.InnerHtml = gStrings.GetString("webform1.p1")
    sp1.InnerHtml = gStrings.GetString("webform1.sp1")
    sp2.InnerHtml = gStrings.GetString("webform1.sp2")
    butOK.Text = gStrings.GetString("webform1.butOK")
    butCancel.Value = gStrings.GetString("webform1.butCancel")
```

```
End If
End Sub
```

Visual C#

```
using System.Globalization;
using System.Threading;
using System.Resources;

private void Page_Load(object sender, System.EventArgs e)
{
    if (!IsPostBack)
    {
        // Определить язык, предпочитаемый пользователем.
        string sLang = Request.UserLanguages[0];
        // Назначить потоку культуру, определяющую форматирование,
        // правила сравнения и т.п.
        Thread.CurrentThread.CurrentCulture =
          CultureInfo.CreateSpecificCulture(sLang);
        // Установить значение свойства CurrentUICulture, чтобы задать
        // сопутствующую сборку, из которой будут загружаться ресурсы
        Thread.CurrentThread.CurrentUICulture = new CultureInfo(sLang);
        // Получить строки из файла ресурсов.
        head1.InnerHtml = gStrings.GetString("webform1.head1");
        p1.InnerHtml - gStrings.GetString("webform1.p1");
        sp1.InnerHtml = gStrings.GetString("webform1.sp1");
        sp2.InnerHtml = gStrings.GetString("webform1.sp2");
        butOK.Text = gStrings.GetString("webform1.butOK");
        butCancel.Value = gStrings.GetString("webform1.butCancel");
    }
}
```

Во время выполнения этот код извлекает из файла ресурсов и отображает строки на языке, который предпочитает пользователь (рис. 15-8).

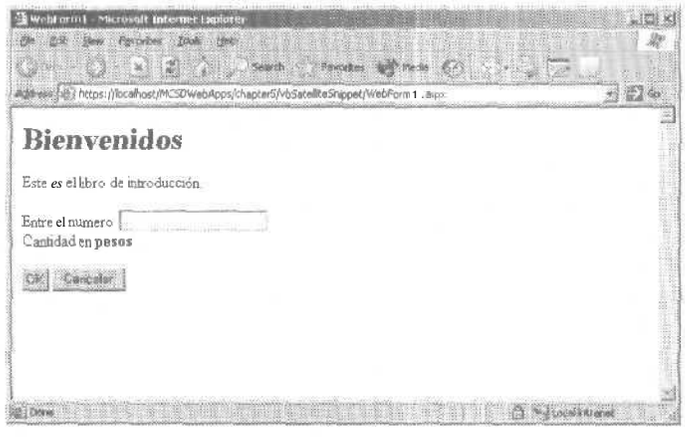


Рис. 15-8. Отображение строк из файла ресурсов

Занятие 3. Другие аспекты глобализации

.NET Framework автоматически обрабатывает большинство параметров, связанных с глобализацией, руководствуясь значением свойства *CurrentCulture*. Однако глобализация затрагивает не только интерфейс приложения, но и методики программирования. На этом занятии мы познакомим вас с рядом общих моментов, которые нужно учесть при разработке интернациональных приложений. **Вы** также узнаете о проблемах, связанных с хранением и представлением данных с применением расширенных наборов символов, используемых в ряде языков.

Изучив материал этого занятия, вы сможете:

- ✓ обсуждать некоторые общие вопросы программирования, связанные с глобализацей;
- ✓ сохранять Web-формы, содержащие специальные символы из различных языков;
- ✓ использовать Web-формы, сохраненные в различных кодировках;
- ✓ определять кодировку символов, запрошенную пользователем;
- ✓ генерировать отклики с использованием различных кодировок.

Продолжительность занятия — около 10 минут.

Общие вопросы программирования

В разделе «Best Practices for Developing World-Ready Applications» справочной системы Visual Studio .NET вы найдете весьма полезный список, где указано, что следует учесть при разработке Интернет-прил ожени й, которые планируется адаптировать для других языков. В следующих разделах более подробно обсуждаются моменты, связанные с Web-приложениями.

Сортировка и сравнение строк

Сортировка и сравнение строк с применением классов .NET Framework корректно работают независимо от текущей культуры. О глобализации придется вспомнить только при написании собственного кода для решения этих задач, например нестандартной процедуры сортировки строк. В этом случае рекомендуется применять класс *CompareInfo* из пространства имен *System.Globalization*, позволяющий выполнять сравнение строк с учетом особенностей, обусловленных различными культурами.

Нестандартные верификаторы

Серверные элементы управления ASP.NET запрашивают некоторые свойства с учетом текущей культуры, это касается и верификаторов. Тем не менее при создании нестандартных верификаторов непременно следует учитывать моменты, связанные с глобализацией.

Например, при разработке процедуры, выполняющей нестандартную проверку вводимых данных на предмет их принадлежности к числовым данным, не следует полагаться на обычную проверку числового значения. В этом случае лучше выполнять проверку с учетом особенностей различных культур. Так, проверку числового значения выполняют при помощи класса *Convert,* помещенного в структуру обработки исключений.

Генерация строк

Поскольку в некоторых языках принято писать и читать справа налево, а значение некоторых слов разительно меняется, если рядом находятся определенные слова, трудно до-

биться того, чтобы генерация строк посредством конкатенации работала во всех культурах. Лучше хранить в файлах ресурсов готовые строки и использовать их по мере необходимости при помощи менеджера ресурсов.

Получение подстрок

В текстах на некоторых языках встречаются знаки, состоящие из двух символов, — так называемые глифы. Это нужно учесть при извлечении подстрок: бессмысленно получать первый символ отклика, если это часть глифа.

Кодировка символов

Разработано множество способов представления символов, составляющих строки, в памяти компьютера. Они называются *кодировкой* (encoding), поскольку определяют кодирование символов в виде чисел для представления в компьютере.

Общеязыковая исполняющая среда использует кодировку UTF-16, в которой для представления символов используются 16-разрядные числа. Их вполне достаточно, чтобы закодировать символы всех языков, в которых есть письменность. К сожалению, размер символа в этой кодировке почти в два раза превышает размеры символов в других распространенных кодировках, поэтому UTF-16 не очень подходит для передачи текстовых данных через Интернет.

По этой причине при интерпретации запросов и генерации откликов ASP.NET по умолчанию использует кодировку UTF-8. Эта кодировка оптимизирована для 127 младших символов ASCII, то есть она позволяет эффективно кодировать тексты на языках, использующих латинский алфавит. Кроме того, UTF-8 поддерживает преемственную совместимость с кодировкой ASCII: программы, читающие тексты в кодировке UTF-8, также способны интерпретировать ASCII-файлы.

Общеязыковая исполняющая среда обрабатывает тексты в различных кодировках, преобразуя их при помощи классов из пространства имен *System. Text.* Как правило, беспокоиться о преобразовании символов в различных кодировках не приходится; об этом вспоминают лишь в двух случаях:

- если предполагается сохранить Web-форму или HTML-файл, содержащие расширенные символы из кодировки UTF-8;
- если необходимо поддерживать кодировки, отличные от UTF-8.

Сохранение файлов в различных кодировках

Если добавили в Web-форму какие-либо символы, кроме 127 первых символов ASCII, Visual Studio .NET предлагает сохранить соответствующий файл в подходящей кодировке (рис. 15-9).

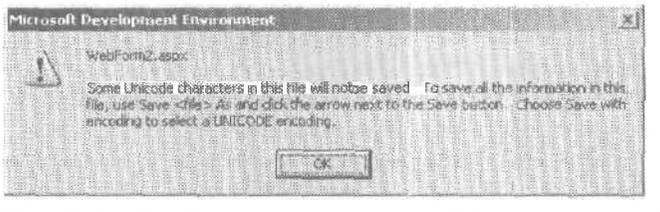


Рис. 15-9. Приглашение к сохранению файла в кодировке UNICODE

Чтобы сохранить файл в подходящей кодировке, выполните следующие действия.

1. В меню File выберите команду Save As, раскройте список Save и выберите элемент Save With Encoding — Visual Studio .NET откроет окно Advanced Save Options (рис. 15-10).

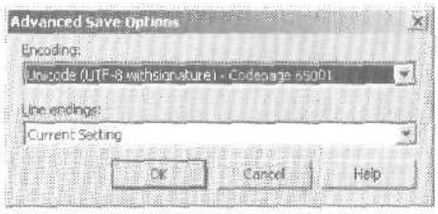


Рис. 15-10. Выбор кодировки при **сохранении** файла

2. **Выберите кодировку, в** которой **нужно сохранить** файл, **и щелкните OK.**

Если файл сохранен в кодировке UTF-8 с сигнатурой, ASP.NET сможет автоматически определить его кодировку в дальнейшем. Если отказаться от сохранения сигнатуры или выбрать другую кодировку, потребуется включить в файл Web.config атрибут *fileEncoding*, например:

```
<globalization requestEncoding="utf-8" responseEncoding="utf-8"
  fileEncoding="utf-8" />
```

Использование других кодировок

По умолчанию Web-приложения интерпретируют содержимое запросов и генерируют отклики с использованием кодировки UTF-8. Эти параметры заданы элементом *globalization* в файле Web.config, как показано в предыдущем разделе.

Если Web-приложение получает запрос в кодировке, отличной от UTF-8, ASP NET интерпретирует его с использованием кодировки, указанной в запросе. Однако ASP NET автоматически не использует указанную в запросе кодировку для генерации отклика на этот запрос.

Чтобы определить, задана ли для запроса определенная кодировка, получите свойство *ContentEncoding* объекта *Request*. Например, следующий код отображает на Web-форме кодировку запроса:

Visual Basic .NET

```
Response.Write(Request.ContentEncoding.WebName)
```

Visual C#

```
Response.Write(Request.ContentEncoding.WebName);
```

Чтобы задать кодировку для отклика, установите свойство *ContentEncoding* объекта *Response* при помощи класса *Encoding* из пространства имен *System. Text*. То же самое можно сделать, используя значение свойства *Content Encoding* объекта *Request*. Так, следующий код генерирует отклик в кодировке *shift_JIS*:

Visual Basic .NET

```
Imports System.Text
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
   Response.ContentEncoding = Encoding.GetEncoding("shift_JIS")
   Response.Write("This response uses the character encoding: ")
```

```
    Response.Write(Response.ContentEncoding.WebName)
End Sub
```

Visual C#

```
using System.Text;
private void Page_Load(object sender. System.EventArgs e)
{
    Response.ContentEncoding = Encoding.GetEncoding("shift_JIS");
    Response.Write("This response uses the character encoding: ");
    Response.Write(Response.ContentEncoding.WebName);
}
```

Примечание Дополнительные сведения о применении различных схем кодировки символов можно найти в следующих разделах справочной системы:

«Unicode in the .NET Framework»

«Encoding Base Types»

«System.Text Namespace»

а также на страницах этих Web-сайтов:

http://www. unicode.org

http://www.w3. org/TR/REC-html40/charset.html

Резюме

- Для определения культуры пользователь применяет массив *UserLanguages* объекта *Request.*
- Чтобы создать Web-приложение, использующее определенную культуру, установите атрибуты *culture* и *uiCulture* для элемента *globalization* в файле Web.config.
- В .NET Framework культуры идентифицируют при помощи кодов языка и региона. Эти коды перечислены в разделе справочной системы, посвященному классу *CultureInfo.*
- Свойство *CurrentCulture* класса *Thread* позволяет назначить Web-приложению культуру, используемую для форматирования дат и чисел, а также определяющую порядок сортировки.
- Чтобы задать сопутствующую сборку для загрузки локализованных ресурсов, воспользуйтесь свойством *CurrentUICulture* класса *Thread.*
- Для создания поддержки отображения в HTML-элементах локализованных строк, загружаемых из файлов ресурсов во время выполнения, добавьте к этим элементам атрибуты *id* и *runat.*
- Имена элементов в файле ресурсов чувствительны к регистру символов и должны быть уникальными в пределах приложения, поэтому рекомендуется использовать для их именования следующую схему: *имя_формы.идентификатор_элемента.*
- Создав Web-форму, использующую символы, отличные от ASCII-символов, сохраните ее в кодировке UTF-8 с сигнатурой — это позволит ASP.NET автоматически определять кодировку файла формы.

Практикум 15. Создание конвертера валют

В этом практикуме вы создадите простое Web-приложение — конвертер валют. Оно будет определять тип преобразования, которое необходимо выполнить, на основе культуры пользователя по следующему алгоритму:

1. если культура не опознана, выполняется преобразование евро в доллары, при этом отображается англоязычный интерфейс как *интерфейс по умолчанию* (fallback interface);
2. если культура определена как en-US (Соединенные Штаты, английский язык) — выполняется преобразование долларов в евро или песо, отображается англоязычный интерфейс;
3. если культура определена как es-MX (Мексика, испанский язык) — выполняется преобразование песо в доллары или евро, отображается интерфейс на испанском языке.

Примерный вид готового Web-приложения показан на рис. 15-11.

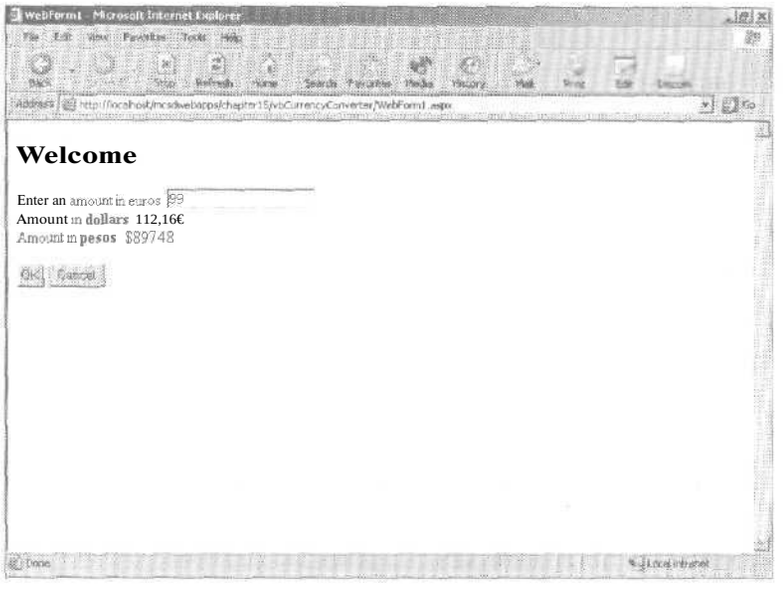


Рис. 15-11. Конвертер валют (с интерфейсом по умолчанию)

Продолжительность практикума — около 20 минут.

Упражнение 1. Создание Web-формы конвертера валют



Конвертер валют состоит из одной Web-формы, которая принимает данные через текстовое поле и по щелчку кнопки OK отображает результат в двух надписях На Web-форме также имеется HTML-кнопка Cancel — она служит для очистки текстового поля.

Поскольку вы создаете локализованное приложение, не забудьте снабдить HTML-элементы, расположенные на Web-форме, атрибутами *id u runat.* Это необходимо, чтобы серверный код мог загрузить в них строки локализованных ресурсов.

► Как создать Web-форму конвертера валют

1. Создайте новый проект Web-приложения и назовите его *CurrencyConverter.*

2. **Переключитесь в режим HTML и** введите следующий **HTML-текст** между **тегами** *<form>* и *</form>* Web-формы:

```
<h1 id="head1" runat="server">Head1</h1>
<span id="sp1" runat="server">Enter an amount: </span>
<asp:TextBox ID="txtCurrency" Runat="server" />
<br>
<span id="sp2" runat="server">Amount in: </span>
<asp:Label ID="lblCurrency1" Runat="server" />
<br>
<span id="sp3" runat="server">Amount in; </span>
<asp:Label ID="lblCurrency2" Runat="server" />
<br><br>
<asp:Button ID="butOK" Text="OK" Runat="server" />
<input id="butCancel" runat="server" type="button"
  onclick="txtCurrency.Text='' " value="Cancel">
```

3. Закончив, переключите Web-форму обратно в режим Design. Это важно, так как в результате в файле кода будут сгенерированы корректные объявления для созданных элементов управления.

Упражнение 2. Создание ресурсов для пользовательского интерфейса



Пока созданная вами Web-форма отображает вместо текста пользовательского интерфейса значения-заполнители, например, в заголовке отображается текст «Head1». В этом упражнении вы создадите строки, которые во время выполнения будут отображаться на месте заполнителей.

► Как создать файл ресурсов для пользовательского интерфейса

1. В меню Project выберите команду Add New Item, затем выберите из списка шаблонов Assembly Resource File. Укажите для нового файла ресурсов имя **strings.resx**.
2. Добавьте в файл ресурсов элементы, показанные на рис. 15-12, чтобы создать значения по умолчанию для элементов пользовательского интерфейса.

Data for data

	name	value	comment	type	roifnutype
▶	webform1.head1	Welcome	(null)	(null)	(null)
	webform1.sp1	Enter an amount in euros:	(nub	(nul)	(null)
	webform1.sp2	Amount in dollars:	(null)	(null)	(null)
	webform1.sp3	Amount in pesos:	(null)	(null)	(null)
	webform1.butOK	OK	(null)	(null)	(null)
	webform1.butCancel	Cancel	(null)	(nul)	(null;
*					

Рис. 15-12. Значения по умолчанию для элементов пользовательского интерфейса

3. Создайте новый файл ресурсов и назовите его **strings.en-US.resx**. Добавьте в него элементы, показанные на рис. 15-13, чтобы создать элементы интерфейса для пользователей с культурой en-US (США).

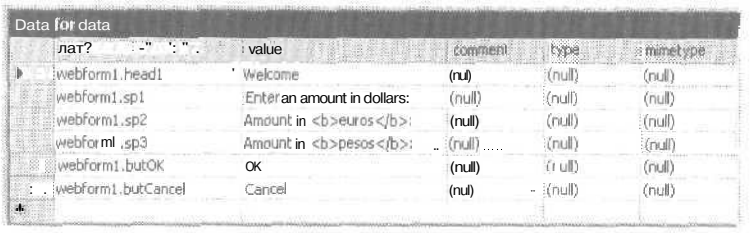


Рис. 15-13. Значения элементов интерфейса для культуры en-US

4. Создайте еще один файл ресурсов, назовите его **strings.es-MX.resx**. Добавьте в него элементы, показанные на рис. 15-14, — так вы создадите интерфейс для пользователей с культурой es-MX (Мексика).

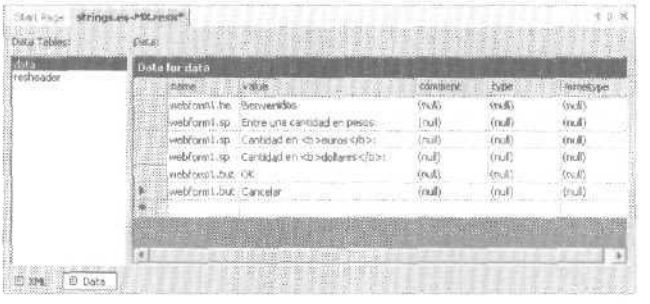


Рис. 15-14. Значения для культуры es-MX

5. Закончив, щелкните Save All, чтобы сохранить результаты работы,

При желании вы можете создать дополнительные файлы ресурсов доя других культур — для этого достаточно включить код культуры в имя файла ресурсов, как вы это делали ранее, При компоновке приложения Visual Studio .NET автоматически скомпилирует ресурсы в виде сопутствующих сборок и сохранит их в соответствующих каталогах.

Упражнение 3. Загрузка ресурсов, соответствующих культуре пользователя



Сейчас вы напишете код, который определяет культуру пользователя, загружает и отображает ресурсы, соответствующие этой культуре.

▶ Как загрузить ресурсы, соответствующие культуре пользователя

1. Поместите в начало файла кода Web-формы следующие операторы:

Visual Basic .NET

```
Imports System.Resources
Imports System.Globalization
Imports System.Threading
```

Visual C#

```
using System.Resources;
using System.Globalization;
using System.Threading;
```

2. Добавьте следующие объявления на уровне класса Web-формы — это необходимо для загрузки ресурсов в объект *Resource Manager:*

Visual Basic .NET

```
Protected gStrings As New ResourceManager("CurrencyConverter.strings", _
  GetType(WebForm1).Assembly)
```

Visual C#

```
protected ResourceManager gStrings = new
  ResourceManager("CurrencyConverter.strings", typeof(WebForm1).Assembly);
```

3. Добавьте к обработчику события *Page_Load* следующий код, который будет определять культуру пользователя, назначать соответствующую культуру потоку приложения и отображать локализованные строки, загруженные из файла ресурсов:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
   If Not IsPostBack Then
        ' Определить язык, предпочитаемый пользователем.
        Dim sLang As String - Request.UserLanguages(0)
        ' Установить значение свойства CurrentUICulture, чтобы задать
        ' сопутствующую сборку, из которой будут загружаться ресурсы.
        Thread.CurrentThread.CurrentUICulture = New CultureInfo(sLang)
        ' Назначить потоку культуру, определяющую
        ' форматирование дат, строк, денежных сумм и т.п.
        Thread.CurrentThread.CurrentCulture - _
          CultureInfoCreateSpecificCulture(sLang)
        ' Получить строки из файла ресурсов.
        head1.InnerHtml = gStrings.GetString("webform1.head1")
        sp1.InnerHtml - gStrings.GetString("webform1.sp1")
        sp2.InnerHtml - gStrings.GetString("webform1.sp2")
        sp3.InnerHtml - gStrings.GetString("webform1.sp3")
        butOK.Text = gStrings.GetString("webform1.butOK")
        butCancel.Value - gStrings.GetString("webform1.butCancel")
   End If
End Sub
```

Visual C#

```
private void Page_Load(object sender, System.EventArgs e)
{
   if (!IsPostBack)
   {
        // Определить язык, предпочитаемый пользователем.
        string sLang  - Request.UserLanguages[0];
        // Установить значение свойства CurrentUICulture, чтобы задать
        // сопутствующую сборку, из которой будут загружаться ресурсы.
```

```
        Thread.CurrentThread.CurrentUICulture = new CultureInfo(sLang);
        // Установить культуру потока для форматирования дат/
        строк/валюты и т.п.
        Thread.CurrentThread.CurrentCulture =
          CultureInfo.CreateSpecificCulture(sLang);
        // Назначить потоку культуру, определяющую
        // форматирование дат, строк, денежных сумм и т.п.
        head1.InnerHtml = gStrings.GetString("webform1.head1");
        sp1.InnerHtml = gStrings.GetString("webform1.sp1");
        sp2.InnerHtml = gStrings.GetString("webform1.sp2");
        sp3.InnerHtml = gStrings.GetString("webform1.sp3");
        butOK.Text = gStrings.GetString("webform1.butOK");
        butCancel.Value = gStrings.GetString("webform1.butCancel");
    }
}
```

Из этого кода ясно, что строки для локализованного пользовательского интерфейса загружаются только один раз — при первом отображении страницы, а ASP.NET автоматически сохраняет значения элементов управления между отображениями страницы.

Упражнение 4. Выполнение действий, зависящих от культуры



В нашем конвертере текущая культура определяет не только отображаемый интерфейс, но и выполняемые преобразования. Сейчас вы напишете кол, выполняющий преобразования в зависимости от культуры пользователя.

▶ Как выполнить преобразования в зависимости культуры

Добавьте в обработчик событий *butOK_Click* следующий код. который будет определять текущую культуру и выполнять различные преобразования в зависимости от результатов:

Visual Basic .NET

```
Private Sub butOK_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles butOK.Click
    ' Получить размер денежной суммы.
    Dim SrcAmount As Double = Convert.ToDouble(txtCurrency.Text)
    Dim DestAmount As Double
    ' Объявить переменные, представляющие культуры,
    ' они потребуются для форматирования валют.
    Dim Europe As New CultureInfo("fr-FR")
    Dim USA As New CultureInfo("en-US")
    Dim Mexico As New CultureInfo("es-MX")
    ' Выполнить преобразования в зависимости от текущей культуры
    ' пользовательского интерфейса.
    Select Case Thread.CurrentThread.CurrentCulture.Name
        Case "en-US"
            ' Конвертировать доллары,
            DestAmount = SrcAmount * 1.1329
            lblCurrency1.Text = DestAmount.ToString("C", Europe)
```

```
            DestAmount = SrcAmount * 9.0655
            lblCurrency2.Text = DestAmount.ToString("C", Mexico)
        Case "es-MX"
            ' Конвертировать песо.
            DestAmount - SrcAmount * 0.125
            lblCurrency1.Text = DestAmount.ToString("C", Europe)
            DestAmount - SrcAmount * 0.1103
            lblCurrency2.Text = DestAmount.ToString("C", USA)
        Case Else
            ' Конвертировать евро.
            DestAmount = SrcAmount * 8.0021
            lblCurrency1.Text = DestAmount.ToString("C", USA)
            DestAmount = SrcAmount * 0.1103
            lblCurrency2.Text = DestAmount.ToString("C", Mexico)
    End Select
End Sub
```

Visual C#

```
private void but()K_Click(object sender, System.EventArgs e)
{
    // Получить размер денежной суммы.
    double SrcAmount = Convert.ToDouble(txtCurrency.Text);
    double DestAmount;
    // Объявить переменные, представляющие культуры,
    // они потребуются для Форматирования валют,
    CultureInfo Europe = new CultureInfo("fr-FR");
    CultureInfo USA = new CultureInfo("en-US");
    CultureInfo Mexico = new CultureInfo("es-MX");
    // Выполнить преобразования в зависимости от текущей
    // культуры пользовательского интерфейса.
    switch (Thread.CurrentThread.CurrentCulture.Name)
    {
        case "en-US":
            // Конвертировать доллары.
            DestAmount = SrcAmount * 1.1329;
            lblCurrency1.Text = DestAmount.ToString("C", Europe);
            DestAmount = SrcAmount * 9.0655;
            lblCurrency2.Text = DestAmount.ToString("C", Mexico);
            break;
        case "es-MX":
            // Конвертировать песо,
            DestAmount - SrcAmount - 0.125:
            lblCurrency1.Text - DestAmount.ToString("C", Europe);
            DestAmount = SrcAmount * 0.1103;
            lblCurrency2.Text = DestAmount.ToString("C", USA);
            break;
```

```
            default:
                // Конвертировать евро.
                DestAmount = SrcAmount * 8.0021;
                lblCurrency1.Text = DestAmount.ToString("C", USA);
                DestAmount = SrcAmount * 0.1103;
                lblCurrency2.Text = DestAmount.ToString("C", Mexico);
                break;
            }
    }
```

Этот код создает объекты *CultureInfo*, представляющие европейскую, американскую и мексиканскую культуры и использует их для выполнения соответствующих преобразований. При помощи этих объектов задают форматирование значений денежных сумм при вызове метода *ToString*. Объект *Europe* использует код культуры для Франции, поскольку евро уже имеет хождение во Франции, а кода культуры для Евросоюза пока что не существует.

В этом примере курсы обмена валют «зашиты» в код, что, естественно, не годится для реального приложения. Для получения текущих курсов обмена как нельзя лучше подходят Web-сервисы XML, применение которых описано в главе 7.

Закрепление материала

Приведенные ниже вопросы помогут вам лучше усвоить основные темы данной главы. Если вы не сумеете ответить на вопрос, повторите материал соответствующего занятия. Ответы для самопроверки — в приложении «Вопросы и ответы» в конце книги.

1. Чем отличаются свойства *CurrentCulture* и *CurrentUICulture*?

2. Как определить культуру, к которой принадлежит пользователь?

3. Что такое «нейтральная культура»?

4. Как кодировка символов влияет на выбор формата для сохранения файла Web-формы?

ПРИЛОЖЕНИЕ

Вопросы и ответы

Глава 1. Введение в Web-программирование

1. Назовите три отличия Web-приложений ASP.NET от традиционных приложений для Windows.

   Основные отличия этих приложений приводятся в следующей таблице.

Предмет рассмотрения	Web-приложение ASP.NET	Неуправляемое Windows -приложение
Архитектура	Использует архитектуру «клиент — сервер»	Работает локально
Исполнение	Представляет собой управляемый код, исполняется под управлением общеязыковой исполняющей среды	Представляет собой неуправляемый код, исполняется под управлением Windows
Пользовательский интерфейс	Интерфейс состоит из Web-форм, отображаемых Интернет-браузером	Интерфейс отображается самим приложением при помощи Win32 API

2. Перечислите основные компоненты .NET Framework.
   - **Общеязыковая исполняющая среда.**
   - **Библиотека классов .NET Framework.**
3. Как в Visual Studio .NET восстановить параметры окон, заданные по умолчанию?

   В меню Tools выберите пункт Options\Reset Window Layout.
4. Почему Code Editor не сможет автоматически завершить следующую строку (на языке Visual C#)?

```
int intX - system.math
```

   Чтобы IntelliSense могла распознать ссылки на пространство имен *System.Math* в программах на языке Visual C#, необходимо записывать части его имени с заглавной буквы.
5. В каких случаях нельзя использовать ASP.NET для создания Web-приложений?

   При разработке Web-приложений, предназначенных для Web-серверов, работающих в ОС, отличной от Windows (например, для Web-сервера Apache под управлением ОС Linux).

Глава 2. Создание приложений Web Forms

1. Где в Visual Studio .NET хранятся проекты Web-приложений?

   Проекты Web-приложений хранятся в отдельных виртуальных каталогах, создаваемых для каждого нового проекта. Физическое размещение файлов проекта определяется свойствами виртуального каталога, которые можно просматривать с помощью IIS.

2. Перечислите основные отличия Web- и Windows-приложений,
   - Стандартные элементы управления Windows нельзя применять с Web-формами — в этом случае годятся серверные и HTML-элементы управления, а также пользовательские или нестандартные элементы управления, специально разработанные для Web-форм.
   - Web-приложения отображаются в окне браузера; Windows-приложения отображают собственные окна, внешний вид и поведение которых они контролируют практически полностью.
   - Web-формы создаются на сервере, затем отсылаются браузеру н сразу после этого уничтожаются. В отличие от них, Windows-формы существуют столько, сколько необходимо программе, и уничтожаются, только когда становятся ненужными.
   - Web-приложения работают на сервере, а их интерфейс отображается на удаленных клиентских компьютерах; исполнение же Windows-приложений и отображение их интерфейса имеет место на одном и том же компьютере.

3. Опишите жизненный цикл Web-приложения. Когда создаются Web-формы и как долго они существуют?

   Работа Web-приложения начинается при первом запросе ресурса, расположенного в пределах его границ; Web-форма создается при ее запросе и уничтожается сразу после отправки клиенту отклика, сгенерированного в результате ее обработки на сервере. Web-приложение завершается с завершением сеанса последнего работающего с ним клиента.

4. Как сохранить данные (например, значение простой переменной) в Web-приложении?

   Это можно сделать при помощи переменных состояния, таких, как *Application State*, *SessionState* или *ViewState*.

5. Что определяет границы Web-приложения?

   Границы Web-приложения определяются структурой виртуальных каталогов Web-приложения в IIS: они начинаются в каталоге, содержащем начальную страницу приложения, и заканчиваются в последнем каталоге, вложенном в него, либо в каталоге с другой начальной страницей.

Глава 3. Работа с Web-объектами

1. Каким образом упорядочены классы .NET Framework?

   Классы .NET Framework упорядочены при помощи пространств имен.

2. Чем различаются модули класса и кода в Visual Basic .NET?

   Во время выполнения создаются экземпляры объектов, объявленных в модулях классов. У экземпляров объектов, созданных таким образом, все значения переменных и свойств хранятся в отдельных областях памяти. У модулей кода не существует экземпляров, поэтому все переменные, объявленные на уровне модуля, доступны любым процедурам из этого модуля.

3. Как в Visual C# объявить метод, чтобы он был доступен без предварительного создания экземпляра этого класса?

   Чтобы создать метод, доступный без предварительного создания экземпляра класса, необходимо объявить его с модификатором *static*.

4. Как вызвать член базового класса из производного класса?

   Для вызова членов базового класса в Visual Basic .NET необходимо ключевое слово *MyBase*, а в Visual C# — ключевое слово *base*.

5. Назовите четыре основных объекта, используемых при разработке Web-приложений.
   Это объекты *Application*, *Page*, *Request* и *Response*.
6. Куда следует поместить следующие элементы данных, чтобы сохранить их между запросами к Web-форме?
   - Элемент управления, создаваемый во время выполнения.
     Элементы управления, созданные во время выполнения, следует сохранять в состоянии отображения (в поле *View State*) объекта *Page*.
   - Объект, необходимый всем пользователям приложения.
     Этот объект следует хранить в переменной состояния на основе объекта *Application*.
   - Личные настройки пользователя.
     Их сохраняют в переменной состояния на основе объекта *Session*.

Глава 4. Создание пользовательского интерфейса

1. В чем главное отличие между серверным элементом управления Button и одноименным HTML-элементом управления?
   В ответ на щелчок серверный элемент управления Button инициирует исполнение на сервере обработчика события ASP.NET *Click*. Щелчок HTML-элемента управления Button запускает обработчик, указанный в атрибуте *onclick* этого элемента управления, причем этот обработчик исполняется на клиенте.
2. Как заставить несколько элементов управления RadioButton, размещенных на Web-форме взаимодействовать, чтобы одновременно удавалось установить только один из них?
   Необходимо задать одинаковые значения для свойства *GroupName* каждого из этих элементов управления.
3. Почему ASP.NET проверяет вводимые данные как на клиентской, так и на серверной стороне?
   Проверка на стороне клиента помогает избежать излишних обменов данными с сервером, поскольку она, как правило, позволяет убедиться в допустимости введенных данных до их отправки на сервер. Проверка данных на клиенте может быть отключена пользователем либо взломана злоумышленником, поэтому требуется повторная проверка на серверной стороне. Это гарантирует получение сервером корректных данных, в то же время позволяя избежать излишнего обмена данными с сервером.
4. Какие верификаторы следует использовать, чтобы убедиться, что клиент ввел допустимый номер телефона?
   Для этого применяют верификаторы RequiredFieldValidator и RegularExpressionValidator. Если имеется список допустимых номеров, можно заменить RegularExpressionValidator на CustomValidator, который сверяет введенные данные со списком.
5. Найдите ошибку в следующем коде:

   Visual Basic .NET

   ```
   Server.Transfer("Default.htm")
   ```

   Visual C#

   ```
   Server.Transfer("Default.htm");
   ```

   Метод *Transfer* нельзя применять с HTML-страницами, он работает только со .aspx-страницами.
6. Почему нельзя открыть новое окно браузера из серверного кода?
   Серверный код исполняется на сервере, тогда как новое окно необходимо создать на стороне клиента. Чтобы исполнить действие на клиенте, например загрузить файл, открыть новое окно или перейти к предыдущему элементу списка history, применяют код, работающий на стороне клиента.

Глава 5. Хранение и извлечение данных с помощью ADO.NET

1. Как и с помощью каких объектов удается быстро подсчитать число записей в таблице базы данных?

   Эта задача решается двумя способами:

   - **выполнением команды SQL, возвращающей число строк таблицы при помощи объектов соединения с базой данных и команды;**
   - **созданием набора данных на основе этой таблицы и подсчетом числа его строк (при помощи объектов соединения и адаптера данных).**

2. Назовите отличие типизированных наборов данных от нетипизированных; в чем преимущества типизированных наборов данных?

   На члены типизированных наборов данных разрешается ссылаться явно, используя их типы данных и имена, тогда как для ссылки на члены нетипизированных наборов данных применяются наборы, например:

   Visual Basic .NET

   ```
   ' Это типизированная ссылка на столбец HomePhone таблицы Contacts.
   DataSet1.Contacts.HomePhoneColumn.Caption = "@Home"
   ' А это - нетипизированная ссылка на тот же столбец,
   DataSet1.Tables("Contacts").Columns("HomePhone").Caption = "@Home"
   ```

   Visual C#

   ```
   // Это типизированная ссылка на столбец HomePhone таблицы Contacts,
   DataSet1.Contacts.HomePhoneColumn.Caption = "@Home";
   // А это - нетипизированная ссылка на тот же столбец.
   DataSet1.Tables["Contacts"].Columns["HomePhone"].Caption = "@Home";
   ```

   Типизированные наборы данных поддерживают проверку ошибок во время разработки, что позволяет выявлять ошибки, вызванные опечатками и несоответствием типов, в случае нетипизированных наборов данных эти ошибки проявляются только во время выполнения.

3. Как сделать, чтобы все формы Web-приложения использовали общие компоненты для соединения с БД?

   Чтобы сделать соединение доступным всем Web-формам приложения, добавьте объект соединения с базой данных в файл Global.asax, затем объявите переменную состояния в обработчике события *Application_Start* (или *Session_Start*).

4. Объясните разницу между обработкой транзакций на уровнях набора данных и БД.

   Наборы данных поддерживают неявные транзакции, поскольку изменения в наборе данных не будут перенесены в базу данных до вызова метода *Update*. Для обработки транзакций в наборе данных применяют метод *Update* с проверкой наличия ошибок. Если исполнение метода *Update* генерирует ошибку, перенос модификаций в базу данных не выполняется. При этом стоит попытаться выполнить модификацию базы данных еще раз, исправив ошибку, либо отменить их, вызывав метод *RejectChanges*.

   Базы данных поддерживают явные транзакции посредством объекта *Transaction*. Этот объект создается на основе объекта соединения с базой данных и назначается всем командам, которые нужно включить в транзакцию, через свойство *Transaction* объектов этих команд. После выполнения команд базу данных проверяют на наличие ошибок. Если таковые обнаружены, можно попытаться исправить их, выполнив команду повторно, либо вернуть базу данных в исходное состояние при помощи метода *RollBack* объекта *Transaction*; в противном случае стоит сделать изменения постоянными, вызвав метод *Commit* объекта *Transaction*.

Глава 6. Перехват и исправление ошибок

1. Почему так важно наличие обработки исключений в готовом приложении?

   Возникновение необработанного исключения в приложении приводит к его краху, при этом пользователю не удается продолжать работу и он теряет результаты, полученные непосредственно перед возникновением исключения. Обработка исключений позволяет перехватывать непредвиденные ситуации, чреватые сбоями.

2. Назовите два способа обработки исключений в Web-приложениях ASP.NET.

   Первый способ обработки исключений — применение специальных структур (блоков), задаваемых ключевыми словами *Try*, *Catch* и *Finally* (в Visual Basic .NET) или *try*, *catch* и *finally* (в Visual C#). Второй способ — использование событий *Error*, позволяющих обрабатывать исключения на уровне страницы, приложения или глобально при помощи методов *GetLastError* и *ClearError* объекта *Server*.

3. Расскажите, для чего нужны страницы сообщений об ошибках и как их использовать.

   Так как Web-приложения работают в Интернете, иногда исключительные ситуации возникают вне области видимости приложения. В этих случаях приложению не удается непосредственно отреагировать на исключения. Подобные исключения обозначаются кодами состояния HTTP, которые IIS обрабатывает, отображая нестандартные страницы сообщений об ошибках, заданные в файле Web.config приложения.

4. Объясните пользу трассировки при обработке исключений.

   Трассировка позволяет незаметно для пользователя регистрировать необычные обстоятельства, возникающие во время работы приложения. При возникновении непредвиденного события приложение может записать в журнал трассировки соответствующее сообщение, которое будет полезно при диагностике и тестировании приложения после его развертывания.

Глава 7. Усложненное программирование с применением Web Forms

1. Напишите HTML-код для гиперссылки, которая будет отправлять сообщение по щелчку пользователя.

```
<a href="mailto:you@microsoft.com?SUBJECT=Sending from a client&BODY=Some
message text.">Send mail</a>
```

2. Напишите код, записывающий на пользовательский компьютер файл cookie с именем пользователя (Rob Young) и текущей датой, который хранится на компьютере пользователя в течение 30 дней.

Visual Basic .NET

```
Dint cookUserInfo As New HttpCookie("UserInfo")
cookUserInfo("Name") = "Rob Young"
cookUserInfo("Time") = DateTime.Now.ToString()
cookUserInfo.Expires - DateTime.Now.AddDays(30)
Response.Cookies.Add(cookUserInfo)
```

Visual C#

```
HttpCookie cookUserInfo = new HttpCookie("UserInfo")
CookUserInfo["Name"] = "Rob Young"
CookUserInfo["Time"] = DateTime.Now.ToString()
cookUserInfo.Expires = DateTime.Now.AddDays(30)
Response.Cookies.Add(cookUserInfo)
```

3. Почему нельзя открыть новое окно браузера из серверного кода? Как открыть страницу в новом окне из клиентского сценария?

   При помощи серверного кода удается выполнять действия только на сервере. Чтобы выполнить какое-либо действие на клиентском компьютере, например открыть новое окно браузера, необходимо воспользоваться кодом, исполняемым на клиенте.

   Новое окно браузера можно открыть посредством DOM-объекта *window*. Вот пример HTML-элемента управления Button, открывающего страницу справочной системы в новом окне:

```
<button id="butHelp" onclick="window.open('help.aspx', 'help',
'height=200,width=300')">Help</button>
```

4. Как отобразить страницу в одном фрейме, щелкнув гиперссылку в другом фрейме?

   Чтобы задать фрейм, в котором следует открыть страницу, определите атрибут *target* элемента *<a>*. Вот пример гиперссылки, отображающей страницу в главном фрейме набора фреймов:

```
<a href="AllTheAnswers.aspx" target="main">Show the answers!</a>
```

Глава 8. Обеспечение безопасности

1. Какой режим аутентификации ASP.NET лучше всего подходит для идентификации и авторизации пользователей корпоративной сети?

   Для этой цели лучше всего подходит аутентификация при помощи встроенных механизмов Windows, поскольку она выполняется с обращением к учетным записям и разрешениям, уже назначенным этим пользователям в корпоративной сети.

2. Чем отличаются списки со сведениями о пользователях из файла Web.config, применяемые для аутентификации средствами Windows и аутентификации с помощью форм?

   Список пользователей, аутентифицируемых средствами Windows, помещают в элемент *<authorization>* файла Web.config, а списки пользователей, аутентифицируемых при помощи форм, хранят в элементе *<credentials>* файла Web.config либо во внешнем файле или базе данных.

3. Как заставить всех пользователей проходить аутентификацию с помощью файла Web.config независимо от режима аутентификации ASP.NET?

   Для этого достаточно определить в файле Web.config следующий элемент *<authorization>*:

```
<authorization>
   <deny users="?" />
</authorization>
```

4. Каким образом использование протокола SSL обеспечивает безопасность Web-приложения?

   Протокол SSL защищает данные, которыми обмениваются приложения, шифруя их перед их пересылкой через Интернет.

5. Как инициировать и завершить взаимодействие, защищенное протоколом SSL?

   Чтобы начать защищенное взаимодействие, укажите в адресе применение протокола HTTPS, например, так:

```
<a href="https://www.contoso.com/secure.htm">Secure page.</a>
```

   Чтобы завершить защищенное взаимодействие, укажите в адресе применение протокола HTTP, например, так:

```
<a href="http://www.contoso.com/Default.htm">Not secure.</a>
```

Глава 9. Компоновка и развертывание Web-приложений

1. Какие разрешения назначаются Web-приложениям по умолчанию?

   По умолчанию Web-приложение работает под учетной записью ASPNET, обладающей ограниченными правами, которые совпадают с правами группы Users.

2. Почему файл Machine.config так важен для развертывания Web-приложений?

   Файл Machine.config управляет многими аспектами работы Web-приложений, включая рециркуляцию процессов, ограничение очереди запросов и частоту проверки соединения с пользователем.

3. Как задать параметры кэширования для Web-приложения?

   Параметры кэширования страниц Web-приложения задают **в IIS при помощи FrontPage** Server **Extensions.**

4. Опишите особенности применения переменных состояния *Session в* Web-приложениях, работающих на Web-фермах и Web-садах.

   Для Web-приложений, **развертываемых на Web-фермах или Web-садах, необходимо указать в** файле **Web.config механизм** доступа **к переменным состояния *Session,* поскольку на протяжении** сеанса **клиента могут обслуживать несколько разных процессов** Web-приложения. **Такой механизм предоставляет доступ к переменным состояния *Session* всем процессам Web-приложения.**

Глава 10. Тестирование Web-приложений

1. **Опишите связи между блочным, комплексным и регрессивным тестированием.**

   Блочное тестирование составляет основу всего процесса тестирования, оно проверяет работоспособность всех фрагментов кода. Комплексное тестирование дополняет его, проверяя корректность совместной работы отдельных фрагментов кода. Регрессивное тестирование проводится многократно согласно плану тестирования. В нем используются задания, созданные ранее для блочного и комплексного тестирования. Цель регрессивного тестирования — выявить нарушения в работе существующего кода, вызванные добавлением новых компонентов.

2. Почему нагрузочное тестирование, как правило, более важно для Web-приложения, чем для автономного Windows-приложения?

   Поскольку Web-приложения, опубликованные в Интернете, доступны **всем желающим, число их пользователей может достигать тысяч и** даже **миллионов. Нагрузочное тестирование позволяет моделировать подобные ситуации для прогнозирования потребности в ресурсах, а также выявления конфликтов из-за ресурсов, «узких»** мест и других **потенциальных проблем, которые невозможно обнаружить в тестах, выполняемых** единичным пользователем.

3. Чем отличаются классы *Debug* и *Trace*?

   При задании стандартных параметров окружения, во время компиляции окончательной версии приложения код, использующий класс ***Debug*, удаляется из нее, а код, использующий** класс ***Trace,* — остается.**

4. Какие два действия нужно выполнить, чтобы сделать компонент, расположенный в .NET-сборке, доступным компоненту COM?

   - **Сначала** следует **зарегистрировать .NET-сборку в системном реестре с помощью утилиты RegAsm.exe.**
   - **Далее необходимо принять меры, чтобы компоненту COM удалось найти .NET-сборку. Для** этого **нужно поместить** сборку **в глобальный кэш сборок или в каталог компонента COM либо выполнить иные действия, которые** сделают **NET-сборку** доступной **компоненту COM.**

Глава 11. Создание нестандартных Web-элементов управления

1. Кратко опишите основные области применения пользовательских, составных и генерируемых элементов управления.
   - **Пользовательские** элементы управления **применяются для** оперативного **объединения нескольких** элементов **управления в группу, способную** решать некоторую **задачу, с целью** многократного **использования** этой **группы в текущем проекте.**
   - **Составные элементы управления позволяют объединять существующие** элементы **управления и компилировать их в сборку, доступную для многократного** использования **в различных** проектах.
   - Генерируемые **элементы управления применяются для создания**совершенно **новых элементов** управления, **которые можно компилировать** в сборку **и многократно** использовать **в** различных проектах.
2. Почему наследование так важно для создания нестандартных Web-элементов управления?

   Составные и генерируемые элементы **управления объявляются** на основе **класса** *WebControl*, **от которого они** наследуют методы. **Последние в** дальнейшем **переопределяют с** целью **создания внешнего вида нестандартных элементов управления.**
3. Назовите главный метод, переопределяемый при создании составных элементов управления.

   Метод *CreateChildControls***перегружают для добавления к составному элементу управления** существующих **элементов управления.**
4. Назовите главный метод, переопределяемый при создании генерируемых элементов управления.

   Метод ***Render*** **переопределяют при создании генерируемых элементов управления.**
5. Чем отличается инициирование событий postback в составных и генерируемых элементах управления?

   События postback **в составном элементе управления может** инициировать **один из** то **дочерних** элементов **управления, тогда как в генерируемом элементе управления** для **этого необходимо реализовать интерфейс** *IPostBackEventHandler* **и написать клиентский сценарий, генерирующий событие** postback.

Глава 12. Работа с мультимедиа

1. Почему для управления мультимедиа, как правило, используют клиентские сценарии, а не серверный кол?

   Обработка событий на сервере требует перезагрузки страницы, что приводит к перезапуску воспроизведения большинства **типов мультимедийных данных, в том числе звука и** видео.
2. Перечислите HTML-элементы для работы со звуком и видео, поддерживаемые большинством браузеров.

   Это HTML-элементы *bgsound*, *img* и ***embed.***
3. Что нужно, чтобы организовать поддержку браузера Netscape Navigator в HTML-элементе объекта ActiveX?

   Для этого необходимо создать в нем вложенный HTML-элемент ***embed*** **с** ActiveX-**объектом** соответствующего **подключаемого модуля для Netscape** Navigator, **внедренного при помощи** элемента **HTML-элемента** *object.* **Если на клиенте используется** Netscape **Navigator,** возможно, **также потребуется модифицировать или отключить** клиентский **код, управляющий ActiveX-объектом.**
4. Назовите ключевую возможность стандарта HTML+TIME, не доступную при использовании других методов анимации.

   Элементы HTML+TIME позволяют **синхронизировать события,** генерируемые на **странице.**
5. Опишите два способа создания временной шкалы при использовании HTML+TIME.

- Первый — при помощи одного из элементов, определяющих временные шкалы, например *t:SEQ*, *t:EXCL* или *t:PAR*.
- Второй — путем добавления к стандартным HTML-элементам атрибута *TIMECONTAINER*.

Глава 13. Форматирование данных, генерируемых Web-приложением

1. В чем преимущество использования каскадных таблиц стилей перед локальными стилями при форматировании Web-приложений?

   Применение CSS позволяет хранить сведения о форматировании отдельно от содержимого Web-форм, что упрощает изменение внешнего вида Web-приложения и поддержание его согласованности.

2. Почему лучше создавать стили для форматирования классов, а не отдельных HTML-элементов?

   Классы стилей позволяют назначать одинаковый формат разным типам элементов Web-приложения. Например, таким образом создают класс стиля, назначающий одинаково броское форматирование тексту, элементам управления, заголовкам и графическим элементам, к которым следует привлечь внимание пользователя.

3. Опишите взаимосвязь CSS и XSL при форматировании Web-приложений.

   Применение CSS и XSL — взаимодополняющие методики форматирования элементов Web-приложения. CSS определяют шрифт, цвет, размер, фон и другие параметры форматирования HTML-элементов Web-страницы, а XSL применяется для преобразования XML-файлов в HTML-документы. Таким образом, XSL управляет положением и видом элементов страниц на основе их содержимого. При использовании совместно с CSS XSL решает задачи «высокого уровня», то есть определяет компоновку и разметку страниц, тогда как на долю CSS выпадает решение «низкоуровневых» задач типа определения шрифта, цвета и других особенностей внешнего вида элемента.

4. Чем отличаются языки HTML и XML?
   - В HTML используются предопределенные имена элементов, например *<p>*, *<hr>* *и* др., а в XML имена элементов определяются разработчиком с целью создания иерархической структуры, упорядочивающей данные.
   - У XML более строгий синтаксис, чем у HTML: для XML-элементов обязательны замыкающие теги, имена элементов чувствительны к регистру, значения атрибутов необходимо заключать в кавычки, а каждый вложенный элемент необходимо закрыть прежде его родительского элемента.
   - **Концептуальное отличие HTML от XML заключается в том, что в XML данные идентифицируются по значению, а не по типу применяемого к ним форматирования.**

Глава 14. Создание справочной системы

1. В какой HTML-атрибут, генерируемый ASP.NET, во время выполнения записывается значение свойства *Tooltip* серверного элемента управления?

   Значение свойства *Tooltip* во время выполнения записывается в атрибут *title*.

2. Чем отличается отображение справочных файлов в формате HTML с помощью метода *showHelp* объекта *window* и в окне браузера?

   Метод *showHelp* «не умеет» открывать разделы компилированных справочных файлов в формате HTML Help по сетевым адресам. Чтобы открыть раздел такого файла при помощи метода *showHelp*, необходимо целиком загрузить этот файл на пользовательский компьютер. Протокол *ms-its:* позволяет отображать в окне браузера разделы справочных фай-

лов HTML Help, заданные сетевыми адресами, но при этом остается недоступным ряд возможностей справочной системы, таких, как оглавление, указатель и средства поиска.

3. Как определить в разделе справочной системы ключевые слова для предметного указателя?

   Для этого к разделу нужно добавить элемент *object* с ActiveX-элементом управления. Вот пример элемента *object,* добавляющего к указателю ключевое слово «numbers»:

```
<Object type="application/x-oleobject" classid="clsid:1e2a7bd0-dab9-11d0-
b93a-00c04fc99f9e">
   <param name="Keyword" value="numbers">
</OBJECT>
```

Глава 15. Глобализация Web-приложений

1. Чем отличаются свойства *CurrentCulture* и *CurrentUICulture*?

   Свойство *CurrentCulture* определяет, как .NET Framework будет форматировать даты, значения денежных сумм и т. п., а свойство *CurrentUICulture* задает сопутствующую сборку, из которой будут загружаться ресурсы.

2. Как определить культуру, к которой принадлежит пользователь?

   Для этого применяют массив *User Languages* объекта *Request.* Элемент этого мас ива с индексом 0 возвращает значение, которое соответствует одному из кодов культур, используемых классом *CultureInfo*, например:

```
SLang = Request.UserLanguages(0)
```

3. Что такое «нейтральная культура»?

   К «нейтральным культурам» относятся языки межнационального общения, в которых не определены другие региональные параметры, такие, как формат значений денежных сумм. Примером может служить код культуры «es», представляющей испанский язык.

4. Как кодировка символов влияет на выбор формата для сохранения файла Web-формы?

   Если Web-форма содержит символы, отсутствующие в ASCII, ее файл необходимо сохранять в подходящей кодировке, например в UTF-8. Если файл Web-формы сохранен с сигнатурой, ASP.NET сможет автоматически распознать кодировку UTF-8; в противном случае следует указать кодировку файла в атрибуте *fileEncoding* элемента *globalization,* определяемом в файле Web.config.

Предметный указатель

.NET Framework 5, 9
— выражение *formattingexpression* 211
— исключения 260
— просмотр типов исключений 260
— структура библиотеки классов 11

A

ACID-критерии 223
ACT (Application Center Test) *5*
ADO.NET 192
— доступ к данным 192
— исполнение команд в БД напрямую 218
— методы исполнения команд 218
— пространство имен 193
— средства для обработки транзакций 224
— типы соединений с БД 193
Application Center Test *см.* ACT
ASP.NET 5
— аутентификация пользователей
— — методы 342
— — способ 342
— компоненты 5
— предоставление анонимного доступа 340
— страницы сообщений об ошибках 267
— форматирование выходных данных 555
authentication *см.* аутентификация
authorization *см.* авторизация

C

caching *см.* кэширование
Cascading Style Sheets *см.* CSS
Certificate Authority *см.* центр сертификации
CLR (common language runtime) 9
common language runtime *СМ.* **CLR**
composite control *см.* составной элемент управления
CSS (Cascading Style Sheets) 556 *см. также* каскадные таблицы стилей
cultures си. культура

D

data binding *см.* привязка данных
delegation *см.* делегирование
deploying *см.* развертывание приложения
Document Object Model *см.* DOM
DOM (Document Object Model) 157, 310

E

embedded plug-in *см.* внедренный модуль
encoding *см.* кодировка
error си. ошибка
error pages *см,* страницы сообщений об ошибках
exception *см.* исключение
exception handling *см.* обработка исключений
exception log *см.* журнал исключений

F

flow layout *см.* потоковая разметка

G

global assembly cache *см.* кэш глобальных сборок
globalization си. глобализация
grid layout *см.* сеточная разметка
GUID *см.* глобально уникальный идентификатор

H

HTML Help Workshop 595
— интерфейс 596
— компиляция 606
— оглавление
— — редатирование 599
— — создание 598
— организация поддержки поиска 606
— параметры проекта 597
— перекрестные ссылки 603
— — ассоциативные ссылки 604
— — на смежные разделы 603
— просмотр 606
— создание справочной системы 595
— указатель 602
— файлы проекта 595
HTML-элемент
— *bgsound* 522
— *embed* 522, 523
— *img* 532
— *marquee* 538, 539
- *object* 522, 525

I

image map *см.* карта изображения
impersonation *см.* олицетворение

L

localization *см.* локализация

M

managed code *см.* управляемый код
master timeline *см.* временная шкала, главная
Microsoft intermediate language *см.* MSIL
MSIL (Microsoft intermediate language) 10
multiple inheritance *см.* концепция множественного наследования

N

namespace collisions *см.* коллизии пространств имен

O

optimization *см.* оптимизация приложения

P

peer-to-peer *см.* одноранговое приложение
process recycling *см.* рециркуляция процессов

R

rendered control *см.* генерируемый элемент управления

S

satellite assemblies *см.* сопутствующая сборка
save point *см.* контрольная точка
scalability *см.* масштабируемость приложения
Secure Sockets Layer *см.* SSL
server controls *см.* серверные элементы управления
session *см.* сеанс
shared methods *см.* общие методы
SMIL (Synchronized Multimedia Integration Language) 544
SSL (Secure Sockets Layer) 368
start-up project *см.* стартовый проект
static methods *см.* статические методы
structured exception handling aw. структурная обработка исключений
stub *см.* заглушка
Synchronized Multimedia Integration Language *СМ.* SMIL

T

template *см.* шаблон
thread *см.* поток
timeline *см.* временная шкала
ToolTip *см.* всплывающая подсказка
tracing *см.* трассировка
transaction *см.* транзакция
tuning *см.* настройка приложения

U

unhandled exceptions *см.* необработанные исключения
user control *см.* пользовательский элемент управления

V

validation control *см.* верификатор
virtual folder *см.* виртуальный каталог

W

W3C (World Wide Web Consortium) 544
Web farm *СМ.* Web-ферма
Web form *СМ.* Web-форма
Web garden *СМ.* Web-сад
Web-приложение 1, 2, 3, 4, 5, 6. 7, 8, 10. 13, 17, 26, 31, 34, 35, 36, 40, 41, 42, 43, 46, 47, 48, 49, 50, 51, 52, 55, 57, 61, 62, 63, 64, 65, 66, 67, 74, 75, 101, 102, 103, 116, 117, 124, 131, 172, 174, 223, 224, 252 ***см.*** *также* Интернет-приложение
— ввод сведений о нем 389
— границы 61
— доступ к БД SQL 196
— идентификация пользователя 286
— изменение уровня защиты 413
— компоненты 7
— компоновка 388
— конфигурирование 391
— копирование 394
— назначение дополнительных разрешений 396
— отправка почты 298
— подготовка виртуального каталога 394
— разделение 62
— регулировка производительности 41 2
— создание 234
— — виртуального каталога 43
— — многостраничного Web-приложения 234
— — проекта 42, 66
— способ исполнения на сервере 63
— схема взаимодействия частей 8
— трассировка 272
— удаленная отладка 456
— установка
— — на сервер службы хостинга 393
— — на собственном сервере 393
— — разделяемых компонентов 395
— устранение неполадок 411
— шаблон 42
Web-сад 416, 417
Web-сервис 2, 3, 5, 12, 31, 49, 101

Web-сервисы XML 319
— применение 320
— в клиентских сценариях 321
-реестр UDDI: 319
Web-ферма 416
Web-форма 4, 7, 8, 9, 16, 21, 22, 23, 24, 31, 32. 42, 47, 48, 49, 50, 52, 53, 54, 57, 59, 65, 66, 67, 71, 74, 87, 89, 101, 104, 109, 115, 121, 126, 131, 136, 137, 139, 147, 149, 153, 166, 167, 168, 172, 173, 174, 175, 177, 178, 180, 181, 184, 185, 187, 189, 191, 198, 209, 211, 214, 215, 218, 232, 233, 234, 235, 236, 237, 238, 239, 243, 245, 247, 252, 256, 265, 268, 270, 273, 278, 281, 283, 284, 296, 303, 305, 314, 328, 329, 332, 333, 336, 343, 345, 352, 354, 355, 357, 375, 376, 385, 415, 434, 458, 464, 465, 466, 467, 468, 470, 472, 474, 482. 483, 484. 491, 493, 497, 503, 506, 508, 517, 521, 522, 523, 532, 533, 551, 552, 555, 556, 558, 559, 560, 562, 563, 564, 565, 566, 567, 568. 570, 584, 585, 587, 589, 590, 591, 593, 594, 609, 615, 616, 620, 630, 631, 632, 634, 635, 637, 639, 643, 644, 645, 646
— включение и выключение трассировки приложения 272
— воспроизведение видео
— — в виде объекта ActiveX 534
— — способы 532
— гиперссылки и методы 164
— графика 147
— для аутентификации 352
— — входная 354
— добавление объектов ActiveX 524
— добавление пользовательского элемента управления 470
— жизненный цикл 224
— загрузка ресурсов из сопутствующей сборки 631
— задание форматируемого элемента при помощи идентификатора 561
— использование наборов данных 209
— использование элемента управления DataGrid 209
— компоненты 9
— наборы фреймов 302
— переходы между формами 125
— подключение справочной системы 607
— применение HTML+TIME 544
— применение таблиц стилей 558
— проверка вводимых данных 176
— работа со службой DTC 232
— создание для ввода имен пользователей 352
— создание формы Mail 331
— сохранение данных 109
— тестирование 429
— элементы управления 126, 128
— — группировка 149
— — разметка 126
добавление клиентского сценария 308
объединение нескольких 168
World Wide Web Consortium *см.* W3C

X

XML Path Language си. XPath
XML-схема 293
XML-файл 292, 570
— базовые элементы 573
— выполнение повторяющихся действий 578
— выполнение условных действий 581
— генерация оглавления 581
— добавление гиперссылок 579
— изменение разметки страницы 574
— правила 570
— совместное использование нескольких шаблонов 577
— создание 572, 573
— — дополнительных шаблонов 575
— сортировка элементов 581
— структура 572
— узлы 572
XPath (XML Path Language) 570
XSL-преобразование 570, 587

А

абстрактный класс 94
автоматизированное тестирование *см.* тестирование
авторизация 340
— основанная на ролях 348
анимация 538
— простых графических элементов 541
— с помощью HTML+TIME 544
— с помощью атрибутов 548
— — *PATH* 549
— — *span* 548
— — *VALUES* 549
— с помощью клиентских сценариев 542
— с помощью объектов и подключаемых модулей 542
— способы 538
— текста 539
анонимный доступ 340
аутентификация 340
- Passport 363
— — включение 363
-Windows 345, 347
— — включение 345
— — и параметры IIS 350
— — извлечение идентификационных данных пользователя 349

— — назначение прав доступа отдельным пользователям 348
— запросов HTML-страниц 343
— пользователей 342
— с помощью базы данных 357
— установка режима 375

Б

блок
— *Class...End Class* 83
- *Module...End Module* 83
- *Try...Catch/try...catch* 189

В

верификатор 156
виртуальный каталог 43
— subweb 43
— этапы создания 43
внедренный модуль 529
воспроизведение видео
— способы добавления видеоданных 532
воспроизведение звука
— добавление в виде объектов ActiveX 524
— способы 522
— фонового 522
— — аддитивность 523
— — управление 523
временная шкала 544
— главная 547
— для отдельных HTML-элементов 548
— монопольная 546
— параллельная 547
всплывающая подсказка 590
выбор провайдера услуг хостинга 35
вызов кода из пространства имен в приложениях 78

Г

генерируемый элемент управления 464, 498
— прием данных от пользователя 511
— создание 498
глобализация 617
— нестандартные верификаторы 639
— сортировка и сравнение строк 639
— способы 618
глобально уникальный идентификатор 291

Д

делегирование 475
директива *Register* 482
добавление элементов управления к пользовательскому интерфейсу 66
добавление, модификация и удаление строк 201

Ж

журнал
— исключений 272
— трассировки
— — запись сообщений 274
— — создание и просмотр 283
— — чтение 276

З

заглушка 429
замещение члена базового класса 89
защищенное подключение 372

И

извлечение значений из базы данных 219
извлечение сведений о пользователе из набора данных 296
инструменты Visual Studio .NET 5
Интернет-приложение 1, 2, 3, 31, 639
интерфейс 98
- *IFigure* 99
— *INamingContainer* 503
— *IPostBackDataHandler* 511
- *IPostBackEventHandler* 508, 510
исключение 254
— *ApplicationException* 262
— *Exception* 259
— *FileNotFoundException* 259
— *System.IO.FileNotFound* 259
— *System.IO.FileNotFoundException* 258
— *System.IO.IOException* 259
— *System.Net.WebException* 259
— *WebException* 259
использование динамических элементов 566

К

карта изображения 143
каскадные таблицы стилей 556
класс 67
— *ApplicationException* 263
— *BrowserWindow* 169
— *Calendar* 619
- *Circle* 89, 460
— *Circle(1)* 91
— *CompareInfo* 639
— *Convert* 639
— *CultureInfo* 619
- *Debug* 389, 454
— *Encoding* 641
- *FlashCardClass* 426, 442, 444
— *FormsAuthentication* 356, 358
- *Global* 102
- *HttpCookie* 287
- *MailMessage* 298, 299

— *Math* 13
- *MathBox* 486, 492
— *Shapes* 460
- *SmtpMail* 298, 299
— *Sphere* 89, 91, 460
— *SuperText* 496
— *TextWriterTraceListener* 455
- *Thread* 624
- *Trace* 454
— *Translator* 86
— *UserLoggedOnException* 263
- *WebControl* 477, 496
- *WebControlI* 477
- *WebForm1* 102
— создание 67
клиентский сценарий
— встроенный 310
— выбор языка сценариев 310
— определение обработчиков событий 314
— особенности 308
— преимущества 308
ключевое слово
— *abstract* 87, 94
— *base* 91
- *catch* 258
- *Catch* 258
— *interface* 87
— *MustInherit* 94
— *MustOverride* 94
— *MyBase* 91
— *Overridable (I)* 89
— *throw* 260
- *Throw* 260
— *virtual* 87
— *virtual (1}* 89
ключевые понятия объектно-ориентированного программирования 83
ключевые слова, используемые при наследовании 88
кодировка 640
— UTF-16 640
- UTF-8 640
— использование других кодировок 641
— сохранение файлов в различных кодировках 640
коллизии пространств имен 77
контрольная точка 2.31
концепция множественного наследования 8!
культура
— генерация строк 639
— для пользовательского интерфейса 636
— концепция 617
— назначение динамическое на уровне потоков 625
— назначение через параметры Web.config 623
— нейтральная 624
— определение во время выполнения 628
— определение при первой загрузке страницы 619
— поддержка нескольких культур 620
кэш глобальных сборок 395
кэширование 412
— настройка для Web-приложения 412
кэшируемые события 59

Л

локализация 617

М

масштабирование
— путем наращивания числа процессоров 416
— путем наращивания числа серверов 417
масштабируемость приложения 416
метод 12
- *__doPostBack()* 510
— *AcceptChanges* 227
— *Answer* 452
— *Array* 12
— *Assert* 454
— *Authenticate* 357
— *Authentication* 351
— *Begin Transaction* 221
- *ClearError* 265, 270
- *Close* 222
— *Console. WriteLine* 428
- *CreateChildControls* 476, 484
— *CreateObject* 442
— *CreateSpecificCulture* 636
- *Debug. WriteLin* 428
— *Delete* 203
— *Eval* 212
— *ExecuteNonQuery* 219, 220
- *ExecuteReader* 219, 220, 221
— *ExecuteScalar* 219
— *Execute* 168
- *Find* 292, 294, 295
— *FindBy* 202
— *Flush* 456
—*focus* 591
— *Form* 166
-*Get* 530
— *GetLastError* 265, 270
— *HashPasswordForStoringInConfigFile* 358
- *IsInRole* 349
— *LoadPostBackData* 511
— *open* 591
— *Open* 169
— *Pi* 13
— *PostedFile* 154
- *Pow* 13

— *Query Siring* 1 1 0
— *RaisePostBackEvent* 510
— *Read* 222
— *ReadXml* *294*
- *ReadXML* 193
- *ReadXmSchema* 294
- *Redirect* 265, 298
— *RedirectFromLoginPage* 356
— *RejectChanges* 225, 227
- *Render* 487, 520
— *RenderEndTag* 501
- *SaveXML* 295
- *Send* 300
— *Set* 530
— *set Interval* 544
- *showHelp* 593
— *Shuffle* 428
— *String* 12
— *Sum* 488
- *To St ring* 628
— *Trace* 437
— *Transfer* 165
— *Translate* 86
— *TypeConverter* 503
— *Update* 203, 224
— *Warn* 274
— *Write* 274, 455, 500
— *WriteLine* 455
- *WriteXML* 193
—.*RaisePost Back Event* 508
модификация записей в базе данных 200

H

наследование 87
настройка приложения 415
необработанные исключения 255

O

обновление БД во время выполнения 207
обработка исключений 255
— *On Error* 254
— использование блоков 255
— ключевые слова 255
— совершенствование 278
— способы 255
обработчик события
— *Application_Begin Request* 55
— *Application_End* 55
— *Application_EndRequest* 55
— *Application_Error* 265
— *Application_Start* 55
- *butOK_Click* 131
-- *butRestore_Click* 227
-*Click* 143, 484
— *Error* 264
-- *Global_Error* 265
— *Page_AbortTransaction* 57
— *Page_CommitTransaction* 57
— *Page_DataBinding* 57
— *Page_Disposed* 57
— *Page_Error* 57, 265
— *Page_Init* 57
— *Page_Load* 53, 57, 169, 198, 211, 214, 287
— *Page_PreRender* 57
— *Page_Unload* 57
— *SelectionChanged* 151
— *Session_End* 55
— *Session_Start* 55, 70
обращение к объектам в ADO.NET 200
общие методы 13
объект
— *Application* 54, 102
- *Browser* 287, 305, 309
- *DataBinder* 214
— *DataRow* 204
- *DataSet* 193
- *FlashCardClass* 70
— *Global* 102
— *HtmlTextWriter* 499
— *Identity* 351
— *OleDbTransaction* *111*
- *Page* 104, 270
— *Request* 105, 619
— *ResourceManager* 635, 637
- *Response* 107, 130, 441
— *Server* 167, 265
— *Session* 54
— *SqlCommand* 219
— *SqlTransaction* *111*
- *Test* 436
— *Thread* 636
— *Trace* 275
- *TranslatorClass* 121
- *User* 349
- *WebForm1* 102
— *window* 316, 591
- *Window* 169
объект ActiveX
— Macromedia ShockWave 542
— Windows Media Player 528
— работа с разными версиями 536
— добавление к Web-форме 524
— работа в Netscape Navigator 529
— установка на клиентский компьютер 529
— установка свойств и методов проигрывателя 527
— элемент *param* 527
объектная модель документа *см.* COM
одноранговое приложение 31
олицетворение 340
оператор
— *EnsureChildControls* 491
- *Imports* 79, 82, 356

— *Inherits* 87
— *using* *79,* 356
оптимизация приложения 415
оснастка MMC 400
— Event Viewer 400
— Performance Log and Alert 400
— определение новых типов событий 401
— просмотр журналов событий 408
— просмотр событий 401
— создание журналов событий 406
отличия серверных и HTML-элементов управления 129
отображение набора данных на Web-форме 197
отсоединенная архитектура 192
ошибка 254

П

панель Projects 16
перегрузка члена базового класса 89
переменная состояния
— *Application* 63, 113, 357
— *Session* 54, 63, 113, 265, 357
переход между страницами в приложениях 164
пользовательский элемент управления 464, 465
— Spin 473
— включение сеточной разметки 474
— добавление к Web-форме 470
— объявление событий 472
— отличия от кода Web-форм 467
— применение в коде 471
— создание интерфейса 466
поток 624
— определение культуры 628
потоковая разметка 466
привязка данных 209
приложения Web Forms 48
проблемы при использовании БД в приложениях 224
проект 26
пространство имен 12, 32
— *QueryStrings* 180
— *System* 12, 76
— *System.Array* 78
— *System.Data* 193
— *System.Data.OleDb* 193
- *System.Data.SqlClient* 193
— *System. Diagnostics* 454
— *System.EnterpriseServices* 224
— *System.Globalization* 617, 619, 639
- *System.Guid* 291
— *System.IO* 180
— *System. Resources* 635
— *System. String* 78
— *System. Text* 640
— *System. Web* *76,* 101
- *System.Web.Security* *352,* 356
- *Wombat.Utils* 78
протокол 3, 11, 324, 373, 374, 385, 386
- HTTP 373
- HTTPS 372
- mailto 298
— — ключи протокола 299
— — создание гиперссылки 298
— *ms-its* 608
-SSL 368, 371
псевдоним *Виртуальный_каталог* 46

Р

развертывание приложения 393
реестр UDDI 319
рециркуляция процессов 4] 1
решение 26

С

свойство
— *Attributes* 475
— *behavior* 566
— *Body* 441
— *ChartTitle* 520
- *Checked* 146
— *CommandText* 204
— *CommandType* 204
— *ContentEncoding* 641
- *ControlToValidate* *59,* 157
- *Cookies* 287
- *CurrentCulture* 624, 636
- *CurrentUICulture* 629, 630, 636
— *DataBinding* 137
— *DataBindings* 212
- *DataRowState* 227
- *DataSource* 137, 211
- *DataValue* 216
— *DeleteCommand* 204
— *EnableViewState* 138
— *Expiration* 287
— *Expires* 288
— *Filename* 536
— *FirstNumber* 452
— *Frames* 305
— *GroupName* 146
— *Identity* 349
— *ImageUrl* 147
- *innerHTML* 637
— *InsertCommand* 204
- *IsEnabled* *275*
— *IsolationLevel*
— — значения 231
— *IsPostBack* 523

— *JavaScript* 309
— *Name* 620
— *Operation* 453
- *Result* 488
— *RowState* 204
— *SecondNumber* 452
— *Selected* 147
— *SelectedDate* 151
— *SelectedDates* 151
— *SelectedItem* 136
— *SelectedItem.Value* 216
— *ShowMessage* 158
— *SmtpServer* 300
— *StoredProcedure* 204
— *Style* 474
- *TableDirect* 204
- *Text* 86, 204, 488, 637
- *ToolTip* 590
- *TraceLevel* 437
— — значения 437
— *Transaction* 228
— *UpdateCommand* 204
— *UserLanguages* 619
— *value* 637
— *Value* 287, 468, 520
— *Values* 488
- *VBScript* 309
- *ViewState* 112
— *window.event.keyCode* 609
сеанс 55
серверные элементы управления 5, 6, 9, 21, 42, 48, 49, 52, 57, 58. 59, 65, 66, 67, 87, 101, 117, 128, 129, 138, 139. 141, 156, 157, 168, 172, 209, 253, 265, 283, 295, 296, 307, 332, 415, 435, 466, 476, 496. 517, 523, 532, 566, 570, 583, 590, 616, 630, 631, 632, 639
-XML 574
— привязка данных 209
— создание Web-приложений 285
— типы событий 58
сертификат 368
— запрос на выдачу 369
— установка в IIS 371
сеточная разметка 466
сигнатура члена 89
событие
— *Application* 55
— *Application_End* 53
— *Application_Start* 57
— *AuthorizeRequest* 350
— *change* 511
-*Click* 472, 523
- *Error* 264
— *onbounce* 540
— *OnChange* 506
— *onclick* 591
— *onfocus* 609
— *onkeydown* 609
— *Page_Init* 53
— *Page_PreRender* 203
— *PassportAuthentication_OnAuthenticate* 363, 364
— postback 508
— — *Click* 508
- *PreRender* 628
— *Session* 55
- *TextChanged* 60
события
— жизненного цикла Web-приложения 52
— проверки 59
— инициирующие возврат формы 58
совместное использование сведений о состоянии
— на уровне приложения 418
— на уровне сеанса 418
создание соединения с БД 205
сопутствующая сборка 630
— файлы ресурсов
— — загрузка 635
— — отображение строк 637
— — типы 633
— — добавление к проекту Web-приложения 633
составной элемент управления 464, 476
- MathBox **485**
— изменение размеров 495
— обработка действий пользователя 505
— определение вида 484
— создание 476
— создание производных 495
— сохранение значений свойств 502
состояние отображения 54
сохранение сведений о пользователе 328
справочная система
- HTML Help Viewer 607
- в формате HTML Help 593, 595
— компилированный файл HelpSample.chm 594
— на отдельных Web-формах 591
— отображение в браузере 608
— отображение контекстно-зависимой справки 609
— способы отображения 590
справочная система Visual Studio .NET 30
— поисковые окна 30
средства сохранения переменных между запросами 109
стартовый проект 26
статические методы 13
стиль 556
— CSS-файл 556
— динамические элементы 566

— для заданных элементов посредством идентификатора 561
— добавление 560
— иерархия 556
— классы 560
— мастер Style Builder 559
— модификация 559
— переключение таблиц во время выполнения 563
— создание 560
— уровни 556
— элемент
— — *ColorPick* 566
— — *movable* 566
— *-Slider* 569
страницы сообщений об ошибках 268
— на уровне приложений 268
— на уровне страницы 270
структурная обработка исключений 255

Т

тестирование 424, 450
— блочное 425
— добавление пользователей и паролей 438
— добавление счетчиков производительности 440
— драйверы 429
— и автоматическая компоновка 445
— имитация различных браузеров 441
— использование нескольких подключений 438
— комплексное 429
— на платформе 434
— нагрузочное 430
— — запись задания 431
— — исполнение теста 434
— план 424
— планирование заданий 447
— получение содержимого откликов 441
— применение результатов 450
— просмотр файлов журнала 449
— регистрация результатов 436
— регрессивное 430
— с помощью ACT 436
— создание командного файла 446
— типы тестирования 425
— цель 423
— цикл тестирования 425
—.NET-сборок 442
— — размещение 443
— — регистрация 443
типы окон Visual Studio .NET 19
— инструментальное окно 19
— окно документов 19
точка *прерывания* 28, 29, 389, 457, 458, 461
— возобновление исполнения приложения 29
— установка 29
точка сохранения *см.* контрольная точка
транзакция 223
— ACID-критерии 223
— использование с наборами данных 224
— этапы обработки 224
трассировка 272
— включение и выключение 272

У

уникальные ключи для идентификации пользователей 291
управляемый код 10
уровни доступа к классам и модулям 84
учетная запись хостинга 34

Ф

файл cookie 110
— особенности 288
— сохранение на клиентском компьютере 287
— хранение и извлечение информации о пользователе 286
файл Web.config 353
— атрибуты 392
— параметры 353
файлы проекта Web Forms 49
фрейм 302
— создание
— — набора фреймов 302
— — фрейма 302
функция
— *CreateObject* 444
— *Instr* **442**
— вспомогательная
— — *AddUser* 358
— — *CheckPassword* 360

Ц

центр сертификации 368

Ш

шаблон 138

Э

элемент управления
— **AdRotator 148**
— Button 131, 485
— Calendar 151
— CheckBox 146, 610
- CheckBoxList 147, 216
— Compare Validator 159

— CustomValidator 161
— DataGrid 139, 197, 225
— — отображение набора данных 209
— DataList 139
— — **категории шаблонов** 212
— — отображение набора данных 211
— DropDownList 133, 216
— File Field 152
— HTML+TIME 546
— — управление различными видами мультимедиа 549
— **Hyperlink 164**
— **Image 147**
— ImageButton 144, 532
— Label 131, 485
— ListBox 133, 216
— Panel 149
— RadioButton 146
- RadioButtonList 147, **216**
- Repeater 139
- Submit 160
— Submit Button 152
— Table 133
— Text Field 152
— TextBox 131, 485
— ValidationSummary 158
— отображающие списки и таблицы 133
элементы управления нестандартные 464
— генерация событий 493
— генерируемые 498
— добавление на панель Toolbox 515
— пользовательские 465
— проект для тестирования 480
— составные 476
— структура шаблона 480
— типы 464

ЛИЦЕНЗИОННОЕ СОГЛАШЕНИЕ MICROSOFT

прилагаемый к книге компакт-диск

ЭТО ВАЖНО - ПРОЧИТАЙТЕ ВНИМАТЕЛЬНО. Настоящее лицензионное соглашение (далее «Соглашение») является юридическим документом, оно заключается между Вами (физическим или юридическимлицом) и Microsoft Corporation (далее «корпорация Microsoft») на указанный выше продукт Microsoft, который включает программное обеспечение и может включать сопутствующие мультимедийные и печатные материалы, а также электронную документацию (далее «Программный Продукт»). Любой компонент, входящий в Программный Продукт, который сопровождается отдельным Соглашением, подпадает под действие именно того Соглашения, а не условий, изложенных ниже. Установка, копирование или иное использование данного Программного Продукта означает принятие Вами данного Соглашения. Если Вы не принимаете его условия, то не имеете права устанавливать, копировать или как-то иначе использовать этот Программный Продукт.

ЛИЦЕНЗИЯ НА ПРОГРАММНЫЙ ПРОДУКТ

Программный Продукт защищен законами Соединенных Штатов по авторскому праву и международными договорами по авторскому праву, а также другими законами и договорами по правам на интеллектуальную собственность.

1. **ОБЪЕМ ЛИЦЕНЗИИ.** Настоящее Соглашение дает Вам право:
   a) **Программный продукт.** Вы можете установить и использовать одну копию Программного Продукта на одном компьютере. Основной пользователь компьютера, на котором установлен данный Программный Продукт, может сделать только для себя вторую копию и использовать ее на портативном компьютере.
   b) **Хранение или использование в сети.** Вы можете также скопировать или установить экземпляр Программного Продукта на устройстве хранения, например на сетевом сервере, исключительно для установки или запуска данного Программного Продукта па других компьютерах в своей внутренней сети, но тогда Вы должны приобрести лицензии на каждый такой компьютер. Лицензию на данный Программный продукт нельзя использовать совместно или одновременно на других компьютерах.
   c) **License Pak.** Если Вы купили эту лицензию в составе Microsoft License Pak, можете сделать ряд дополнительных копий программного обеспечения, входящего и данный Программный Продукт, и использовать каждую копию так, как было описано выше. Кроме того, Вы получаете право сделать соответствующее число вторичных копий для портативного компьютера в целях, также оговоренных выше.
   d) **Примеры кода.** Это относится исключительно к отдельным частям Программного Продукта. заявленным как примеры кода (далее «Примеры»), если таковые входят в состав Программною Продукта.
      i) **Использование и модификация.** Microsoft дает Вам право использовать и модифицировать исходный код Примеров при условии соблюдения пункта (d)(iii) ниже. Вы не имеете права распространять в виде исходного кода ни Примеры, ни их **модифицированную** версию.
      ii) **Распространяемые файлы.** При соблюдении пункта (d)(iii) Microsoft дает Вам право на свободное от отчислений копирование и распространение в виде объектного кода Примеров или их модифицированной версии, кроме тех частей (или их модифицированных версий), которые оговорены в файле Readme, относящемся к данному Программному Продукту, как не подлежащие распространению,
      iii) **Требования к распространению файлов.** Вы можете распространять файлы, разрешенные к распространению, при условии, что: а) распространяете их в виде объектного кода только в сочетании со своим приложением и как его часть; б) не используете название, эмблему или товарные знаки Microsoft для продвижения своего приложения; в) включаете имеющуюся в Программном Продукте ссылку на.авторские права в состав этикетки и заставки своего приложения; г) согласны освободить *от* ответственности и взять на себя защиту корпорации Microsoft от любых претензий или преследований по закону, включая судебные издержки, если таковые возникнут в результате использования или распространения Вашего приложения; и д) не допускаете дальнейшего распространения конечным пользователем своего приложения. По поводу отчислений и других условий лицензии применительно к иным видам использования или распространения распространяемых файлов обращайтесь в Microsoft.

2. **ПРОЧИЕ ПРАВА И ОГРАНИЧЕНИЯ**
   - **Ограничения на реконструкцию, декомпиляцию и дизассемблирование.** Вы не имеете права реконструировать, декомпилировать или дизассемблировать данный Программный Продукт, кроме того случая, когда такая деятельность (только в той мере, которая необходима) явно разрешается соответствующим законом, несмотря на эго ограничение.

- **Разделение компонентов.** Данный Программный Продуктлицензируется как единый продукт. Его компоненты нельзя отделять друг от друга для использования более чем на одном компьютере.
- **Аренда.** Данный Программный Продукт нельзя сдавать в прокат, передавать во временное пользование или уступать для использования в иных целях.
- Услуги **по**техническ**ой поддержке.** Microsoft может (но не обязана)предоставить Вам услуги потехнической поддержке данного Программного Продукта (далее «Услуги»). Предоставление Услуг регулируется соответствующими правилами и программами Microsoft, описанными в руководстве пользователя, электронной документации и/или других материалах, публикуемых Microsoft. Любой дополнительный программный код, предоставленный в рамках Услуг, следует считать частью данного Программного Продукта и подпадающим под действие настоящего Соглашения. Что касается технической информации, предоставляемой **Вами** корпорации Microsoft при использовании ее Услуг, то Microsoft может задействовать эту информацию в деловых шлях, в том числе для технической поддержки продукта и разработки. Используя такую техническую информацию, Microsoft не будет ссылаться на Вас.
- **Передача прав на программное обеспечение. Вы** можете безвозвратно уступить все права, регулируемые настоящим Соглашением, при условии, что не оставите себе никаких копий, передадите все составные части данного Программного Продукта (включая компоненты, мультимедийные и печатные материалы, любые обновления, Соглашение и сертификат подлинности, если таковой имеется) и принимающая сторона согласится с условиями настоящего Соглашения.
- **Прекращение действия Соглашения.** Без ущерба для любых других прав Microsoft может прекратить действие настоящего **Соглашения,** если Вы нарушите его условия. В этом случае Вы должны будете уничтожить все копии данного Программного Продукта вместе со всеми его компонентами.

3. **АВТОРСКОЕ ПРАВО.** Все авторские права и право собственности на Программный Продукт (в том числе любые изображения, фотографии, анимации, видео, аудио, музыку, текст, примеры кода, распространяемые файлы и апплеты, включенные в состав Программного Продукта) и любые его копии принадлежат корпорации Microsoft или ее поставщикам. Программный Продукт охраняется законодательством об авторских правах и положениями международных договоров. Таким образом, Вы должны обращаться с данным Программным Продуктом, хак с любым другим материалом, охраняемым авторскими правами, **с тем исключением,** что Вы можете установить Программный Продукт на один компьютер при условии, что храните оригинал исключительно как резервную или архивную копию. Копирование печатных материалов, поставляемых вместе с Программным Продуктом, запрещается.

ОГРАНИЧЕНИЕ ГАРАНТИИ

ДАННЫЙ ПРОГРАММНЫЙ ПРОДУКТ (ВКЛЮЧАЯ ИНСТРУКЦИИ ПО ЕГО ИСПОЛЬЗОВАНИЮ) ПРЕДОСТАВЛЯЕТСЯ БЕЗ КАКОЙ-ЛИБО ГАРАНТИИ. КОРПОРАЦИЯ MICROSOFT СНИМАЕТ С СЕБЯ ЛЮБУЮ ВОЗМОЖНУЮ ОТВЕТСТВЕННОСТЬ, В ТОМ ЧИСЛЕ ОТВЕТСТВЕННОСТЬ ЗА КОММЕРЧЕСКУЮ ЦЕННОСТЬ ИЛИ СООТВЕТСТВИЕ ОПРЕДЕЛЕННЫМ ЦЕЛЯМ. ВЕСЬ РИСК ПО ИСПОЛЬЗОВАНИЮ ИЛИ РАБОТЕ С ПРОГРАММНЫМ ПРОДУКТОМ ЛОЖИТСЯ НА ВАС.

НИ ПРИ КАКИХ ОБСТОЯТЕЛЬСТВАХ КОРПОРАЦИЯ MICROSOFT, ЕЕ РАЗРАБОТЧИКИ, А ТАКЖЕ ВСЕ, ЗАНЯТЫЕ В СОЗДАНИИ. ПРОИЗВОДСТВЕ И РАСПРОСТРАНЕНИИ ДАННОГО ПРОГРАММНОГО ПРОДУКТА, НЕ НЕСУТ ОТВЕТСТВЕННОСТИ ЗА КАКОЙ-ЛИБО УЩЕРБ (ВКЛЮЧАЯ ВСЕ, БЕЗ ИСКЛЮЧЕНИЯ, СЛУЧАИ УПУЩЕННОЙ ВЫГОДЫ, НАРУШЕНИЯ ХОЗЯЙСТВЕННОЙ ДЕЯТЕЛЬНОСТИ. ПОТЕРИ ИНФОРМАЦИИ ИЛИ ДРУГИХ УБЫТКОВ) ВСЛЕДСТВИЕ ИСПОЛЬЗОВАНИЯ ИЛИ НЕВОЗМОЖНОСТИ ИСПОЛЬЗОВАНИЯ ДАННОГО ПРОГРАММНОГО ПРОДУКТА ИЛИ ДОКУМЕНТАЦИИ. ДАЖЕ ЕСЛИ КОРПОРАЦИЯ MICROSOFT БЫЛА ИЗВЕЩЕНА О ВОЗМОЖНОСТИ ТАКИХ ПОТЕРЬ, ТАК КАК В НЕКОТОРЫХ СТРАНАХ НЕ РАЗРЕШЕНО ИСКЛЮЧЕНИЕ ИЛИ ОГРАНИЧЕНИЕ ОТВЕТСТВЕННОСТИ ЗА НЕПРЕДНАМЕРЕННЫЙ УЩЕРБ, УКАЗАННОЕ ОГРАНИЧЕНИЕ МОЖЕТ ВАС НЕ КОСНУТЬСЯ.

РАЗНОЕ

Настоящее Соглашение регулируется законодательством штата Вашингтон (США), кроме случаев (и лишь в той мере, насколько это необходимо) исключительной юрисдикции того государства, на территории которо о используется Программный Продукт,

Если у Вас возникли какие-либо вопросы, касающиеся настоящего Соглашения, или если Вы желаете связаться с Microsoft по любой другой причине, пожалуйста, обращайтесь в местное представительство Microsoft или пишите по адресу: Microsoft Sales Information Center, One Microsoft Way, Redmond. WA 98052-6399.

Книги Microsoft Press на русском языке по программам сертификации Microsoft

Сертификационный экзамен	Издания, необходимые для подготовки к экзамену	Программа сертификации
№ 70-270 Installing. Configuring and Administering Microsoft Windows XP Professional	**Microsoft Windows XP Professional Учебный курс MCSA/MCSE.** ISBN 5-7502-0209-7, 1008 стр., +CD, 2002 г.	**MCSA/MCSE**
№ 70-305 Developing and Implementing Web Applications with Microsoft Visual Basic .NET and Microsoft Visual Studio .NET **№ 70-315** Developing and Implementing Web Applications with Microsoft Visual C# .NET and Microsoft Visual Studio .NET	**Разработка Web-приложений на Microsoft Visual Basic .NET и Microsoft Visual C# .NET Учебный курс** MCAD/MCSD. ISBN 5-7502-0221-6, 704 стр., +CD, 2003 г.	**MCAD/MCSD**
№ 70-306 Developing and Implementing Windows-based Applications with Microsoft Visual Basic .NET and Microsoft Visual Studio .NET **№ 70-316** Developing and Implementing Windows-based Applications with Microsoft Visual C# .NET and Microsoft Visual Studio .NET	**Разработка Windows-приложений на Microsoft Visual Basic .NET и Microsoft Visual C# .NET Учебный курс MCAD/MCSD.** ISBN 5-7502-0222-4. готовится к изданию	**MCAD/MCSD**
№ 70-300 Analyzing Requirements and Defining Microsoft .NET Solution Architectures	**Анализ требований и определение архитектуры решений Microsoft .NET. Учебный курс MCSD.** ISBN 5-7502-0248-8, готовится к изданию	**MCAD/MCSD**
№ 70-310 Developing XML Web Services and Server Components with Microsoft Visual Basic .NET and the Microsoft .NET Framework **№ 70-320** Developing XML Web Services and Server Components with Microsoft Visual C# and the Microsoft .NET Framework	**Разработка XML Web-сервисов и серверных компонентов на Microsoft Visual Basic .NET и Microsoft C# .NET. Учебный курс MCAD/MCSD.** ISBN 5-7502-0249-6, готовится к изданию	**MCAD/MCSD**
Network+Certification	**Компьютерные сети. Сертификация Network+. Учебный курс** ISBN 5-7502-0190-2, 704 стр., + CD, 2002 г.	**CompTIA**

издательство компьютерной литературы
РУССКАЯ РЕДАКЦИЯ

ПРОДАЖА КНИГ
оптом тел.: (095) 142-0571, e-mail: sale@rusedit.ru
интернет-магазин http://www.ITbook.ru, тел.: (095) 145-4519

Книги Microsoft Press на русском языке по программам сертификации Microsoft

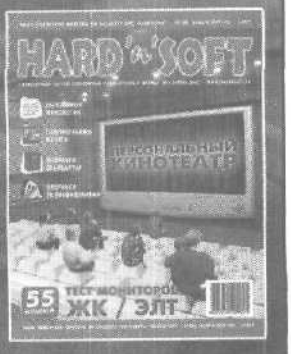
Сертификационный экзамен	Издания, необходимые для подготовки к экзамену	Программа сертификации
№70-100 Analyzing Requirements and Defining Solution Architectures	Принципы проектирования и разработки программного обеспечения. Учебный курс MCSD 2-е изд. ISBN 5-7502-0213-5, 736 стр., +CD, 2002 г.	MCSD
№70-210 Installing, Configuring and Administering Microsoft Windows 2000 Professional	Microsoft Windows 2000 Professional. Учебный курс MCSA/MCSE. 3-е изд. ISBN 5-7502-0246-1, 672 стр., 2003 г.	MCSA/MCSE
№70-215 Installing, Configuring and Administering Microsoft Windows 2000 Server	Microsoft Windows 2000 Server. Учебный курс MCSA/MCSE 3-е изд. ISBN 5-7502-0219-4, 912 стр., +CD, 2002 г.	MCSA/MCSE
№70-216 Implementing and Administering a Microsoft Windows 2000 Network Infrastructure	Администрирование сети на основе Microsoft Windows 2000. Учебный курс MCSE. 2-е изд. ISBN 5-7502-0233-X, 512 стр., +CD, 2003 Г.	MCSA/MCSE
№70-217 Implementing and Administering a Microsoft Windows 2000 Directory Services Infrastructure	Microsoft Windows 2000 Active Directory Services. Учебный курс MCSE. 2-е изд. ISBN 5-7502-0247-X, 800 стр.. 2003 г.	MCSE
№70-218 Managing a Microsoft Windows 2000 Network Environment	Управление сетевой средой Microsoft Windows 2000. Учебный курс MCSA/MCSE ISBN 5-7502-0212-7, 896 стр., 2003 г.	MCSA/MCSE
№70-220 Designing a Microsoft Windows 2000 Network Security	Безопасность сети на основе Microsoft Window* 2000. Учебный курс MCSE. ISBN 5-7502-0176-7, 912 стр., +CD, 2001 г.	MCSE
№ 70-221 Designing a Microsoft Windows 2000 Network Infrastructure	Разработка инфраструктуры сетевых служб Microsoft Windows 2000. Учебным курс MCSE. ISBN 5 7502-0192-9. 992 стр., +CD, 2001 г.	MCSE
№70-226 Designing Highly Available Web Solutions with Microsoft Windows 2000 Server Technologies	Создание Web-решений высокой доступности на основе Microsoft Windows 2000 Server. Учебный курс MCSE. ISBN 5-7502-0196-1, 576 стр., +CD, 2002 г.	MCSE
№ 70-227 Installing, Configuring and Administering Microsoft Internet Security and Acceleration (ISA) Server 2000	Microsoft Internet Security and Acceleration Server 2000. Учебный курс MCSE. ISBN 5-7502-0191-0, 544 стр., +CD. 2002 г.	MCSA/MCSE
№ 70-228 Installing. Configuring and Administering Microsoft SQL Server 2000 Enterprise Edition	Администрирование Microsoft SQL Server 2000. Учебный курс MCSA/MCSE/MCDBA. ISBN 5-7502-0202-X, 816 стр.. +CD, 2002 г.	MCSA/MCSE/MCDBA
№ 70-229 Designing and Implementing Databases with Microsoft SQL Server 2000 Enterprise Edition	Проектирование и реализация баз данных Microsoft SQL Server 2000. Учебный курс MCSE. ISBN 5-7502-0149-X, 704 стр., +CD, 2002 г.	MCSE

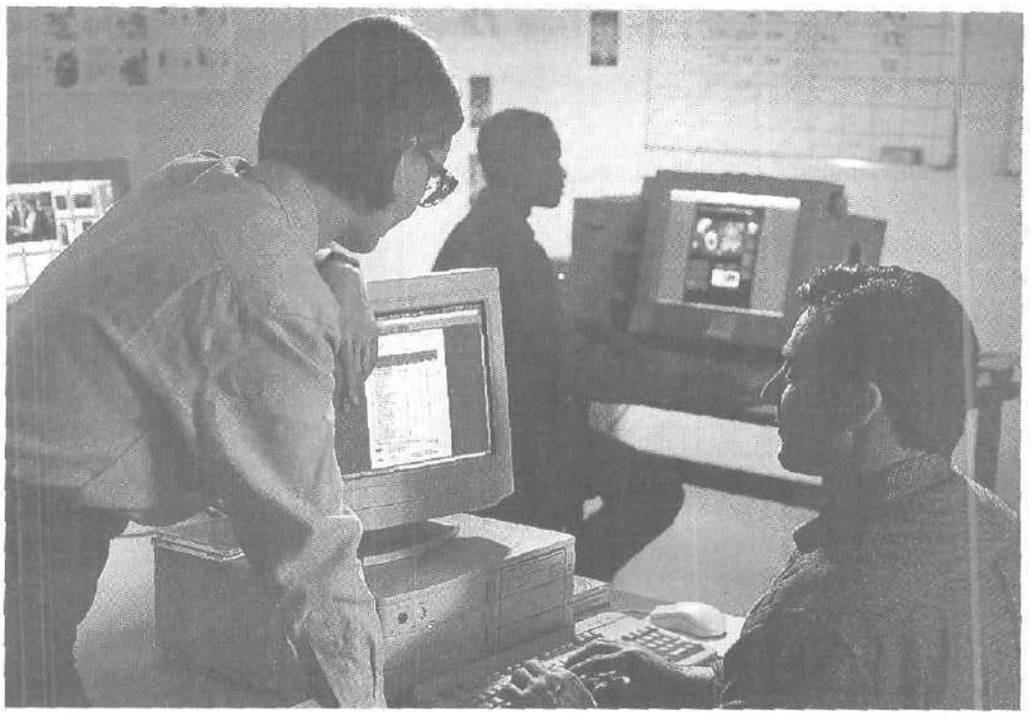
HARD 'n' SOFT

www.hardnsoft.ru



МНОГОГРАННЫЙ
КОМПЬЮТЕРНЫЙ
ЖУРНАЛ





Учебный центр SoftLine

Ваш курс начинается завтра!

Подготовка сертифицированных инженеров и администраторов Microsoft

Авторизованные и авторские курсы по:

* Windows 2000 / XP
* Sun Solaris 8
* Visual Studio .NET
* Электронной коммерции
* Безопасности информационных систем

и еще более 40 курсов по самым современным компьютерным технологиям.

Дневные и вечерние занятия.

Опытные преподаватели.

Индивидуальные консультации.

softline education

Microsoft CERTIFIED Technical Education Center

Учебный центр SoftLine

119991 г, Москва, ул. Губкина, д. 8
тел.: (095) 232 00 23
e-mail: educ@softline.ru
http://education.softline.ru

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ softline

ЛИЦЕНЗИРОВАНИЕ . ОБУЧЕНИЕ • КОНСАЛТИНГ

www.softline.ru . 232 0023 • info@softline.ru

Оперативная информация
для профессионалов в журнале «MSDN Magazine/Русская Редакция»



Своей *популярностью* этот *журнал* обязан содержанию: читатель получает самую свежую *и* актуальную информацию как от *ведущих* разработчиков корпорации *Microsoft*, так и от известных *специалистов* в сфере разработки *приложений*. Глубокие и содержательные статьи *оказывают* неоценимую помощь профессиональному разработчику, *регулярно использующему* средства разработки *Microsoft*.

Из статьи Ольги Дергуновой, главы представительства Microsoft в СНГ (MSDN Magazine/Русская Редакция, спецвыпуск № 1, апрель 2002 г.)

Журнал содержит:

постоянные рубрики

Web: **вопросы и ответы** — вопросы и ответы, связанные с разработкой по Web.

XML — основные спецификации XML (XML Schema, *пространство имен* VS Security, Web-сервисы и т. д.). Ответы на вопросы читателей.

ASP — все аспекты программирования с использованием ASP и ASP.NET.

Доступ к данным — проблемы разработки эффективных приложений для взаимодействия с СУБД, повышение производительности и обеспечение безопасности, а также программирование с использованием ADO.NET.

На переднем крае — передовые технологии, разработки (SOAP, Web-сервисы, .NET Framework и др.).

.NET — Джеффри Рихтер об основных концепциях и понятиях .NET и о *приемах программирования*.

Основы и тонкости — актуальные проблемы программирования, интересующие в основном *начинающих программистов*.

Отладка и оптимизация — эффективные приемы и подходы к решению задач отладки и оптимизации *приложений*.

Под капотом — особенности функционирования системных компонентов и исполняющих сред, создание эффективного программного обеспечения. Для системных программистов и разработчиков.

Изощренный код — необычные приемы, трюки программирования. Для системных программистов и разработчиков.

C++: вопросы и ответы — вопросы и ответы, связанные с программированием на языке C++.

тематические статьи

— глубоко и подробно раскрывают проблемы разработки ПО.

Информационная поддержка журнала «MSDN Magazine/Русская Редакция» — **www.microsoft.com/rus/msdn/magazine**

Оформить **подписку** и приобрести журнал можно в Интернет-магазине

www.ITbook.ru