Visual Studio 2010
для профессионалов

Стр.V1 Visual Studio 2010.indd 1 28.02.2011 14:49:47

Professional Visual Studio® 2010

Nick Randolph
David Gardner
Michael Minutillo
Chris Anderson



Wiley Publishing, Inc.

Pro Visual Studio 2010.indd 2 28.02.2011 14:49:47

Visual Studio 2010
для профессионалов

Ник Рендольф
Дэвид Гарднер
Майкл Минутилло
Крис Андерсон

Москва • Санкт-Петербург • Киев
2011

Стр. 3
Visual Studio 2010.indd 3 28.02.2011 14:49:47

ББК 32.973.26-018.2.75
Р39
УДК 681.3.07

Компьютерное издательство “Диалектика”
Зав. редакцией *С.Н. Тригуб*
Перевод с английского докт. физ.-мат. наук *Д.А. Клюшина*,
В.А. Коваленко, канд. техн. наук *И.В. Красикова*
Под редакцией докт. физ.-мат. наук *Д.А. Клюшина*
По общим вопросам обращайтесь в издательство “Диалектика” по адресу:
info@dialektika.com, http://www.dialektika.com

Рендольф, Ник, **Гарднер**, Дэвид, **Минутилло**, Майкл, **Андерсон**, Крис.
Р39 Visual Studio 2010 для профессионалов. : Пер. с англ. — М. : ООО “И.Д. Вильямс”, 2011. — 1184 с. : ил. — Парал. тит. англ.
ISBN 978-5-8459-1683-9 (рус.)

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм. Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Wiley US.

Copyright © 2011 by Dialektika Computer Publishing.

Original English language edition Copyright © 2010 by Wiley Publishing, Inc., Indianapolis, Indiana

All rights reserved including the right of reproduction in whole or in part in any form. This translation is published by arrangement with Wiley Publishing, Inc.

Wiley, the Wiley logo, Wrox, the Wrox logo, Wrox Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. Visual Studio is a registered trademark of Microsoft Corporation in the United States and/or other countries. All other trademarks are the property of their respective owners. Wiley Publishing, Inc. is not associated with any product or vendor mentioned in this book.

Научно-популярное издание

Ник Рендольф, Дэвид Гарднер, Майкл Минутилло, Крис Андерсон

Visual Studio 2010 для профессионалов

Литературные редакторы *Л.Н. Красножон, И.А. Попова*
Верстка *О.В. Мишутина*
Художественный редактор *Е.П. Дынник*
Корректор *Л.А. Гордиенко*

Подписано в печать 28.02.2011. Формат 70x100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 75,29. Уч.-изд. л. 75,1.
Тираж 1000 экз. Заказ № 0000.

Отпечатано с готовых диапозитивов
в ГУП “Типография “Наука”
199034, Санкт-Петербург, 9-я линия В. О., 12

ООО “И. Д. Вильямс”, 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1683-9 (рус.)
ISBN 978-0-470-54865-3 (англ.)

© Компьютерное изд-во “Диалектика”, 2011, перевод, оформление, макетирование
© Wiley Publishing, Inc., 2010

Оглавление

Введение 34

Часть I. Интегрированная среда разработки 41

Глава 1. Краткий обзор 43
Глава 2. Окна Solution Explorer, Toolbox и Properties 55
Глава 3. Параметры и настройки 74
Глава 4. Рабочая среда Visual Studio 95
Глава 5. Команды Find, Replace и Help 115

Часть II. Начинаем работу 131

Глава 6. Решения, проекты и элементы управления 133
Глава 7. Технология IntelliSense и закладки 163
Глава 8. Сниппеты кода и рефакторинг 182
Глава 9. Окно Server Explorer 204
Глава 10. Моделирование с помощью утилиты Class Designer 221

Часть III. Заглянем поглубже 235

Глава 11. Модульное тестирование 237
Глава 12. Документирование с помощью XML-комментариев 264
Глава 13. Инструменты, обеспечивающие согласованность кода 288
Глава 14. Генерация кода с помощью генератора T4 305
Глава 15. Шаблоны проектов и элементов управления 332
Глава 16. Особенности языков программирования 351

Часть IV. Расширенные клиентские приложения 373

Глава 17. Проекты Windows Forms Applications 375
Глава 18. Подсистема Windows Presentation Foundation (WPF) 391
Глава 19. Бизнес-приложения для пакета Office 421

Часть V. Веб-приложения 439

Глава 20. Веб-формы ASP.NET 441
Глава 21. Платформа ASP.NET MVC 480
Глава 22. Технология Silverlight 514
Глава 23. Механизм Dynamic Data 529
Глава 24. Коллекция программных продуктов SharePoint 555
Глава 25. Платформа Windows Azure 576

Часть VI. Данные 591

Глава 26. Инструменты Visual Database Tools 593
Глава 27. Компоненты DataSet и DataBinding 604

Глава 28. Технология Language Integrated Queries (LINQ) 632
Глава 29. Технология ADO.NET Entity Framework 663
Глава 30. Создание отчетов 686

Часть VII. Службы приложений 719

Глава 31. Платформа Windows Communication Foundation (WCF) 721
Глава 32. Платформа Windows Workflow Foundation (WF) 741
Глава 33. Службы клиентских приложений 765
Глава 34. Службы синхронизации 783
Глава 35. Службы WCF RIA 795

Часть VIII. Конфигурация и ресурсы 807

Глава 36. Файлы конфигурации 809
Глава 37. Строки подключения 830
Глава 38. Файлы ресурсов 840

Часть IX. Отладка 859

Глава 39. Использование окон отладчика 861
Глава 40. Отладка с применением точек прерывания 878
Глава 41. Подсказки по данным, прокси и визуализаторы 891
Глава 42. Отладка веб-приложений 904
Глава 43. Передовые методы отладки 920

Часть X. Построение и развертывание 937

Глава 44. Переход на систему Visual Studio 2010 939
Глава 45. Настройка сборки приложений 945
Глава 46. Контроль версий и подписи сборок 962
Глава 47. Умышленное запутывание, мониторинг и управление приложениями 969
Глава 48. Упаковка и развертывание 991
Глава 49. Развертывание веб-приложений 1013

Часть XI. Настройка и расширение системы Visual Studio 1029

Глава 50. Модель автоматизации 1031
Глава 51. Надстройки 1039
Глава 52. Макросы 1053
Глава 53. Библиотека Managed Extensibility Framework (MEF) 1060

Часть XII. Редакция Visual Studio Ultimate 1085

Глава 54. Редакция Visual Studio Ultimate для архитекторов 1087
Глава 55. Редакция Visual Studio Ultimate для разработчиков 1101
Глава 56. Редакция Visual Studio Ultimate для тестировщиков 1121
Глава 57. Система Team Foundation Server 1138

Предметный указатель 1167

Содержание

Об авторах 28
О техническом редакторе 29
Благодарности 30

Введение 34

Для кого предназначена книга 34
Чему посвящена книга 34
Краткая история системы Visual Studio 35
Единая полноценная среда 35
Структура книги 36
Что необходимо для того, чтобы пользоваться этой книгой 37
Соглашения 38
Исходные файлы 39
Ошибки 39
Сайт p2p.wrox.com 40

Часть I. Интегрированная среда разработки 41

Глава 1. Краткий обзор 43

Начало 43
Инсталляция системы Visual Studio 2010 43
Запуск системы Visual Studio 2010 45
Среда разработки Visual Studio 47
Разработка, создание, отладка и развертывание вашего первого приложения 48
Резюме 53

Глава 2. Окна Solution Explorer, Toolbox и Properties 55

Окно Solution Explorer 55
Типичные задачи 57
Добавление проектов и элементов 58
Добавление ссылок 60
Добавление ссылок на службы 62
Окно Toolbox 63
Настройка компонентов 65
Добавление компонентов 66
Окно Properties 67
Расширение окна Properties 68
Резюме 73

Глава 3. Параметры и настройки 74

Стартовая страница 74
Настройка окна Start Page 75

Код, лежащий в основе элементов управления 77
Компоновка окон 80
Просмотр окон и инструментальных панелей 80
Навигация по открытым элементам 81
Плавающие окна 82
Область редактирования 85
Шрифты и цвета 85
Принципы визуального представления 86
Полноэкранный режим 87
Отслеживание изменений 87
Другие возможности 88
Сокращенные комбинации клавиш 88
Проекты и решения 90
Команды Build и Run 91
Настройки языка VB 92
Настройки импорта и экспорта 93
Резюме 94

Глава 4. Рабочая среда Visual Studio 95
Редактор кода 95
Компоновка окна редактора кода 95
Регионы 96
Сворачивание 97
Форматирование кода 97
Перемещение вперед и назад 99
Дополнительные свойства редактора кода 99
Разделение окна 100
Свободное перемещение окон редактора кода 100
Создание групп закладок 101
Дополнительные возможности 102
Окно Command 103
Окно Immediate 105
Окно Class View 106
Окно Error List 107
Окно Object Browser 107
Окно Code Definition 108
Окно Call Hierarchy 109
Окно Document Outline 110
Разметка HTML 110
Разметка элементов управления 112
Реогранизация инструментальных окон 113
Резюме 114

Глава 5. Команды Find, Replace и Help 115
Окно Quick Find/Replace 116
Команда Quick Find 116
Команда Quick Replace 117
Группа Find Options 117
Шаблонные символы 118

Регулярные выражения 118
Команды Find and Replace 120
Команды Find/Replace in Files 120
Команда Find in Files 120
Find Dialog Options 121
Окна для вывода результатов 122
Команда Replace in Files 122
Команда Find Symbol 123
Команда Navigate To 124
Команда Incremental Search 125
Получение справки 126
Навигация и поиск в справочной системе 127
Конфигурирование справочной системы 128
Резюме 129

Часть II. Начинаем работу 131

Глава 6. Решения, проекты и элементы управления 133
Структура решения 134
Формат файла решения 135
Свойства решения 136
Узел Common Properties 137
Узел Configuration Properties 137
Типы проектов 139
Формат файлов проекта 140
Свойства проекта 141
Закладка Application 141
Раздел Compile (только для языка Visual Basic) 145
Закладка Build (только для языков C# и F#) 147
Закладка Build Events (только для языков C# и F#) 148
Закладка Debug 148
Закладка References (только для языка Visual Basic) 150
Закладка Resources 151
Закладка Services 152
Настройки 153
Пути к ссылкам (только для языков C# и F#) 153
Закладка Signing 154
Закладка My Extensions (только для языка Visual Basic) 155
Закладка Security 155
Закладка Publish 156
Закладка Code Analysis (только для версий VSTS Premium и Ultimate Editions) 156
Свойства проектов Web Application 158
Закладка Web 158
Закладка Silverlight Applications 159
Упаковка и публикация веб-приложений 160
Упаковка и публикация баз данных SQL 160
Проекты типа Web Site 161
Резюме 162

Глава 7. Технология IntelliSense и закладки 163
Технология IntelliSense 163
Основы технологии IntelliSense 164
Дополнение слова и фраз 165
Информация о параметрах 171
Команда Quick Info 172
Технология Intellisense для языка JavaScript 172
Контекст JavaScript IntelliSense 173
Ссылка на еще один файл JavaScript 174
Настройки системы IntelliSense 175
Общие настройки 175
Дополнение инструкции 176
Настройки для языка C# 177
Дополнительные возможности системы IntelliSense 177
Сниппеты 177
Комментарии XML 178
Добавление собственных схем IntelliSense 178
Закладки и окно Bookmark 178
Резюме 180
Глава 8. Сниппеты кода и рефакторинг 182
Введение в сниппеты 183
Хранение сниппетов 183
Сниппеты 184
Использование сниппетов в языке C# 184
Использование сниппетов в языке VB 185
Окружение с помощью сниппета 187
Библиотека Code Snippets Manager 187
Создание сниппетов 188
Просмотр существующих сниппетов 189
Поддержка рефакторинга 192
Операции рефакторинга 193
Операция Extract Method 194
Операция Encapsulate Field 196
Операция Extract Interface 197
Операция Reorder Parameters 199
Операция Remove Parameters 200
Операция Rename 200
Превращение локальной переменной в параметр 200
Операция Generate Method Stub 201
Меню Organize Usings 202
Резюме 203
Глава 9. Окно Server Explorer 204
Серверные соединения 205
Узел Event Logs 205
Классы управления 208
События управления 210
Узел Message Queues 213

Узел Performance Counters 216
Узел Services| 219
Соединения с источниками данных 220
Соединения SharePoint 220
Резюме 220

Глава 10. Моделирование с помощью утилиты Class Designer 221
Создание диаграммы классов 222
Область проектирования 223
Инструментальная панель Toolbox 224
Сущности 224
Соединители 225
Окно Class Details 225
Окно Properties 226
Компоновка 227
Экспорт диаграмм 228
Генерация и рефакторинг кода 228
Генерация кода методом “перетащить и отпустить” 228
Генерация кода с помощью технологии IntelliSense 230
Рефакторинг с помощью надстройки Class Designer 231
Надстройка Modeling Power Toys для системы Visual Studio 231
Усовершенствование визуализации 232
Усовершенствование функциональных свойств 233
Резюме 233

Часть III. Заглянем поглубже 235

Глава 11. Модульное тестирование 237
Первый тест 238
Идентификация тестов с помощью атрибутов 245
Дополнительные тестовые атрибуты 246
Проверка фактов 248
Класс Assert 249
Класс StringAssert 249
Класс CollectionAssert 250
Атрибут ExpectedException 250
Инициализация и возврат ресурсов 252
Атрибуты TestInitialize и TestCleanup 252
Атрибуты ClassInitialize и ClassCleanup 252
Атрибуты AssemblyInitialize и AssemblyCleanup 253
Контекст тестирования 253
Данные 253
Запись результатов теста 255
Модульное тестирование повышенной сложности 257
Специальные свойства 257
Тестирование закрытых членов 258
Тестирование контрактов 260
Управление большим количеством тестов 262
Резюме 262

Глава 12. Документирование с помощью XML-комментариев 264
Встроенное комментирование 265
XML-комментарии 265
Добавление XML-комментариев 266
Дескрипторы XML-комментариев 267
Использование XML-комментариев 279
Информация подсистемы IntelliSense 280
Генерация документов с помощью надстройки GhostDoc 280
Генерация документации с помощью набора утилит Sandcastle 282
Комментарии Task List 284
Резюме 287

Глава 13. Инструменты, обеспечивающие согласованность кода 288
Управление исходным кодом 289
Выбор хранилища исходного кода 289
Accessing Source Control 292
Автономная поддержка механизма управления исходным кодом 297
Стандарты кодирования 298
Анализ кода с помощью надстройки FxCop 299
Проверка стиля с помощью надстройки StyleCop 302
Надстройка Code Contracts 302
Резюме 304

Глава 14. Генерация кода с помощью генератора T4 305
Создание шаблона генератора T4 306
Строительные блоки генератора T4 309
Блоки выражений 310
Управляющие блоки 310
Блоки элементов класса 312
Как работает генератор кода T4 313
Директивы генератора T4 316
Шаблонные директивы 316
Директива Output 317
Директива assembly 317
Директива Import 318
Директива Include 319
Проблемы 319
Ошибки на этапе проектирования 320
Ошибки преобразования во время компиляции 320
Ошибки преобразования во время выполнения 321
Ошибки в сгенерированном коде 321
Преимущества генерирования кода 322
Генераторы Preprocessed Text Templates 325
Использование шаблона Preprocessed Text Templates 327
Различия между шаблонами 329
Советы и трюки 330
Резюме 331

Глава 15. Шаблоны проектов и элементов управления 332
Создание шаблонов 332
Шаблон элемента 332

Шаблон проекта 336
Структура шаблона 337
Параметры шаблона 339
Расположение шаблона 340
Расширение шаблонов 340
Установка шаблона проекта 340
Интерфейс IWizard 342
Генерация шаблона Extended Project Template 346
Комплект Starter Kits 349
Интерактивные шаблоны 349
Резюме 349

Глава 16.Особенности языков программирования 351
Гвозди надо забивать правильными молотками 352
Императивные языки 352
Декларативные языки 352
Динамические языки 353
Функциональные языки 354
Что все это значит? 354
Рассказ о двух языках 355
Компиляция без сборок PIA 356
Обобщенная вариантность 357
Язык Visual Basic 361
Лямбда и анонимные методы 361
Неявное продолжение строки 362
Автоматические свойства с начальными значениями 363
Инициализация коллекций и литералы массивов 363
Обнуляемые необязательные параметры 365
Надстройки Visual Basic PowerPacks 365
Язык C# 366
Позднее связывание с динамическим просмотром 366
Именованные и необязательные параметры 367
Язык F# 368
Наша первая программа на языке F# 368
Исследование свойств языка F# 370
Резюме 372

Часть IV. Расширенные клиентские приложения 373

Глава 17. Проекты Windows Forms Applications 375
Начало 375
Проект Windows Form 376
Свойства Appearance 377
Свойства Layout 378
Свойства Window Style 378
Настройки конструктора форм 378
Добавление и позиционирование элементов управления 380
Вертикальное выравнивание текстовых элементов управления 382
Автоматическое позиционирование нескольких элементов управления 382
Порядок перебора и перекрытия элементов управления 383

Фиксация дизайна элементов управления 384
Установка свойств элементов управления 385
Служебные компоненты 385
Интеллектуальные дескрипторы 386
Контейнерные элементы управления 387
Элементы управления Panel и SplitContainer 387
Элемент управления FlowLayoutPanel 388
Элемент управления TableLayoutPanel 388
Элементы управления стыковкой и закреплением 389
Резюме 390

Глава 18. Подсистема Windows Presentation Foundation (WPF) 391
Что такое WPF 392
Начало работы с системой WPF 393
Основы языка XAML 396
Элементы управления системы WPF 397
Компоновочные элементы управления в системе WPF 398
Конструктор WPF и редактор XAML 401
Работа с редактором XAML 402
Работа с конструктором WPF 403
Инструментальное окно Properties 405
Привязка данных 409
Выбор стиля приложения 413
Взаимодействие с системой Windows Forms 414
Использование элементов управления WPF Control в приложениях Windows Forms 415
Использование элементов управления Windows Forms Control в приложениях WPF 417
Отладка с помощью визуализатора WPF 418
Резюме 420

Глава 19. Бизнес-приложения для пакета Office 421
Выбор типа проекта для пакета Office 422
Настройка документа 423
Надстройки для приложений 423
Создание настройки документа 424
Первый проект VSTO 424
Защита компоновки документа 427
Добавление панели Actions Pane 428
Создание надстройки для приложения 430
Некоторые концепции программы Outlook 430
Создание области формы Outlook 431
Отладка приложений для пакета Office 434
Отмена регистрации надстройки 434
Отключение надстроек 436
Развертывание приложения из пакета Office 436
Резюме 438

ЧАСТЬ V. Веб-приложения 439

Глава 20. Веб-формы ASP.NET 441

Проекты Web Application и Web Site 442
Создание веб-проектов 443
Создание проекта Web Site 443
Создание проекта Web Application 446
Другие веб-проекты 447
Наборы для начинающих, общественные проекты и приложения и общедоступные приложения 449
Проектирование веб-форм 449
Редактор HTML Designer 450
Позиционирование элементов управления и элементов HTML 452
Форматирование элементов управления и HTML-элементов 454
Инструменты CSS 455
Инструменты проверки корректности 458
Элементы управления для веб-приложений 460
Навигационные компоненты 461
Аутентификация пользователей 461
Компоненты для работы с данными 463
Веб-части 467
Страницы мастера 467
Создание “толстого” клиента 469
Разработка приложений с помощью языка JavaScript 469
Работа с платформой ASP.NET AJAX 471
Использование элементов AJAX Control Extenders 473
Средства ASP.NET Web Site Administration 475
Безопасность 476
Настройки приложения 478
Конфигурация ASP.NET в службе IIS 478
Резюме 479

Глава 21. Платформа ASP.NET MVC 480

Шаблон Model-View-Controller 481
Начало работы с платформой ASP.NET MVC 482
Выбор модели 483
Контроллеры и методы действия 484
Визуализация пользовательского интерфейса с помощью представлений 487
Продвинутые возможности технологии MVC 494
Маршрутизация 494
Параметры методов действия 498
Области 501
Подтверждение корректности модели 504
Частичные представления 506
Пользовательские шаблоны представлений 506
Шаблоны Dynamic Data 507
Библиотека jQuery 511
Резюме 513

Глава 22. Технология Silverlight 514
Что такое технология Silverlight 515
Начало работы с платформой Silverlight 516
Средство навигации Navigation Framework 522
Темы 523
Режим запуска вне браузера 525
Резюме 528
Глава 23. Механизм Dynamic Data 529
Создание веб-приложения с помощью механизма Dynamic Data 530
Добавление модели данных 531
Исследование приложения, созданного с помощью механизма Dynamic Data 533
Настройка модели данных 535
Скаффолдинг отдельных таблиц 535
Настройка отдельных полей данных 536
Добавление правил проверки корректности 537
Настройка формата представления 540
Настройка представления 541
Шаблоны страниц 542
Шаблоны полей 545
Шаблоны сущностей 549
Шаблоны фильтров 551
Подключение механизма Dynamic Data к существующим проектам 553
Резюме 554
Глава 24. Коллекция программных продуктов SharePoint 555
Подготовка среды разработки 556
Инсталляция необходимых инструментов 557
Инсталляция платформы SharePoint 2010 559
Изучение платформы SharePoint 2010 560
Создание проекта Sharepoint 562
Создание пользовательских компонентов SharePoint 565
Разработка веб-частей 565
Создание типов контента и списков 567
Добавление получателей событий 569
Создание рабочих процессов на платформе SharePoint 570
Работа с компонентами коллекции Feature 572
Упаковка и развертывание 572
Резюме 575
Глава 25. Платформа Windows Azure 576
Платформа Windows Azure 577
Среда симуляции Development Fabric 578
Хранилище таблиц, массивов двоичных данных и очередей 579
Развертывание приложения 583
Настройка приложения 586
Технология SQL Azure 587
Технология AppFabric 588

Средство Service Bus 588
Средство Access Control Service 589
Резюме 589

Часть VI. Данные 591

Глава 26. Инструменты Visual Database Tools 593
Окна для работы с базами данных в системе Visual Studio 2010 594
Окно Server Explorer 594
Окно Data Sources 600
Редактирование данных 601
Предварительный просмотр данных 602
Резюме 603

Глава 27. Компоненты DataSet и DataBinding 604
Обзор компонентов DataSet 605
Добавление источника данных 606
Проектировщик объектов DataSet 608
Привязка данных 610
Компонент BindingSource 612
Компонент BindingNavigator 614
Выбор источников данных 616
Сохранение изменений 619
Вставка новых компонентов 619
Проверка корректности 621
Настраиваемые объекты DataSet 623
Цепочки объектов BindingSource и DataGridView 625
Работа с источниками данных 626
Компоненты Web Service Data Source 628
Просмотр данных 629
Резюме 631

Глава 28. Технология Language Integrated Queries (LINQ) 632
Провайдеры LINQ 633
Традиционные запросы 633
Фрагменты запроса 635
Инструкция From 635
Инструкция Select 637
Инструкция Where 637
Инструкция Group By 638
Проекции клиентов 639
Инструкция Order By 639
Отладка и выполнение 640
Технология LINQ to XML 641
Литералы языка VB XML 642
Вставка элемента XML как объекта класса XElement 643
Создание документа XML с помощью языка LINQ 644
Запросы XML 646
Поддержка схем 647
Технология LINQ to SQL 649

Создание объектной модели 650
Создание запросов с помощью технологии LINQ to SQL 652
Связывание объектов LINQ to SQL 657
Программа LINQPad 661
Резюме 662

Глава 29. Технология ADO.NET Entity Framework 663
Что такое Entity Framework 664
Сравнение с технологией LINQ to SQL 665
Концепции Entity Framework 665
С чего начать 666
Создание сущностной модели 666
Мастер Entity Data Model Wizard 666
Проектировщик Entity Framework 669
Создание и изменение объектов 673
Создание и изменение связей объектов 676
Наследование объектов 677
Проверка сущностной модели 677
Обновление сущностной модели с изменением базы данных 677
Запросы к сущностной модели 678
Обзор технологии LINQ to Entities 678
Получение контекста объекта 679
Операции CRUD 679
Навигация по связям 683
Расширенная функциональность 684
Обновление базы данных из сущностной модели 684
Добавление бизнес-логики в объекты 685
Классы POCO (Plain Old CLR Objects) 685
Резюме 685

Глава 30. Создание отчетов 686
С чего начать 686
Проектирование отчетов 688
Определение источников данных 689
Элементы управления для отчетов 691
Выражения, заполнители и агрегаты 702
Пользовательский код 705
Макет отчета 708
Подотчеты 711
Мастер отчетов 713
Вывод отчетов 714
Элементы управления Report Viewer 714
Генерация отчета 715
Вывод отчетов в различных форматах 716
Распространение отчетов 717
Резюме 717

Часть VII. Службы приложений 719

Глава 31. Платформа Windows Communication Foundation (WCF) 721
Что такое WCF 721
Первые шаги 722

Определение контрактов 723
Создание контракта службы 724
Создание контракта данных 726
Настройка конечных точек службы WCF 728
Хостинг служб WCF 732
Использование службы WCF 736
Резюме 740

Глава 32. Платформа Windows Workflow Foundation (WF) 741
Что такое Windows Workflow Foundation 741
Зачем использовать рабочий поток Windows 742
Концепции рабочего потока 743
Действия 743
Действия управления потоком 744
Выражения 745
Среда выполнения рабочего потока или планировщик 745
Закладки 745
Персистентность 746
Отслеживание 747
Первые шаги 747
Проектировщик Workflow Foundation 749
Создание рабочего потока 753
Разработка рабочего потока 753
Создание кода действий 755
Выполнение рабочего потока 757
Отладка рабочих потоков 759
Проверка рабочих потоков 760
Хостинг проектировщика рабочего потока 760
Резюме 764

Глава 33. Службы клиентских приложений 765
Клиентские службы 766
Ролевая авторизация 769
Аутентификация пользователя 771
Параметры 773
Форма регистрации 777
Поддержка автономной работы 779
Резюме 782

Глава 34. Службы синхронизации 783
Приложения без постоянного подключения 784
Прямое подключение к серверу 784
Приступая к работе со службами синхронизации 787
Службы синхронизации на нескольких уровнях 789
Фоновая синхронизация 790
Изменения клиента 793
Резюме 794

Глава 35. Службы WCF RIA 795
Первые шаги 795
Службы домена 798

Операции домена 800
Операции запроса 800
Операции вставки, модификации и удаления 800
Операции других типов 801
Использование службы домена в приложении Silverlight 801
Резюме 806

Часть VIII. Конфигурация и ресурсы 807

Глава 36. Файлы конфигурации 809
Файлы .Config 809
Файл machine.config 810
Файл web.config 810
Файл app.config 810
Файл security.config 811
Файл applicationHost.config 811
Схема конфигурации 811
Раздел configSections 812
Раздел startup 813
Раздел runtime 813
Раздел system.runtime.remoting 814
Раздел system.net 814
Раздел cryptographySettings 815
Раздел system.diagnostics 815
Раздел system.web 816
Раздел compiler 816
Атрибуты конфигурации 817
Настройки приложений 818
Использование раздела appSettings 818
Настройки проекта 819
Динамические свойства 820
Разделы пользовательской конфигурации 821
Пользовательские настройки 826
Проекты с настройками 827
Резюме 829

Глава 37. Строки подключения 830
Мастер строк подключения 830
Формат строки подключения к SQL Server 835
Создание строки в коде 836
Шифрование строк подключения 837
Резюме 839

Глава 38. Файлы ресурсов 840
Что такое ресурсы 840
Текстовые файлы ресурсов 841
Resx-файлы ресурсов 842
Бинарные ресурсы 843
Добавление ресурсов 843
Встраиваемые файлы как ресурсы 844

Именование ресурсов 844
Обращение к ресурсам 844
Файлы проектировщика 845
Приложение и ресурсы 846
Изображения управляющих элементов 848
Сопутствующие ресурсы 848
Культуры 848
Создание культурно-зависимых ресурсов 849
Загрузка культурно-зависимых файлов ресурсов 850
Сопутствующие культурно-зависимые ресурсы 850
Доступ к ресурсам 851
Загрузка изображений и пиктограмм 851
Межсборочные ссылки 851
ComponentResourceManager 852
Кодирование файлов ресурсов 853
Реализации интерфейсов ResourceReader и ResourceWriter 853
Реализации интерфейсов ResxResourceReader и ResxResourceWriter 854
Пользовательские ресурсы 854
Резюме 858

Часть IX. Отладка 859

Глава 39. Использование окон отладчика 861
Окно редактирования кода 861
Точки прерывания 862
Подсказки DataTip 862
Окно точек прерывания 862
Окно вывода 863
Окно Immediate 864
Окна Watch 865
Окно QuickWatch 866
Окна Watch 1–4 866
Окна Autos и Locals 867
Окна выполнения кода 867
Окно Call Stack 867
Окно Threads 868
Окно Modules 868
Окно Processes 869
Окна памяти 869
Окна Memory 1–4 870
Disassembly 870
Окно Registers 870
Инструмент IntelliTrace (только в версии Ultimate edition) 871
Окна параллельной отладки 872
Окно Parallel Stacks 873
Окно Parallel Tasks 874
Исключения 874
Настройка инструмента Exception Assistant 876
Развертывание исключения 877
Резюме 877

Глава 40. Отладка с применением точек прерывания 878
Точки прерывания 878
Установка точки прерывания 879
Точка прерывания по адресу 880
Добавление условий прерывания 881
Работа с точками прерывания 883
Точки трассировки 885
Создание точки трассировки 886
Действия точек трассировки 886
Управление выполнением 887
Пошаговое выполнение кода 887
Перемещение точки выполнения 889
Режим Edit and Continue 889
Ограничения 889
Отмена внесения изменений 889
Резюме 890

Глава 41. Подсказки по данным, прокси и визуализаторы 891
Подсказки по данным 892
Атрибуты отладки 893
Атрибут DebuggerBrowsable 893
Атрибут DebuggerDisplay 894
Атрибут DebuggerHidden 895
Атрибут DebuggerStepThrough 896
Атрибут DebuggerNonUserCode 896
Атрибут DebuggerStepperBoundary 896
Прокси-типы 897
Узел Raw View 898
Визуализаторы 899
Усовершенствованные методы 901
Сохранение изменений в объекте 901
Резюме 903

Глава 42. Отладка веб-приложений 904
Отладка серверного кода ASP.NET 905
Исключения веб-приложений 906
Режим Edit and Continue 908
Обработка ошибок 909
Отладка клиентского кода JavaScript 910
Установка точек прерывания в коде JavaScript 910
Отладка динамически генерируемого JavaScript 911
Отладка кода на языке JavaScript с помощью библиотеки ASP.NET AJAX 912
Отладка приложений Silverlight 912
Трассировка 913
Трассировка на уровне страницы 913
Трассировка на уровне приложения 914
Вывод трассировки 915
Обработчик Trace Viewer 916
Пользовательский вывод трассировки 916
Инструмент Health Monitoring 917
Резюме 919

Глава 43. Передовые методы отладки 920
Действия при старте 920
Отладка с применением кода 922
Класс Debugger 923
Классы Debug и Trace 923
Отладка работающих приложений 925
Подключение к процессу Windows 925
Подключение к веб-приложению 926
Удаленная отладка 927
Исходные тексты платформы .NET Framework 929
Отладка многопоточных и распараллеленных приложений 930
Отладка хранимых процедур SQL Server 932
Отладка в смешанном режиме 933
“Посмертная” отладка 933
Генерация файлов дампов 934
Отладка файлов дампов 934
Резюме 936

Часть X. Построение и развертывание 937

Глава 44. Переход на систему Visual Studio 2010 939
Переход к системе Visual Studio 2010 от CBCNTVS Visual Studio 2008 940
Обновление до версии .NET Framework 4.0 943
Резюме 944

Глава 45. Настройка сборки приложений 945
Общие настройки 945
Ручная настройка зависимостей 949
Страница Compile в среде Visual Basic 949
Окно Advanced Compiler Settings 950
События сборки 952
Страницы Build C# 953
Процессор MSBuild 955
Как Visual Studio использует процессор MSBuild 955
Схема MSBuild Schema 958
Контроль версий с помощью задач MSBuild 959
Резюме 961

Глава 46. Контроль версий и подписи сборок 962
Именование сборок 962
Согласованность версий 965
Строго именованные сборки 965
Глобальный кэш сборок 966
Подписание сборок 967
Резюме 968

Глава 47. Умышленное запутывание, мониторинг и управление приложениями 969
Дизассемблер MSIL 970
Декомпиляторы 971
Запутывание кода 972

Инструмент Dotfuscator Software Services 973
Атрибуты запутывания 977
Предостережения 979
Мониторинг приложения и управление им 981
Защита от вмешательства 982
Технологии Runtime Intelligence 983
Срок действия приложения 987
Отслеживание использования приложения 988
Резюме 990

Глава 48. Упаковка и развертывание 991
Инсталляторы Windows 991
Создание инсталлятора 992
Настройка инсталлятора 996
Добавление пользовательских действий 1000
Инсталлятор службы 1003
Технология ClickOnce 1006
Развертывание одним щелчком 1006
Обновление одним щелчком 1010
Резюме 1012

Глава 49. Развертывание веб-приложений 1013
Развертывание веб-сайта 1014
Публикация веб-сайта 1014
Копирование веб-сайта 1015
Развертывание веб-приложения 1015
Публикация веб-приложения 1016
Упаковка веб-приложения 1017
Функциональная возможность Web.config Transformation 1019
Инсталляторы веб-проектов 1021
Программа Web Platform Installer 1022
Расширение Web Platform Installer 1023
Резюме 1027

Часть XI. Настройка и расширение системы Visual Studio 1029

Глава 50. Модель автоматизации 1031
Варианты расширений системы Visual Studio 1031
Модель автоматизации системы Visual Studio 1032
Обзор модели автоматизации 1033
Решения и проекты 1034
Окна и документы 1035
Команды 1036
Отладчик 1037
События 1037
Резюме 1038

Глава 51. Надстройки 1039
Разработка надстройки 1039
Мастер Add-in Wizard 1040
Структура проекта 1042

Тестирование надстройки 1043
Файл с расширением .AddIn 1044
Класс Connect 1045
Создание окна инструмента 1045
Доступ к модели автоматизации Visual Studio 1048
Обработка событий системы Visual Studio 1050
Развертывание надстройки 1051
Резюме 1052

Глава 52. Макросы 1053
Понятие макроса 1053
Инструментальное Окно Macro Explorer 1054
Интегрированная среда разработки Macros 1054
Создание макроса 1055
Запись макросов 1056
Разработка макросов 1056
Выполнение макроса 1058
Развертывание макросов 1059
Резюме 1059

Глава 53. Библиотека Managed Extensibility Framework (MEF) 1060
Основы работы с библиотекой MEF 1061
Импорт и экспорт 1062
Контракты 1064
Каталоги 1066
Возможности библиотеки MEF 1067
Редактор Visual Studio 2010 Editor 1067
Подсистема модели текста 1068
Подсистема представления текста 1069
Подсистема классификации 1069
Операционная подсистема 1069
Расширение редактора 1070
Точки расширения редактора 1071
Службы редактора 1076
Расширение Check Comment Highlighter 1076
Резюме 1082

Часть XII. Редакция Visual Studio Ultimate 1085

Глава 54. Редакция Visual Studio Ultimate для архитекторов 1087
Проекты моделирования 1088
UML-диаграммы 1088
UML Model Explorer 1092
Применение Layer Diagram для проверки архитектуры приложения 1093
Связь с системой Team Foundation Server 1095
Изучение кода 1095
Инструмент Architecture Explorer 1095
Графы зависимостей 1097
Генерация диаграммы последовательности 1099
Резюме 1099

Глава 55. Редакция Visual Studio Ultimate для разработчиков 1101
Метрики кода 1102
Метрика Lines of Code 1103
Метрика Depth of Inheritance 1103
Метрика Class Coupling 1103
Cyclomatic Complexity 1104
Maintainability Index 1104
Код, исключенный из анализа 1104
Инструмент Managed Code Analysis Tool 1104
Инструмент C/C++ code analysis tool 1106
Инструменты для профилирования 1106
Настройка сеансов профилирования 1107
Отчеты 1108
Инструмент Standalone Profiler 1109
Инструмент IntelliTrace 1110
Инструменты для работы с базами данных 1112
Проект SQL-CLR Database 1112
Автономная схема базы данных 1112
Генерация данных 1115
Рефакторинг базы данных 1116
Сравнение схем 1117
Сравнение данных 1118
Статический анализ 1119
Редактор Transact-SQL 1119
Практические советы 1119
Резюме 1120
Глава 56. Редакция Visual Studio Ultimate для тестировщиков 1121
Автоматизированные тесты 1121
Веб-тесты производительности 1122
Тесты загрузки 1125
Модульные тесты баз данных 1127
Тест Coded UI Test 1128
Обобщенные тесты 1129
Упорядоченные тесты 1129
Связывание кода и тестов 1129
Инструмент Code Coverage 1130
Инструмент Test Impact Analysis 1132
Управление тестами в системе Visual Studio 1132
Инструмент Test and Lab Manager 1133
Окно Testing Center 1134
Окно Lab Center 1135
Резюме 1137
Глава 57. Система Team Foundation Server 1138
Командный проект 1139
Шаблоны процессов 1139
Отслеживание рабочих элементов 1141
Запросы рабочих элементов 1141
Типы рабочих элементов 1143

Добавление рабочих элементов 1143
Состояние рабочего элемента 1144
Интеграция приложений Excel и Project 1144
Программа Excel 1144
Программа Project 1145
Управление версиями 1147
Работа с окном Solution Explorer 1148
Извлечение файлов 1149
Запись файла 1149
Разрешение конфликтов 1150
Работа в автономном режиме 1152
История 1153
Аннотация 1153
Архивирование 1154
Ветвление 1155
Слияние 1156
Инструмент Team Foundation Build 1157
Отчеты и интеллектуальные ресурсы 1160
Командный портал 1161
Документация 1161
Управление процессом 1161
Списки коллекции SharePoint 1161
Инструментальные панели 1161
Инструмент Team System Web Access 1162
Администрирование TFS 1163
Автоматизация TFS и настройка процессов 1164
Типы рабочих элементов 1164
Настройка шаблона процесса 1165
Резюме 1166

Предметный указатель 1167

Об авторах

Ник Рендольф в настоящее время работает в основанной им компании Built to Roam, сосредоточившись на разработке полноценных приложений для мобильных устройств. Ранее Ник был соучредителем и менеджером по развитию компании N Squared Solutions, использующей технологии нового поколения. До этого Рэндольф был ведущим разработчиком в компании Intilecta Corporation, где занимался созданием своей прикладной платформы.

После получения научных степеней по техническим наукам (информационные технологии) и коммерции Ник был выдвинут на соискание статуса Microsoft MVP (Most Valuable Professional) за заслуги в работе группы пользователей Perth .NET и достижения в области разработки приложений для мобильных устройств. Он остается активным разработчиком приложений для этих устройств, поддерживая свой блог `http://community.softteq.com/blogs/nick/` и веб-сайт Professional Visual Studio `www.professionalvisualstudio.com/`.

Ника постоянно приглашают на различные мероприятия, включая Tech Ed Australia, MEDC и Code camp. Он также является автором статей для *MSDN Magazine* (ANZ edition), книг *Professional Visual Studio 2005* и *Professional Visual Studio 2008*, а также членом жюри финалов Всемирного конкурса Imagine Cup в 2004, 2005, 2007 и 2008 гг.

Дэвид Гарднер — опытный разработчик приложений для платформы .NET и главный архитектор программного обеспечения компании Intilecta Corporation. Дэвид постоянно стремится разрабатывать отлично спроектированные и высококачественные программные продукты, которые приносили бы удовлетворение пользователям. Начиная с 1990-х годов он работал архитектором проектов, консультантом, разработчиком, а также консультантом в различных организациях Австралии, Новой Зеландии и Малайзии.

Дэвид постоянно участвует в работе группы пользователей Perth .NET и таких мероприятий, как Microsoft TechEd и Microsoft Executive Summit. Он имеет степень бакалавра по компьютерным наукам и сертификат Microsoft Certified Systems Engineer. Дэвид является одним из авторов книги *Professional Visual Studio 2008* и блогов, посвященных системе Visual Studio и платформе .NET, на веб-сайте `www.professionalvisualstudio.com`.

Майкл Минутилло (Michael Minutillo) работает инженером по разработке программного обеспечения для платформы .NET и имеет степень бакалавра по компьютерным наукам. Он начал разрабатывать программы для платформы .NET в начале 2000 года под красноречивым названием “Indiscriminate Information Sponge” (“Копилка беспорядочной информации”), чтобы оплатить учебу в университете, и до сих пор остается активным членом сообщества пользователей платформы .NET.

Майкл регулярно посещает мероприятия, организованные обществом Perth .NET Community of Practice, на которых он рассказывает о новых функциональных возможностях языка C#, библиотеке ASP.NET MVC и принципах разработки через тестирование. В 2009 году Майкл организовал Perth ALT.NET User Group, которая собирается ежемесячно, чтобы обсудить инструменты разработки программного обеспечения и практику работы на платформе .NET.

Майкл поддерживает технический блог на веб-сайте `http://wolfbyte-net.blogspot.com`. Его адрес для контактов: `http://twitter.com/wolfbyte`.

Крис Андерсон более десяти лет был профессиональным разработчиком, специализируясь на создании настольных и мобильных приложений, а также веб-при-

ложений, использующих технологии компании Microsoft и ориентированных на бухгалтерский учет, оценку недвижимости, интеллектуальный анализ данных, производство скоропортящихся товаров, кремацию домашних животных, логистику, выездное обслуживание, продажи и строительство. Он получил степень бакалавра по проектированию компьютерных систем и специальный диплом, свидетельствующий о его практическом опыте (Diploma in Engineering Practise). Крис является совладельцем компании Peer Placements, которая занимается поиском специалистов по разработке программного обеспечения, консультантом и автором книг. В настоящее время он специализируется на технологии Silverlight (особенно в связи с созданием бизнес-приложений на основе этой платформы). Крис выступал на эту тему на таких конференциях, как Code Camp Australia 2009, TechEd Australia 2009, Silverlight Code Camp Australia 2010 и многочисленных семинарах Sydney Silverlight Designer and Developer Network (SDDN), одним из организаторов которых он является.

Крис поддерживает блог `http://chrisa.wordpress.com` и контакты по адресу `http://twitter.com/christhecoder`.

О техническом редакторе

Джо Беннетт (Joe Bennett) более двадцати лет проработала консультантом, разработчиком и архитектором программного обеспечения исключительно на платформе .NET, начиная с ее бета-версии 1.0. Она обожает создавать хорошо спроектированное и надежное программное обеспечение для организаций любого размера и любит делиться своими знаниями и опытом с другими разработчиками.

Джо является главным архитектором программного обеспечения в компании Carolina Software Consultants, LLC и бывшим президентом группы пользователей Triangle .NET User Group, расположенной в округе Триангл штата Северная Каролина. Она часто посещает конференции и мероприятия, имеет степень бакалавра по компьютерным наукам, полученную в Хьюстонском университете.

Благодарности

Писать книгу о системе Visual Studio 2010 иногда было мучительно; однако исследование новых или забытых функциональных возможностей разожгло мой азарт разработчика и желание поработать с самыми современными средствами разработки программного обеспечения, существующими в мире. Как и в предыдущих двух изданиях, это потребовало много времени, и я должен поблагодарить свою супругу Синтию, постоянно вдохновлявшую меня "сделать это", чтобы я снова мог вернуться к жизни.

Выражаю особую благодарность сотрудникам издательства Wrox, которые помогли мне вспомнить навыки технического писателя, в частности Келли Тэлбот, чье внимание к деталям позволило сохранить согласованность текста всей книги, несмотря на то, что у нее четыре автора, и Поля Риса, постоянно возвращавшего нас "на колею" и обеспечившего целостность процесса работы над книгой.

Я благодарен своим соавторам — Дейву, Майку и Крису, — согласившимся работать со мной над этим изданием. Я сомневаюсь, что справился бы с такой работой в одиночку, и очень рад иметь соавторов такого высокого уровня и разделить с ними нагрузку.

И наконец, хочется поблагодарить моих австралийских высококвалифицированных коллег и сотрудников компании Microsoft, любезно отвечавших на все мои вопросы.

Ник Рендольф

Писать книгу, несомненно, очень полезно и ответственно. Я думал, что во второй раз будет проще, но быстро убедился, что ошибался. Однако в ходе этого процесса я накопил много знаний, которые никогда бы ни приобрел при других обстоятельствах.

Техническое обеспечение работы над книгой имеет очень большое значение, и я особенно благодарен сотрудникам издательства Wrox, неустанно работавшим для того, чтобы обеспечить успешную реализацию нашего проекта. Без Поля Риса и Келли Тэлбот, буквально вытягивавшими из нас главу за главой, мы никогда не закончили бы эту работу. Мне было приятно сотрудничать с такими опытными людьми, и я благодарен им за их терпение и профессионализм.

Я испытываю огромную благодарность к своим соавторам Нику Рендольфу, Майклу Минутилло и Крису Андерсону, чья превосходная работа значительно улучшила книгу по сравнению с предыдущим изданием.

Мне было приятно работать над таким крупным проектом и постоянно обсуждать самые современные функциональные возможности. Я благодарен Габриэлу Тороку, Биллу Личу и Майку Мурсу из компании PreEmptive Solutions, Джонатану Картеру из компании Microsoft и знатоку технологии SharePoint Джереми Тейку, чьи советы и предложения значительно улучшили текст глав. Кроме того, благодарю моих коллег, с которыми мы вместе пили кофе, и разработчиков платформы .NET, которые (невольно) напоминали мне, как много я еще должен узнать о платформе .NET.

Особую признательность я испытываю к моим родителям, Джону и Венди, которые всегда поддерживали и вдохновляли меня. Отдельное спасибо моим дочерям, Джасмин и Эмили, которые на время перестали меня постоянно обнимать и щекотать, чтобы папочка смог закончить книгу. Обещаю, что верну вам свой долг с лихвой.

Более всего я должен благодарить мою жену и лучшего друга Джулию, оказывавшую мне огромную помощь. Она точно знала, что ее ожидает, когда я согласился писать эту книгу, и тем не менее оказала мне полную поддержку. Джулия брала на себя

бо́льшую часть домашних обязанностей, когда я был вынужден бросать все и заниматься только книгой, и я искренне признателен ей за любовь и поддержку.

Дэвид Гарднер

Прежде всего я хотел бы поблагодарить своих соавторов Ника и Дейва за приглашение участвовать в этом предприятии. Их доверие вдохновляло меня, когда я работал над книгой на рассвете. Когда я впервые появился на сцене группы Perth .NET, Ник и Дейв приветствовали меня дружеским советом и беседой. Эта атмосфера сохраняется в группе до сих пор. Особенно я благодарен Крису, который постоянно гнал меня к финишной линии. Когда я встречу его лично, я поставлю ему выпивку.

Как все начинающие писатели, я не имел представления о том, что должен делать, когда согласился писать эту книгу. Благодарю коллектив издательства Wrox за то, что они терпеливо меня обучали. Отдельное спасибо Келли Тэлбот, которая постоянно удерживала меня в рамках расписания и координировала работу коллектива, приводившего мои разделы в порядок. Редактирование технической книги такого объема представляет собой очень сложный процесс, требующий согласованной работы четырех авторов.

Писать книгу — отличный способ понять, как много ты еще не знаешь. Пересматривая материал, я постоянно удивлялся, обнаруживая маленькие особенности системы Visual Studio, о которых ничего не знал, но в Твиттере всегда находился кто-нибудь, кто мог посоветовать полезную ссылку или дать совет. Таких людей слишком много, чтобы я мог перечислить их по отдельности, поэтому в истинном "твиттеровском" стиле я просто говорю вам спасибо и знаю, что вы меня слышите.

В заключение хочу выразить особую признательность Барбаре, прекрасной женщине, на которой мне посчастливилось жениться. Когда Ник и Дэвид предложили принять участие в написании книги, я даже не представлял, как это повлияет на нашу жизнь. Барбара подарила мне вдохновение, поддержала мое решение принять вызов и освободила от домашней работы на выходных, забирая детей на прекрасные (и утомительные) прогулки, чтобы я мог закончить главу. Ее и моих трех прекрасных детей, Кьяру, Кейлеба и Уилла, я обещаю взять в следующее путешествие. И я никогда больше не буду писать книги. Обещаю!

Майкл Минутилло

Когда меня пригласили присоединиться к Нику, Дэйву и Майклу в работе над этой книгой, я не имел представления, во что ввязываюсь. Писательство — ужасный и отнимающий много времени процесс (намного больше, чем я мог себе представить). Писать каждую главу — все равно, что писать новую диссертацию, но в сжатые сроки. Причем как только ты заканчиваешь одну главу, то тут же должен начинать другую. Осознание того, что мои главы будут читать тысячи читателей и полагаться на информацию, которая в них содержится, только усиливало напряжение. Тем не менее я горд тем, что мне удалось закончить работу, и надеюсь, что она обогатит читателей знаниями и навыками, повышающими производительность их работы с системой Visual Studio. Кривая обучения оказалась большой, и я вдруг заметил, насколько часто думаю терминами кода, а не на английском языке. Для того чтобы выразить концепции в письменном виде, необходим талант, и я испытываю искреннее уважение к тем, у кого он есть.

Я хотел бы поблагодарить Ника и Дэйва за приглашение войти в коллектив авторов. Я был очень горд этим. Я благодарен всем трем моим соавторам: Нику, Дэйву

и Майклу. Несмотря на то что мы живем в разных концах Австралии, считаю, что мы работали как одна команда, и я благодарен им за поддержку, обратную связь и ободрение. Выражаю признательность также нашему редактору Келли Тэлбот, державшую нас под контролем и героически (хотя и не всегда успешно) пытавшуюся выполнить календарный план. Я получил большую помощь от многих сотрудников компании Microsoft и особенно благодарен Дастину Кэмпбеллу и Джону Валнеру, любезно оказавшими мне поддержку своими ответами.

Переходя к личному, хотел бы поблагодарить моих родителей, Майкла и Нарелле, чей тяжелый труд, благородство и любовь придавали мне вдохновение.

Крис Андерсон

Посвящения

Моей прекрасной Синтии

Ник Рендольф

Моей жене Джулии

Дэвид Гарднер

Барбаре с любовью навеки

Майкл Минутилло

Моим родителям, Майклу и Нарелле

Крис Андерсон

Введение

Система Visual Studio 2010 — огромный программный продукт, с какой бы стороны на нее ни взглянуть. Языки Visual Basic и C#, вобравшие в себя самые современные достижения лучших языков программирования, разработанных компанией Microsoft, в сочетании с большим количеством усовершенствований и новых функциональных возможностей пользовательского интерфейса могут напугать как новичков, так и опытных разработчиков, работающих на платформе .NET.

Книга *Visual Studio 2010 для профессионалов* охватывает все главные аспекты этого инструмента разработки программного обеспечения, демонстрирует способы использования каждой функциональной возможности и содержит советы по наилучшему применению разнообразных компонентов. В ней освещены основные строительные блоки, из которых состоит система Visual Studio 2010, причем пользовательский интерфейс разделен на части, которые удобно осваивать по отдельности. Затем каждый из этих компонентов разбирается на детали, которые изучаются как изолированно, так и в сочетании с остальными частями системы Visual Studio 2010, чтобы сделать работу еще более эффективной.

Для кого предназначена книга

Книга *Visual Studio 2010 для профессионалов* предназначена как для разработчиков программного обеспечения, которые впервые знакомятся с системой Visual Studio, так и для программистов, имеющих определенный опыт, но желающих овладеть теми возможностями, которые они ранее упустили.

Читатели, которые уже работали с предыдущими версиями системы Visual Studio, могут пропустить часть I, посвященную основным конструкциям пользовательского интерфейса, и перейти к следующей части, в которой детально разбираются новые функциональные возможности системы Visual Studio 2010. И все же, несмотря на то, что бо́льшая часть материала, изложенного в части I, может оказаться знакомой, рекомендуем прочитать ее, поскольку в контексте системы Visual Studio 2010 старые функциональные возможности могут получить новое освещение.

Если же читатели только начинают работать с системой Visual Studio 2010, то в первой части они смогут узнать ее основные концепции и ознакомиться с пользовательским интерфейсом, а также освоить способы настройки интегрированной среды разработки по своему усмотрению.

Чему посвящена книга

Система Microsoft Visual Studio 2010, вероятно, является самой сложной интегрированной средой разработки (integrated development environment — IDE), доступной для программистов в настоящее время. Она является результатом долгой истории развития языков программирования и интерфейсов и вобрала в себя достижения многих сред разработки программного обеспечения.

На следующих нескольких страницах мы кратко опишем систему Microsoft Visual Studio 2010, расскажем ее историю и покажем, что она предлагает разработчику. Чи-

татели, знакомые с системой Visual Studio и знающие, как она работает, могут пропустить эту часть и заняться другими аспектами интегрированной среды разработки.

Краткая история системы Visual Studio

Компания Microsoft долго и напряженно работала над средствами разработки программного обеспечения. Фактически ее первым программным продуктом была версия языка BASIC в 1975 году. В то время языки программирования в основном были интерпретируемыми, т.е. компьютер выполнял код строка за строкой. В течение последних трех десятилетий программирование достигло больших успехов, причем одним из крупнейших стало появление интегрированных сред разработки, предназначенных для оказания помощи разработчикам при выборе языка и платформы.

В эру 32-битовых вычислений компания Microsoft стала выпускать полноценные инструменты для разработки программного обеспечения, получившие название интегрированных сред разработки (integrated development environments). Они содержали не только компилятор, но и имели множество других функциональных возможностей, включая контекстно-чувствительный редактор и первичные функции технологии IntelliSense, помогавшие программистам понять, что делать и чего не делать в той или иной ситуации. Вместе с этими функциональными возможностями появились интуитивно понятные визуальные конструкторы пользовательского интерфейса с возможностью перетаскивания компонентов и ассоциированными инструментальными окнами, дающими разработчикам массу возможностей для работы с многочисленными компонентами конкретных окон или элементами управления.

Сначала каждый язык имел свою собственную интегрированную среду разработки, причем система Visual Basic была самой современной с точки зрения создания графического интерфейса и легкости использования, а система Visual C++ была самой мощной и гибкой.

Более поздние версии этих языков стали выходить под эгидой системы Visual Studio 6 в рамках единого крупного пакета “Visual”-инструментов, таких как FoxPro и InterDev. Однако было очевидно, что каждый язык имеет отдельную среду для работы и, как результат, порождает разные подходы к их созданию.

Единая полноценная среда

Система Visual Studio .NET, выпущенная компанией Microsoft в 2002 году, унаследовала многие функциональные возможности и атрибуты ранее выпущенных инструментальных средств. На нее повлияли системы Visual Basic 6, Visual InterDev, Visual C++ и другие инструменты, такие как FoxPro, которые компания Microsoft в основном разрабатывала самостоятельно. Коллектив разработчиков получал советы от разных внешних групп, но системы Visual Studio .NET 2002 и .NET 1.0 были основаны на принципах и преследовали цели, установленные компанией Microsoft.

В 2003 году была выпущена следующая версия Visual Studio .NET 2003, в которую были внесены лишь небольшие улучшения и исправлены ошибки. Два года спустя появились версия Visual Studio 2005 и платформа .NET Framework 2.0.

После этого были выпущены версия Visual Studio 2008, а также платформы .NET Frameworks 3.0 и 3.5. В этих версиях появились новые фундаментальные классы, возможности которых намного превосходили все, что делала компания Microsoft до этого. Однако самой важной частью этих версий была интегрированная среда разработки, многочисленные компоненты которой продолжали согласованно эволюционировать, обеспечивая все более эффективную и удобную работу.

Самый последний выпуск, состоящий из системы Visual Studio 2010 и платформы .NET Framework 4.0, построен на прочном фундаменте. Оболочка кодов была переписана с использованием графической подсистемы Windows Presentation Foundation. Благодаря этому многие визуальные проектировщики этой подсистемы получили новую жизнь. Кроме того, появились новые проектировщики, помогающие создавать приложения по технологии Silverlight и обеспечивающие встроенную поддержку для создания приложений для пакета Office.

Система Visual Studio 2010 имеет несколько вариантов: Express, Professional, Premium и Ultimate. Бо́льшая часть книги использует версию Professional Edition of Visual Studio 2010, но в некоторых частях используются функциональные возможности, доступные только в версиях Premium и Ultimate. Если читатели уже использовали эти версии ранее, то могут просто пролистать главы 54–57 (доступные в Интернете), в которых дан обзор их функциональных возможностей, выходящих за рамки версии Professional Edition.

Структура книги

Эта книга является первым изданием, посвященным основным аспектам системы Visual Studio 2010. Все необходимое сосредоточено в первых пяти главах, начиная со структуры IDE и макета и заканчивая разными опциями и настройками, которые можно изменять для того, чтобы подогнать интерфейс под свои потребности.

Остальная часть книги разбита на одиннадцать частей.

- **Начинаем работу.** В этой части показано, как управлять своими проектами и организовать их по своему усмотрению.
- **Заглянем поглубже.** Несмотря на то что многие графические компоненты системы Visual Studio, облегчающие работу пользователей, обсуждаются во многих главах, пользователям часто требуется помощь в процессе реального создания кода. Эта часть книги посвящена функциональным возможностям, поддерживающим программирование приложений, например, технологии IntelliSense, рефакторингу кода, а также созданию и выполнению модульных тестов. Самая последняя версия платформы .NET Framework лучше поддерживает динамические библиотеки и обеспечивает более точный баланс между функциональными возможностями двух основных языков платформы .NET: C# и VB. В этой части описываются изменения, касающиеся этих языков, а также рассматриваются функциональные возможности, позволяющие создавать более качественный и согласованный код.
- **Полноценные клиентские приложения и веб-приложения.** Поддерживая создание любых программ — от надстроек для пакета Office до "облачных" приложений, — система Visual Studio позволяет разрабатывать приложения для широкого спектра платформ. Эти две части посвящены прикладным платформам, поддерживаемым системой Visual Studio 2010, включая ASP.NET, Office, WPF, Silverlight 2 и ASP.NET MVC.
- **Данные.** Многие приложения используют тот или иной вид хранилищ данных. Система Visual Studio 2010 и платформа .NET Framework обеспечивают полноценную поддержку работы с базами данных и другими источниками. В этой части книги описывается использование коллекции DataSets, инструментов Visual Database Tools, технологии LINQ, служб Synchronization Services и платформы

ADO.NET Entity Framework для построения приложений, работающих с данными. В ней также показано, как можно представить свои данные с помощью системы Reporting.

- **Прикладные службы.** В процессе разработки приложения пользователю, вероятно, понадобится доступ к разным службам, которые могут находиться как внутри, так и вне организации. В этой части описываются технологии, такие как WCF, WF, Synchronization Services и WCF RIA Services, которые используются для соединения с этими службами.
- **Конфигурация и интернационализация.** Встроенная поддержка файлов конфигурации позволяет настраивать функции приложения "на ходу", не повторяя процедуру сборки. Более того, файлы ресурсов можно использовать как для доступа к статическим данным, так и для простой локализации приложения с учетом иностранных языков и культурных особенностей. В этой части показано, как можно использовать конфигурацию платформы .NET и ресурсные файлы.
- **Отладка.** Отладка приложения является одной из наиболее сложных задач, стоящих перед разработчиком, но правильное использование средств отладки, имеющихся в системе Visual Studio 2010, поможет проанализировать состояние приложения и определить причины любых ошибок. В этой части рассматривается поддержка процесса отладки, обеспечиваемая интегрированной средой разработки.
- **Сборка и развертывание.** Кроме обсуждения способов эффективной сборки решений и передачи приложений в руки конечных пользователей, в этой части описывается процесс обновления предыдущих версий.
- **Настройки и расширения системы Visual Studio.** Поскольку функциональные возможности предыдущих версий не всегда были способны удовлетворить потребности программистов, компания Microsoft сделала систему Visual Studio 2010 еще более расширяемой. В этой части описываются автоматическая модель, способы создания надстроек и макросов, а также использование нового каркаса MEF, обеспечивающего расширение системы Visual Studio 2010.
- **Visual Studio Ultimate (доступна в Интернете).** В последней части книги исследуются дополнительные функциональные возможности, доступные только в версиях Premium и Ultimate Visual Studio 2010. Кроме того, в ней показано, как продукт Team Foundation Server обеспечивает управление проектами по разработке программного обеспечения.

Мы старались описывать функциональные возможности системы Visual Studio как можно более последовательно и понятно. Тем не менее читателям могут понадобиться конкретные функции, необходимые для решения конкретных задач. Для того чтобы удовлетворить потребности таких пользователей, в книге расставлены ссылки на главы, где эти свойства описаны более подробно.

Что необходимо для того, чтобы пользоваться этой книгой

Для того чтобы использовать эту книгу с максимальной эффективностью, необходима только одна вещь — система Microsoft Visual Studio 2010 Professional Edition. Установив это программное обеспечение и овладев информацией, изложенной в дан-

ной книге, читатели за очень короткое время смогут узнать, как эффективнее всего использовать систему Visual Studio 2010.

Предполагается, что читатели уже знакомы с традиционной моделью программирования. Для иллюстрации функциональных возможностей системы Visual Studio 2010 в книге использованы языки C# и Visual Basic (VB). Кроме того, предполагается, что читатели могут понять листинги кода без объяснения основных концепций программирования какого бы то ни было языка. Если читатели лишь начинают программировать и хотят изучать язык Visual Basic, то им следует обратиться к книге Тэрона Уиллиса (Thearon Willis) и Брайана Ньюсома (Bryan Newsome) *Beginning Visual Basic 2010*. Аналогично, если читатели ищут хорошую книгу по C#, то советуем прочитать книгу *Beginning Visual C# 2010,* написанную большим коллективом авторов.

В некоторых главах обсуждаются дополнительные программные продукты и инструменты, которые могут использоваться совместно с системой Visual Studio. Ниже перечислены программные продукты, которые можно загрузить либо свободно, либо для испытаний.

- **Code Snippet Editor.** Этот инструмент, поставляемый сторонними разработчиками, был создан для создания сниппетов кода на языке VB. Программа Snippet Editor обсуждается в главе 8.
- **Sandcastle.** Используя надстройку Sandcastle, можно генерировать исчерпывающую документацию для каждого члена и всего класса, входящего в решение, с помощью XML-комментариев. XML-комментарии и программа Sandcastle обсуждаются в главе 12.
- **SQL Server 2008.** Инсталляция системы Visual Studio 2010 включает в себя инсталляцию программы SQL Server 2008 Express, которая позволяет создавать приложения, использующие файлы баз данных. Однако для более крупных промышленных приложений вместо нее следует использовать версию SQL Server 2008. Соединение с базами данных обсуждается в главе 26.
- **Версии Visual Studio 2010 Premium или Ultimate edition**. Это более сложные версии системы Visual Studio, содержащие инструменты для других этапов процесса разработки программного обеспечения, например тестирования и проектирования. Они обсуждаются в главах 54–57, которые доступны в Интернете.
- **Team Foundation Server.** Это инструмент, обеспечивающий управление всем жизненным циклом проекта в системе Visual Studio 2010. Он обсуждается в главе 57.

Соглашения

Для того чтобы облегчить чтение книги, мы используем несколько соглашений.



Врезки с этой предупреждающей пиктограммой содержат важную информацию, непосредственно относящуюся к окружающему их тексту.



Врезки с этой пиктограммой содержат замечания, советы, подсказки или отступления от основной темы.

В книге используются разные стили.

- Мы выделяем новые термины и важные слова, используемые впервые.
- Комбинации клавиш записываются так: <Ctrl+A>.
- Имена файлов, указатели URL и код в тексте выделяются так: `persistance.properties`.
- Код записывается двумя способами:

```
Большинство примеров набрано моноширинным шрифтом.
```

```
Особенно важные фрагменты или отличия от предыдущего сниппета
выделяются полужирным шрифтом.
```

Исходные файлы

Работая с примерами, приведенными в книге, читатели могут набирать код самостоятельно или использовать исходные файлы, которые прилагаются к книге. Все исходные файлы, используемые в книге, можно загрузить с веб-сайта `http://www.wrox.com`. Зайдя на сайт, найдите название книги (с помощью поля Search или списка названий) и щелкните на ссылке Download Code, которая относится к странице, на которой можно получить все исходные файлы. Код, который можно загрузить из веб, отмечается пиктограммой



Заглавие листинга содержит имя соответствующего файла. Если это просто сниппет, то имя файла можно найти в следующей строке.

Сниппет кода имя файла



Поскольку многие книги имеют похожие названия, иногда их легче найти по коду ISBN; наша книга имеет ISBN 978-0-470-54865-3.

Загрузив код, разархивируйте его с помощью своего архиватора. В качестве альтернативы с веб-страницы `http://www.wrox.com/dynamic/books/download.aspx` можно загрузить код не только к этой книге, но и ко всем другим книгам издательства Wrox.

Ошибки

Мы старались, чтобы в тексте и программах не было ошибок. Однако никто не совершенен, и ошибки случаются. Если читатели найдут ошибки в наших книгах, например опечатку или неправильный фрагмент кода, мы будем им очень благодарны. Сообщив нам об ошибке, вы сохраните нервы и время других читателей и одновременно поможете нам повысить качество книги.

Страница для регистрации ошибок, которые могут быть найдены в нашей книге, расположена на веб-сайте `http://www.wrox.com`. Для того чтобы ее найти, зайдите на сайт и найдите нашу книгу с помощью поля Search или списка названий. Затем, нахо-

дясь на странице, посвященной нашей книге, щелкните на ссылке Book Errata. На этой странице вы увидите все ошибки, найденные в нашей книге и подтвержденные редакторами издательства Wrox. Полный список книг, включая ссылки на страницы ошибок, можно найти на странице www.wrox.com/misc-pages/booklist.shtml.

Если вы не обнаружите на странице Book Errata ошибку, найденную вами, зайдите на страницу www.wrox.com/contact/techsupport.shtml, заполните форму и отправьте нам сообщение об этой ошибке. Мы проверим эту информацию и, если она подтвердится, поместим сообщение на странице ошибок и исправим ее в следующих изданиях книги.

Сайт p2p.wrox.com

Для общения с авторами и друг с другом присоединяйтесь к форумам P2P на сайте p2p.wrox.com. Эти форумы представляют собой веб-системы, позволяющие вам размещать сообщения, касающиеся книг издательства Wrox и связанных с ними технологий, а также общаться с другими читателями и пользователями. Эти форумы предлагают подписку на рассылку уведомлений о появлении новых сообщений на интересующие вас темы. На этом форуме представлены авторы и редакторы издательства Wrox, а также другие эксперты и читатели.

На сайте http://p2p.wrox.com вы найдете много разных форумов, которые помогут вам не только понять эту книгу, но и разработать свои собственные приложения. Для того чтобы присоединиться к этим форумам, следует выполнить следующие инструкции.

1. Зайти на сайт p2p.wrox.com и щелкнуть на ссылке Register.
2. Прочитать условия использования форумов и щелкнуть на ссылке Agree.
3. Заполнить анкету и указать дополнительную информацию, которую вы захотите сообщить, а затем щелкнуть на ссылке Submit.
4. Получить электронное сообщение с информацией о том, как подтвердить свою учетную запись и завершить процесс.



Сообщения, размещаемые на форуме, можно читать, и не присоединяясь к ним, но в этом случае вы не сможете размещать на них свои сообщения.

Присоединившись к форуму, вы сможете размещать на нем новые сообщения и отвечать на сообщения других участников. Читать сообщения в веб можно в любое время. Если вы предпочитаете получать уведомления о появлении новых сообщений, щелкните на пиктограмме Subscribe to this Forum возле имени форума в списке всех форумов.

Более подробную информацию об использовании сайта P2P можно найти в разделе, посвященном ответам на часто повторяющиеся вопросы о работе форумов, а также о сайте P2P и книгах издательства Wrox вообще. Для того чтобы прочитать эти ответы, щелкните на ссылке FAQ на любой странице сайта P2P.

ЧАСТЬ I

Интегрированная среда разработки

- **ГЛАВА 1.** Краткий обзор
- **ГЛАВА 2.** Окна Solution Explorer, Toolbox и Properties
- **ГЛАВА 3.** Параметры и настройки
- **ГЛАВА 4**. Рабочая среда Visual Studio 2010
- **ГЛАВА 5**. Команды Find, Replace и Help

1

Краткий обзор

В ЭТОЙ ГЛАВЕ...

- Инсталляция и начало работы со средой разработки Visual Studio 2010
- Создание и запуск первого приложения
- Отладка и развертывание приложения

При разработке программного обеспечения возникает необходимость в инструментах, помогающих писать, отлаживать и разворачивать наши приложения. Система Microsoft Visual Studio 2010 — это следующий этап непрерывной эволюции лучшей в своем классе *интегрированной среды разработки* (integrated development environment — IDE).

В этой главе содержатся начальные сведения о взаимодействии системы Visual Studio 2010 с пользователем и способах работы с меню, панелями инструментов и окнами. Она представляет собой краткий обзор интегрированной среды разработки, и поэтому в ней нет подробностей о том, например, какие настройки можно изменить или как выбрать конфигурацию среды, так как эти темы будут изложены в последующих главах.

Начало

С каждой новой версией Visual Studio процесс инсталляции постоянно улучшается. Это значит, что теперь можно установить и запустить систему Visual Studio 2010 с минимальными усилиями. В этом разделе мы бегло рассмотрим процесс инсталляции и начало работы со средой разработки.

Инсталляция системы Visual Studio 2010

Запустив процесс установки системы Visual Studio 2010, вы увидите диалоговое окно, показанное на рис. 1.1. В этом окне представлены три этапа инсталляции продук-

та. Как и следовало ожидать, на первом этапе происходит собственно инсталляция. Два остальных этапа являются необязательными. Вы можете либо инсталлировать документацию о продукте на своем компьютере, либо использовать ее в режиме онлайн (как правило, в этом случае вы получите доступ к ее новейшей версии). Мы рекомендуем поискать новые *сервис-релизы* (service release). В этом случае вы сможете работать с новейшей версией продукта и сопутствующими инструментами.



Рис. 1.1

В процессе инсталляции вы получите приглашение установить обратную связь с компанией Microsoft (рис. 1.2, *слева*) и согласиться с условиями лицензии на продукт (рис. 1.2, *справа*).

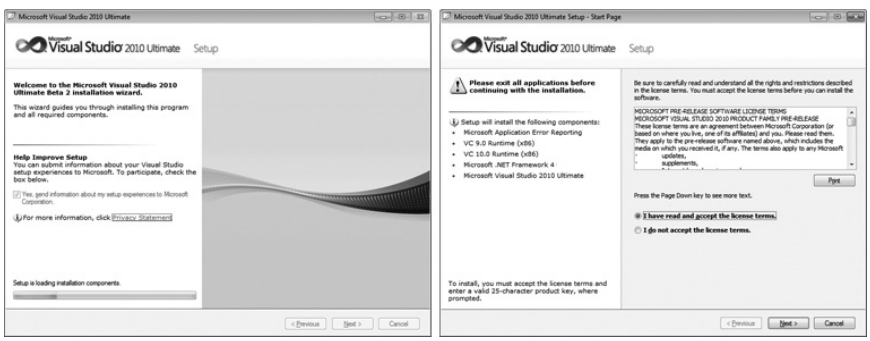


Рис. 1.2

Процесс установки системы Visual Studio 2010 оптимизирован для двух категорий разработчиков: для тех, кто создает управляемые, или .NET-приложения, и тех, кто пишет естественные приложения, т.е. на языке C++ (рис. 1.3, *слева*). Кнопка Customize позволяет выбрать компоненты из полного дерева компонентов, как показано на рис. 1.3, *справа*.

Выбрав компоненты для инсталляции, вы увидите обновленное окно прогресса (рис. 1.4). В зависимости от того, какие компоненты были установлены на вашем ком-

пьютере ранее, вы можете получить предложение перегрузить компьютер посередине процесса установки.

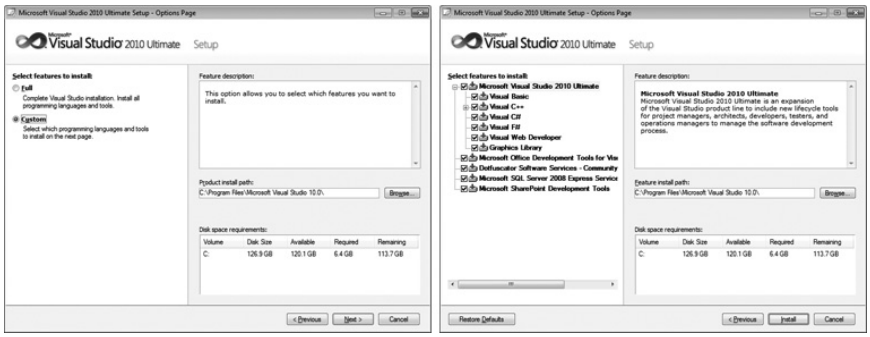


Рис. 1.3

Инсталлировав все компоненты, вы увидите итоговое диалоговое окно, продемонстрированное на рис. 1.4, *справа*. Вы должны внимательно его просмотреть, чтобы убедиться в том, что в процессе инсталляции не возникло никаких ошибок.

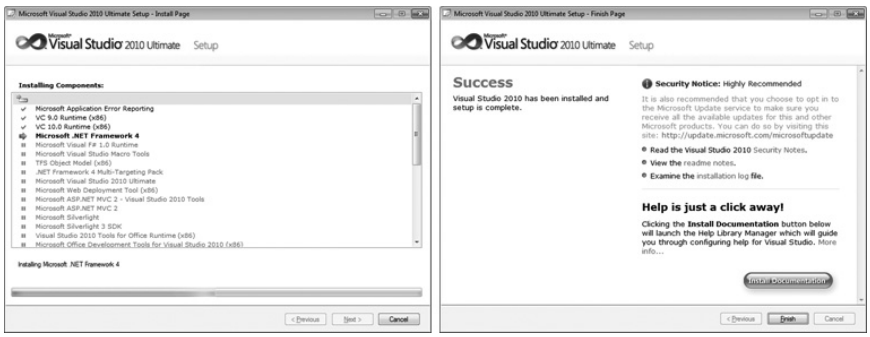


Рис. 1.4

Запуск системы Visual Studio 2010

При запуске на экране появится заставка Microsoft Visual Studio 2010. Как и большинство заставок, она содержит информацию о версии продукта и обладателе лицензии (рис. 1.5).

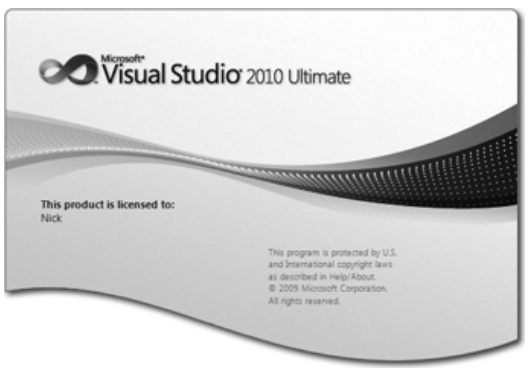


Рис. 1.5

При первом запуске системы Visual Studio 2010 заставка будет отображаться в течение нескольких секунд, а затем вы получите приглашение выбрать установки среды по умолчанию. На первый взгляд, довольно странно спрашивать людей, которые еще не использовали прродукт, как они собираются с ним работать. Тем не менее, поскольк-

ку компания Microsoft объединила большое количество языков и технологий в единой среде разработке, эта среда должна учитывать незначительные (а иногда и заметные) различия между разработчиками.

Если вы остановите свое внимание на разных опциях, перечисленных в этом списке (рис. 1.6), то увидите, что параметры среды, которые можно изменить, включают в себя положение и внешний вид разных окон, меню, панелей инструментов и даже "горячие" клавиши. Например, если в качестве установки по умолчанию вы выберете опцию General Development Settings, то на экране будут описаны изменения, которые произойдут в этом случае. Процедура изменения установок среды, принятых по умолчанию, на более поздних этапах работы описана в главе 3.



Интересно, что, хотя большая часть компонентов системы Visual Studio в настоящее время для отображения своего содержания использует технологию WPF, новая заставка в системе Visual Studio 2010 написана на машинном коде, т.е. появляется сразу же после запуска системы Visual Studio. На программирование волны, колыхающейся внизу экрана, было затрачено много ручной работы, поэтому просто восхищайтесь ею и терпеливо ждите, пока система Visual Studio загружается.



Рис. 1.6



Если вы ранее разрабатывали программы на языке Visual Basic .NET и теперь переходите на новую версию системы Visual Studio, то не используйте опцию Visual Basic Development Settings. Эта опция предназначена для разработчиков, использующих среду VB6, и просто приведет в ярость разработчиков, использующих язык Visual Basic .NET, поскольку они будут использовать разные "горячие" клавиши. Мы рекомендуем использовать общие настройки, так как они будут влиять на стандартные раскладки клавиатур без связи с другими языками программирования.

Среда разработки Visual Studio

В зависимости от выбранных установок среды, щелкнув на кнопке Start Visual Studio, вы, скорее всего, увидите диалоговое окно, сообщающее, что система Visual Studio является конфигурируемой системой разработки. После завершения этого процесса система Visual Studio 2010 открывается и сообщает, что готова к началу работы (рис. 1.7).

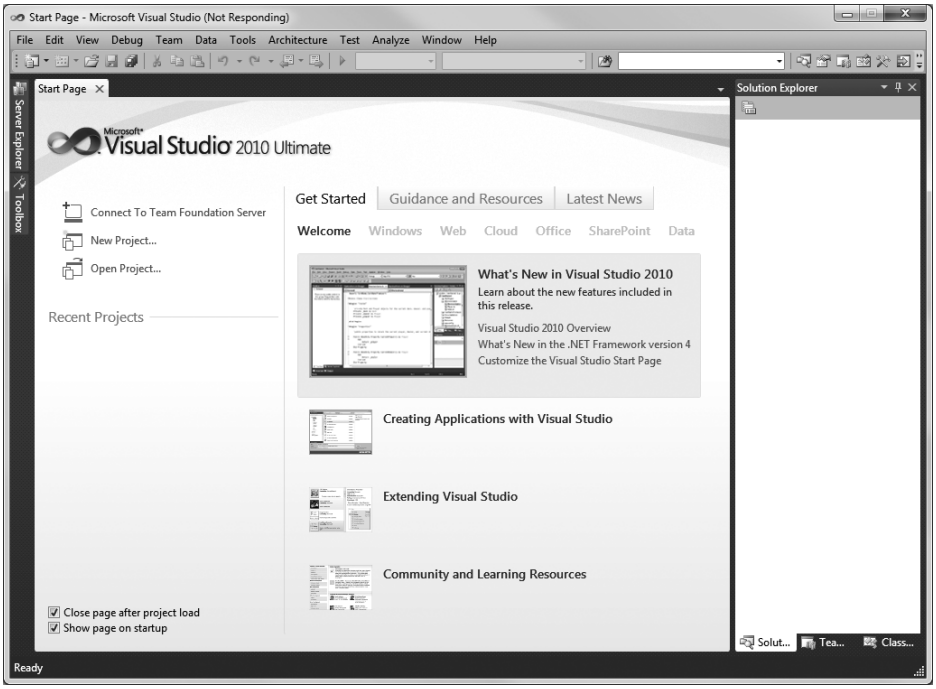


Рис. 1.7

Независимо от выбранных установок в центре экрана вы увидите окно Start Page. Однако содержание этого окна и окружающих его панелей инструментов и окон может изменяться.



Если щелкнуть на сером закругленном прямоугольнике с текстом “Download the latest information for developers to the Start Page”, то на экране появятся новости из канала RSS (RSS feed), посвященные настройкам среды, которые вы указали. Каждый элемент перечисляется в резюме внутри прямоугольника, что позволяет вам, щелкнув на кнопке, получить полное описание. Этот канал можно настроить, изменив свойство Start Page News Channel в узле Environment⇨Startup диалогового окна Options, доступ к которому открывает команда Options меню Tools.

Перед тем как переходить к созданию своего первого приложения, стоит вернуться назад и просмотреть компоненты Visual Studio 2010 IDE. Меню и панели инструментов располагаются в верхней части экрана, а окна второго уровня, или *панели* (panes), появляются в левой, правой и нижней части главного окна. В центре находится главное окно редактирования: каждый раз, когда вы открываете исходный

файл, XML-документ, форму или какой-нибудь другой файл, их содержание появляется в этом окне редактирования.

Одновременно с открытием файлов создается новая закладка, позволяющая переключаться между открытыми файлами. Окно редактирования окружено набором инструментальных окон, обеспечивающих пользователя дополнительной информацией и функциональностью. Если пользователь выбрал стандартные установки, то на экране по умолчанию отображаются окна Solution Explorer и Class View в правой части и окна Server Explorer и Toolbox в левой части. Инструментальные окна в левой части экрана находятся в *свернутом* (collapsed), или *открепленном* (unpinned), состоянии. Если щелкнуть на заголовке инструментального окна, оно откроется; когда фокус перемещается на другие компоненты или пользователь переводит курсор в другую область экрана, инструментальные окна снова сворачиваются. Когда инструментальное окно развернется, вы увидите три пиктограммы в правой верхней части экрана (рис. 1.8, *слева*).

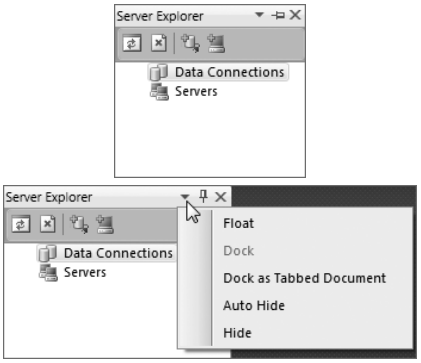


Рис. 1.8

Если хотите, чтобы инструментальное окно оставалось в *развернутом* (expanded), или *закрепленном* (pinned), состоянии, щелкните на средней пиктограмме, которая выглядит как булавка. Эта булавка повернется на 90 градусов, чтобы показать, что окно теперь закреплено. Щелкнув на пиктограмме со знаком ×, вы закроете окно. Если впоследствии захотите повторно открыть это или другое инструментальное окно, выберите его в меню View.



К некоторым инструментальным окнам, таким как окна потоков и контрольных выражений, нет доступа через меню View. В большинстве случаев доступ к этим окнам открывается с помощью альтернативных меню; например, доступ к окнам, связанным с отладкой программ, открывается с помощью меню Debug.

На рис. 1.8, *справа*, показано контекстное меню, появляющееся после щелчка на первой пиктограмме (стрелке, направленной вниз). Каждый пункт в этом списке отражает отдельный способ настройки инструментального окна. Как легко догадаться, команда Float позволяет разместить инструментальное окно в любом месте экрана, независимо от главного окна интегрированной среды разработки. Это удобно, если у вас есть несколько экранов, поскольку в этом случае можно переместить разные инструментальные окна на дополнительный экран и открыть окна редактирования настолько широко, насколько это возможно. Выполнив команду Dock as Tabbed Document, можно создать дополнительную закладку для инструментального окна в окне редактирования. Эффективные методы управления рабочей средой с помощью закрепления и открепления инструментальных окно описываются в главе 4.

Разработка, создание, отладка и развертывание вашего первого приложения

Завершив краткий обзор интегрированной среды разработки Visual Studio 2010, рассмотрим процедуру создания простого приложения, которая продемонстрирует

работу с некоторыми из упомянутых компонентов. Разумеется, этим приложением будет сакраментальная программа “Hello, World”, о которой должен знать каждый разработчик. Ее можно написать как на языке Visual Basic .NET, так и на языке C#. Это зависит от того, на каком из этих языков вам удобнее работать.

1. Начнем с команды File⇨New⇨Project, которая открывает диалоговое окно New Project, как показано на рис. 1.9. Если вы уже работали с более ранними версиями системы Visual Studio, то обнаружите, что это диалоговое окно подверглось значительному изменению. Оно по-прежнему содержит дерево в левой части диалогового окна, группирующего шаблоны по языкам программирования и технологиям, но теперь в правом левом углу есть поле поиска. Правая панель в этом диалоговом окне содержит дополнительную информацию о выбранном шаблоне проекта. И наконец, пользователь может выбрать версию платформы .NET. Каркас для этого приложения выбирается с помощью выпадающего меню в верхней части диалогового окна.

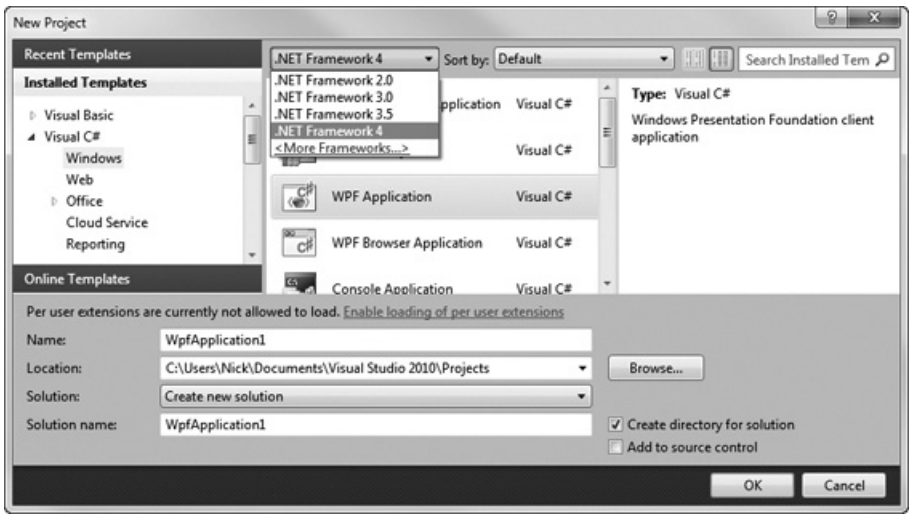


Рис. 1.9

Выберите команду WPF Application в окне Templates (она расположена под корневыми узлами Visual Basic и Visual C# или под дочерним узлом Windows) и в списке Name выберите пункт GettingStarted, прежде чем щелкнуть на кнопке OK. В результате должен отобразиться новый проект приложения WPF, включающий в себя отдельное окно запуска и содержащийся в решении GettingStarted, как показано в окне Solution Explorer на рис. 1.10. Это окно запуска автоматически открывается в окне визуального конструктора, чтобы пользователь имел представление о внешнем виде окна приложения во время его запуска. Обратите внимание на то, что инструментальное окно Properties появляется в правом нижнем углу.

2. Щелкните на инструментальном окне Toolbox. В результате откроется окно с пиктограммой в виде булавки, фиксирующей его в открытом состоянии. Для того чтобы поместить в окно элементы управления, выберите соответствую-

щие элементы в списке Toolbox и перетащите их на форму. В качестве альтернативы можете дважды щелкнуть на элементе управления, и система Visual Studio автоматически добавит его в окно.

3. Добавьте на форму кнопки и текстовые поля, чтобы она выглядела так, как на рис. 1.11. Выберите текстовое поле и инструментальное окно Properties (для того чтобы автоматически открыть окно Properties, можно нажать клавишу <F4>). Измените название текстового поля на txtToSay. Повторите эту процедуру для элемента Button, назвав ее btnSayHello и установив для свойства Content значение Say Hello!

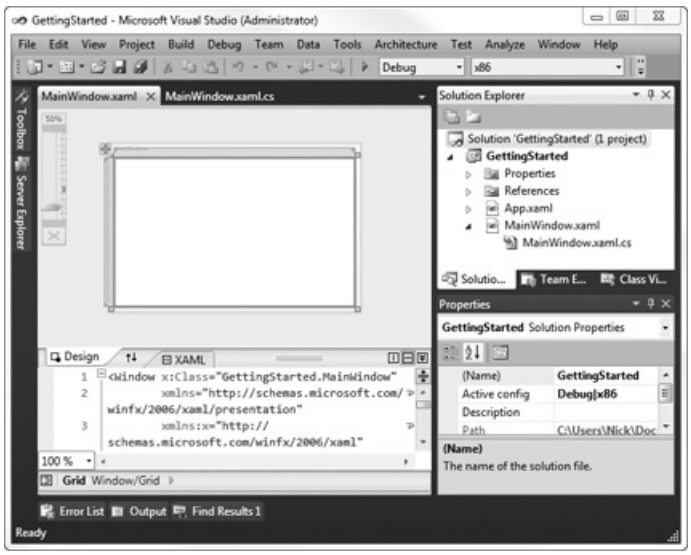


Рис. 1.10

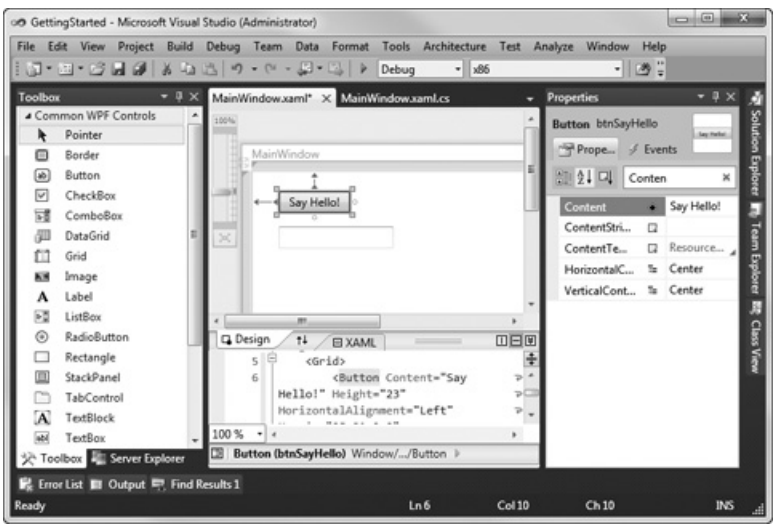


Рис. 1.11

Любое свойство можно быстро найти, набрав его имя в поле поиска, расположенном под полем Name. На рис. 1.11 показано, что в этом поле можно ввест и слово "Conten", чтобы сократить список соответствующих свойств и найт и свойства Content.

Обратите также внимание на то, что после добавления элементов управления в окно после текста соответствующего пункта появляется звездочка (*), означающая, что в этом пункте внесены изменения, которые еще не сохранены. Если вы попытаетесь закрыть этот пункт, не сохранив изменений, то система попросит вас сделать это. При создании приложения любой не сохраненный файл автоматически сохраняется в ходе процесса конструирования решения.



Следует помнить, что некоторые файлы, например файл решения, модифицируются, когда вы вносите изменения в среде Visual Studio 2010, без уведомления пользователя. Если вы попытаетесь выйти из приложения или закрыть решение, система предложит вам сохранить эти изменения.

4. Отмените выбор всех элементов управления, а затем дважды щелкните на кнопке. В результате не только откроется окно редактирования с кодом формы, но и будет создан и связан с приложением обработчик щелчков на кнопке. На рис. 1.12 показано окно с кодом формы после вставки одной строки, выводящей на экран эхо сообщения.

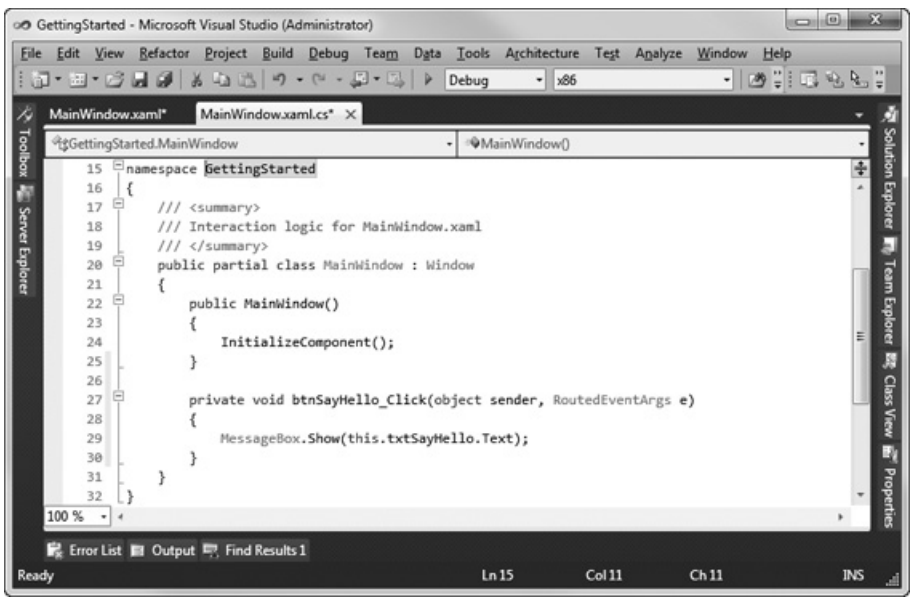


Рис. 1.12

5. Перед тем создать и выполнить приложение, установите курсор в строке, содержащей имя `Messagebox.Show`, и нажмите клавишу <F9>. Это позволит вам установить *контрольную точку* (breakpoint). Когда вы запустите приложение, нажав клавишу <F5>, а затем щелкнете на кнопке “Say Hello! ”, выполнение программы будет прервано в этом месте. Момент достижения контрольной

точки показан на рис. 1.13. *Подсказка* (data tip), появляющаяся на экране в момент, когда курсор мыши находится над указанной строкой, представляет собой содержание свойства txtToSay.Text.

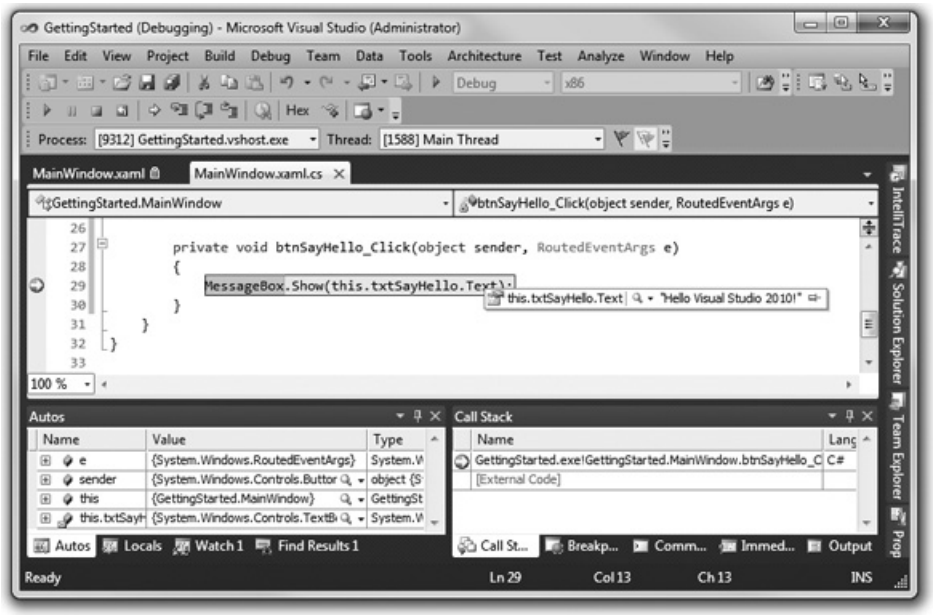


Рис. 1.13

Окно Visual Studio на рис. 1.13 значительно отличается от предыдущих окон, поскольку в его нижней части видны многочисленные новые инструментальные окна, а в верней части показаны новые командные панели. Если вы остановите приложение, то система Visual Studio восстановит предыдущие окна. Система Visual Studio 2010 поддерживает две разные схемы размещения объектов: во время проектирования и во время выполнения приложения. Меню, панели и окна по умолчанию имеют разный вид, когда вы редактируете проект и когда выполняете приложение. Можете изменить эти настройки по своему вкусу, и система Visual Studio 2010 запомнит их.

6. На последнем этапе происходит развертывание приложения. Независимо от того, какое приложение вы создаете — Windows Forms, WPF или веб, — система Visual Studio 2010 позволяет опубликовать ваше решение. Дважды щелкните на узле Properties в окне Solution Explorer и выберите узел Publish, чтобы показать варианты публикации вашего приложения, как показано на рис. 1.14.

На рис. 1.14 показан путь к локальному каталогу для публикации, но вы можете также выбрать сетевую папку, каталог IIS или сайт FTP. Выбрав место, в котором вы хотите опубликовать свое решение, щелкните на кнопке Publish Now, и ваше решение будет опубликовано в указанном месте.

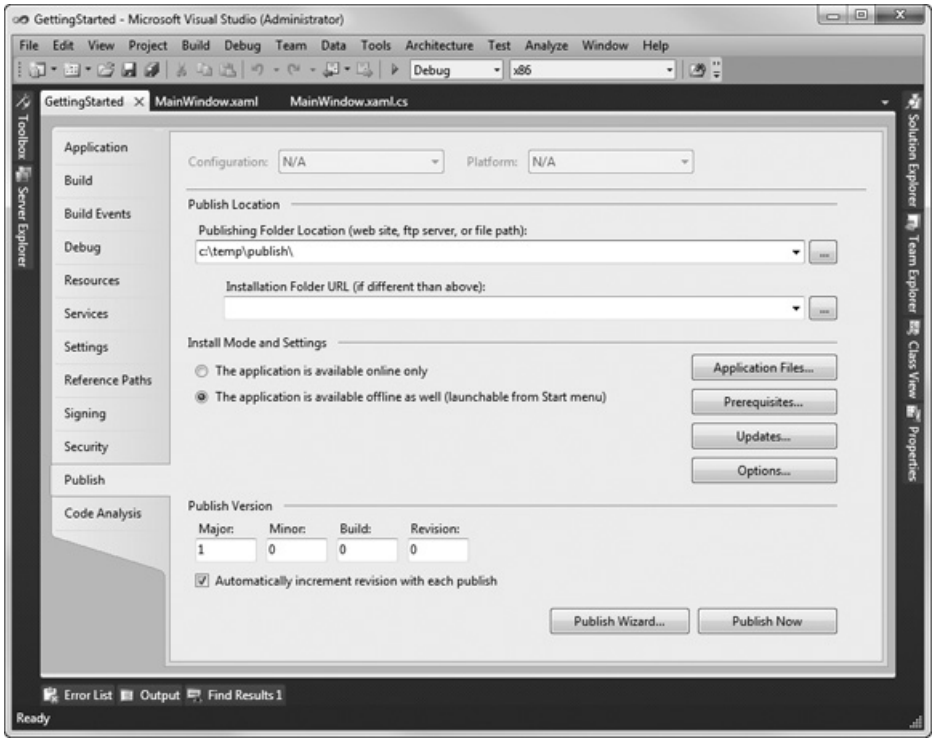


Рис. 1.14

Резюме

Вы ознакомились с тем, как взаимодействуют разнообразные компоненты системы в процессе создания приложения. Перечислим типичные этапы создания решения

1. Используйте меню File для создания решения.
2. Воспользуйтесь окном Solution Explorer, чтобы найти окно, которое необходимо отредактировать, и дважды щелкните на пункте, чтобы он появился в окне рабочего пространства.
3. Перетащите необходимые компоненты из окна Toolbox в окно приложения.
4. Поочередно выберите окно и каждый компонент, а также отредактируйте их свойства в окне Properties.
5. Дважды щелкните на окне или на элементе управления, чтобы получить доступ к коду, лежащему в основе их графического интерфейса.
6. Используйте основное рабочее пространство, чтобы написать код и разработать графический интерфейс, переключаясь между ними с помощью закладок, расположенных вверху окна.
7. Используйте инструментальные панели для запуска программы.
8. Если возникла ошибка, просмотрите окна Error List и Output.

9. Сохраните проект с помощью инструментальной панели или меню команд и выйдите из системы Visual Studio 2010.

В последующих главах вы узнаете, как точнее настроить интегрированную среду разработки для своих потребностей и как система Visual Studio 2010 выдвигает множество предположений в ходе процесса разработки приложения. Вы также ознакомитесь с многими наиболее эффективными приемами работы с системой Visual Studio 2010, которые можно применить на практике.

2

Окна Solution Explorer, Toolbox и Properties

В ЭТОЙ ГЛАВЕ...

- Упорядочение файлов с помощью окна Solution Explorer
- Добавление в решение проектов, элементов и ссылок
- Работа с инструментальным окном Properties
- Добавление собственных свойств в инструментальное окно Properties

В главе 1 вы кратко ознакомились с множеством компонентов, образующих среду интегрированной разработки Visual Studio 2010. Теперь вы получите возможность поработать с тремя наиболее широко используемыми инструментальными окнами — Solution Explorer, Toolbox и Properties.

На протяжении этой и следующих глав мы будем упоминать сокращенные клавишные наборы команд, например <Ctrl+S>. В этих случаях мы полагаемся на общие установки, принятые по умолчанию (см. главу 1). В других профилях могут быть установлены другие комбинации клавиш.

Окно Solution Explorer

Создавая или открывая приложение или просто отдельный файл, система Visual Studio 2010 использует концепцию решения для связывания всех компонентов в единое целое. Как правило, решение состоит из одного или нескольких проектов, каждый из которых, в свою очередь, содержит множество элементов, связанных с ним. В прошлом такими элементами были обычные файлы, но в настоящее время все чаще проекты создаются из элементов, которые образованы из нескольких файлов или вообще не включают в себя ни одного файла. Более подробно проекты, структура решений и связь между элементами решений рассматриваются в главе 6.

Инструментальное окно Solution Explorer (<Ctrl+Alt+L>) обеспечивает удобное визуальное представление решения, проектов и элементов, как показано на рис. 2.1. На

этом рисунке представлены три проекта, образующих дерево: WPF-приложение на языке C#, служебная WCF-библиотека на языке C# и библиотека классов на языке VB.

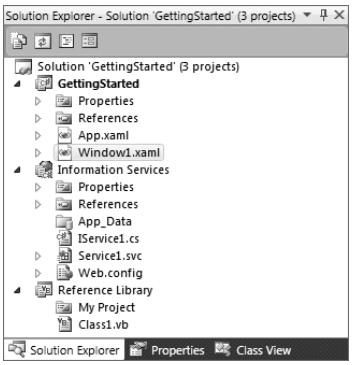


Рис. 2.1

С каждым проектом связана отдельная пиктограмма, которая, как правило, указывает тип проекта и язык, на котором он написан. У этого правила есть несколько исключений, например, *проекты установки* (setup projects) не имеют языка программирования, на котором они создаются.

Один из узлов обращает на себя особое внимание, поскольку он выделен полужирным шрифтом. Это значит, что этот проект является стартовым, иначе говоря, проектом, который запускается, когда вы выполняете команду Debug⇨Start Debugging или нажимаете клавишу <F5>. Для того чтобы сделать проект стартовым, щелкните на соответствующей пиктограмме правой кнопкой мыши и выполните команду Set as StartUp Project. Стартовыми можно сделать сразу несколько проектов. Для этого следует использовать диалоговое окно Solution Properties, которое открывается, когда вы щелкаете правой кнопкой мыши на узле Solution и выбираете пункт Properties.



При определенных параметрах окружения (см. раздел "Начало" в главе 1) узел Solution остается невидимым, если решение содержит только один проект. В этом случае трудно получить доступ к окну Solution Properties. Для того чтобы узел Solution стал видимым, можно либо добавить в решение еще один проект, либо выбрать пункт Always Show Solution в меню Projects и узел Solutions в диалоговом окне Options, которое открывается с помощью команды Tools⇨Options.

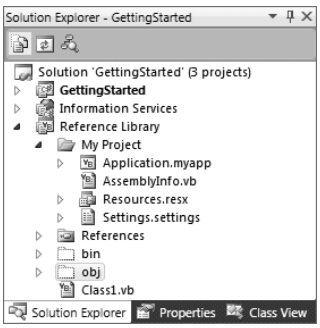


Рис. 2.2

Панель инструментов в верхней части окна Solution Explorer позволяет настроить внешний вид этого окна, а также задать комбинации клавиш для разного внешнего вида отдельных элементов. Например, первая кнопка – Show All Files – открывает листинг решения и позволяет показать на экране дополнительные файлы и папки (рис. 2.2). Вы видите, что узел My Project на самом деле состоит из нескольких файлов, содержащих настройки, ресурсы и информацию о сборке.

В этом развернутом представлении можно увидеть все файлы и папки, относящиеся к структуре проекта. К сожалению, при изменении файловой системы окно Solution Explorer не обновляется автоматически, чтобы отразить эти изменения. Для того чтобы обеспечить правильное представление списка файлов и папок, можно использовать вторую кнопку – Refresh.

Инструментальная панель Solution Explorer чувствительна к контексту, поэтому в зависимости от выбранного типа узла на ней отображаются разные кнопки. Это показано на рис. 2.2, на котором выбрана папка, не содержащаяся в проекте (о чем свидетельствует тусклый цвет пиктограммы), а остальные кнопки (см. рис. 2.1) остаются невидимыми. Короче говоря, если бы эти кнопки были видимыми, то их можно было бы использовать для просмотра кода (в данном случае файла `Window1.xaml.cs`) или открыть

проектировщик, отображающий на экране как схему, так и xaml-представление файла `Window1.xaml`. На рис. 2.2 показана также кнопка Class Diagram.

Типичные задачи

Кроме удобного способа управления проектами и элементами, окно Solution Explorer содержит динамическое контекстное меню, предоставляющее быстрое выполнение наиболее типичных задач, таких как сборка решений или отдельных проектов, обращение к менеджеру конфигурации сборки и открытие файлов. На рис. 2.3 продемонстрировано, как изменяется контекстное меню в зависимости от того, какой элемент был выбран в окне Solution Explorer.

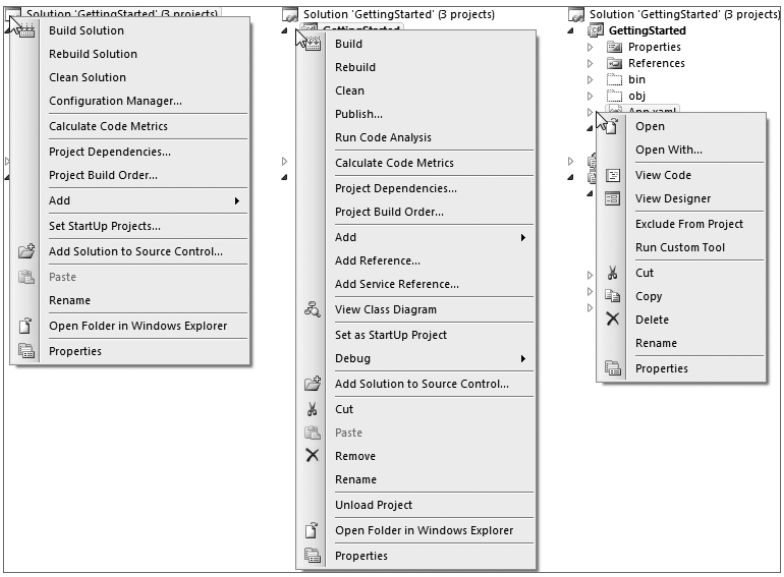


Рис. 2.3



Если вы еще не включили в свой проект диаграмму классов, щелкните на кнопке View Class Diagram. После этого система автоматически добавит в ваш проект эту диаграмму и все классы. Если проект содержит много классов, то эта процедура может занять довольно много времени и в результате возникнет большая и неудобная диаграмма классов. Намного лучше добавлять диаграммы классов вручную, поскольку в таком случае пользователь сохраняет над ними полный контроль.

Первые пункты в левом и среднем меню связаны с созданием либо всего решения, либо выбранного проекта. В большинстве случаев эффективнее всего выполнить команду Build, поскольку она создает лишь те проекты, которые подверглись изменениям. Однако в некоторых случаях пользователю необходимо выполнить команду Rebuild, которая создает все взаимосвязанные проекты независимо от их состояния. Если вы хотите всего лишь удалить все дополнительные файлы, возникшие в процессе создания решения, то можете выполнить команду Clean. Эта команда может оказаться полезной, если вы хотите упаковать свое решение и послать его кому-нибудь

в виде электронного письма и не хотите, чтобы при этом в него попали все временные файлы или файлы вывода, возникшие при создании решения.

Для большинства элементов в окне Solution Explorer первая команда контекстного меню похожа на команду, показанную в правом меню на рис. 2.2: по умолчанию команды Open и Open With... позволяют указать, как будет открыт конкретный элемент. Это особенно полезно при работе с ресурсными XML-файлами. Система Visual Studio 2010 открывает эти файлы, используя встроенный редактор ресурсов, но это не позволяет вам вносить определенные изменения и поддерживать все типы данных, которые вы хотели бы включить в проект (в главе 38 будет показано, как можно использовать свои собственные типы данных в файлах ресурсов). Используя команду Open With..., можно вызвать редактор XML Editor.



Контекстные меню для узлов Solution, Project и Folder содержат команду Open Folder in Windows Explorer, что позволяет быстро открыть окно Windows Explorer для поиска выбранного элемента и не искать в каталогах свое решение.

Добавление проектов и элементов

К основным действиям, выполняемым в окне Solution Explorer, относятся добавление, удаление и переименование проектов и элементов. Для того чтобы добавить новый проект в существующее решение, следует выполнить команду Add⇨New Project в контекстном меню, связанном с узлом Solution. В этом случае откроется диалоговое окно, показанное на рис. 2.4, которое с небольшими изменениями унаследовано от предыдущей версии системы Visual Studio. Теперь шаблоны проектов можно сортировать и искать. Панель, показанная на рис. 2.4, *справа*, содержит информацию о выбранном проекте, например его тип и описание (создание собственных шаблонов проектов и элементов, включая установку их свойств, описано в главе 15).

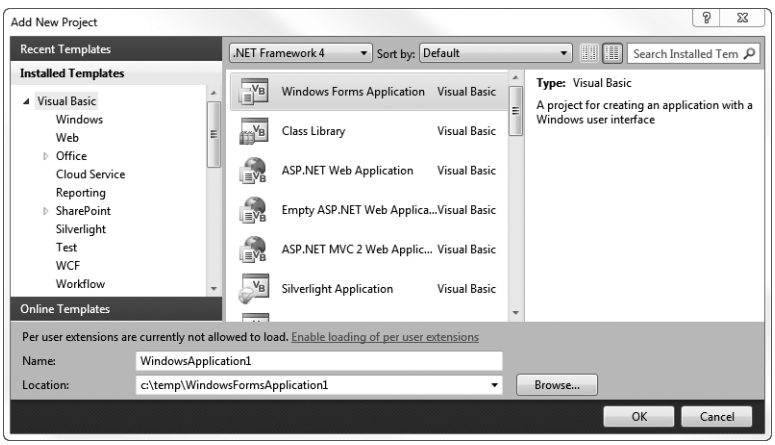


Рис. 2.4

В иерархии Installed Templates, появляющейся в левой части диалогового окна Add New Project, шаблоны в первую очередь упорядочены по языкам, а затем по технологиям. Среди шаблонов прежде всего следует выделить типы Office, позволяющие создавать как приложение, так и надстройки для документов, как это принято в большинстве приложений подобного рода. Хотя надстройки Office по-прежнему ис-

пользуют набор инструментов Visual Studio Tools for Office (VSTO), теперь он встроен в систему Visual Studio 2010 и не требует отдельной инсталляции. В главе 19 будет показано, как можно использовать эти типы проектов для создания надстроек для основных приложений пакета Office. Предусмотрены также закладки Recent Templates и Online Templates. Шаблоны Online Templates можно сортировать и искать точно так же, как Installed Templates, хотя критерии сортировки для них являются более широкими и включают в себя дату создания, рейтинги и частоту загрузки.

Кроме того, следует выделить еще одну особенность этого диалогового окна — возможность выбирать разные версии платформы. Если вы работаете со старым проектом и не хотите переводить его в новую версию платформы .NET Framework, то все равно можете воспользоваться новыми возможностями, такими как улучшенная технология IntelliSense. В противном случае пришлось бы инсталлировать как систему Visual Studio 2010, так и предыдущую версию, чтобы создавать проекты для более старых версий платформы. Выбор платформы также включен в критерии сортировки, что ограничивает список доступных шаблонов проекта лишь теми шаблонами, которые совместимы с выбранной версией платформы .NET Framework.



При открытии существующих решений или проектов в системе Visual Studio 2010 они проходят все этапы работы с мастером обновлений (более подробная информация приведена в главе 44), в ходе которых в файлы проектов и решений вносятся небольшие изменения. К сожалению, эти небольшие изменения связаны с внедрением дополнительных свойств, которые разрушают существующий процесс сборки, если вы используете предыдущую версию системы сборки проектов MSBuild. По этой причине вы будете вынуждены перевести всю свою команду разработчиков на версию Visual Studio 2010.

Одной из наиболее неудачных и плохо понимаемых особенностей системы Visual Studio является концепция проекта Web Site. Этот проект отличается от проекта Web Application, который можно добавить с помощью диалогового окна Add New Project. Для того чтобы добавить проект Web Site, необходимо выполнить команду Add⇨Web Site в контекстном меню, связанном с узлом Solution. Данная команда открывает диалоговое окно, похожее на то, которое показано на рис. 2.5. В этом окне можно выбрать тип создаваемого веб-проекта. В большинстве ситуаций при этом просто определяется тип создаваемого в проекте элемента, заданный по умолчанию.

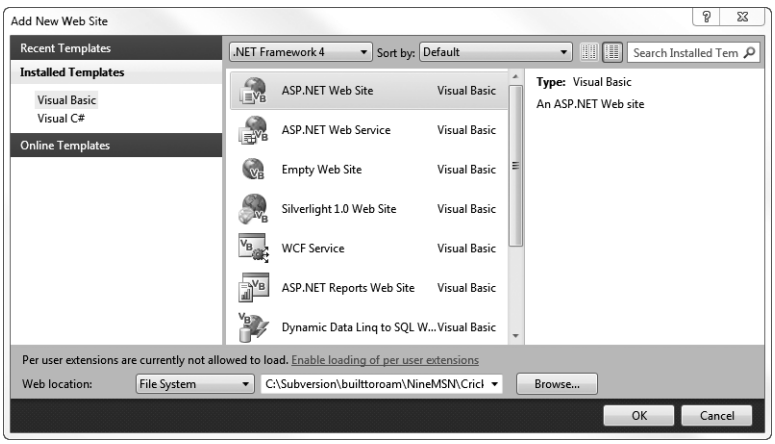


Рис. 2.5



Следует подчеркнуть, что типы веб-проектов, перечисленных на рис. 2.5, совпадают с типами, перечисленными в диалоговом окне Add New Project, связанном с узлом Web. Однако важно понимать, что они не порождают одинаковые результаты, поскольку между проектами Web Site (создаваемыми с помощью диалогового окна Add New Web Site) и проектами Web Application (создаваемыми с помощью диалогового окна Add New Project) существуют значительные различия, которые подробно описываются в главе 20.

Если вы работаете с одним или несколькими проектами, то сначала должны добавить элементы. Для этого выполните команду Add в контекстном меню, которое связано с узлом проекта в окне Solution Explorer. Первое подменю New Item запускает диалоговое окно Add New Item, как показано на рис. 2.6.

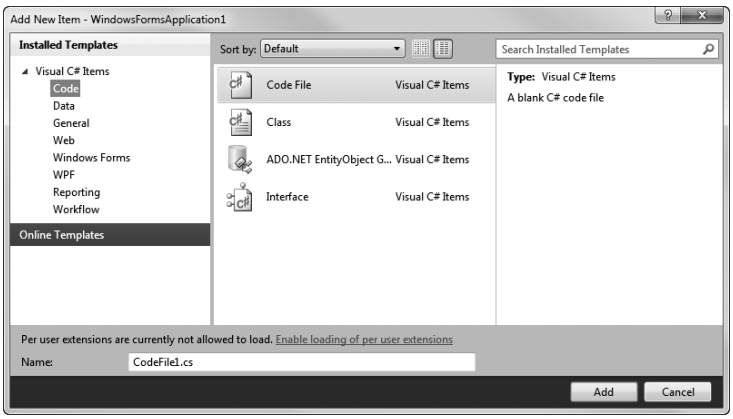


Рис. 2.6

Подобно диалоговым окнам New Project и New Web Site, диалоговое окно Add New Item подверглось "косметической операции". Кроме перечисления только тех шаблонных элементов, которые являются релевантными выбранному проекту, это диалоговое окно позволяет искать инсталлированные шаблоны, а также выходить в режим онлайн для поиска шаблонов, созданных другими.

Возвращаясь к контекстному меню Add, вы заметите большое количество заранее определенных сокращенных команд, таких как User Control и Class. Они делают немного больше, чем простой пропуск стадии поиска подходящего шаблона в диалоговом окне Add New Item. Это диалоговое окно по-прежнему открыто, поскольку вы должны задать имя создаваемого элемента.



Важно понимать, что вы добавляете в проект элементы, а не файлы. Хотя многие шаблоны содержат только один файл, некоторые команды, например Window или User Control, добавят в ваш проект несколько новых файлов.

Добавление ссылок

Выпуская новые технологии разработки программного обеспечения, их авторы каждый раз обещают более удобное повторное использование кода, но в действи-

тельности лишь некоторые из них выполняют данное обещание. Один из способов, с помощью которого Visual Studio 2010 поддерживает повторное использование компонентов, основан на ссылках проекта. Раскрыв любой проект, вы обнаружите множество библиотек .NET Framework, например System и System.Core, на которые для создания проекта необходимо как-то сослаться. По существу, ссылки позволяют компилятору распознавать типы, свойства, поля и имена методов, определенные в сборке. Если хотите повторно использовать класс из сторонней библиотеки или вашу собственную сборку, созданную по технологии .NET, вы должны добавить ссылку на нее с помощью контекстного меню Add Reference, ассоциированного с узлом проекта в окне Solution Explorer.

Когда вы открываете диалоговое окно Add Reference, показанное на рис. 2.7, система Visual Studio 2010 исследует локальный компьютер, кэш глобальной сборки Global Assembly Cache и ваше решение, чтобы представить список известных библиотек, на которые можно ссылаться. Этот список содержит ссылки как на компоненты .NET, так и на компоненты COM, внося их в разные разделы, а также ссылки на проекты и недавно использованные ссылки. В предыдущих версиях системы Visual Studio это диалоговое окно загружалось чрезвычайно медленно. Если компонент, который вам необходим, не указан в списке, выберите закладку Browse, позволяющую найти файл, содержащий искомые компоненты, непосредственно в файловой системе. Если вы заблаговременно загрузили список проектов в ваше решение, представили его как закладку по умолчанию и теперь неторопливо загружаете содержание для остальных закладок, то это диалоговое окно откроется на экране почти мгновенно.

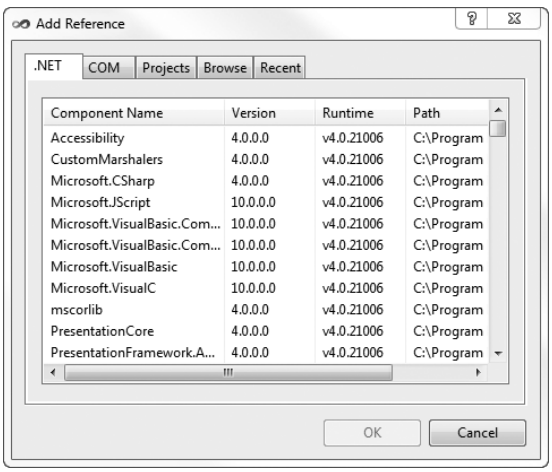


Рис. 2.7

Как и в других проектно-ориентированных средах разработки, начиная с первых версий среды VB, вы можете добавлять в проект ссылки, а не скомпилированные двоичные компоненты. Преимущество этой модели состоит в том, что теперь легче выполнять отладку компонента, на который указывает ссылка, и вы сможете гарантированно работать только с самыми последними версиями всех компонентов, хотя в больших решениях эта процедура может оказаться слишком неповоротливой.



Если ваше решение состоит из многих проектов (понятие "много" зависит от компьютера, но, как правило, крупным считается решение, состоящее более чем из двадцати проектов), то можете разделить его на несколько решений, соответствующих разным подмножествам проектов. В этом случае вы обогатите свой опыт отладки целого приложения и одновременно повысите производительность системы Visual Studio на этапах загрузки и построения решения. В качестве альтернативы можете создать разные конфигурации построения решения (подробнее о них — в главе 45), чтобы создать подмножество проектов.

Добавление ссылок на службы

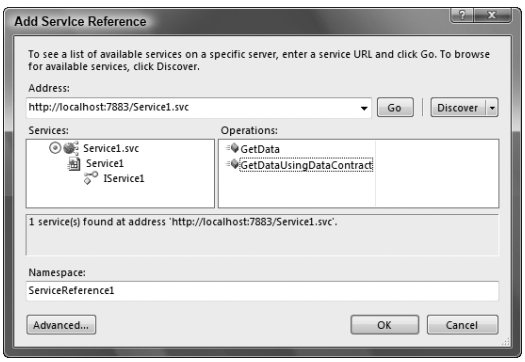


Рис. 2.8

Другой тип ссылок, который обслуживает проводник Solution Explorer, называется *ссылками на службу* (service references). В предыдущих версиях системы Visual Studio они назывались веб-ссылками, но с появлением программной модели Windows Communication Foundation (WCF) в систему Visual Studio была включена более общая команда меню Add Service Reference. Она открывает диалоговое окно Add Service Reference, показанное на рис. 2.8. В данном примере использована возможность открывать выпадающее меню с помощью кнопки Discover, чтобы просмотреть список Services в решении.

К сожалению, это диалоговое окно еще раз демонстрирует, что компания Microsoft не совсем правильно понимает потребности пользователей. Хотя само диалоговое окно допускает изменение размеров, область для вывода ответного сообщения о статусе этого не позволяет, тем самым затрудняя чтение генерируемых сообщений об ошибках. К счастью, если при попытке системы Visual Studio 2010 получить доступ к информации о службе возникла ошибка, появляется гиперссылка, с помощью которой можно открыть диалоговое окно Add Service Reference Error. Как правило, это позволяет получить достаточную информацию, чтобы устранить проблему.

В левом нижнем углу на рис. 2.8 показана кнопка Advanced. Диалоговое окно Service Reference Settings, которое открывается после щелчка на этой кнопке, позволяет вам указать, какие типы считаются частью ссылки на службу. По умолчанию считается, что все локальные системные типы соответствуют типам, опубликованным службой. Если это условие не выполняется, то уточните значения в области Data Type, которая является частью этого диалогового окна. В левом нижнем углу диалогового окна Service Reference Settings расположена кнопка Add Web Reference, позволяющая добавлять в решение более привычные веб-ссылки. Это может оказаться важным, если вы ограничены в ресурсах или пытаетесь поддерживать работу в нескольких системах. Добавление служб в собственное приложение описано в главе 31, посвященной программной модели WCF.

Окно Toolbox

Одно из основных преимуществ над другими интегрированными средами разработки состоит в том, что компания Microsoft предлагает пользователям реальную возможность перемещать элементы при проектировании как веб-, так и сложных клиентских приложений. Все эти элементы размещены в окне Toolbox (<Ctrl+Alt+X>), доступ к которому можно получить посредством меню View (рис. 2.9).

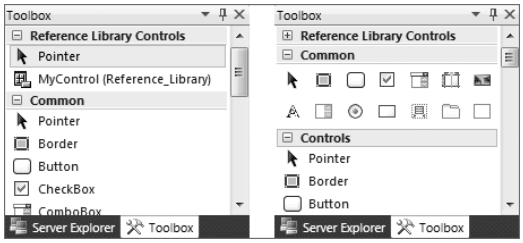


Рис. 2.9



Окно Toolbox имеет интересное свойство: вы можете скопировать фрагмент кода в него, щелкнув на области кода и перетащив ее в окно Toolbox. Вы можете также переименовать и переупорядочить ваши фрагменты кода, сделав их действительно полезными для презентаций или хранения часто используемых фрагментов.

Окно Toolbox содержит все доступные компоненты для активного в данный момент документа, открытого в главном рабочем окне. Ими могут быть визуальные компоненты, например кнопки и текстовые поля; невидимые, сервисные объекты, например таймеры или регистрационные журналы системных событий; и даже элементы дизайна, такие как классы и интерфейсные объекты, используемые в инструменте Class Designer.

Система Visual Studio 2010 группирует доступные компоненты, а не смешивает их в одну кучу. Такая группировка по умолчанию позволяет проще находить требуемые элементы контроля; например, компоненты для работы с данными содержатся в отдельной группе Data.

По умолчанию эти группы представлены в виде списка (см. рис. 2.9, *слева*). Каждый компонент представлен с помощью своей собственной пиктограммы и имеет название. Этим новая версия системы Visual Studio 2010 отличается от старого способа представления доступных объектов, в которых окно Toolbox просто представляло собой список пиктограмм, а пользователь должен был угадать, каким компонентам они соответствуют (см. группу Common Controls на рис. 2.9, *справа*). Вы можете изменять представление каждой группы элементов управления по отдельности. Для этого щелкните правой кнопкой мыши в любом месте области, где расположена группа, и снимите флажок с команды List View в контекстном меню.

Независимо от представления компонентов способ их использования в программе остался прежним: щелкните и перетащите желаемый компонент на форму активного документа или дважды щелкните на компоненте в системе Visual Studio, чтобы автоматически добавить его экземпляр. Визуальные компоненты, такие как кнопки и текстовые поля, появляются на форме, после чего пользователь может перемещать их, изменять размеры или фиксировать с помощью свойств сетки. Невизуальные компо-

ненты, такие как таймер, отображаются в виде пиктограмм с ассоциированными метками в области формы, предназначенной для невидимых компонентов, как показано на рис. 2.10.

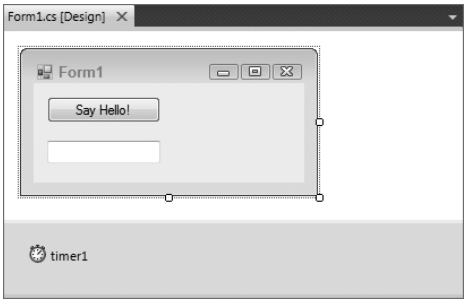


Рис. 2.10

В левом верхнем углу рис. 2.9 показана группа Reference Library Controls с единственным компонентом MyControl. По существу, строка "Reference_Library" – это имя библиотеки классов, определенной в этом же решении. Она содержит элемент управления MyControl. Когда вы начинаете создавать свои собственные компоненты или элементы управления, то, вместо того, чтобы предоставить вам вручную создавать новую закладку и повторять весь процесс добавления каждого элемента, система Visual Studio 2010 автоматически просматривает все проекты вашего решения. Как только компоненты или элементы управления будут идентифицированы (т.е. будет распознан любой класс, реализующий компонент `System.ComponentModel.IComponent` или `System.Windows.FrameworkElement` для программной модели WPF), в проекте будет создана новая закладка, на которую будут добавлены соответствующие элементы с пиктограммами и именами классов, заданными по умолчанию, в данном случае `MyControl` (см. рис. 2.9, *слева*). Когда вы используете компонент, в области невидимых элементов появляется соответствующая пиктограмма.



Система Visual Studio 2010 просматривает все проекты в вашем решении и перед, и после построения решения. Если решение состоит из многих проектов, то этот процесс может занять много времени. В таком случае следует подумать об отключении этой возможности, установив свойство AutoToolboxPopulate узла Windows Forms Designer в диалоговом окне Options равным `false` (Tools⇨Options).

Для того чтобы настроить внешний вид элементов в окне Toolbox, добавьте в проект ваш компонент или элемент в виде побитового изображения размером 16×16 пикселей. Затем выберите вновь вставленное побитовое изображение в окно Solution Explorer и перейдите к окну Properties. Убедитесь, что свойство Build установлено равным Embedded Resource. Теперь достаточно связать атрибут вашего элемента управления с атрибутом изображения ToolboxBitmap.

VB

```
<ToolboxBitmap(GetType(MyControl), "MyControlIcon.bmp")>
Public Class MyControl
```

C#

```
[ToolboxBitmap(typeof(MyControl), "MyControlIcon.bmp")]
public class MyControl
```

Этот атрибут использует ссылку на тип MyControl, чтобы идентифицировать сборку, из которой следует извлечь встроенный ресурс `MyControlIcon.bmp`. Другие перегруженные варианты этого атрибута в качестве единственного аргумента могут использовать путь к файлу. В этом случае вам даже не надо добавлять побитовое изображение в свой проект.



К сожалению, пользователь не имеет возможности изменить способ, которым автоматически генерируемые элементы отображаются в окне Toolbox. Однако, если вы вручную добавляете элемент в окно Toolbox и выбираете ваши компоненты, то можете увидеть свою пиктограмму. Кроме того, если вы используете компонент и перетаскиваете его на форму, то увидите пиктограмму в области невидимых компонентов проектировщика.

Следует отметить, что при настройке окна Toolbox и внешнего вида элементов управления в программной модели Windows Presentation Foundation (WPF) используется понятие хранилища метаданных, а не атрибуты.

Обычно это приводит к появлению дополнительных сборок, которые можно использовать для уточнения внешнего вида элементов управления как в системе Visual Studio 2010, так и в системе Expression Blend.

Настройка компонентов

Перечисление элементов в окне Toolbox в алфавитном порядке — хороший вариант, принятый по умолчанию, поскольку он позволяет найти неизвестный пользователю элемент. Однако, если вы используете только несколько компонентов и вам надоело прокручивать список вверх и вниз, можете создать свои собственные группы элементов управления и переместить существующие типы объектов.

Перемещение отдельного компонента не составляет труда. Найдите его в окне Toolbox, щелкните на нем и перетащите в новое место. Когда достигнете пункта назначения, отпустите кнопку мыши, и компонент будет перемещен в новое место списка. Точно так же можете переместить компонент в другую группу — просто перетаскивайте его вверх или вниз по окну Toolbox, пока не окажетесь в нужной группе. Эти действия можно выполнять как с представлением List, так и с представлением Icon.

Если хотите скопировать компонент из одной группы в другую, а не переместить его, то в ходе перетаскивания удерживайте нажатой клавишу <Ctrl>. Иногда удобно иметь свою собственную группу, чтобы хранить в ней элементы управления и компоненты, которые используются чаще других. Для того чтобы создать новую группу в окне Toolbox, щелкните правой кнопкой мыши в любом месте окна Toolbox и выполните команду Add Tab. Внизу окна Toolbox будет добавлена новая пустая закладка, которой вы должны присвоить имя. После того как вы назовете закладку, добавьте в нее компоненты, следуя инструкциям, описанным в этом разделе.

Если вы впервые запускаете систему Visual Studio 2010, то элементы в каждой группе будут упорядочены по алфавиту. Однако после перемещения элементов вы можете обнаружить, что они перепутаны, и решите просто начать все сначала. Все, что вам для этого необходимо, — щелкнуть правой кнопкой мыши в группе и выполнить команду Sort Items Alphabetically.

По умолчанию элементы управления добавляются в окно Toolbox в порядке следования имен их классов. Это значит, что вы столкнетесь с непонятными именами, особенно, если захотите добавить в окно Toolbox элементы управления COM. Система Visual Studio 2010 позволяет пользователю изменять имена компонентов на что-нибудь более понятное.

Для того чтобы изменить имя компонента, щелкните правой кнопкой мыши в окне Toolbox и выполните команду Rename Item. На месте первоначального названия появится поле редактирования, в которое можете ввести удобное для вас имя, даже если

оно содержит специальные символы. Если же вы еще больше запутались, работая с компонентами из непривычных групп, и потеряли ориентацию, то выполните команду Reset Toolbox из того же самого контекстного меню, появляющегося после щелчка правой кнопкой мыши. Эта команда восстанавливает исходное состояние всех групп в окне Toolbox, причем компоненты сортируются по алфавиту и находятся в группах, которым принадлежали изначально.

Добавление компонентов

Иногда оказывается, что конкретный компонент, который вам нужен, отсутствует в списке Toolbox. Большинство основных компонентов .NET в этом списке есть, но некоторых нет. Например, компонент WebClient class в списке Toolbox по умолчанию не указан. Управляемые приложения могут также использовать компоненты COM. После добавления в окно Toolbox объекты COM могут быть использованы точно так же, как обычные компоненты .NET, и если они закодированы правильно, то вы можете программировать работу с ними как обычно, используя окно Properties и ссылаясь на их методы, свойства и события в своем коде.

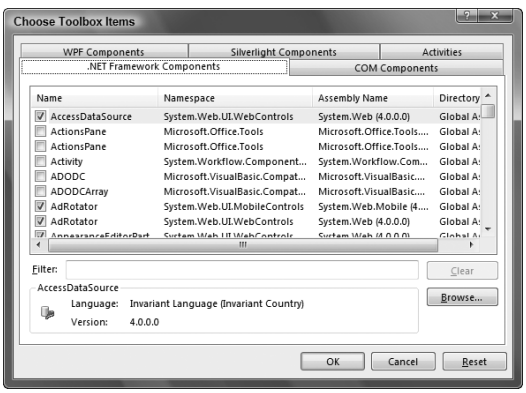


Рис. 2.11

Для того чтобы добавить компонент в свое окно Toolbox, щелкните правой кнопкой мыши на группе компонентов, в которую хотите добавить новый компонент, и выполните команду Choose Items. Через какое-то время (на медленном компьютере этот процесс может занять несколько секунд, потому что машина должна просмотреть кэш .NET, чтобы определить все возможные компоненты, которые можно выбрать) отобразится список .NET Framework components, как показано на рис. 2.11.



Помните: команда Selecting Reset Toolbox удаляет все группы команд, созданные пользователем, поэтому сначала убедитесь, что вам действительно необходимо выполнить эту функцию!

Просмотрите список, чтобы найти элемент, который хотите добавить в окно Toolbox, и установите соответствующий флажок. Можете также добавлять несколько элементов одновременно, выбирая их до того, как щелкнуть на кнопке OK. В то же время можете удалять элементы из окна, сбрасывая соответствующие флажки в списке. Обратите внимание на то, что в этом случае соответствующие элементы удаляются из всех групп, которым они принадлежат, а не только из той группы, которую вы редактируете в данный момент.

Если вам покажется трудным найти требуемый элемент, воспользуйтесь окном Filter, фильтрующим список по имени, пространству имен и имени сборки. В редких ситуациях список может не содержать элемент вообще. Это может произойти с нестандартными компонентами, например, созданными вами или не зарегистрированными в кэше Global Assembly Cache (GAC). Вы по-прежнему можете добавить их, используя

кнопку Browse, чтобы просмотреть реальные файлы на компьютере. Выбирая и отменяя выбор требуемого элемента, щелкните на кнопке OK, чтобы сохранить настройки окна Toolbox.

Компоненты COM, WPF, Silverlight Components и Workflow Activities можно добавить точно так же. Просто щелкните на соответствующей закладке в диалоговом окне, чтобы увидеть список доступных, правильно зарегистрированных компонентов COM. Можете также использовать кнопку Browse, чтобы найти элементы управления, которых нет в списке.

Окно Properties

Одно из наиболее часто используемых инструментальных окон в системе Visual Studio 2010 — окно Properties (<F4>), показанное на рис. 2.12. Окно Properties состоит из сетки свойств и является контекстно-зависимым, отображая только релевантные свойства выделенного элемента, независимо от того, является ли этот элемент узлом в окне Solution Explorer или элементом формы. Каждая строка отображает свойство, имеющее имя и соответствующее значение, записанные в двух столбцах. На рис. 2.12, *справа*, показана обновленная сетка свойств для приложений WPF, содержащая пиктограмму предварительного просмотра и возможности поиска.

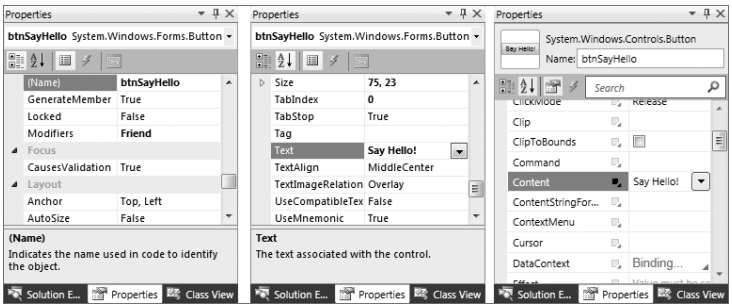


Рис. 2.12

Окно Properties позволяет группировать свойства или сортировать их по алфавиту — можно изменять режимы, используя две первые кнопки, расположенные в верхней части окна. Это окно имеет встроенные редакторы для многих системных типов, таких как цвета, шрифты, якоря и стыковки, которые вызываются, когда вы щелкаете на столбце значений изменяемых свойств. Когда вы выбираете свойство, как показано на рис. 2.12, *посередине*, его имя подсвечивается, а в левой области сетки свойств появляется его описание.

В окне Properties свойства, доступные только для чтения, выделяются серым цветом. Их значения изменить невозможно. В то же время значение `SayHello` для свойства Text (рис. 2.12, *слева*) выделено полужирным шрифтом. Это означает, что данное значение не является заданным по умолчанию. На рис. 2.12, *справа*, между именем и значением свойства Text стоит черный квадратик. Это значит, что пользователь должен задать значение. Проверив следующий код, сгенерированный конструктором, вы увидите, что в сетке свойств у каждого свойства существует строка, выделенная полужирным шрифтом. Добавление строки кода для каждого отдельного свойства элемента управления значительно увеличивает время заполнения формы.



VB

```
Me.btnSayHello.Location = New System.Drawing.Point(12, 12)
Me.btnSayHello.Name = "btnSayHello"
Me.btnSayHello.Size = New System.Drawing.Size(100, 23)
Me.btnSayHello.TabIndex = 0
Me.btnSayHello.Text = "Say Hello!"
Me.btnSayHello.UseVisualStyleBackColor = True
```

Сниппет кода Form1.Designer.vb



C#

```
this.btnSayHello.Location = new System.Drawing.Point(12, 12);
this.btnSayHello.Name = "btnSayHello";
this.btnSayHello.Size = new System.Drawing.Size(100, 23);
this.btnSayHello.TabIndex = 0;
this.btnSayHello.Text = "Say Hello!";
this.btnSayHello.UseVisualStyleBackColor = true;
```

Сниппет кода Form1.Designer.cs



Для веб- и WPF-приложений набор свойств в окне Properties сохраняется в виде файлов разметки с расширением `aspx` или `xaml` соответственно. Как и в проектировщике форм Windows, в окне Properties отображаются только те свойства, которые постоянно хранятся в файле разметки.

Кроме отображения свойств выделенных элементов, окно Properties предлагает средства для подключения обработчиков событий. В окне Properties (рис. 2.13, *слева*) показано представление события, связанного с четвертой кнопкой, — знак молнии в верхней части окна Properties. В данном случае можно увидеть, что существует обработчик события, связанного со щелчком на кнопке. Для того чтобы обработать другое событие, можете выбрать существующие методы в раскрывающемся списке, связанном со столбцом значений. В этом случае будет создан новый обработчик, который будет связан с данным событием. Если будете использовать первый метод, то заметите, что в списке перечислены только методы, соответствующие сигнатуре события.

Некоторые компоненты, такие как DataGridView, связаны со множеством команд или комбинаций клавиш, которые можно выполнить с помощью окна Properties. На рис. 2.13, *справа*, можно увидеть команды для компонента DataGridView: Edit Columns и Add Column. Щелкните на командной ссылке, чтобы открыть диалоговое окно для выполнения этого действия.

Если окно Properties невелико по сравнению с экраном, то бывает трудно прокрутить список свойств. Если щелкнуть правой кнопкой мыши на сетке свойств, то можно сбросить флажки Command и Description и таким образом скрыть эти разделы окна Properties.

Расширение окна Properties

Ранее вы узнали, что система Visual Studio 2010 выделяет свойства, подлежащие изменению, полужирным шрифтом. У вас должен был возникнуть вопрос: откуда система Visual Studio 2010 знает, каким должно быть значение по умолчанию? Ответ заключается в том, что когда окно Properties просматривает объект, чтобы определить, какие свойства должны отражаться в сетке свойств, она перебирает большое количество атрибутов проекта. Эти атрибуты можно использовать для управления отображением свойств, выбора редактора, который должен использоваться для изменения

значений, а также для выбора значения, заданного по умолчанию. Для того чтобы показать, как можно использовать эти атрибуты для создания своих собственных компонентов, начнем с добавления простого автоматического свойства.

VB

```
Public Property Description As String
```

Сниппет кода MyControl.vb

C#

```
public string Description { get; set; }
```

Сниппет кода MyControl.cs

Атрибут Browsable

По умолчанию в сетке свойств отображаются все открытые свойства. Однако вы можете явно управлять его поведением, добавив атрибут Browsable. Если вы установите это свойства равным `false`, то оно не будет показано в сетке свойств.

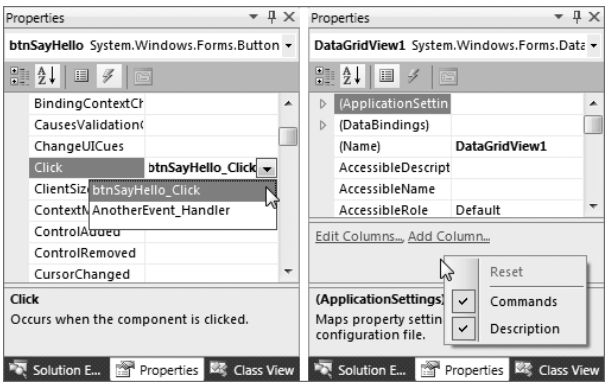


Рис. 2.13

VB

```
<System.ComponentModel.Browsable(False)>
Public Property Description As String
```

Сниппет кода MyControl.vb

C#

```
[System.ComponentModel.Browsable(false)]
public string Description { get; set; }
```

Сниппет кода MyControl.cs

Атрибут DisplayName

Атрибут DisplayName вполне очевиден: он позволяет изменять имя свойства. В данном случае мы можем изменить имя свойства, которое появится в сетке свойств с Description на VS2010 Description.



VB

```
<System.ComponentModel.DisplayName("VS2010 Description")>
Public Property Description As String
```

Сниппет кода MyControl.vb



C#

```
[System.ComponentModel.DisplayName("VS2010 Description")]
public string Description { get; set; }
```

Сниппет кода MyControl.cs

Атрибут Description

Кроме возможности задавать имя свойства по своему усмотрению, целесообразно также предоставлять описание, которое должно появляться в нижней части окна Properties, когда выбирается соответствующее свойство. Это гарантирует, что пользователи компонента знают его свойства.



VB

```
<System.ComponentModel.Description("My first custom property")>
Public Property Description As String
```

Сниппет кода MyControl.vb



C#

```
[System.ComponentModel.Description("My first custom property")]
public string Description { get; set; }
```

Сниппет кода MyControl.cs

Атрибут Category

По умолчанию любое видимое свойство размещается в группе Misc, если окно Properties имеет сгруппированное представление. Используя атрибут Category, вы сможете разместить ваше свойство в любой из существующих групп, таких как Appearance или Data, или в новой группе, если укажете имя группы, которой еще нет.



VB

```
<System.ComponentModel.Category("Appearance")>
Public Property Description As String
```

Сниппет кода MyControl.vb



C#

```
[System.ComponentModel.Category("Appearance")]
public string Description { get; set; }
```

Сниппет кода MyControl.cs

Атрибут DefaultValue

Как вы уже знаете, система Visual Studio 2010 выделяет свойства, которые были изменены по сравнению со своим начальным значением или значением, заданным по умолчанию. Атрибут DefaultValue определяет, как система Visual Studio 2010 выполняет поиск значений, заданных по умолчанию для указанного свойства.



VB

```
Private Const cDefaultDescription As String = "<enter description>"
<System.ComponentModel.DefaultValue(cDefaultDescription)>
Public Property Description As String = cDefaultDescription
```

Сниппет кода MyControl.vb



C#

```
private const string cDefaultDescription = "<enter description>";
private string mDescription = cDefaultDescription;
[System.ComponentModel.DefaultValue(cDefaultDescription)]
public string Description
{
    get
    {
        return mDescription;
    }
    set
    {
        mDescription = value;
    }
}
```

Сниппет кода MyControl.cs

В данном случае, если значение свойства Description установлено равным "<enter description>", система Visual Studio 2010 удаляет строку кода, которая устанавливает это свойство. Если вы изменили это свойство и хотите вернуть его первоначальное значение, щелкните правой кнопкой мыши в окне Properties и выполните команду Reset в контекстном меню.



Следует подчеркнуть, что атрибут DefaultValue не устанавливает начальное значение вашего свойства. Если вы задаете атрибут DefaultValue, то рекомендуется одновременно установить начальное значение точно так же, как и в предшествующем коде.

Атрибут AmbientValue

Одно из свойств, которое мы все принимаем как должное, но лишь некоторые действительно понимают до конца, — это концепция сочетаемости объекта с внешним окружением. Типичными примерами являются цвета и шрифты переднего и заднего плана: если их не установить явно в окне Properties, они наследуются, но не от базового класса, а от своих родительских элементов управления. Более широкое определение концепции сочетаемости трактует его как свойство, которое принимает свое значение от другого источника.

Как и атрибут DefaultValue, атрибут AmbientValue используется для того, чтобы служить индикатором для системы Visual Studio 2010, когда она не должна добавлять код в файл проектировщика. К сожалению, при работе со свойствами сочетаемости невозможно жестко задать в коде значение, с которым следует сравнивать текущее значение, поскольку оно зависит от значения, заданного в источнике свойств. Из-за этого, когда вы определяете атрибут AmbientValue, вы поручаете проектировщику искать функцию ShouldSerialize*PropertyName*. В данном случае эта функция называется ShouldSerializeDescription, и этот метод вызывается для того, чтобы определить, следует ли хранить текущее значение свойства в исходном файле проектировщика.



VB

```
Private mDescription As String = cDefaultDescription
<System.ComponentModel.AmbientValue(cDefaultDescription)>
Public Property Description As String
    Get
            If Me.mDescription = cDefaultDescription AndAlso
                                 Me.Parent IsNot Nothing Then
```

```
            Return Parent.Text
        End If
        Return mDescription
    End Get
    Set(ByVal value As String)
        mDescription = value
    End Set
End Property

   Private Function ShouldSerializeDescription() As Boolean
     If Me.Parent IsNot Nothing Then
        Return Not Me.Description = Me.Parent.Text
     Else
        Return Not Me.Description = cDefaultDescription
    End If
End function
```

Сниппет кода MyControl.vb



C#

```
private string mDescription = cDefaultDescription;
[System.ComponentModel.AmbientValue(cDefaultDescription)]
public string Description{
     get{
           if (this.mDescription == cDefaultDescription & &
               this.Parent != null){
               return Parent.Text;
           }
       return mDescription;
      }
      set{
         mDescription = value;
      }
}
private bool ShouldSerializeDescription(){
      if (this.Parent != null){
           return this.Description != this.Parent.Text;
      }
      else{
      return this.Description != cDefaultDescription;
      }
}
```

Сниппет кода MyControl.cs

Когда вы создаете элемент управления с данным свойством, начальное значение должно быть установлено равным константе `DefaultDescription`, но в проектировщике вы можете увидеть значение, соответствующее значению `Parent.Text`. В исходном файле проектировщика может также не оказаться строки, явно задающей это свойство, что отражено в окне Properties, в котором это значение не выделено полужирным шрифтом. Если вы измените значение этого свойства на что-нибудь другое, отличающееся от константы DefaultDescription, то увидите, что оно будет выделено полужирным шрифтом и в исходный файл проектировщика будет добавлена соответствующая строка. Если вы восстановите значение этого свойства, то базовое значение будет восстановлено и станет равным значению, определенному атрибутом AmbientValue, но вы увидите лишь, что окно по-прежнему отображает значение `Parent.Text`.

Резюме

В этой главе вы увидели в действии три основных инструментальных окна. Зная правила манипулирования этими окнами, вы сэкономите много времени на этапе разработки. Однако истинная мощь системы Visual Studio 2010 проявляется, когда вы начинаете инкорпорировать свой опыт разработчика в собственные компоненты. Это может оказаться полезным, даже если ваши компоненты будут использоваться только в вашей организации. Эффективно используя проектировщик, можно улучшить не только процесс проектирования с помощью собственных элементов управления, но и повысить производительность создаваемого приложения.

3

Параметры и настройки

В ЭТОЙ ГЛАВЕ...

- Настройка стартовой страницы системы Visual Studio 2010
- Выбор параметров
- Управление внешним видом окон

Теперь, когда вы в общих чертах знаете устройство системы Visual Studio 2010, настало время научиться настраивать интегрированную среду разработки по своему усмотрению. В этой главе показано, как управлять инструментальными окнами, оптимизировать окно редактирования кода, чтобы получить максимальное пространство для просмотра, и изменять шрифты и цвета, чтобы снизить утомляемость разработчиков.

По мере совершенствования системы Visual Studio количество ее настроек, с помощью которых можно оптимизировать разработку программ, значительно увеличилось.

К сожалению, если вы время от времени не исследуете диалоговое окно Options (Tools⇨Options) самым тщательным образом, то, скорее всего, упустили одну или две возможности, которые могли оказаться полезными. В данной главе вы найдете много рекомендаций, которые заслуживают дальнейшего изучения.

Большое количество надстроек системы Visual Studio добавляют в окно Options свои собственные узлы, поскольку это позволяет выполнять настройку конфигурации системы за один шаг. Следует также подчеркнуть, что профили настроек, продемонстрированные в главе 1, содержат лишь малую часть всех возможностей. Для того чтобы увидеть весь список опций, следует установить флажок Advanced.

Стартовая страница

По умолчанию, когда вы начинаете новый сеанс работы с системой Visual Studio 2010, открывается стартовая страница. Вы можете изменить эту настройку с помощью узла

Environment⇨Startup в диалоговом окне Options. В качестве альтернативы можно открыть окно Home Page (его можно настроить с помощью узла Environment⇨Web Browser), последнее загруженное решение, диалоговое окно нового проекта или вообще ничего не делать.

Причина, по которой большинство разработчиков предпочитают панель Start Page, заключается в том, что она представляет собой полезную стартовую позицию, с которой можно перейти к любому действию. На рис. 3.1, *слева*, мы видим ссылки для связи с сервером Team Foundation Server, а также ссылки для создания или открытия проектов. Существует также список недавних проектов, позволяющих вам быстро открыть проекты, над которыми вы недавно работали. Когда курсор мыши перемещается по левой части проекта, на экране появляется значок в виде горизонтальной булавки. После щелчка левой кнопкой мыши эта булавка изменяет ориентацию на вертикальную. Это значит, что данный проект был включен в список Recent Projects. В качестве альтернативы можно щелкнуть на проекте правой кнопкой мыши и либо открыть содержащий его каталог (это полезно, если вы хотите лишь найти проект на диске, а не открыть его), либо удалить проект из списка. В левом нижнем углу есть два флажка, с помощью которых можно указывать, следует ли закрывать панель Start Page после открытия проекта и надо ли его открывать с самого начала. Если по какой-то причине панель Start Page закрыта и вы хотите открыть ее снова, выполните команду View⇨Start Page.

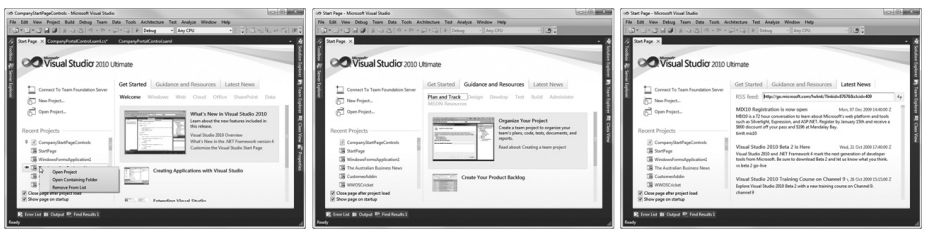


Рис. 3.1

В правой части панели Start Page расположен ряд закладок. Закладка Get Started (см. рис. 3.1, *слева*) содержит разнообразные подразделы множества тем, включая Windows, Web, Office и Cloud. Щелкнув на любой из них, можно найти информацию о том, как работать с указанными технологиями и новинками системы Visual Studio 2010.

В закладке Guidance and Resources, которая находится посередине окна (см. рис. 3.1, *справа*), расположены ссылки на темы, посвященные наилучшим методам работы, советам и другим сведениям о проектировании, конструировании, тестировании и развертывании программного обеспечения с помощью системы Visual Studio 2010.

В заключение отметим, что закладка Latest News (см. рис. 3.1, *справа*) позволяет получить самые свежие новости о системе Visual Studio 2010 и пакете .NET Framework. Для отображения новостей в панели Start Page можете использовать либо RSS-канал, либо свой собственный канал новостей.

Настройка окна Start Page

В системе Visual Studio 2010 панель Start Page играет роль элемента управления WPF, внедренного в оболочку интегрированной среды разработки. Благодаря этому

существует возможность настраивать панель Start Page так, чтобы она отображала релевантные информацию и действия. Вместо изменения панели Start Page система Visual Studio поддерживает окна, созданные пользователем, или окна Custom Start Pages. По умолчанию эти свойства отключены, поэтому, прежде чем модифицировать панель Start Page, установите флажок Allow Custom Start Page в закладке Environment⇨Startup диалогового окна Options.

Поскольку панель Start Page представляет собой всего лишь элемент управления WPF, можете просто создать новый проект управления WPF и разработать свою собственную страницу с нуля. Однако в большинстве ситуаций намного проще скопировать панель Start Page и настроить ее по-своему. Для этого необходимо скопировать панель Start Page, заданную по умолчанию, из инсталляционного каталога в свой каталог Documents.

1. Скопируйте содержимое каталога `C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\StartPages\<culture>` в каталог `C:\Users\<user name>\Documents\Visual Studio 10\StartPages` (возможно, вам придется создать этот каталог, если он не предусмотрен по умолчанию).
2. Дважды щелкните на файле `StartPage.csproj`, чтобы открыть проект в системе Visual Studio 2010 (рис. 3.2). Как видите, вы можете изменить панель Start Page, используя либо средства проектирования элементов WPF, либо непосредственно на странице XAML. Страница XAML состоит из множества разделов. Наилучшие места для начала изменений — те, в которых в существующих закладках определены дескрипторы TabItem. Для того чтобы создать дополнительную закладку, скопируйте одну из существующих закладок и модифицируйте ее, включав в нее собственное содержание. Например, для того чтобы добавить информацию о своей компании, добавьте после закладки Latest News дескриптор, приведенный ниже.

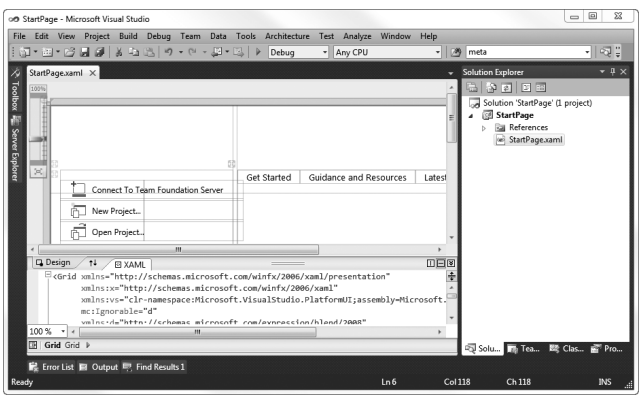


Рис. 3.2

XAML

```
<!-- Company Tab -->
<TabItem Header="Company" Height="Auto" x:Uid="Company_Tab"
         DataContext="{Binding Links.Content, Converter=
                       {StaticResource
StringToXmlDataProviderConverter}}">
```

```
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"></RowDefinition>
                <RowDefinition Height="*"></RowDefinition>
            </Grid.RowDefinitions>
            <TextBlock Foreground="#E8E8E8" Margin="15" TextWrapping="Wrap"
                   x:Uid="Information_Paragraph" >
                The Company has put a lot of effort into writing this custom
                Start Page so that you can quickly access information
                relevant to your job, the projects you are working on etc.
            </TextBlock>
        </Grid>
    </TabItem>
```

Сохранив это изменение, щелкните на закладке Start Page в системе Visual Studio 2010 и убедитесь, что ваши изменения уже учтены и на странице есть закладка Company (рис. 3.3).



Рис. 3.3

Разумеется, если вас не устраивает внешний вид панели Start Page, установленный по умолчанию, удалите все существующие элементы и замените их своими собственными, наделив их собственной информацией и функциональными возможностями. Для этого выберите эти элементы либо в проектировщике, либо на странице XAML и удалите их. Затем используйте любой элемент управления WPF, расположенный в окне Toolbox, чтобы разработать свой интерфейс панели Start Page.

Код, лежащий в основе элементов управления

Для того чтобы расширить функциональные возможности панели Start Page, можно предусмотреть выполнение определенного кода при наступлении конкретного события. Сама по себе панель Start Page не имеет исходного файла, в котором можно было бы записать код обработчика событий. Однако можно инкапсулировать функциональные возможности в пользовательском элементе управления и использовать его на своей панели Start Page.

Открыв свой проект Start Page, процедура создания которого описана в предыдущем разделе, добавьте новый проект, в котором будет содержаться элемент управления. Для этого выберите команду File⇨Add⇨New Project и команду WPF User Control

Library из узла Windows, расположенного под узлом VB или C#. Присвойте проекту имя, например CompanyStartPageControls, и щелкните на кнопке OK. Затем следуйте инструкциям, приведенным ниже, чтобы создать кнопку, запускающую веб-сайт вашей компании.

1. Переименуйте файл UserControl1.xaml как CompanyPortalControl.xaml в окне Solution Explorer.
2. Дважды щелкните на файле CompanyPortalControl.xaml в окне Solution Explorer, чтобы открыть его в окне проектировщика.
3. Замените XAML-файл следующей разметкой, создающей кнопку с обработчиком события Click.

XAML

```
<UserControl x:Class="CompanyStartPageControls.CompanyPortalControl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid>
        <Button Click="LaunchWebsite">Company Website</Button>
    </Grid>
</UserControl>
```

4. Щелкните правой кнопкой мыши на файле CompanyPortalControl.xaml в окне Solution Explorer и выполните команду View Code.
5. Замените код следующими строками, реализующими обработчик события Click, запускающим веб-сайт компании.

VB

```
Class CompanyPortalControl
     Private Sub LaunchWebsite(ByVal sender As Object,
                               ByVal e As RoutedEventArgs)

System.Diagnostics.Process.Start("http://www.builttoroam.com")
     End Sub
End Class
```

C#

```
namespace CompanyStartPageControls{
    public partial class CompanyPortalControl : UserControl{
       public CompanyPortalControl(){
           InitializeComponent();
       }
       private void LaunchWebsite(object sender, RoutedEventArgs e){
           System.Diagnostics.Process.Start(
                                 @"http://www.builttoroam.com");
       }
    }
}
```

Итак, вы создали элемент управления, в котором инкапсулированы его основные функциональные возможности. Для его использования на вашей панели Start Page выполните следующие действия.

1. Щелкните правой кнопкой мыши на проекте StartPage в окне Solution Explorer и выполните команду Add Reference. В закладке Projects выберите проект `CompanyStartPageControls` и щелкните на кнопке OK.
2. Выполните повторную сборку решения, выполнив команду Build⇨Rebuild Solution.
3. Дважды щелкните на файле `StartPage.xaml` в окне Solution Explorer, чтобы открыть его в окне проектировщика.
4. В окне Toolbox вы увидите закладку CompanyStartPageControls, на которой находится кнопка CompanyPortalControl. Перетащите этот элемент на панель StartPage ниже информации о компании. XAML-код элемента Company TabItem должен выглядеть так, как показано ниже.

XAML

```
<!-- Company Tab -->
<TabItem Header="Company" Height="Auto" x:Uid="Company_Tab"
         DataContext="{Binding Links.Content, Converter=
                        {StaticResource
StringToXmlDataProviderConverter}}">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition Height="*"></RowDefinition>
        </Grid.RowDefinitions>
        <TextBlock Foreground="#E8E8E8" Margin="15" TextWrapping="Wrap"
               x:Uid="Information_Paragraph" >
            The Company has put a lot of effort into writing this custom
            Start Page so that you can quickly access information
            relevant to your job, the projects you are working on etc.
        </TextBlock>
        <my:CompanyPortalControl Grid.Row="1" HorizontalAlignment="Left"
         Margin="6,12,0,0" VerticalAlignment="Top" />
    </Grid>
</TabItem>
```

Если теперь вы сохраните все файлы (<Ctrl+Shift+S>) и выберете закладку Start Page, то увидите сообщение об ошибке, потому что сборку, содержащую элемент управления CompanyPortalControl, найти невозможно. Для того чтобы устранить эту проблему, необходимо скопировать динамическую библиотеку `c:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\StartPageAssemblies`. Этот каталог по умолчанию не существует, поэтому его необходимо создать заранее, чтобы скопировать туда сборку. Скопировав библиотеку `CompanyStartPageControls.dll`, перезапустите систему Visual Studio. Ваша стартовая страница теперь должна выглядеть так, как на рис. 3.4

Щелкнув на кнопке Company Website, вы запустите портал вашей компании в вашем внешнем веб-браузере непосредственно из панели Start Page.



Рис. 3.4

Компоновка окон

Если вы не знакомы с системой Visual Studio, то поведение многочисленных инструментальных окон может показаться вам неправильным, потому что они появляются в случайных местах, а затем появляются и исчезают по мере того, как вы пишете код (этап проектирования) и выполняете его (время выполнения). На самом деле система Visual Studio 2010 запоминает расположение инструментальных окон для каждого из этих режимов работы, что позволяет оптимизировать разработку и отладку кода.

Открывая разные элементы в окне Solution Explorer, вы увидите множество инструментальных панелей, пересекающих верхнюю часть экрана, в зависимости от типа открытого файла. Каждая инструментальная панель имеет встроенную связь с конкретными расширениями файла, поэтому система Visual Studio знает, какую инструментальную панель отобразить, когда открывается тот или иной файл. Если вы закроете инструментальную панель в тот момент, когда файл с соответствующим расширением открыт, система Visual Studio запомнит ее, когда вы в будущем откроете файл с таким же расширением.



Связь между инструментальными панелями и расширениями файлов можно изменить с помощью команды Customize (Tools⇨Customize). Выберите соответствующую инструментальную панель и щелкните на кнопке Reset.

Просмотр окон и инструментальных панелей

Если инструментальное окно или панель были закрыты, то иногда их трудно найти снова. К счастью, доступ к большинству наиболее часто используемых инструментальных окон можно получить с помощью окна View. Другие инструментальные окна, в основном связанные с отладкой, расположены под меню Debug.

Все инструментальные панели, доступные в системе Visual Studio 2010, перечислены в пункте меню View⇨Toolbars. Каждая инструментальная панель, которую можно увидеть в текущий момент, отмечена галочкой возле соответствующего пункта меню. Вы можете также открыть список инструментальных панелей, щелкнув правой кнопкой мыши на любом свободном месте в области инструментальной панели в верхней части окна Visual Studio.

Если инструментальная панель является видимой, можете настроить ее с помощью кнопок, команды View⇨Toolbars⇨Customize или меню Tools. В качестве альтернативы, как показано на рис. 3.5, выбрав стрелку, направленную вниз и расположенную в конце инструментальной панели, можно открыть список всех кнопок, доступных на панели, и проверить кнопки, которые вы хотите на ней увидеть.

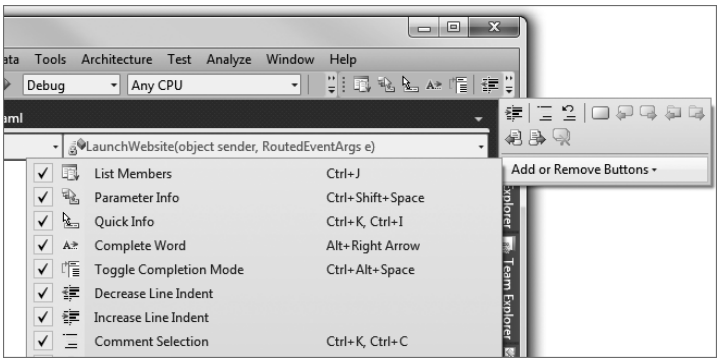


Рис. 3.5

Навигация по открытым элементам

Открыв множество элементов, вы обнаружите, что вышли за пределы области редактирования и не видите больше все открытые закладки. Разумеется, вы можете вернуться в окно Solution Explorer и выбрать конкретный элемент. Если он уже открыт, то будет показан на экране без возвращения в его сохраненное состояние. Однако этот способ неудобен, если требуется найти элемент управления в окне Solution Explorer.

К счастью, в системе Visual Studio 2010 есть много комбинаций клавиш для получения списка открытых элементов. Как и большинство документно-ориентированных приложений, система Visual Studio имеет меню Windows. Когда вы открываете элемент, его заголовок добавляется в нижний раздел этого меню. Для того чтобы отобразить открытый элемент, только что выбранный в меню Windows, щелкните на пункте меню Windows. В результате откроется модальное диалоговое окно, в котором можно выбрать требуемый элемент.

В качестве альтернативы можно использовать выпадающее меню в окне области закладок области редактирования. На рис. 3.6 показан раскрывающийся список открытых элементов, в котором можете выбрать требуемый элемент.

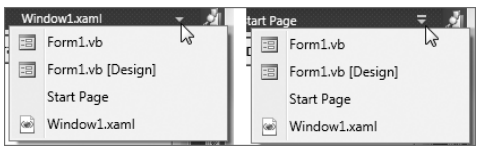


Рис. 3.6

Правая часть рис. 3.6 совпадает с левой частью, за исключением выпадающей пиктограммы. Это меню также содержит стрелку, направленную вниз, но на этот раз она имеет черту, параллельную верхней границе окна. Эта черта означает, что количество закладок превышает число закладок, которые могут поместиться в верхней части области редактирования.

Еще один способ навигации по открытым элементам — нажать комбинацию клавиш <Ctrl+Tab>, которая открывает временное окно, показанное на рис. 3.7. Это окно явля-

ется временным потому, что исчезнет, когда вы отпустите клавишу <Ctrl>. Но пока это окно остается открытым, для перемещения из одного открытого окна в другое можно использовать клавиши навигации (со стрелками) или клавишу табуляции (<Tab>).

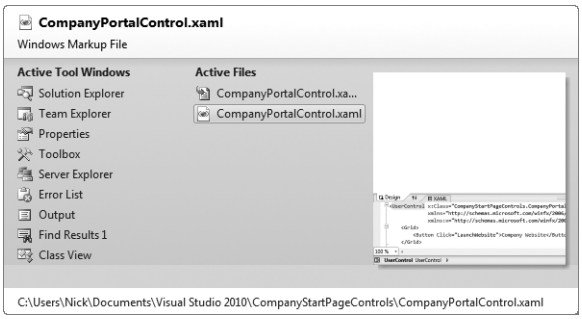


Рис. 3.7

Окно, открывающееся после нажатия клавиш <Ctrl+Tab>, разделено на три части: активные инструментальные окна, активные файлы (эти элементы должны быть действительно активными, потому что некоторые элементы могут соответствовать нескольким файлам) и окно просмотра выбранного элемента. По мере увеличения количества активных файлов или инструментальных окон нижняя граница этого окна отодвигается вниз, пока количество элементов не достигнет пятнадцати, после чего создается дополнительный столбец.



Если вы достигли момента, когда возникает дополнительный столбец активных файлов, то следует подумать о том, не закрыть ли некоторые или все не используемые файлы. Чем больше файлов открыто в системе Visual Studio 2010, тем больше памяти она использует и тем медленнее работает.

Щелкнув правой кнопкой мыши на закладке, содержащей открытый элемент, вы увидите скрытое контекстное меню, обеспечивающее быстрый доступ к наиболее часто выполняемым задачам, таким как сохранение или закрытие файла, связанного с данной закладкой. Наиболее полезными действиями являются команды Close All But This и Open Containing Folder. Эти команды самоочевидны: первая из них закрывает все закладки, за исключением той, на которой вы щелкнули, чтобы открыть контекстное меню, а вторая — открывает каталог, содержащий файл, в окне Windows Explorer. Поскольку все окна являются плавающими, у них есть команды Float и Dock as Tabbed Document, которые становятся доступными в зависимости от того, в каком состоянии находится закладка.

Плавающие окна

Каждое инструментальное окно имеет положение, заданное по умолчанию, которое можно восстановить с помощью команды меню View. Например, команда View⇨ Toolbox открывает инструментальное окно Toolbox, прикрепленное к левому краю окна Visual Studio. Помимо того, что инструментальное окно может быть открыто и прикреплено в краю экрана, оно может находиться в двух состояниях: закрепленном и открепленном. Как уже говорилось в главе 1, вы можете переключаться между этими состояниями, щелкая на значке вертикальной булавки, чтобы открепить инструментальное окно, или на значке горизонтальной булавки, чтобы закрепить его.

Открепленное окно смещается от края рабочей области интегрированной среды разработки, оставляя видимым дескриптор, в котором отображается заголовок данного инструментального окна. Такая анимация может показаться надоедливой и занимает много времени, если инструментальные окна оставить открепленными. Эту ситуацию можно контролировать либо с помощью узла Environment диалогового окна Options, указав, должна ли система Visual Studio автоматически учитывать опыт зрительного восприятия, основанный на эффективности работы клиента, либо отключив опцию Enable rich client visual experience.

Большинство разработчиков устраивает расположение инструментальных окон, заданное по умолчанию, но иногда они хотят самостоятельно указать место, где должно появляться инструментальное окно. Среда Visual Studio 2010 имеет изощренную систему управления компоновками инструментальных окон. В главе 1 было показано, как использовать выпадающие меню, а также кнопки Pin и Close, расположенные в верхней части инструментального окна, чтобы сделать *плавающим* (floating) или *закрепляемым* (dockable) не только окно, но и часть основной области редактирования (используя команду Tabbed Document).

Если инструментальное окно является закрепляемым, то у вас есть множество возможностей для управления его местоположением. На рис. 3.8 показана верхняя часть окна Properties, которое было перетащено вправо из своего первоначального положения, заданного по умолчанию. Для того чтобы начать перетаскивание, убедитесь, что инструментальное окно находится в закрепленном состоянии, а затем щелкните либо на заголовке, расположенном в верхней части окна, либо на ярлыке, расположенном внизу окна, и перетащите курсор мыши в требуемом направлении. Если вы щелкнете на заголовке, то увидите, что все инструментальные окна в этом разделе интегрированной среды разработки также переместятся. Щелкнув на ярлыке, вы переместите только соответствующее инструментальное окно.

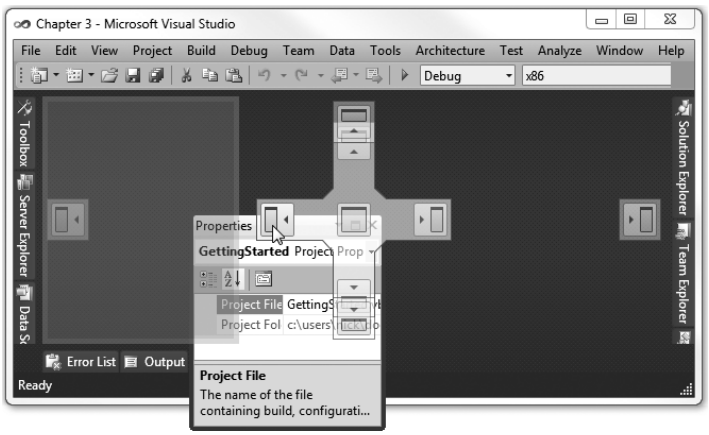


Рис. 3.8

Перетаскивая инструментальное окно по рабочей области системы Visual Studio 2010, вы увидите полупрозрачные пиктограммы, появляющиеся в разных местах. Эти пиктограммы оказывают большую помощь при выборе точной позиции. На рис. 3.9 показано, что окно Data Sources было прикреплено к левому краю. Затем, когда окно Properties было установлено над левой пиктограммой средней части экрана, в существующем инструмен-

тальном окне вновь возникла голубая тень. Это означает, что инструментальное окно Properties было прикреплено к правому краю инструментального окна Data Sources и будет видимым, если выбрана именно эта компоновка окна. Если бы была выбрана левая пиктограмма, то инструментальное окно Properties было бы снова прикреплено к левому краю рабочей области, но на этот раз слева от окна Data Sources.

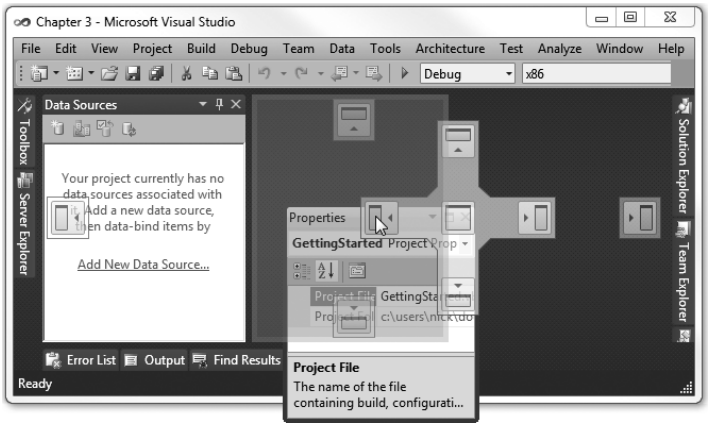


Рис. 3.9

Кроме того, если инструментальное окно Properties поместить поверх инструментального окна Data Sources, как показано на рис. 3.10, то средняя часть будет размещена поверх существующего инструментального окна. Это значит, что инструментальное окно Properties будет размещено в существующем инструментальном окне. Перемещая окно поверх разных квадрантов, вы увидите, что голубая тень означает, где будет размещено инструментальное окно, когда вы отпустите кнопку мыши. На рис. 3.10 это означает, что инструментальное окно Properties появится ниже инструментального окна Data Sources.

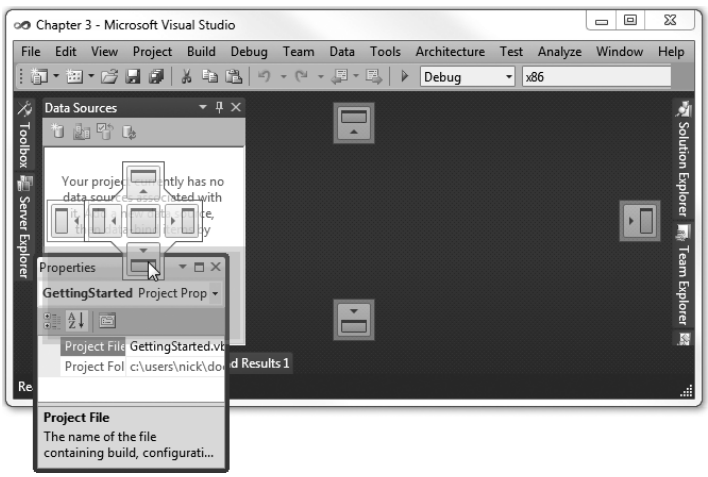


Рис. 3.10



Если вы работаете с большим экраном или с несколькими экранами, целесообразно уделить особое внимание удобной компоновке часто используемых инструментальных окон. Если вы работаете с несколькими экранами, то использование плавающих инструментальных окон означает, что вы можете размещать их далеко от основной области редактирования, обеспечивая максимальную площадь экрана. Если у вас маленький экран, то придется постоянно указывать, какое окно является видимым, и в этом случае важно хорошо разбираться в компоновке закрепленных окон.

Область редактирования

Как большинство интегрированных сред разработки, система Visual Studio 2010 построена на основе главного окна редактирования кода. Со временем система эволюционировала и теперь представляет собой нечто большее, чем простой текстовый редактор. Несмотря на то что большинство разработчиков по-прежнему проводят значительное время, работая над кодом программ в области редактирования, все большее количество проектировщиков имеют возможность решать такие задачи, как создание форм, настройка параметров проекта и редактирование ресурсов. Независимо от того, пишете ли вы код или создаете форму проекта, вам придется провести много времени в области редактирования системы Visual Studio 2010. По этой причине важно знать, как наладить среду так, чтобы ваша работа была более эффективной.

Шрифты и цвета

Одной из первых настроек, которую презентаторы программ обычно изменяют в системе Visual Studio, стремясь повысить читабельность кода, являются шрифты и цвета в области редактирования. Однако это делают не только презентаторы. Выбрав шрифт и цвет, которые не утомляют глаза, вы можете повысить производительность своей работы. На рис. 3.11 показаны узлы Fonts и Colors в диалоговом окне Options, позволяющие настроить шрифт, размер, цвет и стиль разных элементов экрана. Следует лишь помнить, что эти узлы диалогового окна Options загружаются очень долго, поэтому старайтесь не открыть их непреднамеренно.

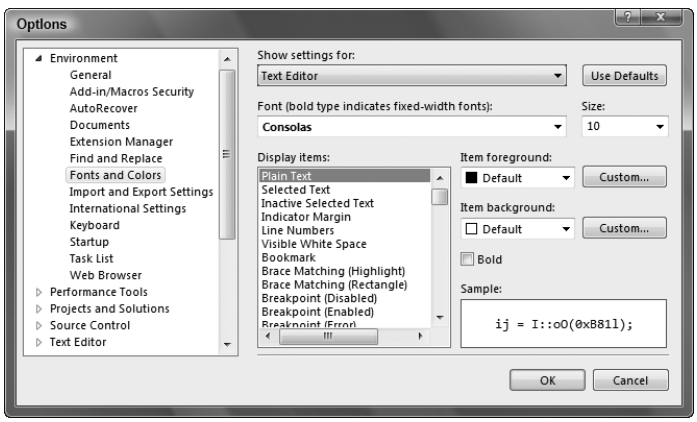


Рис. 3.11

Для того чтобы изменить визуальное представление конкретного текстового элемента в системе Visual Studio 2010, сначала выберите область интегрированной среды разработки, к которой хотите применить новые настройки. Например, на рис. 3.11 был выбран элемент Text Editor. Это предопределило элементы, которые должны были появиться в списке Display Items. Выбрав требуемый элемент в этом списке, настройте шрифт и цвета.



Некоторые элементы этого списка, например Plain Text, используются во многих областях системы Visual Studio 2010, поэтому последствия изменения шрифтов и цветов иногда могут быть непредсказуемыми.

Выбирая шрифт, помните, что пропорциональные шрифты, как правило, не так эффективны для написания программ, как непропорциональные (известные также как моноширинные). Как показано на рис. 3.11, моноширинные шрифты выделены полужирным, что позволяет легко найти их.

Принципы визуального представления

Редактируя текст, система Visual Studio 2010 автоматически раскрашивает элементы кода в зависимости от типа файла. Например, в коде на языке VB ключевые слова выделяются синим цветом, имена переменных и ссылки на классы — черным, а строковые литералы — красным. На рис. 3.12 видно, что вдоль левого края экрана перемещается линия. Она используется для того, чтобы показать границы блоков кода. Щелкнув на знаке "минус", можете уменьшить объем кода метода `btnSayHello_Click` или всей формы `Form1`.

На рис. 3.12–3.15 продемонстрированы разные особенности визуального представления кода. Читатели, знакомые с языком VB.NET, сразу поймут, что на рис. 3.12 недостает конца строки, в которой метод обрабатывает событие Click, связанное с кнопкой btnSayHello. Это объясняется тем, что часть строки уходит за край кодового окна. Для того чтобы увидеть конец этой строки, разработчик должен либо прокрутить окно вправо, либо переместить туда курсор, используя навигационные клавиши. На рис. 3.13 показана процедура сворачивания текста, которая выполняется с помощью диалогового окна Options (узел Text⇨Editor⇨All Languages⇨General).

```
Public Class Form1
    Private Sub btnSayHello_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        MessageBox.Show(Me.txtToSay.Text)
    End Sub
End Class
```

Рис. 3.12

```
Public Class Form1
    Private Sub btnSayHello_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnSayHello.Click
        MessageBox.Show(Me.txtToSay.Text)
    End Sub
End Class
```

Рис. 3.13

```
1 Public Class Form1
2     Private Sub btnSayHello_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
      Handles btnSayHello.Click
3         MessageBox.Show(Me.txtToSay.Text)
4     End Sub
5 End Class
```

Рис. 3.14

К сожалению, выполнение процедуры сворачивания текста может затруднить работу со свернутыми строками. К счастью, в системе Visual Studio 2010 есть команда (непосредственно под флажком, включающим сворачивание текста в диалоговом окне Options), которая может выводить на экран визуальные глифы в конце каждой строки, чтобы выделить свернутые строки, как показано на рис. 3.14. На этом рисунке продемонстрированы еще две возможности визуального представления кода. Во-первых, слева от маркеров блоков кода указаны номера строк. Эту возможность можно подключить с помощью флажка Line Numbers, расположенного под флажками Word Wrap и VisualGlyphs. Другая возможность — это точки, обозначающие пробелы в тексте кода. В отличие от других элементов визуального представления эту возможность можно включить с помощью команды Edit⇨Advanced⇨View White Space, при условии, что область редактирования находится в фокусе.

Полноэкранный режим

Когда пользователь открывает много инструментальных окон и панелей, пространство для редактирования кода быстро уменьшается. По этой причине в системе Visual Studio 2010 предусмотрен полноэкранный режим, запустить который можно с помощью команды View⇨Full Screen. В качестве альтернативы можно нажать комбинацию клавиш <Shift+Alt+Enter>, которая включает и отключает полноэкранный режим. На рис. 3.15 показана верхняя часть окна системы Visual Studio 2010 в полноэкранном режиме, в котором не видны ни инструментальные панели, ни инструментальные окна, причем окно раскрыто максимально широко, так, что даже кнопки Minimize, Restore и Close не видны.

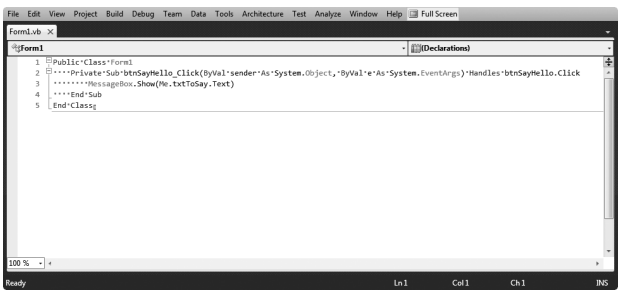


Рис. 3.15



Если вы используете несколько экранов, то полноэкранный режим может оказаться очень полезным. Открепите инструментальные окна и расположите их на втором мониторе. Когда окно редактирования находится в полноэкранном режиме, вы получаете доступ к инструментальным окнам, не переключая режимы. Если вы открепите окно редактирования кода, оно не перейдет в полноэкранный режим.

Отслеживание изменений

Для того чтобы обогатить возможности редактирования, система Visual Studio 2010 предусматривает отслеживание строк, изменившихся в ходе сеанса редактирования. Когда вы открываете файл для редактирования, ни одна из строк не раскрашена. Но ко-

гда вы начнете редактировать текст, то заметите желтую метку возле измененных строк. На рис. 3.16 показано, что строка, содержащая ссылку на объект MessageBox, была модифицирована за время, прошедшее с момента последнего сохранения файла.

```
Public Class Form1
    Private Sub btnSayHello_Click(ByVal sender As System.Object,
                                  ByVal e As System.EventArgs) Handles btnSayHello.Click
        MessageBox.Show("This line has changed but this file hasn't been saved")
    End Sub
End Class
```

Рис. 3.16

После сохранения файла в конце измененных строк появится зеленый маркер. На рис. 3.17 видно, что первая строка, содержащая ссылку на объект MessageBox, была изменена за время, прошедшее с открытия файла, но эти изменения уже записаны на диск. В то же время вторая строка, содержащая ссылку на объект MessageBox, еще не была сохранена.

```
Public Class Form1
    Private Sub btnSayHello_Click(ByVal sender As System.Object,
                                  ByVal e As System.EventArgs) Handles btnSayHello.Click
        MessageBox.Show("This line has changed adn the file has been saved,")
        MessageBox.Show("but this line has changed since then!")
    End Sub
End Class
```

Рис. 3.17



Если вы не считаете полезной возможность отслеживать изменения строк, отключите ее, сбросив флажок Text Editor⇨General⇨Track Change в окне Options.

Другие возможности

Существует еще много возможностей настроить систему Visual Studio, о которых мы не говорили. В оставшейся части главы мы рассмотрим некоторые полезные опции, которые могут повысить вашу производительность.

Сокращенные комбинации клавиш

В системе Visual Studio 2010 одно и то же всегда можно сделать разными способами. Меню, инструментальные панели и разнообразные инструментальные окна предоставляют непосредственный доступ ко многим командам, но, несмотря на огромное количество возможностей, далеко не все из них можно осуществить с помощью графического интерфейса. Вместо этого команды выполняются с помощью комбинаций клавиш (с учетом того, что большинство из них содержится в меню и инструментальных панелях).

Комбинации клавиш варьируются от всем известной <Ctrl+Shift+S> для сохранения всех изменений, до малоизвестной <Ctrl+Alt+E>, открывающей диалоговое окно Exceptions. Как вы могли догадаться, можно задать свои собственные комбинации клавиш и даже изменять существующие. И что еще лучше, можете фильтровать комбинации клавиш, для того чтобы применять их в определенном контексте, т.е. применять одну и ту же комбинацию клавиш для выполнения разных действий в зависимости о того, что вы в данный момент делаете.

На рис. 3.18 показаны узел Keyboard в разделе Environment диалогового окна Options и комбинации клавиш, предусмотренные по умолчанию. Если хотите изменить эти комбинации, выберите их из раскрывающегося списка и щелкните на кнопке Reset.

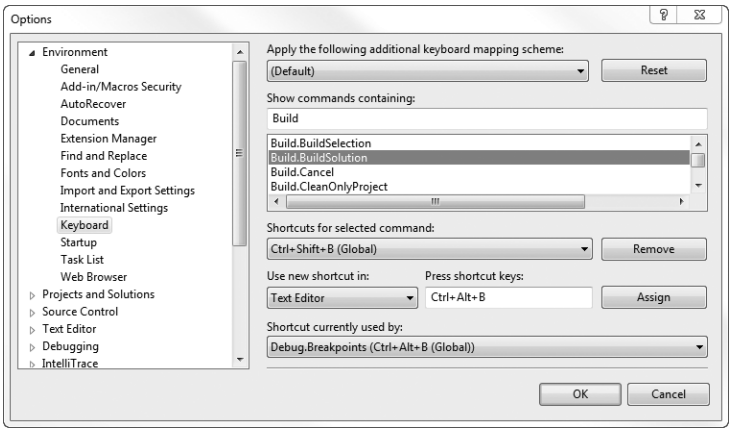


Рис. 3.18



Клавишные схемы хранятся в VSK-файлах каталога `C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE`. Этот формат использовался в версиях, предшествующих Visual Studio 2005. Для того чтобы импортировать клавишные схемы из системы Visual Studio 2005, используйте возможности импортирования, описанные в конце главы. В более ранних версиях просто скопируйте VSK-файл в указанный выше каталог, и тогда вы сможете выбрать соответствующие клавишные схемы в следующий раз, когда откроете окно Options.

Список, показанный на рис. 3.18, *посередине*, содержит все команды, доступные в системе Visual Studio 2010. К сожалению, этот список слишком длинный, а изменить размеры диалогового окна Options невозможно, поэтому просматривать список довольно сложно. Для того чтобы упростить поиск, можете отфильтровать список команд с помощью текстового окна Show Commands Containing. На рис. 3.18 отфильтрованы все команды, содержащие слово *Build*, и в этом списке была выбрана команда `Build.BuildSolution`. Поскольку этой команде уже была назначена комбинация клавиш, следует открыть раскрывающийся список Shortcuts for Selected Command и щелкнуть на кнопке Remove. Существует возможность задавать несколько комбинаций клавиш для одной и той же команды, поэтому раскрывающийся список позволяет вам удалять отдельные комбинации клавиш.



Возможность задавать несколько комбинаций клавиш для одной и той же команды оказывается полезной, если вы хотите сохранить комбинацию, заданную по умолчанию, чтобы другие разработчики чувствовали себя в привычной обстановке, работая с вашими настройками, или если хотите добавить свою собственную комбинацию клавиш.

Остальные элементы этого диалогового окна позволяют присваивать комбинации клавиш выбранной вами команде. Перейдите к текстовому окну Press Shortcut Keys и, как подсказывает его название, нажмите соответствующие клавиши. На рис. 3.18 продемонстрирована ситуация, в которой были нажаты клавиши <Ctrl+Alt+B>. Как показано в нижней части диалогового окна, эта комбинация уже была назначена другой команде. Если вы щелкнете на кнопке Assign, то эта комбинация клавиш будет назначена команде `Build.BuildSolution`.

Для того чтобы ограничить применение комбинаций клавиш определенным контекстом системы Visual Studio 2010, выберите контекст из раскрывающегося списка Use New Shortcut In. Пункт Global означает, что данная комбинация клавиш применена ко всей системе, но если хотите, чтобы она относилась только к окну редактирования, то выберите пункт Text Editor, как показано на рис. 3.18.



В главе 52 показано, как создать макрос, чтобы облегчить выполнение определенных действий. Этим макросам также можно назначить комбинации клавиш.

Проекты и решения

Несколько опций связано с проектами и решениями. Первая из них, вероятно, самая полезная — место хранения ваших проектов, заданное по умолчанию. Для хранения проектов система Visual Studio 2010 использует стандартный путь Documents and Settings (рис. 3.19), но вас этот каталог может не устроить.

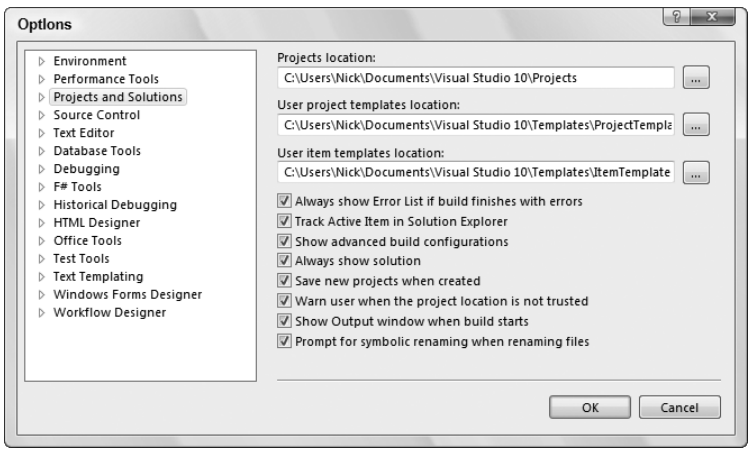


Рис. 3.19

Можете также изменить место для хранения временных файлов. Если ваша организация использует общее сетевое хранилище для корпоративных проектных шаблонов, то можете изменить настройки Visual Studio 2010, заданные по умолчанию, и указать удаленный адрес, а не сетевой диск.

Вы можете настроить большое количество опций, регламентирующих управление проектами и решениями в системе Visual Studio 2010. Одна из наиболее интересных опций —Track Active Item в окне Solution Explorer. Если включить эту опию, то по мере перемещения с элемента на элемент макет окна Solution Explorer будет изменяться,

гарантируя, что текущий элемент будет находиться в фокусе. При этом открываются проекты и каталоги (правда, после этого они уже не закрываются), что в большом решении может оказаться очень неудобным, потому что вам придется постоянно закрывать проекты, по которым вы проходите.

Другая опция, связанная с решениями, не показана на рис. 3.19. Это список внесистемных файлов в окне Solution Explorer. Предположим, вы работаете над решением и должны проверить XML-документ, который не включен в решение. Система Visual Studio 2010 без проблем откроет этот файл, но вам придется повторно открывать его каждый раз, когда вы будете открывать свое решение.

В качестве альтернативы можно выбрать опцию Environment Documents Show Miscellaneous Files in Solution Explorer в диалоговом окне Options, тогда файл будет временно включаться в ваше решение. Каталог для разнообразных файлов, в который будет включен этот файл, показан на рис. 3.20.

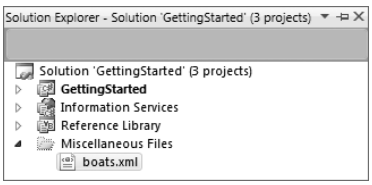


Рис. 3.20



Система Visual Studio 2010 автоматически управляет списком внесистемных файлов, сохраняя только самые последние и подсчитывая количество файлов, определенных в диалоговом окне Options. Вы можете заставить систему Visual Studio включить в этот список до 256 файлов, а затем эти файлы будут исключены из списка в зависимости от времени последнего доступа к ним.

Команды Build и Run

С помощью узлов Projects and Solutions⇨Build и Run, показанных на рис. 3.21, можно настраивать поведение системы Visual Studio 2010 при создании проектов. Первый раздел, заслуживающий внимания, — Before Building. По умолчанию, если выбран пункт Save All Changes, система Visual Studio применит все изменения к решению до его компиляции. Если в процессе сборки возникнет сбой при отладке, то код не пострадает. Можете изменить эту опцию на Prompt to Save All Changes, если не хотите заранее сохранять все изменения по умолчанию, но мы не рекомендуем это делать. Данная настройка заставит систему сообщать вам о не сохраненных модификациях, сделанных в вашем решении, чтобы вы дважды перепроверили изменения до начала компиляции.

Для того чтобы уменьшить время, затрачиваемое на сборку решения, можете увеличить максимальное количество параллельно выполняемых процессов. Система Visual Studio 2010 может выполнять параллельно только независимые проекты, но если у вас много независимых проектов, то вы получите значительный выигрыш во времени.

На рис. 3.21 показано, что к устаревшим проектам будет применяться команда Always Build, а если при сборке решения возникнут ошибки, то они выполняться не будут. Обе эти опции могут увеличить производительность вашего труда, но учтите, что они могут выполнять свою работу, не ставя вас в известность.



Последняя настройка, заслуживающая внимания на рис. 3.21, — MSBuild project build output verbosity. В большинстве случаев система Visual Studio 2010 выдает достаточно подробные сообщения об ошибках, возникших при отладке. Однако в некоторых ситуациях, особенно при создании проектов ASP.NET, пользователю необходимо получать более подробные сообщения, чтобы правильно определить причины ошибки. Новшеством в системе Visual Studio 2010 является возможность управлять уровнем детализации файла регистрации независимо от вывода результатов.

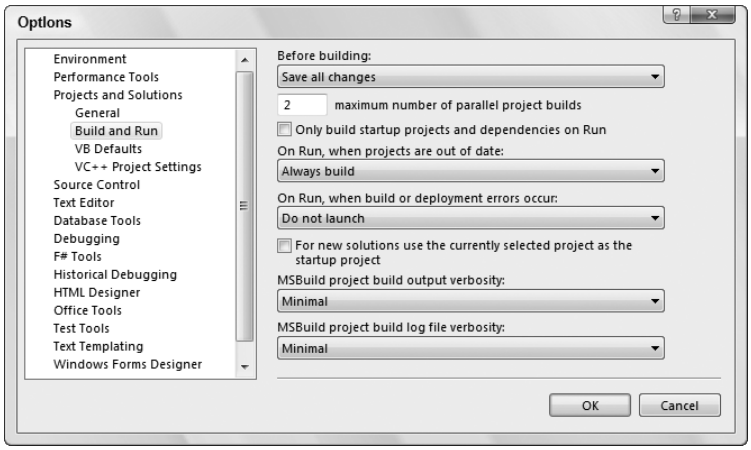


Рис. 3.21

Настройки языка VB

Программисты, работающие на языке VB, имеют возможность задать четыре настройки компилятора для определения конфигурации на уровне проектов или файлов. Вы также можете установить настройки по умолчанию с помощью узла Projects and Solutions⇨VB Defaults в диалоговом окне Options. Предыдущие версии языка VB имели настройку Option Explicit, которая требовала, чтобы все переменные были определены до их первого использования в коде. Когда появилось это требование, многие эксперты рекомендовали, чтобы оно стало постоянным, поскольку это позволяет предотвратить множество проблем, возникающих на этапе выполнения VB-приложений и вызванных неправильным использованием переменных.



Мы настоятельно рекомендуем использовать настройку Option Strict, чтобы гарантировать, что ваш код не выполнит непредвиденное и неявное приведение типов. Если вы не используете настройку Option Strict в сочетании с новыми возможностями языка, то вряд ли сможете достичь максимальной эффективности его использования. Узел Environment⇨Import and Export Settings в диалоговом окне Options позволяет задавать коллективный файл настроек. Он может быть расположен в сетевом хранилище, а система Visual Studio 2010 будет автоматически применять новые настройки, если этот файл будет изменен.

Настройка Option Strict продвигает хорошую практику программирования на один шаг вперед, заставляя разработчиков приводить переменные к правильным типам, а не заставлять компилятор угадывать соответствующий метод приведения типов. Это также позволяет предотвратить проблемы на этапе выполнения программ и повышает производительность труда.

Настройки импорта и экспорта

Точно настроив интегрированную среду разработки по своему вкусу, вы можете сохранить эти настройки на будущее. Для этого можно экспортировать настройки интегрированной среды в файл или даже передать ряду инсталляций системы Visual Studio 2010, чтобы во всех установленных системах были одинаковые настройки.

Для того чтобы экспортировать выбранную конфигурацию, выберите команду Tools⇨Import and Export Settings, чтобы запустить мастер Import and Export Settings Wizard, как показано на рис. 3.22. На первом этапе работы этого мастера следует выбрать настройку Export, а также настройки, которые следует сохранить при выполнении процедуры экспорта.



Рис. 3.22

Как показано на рис. 3.22, можно экспортировать множество сгруппированных настроек. На этом снимке экрана видно, как раскрывается раздел Options, демонстрируя, что следует сохранить настройки Debugging и Projects, а также конфигурации Text Editor и Windows Forms Designer. Маленькие пиктограммы с восклицательным знаком свидетельствуют о том, что некоторые настройки не были предназначены для экспорта по умолчанию, поскольку они содержат информацию, которая может нарушить конфиденциальность информации. В этом разделе вы должны сделать свой выбор вручную, если действительно хотите, чтобы эта информация была сохранена в резервном файле. Выбрав настройки, которые хотите экспортировать, переходите к следующему этапу работы мастера, который может занять несколько минут, в зависимости от того, сколько настроек вы экспортируете.

Импортировать файл настроек очень легко. Для этого используется тот же самый мастер, но теперь на первом этапе вам следует выбрать команду Import. Вместо простой перезаписи текущей конфигурации мастер позволяет вам сначала сохранить резервную копию текущих настроек.

Затем можете выбрать существующий файл конфигурации из списка. Это тот же самый список файлов, из которого вы выбираете настройки при первом запуске системы Visual Studio 2010. Кроме того, можете просмотреть файлы настроек, созданные вами заранее. Выбрав файл настроек, можете импортировать только разделы конфигурации или всю ее целиком.

Мастер по умолчанию исключает несколько разделов, таких как External Tools или Command Aliases, чтобы вы не могли непреднамеренно уничтожить пользовательские настройки. Убедитесь, что вы выбрали эти разделы, если хотите восстановить все настройки.



Если вы просто хотите восстановить одну из конфигураций системы Visual Studio 2010, заданных по умолчанию, выполните команду Reset All Settings на первом этапе работы мастера и не выполняйте весь процесс импорта.

Резюме

В главе изложены лишь основные сведения о выборе полезных настроек, позволяющих вам настроить интерфейс системы Visual Studio по своему вкусу; кроме этого вам доступно много других возможностей. Эти многочисленные настройки позволяют настраивать редактирование кода, добавлять элементы управления в вашу форму и даже выбирать методы при отладке кода. Настройки, доступные в диалоговом окне Options, также позволяют вам управлять процессом создания приложений и даже задавать собственные комбинации клавиш для выполнения команд. В оставшейся части книги мы еще не раз вернемся к диалоговому окну Options для настройки процессов компиляции, отладки и написания макросов.

4

Рабочая среда Visual Studio

В ЭТОЙ ГЛАВЕ...

- Работа с редактором кода
- Исследование основных инструментальных окон системы Visual Studio
- Реорганизация рабочего пространства

До сих пор мы рассказывали о том, как начать работу с системой Visual Studio 2010 и как удобно настроить интегрированную систему разработки. В этой главе мы научим вас эффективно использовать некоторые встроенные команды, комбинации клавиш, а также вспомогательные инструментальные окна. Это поможет вам писать программы и проектировать формы.

Редактор кода

Разработчики проводят значительное время, работая над текстами программ. Следовательно, очень важно уметь правильно скомпоновать код и перемещаться по его тексту. В системе Visual Studio 2010 реализован совершенно новый редактор кода, основанный на платформе визуализации WPF (Windows Presentation Foundation). Он предоставляет разработчикам множество новых возможностей в дополнение к тем, которые были унаследованы от предыдущих версий системы.

Компоновка окна редактора кода

Открывая исходный файл для редактирования, вы работаете с окном редактора кода, показанным на рис. 4.1. Основой этого окна является окно кода, в котором отображается текст программы.

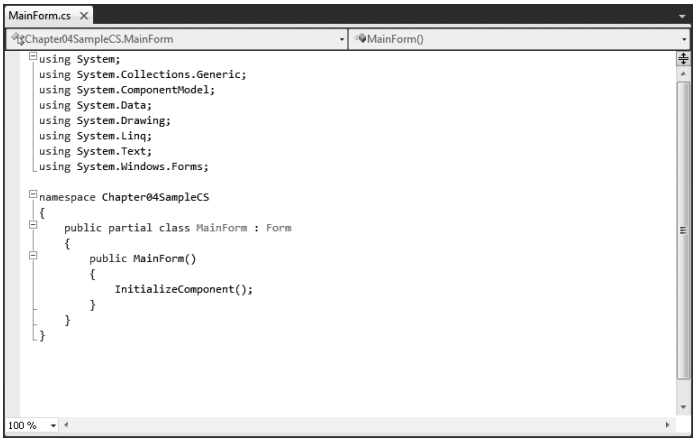


Рис. 4.1

Над окном кода расположены два раскрывающихся списка, которые помогут вам перемещаться по файлу кода. В первом из этих списков перечислены классы, содержащиеся в файле кода, а во втором — члены класса, выбранного из первого списка. Классы и их члены перечислены в алфавитном порядке. Это облегчает поиск метода или определения члена класса в файле.

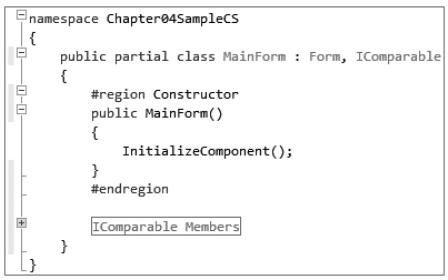


Рис. 4.2

Когда вы редактируете программу в окне редактора кода, строки кода, которые были модифицированы с момента открытия файла, сопровождаются метками в левом поле (рис. 4.2) — желтыми для не сохраненных изменений и зелеными для сохраненных.

Регионы

Класс считается разработанным правильно, если он предназначен для единственной цели и не слишком сложный или длинный. Однако в некоторых ситуациях вы будете вынуждены реализовать так много интерфейсов, что ваш файл кода станет громоздким. Существует множество возможностей, позволяющих справиться с этой задачей, например, разделить код на фрагменты, записанные в разных файлах, или свернуть его фрагменты с помощью регионов, облегчив навигацию по программе.

Использование частичных классов (т.е. в классов, определение которых разбросано по нескольким файлам) позволяет в ходе проектирования записывать код в физически разные файлы, которые в совокупности представляют собой логически единый класс. Преимущество использования отдельных файлов заключается в эффективном группировании всех связанных друг с другом методов, например, методов, реализующих интерфейс. В то же время эта стратегия имеет недостаток: перемещение по коду требует постоянного переключения между его файлами.

В качестве альтернативы можно использовать именованные *регионы кода* (code regions), позволяющие сворачивать разделы кода, которые в данный момент не используются. На рис. 4.2 показаны два региона: Constructor и IComparable. Если щелкнуть на

знаке "минус", расположенном ниже директивы `#region`, то можно сжать регион в одну строку, а если на знаке "плюс" — то снова развернуть его.



Не обязательно разворачивать регион, чтобы увидеть код, который в нем записан. Просто переместите курсор мыши над регионом, и вы увидите код в контекстном окне указателя.

Сворачивание

Кроме регионов, которые необходимо определять вручную, в системе Visual Studio 2010 есть *структуры кода* (outlines), позволяющие автоматически сворачивать методы, комментарии и определения классов. Режим автоматического сворачивания включен по умолчанию. Если он был отключен, то его можно снова включить с помощью команды Edit⇨Outlining⇨Start Automatic Outlining.

На рис. 4.3 показаны четыре сворачиваемых региона. Один из них называется `Constructor`, а остальные три автоматические структуры кода позволяют свернуть класс, XML-комментарии и метод конструктора (который был сжат). Автоматические структуры можно сворачивать и разворачивать так же, как и регионы, которые определяются вручную.

```
public partial class MainForm : Form
{
    #region Constructor
    /// <summary>
    /// Constructs the MainForm form
    /// </summary>
    public MainForm()...
    #endregion
}
}
```

Рис. 4.3

Меню Edit⇨Outlining содержит большое количество команд, помогающих переключать режимы сворачивания, то сжимая весь файл до определений методов и свойств (Edit⇨Outlining⇨Collapse to Definitions), то разворачивая все свернутые фрагменты кода вновь (Edit⇨Outlining⇨Stop Outlining). Другой способ разворачивать и сворачивать регионы основан на использовании комбинации клавиш <Ctrl+M>, <Ctrl+M>. Эта комбинация позволяет переключаться между двумя режимами.



Разработчикам, работающим на языке C# , знаком трюк, основанный на комбинации клавиш <Ctrl+]>, которая позволяет легко перемещаться от начала региона, фрагмента или блока к его началу, и наоборот.

Форматирование кода

По умолчанию система Visual Studio 2010 помогает писать читабельный код, автоматически выравнивая текст. Однако ее можно настроить так, чтобы контролировать внешний вид кода. Общим для всех языков программирования является возможность управлять событиями, которые должны произойти при создании новой строки. На рис. 4.4 показан узел Tabs, расположенный под узлом Text Editor⇨All Languages диалогового окна Options. В этом узле определены значения, заданные по умолчанию для всех языков. Эти значения можно изменить, используя узел Basic⇨Tabs (для языка VB.NET), C#⇨Tabs, или другие узлы, соответствующие другим языкам.

По умолчанию для языков C# и VB.NET включен режим интеллектуального выравнивания, который, помимо прочего, автоматически осуществляет выравнивание при открытии и закрытии вложений. Интеллектуальное выравнивание доступно не для всех языков. В таких ситуациях вместо него используется выравнивание блоков.

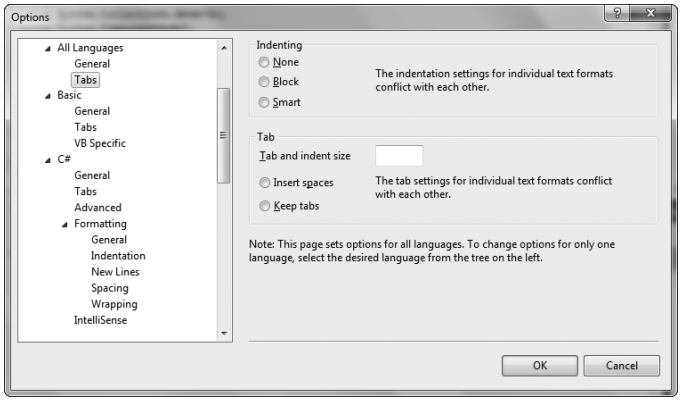


Рис. 4.4



Если вы работаете с маленьким экраном, то, возможно, захотите уменьшить шаги табуляции и размеры отступов, чтобы оптимизировать использование экрана. Задавая шаги табуляции и размеры отступов одинаковыми, вы легко сможете выравнивать свой код с помощью клавиши табуляции.

Интересно, что это диалоговое окно дает разработчикам, работающим на языке C#, много возможностей для управления компоновкой своего кода. В узле VB Specific есть только один флажок Pretty listing (reformatting) of code, который при включении позволяет поддерживать единообразный внешний вид кода и освобождает пользователя от необходимости заботиться о методах выравнивания, свертках, определениях классов и пространствах имен. С другой стороны, разработчики, работающие на языке C#, могут контролировать практически каждый аспект форматирования кода с помощью дополнительных узлов, как показано на рис. 4.4.

Режим Smart Indenting в системе Visual Studio практически всегда прекрасно справляется с автоматическим выравниваем кода по мере его ввода или вставки в окно редактора кода, но иногда встречаются плохо отформатированные фрагменты кода, которые трудно читать. Для того чтобы система Visual Studio переформатировала весь документ и установила позиции фигурных скобок и отступов строк, выберите команду Edit⇨Advanced⇨Format Document или нажмите комбинацию клавиш <Ctrl+K>, <Ctrl+D>. Для того чтобы переформатировать только что выделенный блок кода, выберите команду Edit⇨Advanced⇨Format Selection или нажмите комбинацию клавиш <Ctrl+K>, <Ctrl+F>.

Для того чтобы выровнять целый блок кода на одном уровне, не изменяя каждую строку по отдельности, выберите блок и нажмите клавишу <Tab>. В начало каждой строки будет вставлен символ табуляции. Для того чтобы отменить выравнивание блока по одному уровню, выделите его и нажмите комбинацию клавиш <Shift+Tab>.



Вы могли заметить, что команды Tabify/Untabify Selected Lines в меню Edit⇨Advanced сильно отличаются от команды Format Selection. Эти команды превращают ведущие пробелы в символы табуляции, и наоборот, а не вычисляют заново отступы, как это делает команда Format Selection.

Перемещение вперед и назад

Когда вы перемещаетесь по элементам интерфейса и между ними, система Visual Studio 2010 отслеживает вашу позицию почти так же, как веб-браузер отслеживает сайты, которые вы посетили. Используя команды Navigate Forward и Navigate Backward из меню View, вы сможете легко перемещаться вперед и назад между разными элементами проекта, в которые внесли изменения. Для перемещения назад используется комбинация клавиш <Ctrl+–>, а для перемещения вперед — <Ctrl+Shift+– >.

Дополнительные свойства редактора кода

Редактор кода в системе Visual Studio имеет очень много возможностей, которые трудно исчерпывающим образом описать в этой книге. Однако наиболее полезные из них мы приводим ниже.

Подсветка ссылок

Подсветка ссылок — новое свойство системы Visual Studio 2010. Система выделяет все вхождения имени в области видимости (например, имени метода или свойства), на которое указывает курсор (рис. 4.5). Благодаря этому легче будет найти места, где это имя еще используется в коде. Вы сможете легко перемещаться между этими местами с помощью комбинации клавиш <Ctrl+Shift+Up/Down>.

```
foreach (Account account in assignedAccounts)
{
    account.BeginEdit();
    account.DivisionID = 0;
    account.BranchID = 0;
    account.CostCentreID = 0;
    account.EndEdit();
}
```

Рис. 4.5

Укрупнение кода

С помощью комбинации <Ctrl+Mouse Wheel> можно увеличивать и уменьшать масштаб кода (по существу, увеличивая и уменьшая текст). Это свойство может оказаться особенно полезным, когда необходимо продемонстрировать код группе людей. В левом нижнем углу окна редактора кода есть раскрывающийся список, позволяющий выбрать заранее заданные уровни увеличения текста.

Сворачивание текста

Чтобы включить опцию сворачивания текста в окне редактора кода, выберите команду Tools⇨Options, откройте узел Text Editor, выберите подузел All Languages и установите флажок Word Wrap. Можете также отобразить глиф *стрелки возврата* (return arrow) в том месте, где текст должен быть свернут, установив флажок Show Visual Glyphs for Word Wrap, расположенный ниже флажка Word Wrap.

Чтобы применить эти установки к текущему проекту, выберите команду Edit⇨ Advanced⇨Word Wrap.

Номера строк

Для того чтобы отслеживать положение курсора в файле, полезно пронумеровать строки в окне редактора кода (рис. 4.6). Для того чтобы включить режим нумерации строк, выберите команду меню Tools⇨Options, откройте узел Text Editor, выберите подузел All Languages и установите флажок Numbers.

Разделение окна

Иногда удобно разделить один и тот же файл кода на две части и просматривать их одновременно в разных окнах. Можете разделить активное окно редактора кода на верхнюю и нижнюю части с помощью *линии разбивки* (splitter bar). Эти окна можно прокручивать независимо друг от друга и отображать в них содержимое одного и того же файла в одно и то же время (рис. 4.7).

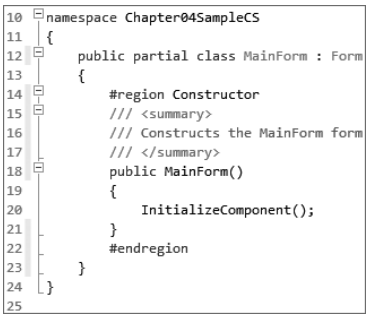


Рис. 4.6

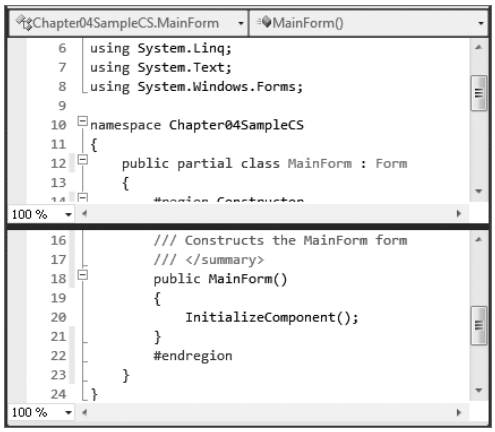


Рис. 4.7

Для того чтобы разделить окно редактора кода, выберите команду Split из меню Window. В качестве альтернативы можно перетащить *метку-манипулятор* (handle), расположенную прямо над панелью вертикальной прокрутки, в позицию, где должна располагаться линия разбивки.

Чтобы настроить размеры каждого окна, достаточно передвинуть линию разбивки вверх или вниз. Удалить линию разбивки просто: дважды щелкните на ней или выберите команду Remove Split из меню Window.

Свободное перемещение окон редактора кода



В системе Visual Studio 2010 появилось новшество, которого долго ожидали программисты, работающие с несколькими дисплеями, — возможность "освобождать" окна редактора кода (и инструментальные окна) и перемещать их за пределы главного окна системы Visual Studio (как показано на рис. 4.8) и даже на другой дисплей. Это позволяет одновременно работать с несколькими окнами редактора кода, расположенными на разных дисплеях. Для того чтобы освободить окно, сначала убедитесь, что оно находится в фокусе, а затем выберите команду Float из окна Window. В качестве альтернативы можно щелкнуть правой кнопкой мыши на заголовке окна и выбрать команду Float из контекстного меню или щелкнуть и перетащить закладку для этого окна (при этом окно освобождается) и перетащить его в то место, где вы хотите его расположить.

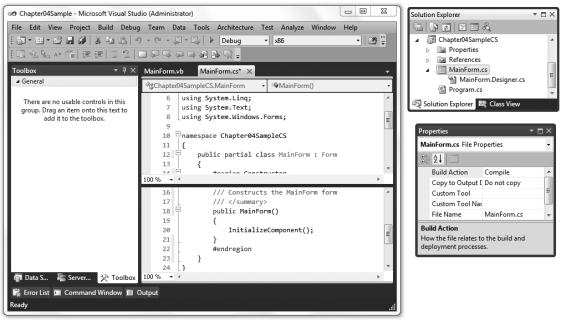


Рис. 4.8

Когда разделение окна редактора кода на две части для одновременного просмотра разных фрагментов одного и того же файла (см. предыдущий раздел) оказывается неудобным, можно вместо этого использовать плавающее окно, разместив его, например, на другом дисплее (если вы работаете с несколькими дисплеями). Для этого можно воспользоваться специальным трюком (поскольку двойной щелчок мышью на файле в окне Solution Explorer просто активизирует существующее окно редактора кода для данного файла) — выбрать команду New Window в меню Window. Эта команда откроет просматриваемый в данный момент файл в новом окне, которое можно сделать свободным и разместить в любом месте.

Создание групп закладок

Если у вас нет нескольких мониторов, то все равно есть возможность открыть одновременно несколько окон редактора кода. Для этого следует создать группы закладок и создать из них мозаику, чтобы отобразить одновременно. Каждая закладка окна редактора кода в этой группе отображается как отдельный элемент мозаики. Таким образом, можно создать несколько групп закладок. Их количество ограничено лишь размерами экрана, на котором они должны размещаться. Элементы мозаики можно размещать как вертикально, так и горизонтально, правда, применить эти два способа одновременно невозможно.

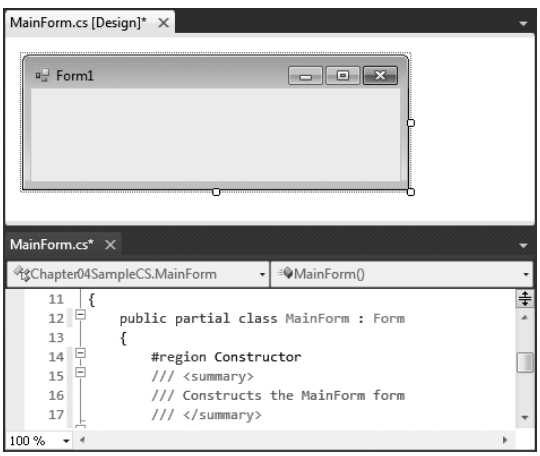


Рис. 4.9

Для того чтобы начать этот процесс, откройте в окне редактора кода несколько закладок. Убедитесь, что закладка окна редактирования кода находится в фокусе, а затем выберите команду Window⇨New Horizontal Tab Group or Window⇨New Vertical Tab Group из открывшегося меню. В результате будет создана новая группа закладок и мозаика для них (рис. 4.9).

В качестве альтернативы можно просто перетащить закладку вниз или за существующую и прикрепить ее в этой позиции, получив тот же самый результат. Можно

также перетаскивать закладки из одной группы закладок в другую, используя команды Window⇨Move to Next Tab Group и Window⇨ Move to Previous Tab Group. Эти команды доступны также в меню, которое открывается после щелчка правой кнопкой мыши на закладке. Для того чтобы восстановить пользовательский интерфейс, имеющий только одну группу закладок, переместите закладки из новой группы (или групп) закладок назад в исходное место, и мозаика исчезнет.

Дополнительные возможности

Для того чтобы достичь действительно высокой производительности труда, полезно знать о многочисленных скрытых возможностях редактора кода, которые позволяют сэкономить массу времени. Рассмотрим некоторые из наиболее полезных команд, которые скрыты в редакторе кода.

"Закомментирование" и "раскомментирование" блока кода

Часто возникает необходимость "закомментировать" блок кода или "раскомментировать" его. При этом не хотелось бы добавлять символы комментариев в начало каждой строки или удалять их оттуда, особенно, если этих строк в блоке много. Разумеется, в языке C# можно свернуть блок кода, разместив его между символами `/*` и `*/`, чтобы "закомментировать", но такой тип комментариев в языке Visual Basic отсутствует. Кроме того, даже в языке C# возникают проблемы, когда требуется "закомментировать" блок, который уже содержит в себе подобные комментарии.

В системе Visual Studio есть простое средство для "закомментирования" и "раскомментирования" блоков. Для этого следует выделить блок, затем выбрать команду Edit⇨ Advanced⇨Comment Selection, чтобы "закомментировать" его, или Edit⇨Advanced⇨ Uncomment Selection, чтобы "раскомментировать" его. Проще всего выбрать эти команды (скорее всего, именно их вы будете чаще всего использовать) с помощью комбинаций клавиш. Нажмите комбинации клавиш <Ctrl+K>, <Ctrl+C>, чтобы "закомментировать" блок кода, и <Ctrl+K>, <Ctrl+U> — чтобы "раскомментировать" его. Еще одним простым средством для доступа к этим командам является инструментальная панель Text Editor.

Выделение блока

```
private readonly string value1 = "";
private readonly string value2 = "";
private readonly string value3 = "";
private readonly string value4 = "";
private readonly string value5 = "";
```

Рис. 4.10

Выделение блока, которое называют также *выделением окна* (box selection), *столбца* (column selection), *прямоугольника* (rectangle) или *вертикального текста* (vertical text selection), как показано на рис. 4.10, подразумевает выделение текстового блока, а не строк (процедура, которая называется *выделением потока* (stream selection)). Для того чтобы отметить текстовый блок, нажмите клавишу <Alt> и не отпускайте ее, пока выделяете текст мышью, или используйте комбинацию клавиш <Shift+Alt+Arrow>. Это свойство может оказаться полезным, когда необходимо удалить часть вертикально выровненного кода (например, префикс в объявлениях переменных).

Многострочное редактирование

Многострочное редактирование — это новое свойство, появившееся в системе Visual Studio 2010 и расширяющее возможность выделения блоков. В предыдущих версиях системы после выделения вертикального текстового блока вы могли только удалять, вырезать или копировать блок. Теперь в системе Visual Studio 2010 после выде-

ления вертикального текстового блока можно в каждой строке набрать текст, который заменит выделенный фрагмент. Это удобно, например, когда требуется заменить уровень доступа к группе переменных с `private` на `protected`.



Можно также вставить текст одновременно в несколько строк, создав блок с нулевой шириной и просто начав печатать.

Кольцевой буфер обмена

Система Visual Studio отслеживает последние двадцать фрагментов текста, которые были скопированы или вырезаны в буфер обмена. Для того чтобы вставить текст, который был ранее скопирован в буфер обмена, но был перезаписан, вместо обычной комбинации <Ctrl+V>, предназначенной для вставки, можно использовать комбинацию <Ctrl+Shift+V>. Нажав клавишу <V> и удерживая клавиши <Ctrl+Shift>, можно выполнять циклы просмотра элементов буфера обмена.

Полноэкранный режим

Можете максимизировать представление кода, подлежащего редактированию, выбрав команду View⇨Full Screen или нажав комбинацию клавиш <Shift+Alt+Enter>, чтобы увеличить до максимума окно редактора кода, скрыв другие инструментальные окна и панели. Для того чтобы вернуться в обычный режим, снова нажмите комбинацию клавиш <Shift+Alt+Enter> или щелкните на кнопке переключения Full Screen, добавленной в конце панели меню.

Переход к определению

Для того чтобы быстро перейти к определению класса, метода или члена, расположенного ниже курсора, щелкните правой кнопкой мыши и выберите команду Go To Definition или нажмите клавишу <F12>.

Поиск всех ссылок

В системе Visual Studio 2010 есть возможность найти точку вызова метода или ссылки на свойство, щелкнув правой кнопкой мыши на его определении и выбрав команду Find All References в контекстном меню или поместив курсор на определение метода и нажав комбинацию клавиш <Shift+F12>. Эти команды открывают инструментальное окно Find Symbol Results, содержащее позиции, в которых существуют ссылки на указанный метод или свойство. Теперь можно дважды щелкнуть на ссылке в окне результатов, чтобы переместиться на искомую позицию в окне редактирования кода.



Это свойство несколько устарело после появления нового окна Call Hierarchy, которое будет рассмотрено ниже. Однако оно по-прежнему позволяет быстро находить используемый метод без перемещения по окну Call Hierarchy.

Окно Command

По мере углубления ваших знаний о системе Visual Studio 2010 вы будете проводить все меньше времени в поисках функциональных свойств и чаще будете исполь-

зовать комбинации клавиш, чтобы перемещаться в интегрированной среде разработки и выполнять определенные действия. Одно из инструментальных окон, которое открывается чаще других, — окно Command (View⇨Other Windows⇨Command Window (<Ctrl+Alt+A>)). Когда открыто это окно, можете выполнить любую существующую команду или макрос, а также любой дополнительный макрос, который можете написать и сохранить. На рис. 4.11 показано, как с помощью технологии IntelliSense открыть список команд, которые можно выбрать в окне Command. Этот список будет содержать все макросы, определенные в текущем решении.



Рис. 4.11

Полный список команд системы Visual Studio можно открыть с помощью узла Environment⇨Keyboard, содержащегося в диалоговом окне Options (Tools⇨Options). Все команды имеют одинаковый синтаксис, основанный на области интегрированной среды разработки, для которой они созданы. Например, можете открыть окно отладки (Debug⇨Windows⇨Output), набрав команду `Debug.Output` в окне Command.

Все команды разделяются на три группы. Одни являются просто методами быстрого открытия инструментальных или диалоговых окон (которые открываются, если до этого были закрыты). Например, команда `File.NewFile` открывает новое диалоговое окно. Другие команды запрашивают информацию о текущем решении или отладчике. Команда `Debug.ListThreads` позволяет получить список текущих потоков, а команда `Debug.Threads` открывает инструментальное окно Threads. К третьему типу принадлежат команды, которые выполняют действия без открытия диалогового окна. К ним относится большинство макросов и команд, принимающих аргументы (полный список этих команд, включая аргументы, которые они принимают, можно найти в документации MSDN). Между этими группами существуют пересечения: например, команда `Edit.Find` может выполняться как с аргументами, так и без них. Если эта команда выполняется без аргументов, то открывается диалоговое окно Find and Replace. В качестве альтернативы следующая команда найдет все вхождения строки `MyVariable` в текущем документе (`/d`) и поставит маркер на границе окна кода напротив соответствующих строк (`/m`).

```
> Edit.Find MyVariable /m /d
```

Несмотря на то что в окне Command применяется технология IntelliSense, набор часто используемых команд может показаться утомительным. В системе Visual Studio 2010 существует возможность присваивать некоторым командам альтернативные имена, или псевдонимы. Например, для присваивания псевдонима можно использовать команду `alias`, а для поиска команды, использованной ранее, команду `e?`.

```
> alias e? Edit.Find MyVariable /m /d
```

Определив этот псевдоним, можно легко выполнить эту команду в любом месте интегрированной среды разработки: нажмите комбинацию клавиш <Ctrl+Alt+A>, чтобы установить фокус на окно Command, затем наберите команду `e?`, чтобы выполнить команду “найти и пометить”.

В начале работы с системой Visual Studio 2010 будет импортировано большое количество псевдонимов, заданных по умолчанию и относящихся к настройкам среды. Перечислить их можно с помощью команды `alias` без аргументов. В качестве альтернативы при желании найти, какие именно команды относятся к конкретному псевдониму, можете выполнить команду `alias`, задав в качестве аргумента соответствующий псевдоним. Например, выполнив запрос к предыдущем псевдониму `e?`, будет получено следующее.

```
>alias e?
alias e? Edit.Find SumVals /m /doc
```

К псевдонимам можно применять два дополнительных переключателя. Переключатель `/delete` вместе с псевдонимом удаляет ранее определенный псевдоним. Если вы хотите удалить все ранее определенные вами псевдонимы и вернуться к предыдущим именам, можно использовать переключатель `/reset`.

Окно Immediate

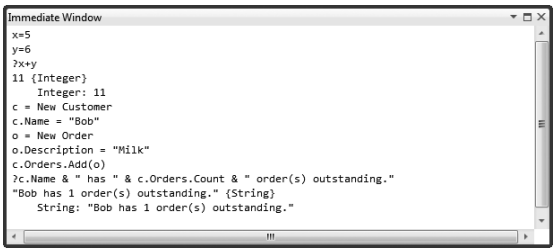


Рис. 4.12

Довольно часто при создании кода или отладке программы возникает необходимость вычислить простое выражение либо для того, чтобы проверить часть кода, либо для того, чтобы вспомнить, как работает тот или иной его фрагмент. В таких ситуациях удобно использовать окно Immediate, позволяющее вычислять инструкции сразу после их ввода. На рис. 4.12 показано много инструкций — от простого присваивания и операций вывода на печать до создания объекта и сложных манипуляций.



Если вы пишете программы на языке Visual Basic, то не сможете явно объявить переменную в окне Immediate (например, `Dim x as Integer`), но это можно сделать неявно с помощью оператора присваивания. В примере, продемонстрированном на рис. 4.12, создается новый объект класса `Customer`, который присваивается переменной *c*, а затем используется в ряде операций. Если вы пишете программы на языке C#, то новые переменные в окне Immediate должны быть явно объявлены еще до присваивания им какого-либо значения.

Окно Immediate частично поддерживает технологию IntelliSense, и вы можете использовать стрелки для просмотра истории выполненных команд. Значения переменных можно вывести на экран с помощью инструкции `Debug.Print`. В качестве альтернативы можете использовать их псевдоним `?`. В языке C# ни тот ни другой прием не нужен; для того чтобы вывести на экран значение переменной, наберите имя переменной в окне и нажмите клавишу <Enter>.

Если вы выполняете какую-нибудь команду в окне Immediate, работая в режиме проектирования, то система Visual Studio, прежде чем выполнить команду, сначала соберет решение. Если ваше решение не скомпилируется, то выражение не будет вычислено, пока ошибка компиляции не будет устранена. Если код исполняемой коман-

ды содержит активную контрольную точку, то в этом месте выполнение команды будет остановлено. Это может быть полезным, если вы работаете с конкретным методом, который требуется проверить без запуска всего приложения.

Открыть окно Immediate можно с помощью команды Debug⇨Windows⇨Immediate или комбинации клавиш <Ctrl+Alt+I>, но, если вы работаете, поочередно переключаясь между окнами Command и Immediate, то, вероятно, захотите использовать заранее установленные псевдонимы `cmd` и `immed` соответственно.



Обратите внимание на то, что для того чтобы выполнить команду в окне Immediate, необходимо добавить префикс > (например, для того, чтобы перейти к окну Command, надо выполнить команду `>cmd`); в противном случае система Visual Studio попытается вычислить команду. Кроме того, вы должны быть уверены, что язык, используемый в окне Immediate, совпадает с языком, на котором создается активный проект. Примеры, приведенные на рис. 4.12, работают, только если текущий активный проект создан на языке Visual Basic.

Окно Class View

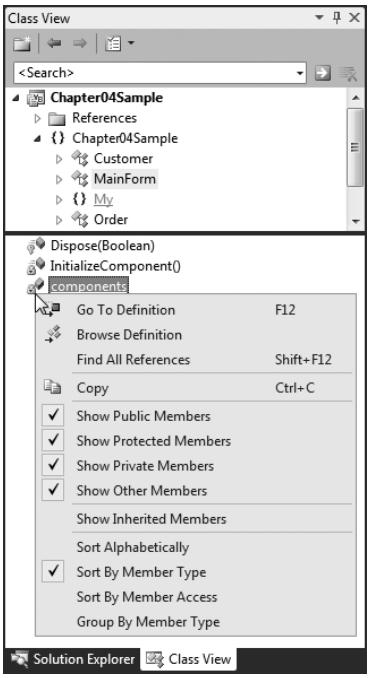


Рис. 4.13

Хотя окно Solution Explorer, вероятно, является наиболее полезным инструментальным окном для навигации по вашему решению, с его помощью иногда трудно найти конкретные классы и методы. Инструментальное окно Class View обеспечивает вам альтернативный взгляд на решение, перечисляя пространства имен, классы и методы, чтобы вам было легко их найти. На рис. 4.13 показано простое приложение для операционной системы Windows, содержащее одну форму (MainForm), выделенную в иерархии классов. Обратите внимание на то, что в иерархии есть два узла с именем Chapter04Sample. Первый — это имя проекта (а не сборки, как вы могли подумать), второй — пространство имен, которому принадлежит форма MainForm. Раскрыв узел References, можно увидеть список сборок, на которые ссылается данный проект. Изучая глубже в каждую из этих сборок, вы увидите список пространств имен, за которым следует класс, содержащийся в данной сборке.

В нижней части рис. 4.13 показан класс `Form1`. Открыв контекстное меню щелчком правой мыши, вы можете либо отфильтровать этот список по степени доступности, упорядочить и сгруппировать его, а также использовать для перехода к выбранному члену класса. Например, выбрав команду Go To Definition для метода `InitializeComponent()`, вы откроете файл `Form1.Designer.vb`.

Окно Class View полезно для навигации по сгенерированным членам, которые обычно записаны в файле, скрытом по умолчанию в окне Solution Explorer (как файл

проектировщика в предыдущем примере). Он также облегчает перемещение по классам, добавленным в существующий файл, — в результате в одном и том же файле могут оказаться повторяющиеся классы, что не желательно. Поскольку имя файла не совпадает с именем класса, довольно трудно просматривать класс с помощью окна Solution Explorer; следовательно, окно Class View представляет собой хорошую альтернативу.

Окно Error List

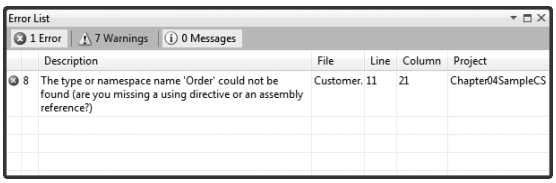


Рис. 4.14

Как показано на рис. 4.14, окно Error List содержит ошибки компиляции, предупреждения и сообщения о вашем решении. Можете открыть окно Error List, выбрав команду View⇨Error List или нажав комбинацию клавиш <Ctrl+\>, <Ctrl+E>. Ошибки появляются в списке, когда вы редактируете код и компилируете проект. Если дважды щелкнуть на ошибке в списке ошибок, то откроется файл и вы увидите строку кода, содержащую данную ошибку.

Элементы списка, которые вы хотите увидеть, можно отфильтровать по типам ошибок (Errors, Warnings и/или Messages), переключая кнопки, расположенные вверху списка.

Окно Object Browser

Другой способ просмотра классов, образующих ваше приложение, основан на использовании инструментального окна Object Browser. В отличие от большинства остальных инструментальных окон, которые по умолчанию прикреплены к краю основного окна системы Visual Studio 2010, окно Object Browser появляется в области редактирования. Для того чтобы увидеть окно Object Browser, выберите команду View⇨Object Browser или нажмите комбинацию клавиш <Ctrl+Alt+J> (или клавишу <F2> в зависимости от установок вашей клавиатуры). Как видно на рис. 4.15, в верхней части окна Object Browser находится раскрывающийся список, позволяющий выбрать область видимости объекта. В него входит набор заранее установленных значений, таких как All Components, .NET Framework 4.0, My Solution или Custom Component Set.

В данном случае выбран элемент My Solution и введена строка поиска `Sample`. Содержание главного окна состоит из всех пространств имен, классов и членов, имена которых соответствуют строке поиска.

В правом верхнем углу рис. 4.15 показан список членов выбранного класса (`MainForm`), а в окне, расположенном ниже, — полное определение класса, содержащее его базовый класс и информацию о пространстве имен. Одним из элементов списка Browse (см. рис. 4.15) является элемент Custom Component Set. Для того чтобы определить, какие сборки будут включены в это множество, щелкните на эллипсисе, расположенном следом за раскрывающимся списком, или выберите элемент Edit Custom Component Set в самом списке. В результате откроется диалоговое окно, аналогичное приведенному на рис. 4.16.

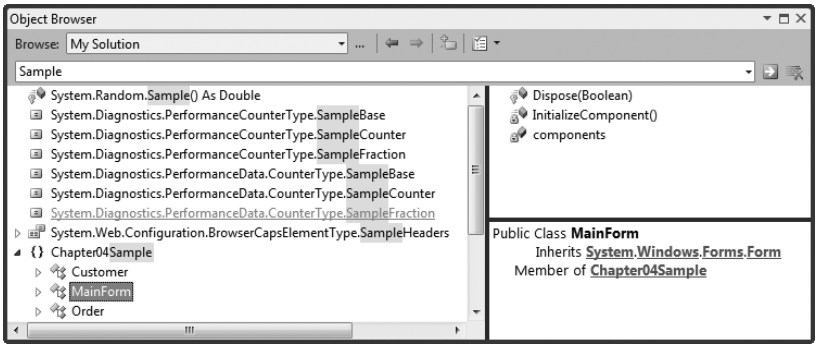


Рис. 4.15

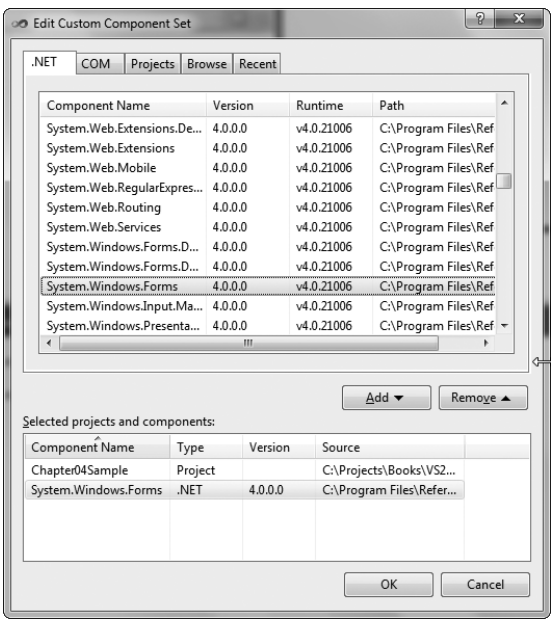


Рис. 4.16

Выбрав элементы в верхней части окна и щелкнув на кнопке Add, вы вставите сборку в множество компонентов. Аналогично, выбрав элемент в разделе, расположенном ниже, и щелкнув на кнопке Remove, вы удалите сборку из множества компонентов. По окончании настройки множества компонентов сохраняются между сессиями системы Visual Studio.

Окно Code Definition

Просматривая код, вы обнаружите вызов метода, который хотели бы увидеть, не покидая своей текущей позиции в окне редактора кода. В этом случае удобно использовать окно Code Definition, предназначенное для демонстрации исходного кода метода, на вызов которого установлен курсор (рис. 4.17). Это окно открывается после

выбора команды View⇨Other Windows⇨Code Definition (<Ctrl+\>, <Ctrl+D>). Окно Code Definition представляет собой просто еще одно окно редактора кода, в котором доступны многие из его команд (например, вставка контрольных точек, просмотр иерархии вызовов и т.д.), но в режиме "только для чтения". Для того чтобы отредактировать код этого метода, щелкните правой кнопкой мыши в любом месте окна Code Definition и выберите команду Edit Definition. В ответ на эту команду в окне редактора кода откроется исходный файл с этим методом, а фокус будет перемещен на определение этого метода.

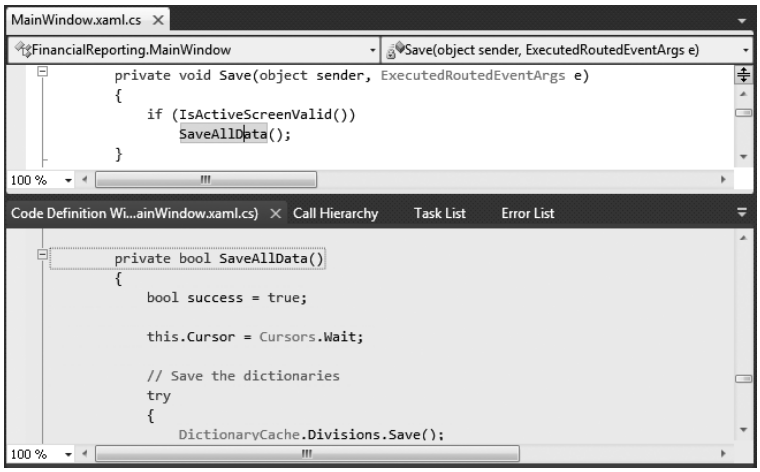


Рис. 4.17



Вы можете также использовать окно Code Definition наряду с окнами Class View и Object Browser для просмотра кода выбранного члена класса.

Окно Call Hierarchy

Новое свойство системы Visual Studio 2010 — окно Call Hierarchy — отображает все вызовы, поступающие в метод и исходящие из него (а также свойства или конструкторы, но при условии, что на них ссылаются как на методы). Это позволяет увидеть, где используется метод и как эти вызовы влияют на другие методы. Кроме того, благодаря этому окну можно легко отследить выполняемые ветки и поток управления.

Для того чтобы увидеть иерархию вызовов метода, выберите определение этого метода в окне редактора кода и выберите команду View Call Hierarchy в контекстном меню, появляющемся после щелчка правой кнопкой мыши. Это действие добавляет данный метод в дерево, отображаемое в окне Call Hierarchy вместе с двумя подузлами — Calls To (MethodName) и Calls From (MethodName), как показано на рис. 4.18.

Раскрыв узел Calls To (MethodName), можно увидеть список всех методов, которые вызывают данный метод. Раскрыв узел Calls From (MethodName), можно увидеть все остальные методы, которые вызываются из указанного метода. Окно Call Hierarchy позволяет проследить результаты анализа вызовов и построить иерархию потока выполнения программы — т.е. увидеть, какие коды вызывают данный метод, какие методы вызывает он, и т.д.

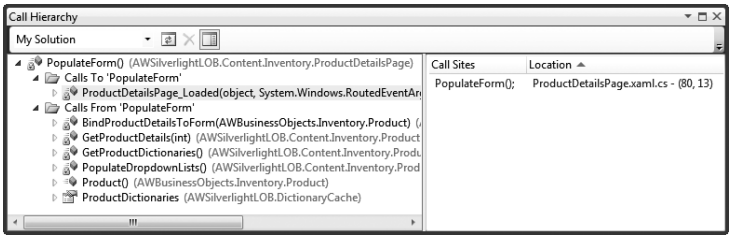


Рис. 4.18

Дважды щелкнув на методе, можно перейти на определение метода в окне редактора кода.



Кроме того, иерархию вызовов можно увидеть в окне Class View или Object Browser, если щелкнуть правой кнопкой на методе и выбрать команду View Call Hierarchy в контекстном меню.

Несмотря на то что окно Call Hierarchy может быть как плавающим, так и закрепленным, оно работает не так, как окно Code Definition. Перемещение по разным методам в окне редактора кода не приводит к автоматическому выводу на экран иерархии классов для метода, на который указывает курсор. Для того чтобы это произошло, необходимо явно потребовать, чтобы на экран была выведена иерархия классов для этого метода и показано, в какой точке она должна отобразиться в окне Call Hierarchy. Окно Call Hierarchy может отображать иерархию классов для нескольких методов сразу, и каждый раз в окно будет добавляться новая иерархия вызовов для нового метода, а не обновляться существующая иерархия классов. Если вы не хотите, чтобы отображалась иерархия классов для конкретного метода, удалите ее, выбрав окно и нажав клавишу <Delete> (или на красном крестике на инструментальной панели).



Это окно может оказаться очень удобным при работе с неизвестным проектом или при переделке проекта.

Окно Document Outline

Редактировать HTML-файлы с помощью визуального проектировщика или средств просмотра кода иногда сложно, особенно, если приходится работать с большим количеством вложенных элементов. Когда система Visual Studio .NET впервые появилась на рынке, средство, известное как *разметка документов* (document outlining), экономило как минимум один рабочий день. На самом деле это свойство оказалось настолько удобным для работы с HTML-файлами, что его перекупили для работы с формами и элементами управления, не связанными с веб. В данном разделе кратко описано окно Document Outline и показано, насколько эффективными бывают манипуляции с HTML-документами, формами и средствами управления.

Разметка HTML

Основной целью окна Document Outline является представление HTML-страниц в виде, допускающем простую навигацию по ним, так, чтобы можно было без труда

находить разные HTML-элементы и контейнеры, в которых они находятся. Поскольку получить правильные HTML-макеты довольно сложно, особенно, если страница содержит много компонентов .NET, которые можно включить на страницу ASP.NET, окно Document Outline оказывается удобным способом для того, чтобы найти правильную позицию конкретного компонента.

На рис. 4.19 показана типичная HTML-страница. Без окна Document Outline выбирать маленькие или невидимые элементы в проектировщике было бы довольно трудно. Окно Document Outline (View⇨Other Windows⇨Document Outline), показанное на рис. 4.19, *слева*, позволяет легко выбирать элементы в иерархии, чтобы определить, в каком месте страницы они расположены, и установить их свойства.

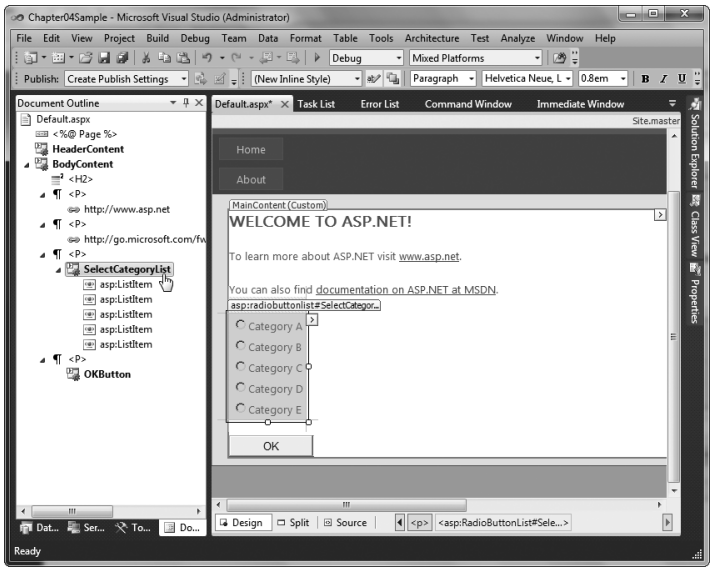


Рис. 4.19

Система Visual Studio анализирует содержимое текущего активного файла и отражает его в виде дерева, содержащего каждый элемент иерархии страницы. В этом дереве отображается значение `Name` или `ID` каждого элемента (если они были присвоены), а неименованные элементы перечисляются с помощью их HTML-дескрипторов.

Когда вы выбираете какой-нибудь элемент в окне Document Outline, окно Design обновляется, позволяя выбрать этот компонент и его наследников. На рис. 4.19 показан выбор элемента `SelectCategoryList` в списке дескрипторов элементов управления RadioButtonList, расположенном в окне Document Outline. Соответствующий элемент управления выделен в окне Design. Это позволяет увидеть, в каком месте страницы расположен данный элемент. Соответственно, выбирая элемент управления или компонент в окне Design, вы выбираете дескриптор в иерархии страницы, показанной в окне Document Outline (впрочем, для того, чтобы окно Document Outline было обновлено правильно, вам придется снова установить фокус на него).

Разметка элементов управления

Окно Document Outline существует в системе Visual Studio с первой версии .NET для HTML-файлов, но редко применялось для файлов других типов. После появления версии Visual Studio 2003 была разработана надстройка, обеспечивающая аналогичные возможности для элементов управления Windows Forms.

Этот инструмент стал настолько популярным, что компания Microsoft внедрила его функциональные возможности в инструментальное окно Document Outline, и теперь аналогичным образом можно просматривать элементы управления Windows Forms.

На рис. 4.20 показана типичная сложная форма со многими панелями, создающая структуру и обеспечивающая средства управления для визуальных элементов. Каждый компонент в окне Document Outline представлен своим именем и типом. Как только компонент выбран в окне Document Outline, соответствующий визуальный элемент отображается на макете Design view.

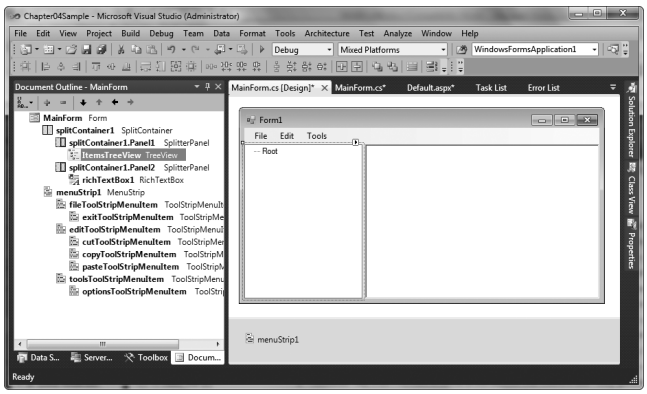


Рис. 4.20

Это значит, что если элементом управления является меню (как на рис. 4.20), то система Visual Studio автоматически откроет меню и выберет команду, готовую для редактирования. Легко представить, что это невероятно полезно для навигации по вашим макетам форм и часто позволяет быстро найти затерянные элементы.

Окно Document Outline обладает бо́льшими функциональными возможностями, когда оно используется в режиме Control Outline, а не как простое средство навигации. Щелкнув правой кнопкой мыши на элементе, откройте небольшое контекстное меню команд, которые можно быстро выполнить. Наиболее очевидным является доступ к окну Properties. Одним из наиболее рутинных занятий является переименование компонентов после того, как они уже были добавлены в форму. Можете выбирать их по очереди и устанавливать свойство Name в окне Properties, но, используя окно Document Outline, можно просто выбрать команду Rename в контекстном меню, и система Visual Studio автоматически переименует компоненты в проектном коде, тем самым обновив свойство Name без необходимости прокручивать список Properties.

Сложный дизайн формы иногда может привести к неожиданным результатам. Это часто происходит, когда компонент размещен в неправильном или неподходящем контейнере элементов управления. В этой ситуации можно переместить компонент в правильный контейнер. Разумеется, сначала необходимо обнаружить неполадку, даже если проблема еще не проявилась.

Окно Document Outline поможет в решении обеих этих задач. Во-первых, используя иерархическое представление, вы можете легко найти любой компонент и его базовые контейнерные элементы. В примере, приведенном на рис. 4.20, показано, что элемент управления TreeView принадлежит панели Panel1, которая в свою очередь принадлежит контейнеру SplitContainer, который в свою очередь принадлежит объекту ContentPanel. Тем самым можно легко идентифицировать ситуацию, в которой элемент размещен на макете некорректно.

Если возникает необходимость переместить компонент, бывает трудно сохранить макет в правильном порядке. В окне Document Outline эта задача решается просто. Достаточно перетащить и отпустить элемент управления в правильную позицию в иерархии. Например, после перемещения элемента управления TreeView на панель Panel2 он одновременно получает доступ как к панели Panel2, так и к элементу управления ListView.

Пользователь также имеет возможность вырезать, копировать и вставлять отдельные элементы или целые наборы контейнеров и их содержимое с помощью контекстного меню, которое открывается после щелчка правой кнопкой мыши. Функция "копировать–вставить" является особенной полезной, поскольку вы можете дублировать части макета в разные места формы, не пользуясь методом проб и ошибок при выборе правильных элементов на закладке Design view, или переупорядочивать их для дублирования в коде, записанном в файле `Designer.vb`.



Вырезав элемент, не забудьте немедленно вставить его в нужную позицию.

Реорганизация инструментальных окон

Интегрированная среда разработки Visual Studio имеет гибкие возможности для позиционирования инструментальных окон, чтобы обеспечивать максимальное удобство в работе и полностью использовать экран дисплея. Инструментальные окна можно закрепить, сделать их плавающими или минимизировать их, отображая лишь корешки на крае экрана.

При перемещении инструментального окна на экране появляется ряд указателей, помогающих установить окно в закрепленное положение. Для того чтобы закрепить окно, переместите его в позицию одного из указателей. Если вы перетащите окно так, что оно будет закрывать лишь часть указателя, то будет выделена часть интегрированной среды разработки, в которой окно будет закреплено, если вы его отпустите (рис. 4.21).

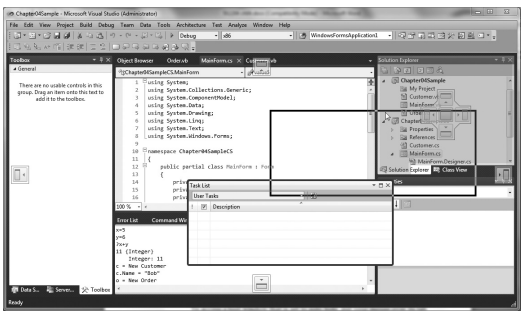


Рис. 4.21

Для того чтобы открепить закрепленное инструментальное окно, щелкните на нем и перетащите в новую позицию (при этом необходимо следить, чтобы не отпустить его на один из появившихся указателей). Нажав клавишу <Ctrl> в ходе перемещения окна, можно предотвратить появление указателей и нежелательную привязку к ним окна. Если окно закреплено и стало частью группы корешков (т.е. группы окон, занимающих одно и то же пространство и допускающих перемещение по ним с помощью щелчков на корешках), то для его перемещения достаточно щелкнуть и перетащить корешок. А если щелкнуть и перетащить заголовок инструментального окна, то это приведет к перемещению всей группы корешков.

Для того чтобы открыть инструментальное окно, которые было автоматически скрыто, поместите курсор мыши на его корешке и перетащите его. Для того чтобы автоматически скрыть инструментальное окно, щелкните на кнопке в заголовке окна и щелкните снова, находясь в режиме автоматического сокрытия окна, чтобы вернуть окно в закрепленное положение.



После вывода инструментального окна из закрепленного положения и его перемещения в другое место (например, на другой монитор) дважды щелкните на заголовке, удерживая нажатой клавишу <Ctrl>, чтобы вернуть окно в предыдущее закрепленное положение.

Резюме

В этой главе мы рассмотрели множество инструментальных окон, помогающих не только писать код, но и макетировать и испытывать его. Эффективное использование этих окон резко сокращает объем времени, необходимый для того, чтобы запустить ваше приложение и проверить написанный код. Это в свою очередь повышает общую производительность труда и исключает потери времени из-за ожидания в ходе выполнения приложения.

5

Команды Find, Replace и Help

В ЭТОЙ ГЛАВЕ...

- Использование инструментов поиска и замены в системе Visual Studio
- Навигация по справочной системе

Для того чтобы разработка была продуктивной, необходимо перемещаться по базе программ и быстро находить требуемый код. Система Visual Studio 2010 предоставляет не одну, а множество функций поиска, предназначенных для выполнения конкретных задач. В начале этой главы обсуждается каждая из этих функций, а также описываются ситуации, в которых они могут понадобиться.

Система Visual Studio 2010 — это весьма сложная среда для разработки программ, охватывающая множество языков программирования и основанная на широком использовании библиотек и компонентов. Практически невозможно знать все тонкости работы интегрированной среды разработки, владеть всеми языками программирования и даже полностью освоить технологию .NET Framework. По мере развития технологии .NET Framework и системы Visual Studio удерживать в памяти всю информацию о новшествах становилось все труднее; более того, на самом деле разработчикам чаще требуется знать только часть этой информации. Разумеется, у пользователей периодически будет возникать необходимость получить более полную информацию по конкретному вопросу. Для того чтобы помочь им в этих ситуациях, система Visual Studio 2010 предлагает обширную документацию в виде библиотеки MSDN Library Visual Studio 2010 Edition. Позже в главе мы обсудим методы исследования этой документации в связи с разработкой проектов в системе Visual Studio 2010.

Окно Quick Find/Replace

Простейшим инструментом поиска в системе Visual Studio 2010 является диалоговое окно Quick Find. Функциональные возможности поиска и замены в системе Visual Studio 2010 разделены между двумя командами, имеющими общие диалоговые окна и свойства, Quick Find и Quick Replace, которые выполняют поиск информации, которую необходимо быстро найти в открытом документе или текущем проекте. Эти два инструмента имеют ограниченные возможности для фильтрации и расширенного поиска, как будет показано далее, даже существующие свойства обеспечивают работу мощной поисковой системы, намного превосходящую практические потребности большинства пользователей.



Этот инструмент поиска лучше всего настроен на поиск и замену текста (в отличие от поиска и замены символа).

Команда Quick Find

Термин Quick Find в системе Visual Studio 2010 используется как название инструмента, имеющего самые главные функциональные возможности для поиска. По умолчанию он находит отдельное слово или фразу в текущем документе, но даже команда Quick Find имеет дополнительные возможности для расширенного поиска за пределами активного модуля и даже для использования шаблонных символов и регулярных выражений в запросе.

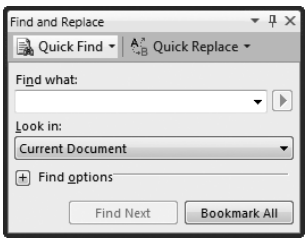


Рис. 5.1

Для того чтобы выполнить команду Find, нажмите стандартную комбинацию клавиш <Ctrl+F> или выберите команду Edit⇨Find and Replace⇨Quick Find. В ответ система Visual Studio откроет основное диалоговое окно Find and Replace, в котором фокус по умолчанию находится на команде Quick Find (рис. 5.1).

Наберите поисковый запрос в поле ввода Find what или выберите один из предыдущих запросов, щелкнув на треугольнике, направленном вниз, и прокрутив список запросов. По умолчанию область поиска ограничена текущим документом или редактируемым окном, если только пользователь не выделил несколько строк текста сразу (в этом случае по умолчанию областью поиска является выделенный фрагмент). Раскрывающийся список Look in предоставляет пользователю дополнительные возможности, учитывающие контекст самого поиска, включая элементы Selection, Current Block, Current Document, Current Project, Entire Solution и All Open Documents.

Команды поиска и замены непрерывно просматривают выделенную область в поисках заданных терминов, останавливая выполнение только тогда, когда процесс поиска вернулся в исходную точку. Когда система Visual Studio находит искомый термин, она выделяет совпадение и прокручивает окно редактора кода, чтобы пользователь увидел этот фрагмент. Если совпадение уже является видимым, то система Visual Studio не выполняет прокрутку окна редактора кода, а просто выделяет новое совпадение. Если же прокрутка окна не нужна, она пытается разместить листинг кода так, чтобы совпадение оказалось посередине окна редактора кода.



Выбрав команду Quick Find в первый раз, можете больше не открывать диалоговое окно, а просто нажимайте клавишу <F3>, если требуется выполнить тот же самый поисковый запрос.



Рис. 5.2

В инструментальном окне Standard есть поле ввода Quick Find (рис. 5.2), на самом деле предназначенное для решения нескольких задач. Для того чтобы установить фокус на это окно, нажмите комбинацию клавиш <Ctrl+/>. После этого введите фразу поиска и нажмите клавишу <Enter>, чтобы найти следующее совпадение в текущем открытом файле. Если перед поисковым запросом вставить префикс >, система Visual Studio 2010 попытается выполнить команду так, будто она была введена в окне Command (см. главу 4).

Команда Quick Replace

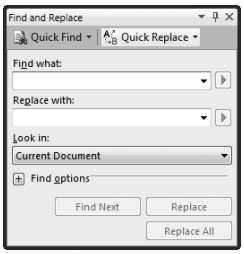


Рис. 5.3

Выполнение команды Quick Replace похоже на выполнение команды Quick Find. Между этими командами можно переключаться, щелкая на соответствующих кнопках, расположенных в верхней части экрана. Если хотите перейти непосредственно к окну Quick Replace, нажмите комбинацию клавиш <Ctrl+H> или выберите команды Edit Find и Replace Quick Replace. Функциональные возможности окна Quick Replace (рис. 5.3) аналогичны функциональным возможностям окна Quick Find, но во втором окне есть дополнительное поле, в котором можно указать текст, используемый для замены.

Поле Replace With аналогично полю Find What — в нем можно либо набрать новую строку для замены, либо выбрать одну из ранее набранных строк в раскрывающемся списке.



Для того чтобы удалить повторяющиеся значения, просто выполните замену, ничего не указывая в поле ввода Replace With. Эта возможность позволяет находить все вхождения заменяемых значений и решать, следует ли их удалить.

Группа Find Options

Иногда необходимо отфильтровать результаты поиска, используя разные критерии. Именно в этой ситуации можно воспользоваться возможностями поиска. Во-первых, для того чтобы вывести на экран группу вариантов поиска (доступную во всех командах поиска и замены), следует щелкнуть на пиктограмме Find Options. В открывшемся диалоговом окне, содержащем набор флажков и раскрывающийся список, можно выбирать варианты поиска, как показано на рис. 5.4.

Эти варианты позволяют уточнить вид поиска: чувствительный к регистру (Match Case) или полное совпадение (Match Whole Word). Можно также изменить направление поиска (Search Up) и указать, что вы собираетесь выполнять сложный поиск, используя шаблонные символы или регулярные выражения.

Шаблонные символы

Шаблонными (wildcards) называют простые текстовые символы, которые могут заменять собой один или несколько символов. Они знакомы многим пользователям системы Windows. На рис.5.5 продемонстрирована работа инструмента Expression Builder в ситуации, когда пользователь установил флажок Use и выбрал пункт Wildcards в раскрывающемся списке. Несмотря на то что в шаблонном поиске можно использовать дополнительные символы, наиболее часто используются символы ? для отдельного символа и * — для группы неизвестных символов или переменных.

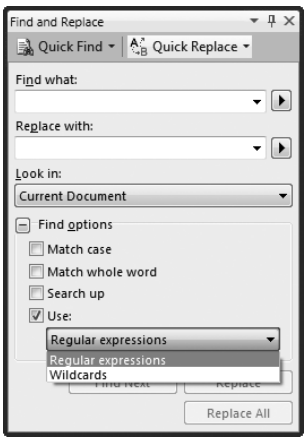


Рис. 5.4

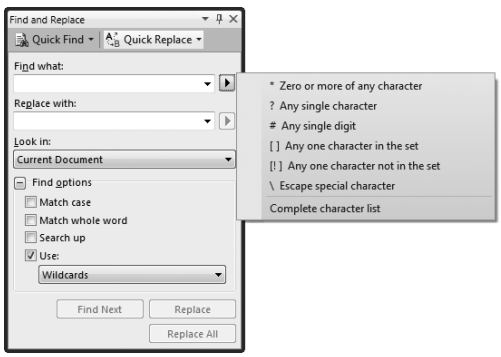


Рис. 5.5

Регулярные выражения

Регулярные выражения выводят поиск на совершенно новый уровень, позволяя выполнять сложные сравнения текстовых фрагментов с помощью инструмента RegEx, встроенного в систему Visual Studio 2010. Несмотря на то что в этой книге мы не планируем очень подробно описывать регулярные выражения, они заслуживают упоминания как важная функциональная возможность диалогового Find and Replace, которую можно эффективно использовать.

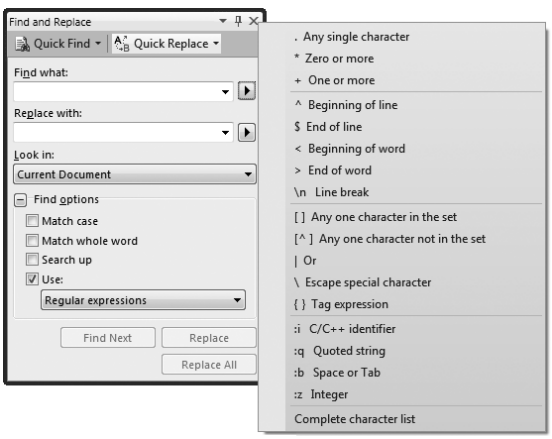


Рис. 5.6

На рис. 5.6 еще раз показан инструмент Expression Builder, на этот раз для создания регулярного выражения с помощью установки флажка Use и соответствующего раскрывающегося списка. На этом рисунке показано, как легко можно создать регулярные выражения с помощью меню, содержащего наиболее часто используемые фразы и символы шаблонных выражений вместе с их описанием.

Ниже приведен пример обратного присваивания, демонстрирующий эффективное использование регулярных выражений. Допустим, у нас есть код

VB

```
Description = product.Description
Quantity = product.Quantity
SellPrice = product.SellPrice
```

C#

```
Description = product.Description;
Quantity = product.Quantity;
SellPrice = product.SellPrice;
```

и мы хотим изменить направление присваивания.

VB

```
product.Description = Description
product.Quantity = Quantity
product.SellPrice = SellPrice
```

C#

```
product.Description = Description;
product.Quantity = Quantity;
product.SellPrice = SellPrice;
```

Для этого очень удобно выполнить команду Quick Replace с регулярными выражениями, а не изменять каждую строку кода вручную. Выберите в списке регулярные выражения и введите следующий "текст".

VB

```
{ < .*} = {.*}
```

C#

```
{ < .*} = {.*};
```

а затем "текст", который служит заменой.

VB

```
\2 = \1
```

C#

```
\2 = \1;
```

Коротко говоря, вы ищете две группы (определенные фигурными скобками), разделенные знаком равенства. Первая группа должна содержать в качестве первого символа слова символ <, а затем — любые символы (`.*`). Вторая группа должна содержать любые символы вплоть до символа конца строки в примере на языке VB или двоеточия в примере на языке C#. Затем, выполняя замену, вы просто вставляете символы из первой группы, знак равенства (окруженный пробелами с обеих сторон) и символы из первой группы (за которым в языке C# следует двоеточие). Если вы не знакомы с регулярными выражениями, придется потратить определенное время на обучение, но быстрота и легкость этого инструмента компенсируют ваши усилия.



Обратите внимание на то, что регулярные выражения в инструменте Quick Find имеют немного иной синтаксис по сравнению со стандартными регулярными выражениями, которые используются в технологии .NET Framework.

Команды Find and Replace

Настроить функциональные возможности поиска и замены можно с помощью их собственных флажков, содержащихся в диалоговом окне Tools⇨Options. Флажки Find and Replace в группе Environment позволяют включать и отключать информационные сообщения и предупреждения, а также указывать, следует ли автоматически заполнять поле ввода Find what выделенным фрагментом текста из окна редактирования. Кроме того, в этой группе есть также флажок, позволяющий скрывать диалоговое окно Find после выполнения команды Quick Find или Quick Replace, что может оказаться удобным, если вы обычно ищете только первое совпадение.

Команды Find/Replace in Files

Команды Find in Files и Replace in Files расширяют область поиска за пределы текущего решения на целые каталоги и структуры папок и даже позволяют выполнять массовые замены любых совпадений, соответствующих заданным критериям и фильтрам. Кроме того, пользователю доступны дополнительные флажки, позволяющие размещать результаты поиска в одном или двух инструментальных окнах, чтобы по ним было легко перемещаться.



Это средство поиска лучше всего работает, когда требуется найти простой текст в файле, который не обязательно является частью вашего решения.

Команда Find in Files

Команда Find in Files действительно является мощным инструментом поисковой машины, встроенной в систему Visual Studio. Она не ограничивает поиск файлами, входящими в текущее решение, а дает возможность выполнять поиск по заданному критерию, просматривая целые каталоги (включая их подкаталоги).

Диалоговое окно Find in Files, показанное на рис. 5.5, можно открыть, выбрав команду Edit⇨Find and Replace⇨Find in Files. В качестве альтернативы, если вы уже открыли диалоговое окно Quick Find, можно переключиться в режим Find in Files, щелкнув на небольшом треугольнике, расположенном сразу после имени команды Quick Find, и выбрав в открывшемся меню команду Find in Files. Для того чтобы открыть это диалоговое окно, можно также использовать комбинацию клавиш <Ctrl+Shift+F>.

Большинство флажков в окне Quick Find при этом останутся доступными, включая шаблонные символы и регулярные выражения, но вместо поиска в проекте или решении пользователь теперь может самостоятельно указывать в поле Look In другую область поиска. Пользователь может либо задать область поиска, набрав соответствующий текст, либо щелкнуть на эллипсисе и открыть диалоговое окно Choose Search Folders, показанное на рис. 5.8.

Пользователь имеет возможность перемещаться по всей файловой системе, включая сетевые диски, и добавлять каталоги, которые должны входить в область поиска. Это позволяет добавлять огромные иерархии файлов в один поисковый запрос. Для того чтобы указать каталоги, которые следует просмотреть, можно использовать список Available Folders, расположенный слева от списка Selected Folders. Чтобы добавить выбранные каталоги в список Selected Folders, достаточно щелкнуть на стрелке, направленной вправо

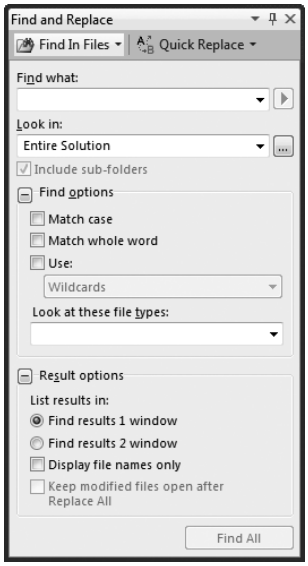


Рис. 5.7

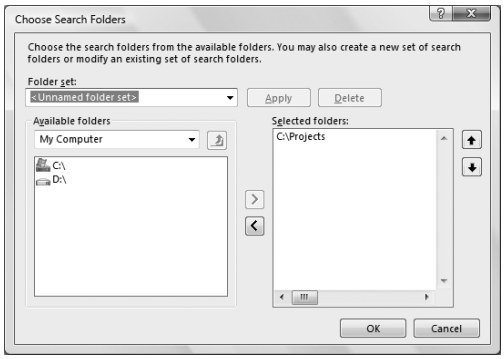


Рис. 5.8

В этом списке порядок поиска можно уточнять с помощью стрелок, направленных вверх и вниз. Добавив каталог в список Selected Folders, щелкните на кнопке OK, чтобы в поле Look In отобразился список путей, разделенных точкой с запятой. Для того чтобы сохранить этот список каталогов для будущего использования, введите имя в раскрывающийся список Folder Set и щелкните на кнопке Apply.



Процесс сохранения каталогов для поиска мало интуитивно понятен, но если интерпретировать кнопку Apply как некую усовершенствованную разновидность кнопки Save, то это диалоговое окно станет более-менее понятным.

Find Dialog Options

Флажки в диалоговом окне Find in Files аналогичны флажкам в диалоговом окне Quick Find. Поскольку поиск выполняется по файлам, которые не обязательно открыты в интегрированной среде разработки и могут даже не быть исходными файлами программ, флажок Search Up больше не используется. Кроме того, для того чтобы сузить поле поиска и задать тип просматриваемых файлов, предусмотрен дополнительный фильтр.

Раскрывающийся список Look at these file types содержит несколько наборов расширений, каждый из которых связан с определенным языком программирования. Это облегчает поиск в кодах, написанных на языках Visual Basic, C#, J# и др.

Пользователь может также задать свое собственное расширение, если он работает с каким-нибудь языком, который не поддерживается компанией Microsoft, или хочет использовать команду Find in Files для целей, не связанных с разработкой программы. Это позволяет ограничить область поиска лишь файлами, имеющими соответствующие расширения.

Кроме флажков, регламентирующих процесс поиска, в системе существуют средства настройки представления результатов поиска. Выполняя поиск, пользователь может выбрать одно из двух окон для вывода результатов. Полный список результатов поиска может быть довольно длинным, но если пользователя интересуют только имена файлов, в которых найдены совпадения, то он может установить флажок Display Filenames Only, и тогда в окне результатов каждому файлу будет соответствовать только одна строка.

Окна для вывода результатов

Результаты выполнения команды Find in Files отображаются в одном из двух окон. Они открываются как инструментальные окна, прикрепленные к нижнему краю рабочего пространства интегрированной среды разработки. Для каждой строки, содержащей критерий поиска, окно результатов содержит полную строку информации, содержащую имя файла и путь к нему, а также номер строки, содержащей совпадение и саму строку текста, так что пользователь сразу видит контекст (рис. 5.9).

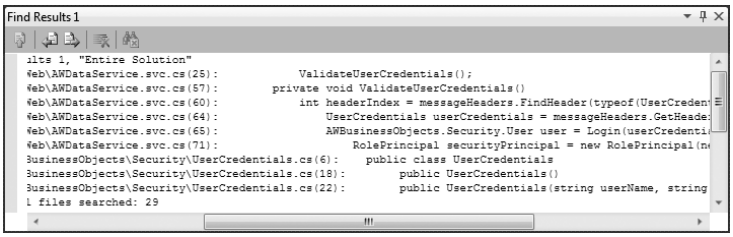


Рис. 5.9

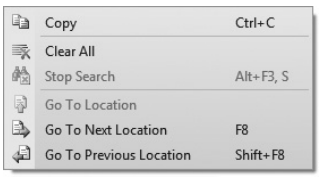


Рис. 5.10

На верхней кромке каждого окна результатов расположена инструментальная панель (рис. 5.10, *слева*), обеспечивающая навигацию по самим результатам. Эти команды можно также выбрать в контекстном меню, как показано на рис. 5.10, *справа*. Достаточно дважды щелкнуть на конкретном совпадении, чтобы перейти на соответствующую строку кода.

Команда Replace in Files

Несмотря на очевидную полезность команды, которая выполняет поиск в большом количестве файлов и обнаруживает множество совпадений, соответствующих заданному критерию, команда Replace in Files еще полезнее. Для того чтобы выбрать эту команду, можно нажать комбинацию клавиш <Ctrl+Shift+H> или щелкнуть на треугольнике, расположенном рядом с заголовком окна Quick Replace. Команда Replace in Files делает то же самое, что и команда Find in Files, и имеет точно такие же возможности.

Ее основное отличие заключается в том, что она выполняет не только поиск, но и замену текста в файлах. При массовых заменах эта команда запрашивает подтверждение, что часто оказывается очень удобно. Для того чтобы сохранить возможность исправить ошибку после замены, следует установить флажок Keep Modified Files Open After Replace All (рис. 5.11, *снизу*).

Обратите внимание на то, что эти возможности доступны, только если выполняется команда Replace All. При выполнении команды Replace система Visual Studio откроет файл, содержащий следующее совпадение, и оставит его открытым в любом случае.

Независимо от того, установлен флажок Keep Modified Files Open After Replace All или нет, после выполнения команды Replace All система Visual Studio сообщит, сколько изменений было внесено. Если вы не хотите видеть это сообщение, закройте это диалоговое окно, чтобы оно не открывалось при выводе результатов следующего поиска.



Важно помнить: если не установить флажок Keep Modified Files Open After Replace All и выполнить массовую замену в большом количестве файлов, то они мгновенно изменятся и сделать откат будет невозможно. Думайте, что делаете!

Команда Find Symbol

Команда Find Symbol позволяет находить класс, метод, свойство или другие типы символов. В то время как команда Quick Find, по существу, выполняет поиск простого текста в указанной области (текущий документ, текущий проект и т.д.), команда Find Symbol ограничена поиском только символов.

Диалоговое окно Find Symbol можно открыть с помощью комбинации клавиш <Alt+F12> или команды Edit⇨Find and Replace⇨Find Symbol. В качестве альтернативы можно переключить обычное диалоговое окно Find and Replace в режим Find Symbol, щелкнув на треугольнике, расположенном следом за именем Quick Find или Find in Files.

Диалоговое окно Find Symbol (рис. 5.12) имеет немного другие флажки по сравнению с остальными диалоговыми окнами поиска. В нем область поиска ограничена не текущим документом или решением, как в окне Quick Find, или файловой системой, как в окне Find in Files. Команда Find Symbol может выполнять поиск в решении, списке всех компонентов и даже во всем пакете .NET Framework. Кроме того, в область поиска можно включить любые ссылки, добавленные в решение. Для того чтобы составить свое собственное множество компонентов, среди которых следует выполнить поиск, щелкните на эллипсисе рядом с полем Look in, а затем просмотрите и выберите компоненты .NET и COM, зарегистрированные в системе, или просмотрите файлы и проекты.



Возможности команды *Find* также упрощены. Поиск можно выполнять только в целых словах, подстроках (по умолчанию) или префиксах.

После того как пользователь щелкнет на кнопке Find All, система скомпилирует список результатов поиска и выведет их в специальном инструментальном окне Find Symbol Results. По умолчанию это окно располагается там же, где и окно Find Results, — внизу рабочего пространства интегрированной системы разработки. В нем выводятся все результаты со всеми ссылками на конкретный объект или компонент. Это чрезвычайно удобно, если необходимо определить, где и как используется конкретный объект и в каком месте проекта есть ссылки на него.

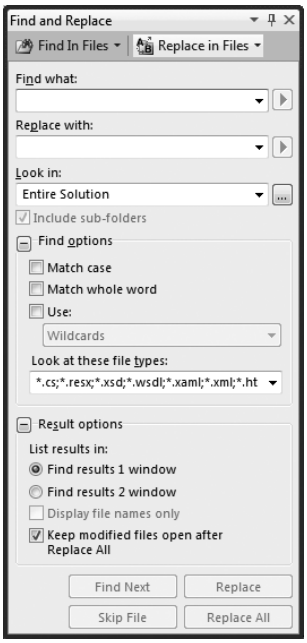


Рис. 5.11

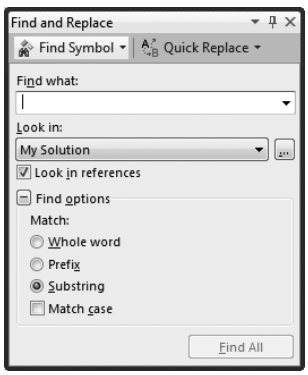


Рис. 5.12



Этот поисковый инструмент лучше всего работает в ситуациях, когда необходимо найти все экземпляры символа и создать список всех совпадений в выбранной области поиска, чтобы можно было легко просматривать большое количество результатов. Ограничив предмет поиска только символами, пользователь проигнорирует посторонний текст, например комментарии, код внутри методов и т.д. Этот поиск не ограничен только кодом, но может быть распространен на весь пакет .NET Framework и набор ссылок.

Команда Navigate To

Команда Navigate To — новый мощный инструмент поиска в системе Visual Studio 2010, обеспечивающий альтернативу стандартным функциям поиска символов. Как и команда Find Symbol, команда Navigate To ограничена лишь поиском символов; для того чтобы оценить ее полезность, следует рассмотреть большое количество отличий между нею и командой Find Symbol.

В противоположность команде Find Symbol команда Navigate To выводит результаты по мере того, как пользователь вводит поисковый запрос. Чем больше текста запроса набрано, тем уже становится множество результатов. Дважды щелкнув на одном из результатов, пользователь может закрыть диалоговое окно и перейти к этому результату.

Одно из наиболее важных свойств команды Navigate To заключается в том, как именно она выполняет поиск. Допустим, пользователь ищет метод класса `ProductSummary`. В этом инструменте поиска пробелы, по существу, играют роль оператора AND, поэтому

в ответ на запрос `prod sum` (т.е. на поиск имен `prod` и `sum` в одном и том же модуле) команда возвращает класс `ProductSummary` точно так же, как в ответ на запрос `sum prod`.

Другое уникальное свойство команды Navigate To заключается в том, что она выполняет поиск, используя "верблюжий стиль". Для того чтобы найти класс `ProductSummary`, можно набрать в запросе `PS` (прописные буквы из имени), чтобы получить искомый метод в качестве результата (рис. 5.13) — очень мощное свойство, доступное только в этом инструменте поиска.

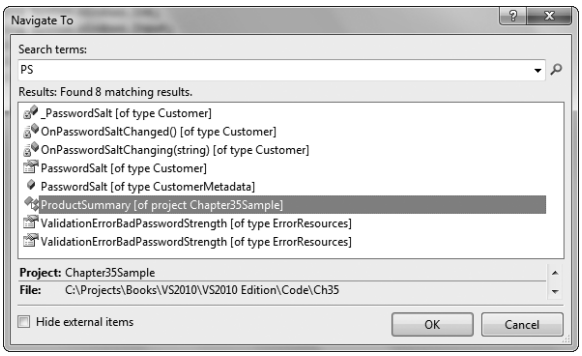


Рис. 5.13

Если весь текст поискового запроса введен в виде строчных букв, то совпадение не будет зависеть от регистра. Однако, если запрос содержит прописные буквы, то поиск становится чувствительным к регистру. Для того чтобы открыть окно Navigate To, можно нажать комбинацию клавиш <Ctrl+,> (запятая).



Этот инструмент поиска работает лучше всего в ситуациях, когда необходимо найти единственный экземпляр символа и перейти к нему. Преимуществами является "живая" реакция на ввод текста и учет "верблюжьего стиля".

Команда Incremental Search

Для того чтобы найти что-то в текущем окне кода, не открывая диалогового окна, можно использовать команду Incremental Search, которая выполняется в ответ на выбор команды Edit⇨Advanced⇨Incremental Search или нажатие комбинации клавиш <Ctrl+I>. Эта команда находит следующее совпадение на основе поискового запроса. Для того чтобы поиск выполнялся вверх по документу, а не вниз, следует нажать комбинацию клавиш <Ctrl+Shift+I>.

Сразу после открытия окна Incremental Search пользователь может начинать ввод искомого текста. Указатель мыши превратится в значок с изображением бинокля и стрелки, направленной вниз. По мере ввода каждого символа редактор будет переходить к следующему совпадению по отношению к введенному тексту. Например, после ввода буквы `f` отобразится первое слово, содержащее эту букву, например *offer*. Если пользователь наберет букву `o`, то курсор переместится на первое слово, содержащее сочетание букв *fo*, например *form* и т.д.

Если весь искомый текст вводится строчными буквами, то совпадение не будет зависеть от регистра. Однако если поисковый запрос содержит хотя бы одну прописную букву, то поиск становится чувствительным к регистру.



Этот инструмент поиска наиболее эффективен в тех же ситуациях, что и команда Quick Find, но поиск на этот раз проводится только в текущем файле и не имеет дополнительных возможностей. Если не учитывать дополнительные возможности команды Quick Find, то эти команды полностью совпадают. Но если пользователь хочет найти результат, внести изменение, а затем найти следующий результат, то лучше использовать команду Quick Find, поскольку иначе, найдя следующий результат, он должен будет заново ввести поисковый запрос для команды Incremental Search.

Получение справки

Разработчик имеет в своем распоряжении широкий спектр технологий. Они не только очень быстро развиваются, но и вынуждают программистов постоянно овладевать новыми знаниями. Поскольку все обо всем знать невозможно, разработчик должен непрерывно учиться. Часто знание того, как найти информацию о той или иной технологии, является таким же важным, как и способность ее освоить. К счастью, существует множество информационных источников, содержащих информацию об этих технологиях. Включение в интегрированные среды разработки технологии автоматического дополнения IntelliSense, произошедшее около десяти лет назад, стало одним из важнейших обстоятельств, облегчивших работу разработчиков, но этот инструмент не может полностью заменить полноценную справочную систему, содержащую подробную информацию. Чтобы решить эту задачу, в среду Visual Studio включена справочная система для разработчиков.

Проще всего получить справку в среде Visual Studio 2010, использовав тот же метод, что и в любом другом Windows-приложении, — нажав универсальную клавишу <F1>. Среда Visual Studio 2010 имеет свою фирменную справочную систему, использующую технологию Microsoft Help 3. Вместо использования специальной “оболочки” и предоставления пользователю средств навигации по ней справочная система теперь открывает окно браузера. Для доступа к более сложным возможностям справочной системы, например возможностям поиска (при работе в автономном режиме), существует специальное приложение “слушатель” (listener), выполняющееся в области системных уведомлений и обслуживающее запросы пользователей. Подчеркнем, что адрес в адресной строке браузера ссылается на локальный веб-сервер или на компьютер пользователя. Как в интерактивном, так и в автономном режиме справочная система и выглядит, и работает почти одинаково, но в этой главе мы обратим особое внимание на автономный режим.



Иногда справочная система возвращает сообщение Service Unavailable. Возможная причина этой ошибки заключается в том, что “слушатель” справочной системы больше не выполняется в области системных уведомлений. Просто откройте справочную систему, находясь в среде Visual Studio, и “слушатель” будет автоматически запущен вновь.

Справочная система в среде Visual Studio является контекстно-зависимой. Это означает, что если курсор в настоящий момент расположен в определении класса в проекте и пользователь нажал клавишу <F1>, то в окне справки моментально откроется мини-учебник о текущей инструкции и способах ее использования (рис. 5.14).

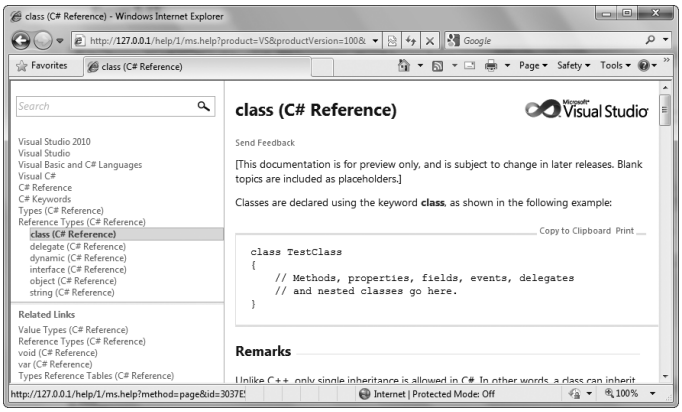


Рис. 5.14

Это невероятно полезное свойство, поскольку в этом случае вы быстрее получите необходимую информацию, чем если просто нажмете клавишу <F1> и станете искать нужное описание, листая справочную систему.

Однако в некоторых ситуациях следует перейти сразу к содержанию справочной системы. Среда Visual Studio 2010 позволяет сделать это с помощью команды Visual Studio Documentation меню Help (рис. 5.15).

Кроме использования справочной системы, пользователь также может зайти на форумы MSDN и сообщить о проблеме.

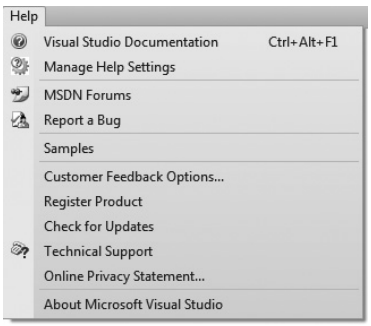


Рис. 5.15

Навигация и поиск в справочной системе

Навигация по справочной системе очень напоминает перемещение по веб. В левой части окна браузера содержатся ссылки на страницы, принадлежащие тому же разделу справочной системы, что и страница, просматриваемая в данный момент, а также несколько ссылок, связанных с текущей страницей.

В левом верхнем углу окна браузера имеется окно для ввода поискового запроса, который вводится точно так же, как и в любой поисковой системе, например Google или Bing. Поиск является полнотекстовым и распространяется на все страницы справочной системы, причем запрос не обязательно должен содержаться в заголовках страниц. В итоге пользователь получит результаты, аналогичные результатам, которые он мог бы получить с помощью любой поисковой машины. В окне выводится по одной строке о каждом результате, чтобы можно было найти требуемый. Чтобы прочитать нужную статью, следует щелкнуть на соответствующей строке.

Конфигурирование справочной системы

Когда пользователь впервые обращается к справочной системе, целесообразно настроить ее под свои нужды. Для этого можно выбрать команду Help⇨Manage Help Settings, чтобы открыть диалоговое окно Help Library Manager, показанное на рис. 5.16



Рис. 5.16

Первая команда, Choose online or local help, открывает другое окно в диалоговом окне, позволяющее выбрать интерактивный или автономный режим работы. Если пользователь выбирает интерактивный режим, то нажатие клавиши <F1> или открытие справки из меню Help автоматически приведет его на соответствующую страницу в документации MSDN, расположенной в Интернете (с учетом текущего контекста системы Visual Studio). Выбирая автономный режим, пользователь попадет на соответствующую страницу в документации, установленной локально (при условии, что она действительно была развернута на данном компьютере).

Преимущество интерактивного режима над автономным заключается в том, что он всегда предоставляет обновленную информацию и не требует затрат памяти на жестком диске пользователя (при условии, что он не разворачивал справочную систему локально). Недостатком является необходимость постоянно поддерживать активное соединение с Интернетом и временами более медленная работа (в зависимости от ширины полосы пропускания), чем в автономном режиме. По существу, это компромисс, на который пользователь должен пойти самостоятельно.

Команда Check for updates online выполняет проверку обновлений в каждом разделе инсталлированной документации. На экран будут выведены инсталлированные разделы документации и проверены обновления для каждого из них. После завершения проверки в нижней части экрана система покажет примерный объем загружаемой информации. Эта величина может оказаться довольно значительной, поэтому продолжать работу целесообразно, только если полоса пропускания достаточна для загрузки такого объема информации.

Команда Find content online позволяет загрузить и добавить в библиотеку дополнительные разделы документации. Соответствующее диалоговое окно получает из Интернета список доступных разделов документации и их размеры. Пользователь может щелкнуть на кнопке Add, расположенной после каждого раздела, который может быть загружен. В этот момент также надо оценить, насколько велики файлы для загрузки.

Команда Find content on disk позволяет инсталлировать разделы документации из локальных источников, например с CD/DVD, карты SD или с жесткого диска. Для этого необходимо перейти к *манифестному файлу* (manifest file), имеющему расширение `.msha`, который сопровождает документацию, и команда Help Library Manager извлечет его из источника и инсталлирует сопутствующую документацию.

Последняя команда, Remove content, позволяет удалить разделы документации с локального диска и освободить дисковую память. На экране будут показаны инсталлированные к данному моменту разделы документации, и пользователь может полностью удалить любой раздел документации, щелкнув на кнопке Remove, расположенной после заголовка раздела.

Резюме

Система Visual Studio 2010 имеет много инструментов поиска и замены, каждый из которых наиболее эффективен в конкретных ситуациях и позволяет пользователю быстро и легко находить и модифицировать код.

Новая справочная система представляет собой мощный интерфейс, обеспечивающий доступ к документации, сопровождающей пакет инсталляции Visual Studio 2010. Возможность переключаться между автономным и интерактивным режимами позволяет устанавливать баланс между скоростью поиска информации в локальной документации и релевантностью информации, найденной в веб. Кроме того, для каждого результата поиска она показывает один абзац, независимо от его расположения, чтобы минимизировать количество ложнопозитивных результатов.

ЧАСТЬ II

Начинаем работу

- **ГЛАВА 6.** Решения, проекты и элементы управления
- **ГЛАВА 7.** Технология IntelliSense и закладки
- **ГЛАВА 8.** Сниппеты кода и рефакторинг
- **ГЛАВА 9.** Окно Server Explorer
- **ГЛАВА 10.** Моделирование с помощью утилиты Class Designer

6

Решения, проекты и элементы управления

В ЭТОЙ ГЛАВЕ...

- Создание и настройка решений и проектов
- Управление компиляцией, отладкой и развертыванием приложения
- Настройка свойств нескольких проектов
- Подключение ресурсов и настроек приложения
- Навязывание хорошего стиля кодирования с помощью инструментов анализа кода
- Модификация настроек конфигурации, пакетов и развертывания веб-приложений

В отличие от простейших программ, таких как “Hello World”, большинство приложений состоит из нескольких исходных файлов. Это обстоятельство порождает массу проблем, в частности, как назвать файлы, где их разместить и можно ли их использовать повторно. В интегрированной среде разработки Visual Studio 2010 принята концепция решения (solution), состоящего из ряда проектов, которые в свою очередь состоят из ряда элементов, благодаря которой разработчики могут работать с исходными файлами. Интегрированная среда разработки имеет множество встроенных инструментов, позволяющих упростить этот процесс, обеспечив разработчикам доступ к большей части их приложений. В этой главе рассматриваются структура решений и проектов, доступные типы проектов и способы настройки их конфигурации.

Структура решения

Работая с системой Visual Studio, пользователь открывает решение. При повторном редактировании специальных файлов создается временное решение, которое можно уничтожить по окончании работы. Однако решение позволяет управлять текущими файлами, поэтому в большинстве случаев его сохранение означает, что пользователь может вернуться к тому, что он делал накануне, и вновь открыть файлы, с которыми он работал.



Решение можно интерпретировать как контейнер связанных между собой проектов. Проекты внутри решения не обязательно должны быть написаны на одном и том же языке программирования или иметь одинаковый тип. Например, решение может содержать веб-приложение ASP.NET, написанное на языке Visual Basic, библиотеку на языке F# и WPF-приложение, написанное на языке C#. Решение позволяет пользователю открыть все эти проекты в интегрированной среде разработки, а также управлять общими настройками для их создания и развертывания.

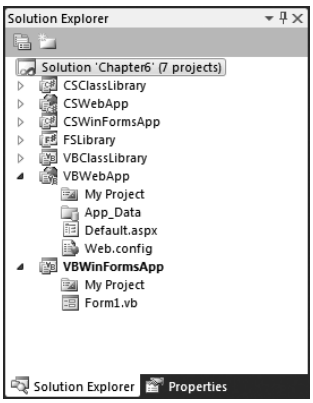


Рис. 6.1

Наиболее распространенным способом структурирования приложений в среде Visual Studio является одно отдельное решение, содержащее много проектов. Каждый проект можно создать из набора исходных файлов и папок. Главное окно, в котором пользователь работает с решениями и проектами, называется Solution Explorer (рис. 6.1).

Для организации работы с исходным кодом и предотвращения его ассоциации с приложениями (за исключением веб-приложений, в которых существуют специальные папки, имеющие особое предназначение в данном контексте) используются папки (folders). Некоторые разработчики используют имена папок, соответствующие пространствам имен, которым принадлежат классы. Например, если класс `Person` находится в папке DataClasses в проекте FirstProject, то полностью квалифицированное имя класса может выглядеть как `FirstProject.DataClasses.Person`.

Папки решения (solution folders) — полезный способ организации проектов в большом решении. Они отображаются только в окне Solution Explorer — физически в файловой системе их не существует. Такие действия, как Build или Unload, можно легко выполнять над всеми проектами, включенными в папку решения. Для того чтобы разгрузить окно Solution Explorer, папки решения могут быть свернуты или скрыты.

Скрытые проекты по-прежнему создаются, когда пользователь создает решение. Поскольку папки проекта не соответствуют физическим папкам, их можно добавлять, переименовывать и удалять в любое время, не рискуя повредить связи между файлами или потерять контроль над исходными файлами.

Бытует распространенное заблуждение, что отдельный проект обязательно должен соответствовать отдельной .NET-сборке. Хотя в принципе это верно, довольно часто одну сборку могут представлять несколько проектов. Однако эта ситуация системой Visual Studio 2010 не поддерживается, поэтому в этой книге мы будем считать, что один проект соответствует одной сборке.



Папка Miscellaneous Files – это специальная папка решения, которую можно использовать для того, чтобы следить за тем, какие еще файлы, не являющиеся частью какого-либо проекта в решении, были открыты в системе Visual Studio. По умолчанию папка Miscellaneous Files скрыта. Для того чтобы сделать ее видимой, следует выполнить команду Tools⇨Options⇨ Environment⇨Documents.

Несмотря на то что формат файла решения, принятый в предыдущих версиях, не изменился, в системе Visual Studio 2008 открыть файл решения, созданный в версии Visual Studio 2010, невозможно.

Кроме информации о файлах, содержащихся в приложении, файлы решения и проектов могут содержать и другие записи, например, о том, как именно должен быть скомпилирован конкретный файл, об установках проекта, о ресурсах и многом другом. Система Visual Studio 2010 имеет немодальное диалоговое окно для редактирования свойств проекта, в то время как свойства решения по-прежнему открываются в отдельном окне. Как и следовало ожидать, свойствами проекта считаются те свойства, которые относятся только к данному проекту, например информация о сборке и связях, а свойства решения определяют общую конфигурацию для сборки приложений.

Формат файла решения

Система Visual Studio 2010 фактически создает для решения два файла, имеющих расширения `.suo` и `.sln` (solution file). Первый — это довольно неинтересный бинарный файл, который сложно редактировать. Он содержит информацию, специфичную для пользователя, например, какие файлы были открыты, когда решение закрывалось в последний раз и где находились контрольные точки. Этот файл скрыт, поэтому он не должен появляться в папке решения при использовании Windows Explorer, если не снять с него соответствующую метку.



Иногда файл *.suo* оказывается поврежденным, и это вызывает непредсказуемые последствия при сборке и редактировании приложений. Если при работе с конкретным решением система Visual Studio становится нестабильной, необходимо выйти из нее и удалить файл с расширением *.suo*. Он будет создан заново системой Visual Studio, когда решение будет открыто в следующий раз.

Файл решения с расширением `.sln` содержит информацию о решении, например список проектов, конфигурации сборки и другие настройки, не специфичные для проекта. В отличие от многих файлов, используемых в системе Visual Studio 2010, файл решения не является XML-документом. Он хранит информацию в блоках, как показано в следующем примере.

```
Microsoft Visual Studio Solution File, Format Version 11.00
# Visual Studio 10
Project("{F184B08F-C81C-45F6-A57F-5ABD9991F28F}") = "FirstProject",
    "FirstProject\FirstProject.vbproj",
    "{D4FAF2DD-A26C-444A-9FEE-2788B5F5FDD2}"
EndProject
Global
    GlobalSection(SolutionConfigurationPlatforms) = preSolution
         Debug|Any CPU = Debug|Any CPU
```

```
    EndGlobalSection
    GlobalSection(ProjectConfigurationPlatforms) = postSolution
      {D4FAF2DD-A26C-444A-9FEE-2788B5F5FDD2}.Debug|Any CPU.ActiveCfg =
                                                        Debug|Any CPU
      {D4FAF2DD-A26C-444A-9FEE-2788B5F5FDD2}.Debug|Any CPU.Build.0 =
                                                        Debug|Any CPU
    EndGlobalSection
    GlobalSection(SolutionProperties) = preSolution
       HideSolutionNode = FALSE
    EndGlobalSection
EndGlobal
```

В этом примере решение состоит из одного проекта `FirstProject`, а раздел `Global` содержит настройки, которые применяются к решению. Например, само решение будет видимым в окне Solution Explorer, потому что настройка `HideSolutionNode` установлена равной `FALSE` . Если изменить ее на `TRUE` , имя решения не будет отображаться в системе Visual Studio.



Если решение состоит из проектов, которые не нацелены на использование платформы .NET Framework версии 4.0, решение можно открыть в системе Visual Studio 2008, выполнив небольшое редактирование файла `.sln`. Необходимо просто заменить две первые строки файла приведенными ниже, и решение откроется без ошибок.

```
Microsoft Visual Studio Solution File, Format Version 10.00
# Visual Studio 2008
```

Свойства решения

Для того чтобы открыть диалоговое окно Properties, необходимо щелкнуть правой кнопкой мыши на узле Solution в окне Solution Explorer и выбрать команду Properties. Это диалоговое окно содержит два узла: Common properties и Configuration properties, как показано на рис. 6.2.

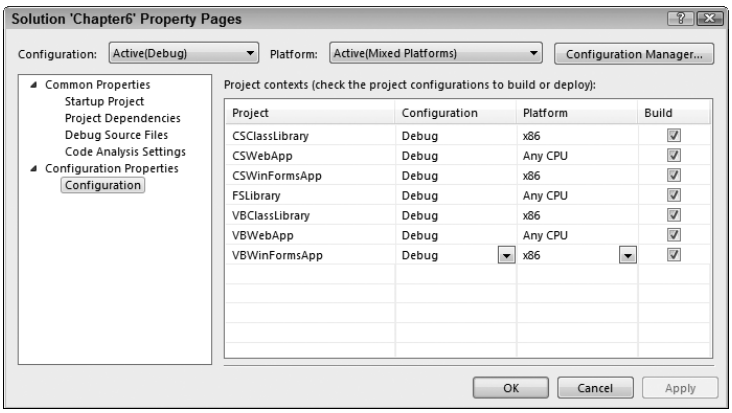


Рис. 6.2

Более подробно узлы Common properties и Configuration properties описываются в следующих разделах.

Узел Common Properties

Определяя проект Startup Project для приложения, пользователь имеет три возможности, которые являются практически очевидными. Выбор Current Selection запускает проект, который в данный момент находится в фокусе окна Solution Explorer. Вариант Single Startup гарантирует, что каждый раз будет запускаться один и тот же проект. Эта установка задается по умолчанию, поскольку большинство приложений имеют только один стартовый проект. Последний вариант, Multiple Startup Projects, позволяет запускать несколько проектов в определенном порядке. Это может быть полезным при работе с приложением клиент/сервер в рамках одного решения, причем требуется, чтобы и клиент, и сервер выполнялись одновременно. При выполнении нескольких проектов важно контролировать порядок их запуска. Для управления порядком запуска проектов можно использовать навигационные кнопки, расположенные после списка проектов.

Раздел Project Dependencies используется для того, чтобы задавать другие проекты, от которых зависит конкретный проект. В большинстве случаев система Visual Studio сама управляет этими свойствами, когда пользователь добавляет или удаляет связи между проектами и данным проектом. Однако иногда пользователь может самостоятельно создать связи между проектами, чтобы они собирались в заданном порядке. Система Visual Studio использует этот список зависимостей, для того чтобы определить порядок сборки проектов. Окно этого раздела предотвращает неосторожное добавление циклических связей и удаление необходимых зависимостей между проектами.

В разделе Debug Source Files можно создать список каталогов, в которых система Visual Studio может искать исходные файлы при отладке. Этот список задается по умолчанию и просматривается перед открытием диалогового окна Find Source. Кроме того, пользователь может перечислить исходные файлы, которые система Visual Studio не должна искать. Если щелкнуть на кнопке Cancel в момент, когда система предлагает найти конкретный исходный файл, то он будет добавлен в этот список.

Раздел Code Analysis Settings доступен только в версии Visual Studio Team Suite. Это позволяет выбирать набор правил статического анализа кода, которые будут применяться к конкретному проекту. Более подробно раздел Code Analysis обсуждается далее.

Узел Configuration Properties

И проекты, и решения имеют конфигурации для сборки, определяющие, какие элементы должны быть собраны и почему. Это может сбить пользователя с толку, потому что на самом деле между конфигурацией проекта, определяющей, как должны собираться элементы, и конфигурацией решения, определяющей, какие проекты должны быть собраны, нет никакой корреляции, кроме случаев, когда они имеют одинаковые имена. Новое решение определит конфигурации Debug и Release (решения), что эквивалентно сборке всех проектов в решении с помощью конфигураций Debug и Release (проекта).

Например, может быть создана новая конфигурация решения Test, состоящая из двух проектов: `MyClassLibrary` и `MyClassLibraryTest`. Когда пользователь создает свое приложение в конфигурации `Test`, он хочет, чтобы проект `MyClassLibrary` был собран в режиме `Release`, чтобы тестировать его в виде, максимально приближенном к окончательной версии. Однако, чтобы проверить тестируемый код, необходимо собрать тестовый проект в режиме `Debug`.

Когда пользователь собирает проект в режиме Release, он не хочет, чтобы решение Test было собрано или развернуто вместе с приложением. В данном случае в конфигурации решения Test можно указать, что пользователь хочет, чтобы проект MyClassLibrary был собран в режиме Release, а проект MyClassLibraryTest вообще не собирался.



Пользователь может легко переключаться между этими конфигурациями с помощью меню Configuration стандартной инструментальной панели. Однако переключаться между платформами не так легко, потому что меню Platform нет ни в одной инструментальной панели. Для того чтобы сделать ее доступной, необходимо выбрать команду View⇨ Toolbars⇨Customize. Затем элемент Solution Platforms из категории Build на закладке Command можно перетащить на инструментальную панель.

Следует отметить, что, выбрав узел Configuration Properties в диалоговом окне Solution Properties, как показано на рис. 6.2, можно получить доступ к раскрывающимся спискам Configuration и Platform. Раскрывающийся список Configuration содержит все доступные конфигурации решения: Debug и Release (заданные по умолчанию), Active и All. Аналогично в раскрывающемся списке Platform перечислены все доступные платформы. Как только пользователь получит доступ к этим раскрывающимся спискам, он может на этой же странице задать настройки для каждой конфигурации и/или платформы. Для того чтобы добавить новые конфигурации и/или платформы для решения, пользователь может также использовать кнопку Configuration Manager.

При добавлении новых конфигураций решения существует возможность (предусмотренная по умолчанию) создания соответствующих конфигураций для существующих проектов (по умолчанию все проекты будут собираться в новой конфигурации решения), а также возможность создать новую конфигурацию на основе существующих. Если флажок Create Project Configurations установлен и новая конфигурация основана на существующей, то новые конфигурации проекта будут копировать конфигурации проекта, заданные для существующей конфигурации.

Возможности, доступные для создания новых платформных конфигураций, ограничены типами доступных центральных процессоров: Itanium, x86 и x64. Новая платформная конфигурация также может основываться на существующих платформных конфигурациях, и существует возможность создания платформной конфигурации для проекта.

В конфигурационном файле решения можно также задать тип центрального процессора, для которого оно собирается. Это особенно удобно, если нужно развернуть приложение для компьютеров с 64-битовой архитектурой. Установить настройки для всех этих решений можно непосредственно в контекстном меню, которое открывается после щелчка правой кнопкой мыши на узле Solution node в окне Solution Explorer. В то время как команда Set Startup Projects открывает окно конфигурации решения, команды Configuration Manager, Project Dependencies и Project Build Order открывают окно Configuration Manager and Project Dependencies. Команды Project Dependencies и Project Build Order отображаются в окне, только если решение состоит из нескольких проектов.

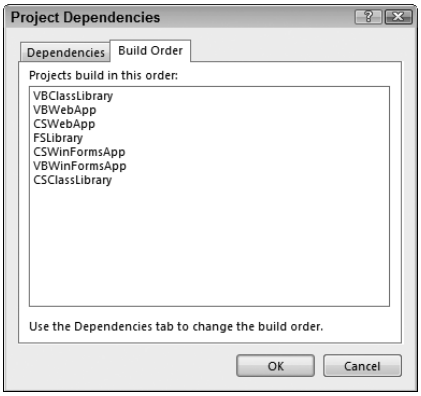


Рис. 6.3

Команда Project Build Order открывает окно Project Dependencies и перечисляет порядок сборки, как показано на рис. 6.3. Закладка Build Order демонстрирует порядок, в котором должны собираться проекты в соответствии с зависимостями между ними. Это может оказаться полезным, если пользователь поддерживает ссылки на бинарные сборки проектов, а не ссылки на проекты. Кроме того, эту возможность можно использовать для двойной проверки того, что проекты будут собраны в правильном порядке.

Типы проектов

В системе Visual Studio проекты, написанные на языках Visual Basic и C#, разделяются на разные категории. За исключением проектов для веб-сайтов, которые будут отдельно рассмотрены в данной главе, каждый проект содержит файл проекта (`.vbproj` или `.csproj`) со схемой MSBuild. Выбрав шаблон проекта, пользователь создает новый проект, имеющий конкретный тип, начальные классы и настройки. Ниже описаны некоторые из наиболее общих категорий проектов в системе Visual Studio.

- **Windows.** Категория проектов Windows — самая крупная. Она включает в себя большинство распространенных типов проектов, выполняемых в пользовательских операционных системах. В частности, она содержит выполняемые проекты Windows Forms, проекты приложений Console и приложения Windows Presentation Foundation (WPF). Эти типы проектов предусматривают создание сборки (`.exe`), выполняемой непосредственно пользователем. Категория Windows включает также несколько типов библиотечных сборок, на которые легко ссылаться из других проектов. К ним относятся как библиотеки классов, так и библиотеки элементов управления для приложений Windows Forms и WPF. Файлы библиотек классов имеют хорошо известные расширения DLL. К этой же категории относится тип проектов Windows Service.
- **Web.** Категория Web содержит типы проектов, выполняемых на платформе ASP.NET. К ней относятся веб-приложения ASP.NET, веб-службы XML, библиотеки элементов управления для использования в веб-приложениях, включая веб-приложения, снабженные средствами AJAX.
- **Office.** Как следует из названия этой категории, к ней относятся управляемые программные надстройки для продуктов Microsoft Office, таких как Outlook, Word или Excel. Проекты этих типов используют средства Visual Studio Tools for Office (VSTO) и способны создавать надстройки для большинства продуктов, включая пакеты Office 2003 и Office 2007.
- **SharePoint.** Эта категория с самоочевидным названием содержит проекты, предназначенные для веб-платформ Windows SharePoint Services, таких как SharePoint Workflows или Team Sites.
- **Database.** Данная категория содержит типы проектов для создания кода, который можно использовать на сервере SQL Server. К ним относятся хранимые

процедуры, пользовательские типы и функции, триггеры и специальные агрегирующие функции.

- **Reporting.** К этой категории относится тип проекта, идеально подходящий для быстрой генерации сложных отчетов об источнике данных.
- **Silverlight.** Эта категория содержит типы для создания приложений Silverlight Applications или проектов Class Library.
- **Test.** Данная категория включает в себя тип для проектов, содержащих тесты, использующие каркас для тестирования модулей MSTest.
- **WCF.** Эта категория содержит большое количество типов проектов для создания приложений, обеспечивающих веб-службы с использованием технологии Windows Communication Foundation (WCF).
- **Workflow.** Эта категория содержит большое количество типов проектов для создания библиотек и приложений, реализующих последовательные процессы и конечные автоматы.

Диалоговое окно New Project в системе Visual Studio 2010, показанное на рис. 6.4, позволяет просматривать и создавать любые из указанных типов проектов. Искомая версия .NET Framework указана в раскрывающемся списке, расположенном в правом верхнем углу этого диалогового окна. Если тип проекта не поддерживается выбранной версией .NET Framework, например тип приложения WPF для платформы .NET Framework 2.0, то проект отображаться не будет.

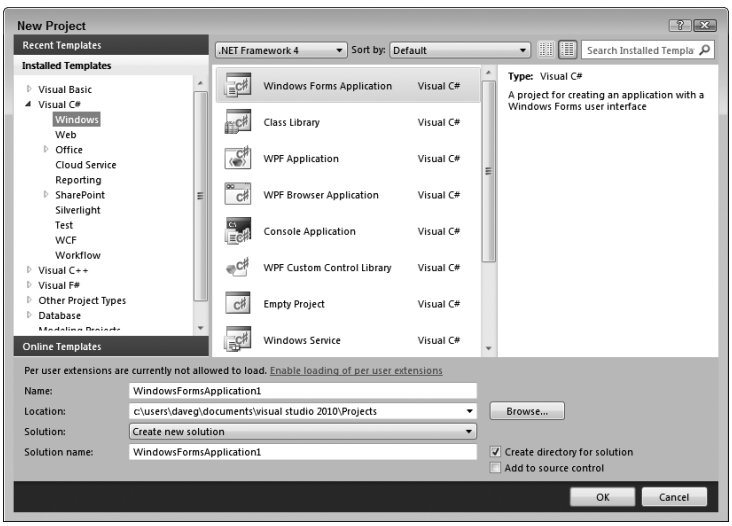


Рис. 6.4

Формат файлов проекта

Файлы проекта (`.csproj`, `.vbproj` или `.fsproj`) — это текстовые файлы в формате XML-документа, согласованные со схемой MSBuild. Схемные XML-файлы для новейшей версии MSBuild вместе с платформой .NET Framework по умолчанию инсталлируются в каталог `C:\WINDOWS\Microsoft.NET\Framework\v4.0.20506\MSBuild\Microsoft.Build.Core.xsd`.



Для того чтобы отобразить файл проекта в формате XML, следует щелкнуть правой кнопкой мыши на проекте и выбрать команду Unload. Затем необходимо снова щелкнуть правой кнопкой мыши на проекте и выбрать команду Edit <project name>. В результате в окне редактора XML, поддерживающего технологию IntelliSense, появится файл проекта.

В файле проекта хранятся настройки сборки и конфигурации проекта, а также информация обо всех файлах, включенных в проект. В некоторых ситуациях создается специальный пользовательский проект (`.csproj.user` или `.vbproj.user`), в котором хранятся настройки пользователя, относящиеся к запуску и отладке. Файл с расширением `.user` также является XML-файлом, согласованным со схемой MSBuild.

Свойства проекта

Для того чтобы открыть свойства проекта, нужно щелкнуть правой кнопкой мыши на узле Project в окне Solution Explorer и выбрать команду Properties либо дважды щелкнуть на узле My Project (или Properties, если проект создается на языке C#), расположенном под узлом Project. В отличие от свойств решения свойства проекта отображаются не в модальном окне, а на дополнительной закладке, расположенной рядом с исходными файлами. Это сделано отчасти для того, чтобы было легче перемещаться между файлами программы и свойствами проекта, а отчасти для того, чтобы можно было одновременно открыть свойства нескольких проектов. На рис. 6.5 показаны настройки проекта, имеющего тип Visual Basic Windows Forms. В этом разделе можно пройти по всем вертикальным закладкам редактора проекта как для проектов на языке Visual Basic, так и для проектов на языке C#.

Редактор свойств проекта содержит ряд вертикальных закладок, группирующих свойства. Если свойство на закладке было изменено, то на соответствующей вертикальной закладке добавляется звездочка. Однако эта функциональная возможность ограничена, поскольку она не позволяет понять, какие именно поля на закладке были изменены.

Закладка Application

Закладка Application, показанная на рис. 6.5 для проекта, имеющего тип Visual Basic Windows Forms, позволяет разработчику вводить информацию о сборке, которая будет создана в ходе компиляции проекта. К этой информации относятся такие атрибуты, как тип вывода (т.е. Windows или Console Application, Class Library, Windows Service или Web Control Library), пиктограмма приложения и запускаемый объект. Закладка Application для приложения C#, показанная на рис. 6.6, имеет другой формат и обеспечивает дополнительные возможности, например возможность выбрать целевую версию платформы .NET Framework.

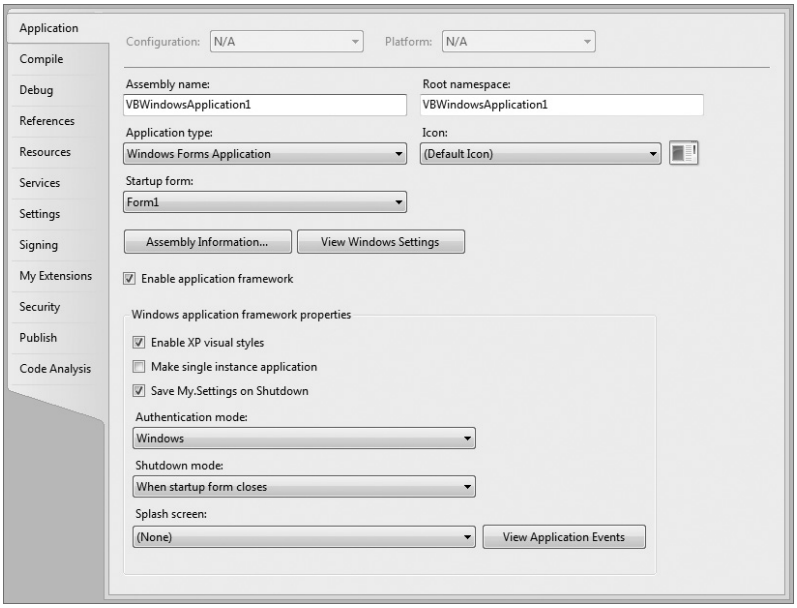


Рис. 6.5

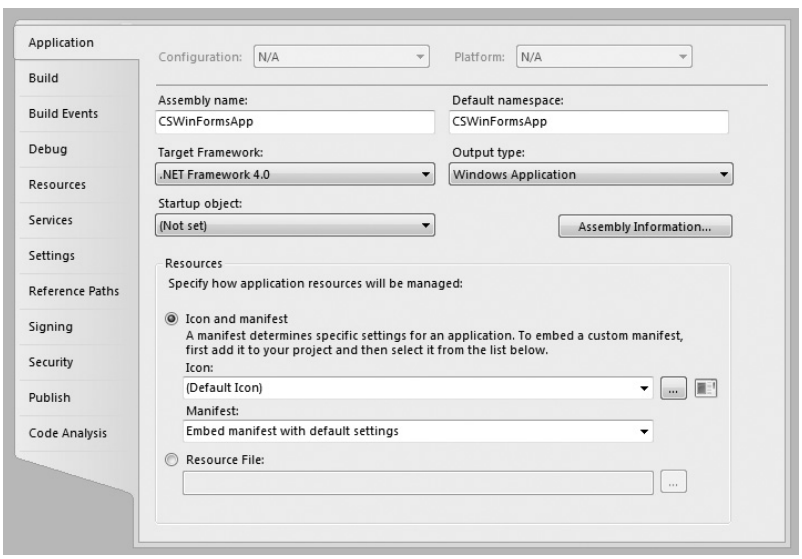


Рис. 6.6



Для того чтобы изменить версию платформы .NET Framework для проекта на языке Visual Basic, используется группа Advanced Compile Options на закладке Compile.

Информация о сборке

Атрибуты, которые ранее задавались вручную в файле `AssemblyInfo`, содержащемся в проекте, теперь можно задать также с помощью кнопки Assembly Information. Это важная информация, поскольку она отображается в момент инсталляции приложения и при просмотре свойств файла в окне Windows Explorer. На рис. 6.7, *слева*, показана информация о сборке для простого приложения, а на рис. 6.7, *справа*, — свойства скомпилированного выполняемого файла.

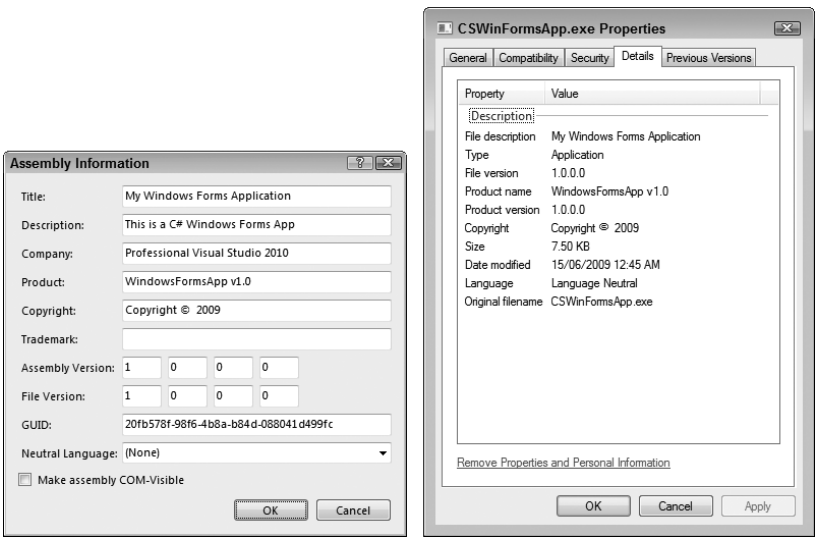


Рис. 6.7

Каждый из наборов свойств в диалоговом окне Assembly Information представляется атрибутом, который применяется ко всей сборке. Это значит, что из кода можно послать запрос к сборке и извлечь эту информацию. При работе с языком Visual Basic простой способ извлечения этой информации обеспечивает пространство имен `My.Application.Info`.

Настройки User Account Control

Система Visual Studio 2010 поддерживает разработку приложений, работающих с компонентом User Account Control (UAC) под управлением операционных систем Windows Vista и Windows 7. Этот процесс включает в себя генерирование манифестного файла сборки, представляющего собой XML-файл, уведомляющий операционную систему о запросах приложения на получение административных привилегий при запуске. В приложениях на языке Visual Basic для генерации учетной записи пользователя и добавления манифестного файла сборки в приложение пользователя используется кнопка View Windows Settings на закладке Application. В следующем коде продемонстрирован манифестный файл, сгенерированный по умолчанию системой Visual Studio.

```
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1"
       xmlns:asmv1="urn:schemas-microsoft-com:asm.v1"
       xmlns:asmv2="urn:schemas-microsoft-com:asm.v2"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
  <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <!-- UAC Manifest Options
           Если вы хотите изменить уровень Windows User Account Control,
            замените узел requestedExecutionLevel одним из следующих.

        <requestedExecutionLevel level="asInvoker" />
        <requestedExecutionLevel level="requireAdministrator" />
        <requestedExecutionLevel level="highestAvailable" />

           Если вы хотите обеспечить обратную совместимость с помощью
           технологии File and Registry Virtualization,
           удалите узел requestedExecutionLevel.
        -- >
        <requestedExecutionLevel level="asInvoker" />
      </requestedPrivileges>
      <applicationRequestMinimum>
        <defaultAssemblyRequest permissionSetReference="Custom" />
        <PermissionSet ID="Custom" SameSite="site" />
      </applicationRequestMinimum>
    </security>
  </trustInfo>
</asmv1:assembly>
```

Если значение запрошенного уровня исполнения, заданное по умолчанию, было изменено с `asInvoker` на `require Administrator`, то система Windows при запуске приложения выдаст приглашение UAC. Если запросы UAC допускаются и приложение, требующее прав администратора, запущено в режиме Debug, то система Visual Studio 2010 также выведет приглашение на повторный запуск с правами администратора. На рис. 6.8 показано приглашение, свидетельствующее о том, что система Windows позволяет перезапустить среду Visual Studio с правами администратора.

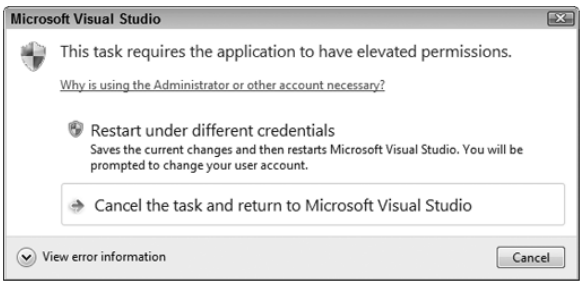


Рис. 6.8

Если пользователь согласен на повторный запуск, то система Visual Studio будет не только перезапущена и предоставит пользователю права администратора, но и заново откроет решение, включая все ранее открытые файлы. Она даже запомнит позицию курсора, в которой он находился до перезапуска.

Настройки программных компонентов Application Framework (только для языка Visual Basic)

В проектах типа Visual Basic Windows Forms можно задавать дополнительные настройки, поскольку они используют набор программных компонентов Application

Framework, предназначенный исключительно для языка Visual Basic. Это позволяет расширить стандартную событийную модель и обеспечить ряд событий, происходящих с приложением, а также настройки, управляющие поведением этого приложения. Для того чтобы получить доступ к набору программных компонентов Application Framework, необходимо установить флажок Enable Application Framework. Имеются три флажка, управляющих поведением набора компонентов Application Framework.

- **Enable XP Visual Styles.** Визуальные стили XP представляют собой свойство, которое значительно улучшает внешний вид и впечатление от приложения, выполняемого под управлением операционной системы Windows XP или систем Windows более поздних версий, поскольку они сглаживают углы используемых кнопок и элементов управления, динамически изменяющих цвет при прохождении над ними курсора мыши. В приложениях на языке Visual Basic стили XP доступны по умолчанию и могут быть отключены с помощью диалогового окна Project Settings. Кроме того, ими можно управлять из кода с помощью метода `EnableVisualStyles` в классе `Application`.
- **Make Single Instance Application.** Большинство приложений поддерживает параллельное выполнение нескольких экземпляров приложения. Однако некоторые приложения, открытые более двух или трех раз, могут быть запущены только последовательно, и в таких случаях при каждом выполнении просто вызывается их оригинальный вариант. Таким приложением может быть редактор документов, причем при каждом его новом выполнении просто открывается новый документ. Это функциональное свойство можно легко добавить, указав, что приложение может существовать только в одном экземпляре.
- **Save My.Settings on Shutdown.** Выбрав команду Selecting the Save My.Settings on Shutdown, пользователь гарантирует, что любые изменения, внесенные в пользовательских настройках, будут сохранены до завершения выполнения приложения.

Раздел настройки программных компонентов Application Framework также позволяет выбрать режим аутентификации для приложения. По умолчанию он настроен на операционную систему Windows и текущего зарегистрированного пользователя. Выбрав команду Application-defined, пользователь может использовать свой собственный модуль аутентификации. Он может также идентифицировать форму, которую следует использовать в качестве заставки при первом запуске приложения, и задать поведение приложения по завершении его работы.

Раздел Compile (только для языка Visual Basic)

Как показано на рис. 6.9, раздел Compile в настройках проекта позволяет разработчикам управлять тем, как и где собирать проект. Например, можно модифицировать путь вывода, указав альтернативный каталог. Это может оказаться важным, если этот вывод должен использоваться в процессе сборки.

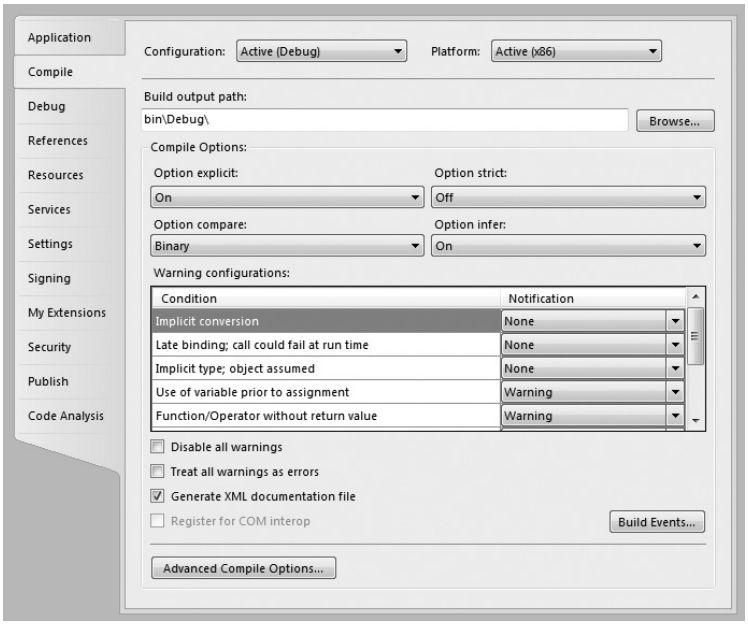


Рис. 6.9

Раскрывающийся список Configuration в верхней части закладки позволяет задавать разные настройки для конфигураций сборки `Debug` и `Release`. Если диалоговое окно не выводит список Configuration, то следует выбрать команду Show Advanced Build Configurations в узле Projects and Solutions в окне Options, которое открывается из меню Tools. К сожалению, это свойство не относится к некоторым профилям настроек, например, к профилю `Visual Basic Developer`.

Некоторые свойства, характерные для языка Visual Basic, можно настроить в окне Compile. С помощью флажка Explicit можно указать, должны ли переменные, используемые в коде, быть явно определенными. Используя флажок Strict, можно потребовать, чтобы переменные были определены на этапе раннего, а не позднего связывания. Установив флажок Compare, можно регламентировать метод сравнения строк — с помощью бинарных или текстовых операторов сравнений. С помощью флажка Infer можно указать, следует ли допускать вывод локальных типов в объявлениях переменных, или эти типы должны быть явно указаны.



Всеми этими четырьмя вариантами компиляции можно управлять либо на уровне свойств проекта, либо на уровне свойств файлов. Опции компиляции на уровне файлов замещают опции, заданные на уровне проекта.

Окно Compile также содержит большое количество настроек компиляции, управление которыми позволяет улучшать читабельность кода. Например, неиспользуемые переменные могут вызывать только предупреждения, а пути выполнения кода, которые не возвращают никаких значений, являются серьезными недостатками и должны генерировать сообщение об ошибке при сборке. Кроме того, существует возможность либо отключить все предупреждения вообще, либо трактовать их все как сообщения об ошибках.

Разработчики, использующие язык Visual Basic, имеют также возможность генерировать XML-документы. Разумеется, поскольку генерация документов требует времени, рекомендуется отключить эту возможность при отладке сборки. Это позволит ускорить цикл отладки; однако пользователь не получит никаких сообщений об отсутствии генерации XML-документации, если эта возможность отключена.

Последний элемент окна Compile — кнопка Build Events. Щелкнув на этой кнопке, пользователь увидит команды, которые могут быть выполнены до и после сборки. Поскольку не каждая сборка завершается успешно, событие, генерируемое после сборки, может зависеть от успешности самой сборки. Проекты на языке C# имеют отдельную закладку Build Events на страницах свойств проекта, на которых можно настроить события, генерируемые до и после сборки.

Закладка Build (только для языков C# и F#)

Закладка Build, предусмотренная для языка C# (рис. 6.10), — эквивалент закладки Compile в языке Visual Basic. Эта закладка позволяет разработчику задать настройки для сборки проекта. Например, пользователь может установить флажок Optimize code и в результате получить более компактную, быструю и эффективную сборку. Однако такая оптимизация обычно увеличивает продолжительность сборки и поэтому не рекомендуется для сборки в режиме Debug.

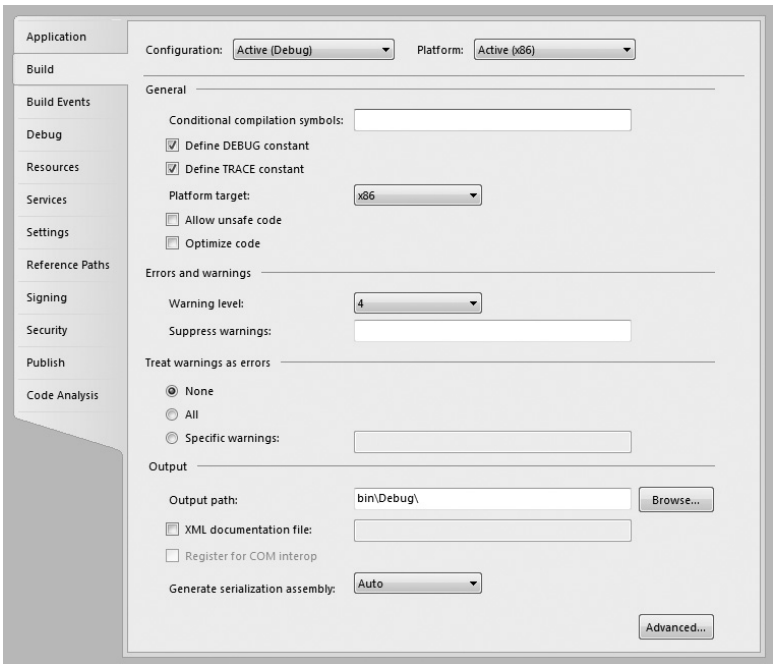


Рис. 6.10

На закладке Build можно установить флажки для определения констант компиляции DEBUG и TRACE. В качестве альтернативы можно легко установить свои собственные константы в текстовом поле Conditional compilation symbols. Значения этих констант можно запросить из кода в ходе компиляции. Например, константу DEBUG можно запросить следующим образом.

C#

```
#if(DEBUG)
    MessageBox.Show("Константа DEBUG определена");
#endif
```

VB

```
#If DEBUG Then
    MessageBox.Show("Константа DEBUG не определена")
#End If
```

Константы компиляции определяются в диалоговом окне Advanced Compiler Settings, которое открывается после щелчка на кнопке Advanced Compile Options..., расположенной на закладке Compile. Раскрывающийся список Configuration, расположенный в верхней части закладки, позволяет задавать разные параметры сборки для конфигураций Debug и Release. Более подробная информация о параметрах команды Build приведена в главе 45.

Закладка Build Events (только для языков C# и F#)

Закладка Build Events позволяет выполнить дополнительные действия до или после сборки. На рис. 6.11 показано событие, генерируемое после каждой успешной сборки, в результате которого выполняется команда FXCop Static Code Analysis.

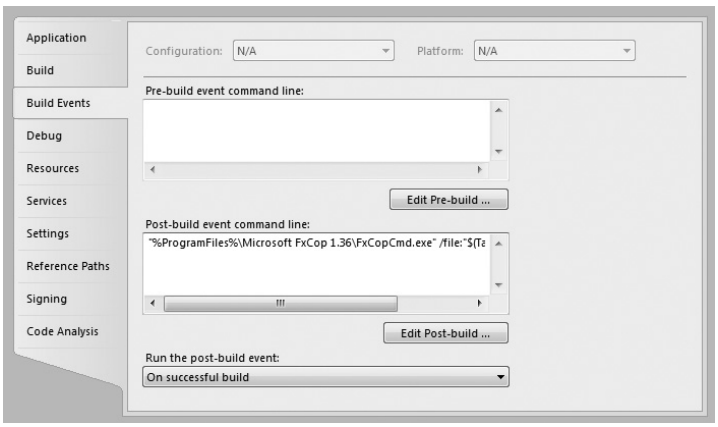


Рис. 6.11

В командных строках можно использовать переменные окружения, такие как `ProgramFiles`, поместив их между знаками процента, как между скобками. Кроме того, разработчику доступны многие макросы, такие как `ProjectPath`, которые перечислены в диалоговом окне Edit Pre-build and Edit Post-build.

Закладка Debug

Закладка Debug, продемонстрированная на рис. 6.12, определяет, как должно выполняться приложение в системе Visual Studio 2010. Эта закладка остается невидимой для веб-приложений — вместо нее в этих ситуациях используется закладка Web.

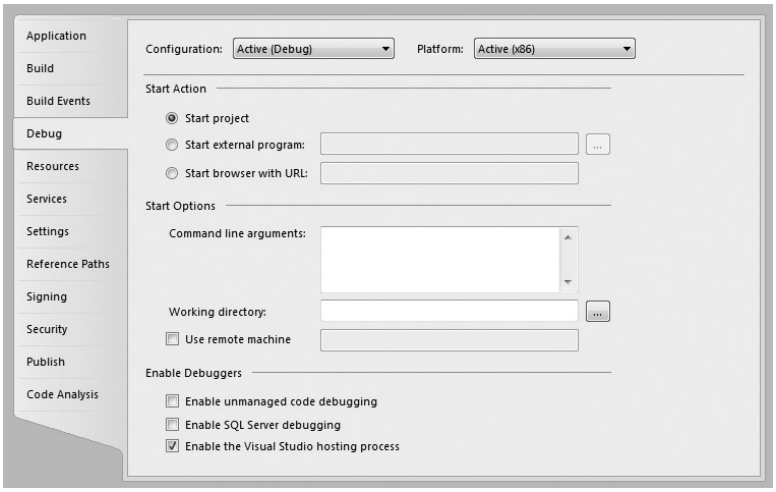


Рис. 6.12

Группа Start Action

Когда настраивается запуск проекта, этот набор переключателей определяет, что именно должно произойти при запуске приложения. Именно этот набор действий приведет к вызову объекта Startup, указанного на закладке Application. Другие параметры определяют либо выполнение внешних программ, либо открытие конкретного веб-сайта.

Группа Start Options

Кроме того, разработчик может задать дополнительные аргументы командной строки, влияющие на запуск приложения (обычно они используются в сочетании с действием, выполняемым на старте) и начальный рабочий каталог. Можно также потребовать запуска приложения на удаленном компьютере. Разумеется, все это возможно, только если существует доступ к этому удаленному компьютеру.

Группа Enable Debuggers

Используя параметры этой группы, можно уточнить процесс отладки, включив в него неуправляемый код и SQL Server. Можно также включить *главный процесс* (hosting process) системы Visual Studio, который имеет много преимуществ, связанных с производительностью и функциональностью отладки. Эти преимущества разделяются на три категории. Во-первых, главный процесс действует как фоновый главный компьютер (background host) для отлаживаемого приложения. Для того чтобы отладить приложение, необходимо решить разные административные задачи, например создать домен приложения и соответствующий отладчик, что требует времени. Включив главный процесс, все эти задачи можно решить в фоновом режиме. В результате время загрузки во время отладки намного сократится.

Во-вторых, в системе Visual Studio 2010 довольно легко создавать, отлаживать и разворачивать приложения, которые выполняются в условиях *частичного доверия* (partial trust). Главный процесс в таких условиях является важным инструментом, поскольку именно он дает разработчику возможность запускать и отлаживать приложе-

ние в режиме частичного доверия. Без этого процесса приложение запускалось был в режиме *полного доверия* (full trust), а отладка в режиме частичного доверия была бы невозможной.

Последнее преимущество, которое обеспечивает разработчикам главный процесс, заключается в вычислении выражений в ходе разработки. Этот эффект является оптической иллюзией, поскольку на самом деле главный процесс выполняется в фоновом режиме. Однако, используя окно Immediate во время редактирования текста, разработчик может легко вычислять выражения, вызывать методы и даже переходить к контрольным точкам без запуска всего приложения.

Закладка References (только для языка Visual Basic)

Закладка References позволяет разработчику ссылаться на классы, принадлежащие к другим .NET-сборкам, проектам или библиотекам DLL. Как только проект или библиотека DLL добавлены в список ссылок, класс становится доступен либо по своему полному имени, включающему пространство имен, либо просто по имени, если пространство имен импортировано в исходный файл. На рис. 6.13 показана закладка References для проекта, ссылающегося на множество сборок.

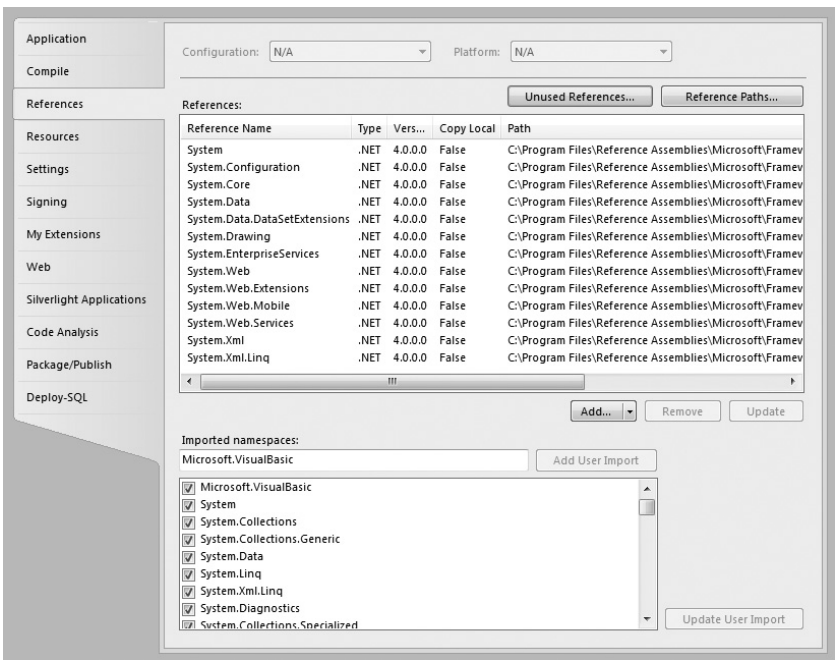


Рис. 6.13

Одной из особенностей этой закладки, предназначенной для разработчиков программ на языке Visual Basic, является кнопка Unused References, позволяющая выполнять поиск ссылок, которые можно удалить. Кроме того, можно добавлять *пути к ссылкам* (reference path), что дает возможность включать все сборки, расположенные в данном месте.

Как только сборка добавлена в список ссылок, любой открытый класс, содержащийся в этой сборке, становится видимым в проекте. Если класс вложен в пространство имен (которое может быть вложенной иерархией), то для ссылки на класс необходимо указывать его полное имя. В языках Visual Basic и C# предусмотрен механизм импортирования пространств имен, чтобы на классы можно было ссылаться непосредственно. Раздел References позволяет осуществлять глобальный импорт пространств имен для классов, принадлежащих проекту, без явного импорта в файл класса.

Ссылки на внешние сборки могут быть ссылками на файл или проект. Ссылка на файл — это прямая ссылка на отдельную сборку. Такие ссылки создаются с помощью закладки Browse в диалоговом окне Add Reference. Ссылка на проект — это ссылка на отдельный проект в решении. Все сборки, создающиеся проектом, динамически добавляются как ссылки. Ссылки на проекты создаются с помощью закладки Project в диалоговом окне Add Reference.



Обычно не следует добавлять ссылку на файл, если нужна ссылка на проект, существующий в том же самом решении.

Преимущество ссылки на проект заключается в том, что она создает зависимость между проектами в системе сборки. Зависимый проект может быть собран заново, если он был изменен с момента последней сборки проекта, который на него ссылается. Ссылка на файл не создает зависимости между сборками, поэтому возможна ситуация, когда проект, ссылающийся на файл, будет собран, а зависимый проект — нет. Однако это может создать проблемы в ссылающемся проекте, который ожидает другую версию зависимого проекта.

Закладка Resources

Ресурсы проекта можно добавлять и удалять с помощью закладки Resources, показанной на рис. 6.14. В приведенном примере в приложение были добавлены четыре пиктограммы. Ресурсами могут быть изображения, текст, файлы и другие сериализуемые классы.

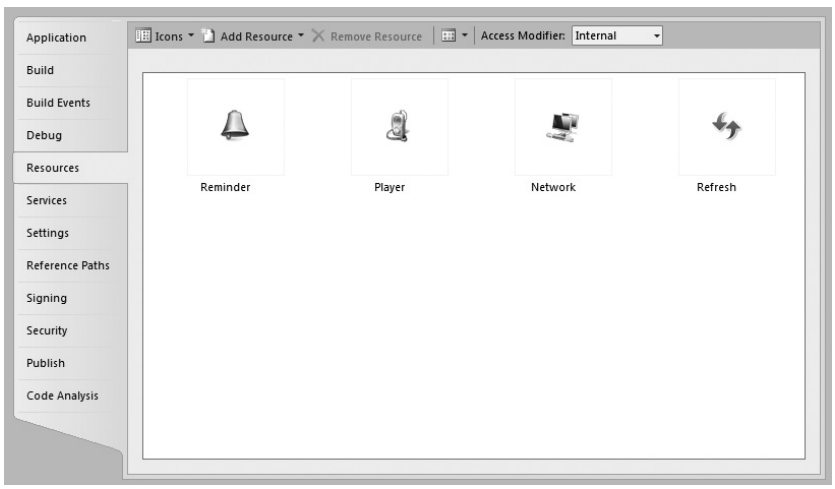


Рис. 6.14

Этот интерфейс существенно облегчает работу с файлами ресурсов во время проектирования. Использование файлов ресурсов для хранения констант приложения и его интернационализации описано в главе 38.

Закладка Services

Клиентские службы позволяют приложениям, созданным для системы Windows, использовать службы аутентификации, ролей и профилей, предусмотренные технологией Microsoft ASP.NET 2.0. Клиентские службы позволяют централизовать профили пользователей многочисленных веб- и Windows-приложений и осуществлять их администрирование.

На рис. 6.15 показана закладка Services, которая используется для конфигурирования клиентских служб, предназначенных для Windows-приложений. При включении этих служб для каждого из них должен быть указан адрес URL узла службы ASP.NET. Он будет храниться в файле app.config. Разработчику доступны следующие клиентские служб.

Authentication. Эта служба позволяет идентифицировать личность пользователя либо с помощью службы аутентификации Windows, либо специальной системы аутентификации, основанной на формах и обеспечиваемой приложением.

Roles. Данная служба организовывает роли, приписанные аутентифицированному пользователю. Это позволяет пользователю открывать доступ к разным частям приложения. Например, администраторам могут быть предоставлены некоторые административные права.

Web Settings. Эта служба хранит на сервере настройки пользовательских приложений, что позволяет распределять их по многим компьютерам и приложениям.

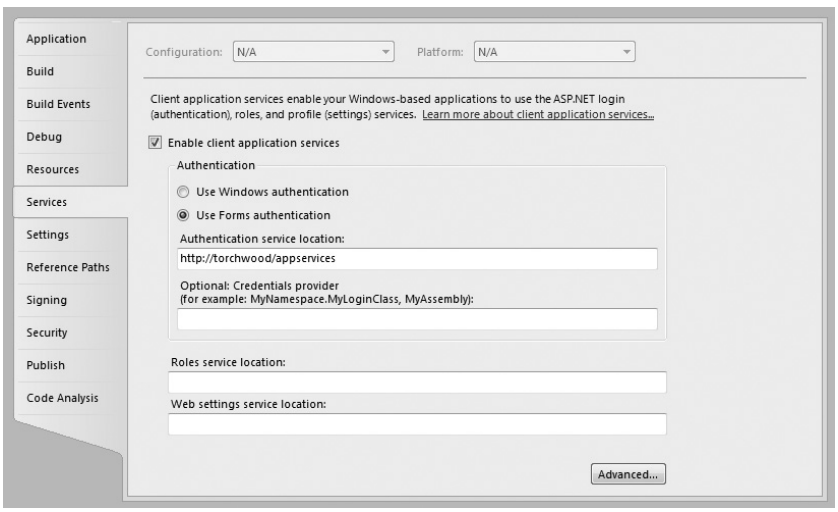


Рис. 6.15

Для расширения веб-служб клиентские службы используют модель провайдеров. Помимо прочего, провайдеры служб обеспечивают автономную работу, основанную на использовании локального кэша, что гарантирует продолжение работы приложения при утрате сетевого соединения. Клиентские службы приложения обсуждаются в главе 33.

Настройки

Настройки проекта могут иметь любой тип. Они представляют собой пары "имя–значение", из которых во время выполнения приложения можно извлечь значение. Настройки могут распространяться либо на приложение, либо на пользователя (рис. 6.16) и хранятся в файлах `Settings.settings` и `app.config`. Последний файл после компиляции приложения переименовывается с учетом имени выполняемого модуля, например `SampleApplication.exe.config`.

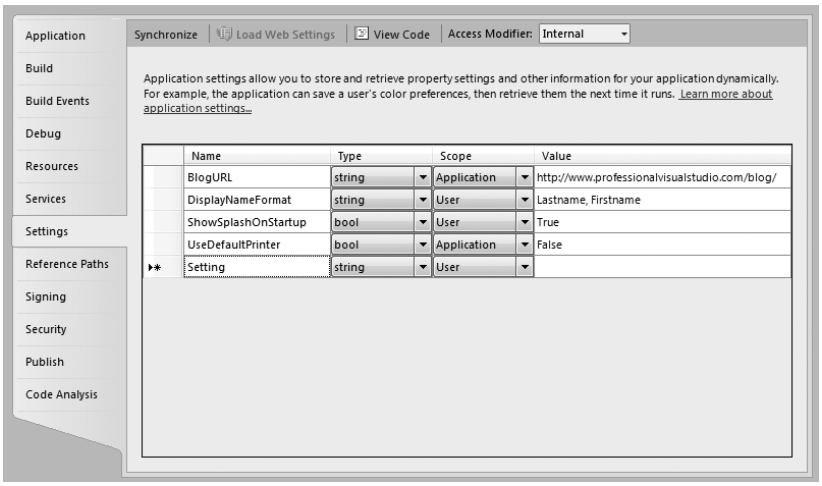


Рис. 6.16

Настройки приложений во время их выполнения доступны только для чтения. Их можно изменить, только вручную редактируя файл конфигурации. Настройки пользователей могут динамически изменяться во время выполнения приложения. Кроме того, они могут иметь разные значения, соответствующие разным пользователям, запускающим приложение. Значения настроек пользователей, заданные по умолчанию, хранятся в файле `app.config`, а настройки пользователей — в файле `user.config`, расположенном в каталоге, где хранятся приватные данные пользователей.

Настройки приложений и пользователей более подробно рассмотрены в главе 36.

Пути к ссылкам (только для языков C# и F#)

Закладка Reference Paths, показанная на рис. 6.17, используется для задания специальных каталогов, где следует искать сборки, на которые имеются ссылки. При добавлении ссылки на сборку система Visual Studio разрешает ссылку, по порядку просматривая следующие каталоги.

1. Каталог проекта.
2. Каталоги, указанные в списке Reference Paths.
3. Каталоги, файлы из которых показаны в окне Add Reference.
4. Каталоги объектных файлов для проекта. Обычно это относится только к сборкам, созданным по технологии COM Interop.

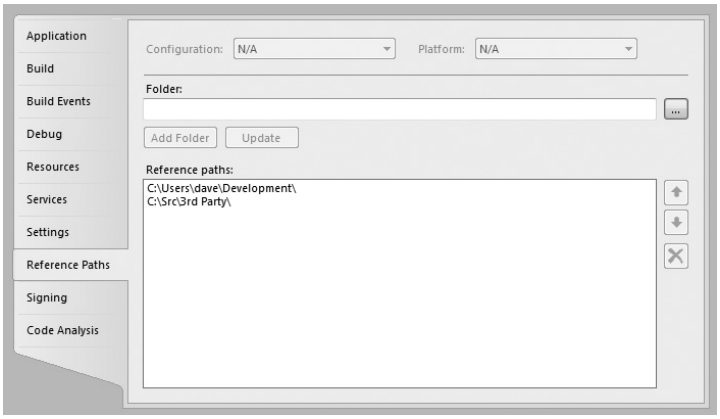


Рис. 6.17

Закладка Signing

На рис. 6.18 показана закладка Signing, предоставляющая разработчикам возможность определить способ подписи сборок во время подготовки к развертыванию. Сборка может быть подписана с помощью выбора ключевого файла. Новый ключевой файл можно создать, выбрав команду < New ...> из раскрывающегося списка файлов.

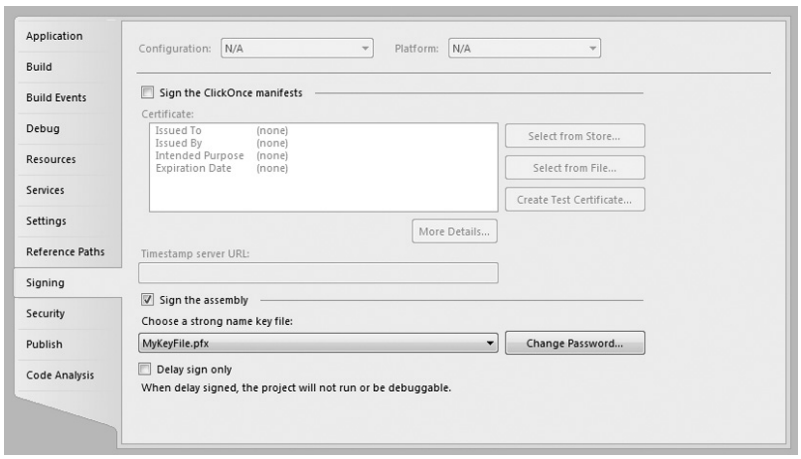


Рис. 6.18

Модель развертывания приложения ClickOnce позволяет запустить приложение на веб-сайте, на котором пользователь может один раз щелкнуть мышью, чтобы начать загрузку и инсталляцию. Поскольку эта модель предполагает поддержку развертывания через Интернет, организация должна иметь возможность подписать пакет развертывания. Закладка Signing обеспечивает интерфейс для задания сертификата, который используется для подписи манифестов ClickOnce.

Более подробно процесс подписания сборок рассмотрен в главе 46, а модель развертывания приложений ClickOnce описана в главе 48.

Закладка My Extensions (только для языка Visual Basic)

Закладка My Extensions, показанная на рис. 6.19, позволяет добавлять в сборки ссылки, расширяющие пространство имен My в языке Visual Basic. Соответствующие методы расширения были изобретены для того, чтобы можно было разделить на части проект LINQ без внесения изменений в библиотеку базовых классов. Они позволяют пользователям добавлять новые методы в существующие классы, не используя ни наследование, чтобы создать подкласс, ни повторную компиляцию класса.

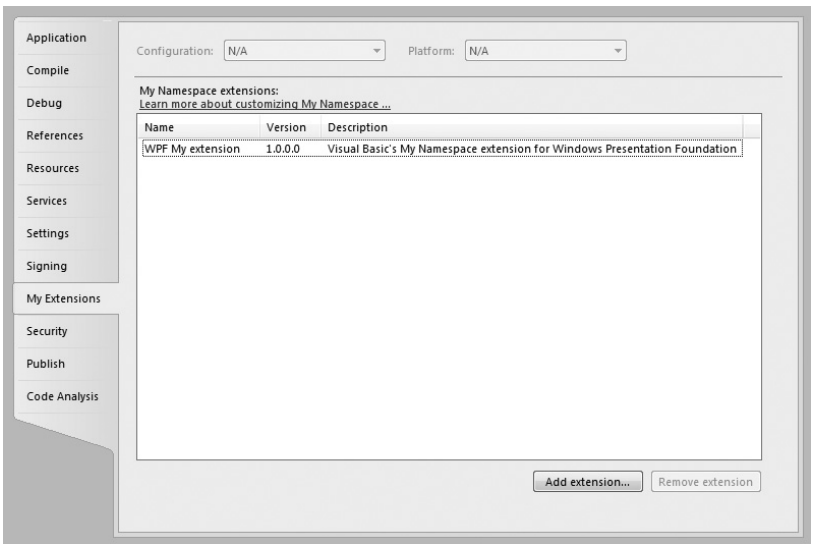


Рис. 6.19

Пространство имен My было изобретено для того, чтобы упростить доступ к общим библиотечным методам. Например, имя `My.Application.Log` открывает доступ к методам для внесения или удаления записи из журнала регистрации с помощью одной-единственной сроки кода. В этом смысле пространство имен My является идеальным для добавления пользовательских классов и методов, обеспечивающих полезные функциональные возможности, информации о глобальном состоянии или конфигурации, а также служб, которые могут использоваться несколькими приложениями.

Закладка Security

Приложения, развернутые по модели ClickOnce, могут быть запущены в режиме ограниченного, или частичного, доверия. Например, если пользователь с низким уровнем привилегий выберет приложение ClickOnce с веб-сайта через Интернет, то это приложение должно быть выполнено в режиме частичного доверия, определенном соответствующей зоной Интернета.

Как правило, это означает, что приложение не может иметь доступа к локальной файловой системе, имеет ограниченные сетевые возможности и не имеет прав доступа к локальным устройствам, таким как принтеры, базы данных и компьютерные порты. Закладка Security, показанная на рис. 6.20, позволяет определить уровень доверия, требуемый для того, чтобы приложение работало корректно.

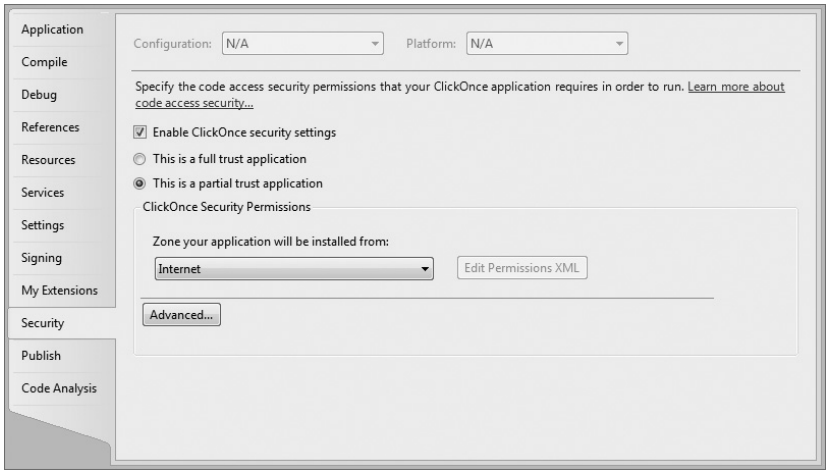


Рис. 6.20

Модифицируя разрешающие настройки, требуемые моделью ClickOnce, можно ограничить круг лиц, имеющих право загружать, инсталлировать с приложением и выполнять с ним какие-либо действия. Если разработчик хочет обеспечить самую широкую аудиторию для своего приложения, то он должен указать, что приложение должно запускаться в режиме частичного доверия с настройками безопасности, заданными по умолчанию, для данной зоны Интернета. В качестве альтернативы можно указать, что запуск приложения требует полного доверия. Это обеспечит приложению доступ ко всем локальным ресурсам, но его аудитория будет неизбежно ограничена только локальными администраторами.

Закладка Publish

Модель развертывания ClickOnce можно разделить на две фазы: первоначальная публикация приложения и последующие обновления, а также загрузка и инсталляция исходного приложения и его последующих версий. С помощью закладки Publish (рис. 6.21), разработчик может развернуть существующее приложение, используя модель ClickOnce.

Если режим инсталляции для приложения ClickOnce открывает доступ к нему в автономном режиме после загрузки с веб-сайта, то приложение будет инсталлировано на локальном компьютере. Имя приложения будет указано в меню Start и списке Add/Remove Programs. Если приложение запущено и доступ к исходному веб-сайту открыт, то приложение определит, доступны ли какие-либо обновления. Если обновления существуют, то пользователей попросят определить, хотят ли они инсталлировать обновления.

Модель развертывания ClickOnce объясняется в главе 48.

Закладка Code Analysis (только для версий VSTS Premium и Ultimate Editions)

Большинство разработчиков, работающих в коллективе, должны следовать согласованным стандартам кодирования. Как правило, организации используют существующие стандарты кодирования или разрабатывают свои собственные. К сожалению, стандарты

полезны, только если программистов можно принудить следовать им, и единственный способ, позволяющий сделать это эффективно, — использование специального инструмента. Прежде эту задачу решали с помощью внешних утилит, таких как FXCop. В версиях Visual Studio 2010 VSTS Premium и Ultimate Editions существует возможность проводить статический анализ кода в интегрированной среде разработки.

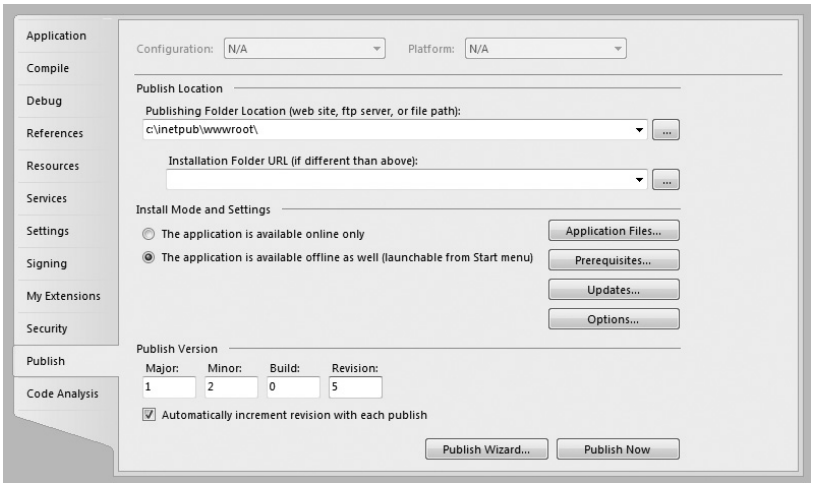


Рис. 6.21

С помощью закладки Code Analysis, показанной на рис. 6.22, анализ кода можно сделать частью процесса сборки. Поскольку этот процесс требует довольно много времени, его можно включать только в конфигурацию окончательного варианта приложения или в тестовую конфигурацию сборки. Независимо от того, включен ли анализ кода в проект, соответствующую команду можно вручную выбрать в меню Build.

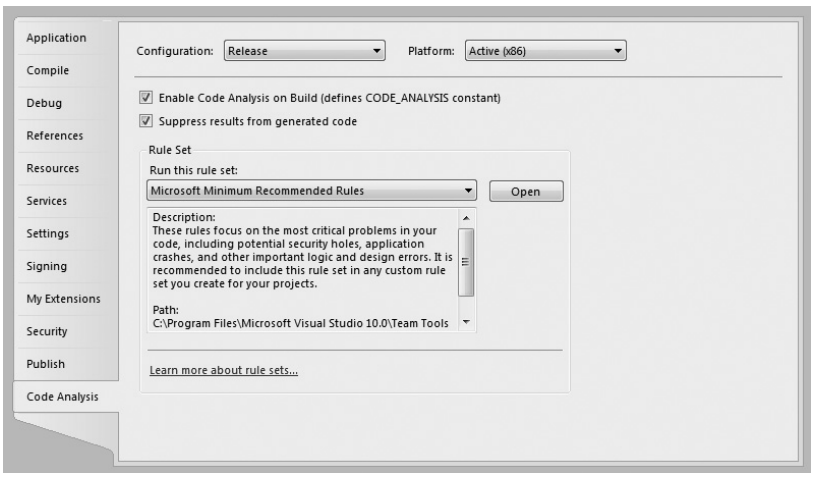


Рис. 6.22

Не все правила, определенные в окне Code Analysis, приемлемы для всех организаций или приложений. Это окно позволяет разработчику самостоятельно выбирать применяемые правила, а также решать, следует ли генерировать предупреждения или

сообщения об ошибке при их нарушении. Отменив выбор правила в столбце Rules, пользователь отменяет само правило. Дважды щелкая на ячейке в столбце Status, пользователь может переключаться между вариантами выбора того, что должно произойти при нарушении правила: вывод предупреждения или сообщения об ошибке.

Утилита FXCop рассмотрена в главе 13, а средства анализа кода, внедренные в систему Visual Studio, более подробно описаны в главе 55.

Свойства проектов Web Application

Из-за уникальных требований, предъявляемых к веб-приложениям, для свойств проектов, имеющих тип Web Application для платформы ASP.NET, в системе Visual Studio 2010 предусмотрены четыре дополнительные закладки. Эти закладки управляют способами запуска веб-приложений из системы Visual Studio, а также вариантами упаковки и развертывания.

Закладка Web

Закладка Web, показанная на рис. 6.23, управляет тем, как запускать из системы Visual Studio проекты, имеющие тип Web Application. Система Visual Studio имеет встроенный веб-сервер, предназначенный для осуществления процессов развертывания. Закладка Web позволяет разработчикам задавать конфигурацию порта и виртуальный путь для веб-сервера. Кроме того, на этой закладке можно включить схему аутентификации NTLM.

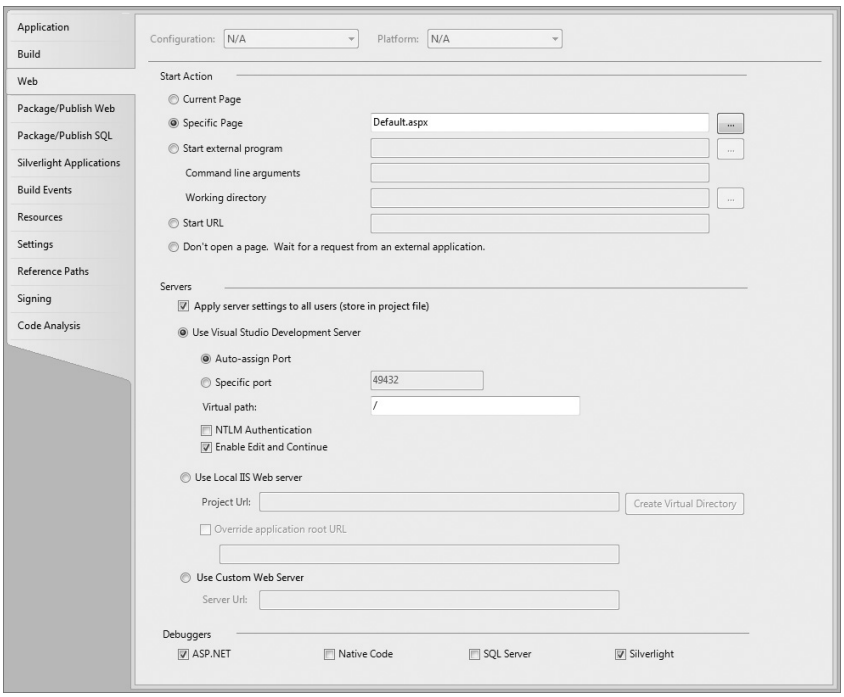


Рис. 6.23



Команды Edit и Continue позволяют редактировать файлы выделенного кода (code-behind class files) и самостоятельные файлы классов в ходе сессии отладки. Редактирование кода HTML на страницах `an.aspx` или `.ascx` допускается независимо от этой установки; тем не менее редактирование встроенного кода на странице `aspx` или в файле `.ascx` не разрешается никогда.

Настройки отладки для веб-приложений описаны в главе 42.

Закладка Silverlight Applications

Закладка Silverlight Applications обеспечивает простой способ установить связь с проектом типа Silverlight и сделать его главным в веб-приложении для платформы ASP.NET. Добавляя приложение типа Silverlight в проект типа Web Application, разработчик может выбрать существующий проект типа Silverlight, если он существует в текущем решении, или создать новый, как показано на рис. 6.24.

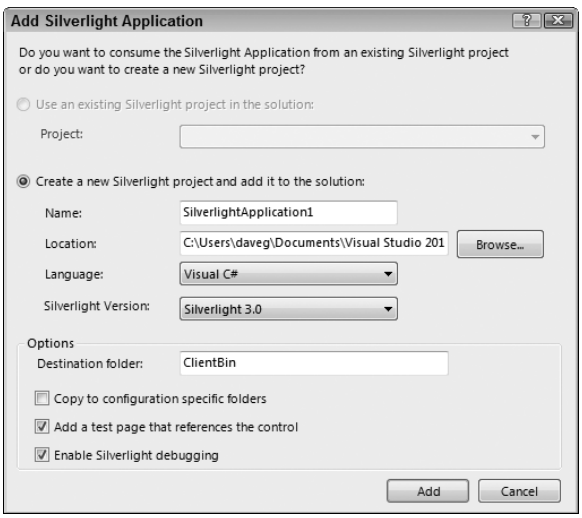


Рис. 6.24

Диалоговое окно позволяет разработчику выбрать место и язык нового проекта, а также настройки, регламентирующие включение приложения типа Silverlight в текущее веб-приложение.

Если при добавлении нового приложения типа Silverlight разработчик выберет настройки, заданные по умолчанию, то система Visual Studio создаст ссылку на новый проект и сгенерирует три файла в веб-приложении: статическую HTML-страницу, веб-форму ASP.NET и файл JavaScript, описывающий логику загрузки приложений типа Silverlight и инсталлирующий по необходимости рабочий модуль.

Разработка приложений типа Silverlight и варианты из внедрения в существующие веб-страницы рассматриваются в главе 22.

Упаковка и публикация веб-приложений

Развертывание приложений всегда ставило разработчиков перед трудным выбором, особенно, если речь шла о сложных веб-приложениях. Типичное веб-приложение содержит не только большое количество исходных файлов и сборок, но и изображения, стилевые схемы и файлы JavaScript files. Для того чтобы еще больше усложнить ситуацию, можно сделать их зависимыми от конкретной конфигурации веб-сервера из набора IIS (Internet Information Server).

Система Visual Studio 2010 упрощает этот процесс, разрешая разработчику упаковывать проект типа Web Application вместе с необходимыми файлами project и настройками в отдельный сжатый файл (`.zip`). На рис. 6.25 показаны настройки упаковки и развертывания, доступные для приложении типа Web Application на платформе ASP.NET.

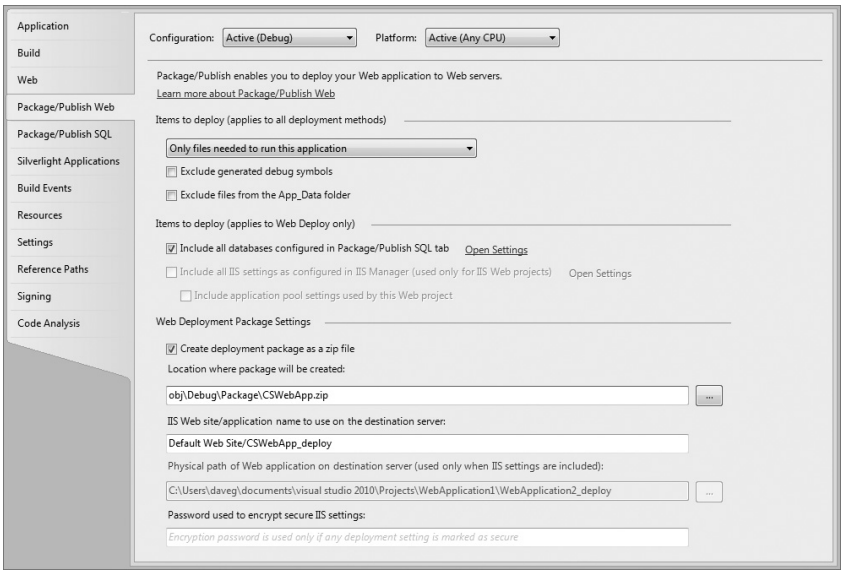


Рис. 6.25

Подробнее процесс развертывания веб-приложений рассмотрен в главе 49.

Упаковка и публикация баз данных SQL

Все веб-приложения, кроме самых простых, используют какую-нибудь базу данных. Веб-приложения для платформы ASP.NET обычно используют базы данных SQL Server. Упаковка веб-приложений и развертывание функциональных возможностей с помощью системы Visual Studio 2010 включает в себя упаковку одной или нескольких баз данных SQL Server. Как показано на рис. 6.26, при создании пакета разработчик может задать строку для связи с исходной базой данных и позволить системе Visual Studio создавать сценарии SQL только для схемы баз данных или и схемы, и данных. Кроме того, разработчик может создать свои собственные сценарии SQL для выполнения до либо после сценария, сгенерированного автоматически.

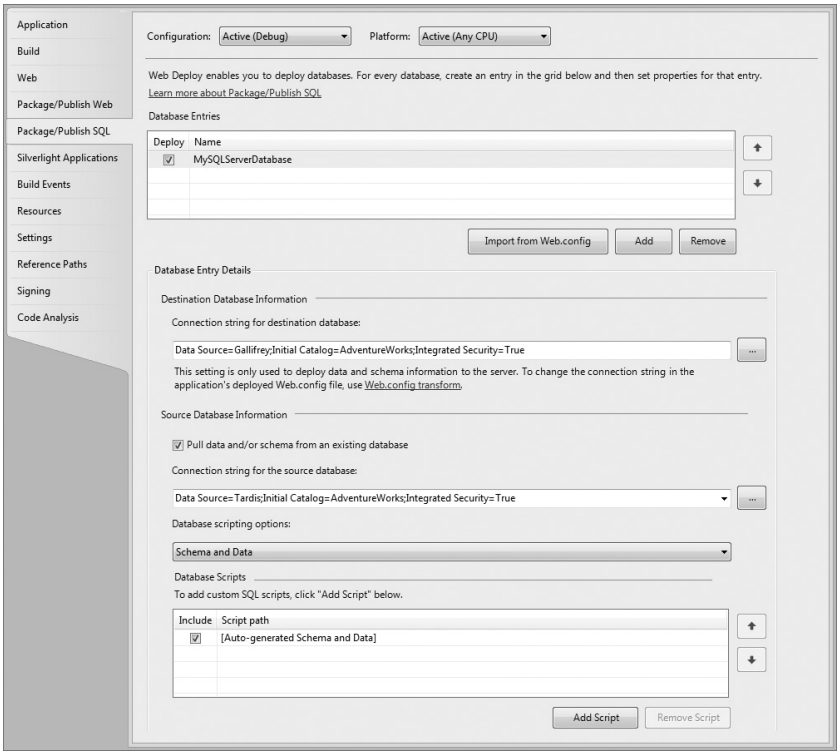


Рис. 6.26

Варианты развертывания веб-приложения подробно исследуются в главе 49.

Проекты типа Web Site

С функциональной точки зрения проекты типа Web Site довольно сильно отличаются от проектов других типов. Проекты типа Web Site не содержат файлов `.csproj` или `.vbproj`, т.е. они имеют множество ограничений, распространяющихся на сборку, ресурсы и управления ссылками. Вместо этого для определения содержания проекта типа Web Site используется структура каталогов. Все файлы в структуре каталогов являются неявной частью проекта.

К преимуществам проектов типа Web Site относится динамическая компиляция, позволяющая редактировать страницы без переделки всего сайта. Файл может быть сохранен и просто повторно загружен в браузер. Это позволяет использовать чрезвычайно короткие программы и циклы отладки. Компания Microsoft впервые внедрила проекты типа Web Site в системе Visual Studio 2005, однако вскоре она получила от пользователей многочисленные просьбы вернуть модель Application Project, которую можно было загрузить дополнительно. К моменту выпуска Service Pack 1 проекты типа Web Application были возвращены в систему Visual Studio в качестве ее собственного типа проекта.



С момента выхода версии Visual Studio 2005 идут непрекращающиеся дебаты о том, что лучше — проекты типа Web Site или типа Web Application. К сожалению, простого ответа не существует. Каждый из них имеет свои преимущества и недостатки, и решение должно быть основано на требованиях разработчика и принятой технологии разработки.

Дальнейшее обсуждение проектов типа Web Site и Web Application продолжено в главе 20.

Резюме

В этой главе речь шла о том, как настроить решение и проект с помощью пользовательского интерфейса, предоставленного системой Visual Studio 2010. В частности, рассматривались следующие вопросы.

- Создание и конфигурирование решений и проектов.
- Управление компиляцией, отладкой и развертыванием приложения.
- Настройка многих свойств, связанных с проектом.
- Включение ресурсов и настроек для приложений.
- Навязывание стандартов кодирования с помощью средств Code Analysis Tools.
- Модификация настроек конфигурации, упаковки и развертывания веб-приложений.

В последующих главах многие из этих тем, например сборка и развертывание проектов, а также использование файлов ресурсов, будут описаны более подробно.

7

Технология IntelliSense и закладки

В ЭТОЙ ГЛАВЕ...

- Повышение эффективности работы с использованием контекстной помощи
- Обнаружение и исправление простых ошибок
- Комбинации клавиш
- Генерирование кода
- Перемещение по исходному коду с помощью закладок

Одним из достижений, которым давно славится компания Microsoft, является автоматическая помощь при редактировании кода. В ранних версиях среды Visual Basic уже была частично реализована технология IntelliSense, но с появлением системы Visual Studio .NET компания Microsoft распространила эту технологию на всю среду интегрированной разработки. В последних выпусках системы технология IntelliSense стала настолько вездесущей, что ее в шутку стали называть IntelliSense Everywhere.

В этой главе описываются многочисленные способы применения технологии IntelliSense, помогающие разработчикам писать программы. К ним относятся обнаружение и исправление синтаксических ошибок, использование контекста и автоматическое дополнение имен переменных. В главе также объясняется, как установить и использовать закладки для более удобного перемещения по тексту программы.

Технология IntelliSense

IntelliSense — это общее название автоматизированной справочной системы и технологии, лежащей в основе приложений, созданных с помощью инструментов компании Microsoft. Наиболее широко известным аспектом технологии IntelliSense

является подчеркивание волнистой линией слов с орфографическими ошибками в текстовом редакторе Microsoft Word, а также небольшие визуальные индикаторы в листах электронной таблицы Microsoft Excel, информирующие пользователя о том, что содержание конкретной ячейки не соответствует ожидаемому.

Даже эти простые индикаторы позволяют быстро выполнять соответствующие действия. Например, если щелкнуть правой кнопкой на слове, подчеркнутом красной волнистой линией в текстовом редакторе Word, отобразится список предлагаемых альтернатив. Другие приложения имеют аналогичные возможности.

В системе Visual Studio уже довольно давно были реализованы аналогичные возможности. Фактически простейшие функции технологии IntelliSense были реализованы уже в среде разработки Visual Basic 6. С каждым новым выпуском системы Visual Studio компания Microsoft совершенствовала технологию IntelliSense, делая ее все более контекстно-чувствительной и внедряя ее более широко, чтобы разработчик всегда мог легко получить необходимую информацию.

В системе Visual Studio 2010 название IntelliSense относится к большому количеству функций, начиная с визуальной реакции на неправильный код и интеллектуальных дескрипторов для проектирования форм до комбинаций клавиш, нажатие которых приводит к вставке целых фрагментов кода. Эти свойства, вместе взятые, обеспечивают разработчику более глубокое понимание происходящего, более высокую эффективность его работы и позволяют ему контролировать свою программу. Некоторые из этих возможностей для системы Visual Studio 2010 являются новыми. К ним относятся, например, *режим подсказок* (suggestion mode) и генерация кода в зависимости от его использования (Generate From Usage). Эти возможности предназначены для поддержки альтернативного стиля разработки, получившего название *разработка через тестирование* (test-driven development — TDD).

Основы технологии IntelliSense

Простейшим свойством технологии IntelliSense является немедленная реакция на неправильный код в листинге программы. На рис. 7.1 приведен один из примеров, в котором для инициализации объекта используются данные неизвестного типа. Поскольку тип этих данных неизвестен в том месте, где введен код, система Visual Studio подчеркивает строку красной (для языка C#) или синей (для языка VB) волнистой линией, чтобы обозначить проблему.



Изменить цвет линии подчеркивания можно с помощью группы переключателей *Fonts and Colors* в окне *Options*.

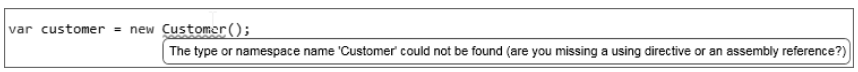


Рис. 7.1

Оставив курсор мыши над неправильным фрагментом кода, разработчик увидит подсказку, объясняющую проблему. В данном примере курсор находится над типом данных, поэтому разработчик увидит подсказку “The type or namespace name ‘ Customer’ could not be found ”.

Система Visual Studio имеет возможность обнаруживать такие ошибки, потому что она непрерывно осуществляет предварительную компиляцию кода, который вводится разра-

ботчиком, и ищет все, что может привести к ошибкам компиляции. Если бы разработчик добавил класс `Customer` в свой проект, то система Visual Studio автоматически обработала бы это действие и удалила бы маркер, поставленный средствами IntelliSense.

На рис. 7.2 показан интеллектуальный дескриптор, ассоциированный с этой ошибкой. Он применяется только к ошибкам, для которых система Visual Studio 2010 может предложить конструктивный способ исправления. Непосредственно под неправильным фрагментом кода на экране появляется небольшой синий (для языка C#) или красный (для языка VB) маркер в виде прямоугольника. Поместив курсор мыши на этот маркер, разработчик увидит меню команд интеллектуального дескриптора для ошибок данного типа. В данном случае это меню содержит команды для генерации пользовательского класса `Customer`, с помощью которого система Visual Studio может определить способ его использования.

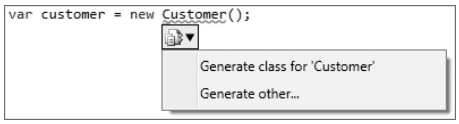


Рис. 7.2

Для вызова интеллектуального дескриптора IntelliSense во всех приложениях компании Microsoft используется комбинация клавиш <Shift+Alt+F10>, но в системе Visual Studio 2010 предложена более удобная комбинация <Ctrl+.>.

Технология интеллектуальных дескрипторов, применяемая в системе Visual Studio, предназначена не только для редактирования кода. По существу, система Visual Studio 2010 позволяет применять интеллектуальные дескрипторы и к визуальным элементам управления, когда разработчик редактирует форму или элемент управления в окне Design view (рис. 7.3).

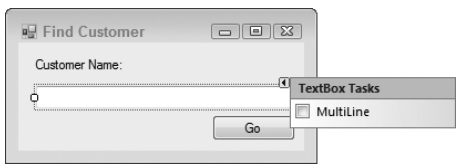


Рис. 7.3

Когда разработчик выбирает команду, имеющую интеллектуальный дескриптор, в правом верхнем углу элемента управления появляется маленький треугольник. Для того чтобы открыть список Tasks для интеллектуального дескриптора, необходимо щелкнуть на этой кнопке. На рис. 7.3 показан список Tasks для стандартного элемента управления TextBox.

Комбинации клавиш, вызывающие интеллектуальный дескриптор, применяются и для визуальных элементов управления.

Дополнение слова и фраз

Мощь технологии IntelliSense в системе Visual Studio 2010 становится очевидной сразу, как только пользователь начинает набирать код. По мере набора текста на экране открываются разнообразные списки, помогающие выбрать правильные типы членов класса, функций и их параметров, тем самым уменьшая количество потенциальных ошибок компиляции еще до того, как разработчик закончит набирать текст программы. Ознакомившись с функционированием инструментов IntelliSense, легко заметить, что существует возможность значительно сократить объем кода, который требуется набрать на самом деле. Это особенно важно для пользователей, работающих на таких многословных языках, как Visual Basic.

Контекст

В системе Visual Studio 2010 технология IntelliSense проявляет себя сразу. как только программист начинает набирать текст в окне редактора кода. На рис. 7.4 показано, как работает технология IntelliSense в ходе создания цикла `For` в программе на языке Visual Basic. Как только программист наберет букву `f`, в левой части изображения откроется список доступных слов, начинающихся с буквы `f`, который постепенно сокращается по мере набора остальных букв. Легко заметить, что этот список содержит все варианты — инструкции, классы, методы и свойства, — содержащие введенные буквы (в данном случае для всех имен, содержащих слово `For`).

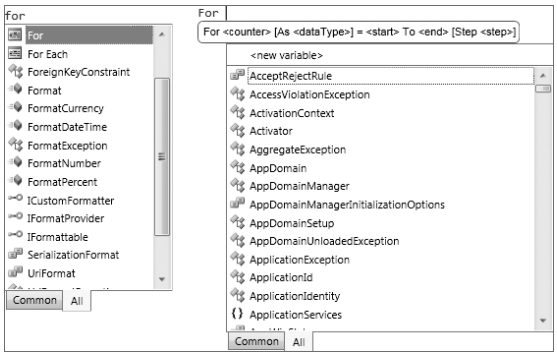


Рис. 7.4

Обратите внимание на правый список на рис. 7.4, где после слова `for` вставлен пробел. Теперь открывшийся список IntelliSense содержит все варианты, которые можно вставить в данную позицию в коде. Кроме того, на экране появляется подсказка, напоминающая о синтаксисе инструкции `For`. И наконец, прямо над списком IntelliSense отображается элемент `<new variable>`. Он свидетельствует о том, что в этом месте разработчик может объявить новую переменную.



Элемент `<new variable>` появляется только при редактировании программ на языке Visual Basic.

Несмотря на полезность списка IntelliSense, учитывающего вводимые программистом буквы, этот инструмент является “обоюдоострым”. Довольно часто разработчикам приходится искать переменную или член класса, не зная их полное имя. В этом случае можно ввести первые несколько букв наугад, а затем просмотреть список в поисках правильного варианта. Очевидно, что этот прием не сработает, если введенные буквы уже исключили правильный вариант. Для того чтобы увидеть полный список вариантов, нажмите клавишу <Backspace>, пока список IntelliSense открыт.

В предыдущих версиях системы Visual Studio технология IntelliSense позволяла находить лишь члены классов, имена которых начинались с символов, набранных программистом в окне редактора кода. В системе Visual Studio 2010 это не так. Теперь можно найти слова, которые появляются и в середине имен членов класса. Для этого система осуществляет поиск границ слова в именах членов классов. На рис. 7.5 приведен пример на язы-

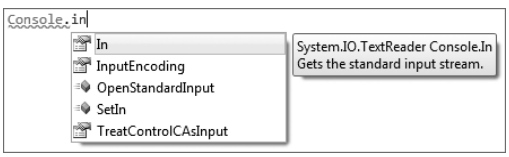


Рис. 7.5

ке C#, в котором, набрав слово `Console.in`, пользователь получает варианты `In`, `InputEncoding`, `OpenStandardInput`, `SetIn` и `TreatControlCAsInput`, но не `LargestWindowHeight`, несмотря на то, что это слово содержит подстроку “in”.



Если вы точно знаете, что ищете, введите первую букву каждого слова в верхнем регистре. Например, если набрать `System.Console.OSI`, то подсистема IntelliSense найдет слово `OpenStandardInput`.

Если информация, выдаваемая средствами IntelliSense, мешает увидеть другие строки программы или пользователь хочет просто скрыть список, достаточно нажать клавишу <Esc>. В качестве альтернативы, если требуется лишь увидеть строки программы, скрытые за списком IntelliSense, не закрывая его полностью, можно нажать и удерживать клавишу <Ctrl>. В этом случае список IntelliSense станет полупрозрачным и позволит увидеть скрытый под ним код (рис. 7.6).



Рис. 7.6

Список IntelliSense предназначен не только для удовлетворения информационных потребностей. Разработчик может выбрать элемент этого списка, и система Visual Studio вставит его полный текст в окно редактора кода. Выбрать элемент в списке можно разными способами. Например, можно дважды щелкнуть мышью на требуемом элементе; можно также использовать клавиши с изображениями стрелок для выбора разных элементов, а затем нажать клавишу <Enter> или <Tab>, чтобы вставить в программу текст; и наконец, когда элемент в списке подсвечен, он автоматически будет выбран, если пользователь введет *символ фиксации* (commit character). Символами фиксации называют символы, которые не допускаются в именах членов классов. Примерами таких символов являются круглые и фигурные скобки, математические символы и двоеточие.

Список членов классов

Поскольку технология IntelliSense разрабатывается уже давно, большинство разработчиков знакомы со списком членов классов, который появляется при наборе имен, завершающихся точкой. Это значит, что программист собирается сослаться на член класса и система Visual Studio автоматически выводит список членов классов, доступных для данного объекта (рис. 7.7). При первом открытии списка членов класса, доступных для данного объекта, система Visual Studio выводит список в алфавитном порядке, оставляя видимой его верхнюю часть. Однако, если этот список уже использовался ранее, система подсвечивает последний член, к которому обращался разработчик, чтобы ускорить выполнение повторяющихся действий.

На рис. 7.7 продемонстрирован также еще один полезный аспект списка членов классов для программистов, работающих на языке Visual Basic. Корешки закладок Common и All (в нижней части списка членов класса) позволяют увидеть либо совместно используемые члены классов, либо их полный список.



Фильтровать список свойств, методов и событий по свойствам доступа можно только для языка Visual Basic.

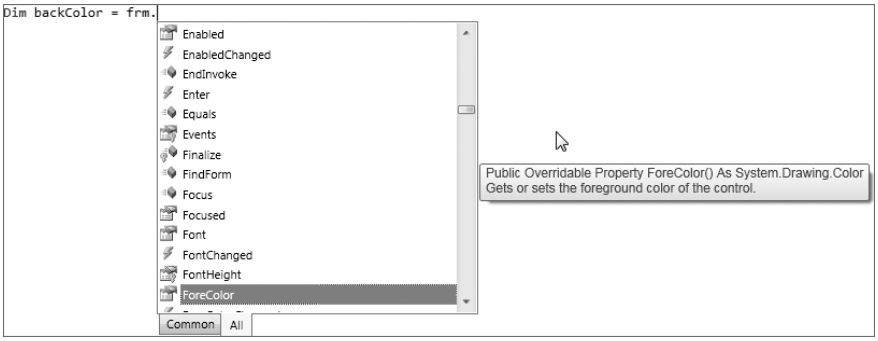


Рис. 7.7

Режим подсказок

По умолчанию, когда система Visual Studio 2010 выводит на экран список членов IntelliSense, можно выбрать только один элемент списка. По мере ввода последующих символов подсветка перемещается на элемент, который лучше остальных соответствует набранным символам. Если нажать клавиши <Enter> или <Space> или набрать один из символов фиксации (например, открывающую скобку), то выделенный в данный момент элемент списка будет вставлен в окно редактора кода. Это поведение, предусмотренное по умолчанию, называется *режимом дополнения* (completion mode).

Как правило, режим дополнения обеспечивает ожидаемое функционирование системы и позволяет сэкономить много усилий, связанных с набором текста, но в некоторых случаях могут возникнуть проблемы. Один из них возникает при разработке через тестирование, когда пользователь часто ссылается на члены классов, которые еще не были определены. Это вынуждает систему IntelliSense выбирать члены классов, не устраивающие программиста, и вставлять нежелательный текст.

Для того чтобы этого избежать, компания Microsoft в интегрированной среде разработки Visual Studio 2010 предложила новый режим функционирования системы IntelliSense, который называется *режимом подсказок* (suggestion mode). Когда система IntelliSense работает в режиме подсказок, один из членов классов в списке находится в фокусе, но не выбирается по умолчанию. По мере набора текста система IntelliSense перемещает фокус на элемент, наиболее точно соответствующий набранным символам, но не вставляет его автоматически. Вместо этого набранные символы вставляются в начало списка IntelliSense, и если пользователь введет один из символов фиксации или нажмет клавишу <Space> или <Enter>, то в окно редактора будет вставлена точно набранная строка.



Элементы списка IntelliSense можно по-прежнему выбирать с помощью навигационных клавиш или клавиши <Tab>.



Рис. 7.8

На рис. 7.8 приведен пример задачи, для решения которой предназначен режим подсказок. В левой части рисунка программист набирает тест для нового метода `Load` из класса `CustomerData`. Класс `CustomerData` еще не содержит метода `Load`, но в нем уже есть класс `LoadAll` .

На рис. 7.8, *справа*, показано, что программист набрал имя `Load`, за которым следует открывающая круглая скобка. Система IntelliSense неправильно предположила, что разработчик подразумевал метод `LoadAll`, и вставила его в окно редактора.

Для того чтобы избежать этого, можно включить режим подсказок, нажав клавиши <Ctrl+Alt+Space>. Теперь, когда программист наберет имя `Load`, оно появится в начале списка IntelliSense. Набрав открывающую круглую скобку, разработчик получит метод Load, что и требовалось (рис. 7.9).

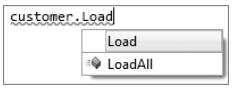


Рис. 7.9



Система IntelliSense останется в режиме подсказок, пока пользователь не нажмет комбинацию клавиш <Ctrl+Alt+Space>, тем самым включив режим дополнения.

Stub Completion

Кроме автоматического дополнения слов и фраз, система IntelliSense может также работать в режиме заглушки. Это свойство можно увидеть в среде разработки Visual Basic, когда программист создает функцию, вставляя ее объявление и нажимая клавишу <Enter>. Среда Visual Studio автоматически переформатирует строку, добавляя соответствующее ключевое слово ByVal для параметров, которые не были явно определены своим контекстом, и строку End Function, завершающую код функции. Другой пример можно увидеть при редактировании XML-документа. Когда программист набирает *открывающий дескриптор* (open tag) нового элемента, система Visual Studio автоматически добавляет закрывающий дескриптор для него.

Режим заглушки в системе Visual Studio 2010 представляет собой новый этап, позволяя программистам делать то же самое при перегрузке интерфейсов и методов. Когда разработчик добавляет определенные конструкции кода, такие как интерфейс или определение класса, в языке C#, система Visual Studio дает ему возможность автоматически генерировать код, необходимый для реализации данного интерфейса. Для того чтобы продемонстрировать, как это работает, рассмотрим этапы, которые проходит система IntelliSense, генерируя реализацию интерфейса для простого класса.

1. Запускаем систему Visual Studio 2010 и создаем проект типа C# Windows Forms Application. Когда интегрированная среда разработки сгенерирует первоначальный код, открываем файл `Form1.cs` в окне редактора кода.
2. В начало файла добавляем директиву `using`, чтобы открыть доступ к пространству имен `System.Collections`.
   ```
   using System.Collections;
   ```
3. Добавляем в начало определения класса строку
   ```
   public class MyCollection: IEnumerable
   ```
   Поскольку мы набираем интерфейс IEnumerable, система Visual Studio сначала добавляет красную волнистую линию в конце фрагмента, чтобы показать, что в определении класса нет фигурных скобок, а затем добавляет индикатор интеллектуального дескриптора перед именем интерфейса (рис. 7.10).



Рис. 7.10

4. Устанавливаем курсор мыши на индикатор интеллектуального дескриптора. Когда появится пиктограмма, щелкаем на ней, чтобы открыть меню доступных команд, как показано на рис. 7.11.

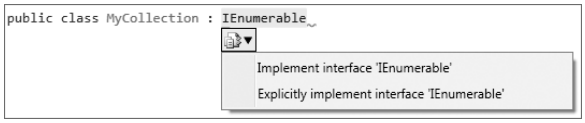


Рис. 7.11

5. Выбираем команду Implement interface 'IEnumerable', и система Visual Studio 2010 автоматически генерирует оставшуюся часть кода, необходимую для завершения минимального определения интерфейса. Поскольку система видит, что определение класса является неполным, она одновременно добавляет фигурные скобки, чтобы исправить эту ошибку.

Интерфейс, который получится в результате, показан на рис. 7.12.

```
public class MyCollection : IEnumerable
{
    IEnumerator IEnumerable.GetEnumerator()
    {
        throw new NotImplementedException();
    }
}
```

Рис. 7.12

Обработчики событий также можно автоматически сгенерировать с помощью системы Visual Studio 2010. Интегрированная среда разработки генерирует их точно так же, как и реализацию интерфейсов. Когда программист набирает первую часть инструкции (например, `myBase.OnClick+=`), система Visual Studio выполняет автоматическое дополнение, которое можно подтвердить, нажав клавишу <Tab>.

Свойство Generate From Usage

Вместо генерирования кода в соответствии с существующим определением иногда удобнее генерировать определение кода элемента в зависимости от способа, которым он используется. Это особенно удобно при разработке через тестирование, когда разработчик пишет тесты для классов, которые еще не определены. В этом случае желательно иметь возможность генерировать классы по самим тестам. Для этой цели в языки C# и Visual Basic было включено новое свойство — Generate From Usage.

Для того чтобы понять, как это свойство можно использовать на практике, рассмотрим этапы создания очень простого класса `Customer` с помощью некоего клиентского кода, использующего этот класс, а затем генерирующего его в зависимости от способа использования.

1. Запускаем систему Visual Studio 2010 и создаем проект типа C# Command Line. Когда интегрированная система разработки будет готова к работе, открываем файл `Program.cs`.
2. Включаем в метод `Main` следующий код.

C#

```
Customer c = new Customer
{
    FirstName = "Joe",
    LastName = "Smith"
};

Console.WriteLine(c.FullName);
c.Save();
```

3. Оба раза, когда имя `Customer` встречается в тексте программы, оно подчеркивается красной волнистой линией. Щелкаем правой кнопкой на одном из них и выбираем команду Generate⇨Class. Она создает новый класс `Customer` в нашем проекте. Открываем файл `Customer.cs` и видим пустое объявление

класса. Система Visual Studio выяснит, что `FirstName`, `LastName`, `FullName` и `Save` не являются членами класса.

4. Для каждого не существующего в классе свойства щелкаем правой кнопкой на его имени и выбираем команду Generate⇨Property. Снова открываем файл `Customer.cs` и видим, что система Visual Studio способна сама создать его реализацию.
5. То же самое можно повторить для метода `Save`, щелкнув на его имени правой кнопкой и выбрав команду Generate⇨Method Stub.



Несмотря на то что сгенерированные свойства и классы можно использовать сразу, заглушки методов создаются так, чтобы они генерировали исключение NotImplementedException.

Если неопределенный код, который разработчик пытается сгенерировать, является типом, то можно выбрать команду Generate Class или Generate Other. Если выбрать команду Generate Other, то откроется диалоговое окно Generate New Type (рис. 7.13). Оно предоставляет пользователю много возможностей для конфигурации нового типа, включая выбор его разновидности: класс, перечисление, интерфейс или структура; уровень доступа: *открытый* (public), *закрытый* (private) или *внутренний* (internal), а также его местонахождение.

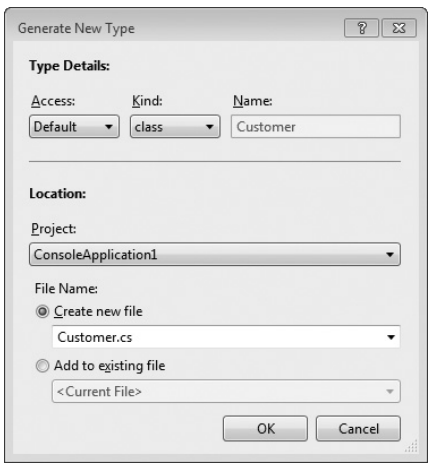


Рис. 7.13

Информация о параметрах

В старых версиях инструментов разработки программ, выпущенных компанией Microsoft, например в системе Visual Basic 6, как только пользователь создавал вызов функции, при наборе параметров система IntelliSense сразу выводила на экран информацию о них. К счастью, это невероятно полезное свойство по-прежнему доступно в системе Visual Studio 2010.

Старый способ представления информации о параметрах имел один недостаток: информация появлялась только тогда, когда разработчик действительно модифицировал вызов функции. Следовательно, он мог увидеть полезную подсказку только при создании вызова функции или при его изменении, но не при обычном просмотре кода.

В результате разработчики иногда непреднамеренно вносили ошибки в свои программы, поскольку они умышленно модифицировали вызовы функций так, чтобы увидеть информацию о ее параметрах, а потом забывали их исправить.

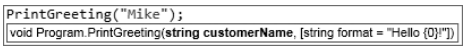


Рис. 7.14

Система Visual Studio 2010 исключила этот риск, предоставив пользователям легко доступную команду, которая выводит информацию на экран, не требуя модификации кода. Нажав комбинацию клавиш <Ctrl+Shift+Space>, можно вывести на экран информацию о вызове функции, как показано на рис. 7.14. Кроме того, можно получить доступ к этой информации, выбрав команду Edit⇨IntelliSense⇨Parameter Info.



На рис. 7.14 метод `PrintGreeting` имеет два параметра. Второй параметр является необязательным и выводится в квадратных скобках с указанием его значения по умолчанию. Программисты, работающие на языке VB, уже знакомы с этой синтаксической конструкцией, но для языка C# 4.0 она является новой.

Команда Quick Info

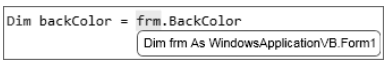


Рис. 7.15

Иногда программист хочет видеть информацию об объекте или интерфейсе, не модифицируя его код. Нажав комбинацию клавиш <Ctrl+K>, <Ctrl+I>, можно увидеть короткую подсказку, объясняющую предназначение объекта и способ его объявления (рис. 7.15).



Комбинации клавиш в системе VS2010 зависят от выбранных параметров инсталляции (например, Visual Basic Developer, Visual C# Developer и т.д.). Все комбинации клавиш в этой главе соответствуют настройкам профиля General Developer Profile.

Эту подсказку можно увидеть также, выбрав команду Edit⇨IntelliSense⇨Quick.

Технология Intellisense для языка JavaScript

При разработке веб-приложений язык JavaScript позволяет обеспечить более широкие возможности для их клиентов. В отличие от языков C# и Visual Basic, которые являются компилируемыми, язык JavaScript является интерпретируемым, т.е. его синтаксические конструкции не проверяются, пока он не будет загружен в браузер. Несмотря на то что этот язык дает много гибких возможностей на этапе выполнения, он требует внимательности, опыта и глубокого тестирования, иначе неизбежны многочисленные ошибки.

Кроме того, при разработке компонентов на языке JavaScript для использования в браузере необходимо следить за большим количеством несопоставимых элементов. К ним относятся свойства самого языка JavaScript, элементы HTML DOM, а также собственные и внешние библиотеки. К счастью, система Visual Studio 2010 способна предоставить все достижения технологии IntelliSense для полноценного использова-

ния языка JavaScript, чтобы следить за всеми этими элементами и предупреждать разработчика о синтаксических ошибках.

Как только разработчик вводит в окно редактора кода текст на языке JavaScript, система Visual Studio перечисляет ключевые слова, функции, параметры, переменные, объекты и свойства так же, как она это делает для программ на языке C# или Visual Basic. Это относится как к встроенным функциям и объектам языка JavaScript, так и к функциям и объектам, определенным разработчиком в своем собственном сценарии или включенным во внешнюю библиотеку. Система Visual Studio позволяет также выявить и выделить синтаксические ошибки в программе на языке JavaScript.



Начиная с версии Internet Explorer 3.0 компания Microsoft поддерживает свой собственный вариант языка JavaScript, именуемый JScript. С формальной точки зрения средства JavaScript в системе Visual Studio 2010 разработаны для работы с диалектом Jscript, поэтому в командах меню и заголовках окон иногда встречается это название. На практике разница между этими двумя языками настолько мала, что программные средства одинаково успешно работают с любым из них.

Контекст JavaScript IntelliSense

Для того чтобы предотвратить случайные ссылки на несуществующие элементы в языке JavaScript, система Visual Studio 2010 использует понятие "контекст IntelliSense", основываясь на местоположении редактируемого блока конструкций на языке JavaScript. Этот контекст создается следующими элементами.

- Текущий блок сценария. Включает в себя блоки сценариев для файлов `.aspx`, `.ascx`, `.master`, `.html` и `.htm`.
- Любой файл сценариев, импортированный на текущую страницу с помощью элемента `<script/>` или элемента управления ScriptManager. В этом случае импортируемый файл должен иметь расширение `.js`.
- Любой файл сценариев, на который есть ссылка в виде директивы ссылок (см. раздел "Ссылка на еще один файл JavaScript " в этой главе).
- Любая ссылка на веб-службы XML.
- Элементы библиотеки AJAX компании Microsoft (если требуется создать веб-приложение для платформы ASP.NET с возможностями библиотеки AJAX-enabled).



Система Visual Studio отслеживает файлы по контексту и обновляет средства JavaScript IntelliSense, как только один из них был изменен. Иногда такое обновление может оказаться незавершенным, и данные JavaScript IntelliSense могут оказаться устаревшими. Разработчик может заставить систему Visual Studio обновить данные JavaScript IntelliSense, выполнив команду Edit⇨IntelliSense⇨Update JScript IntelliSense.

Иногда происходят ошибки, и система Visual Studio оказывается неспособной создать контекст JavaScript IntelliSense. Тем не менее часто система Visual Studio имеет возможность выяснить причину ошибки и послать разработчику сообщение о том,

как ее можно исправить. На рис. 7.16 показано, как система Visual Studio обнаружила, что разработчик сослался на несуществующий файл JavaScript. Если добавить этот файл в проект, то система Visual Studio обнаружит его и удалит индикатор ошибки и сообщение о ней. Несмотря на то что поиск ошибок обычно осуществляется в фоновом режиме, пользователь может заставить систему Visual Studio проверить страницу, выбрав команду Edit⇨Advanced⇨Validate Document.

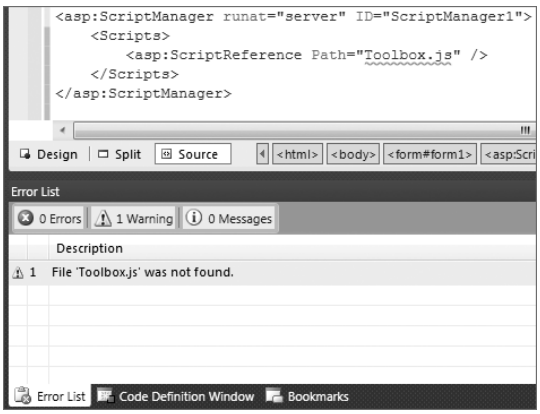


Рис. 7.16

Ссылка на еще один файл JavaScript

Иногда один файл JavaScript создается на основе функциональных возможностей другого. Когда это происходит, эти файлы обычно ссылаются друг на друга на какой-то странице, используя друг друга, но не имеют явно определенных прямых ссылок. Из-за отсутствия явных ссылок система Visual Studio 2010 не может добавить файл с базовыми функциональными возможностями в контекст JavaScript IntelliSense, и в таком случае разработчик не получает полной поддержки со стороны системы IntelliSense.

Для того чтобы система Visual Studio могла найти базовый файл и добавить его в контекст, разработчик может вставить ссылку, используя *директиву ссылок* (references directive). Директива ссылок — это особая разновидность комментария, содержащего информацию о местоположении другого файла. С помощью директивы ссылок можно сослаться на любой из перечисленных ниже объектов.

- **Другие файлы языка JavaScript.** К ним относятся файлы с расширением `.js` и файлы языка JavaScript, внедренные в сборки. Они не содержат абсолютных путей, поэтому файл, на который имеется ссылка, должен принадлежать текущему проекту.
- **Файлы веб-службы (.asmx).** Они также должны быть частью текущего проекта, причем файлы веб-службы в проектах типа Web Application не поддерживаются.
- **Страницы, содержащие сценарии JavaScript.** Так можно сослаться только на одну страницу. Наличие ссылки на любую страницу исключает возможность других ссылок.

Рассмотрим несколько примеров директив ссылок. Они должны появляться до любого кода в файле на языке JavaScript.

JavaScript

```
// Файл JavaScript в текущей папке
/// <reference path="Toolbox.js" />

// Файл JavaScript в родительской папке
/// <reference path="../Toolbox.js" />

// Файл JavaScript на пути, заданном относительно корневой папки сайта
/// <reference path="~/Scripts/Toolbox.js" />

// Файл JavaScript, внедренный в сборку
/// <reference name="Ajax.js" path="System.Web.Extensions, … " />

// Файл веб-службы
/// <reference path="MyService.asmx" />

// Стандартная страница
/// <reference path="Default.aspx" />
```



Существует несколько ограничений, наложенных на директивы ссылок. Во-первых, директивы ссылок, ссылающиеся на путь, внешний по отношению к проекту, игнорируются. Во-вторых, директивы ссылок не могут вычисляться рекурсивно, поэтому для создания контекста используются только те директивы, которые существуют в редактируемом файле. Директивы ссылок в других файлах в контексте не используются.

Настройки системы IntelliSense

Система Visual Studio 2010 устанавливает много настроек средств IntelliSense по умолчанию, но многие из них можно изменить с помощью диалогового окна Options, если их значения, заданные по умолчанию, не устраивают разработчика. Некоторые из этих настроек связаны только с определенным языком программирования.

Общие настройки

Сначала рассмотрим настройки из группы Keyboard в разделе Environment. Каждой команде, доступной в системе Visual Studio, соответствует вполне определенная запись в списке комбинаций клавиш (см. диалоговое окно Options на рис. 7.17, которое открывается по команде Tools⇨Options).

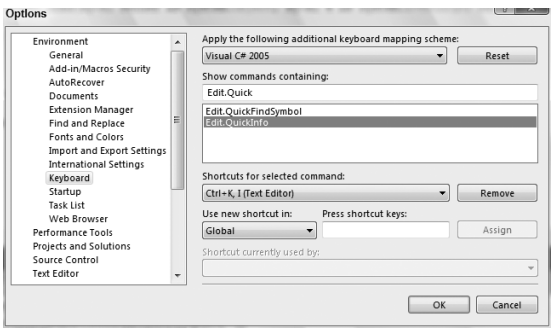


Рис. 7.17

Комбинации клавиш, заданные по умолчанию, можно заменить или добавить новые. Комбинации клавиш для команд системы IntelliSense приведены в табл. 7.1.

Таблица 7.1. Комбинации клавиш, заданные по умолчанию, для команд системы IntelliSense

Edit.QuickInfo	<Ctrl+K>, <Ctrl+I>	Выводит на экран информацию команды Quick Info о выбранном элементе
Edit.CompleteWord	<Ctrl+Space>	Пытается дополнить слово, если существует однозначное соответствие, или выводит список, если есть несколько совпадений
Edit.ToggleConsumeFirst CompletionMode	<Ctrl+Alt+Space>	Переключает систему IntelliSense из режима подсказки в режим дополнения
Edit.ParameterInfo	<Ctrl+Shift+Space>	Выводит на экран информацию о списке параметров вызываемой функции
Edit.InsertSnippet	<Ctrl+K>, <Ctrl+X>	Открывает диалоговое окно Code Snippet, в котором можно выбрать фрагмент кода, чтобы вставить его автоматически (подробнее об этом – в следующей главе)
Edit.GenerateMethodStub	<Ctrl+K>,<Ctrl+M>	Генерирует полную заглушку метода по шаблону
Edit.ImplementAbstract ClassStubs	None	Генерирует определения абстрактного класса из заглушки
Edit.ImplementInterfaceStubs Explicitly	None	Генерирует явную реализацию интерфейса для определения класса
Edit.ImplementInterfaceStubs Implicitly	None	Генерирует неявную реализацию интерфейса для определения класса

Используя способы, описанные в главе 3, можно добавить новые комбинации клавиш для любой команды.

Дополнение инструкции

Работой системы IntelliSense можно управлять либо в глобальном масштабе (рис. 7.18), либо для каждого языка программирования в отдельности. Для того чтобы определить способ вывода списка членов классов или вообще отменить его вывод, на закладке General в группе All Languages диалогового окна Options можно изменить настройки Statement Completion.

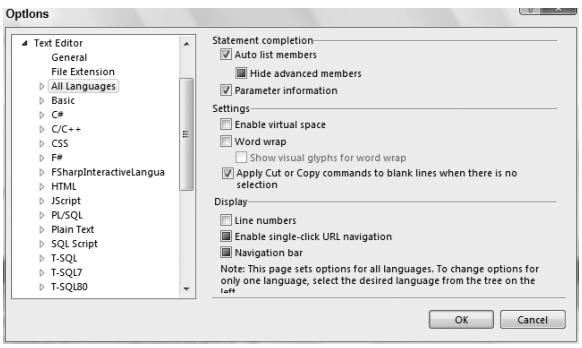


Рис. 7.18

Настройки для языка C#

Кроме общих настроек интегрированной среды разработки и языков программирования для системы IntelliSense, некоторые языки, например C#, имеют дополнительную закладку для определения собственных настроек. С помощью закладки настроек для языка C#, показанной на рис. 7.19, можно также задать способ вызова и использования средств технологии IntelliSense.

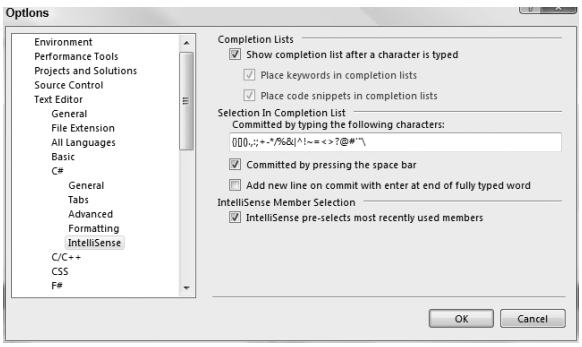


Рис. 7.19

Во-первых, можно отключить список дополнений, чтобы они не появлялись автоматически. Некоторые разработчики предпочитают делать именно так, потому что списки членов классов мешают им набирать тексты своих программ. Если список дополнений выводится не автоматически, а только по вызову, то пользователь может сам решить, что включать в эти списки, помимо стандартных записей, включая ключевые слова и фрагменты кода.

Для того чтобы выбрать запись в списке членов классов, можно использовать любой из символов, доступных в разделе Selection In Completion List, или нажать клавишу пробела. В заключение, как указывалось ранее, система Visual Studio автоматически выделит тот член класса в списке, который использовался последним. В языке C# можно либо отключить это свойство, либо очистить историю.

Дополнительные возможности системы IntelliSense

Кроме основных аспектов технологии IntelliSense, в системе Visual Studio 2010 реализованы также дополнительные функциональные возможности интегрированной среды разработки, которые тоже относятся к этой технологии. Эти свойства подробно обсуждаются в другой главе, а в этом разделе приводится лишь их краткое описание.

Сниппеты

Сниппеты (code snippets) — это фрагменты кода, которые можно автоматически генерировать и вставлять в свой собственный код вместе с соответствующими ссылками и директивами `using`, помечая переменные фразы, чтобы облегчить их замену.

Для того чтобы открыть диалоговое окно Code Snippets, следует нажать клавиши <Ctrl+K>, <Ctrl+X>.

В поисках требуемого сниппета можно перемещаться по иерархии каталогов сниппетов (рис. 7.20). Если разработчик знает комбинацию клавиш, установленную для конкретного сниппета, он может просто нажать ее и клавишу <Tab>, и система Visual Studio вставит сниппет, не открывая диалоговое окно. Мощь сниппетов будет продемонстрирована в главе 8.

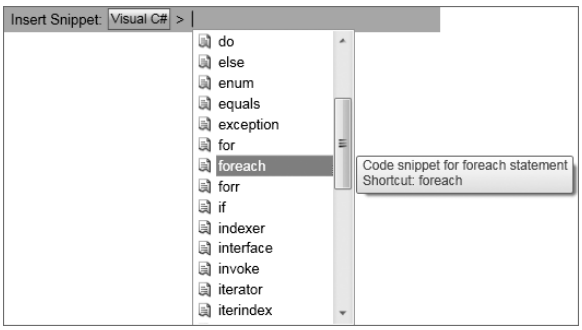


Рис. 7.20

Комментарии XML

Комментарии XML описаны в главе 12 как способ автоматического документирования проектов и решений. Еще одним преимуществом использования XML-комментариев в своем коде является то, что система Visual Studio может использовать их в своей системе IntelliSense для вывода подсказок и более полной информации о параметрах, помимо простой информации о типах, как это делается в обычных пользовательских классах.

Добавление собственных схем IntelliSense

Пользователь может добавить собственные схемы IntelliSense, которые бывают полезными для редактирования XML- и HTML-файлов. Для этого следует создать правильно отформатированный XML-файл и инсталлировать его в подкаталог `Common7\Packages\schemas\xml` каталога инсталляции системы Visual Studio (по умолчанию `C:\Program Files\Microsoft Visual Studio 10.0`). Например, можно расширить возможности технологии IntelliSense для XML-редактора, включив свою собственную схему определений. Описание процесса создания таких схем выходит за рамки нашей книги, но их можно найти в Интернете, послав поисковый запрос “IntelliSense schema in Visual Studio”.

Закладки и окно Bookmark

Закладки в системе Visual Studio 2010 позволяют пометить место в коде, чтобы к нему можно было легко вернуться впоследствии. Как показано на рис. 7.21, эти закладки представляют собой индикаторы на левом поле кода.

Для того чтобы включать и отключать закладки на строке, можно использовать комбинацию клавиш <Ctrl+K>, <Ctrl+K>. В качестве альтернативы можно также использовать команду Edit⇨Bookmarks⇨Toggle Bookmark.



Напомним, включение и отключение закладки означает, что, применив команду к уже помеченной строке, вы удаляете с нее закладку.

На рис. 7.21 показан раздел окна редактора кода с двумя видами закладок. Закладка вверху находится в нормальном состоянии, которое изображается затененным голубым прямоугольником. Закладка внизу отключена и представлена в виде белого прямоугольника. Отключение закладки позволяет пользователю использовать ее в дальнейшем, но при этом функции навигации по закладкам будут ее игнорировать.

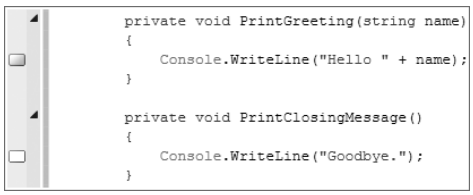


Рис. 7.21

Для того чтобы отключить закладку, можно использовать команду Edit⇨Bookmarks⇨Enable Bookmark. С помощью этой же команды можно снова включить закладку. На первый взгляд, это противоречит логике, поскольку на самом деле разработчик хочет отключить активную закладку, но по некоторым причинам этот пункт меню невозможно обновить с учетом контекста курсора.



Существует возможность задать комбинацию клавиш для включения и отключения закладок, если это приходится делать слишком часто. Для этого необходимо открыть страницу Keyboard Options в группе Environment окна Options и найти в списке команду `Edit.EnableBookmark`.

Кроме включения и отключения закладок, система Visual Studio позволяет открыть окно Bookmarks, показанное на рис. 7.22. Для того чтобы открыть это инструментальное окно, следует нажать комбинацию клавиш <Ctrl+K>, <Ctrl+W> или выбрать команду View⇨Bookmark Window. По умолчанию это окно прикреплено к нижнему краю окна интегрированной среды разработки и использует это пространство совместно с другими инструментальными окнами, такими как Task List и Find Results.

На рис. 7.22 показаны некоторые полезные свойства закладок в системе Visual Studio 2010. Первое свойство — это способность создавать каталоги логических групп закладок. Например, обратите внимание на то, что каталог Old Bookmarks содержит закладку `Bookmark3`.

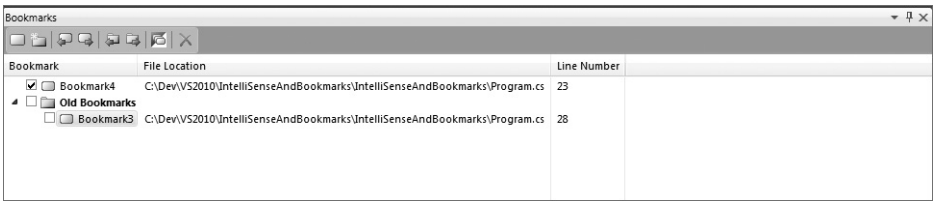


Рис. 7.22

Для того чтобы создать каталог закладок, достаточно щелкнуть на пиктограмме New Folder на панели инструментов, расположенной вдоль верхней части окна Bookmarks (вторая кнопка слева). В результате будет создан пустой каталог (с именем Folder1, заданным по умолчанию, за которым будет создан каталог с именем Folder2 и т.д.), а его имя будет находиться в фокусе. Закладку можно поместить в каталог, выбрав ее вхождения в списке и перетащив на требуемое место. Обратите внимание на то, что невозможно создать иерархию каталогов, но эта возможность представляется чрезвычайно маловероятной. Закладки можно переименовывать точно так же, как и каталоги. Если это приходится делать постоянно, то имеет смысл выбрать имена, предлагаемые по умолчанию: `Bookmark1`, `Bookmark2` и т.д. Каталоги — не только удобный способ группирования закладок; они также обеспечивают удобный способ включения и отключения целых групп закладок путем простой установки или сброса флажка, расположенного после имени каталога.

Для того чтобы перейти непосредственно к закладке, следует дважды щелкнуть в инструментальном окне Bookmarks. В качестве альтернативы, если нужно пройти по включенным закладкам, определенным в проекте, можно использовать команды Previous Bookmark (<Ctrl+K>, <Ctrl+P>) и Next Bookmark (<Ctrl+K>, <Ctrl+N>). Перемещение по закладкам можно ограничить конкретным каталогом, сначала выбрав закладку в каталоге, а затем выбрав команды Previous Bookmark in Folder (<Ctrl+Shift+K>, <Ctrl+Shift+P>) и Next Bookmark in Folder (<Ctrl+Shift+K>, <Ctrl+Shift+N>).

Последние две пиктограммы в окне Bookmarks —Toggle All Bookmarks, которая позволяет отключить (и повторно включить) все закладки, определенные в проекте, и Delete, которая используется для удаления каталога или закладки из списка.



Удаление каталога приводит к удалению всех закладок, содержащихся в этом каталоге. Система Visual Studio предусматривает вывод окна для подтверждения удаления, чтобы предотвратить случайную потерю закладок. Удаление закладки одновременно означает ее отключение.

Закладками можно также управлять с помощью подменю Bookmarks, которое находится в меню Edit. В системе Visual Studio 2010 закладки сохраняются между сессиями, что делает постоянные закладки все более важным элементом организации кода.

Списки Task содержат специальные варианты закладок, которые выводятся только в собственных инструментальных окнах. Единственная связь, которая остается между ними, заключается в том, что в меню Bookmarks по-прежнему существует команда Add Task List Shortcut. Следует помнить, что она задает комбинацию клавиш не для окна Bookmarks, а для списка Shortcuts в окне Task List.

Резюме

Функциональные возможности технологии IntelliSense выходят за пределы основного окна редактирования кода. Мощь технологии IntelliSense, проявляющаяся в автоматическом дополнении инструкций и параметров, можно использовать во многих других инструментальных окнах, например Command и Immediate. Любые ключевые слова и даже переменные и объекты, известные в текущем контексте на протяжении сессии отладки, могут быть найдены с помощью списка членов классов IntelliSense. Технология IntelliSense во всех ее проявлениях значительно повышает эффективность работы сис-

темы Visual Studio по сравнению с большинством других доступных инструментов. Постоянно отслеживания нажатия клавиш, чтобы обеспечить обратную визуальную связь или автоматическое дополнение и генерацию кода, система IntelliSense позволяет разработчикам исключительно быстро, эффективно и правильно набирать программы с первой попытки. В следующей главе будет рассмотрен механизм генерации сниппетов, представляющий мощное дополнение к технологии IntelliSense.

В этой главе также показано, как можно расставить закладки в коде и перемещаться от одной закладки к другой. Освоив комбинации клавиш для сокращенного набора команд, пользователь может значительно повысить производительность своей работы.

8

Сниппеты кода и рефакторинг

В ЭТОЙ ГЛАВЕ...

- Использование сниппетов
- Создание собственных сниппетов
- Рефакторинг кода

Одно из преимуществ *интегрированной среды разработки* (Integrated Development Environment — IDE) над обычным текстовым редактором заключается в том, что она позволяет повысить производительность и эффективность работы, ускоряя процесс работы над программой. Двумя наиболее мощными возможностями повышения производительности работы разработчика в системе Visual Studio 2010 являются поддержка сниппетов и инструменты для рефакторинга кода.

Сниппеты (code snippets) — это небольшие фрагменты текста, которые можно вставлять в основной код приложения, а затем настраивать в соответствии с его требованиями. Они не позволяют генерировать законченные приложения или целые файлы в отличие от шаблонов проекта и элементов управления. Вместо этого они используются для вставки часто используемых структур или запутанных блоков программ, которые тяжело запомнить. В первой части главы будет показано, как, используя сниппеты, можно значительно увеличить эффективность кодирования.

В главе также рассматриваются инструменты рефакторинга кода в системе Visual Studio 2010. *Рефакторинг* (refactoring) — это процесс переработки кода с целью его улучшения без изменения функциональных возможностей. Он может заключаться в упрощении метода, в выявлении шаблона часто повторяющегося кода и даже в оптимизации раздела кода, чтобы повысить его эффективность.

Несмотря на то что в системе Visual Studio уже были предусмотрены инструменты рефакторинга кода на языке C#, к сожалению, для языка VB таких средств не было.

Чтобы устранить этот недостаток предыдущих версий системы Visual Studio, компания Microsoft купила лицензию на программу Refactor!, разработанную компанией Developer Express для языка VB. Для реализации рефакторинга кода на языке VB в системе Visual Studio 2010 на основе программы Refactor! была разработана надстройка CodeRush Xpress (также компании Developer Express). И хотя компания Microsoft лицензировала надстройку CodeRush Xpress, ее по-прежнему необходимо загружать и инсталлировать отдельно от системы Visual Studio. Загрузить ее можно с сайта центра разработчиков программ на языке VB, расположенного по адресу `http://msdn.microsoft.com/vbasic/`, следуя по ссылкам Downloads⇨Tools and Utilities. (Для русскоязычных пользователей предназначена веб-страница `http://msdn.microsoft.com/ru-ru/vbasic/default.aspx` со ссылкой Файлы для загрузки. — *Примеч. ред.*)

Надстройка CodeRush Xpress предоставляет пользователям широкую поддержку рефакторинга, дополняющую встроенные средства, доступные для разработчиков, работающих на языке C#. Однако обсуждение в этой главе ограничивается только встроенной поддержкой рефакторинга кода, предоставляемой системой Visual Studio 2010 (для программистов, работающих на языке C#) и соответствующими командами надстройки CodeRush Xpress (для программистов, работающих на языке VB).

Введение в сниппеты

Сниппеты уже давно существовали во многих формах, но в ранних системах Visual Studio и в среде VB6 для их поддержки обычно требовались надстройки сторонних поставщиков. Система Visual Studio 2010 обеспечивает интенсивную поддержку сниппетов, что позволяет вставлять в файл блоки кода вместе с заранее подставленными переменными, облегчая их дальнейшую настройку.

Хранение сниппетов

Прежде чем исследовать сниппеты, рассмотрим простейшие средства, с помощью которых система Visual Studio позволяет вставлять в файл заранее заготовленные блоки текста. Наряду с элементами управления, которые вставляются в форму, панель компонентов Toolbox может содержать блоки текстов (например, кода), которые тоже можно вставлять в файл. Для того чтобы добавить блок кода (или другого текста) на панель Toolbox, достаточно выбрать текст в окне редактора и перетащить его на панель Toolbox. В результате будет создан элемент панели Toolbox, именем которого является первая строка кода. Эти фрагменты можно переименовывать, переставлять и группировать так, как и любые другие компоненты инструментальной панели Toolbox. Для того чтобы вставить блок кода в файл, его следует перетащить с панели Toolbox в намеченное место файла, как показано на рис. 8.1, или дважды щелкнуть на элементе панели Toolbox, чтобы вставить фрагмент текста в текущую позицию курсора в активном файле.

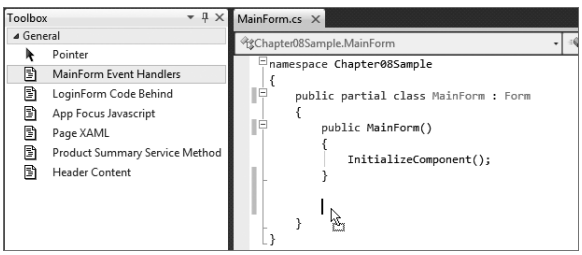


Рис. 8.1



Многие презентаторы часто используют эту возможность, чтобы быстро вставлять большие блоки кода при создании программ в ходе презентаций.

Это простейшая форма сниппетов в системе Visual Studio 2010, но ее простота означает ограниченные функциональные возможности, в частности, нельзя модифицировать и разделять фрагменты кода с другими пользователями. Тем не менее данный метод работы с небольшими фрагментами кода в некоторых ситуациях может оказаться полезным, чтобы обеспечить их быстрое использование.

Сниппеты

Теперь рассмотрим намного более полезный способ вставки блоков кода в файл: *сниппеты*. Они записываются в отдельных XML-файлах вместе с переменными параметрами, облегчающими настройку сниппета для конкретной задачи. Сниппеты являются частью технологии IntelliSense и значительно упрощают поиск и вставку блоков кода в текст программы.



Сниппеты на языке VB также позволяют добавлять ссылки на сборки и вставлять инструкции импорта.

Система Visual Studio 2010 изначально содержит много заранее подготовленных сниппетов для двух основных языков, VB и C#, а также для языков JavaScript, HTML и XML. Эти сниппеты организованы в логические иерархии, поэтому их можно легко найти. Вместо инструментальной панели Toolbox для поиска сниппета можно также использовать команды или комбинации клавиш, открывающие списки основных групп сниппетов.

Кроме того, пользователь может создавать свои собственные сниппеты и хранить их в той же самой библиотеке. Поскольку каждый сниппет хранится в специальном XML-файле, их можно использовать совместно с другими разработчиками.

Сниппеты можно вставлять в следующие части программы.

- **Объявление класса.** В этом случае сниппет, по существу, генерирует целый класс.
- **Объявление члена класса.** В этом случае сниппет содержит определение членов, в частности, методов, свойств и процедур для обработки событий. Это означает, что его следует вставлять за пределами существующих членов класса.
- **Тело члена класса.** Эта часть кода предназначена для сниппетов, которые вставляются в уже определенный член класса, например в метод.

Использование сниппетов в языке C#

Команда Insert Snippet является одной из функциональных возможностей технологии IntelliSense. Она встроена в редактор кода и вначале выводит строку "`Insert Snippet`" вместе с раскрывающимся списком групп сниппетов, в которых можно выбрать подходящий фрагмент. Когда пользователь выбирает группу, содержащую требуемый сниппет (используя навигационные клавиши со стрелками, направленными вверх и вниз, и клавишу <Tab>), система показывает ему список сниппетов. Для выбора нужного сниппета достаточно дважды щелкнуть на его имени (или нажать клавишу <Tab> или <Enter>).

Для того чтобы вставить сниппет в код на языке C#, пользователю достаточно показать место, куда он хочет вставить генерируемый код, а затем открыть список Insert Snippet с помощью комбинации клавиш <Ctrl+K>, <Ctrl+X>. Существуют еще два способа запустить процесс вставки сниппетов. Во-первых, можно щелкнуть правой кнопкой мыши в окне редактора кода и выбрать команду Insert Snippet из открывшегося контекстного меню. Во-вторых, можно выбрать команду Edit⇨IntelliSense⇨Insert Snippet.

В результате система Visual Studio откроет список Insert Snippet, как показано на рис. 8.2. Когда пользователь прокручивает этот список и устанавливает курсор мыши на его элементы, на экране появляется подсказка, в которой содержится краткое описание сниппета и указывается его сокращенное имя для вставки в программу.

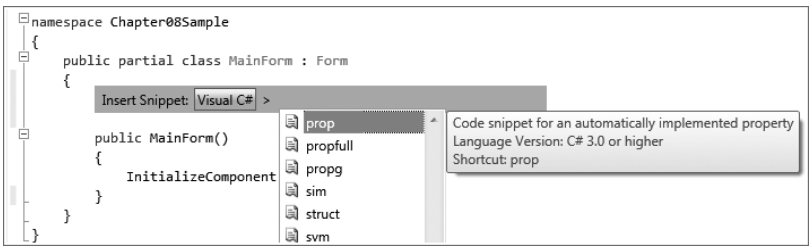


Рис. 8.2

Для того чтобы воспользоваться этим именем, достаточно набрать его в редакторе кода (обратите внимание на то, что сниппет указан в списке IntelliSense) и дважды нажать клавишу <Tab>. В результате сниппет будет вставлен в требуемую позицию.

На рис. 8.3 показан результат выбора сниппета Automatically Implemented Property. Довольно часто получившийся код требуется модифицировать. Фрагменты текста, которые обычно можно изменять (переменные для замены), выделяются подсветкой.

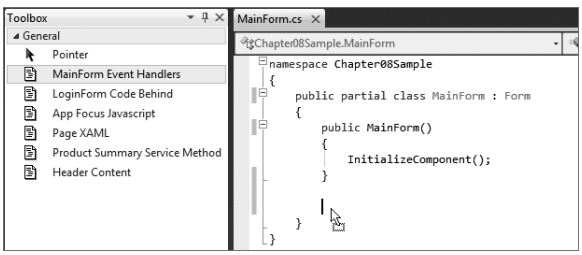


Рис. 8.3

Изменяя переменные в сгенерированном сниппете, система Visual Studio 2010 оказывает пользователю дополнительную помощь. Нажатие клавиши <Tab> переносит подсветку на следующую переменную, значение которой можно заменить своим собственным. Для того чтобы вернуться назад, необходимо нажать клавиши <Shift+Tab>. Таким образом, разработчик может легко найти разделы кода, которые необходимо изменить, не выбирая следующий фрагмент вручную. Некоторые сниппеты используют общие переменные. Это значит, что изменение значения такой переменой в одном месте приводит к его изменению во всех экземплярах.

Для того чтобы скрыть подсветку переменных в сниппете, можно продолжить редактирование или нажать клавишу <Enter> или <Esc>.

Использование сниппетов в языке VB

Сниппеты в языке VB имеют дополнительные возможности по сравнению со сниппетами в языке C#, например, они могут автоматически добавлять в проект ссыл-

ки на сборки, а также вставлять инструкции импорта в файл, содержащий компилируемый код.

Для того чтобы использовать сниппет, сначала необходимо выбрать, в какое место листинга программы следует вставить сгенерированный код, и установить в этом месте курсор. Пользователю не следует беспокоиться о ссылках и инструкциях сборки; они будут помещены в правильные места. Затем, как и в случае сниппетов в программах на языке C#, с помощью одного из следующих методов можно открыть список Insert Snippet.

- Комбинация клавиш <Ctrl+K>, <Ctrl+X>.
- Щелчок правой кнопкой мыши и выбор команды Insert Snippet из контекстного меню.
- Выбор команды Edit⇨IntelliSense⇨Insert Snippet.

В системе VB есть еще один способ, с помощью которого можно открыть список Insert Snippet List: просто ввести знак вопроса (`?`) и нажать клавишу <Tab>.

Например, пройдем по иерархии и вставим сниппет `Draw a Pie Chart` (рис. 8.4).

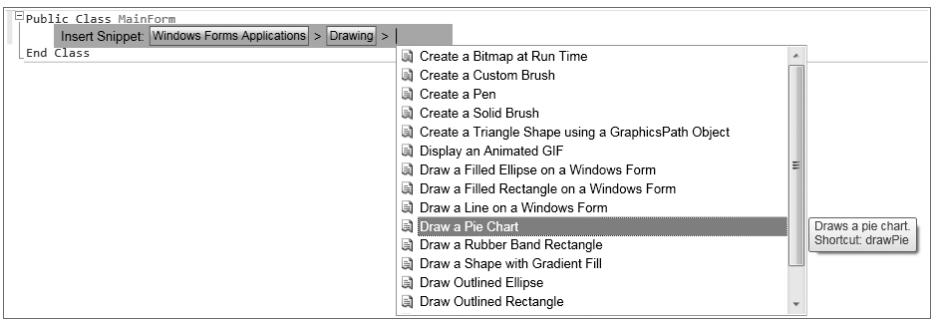


Рис. 8.4

На рис. 8.4 видно, что в тексте подсказки имеется строка "`Shortcut: drawPie`". Этот текст означает, что выбранный сниппет имеет сокращенное название, которое можно использовать для его автоматического вызова без перебора элементов иерархии сниппетов. Как и при работе с программами на языке C#, достаточно ввести это сокращенное название в окне редактора кода и нажать клавишу <Tab>. В системе VB эти названия не чувствительны к регистру, поэтому данный пример можно сгенерировать, набрав слово "`drawpie`" и нажав клавишу <Tab>. Обратите внимание на то, что это имя не появляется в списке IntelliSense для языка VB, как это происходило для языка C#.

Если после вставки сниппет содержит переменные для замены, пользователь может вставить их значения, а затем перемещаться между ними, нажимая клавишу <Tab>, как было описано выше при обзоре сниппетов для языка C#. Для того чтобы скрыть подсветку этих переменных сниппета, можно просто продолжить кодирование или щелкнуть правой кнопкой мыши и выбрать команду Hide Snippet Highlighting. Если необходимо подсветить все переменные для замены, входящие в сниппет, вставленные после открытия файла, достаточно щелкнуть правой кнопкой мыши и выбрать команду Show Snippet Highlighting.

Окружение с помощью сниппета

Последнее действие по рефакторингу, доступное в языке C# (и в языке VB с помощью надстройки CodeRush Xpress), — возможность окружить сниппетом существующий блок кода. Например, для того чтобы погрузить существующий блок кода в условный блок "try-catch", достаточно щелкнуть правой кнопкой мыши и выбрать команду Surround With или выбрать блок кода и нажать клавиши <Ctrl+K>, <Ctrl+S>.

В ответ откроется диалоговое окно Surround With, содержащее список сниппетов, доступных для окружения выбранной строки кода (рис. 8.5).

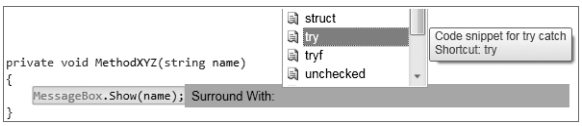


Рис. 8.5

Выбор сниппета `try` порождает следующий код.

VB

```
Public Sub MethodXYZ(ByVal name As String)
    Try
        MessageBox.Show(name)
    Catch ex As Exception
        Throw
    End Try
End Sub
```

C#

```
public void MethodXYZ(string name)
{
    try
    {
        MessageBox.Show(name);
    }
    catch (Exception)
    {
        throw;
    }
}
```

Библиотека Code Snippets Manager

Code Snippets Manager — это главная библиотека сниппетов в системе Visual Studio 2010. Для доступа к ней необходимо выбрать команду Tools⇨Code Snippet Manager или нажать комбинацию клавиш <Ctrl+K>, <Ctrl+B>.

Когда библиотека Code Snippets Manager открывается в первый раз, в ней отображаются доступные сниппеты на языке HTML, хотя с помощью списка Language пользователь может сам задать требуемый язык программирования. На рис. 8.6 показан экран дисплея при редактировании проекта на языке C#. Иерархическая структура папок по умолчанию копирует множество папок на персональном компьютере, но по мере того как пользователь будет добавлять файлы со сниппетами из разных мест и вставлять их в разные группы, новые сниппеты будут распределяться по соответствующим папкам.

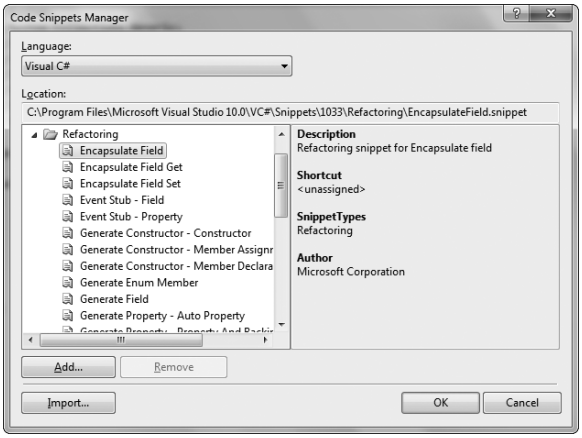


Рис. 8.6

Если разработчику необходимо включить в библиотеку целый файл сниппетов, например, для того, чтобы импортировать сниппеты, разработанные в компании, достаточно щелкнуть на кнопке Add. В результате откроется диалоговое окно, в котором можно выбрать требуемую папку. Папки, добавленные с помощью такого способа, вставляются в вершину иерархии — на уровне основных групп сниппетов, заданных по умолчанию. Однако пользователь может добавлять папки, содержащие подпапки, которые будут добавлены в это дерево в качестве дочерних узлов.

Удалить папку очень просто, даже небезопасно просто. Достаточно выбрать узел дерева, который нужно удалить, и щелкнуть на кнопке Remove. В результате этот узел, все его дочерние узлы и сниппеты будут удалены из библиотеки Code Snippets Manager, причем окно подтверждения при этом не открывается. Если пользователь сделал это случайно, то лучше всего щелкнуть на кнопке Cancel и открыть диалоговое окно снова. Если пользователь внес изменения, которые он хочет сохранить, то достаточно выполнить все действия, описанные выше. Однако искать место для папки, случайно удаленной из списка сниппетов, заданных по умолчанию, может оказаться утомительным занятием.

Сниппеты, инсталлированные вместе с системой Visual Studio 2010, расположены глубоко в иерархии папок инсталляции. По умолчанию библиотека сниппетов при запуске под управлением 32-битовой системы Windows инсталлируется в папке `%programfiles%\Microsoft Visual Studio 10.0\VB\Snippets\1033` для языка VB и в папке `%programfiles%\Microsoft Visual Studio 10.0\VC#\Snippets\1033` для языка C# (если используется 64-битовая система Windows, то строку `%programfiles%` следует заменить строкой `%programfiles(x86)%`). Отдельные файлы сниппетов можно импортировать в библиотеку с помощью кнопки Import. Преимущество этого метода над использованием кнопки Add заключается в том, что в этом случае пользователь получает возможность указать место каждого сниппета в структуре библиотеки.

Создание сниппетов

Система Visual Studio 2010 не содержит средств для создания или редактирования сниппетов. Тем не менее программа Bill McCarthy's Snippet Editor позволяет создавать свои собственные сниппеты, модифицировать их и управлять ими (включая поддержку для языков VB, C#, HTML, JavaScript и XML). Сначала программа Snippet Editor была внутренним проектом компании Microsoft, но теперь она представляет собой открытое программное обеспечение, расположенное на веб-сайте CodePlex, на котором компания Bill исправляет существенные ошибки и вносит дополнительные функциональные возможности. Благодаря помощи других профессионалов программа Snippet Editor

поддерживает теперь много разных языков программирования. Загрузить редактор сниппетов можно по адресу `http://snippeteditor.codeplex.com`.

Создание сниппетов с помощью редактирования соответствующих XML-файлов может оказаться утомительным и сложным процессом, а программа Snippet Editor делает это много проще. Начиная работать с редактором Snippet Editor, пользователь видит в правом левом углу раскрывающийся список, в котором следует выбрать систему Visual Studio 2010. Ниже приведено дерево, содержащее все сниппеты, известные в системе Visual Studio 2010. Раскрыв узел, можно увидеть совокупность папок, аналогичных иерархии папок в библиотеке сниппетов (рис. 8.7).

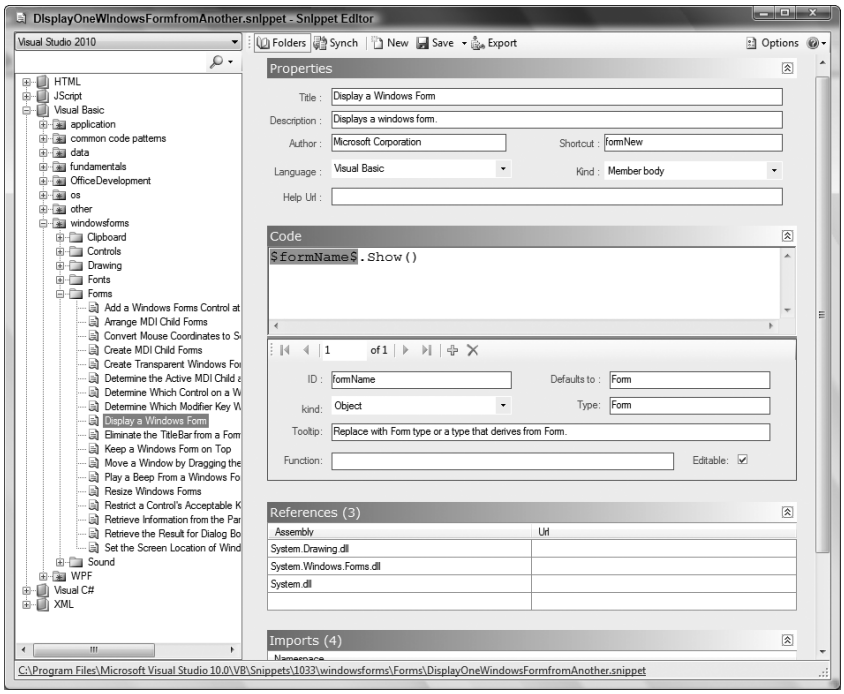


Рис. 8.7

Просмотр существующих сниппетов

Одним из преимуществ программы Snippet Editor является удобное представление структуры любого файла сниппетов, существующего в системе. Благодаря этому пользователь может просматривать сниппеты, установленные в системе Visual Studio по умолчанию, чтобы понять, как лучше создать свои собственные сниппеты.

Пользователь может перейти к интересующему его сниппету и дважды щелкнуть на его имени, чтобы он открылся в окне Editor. На рис. 8.7 показан просто сниппет Display a Windows Form. Четыре панели содержат всю информацию об этом сниппете. Эти панели описаны в табл. 8.1.

Переходя по этим закладкам, разработчик может анализировать свойства существующего сниппета и его переменные, предназначенные для замены. В примере, показанном на рис. 8.7, существует только одна область с ID, равным `formName`, и значением по умолчанию, равным “`Form`”.

Таблица 8.1. Информационные панели для сниппетов

Панель	Функция
Properties	Содержит основные свойства сниппета, включая заголовок, комбинацию клавиш и описание
Code	Определяет код для сниппета, включая все области замены Literal и Object
References	Если для сниппета необходимы ссылки на сборку, эта закладка позволяет разработчику определить их
Imports	Как и закладка References, позволяет разработчику определить все инструкции `Imports`, необходимые для корректной работы сниппета

Для того чтобы продемонстрировать, насколько просто программа Snippet Editor создает собственные сниппеты пользователя, перейдем к следующему упражнению, в котором создадим сниппет, порождающий три процедуры, включая вспомогательную процедуру.

1. Запускаем программу Snippet Editor и создаем новый сниппет. Для этого выбираем в дереве предназначенную папку, щелкаем на ней дважды и выбираем команду Add New Snippet из появившегося контекстного меню.
2. В ответ на приглашение вводим имя сниппета "`Create A Button Sample`" и щелкаем на кнопке OK. Дважды щелкаем на новом элементе, чтобы открыть окно Editor.



Обратите внимание на то, что сниппет не открывается автоматически в окне Editor, — не замените свойства другого сниппета по ошибке!

3. Теперь в первую очередь редактируем поля Title, Description и Shortcut (рис. 8.8).

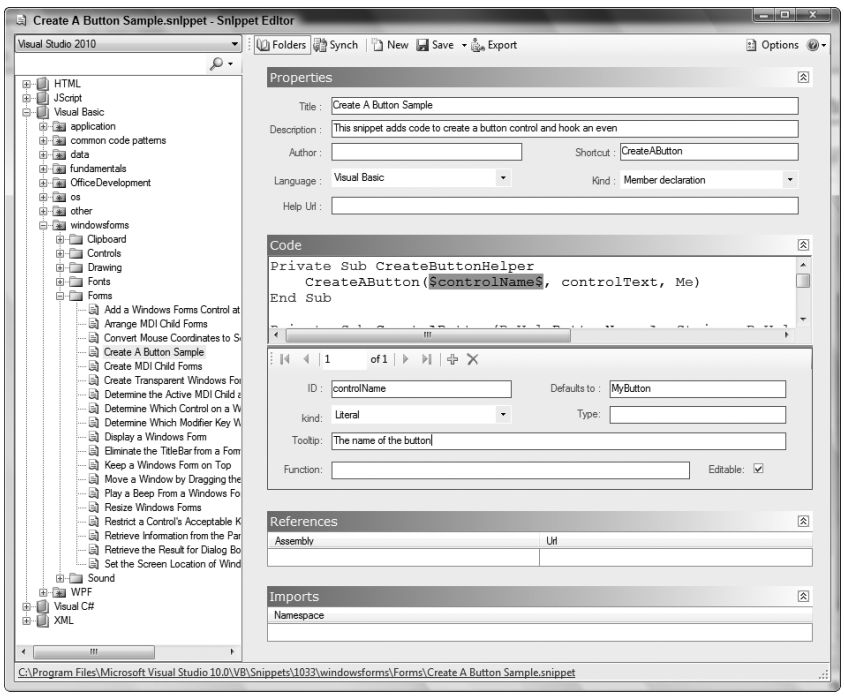


Рис. 8.8

- Поле Title: `Create A Button Sample` (Создание кнопки. — *Примеч. ред.*)
- Поле Description. `This snippet adds code to create a button controland hook an even` (Этот сниппет добавляет код для создания кнопки управления и связывает с ней обработчик события. — *Примеч. ред.*)
- Поле Shortcut: `CreateAButton`

4. Поскольку этот сниппет содержит определения членов класса, следует установить для переменной Type значение "`Member Declaration`".
5. В окне Editor вставляем код, необходимый для создания трех процедур.

VB

```
Private Sub CreateButtonHelper
    CreateAButton(controlName, controlText, Me)
End Sub

Private Sub CreateAButton(ByVal ButtonName As String, _
                          ByVal ButtonText As String, _
                          ByVal Owner As Form)
    Dim MyButton As New Button

    MyButton.Name = ButtonName
    MyButton.Text = ButtonName
    Owner.Controls.Add(MyButton)

    MyButton.Top = 0
    MyButton.Left = 0
    MyButton.Text = ButtonText
    MyButton.Visible = True

       AddHandler MyButton.Click, AddressOf ButtonClickHandler
End Sub

Private Sub ButtonClickHandler(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs)
    MessageBox.Show("The " & sender.Name & " button was clicked")
End Sub
```

C#

```
private void CreateButtonHelper()
{
    CreateAButton(controlName, controlText, this);
}

private void CreateAButton(string ButtonName, string ButtonText,
                           Form Owner)
{
    Button MyButton = new Button();

    MyButton.Name = ButtonName;
    MyButton.Text = ButtonName;
    Owner.Controls.Add(MyButton);

    MyButton.Top = 0;
    MyButton.Left = 0;


MyButton.Text = ButtonText;
    MyButton.Visible = true;
```

```
    MyButton.Click += MyButton_Click;
}

private void MyButton_Click(object sender, EventArgs e)
{
    MessageBox.Show("The " + sender.Name + " button was clicked");
}
```

6. Обратите внимание на то, что этот код отличается от кода, представленного на рис. 8.8, тем, что слово `controlName` не подсвечено. На рис. 8.8 этот аргумент представляет собой поле для замены. Для того чтобы сделать слово полем для замены, необходимо выделить его, щелкнуть правой кнопкой мыши и выбрать команду Add Replacement (или щелкнуть на кнопке Add, которая расположена ниже окна кода).
7. Изменяем значения заменяемых свойств следующим образом.
   - Поле ID : `controlName`.
   - Поле Defaults to: "`MyButton`".
   - Поле Tooltip: имя кнопки.
8. Повторяем то же самое для элемента controlText.
   - Поле ID: `controlText`.
   - Поле Defaults to: "`Click Me!`".
   - Поле Tooltip: текстовое свойство кнопки.

Наш сниппет закончен и готов к использованию. Теперь его можно вставить в окно редактора кода системы Visual Studio 2010.

Поддержка рефакторинга

Существует много способов вызвать инструменты для рефакторинга в системе Visual Studio 2010, включая контекстное меню, которое открывается после щелчка правой кнопкой мыши, интеллектуальные дескрипторы и меню Refactor в главном меню (только для разработчиков, работающих на языке программирования C#).

На рис. 8.9 показано контекстное меню Refactor, предназначенное для разработчиков, работающих на языке программирования C#. Полный список действий по рефакторингу, доступных для разработчиков, работающих на языке программирования C# в системе Visual Studio 2010, включает операции Rename, Extract Method, Encapsulate Field, Extract Interface, Promote Local Variable to Parameter, Remove Parameters и Reorder Parameters. Можно также использовать операции Generate Method Stub и Organize Usings, которые с некоторой натяжкой также можно отнести к рефакторингу.

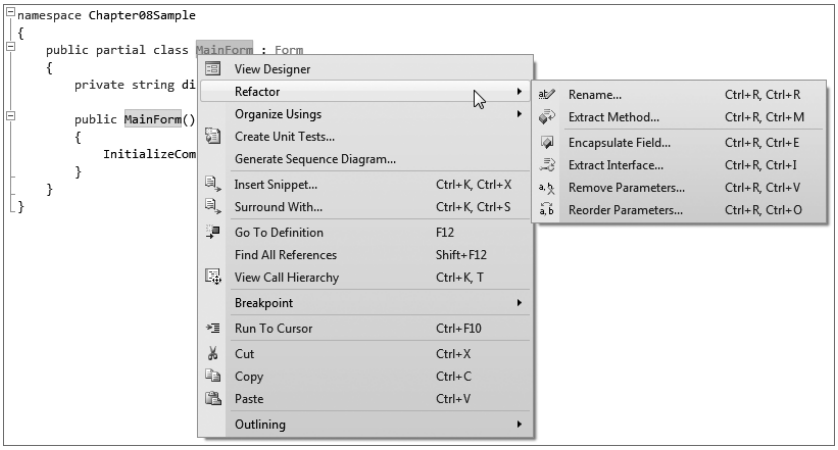


Рис. 8.9

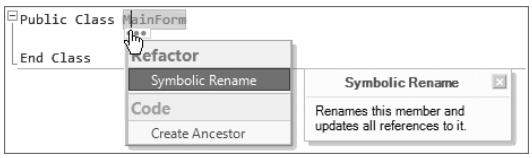


Рис. 8.10

Встроенная поддержка рефакторинга, предлагаемая системой Visual Studio 2010 для разработчиков проектов на языке VB, ограничена символическими методами Rename и Generate Method Stub. Дополнительная поддержка рефакторинга для разработчиков, работающих на языке программирования VB, предоставляется надстройкой CodeRush Xpress. Доступ к этой надстройке открывается либо с помощью контекстного меню, которое появляется после щелчка правой кнопкой мыши (его содержание динамически изменяется, поэтому в каждой ситуации доступными являются только релевантные операции рефакторинга), либо с помощью интеллектуальных дескрипторов (как показано на рис. 8.10), которые появляются на экране, когда для текущего выделения можно сделать рефакторинг (активизируемый с помощью комбинации клавиш <Ctrl+`>).

Надстройка CodeRush Xpress осуществляет поддержку всех средств рефакторинга, уже существующих в языке C#, и, помимо этого, добавляет много новых (для обоих языков). Примерами традиционного рефакторинга являются операции Create Overload, Flatten Conditional, Inline Temp, Introduce Constant, Introduce Local, Move Declaration Near Reference, Move Initialization to Declaration, Remove Assignments to Parameters, Rename, Reorder Parameters, Replace Temp with Query, Reverse Conditional, Safe Rename, Simplify Expression, Split Initialization from Declaration и Split Temporary Variable.

Операции рефакторинга

В следующих разделах описывается каждая из операций рефакторинга и приводятся примеры использования встроенной поддержки рефакторинга в языке C# и поддержки, обеспечиваемой надстройкой CodeRush Xpress для языка VB.

Операция Extract Method

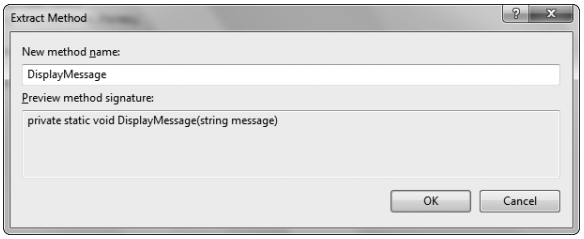


Рис. 8.11

Чтобы начать рефакторинг длинного метода, лучше всего разбить его на несколько небольших частей. Для этого необходимо выделить область кода, которую разработчик хочет удалить из исходного метода, и выбрать команду Extract Method из контекстного меню. В языке C# это приведет к появлению приглашения ввести имя нового метода, как показано на рис. 8.11. Если в удаляемом блоке кода есть переменные, которые использовались ранее, они автоматически становятся переменными в сигнатуре нового метода. После подтверждения имени нового метода его код вставляется сразу после исходного метода. Удаленный блок метода заменяется вызовом нового метода.

Например, в следующем сниппете, для того чтобы вынести условное логическое выражение в отдельный метод, необходимо выделить код, выделенный полужирным шрифтом, и выбрать команду Extract Method из контекстного меню, появляющегося после щелчке правой кнопкой мыши.

C#

```
private void button1_Click(object sender, EventArgs e)
{
    string connectionString = Properties.Settings.Default.ConnectionString;
    if (connectionString == null)
    {
        connectionString = "DefaultConnectionString";
    }
    MessageBox.Show(connectionString);
    /* ... Длинный метод ... */
}
```

На этом месте автоматически генерируется следующий код.

C#

```
Private void button1_Click(object sender, EventArgs e)
{
    string connectionString = Properties.Settings.Default.ConnectionString;
    connectionString = ValidateConnectionString(output);
    MessageBox.Show(connectionString);
    /* ... Длинный метод ... */
}

   private static string ValidateConnectionString(string connectionString)
{
    if (connectionString == null)
    {
        connectionString = "DefaultConnectionString";
    }
    return connectionString;
}
```

При работе с программами на языке VB надстройка CodeRush Xpress выполняет эту операцию рефакторинга немного иначе. После того как разработчик выделит код, подлежащий замене, надстройка CodeRush Xpress предлагает ему выбрать место, в которое следует вставить новый метод. Это должно помочь разработчику организовать свои методы по группам в соответствии с алфавитом или функциональными свойствами.

На рис. 8.12 показана подсказка, которая появляется на экране и помогает разработчику правильно выбрать место для нового метода путем перемещения курсора.

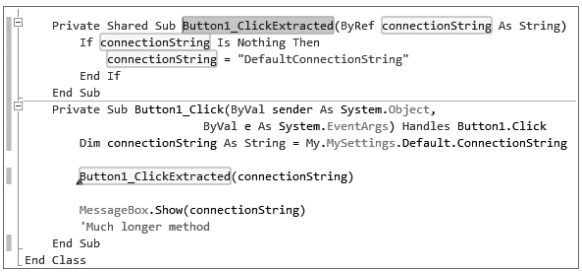


Рис. 8.12

Выбрав место для вставки нового метода, надстройка CodeRush Xpress вставляет его туда, присваивая ему произвольно выбранное имя. При этом она подсвечивает имя метода, позволяя разработчику переименовать метод либо на месте его расположения, либо в месте его вызова (рис. 8.13).

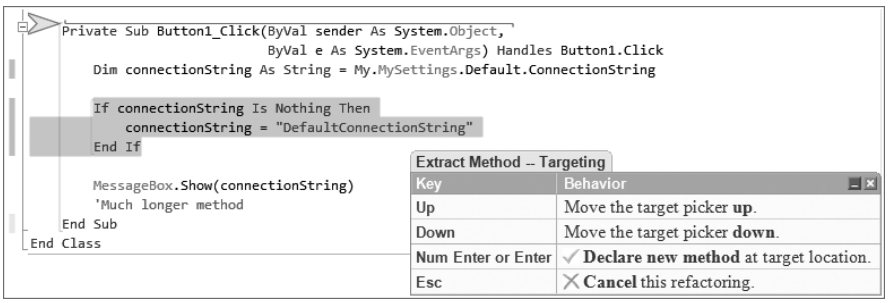


Рис. 8.13

Выполним команду Extract Method для рефакторинга следующего кода.

VB

```
Private Sub Button1_Click(ByVal sender As System.Object,
                         ByVal e As System.EventArgs) Handles Button1.Click
   Dim connectionString As String = My.MySettings.Default.ConnectionString
   If connectionString Is Nothing Then
       connectionString = "DefaultConnectionString"
   End If
   MessageBox.Show(connectionString)
   'Длинный метод
End Sub
```

Затем переименуем метод, дав ему осмысленное имя.

VB

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

```
Dim connectionString As String = My.MySettings.Default.ConnectionString
    ValidateConnectionString(connectionString)
    MessageBox.Show(connectionString)
    'Длинный метод
End Sub

   Private Shared Sub ValidateConnectionString(ByRef connectionString As
String)
    If connectionString Is Nothing Then
        connectionString = "DefaultConnectionString"
    End If
End Sub
```

Операция Encapsulate Field

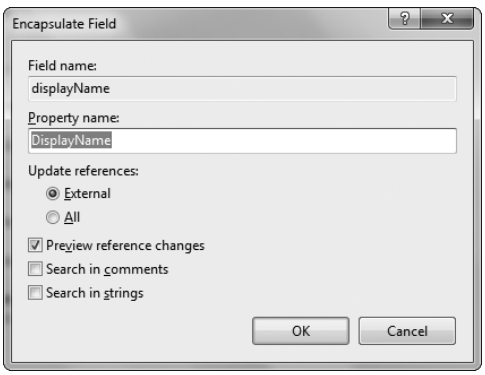


Рис. 8.14

При рефакторинге часто требуется инкапсулировать в существующий класс некое свойство. Для этого предназначена операция Encapsulate Field. Чтобы выполнить это действие, следует выбрать переменную, которую требуется инкапсулировать, и выбрать соответствующую команду в контекстном меню. Это дает разработчику возможность указать свойство и выбрать место, в котором следует искать ссылку на него (рис. 8.14).

На следующем шаге после указания нового свойства следует определить, какие ссылки на переменную класса следует заменить ссылкой на новое свойство. На рис. 8.15 показано окно предварительного просмотра, которое открывается по завершении поиска ссылок. В верхней панели окна размещено дерево, в котором указываются поля и методы, имеющие ссылки на данную переменную. Флажок, расположенный после каждой строки, указывает, следует ли выполнять замену. Выбрав строку в верхней части окна, необходимо установить на нее фокус в нижней панели. Как только будет обеспечена корректность каждой ссылки, инкапсуляция считается завершенной. Переменная класса устанавливается закрытой, а соответствующие ссылки обновляются.

Операция рефакторинга Encapsulate Field в надстройке CodeRush Xpress выполняется аналогично, за исключением того, что она автоматически присваивает имя свойству, основываясь на имени переменной класса. Интерфейс для обновления ссылок также отличается, как показано на рис. 8.16. Вместо модального диалогового окна надстройка CodeRush Xpress выводит на экран визуальную подсказку, позволяющую переходить по ссылкам (или перебирать их с помощью клавиши <Tab>). Если требуется замена, разработчик должен поставить метку или нажать клавишу <Enter>. В отличие от диалогового окна, предусмотренного для языка C#, в котором флажки можно было устанавливать и сбрасывать сколько угодно раз, в системе рефакторинга, предусмотренной для языка VB, однажды выбранная замена больше не отменяется.

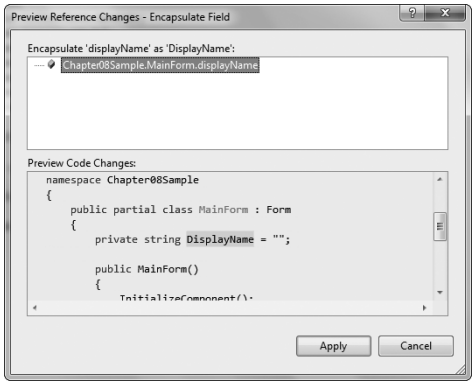


Рис. 8.15

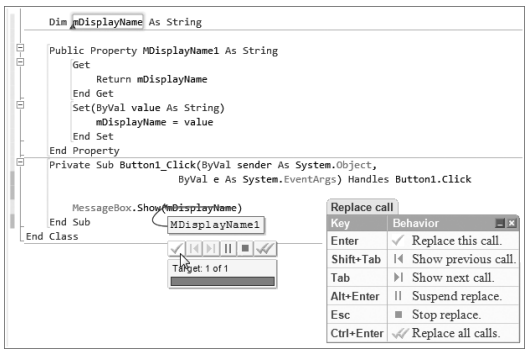


Рис. 8.16

Операция Extract Interface

По мере того как проект проходит этапы развития от прототипа на ранней стадии разработки до полной реализации или стадии роста, часто возникает необходимость выделить основные методы в виде интерфейса, доступного другим приложениям, или определить границы между непересекающимися системами. Прежде для этого приходилось копировать весь метод в новый файл и удалять его содержимое, оставляя лишь заглушку интерфейса. Операция рефакторинга Extract Interface позволяет извлечь интерфейс, используя любое количество методов в классе. При выполнении этой операции открывается диалоговое окно, показанное на рис. 8.17, которое позволяет пользователю выбрать метод, который он хочет включить в интерфейс. После того как разработчик сделает выбор, эти методы добавляются в новый интерфейс. Кроме того, новый интерфейс добавляется в исходный класс.

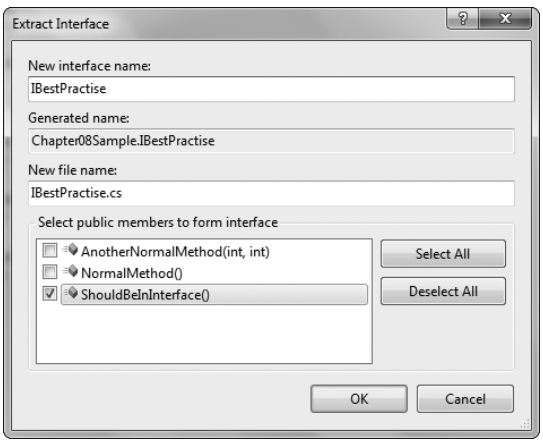


Рис. 8.17

В следующем примере требуется извлечь в отдельный интерфейс первый метод.

C#

```
public class ConcreteClass
{
    public void ShouldBeInInterface()
    { /* ... */ }
    public void AnotherNormalMethod(int ParameterA, int ParameterB)
    { /* ... */ }
    public void NormalMethod()
    { /* ... */ }
}
```

После выбора команды Extract Interface из контекстного меню, появляющегося после щелчка правой кнопкой мыши, и выбора метода ShouldBeInInterface, подлежащего извлечению, в диалоговом окне Extract Interface, в новой файле возникает новый интерфейс, а исходный файл обновляется следующим образом.

C#

```
interface IBestPractice
{
    void ShouldBeInInterface();
}

   public class ConcreteClass: Chapter08Sample.IBestPractice
{
    public void ShouldBeInInterface()
    { /* ... */ }
    public void NormalMethod(int ParameterA, int ParameterB)
    { /* ... */ }
    public void AnotherNormalMethod()
    { /* ... */ }
}
```

Извлечь интерфейс можно также с помощью надстройки CodeRush Xpress, однако она не позволяет указывать, какие именно методы следует включить в интерфейс. В отличие от операции извлечения интерфейса в языке C#, которая размещает интерфейс в отдельном файле (согласно рекомендациям), надстройка CodeRush Xpress просто извлекает все открытые методы класса в отдельный интерфейс в том же самом исходном файле. Например, применим операцию Extract Interface в надстройке CodeRush Xpress к следующему классу.

VB

```
Public Class ConcreteClass
    Public Sub ShouldBeInInterface()
    '...
    End Sub

    Public Sub NormalMethod(ByVal ParameterA As Integer,
                            ByVal ParameterB As Integer)
    '...
    End Sub

    Public Sub AnotherNormalMethod()
    '...
    End Sub
End Class
```

В результате получим следующий код.

VB

```
Public Interface IConcreteClass
    Sub ShouldBeInInterface()
    Sub NormalMethod(ByVal ParameterA As Integer,_
                     ByVal ParameterB As Integer)
    Sub AnotherNormalMethod()
End Interface

Public Class ConcreteClass
    Implements IConcreteClass

Public Sub ShouldBeInInterface() Implements ConcreteClass.ShouldBeInInterface
      '...
    End Sub
```

```
Public Sub NormalMethod(ByVal ParameterA As Integer,
                        ByVal ParameterB As Integer) _
                        Implements IConcreteClass.NormalMethod
      '...
    End Sub

    Public Sub AnotherNormalMethod() Implements
IConcreteClass.AnotherNormalMethod
      '...
    End Sub
End Class
```

Операция Reorder Parameters

Иногда требуется переупорядочить параметры. Часто это делают по “косметическим” причинам, но эта операция может помочь повысить читабельность и иногда является необходимой при реализации интерфейсов.

Диалоговое окно Reorder Parameters dialog, показанное на рис. 8.18, позволяет переставлять параметры в списке в соответствии с требуемым порядком.

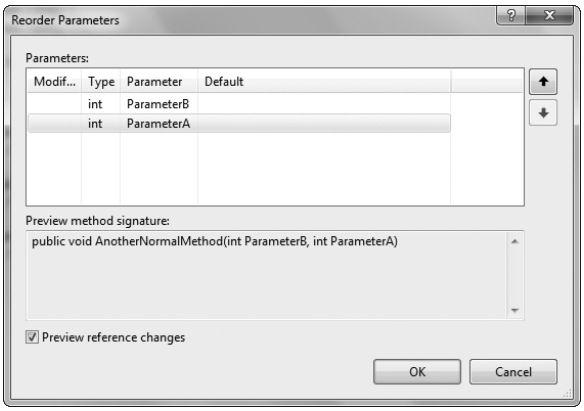


Рис. 8.18

Как только требуемый порядок будет достигнут, пользователь может предварительно просмотреть результат. По умолчанию параметры в каждом вызове данного метода переставляются автоматически в соответствии с новым порядком. Диалоговое окно Preview, аналогичное показанному на рис. 8.15, позволяет контролировать обновление вызовов метода.

Переупорядочение параметров с помощью надстройки CodeRush Xpress имеет более интуитивный характер, чем перестановка параметров в языке C#. Здесь пользователь получает визуальную подсказку, а не диалоговое окно, как показано на рис. 8.19.

Выбранный параметр можно перемещать влево или вправо в списке параметров, а перебирать параметры можно с помощью клавиши <Tab>. Как только параметры будут установлены в правильном порядке, интерфейс поиска и замены, показанный на рис. 8.16, позволит разработчику проверить все изменения.

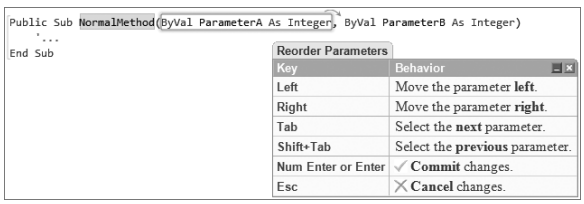


Рис. 8.19

Операция Remove Parameters

Удаление параметра из метода с помощью соответствующей операции рефакторинга значительно сокращает объем поиска ошибок компиляции, которые могут при этом возникнуть. Кроме того, эта операция очень полезна при многочисленных перегрузках метода, при которых удаление параметра не может порождать ошибки компиляции. В этом случае ошибка во время выполнения программы может возникнуть только по семантической, а не по синтаксической причине.

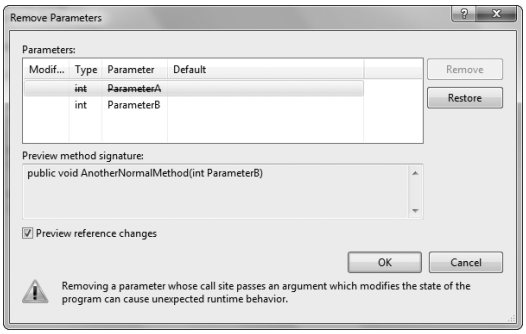


Рис. 8.20

На рис. 8.20 показано диалоговое окно Remove Parameters, которое используется для удаления параметров из списка параметров. Если параметр был удален непреднамеренно, его легко восстановить. Как указывает предупреждение, размещенное в этом диалоговом окне, удаление параметров часто приводит к неожиданным функциональным ошибкам, поэтому важно контролировать внесенные изменения. Для того чтобы оценить внесенные изменения, можно снова использовать окно предварительного просмотра.

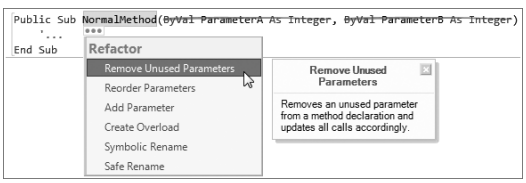


Рис. 8.21

Надстройка CodeRush Xpress поддерживает удаление только неиспользуемых параметров, как показано на рис. 8.21.

Операция Rename

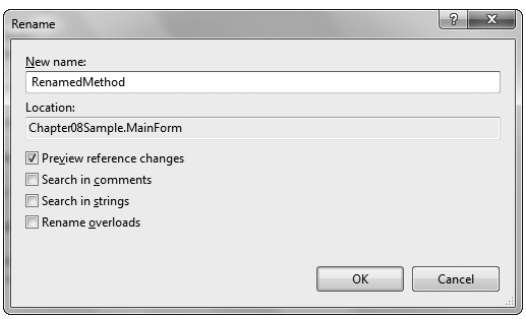


Рис. 8.22

Система Visual Studio 2010 обеспечивает переименование переменных, методов и классов как в языке C#, так и в языке VB. Диалоговое окно Rename для языка C# приведено на рис. 8.22; оно довольно похоже на соответствующее окно для языка VB, хотя в нем нет функций поиска комментариев или строк.

В отличие от поддержки переименования переменных в языке C#, предусматривающей использование окна предварительного просмотра, в котором можно было подтвердить внесенные изменения, функции переименования в языке VB просто переименовывают все ссылки на данную переменную.

Превращение локальной переменной в параметр

Один из наиболее распространенных методов рефакторинга заключается в адаптации существующего метода так, чтобы он получал дополнительный параметр. Превращение локальной переменной в параметр делает метод более общим.

Кроме того, это повышает степень повторного использования кода. Интуитивно ясно, что эта операция должна порождать ошибки компиляции в местах вызова метода. Однако локальная переменная, которая превращается в параметр, должна иметь начальное постоянное значение, которое добавляется как значение параметра во все вызовы метода, чтобы предотвратить любые изменения его функциональных возможностей. Начнем со следующего сниппета и превратим локальную переменную в параметр.

VB

```
Private Sub MethodA()
    MethodB()
End Sub

Private Sub MethodB()
    Dim output As String = "Test String"
    MessageBox.Show(output)
End Sub
```

C#

```
public void MethodA()
{
    MethodB();
}
public void MethodB()
{
    string output = "Test String";
    MessageBox.Show( output);
}
```

После превращения локальной переменной в параметр легко убедиться, что начальное значение было передано через параметр во все места вызова метода.

VB

```
Private Sub MethodA()
    MethodB("Test String")
End Sub

Private Sub MethodB(ByVal output As String)
    MessageBox.Show(output)
End Sub
```

C#

```
public void MethodA()
{
    MethodB("Test String");
}

public void MethodB(string output)
{
    MessageBox.Show( output);
}
```

Операция Generate Method Stub

Когда разработчик пишет код, он может прийти к выводу, что ему нужен вызов метода, который еще не написан. Например, следующий сниппет иллюстрирует новый метод, который должен быть сгенерирован на более поздней стадии.

VB

```
Private Sub MethodA()
    Dim InputA As String
    Dim InputB As Double
    Dim OutputC As Integer = NewMethodIJustThoughtOf(InputA, InputB)
End Sub
```

C#

```
public void MethodA()
{
    string InputA;
    double InputB;
    int OutputC = NewMethodIJustThoughtOf(InputA, InputB);
}
```

Разумеется, этот код порождает ошибку при сборке, поскольку вызываемый метод еще не определен. Используя операцию рефакторинга Generate Method Stub (доступную с помощью интеллектуального указателя в самом коде), можно сгенерировать заглушку метода. Как видно из следующего примера, заглушка метода содержит входные параметры и тип возвращаемого значения.

VB

```
Private Function NewMethodIJustThoughtOf(ByVal InputA As String,
                                         ByVal InputB As Double) As Integer
    Throw New NotImplementedException
End Function
```

C#[1]

```
private int NewMethodIJustThoughtOf(string InputA, double InputB)
{
    throw new Exception("Метод или операция не реализованы.");
}
```

Меню Organize Usings

Очень полезно поддерживать упорядоченный список директив `using` в каждом файле (на языке C#) и ссылаться только на те пространства имен, которые действительно необходимы в данном файле. Команда Organize Usings, доступная в контекстном меню, которое открывается после щелчка пользователем правой кнопкой мыши в окне редактора кода (рис. 8.23), поможет в обоих этих случаях.

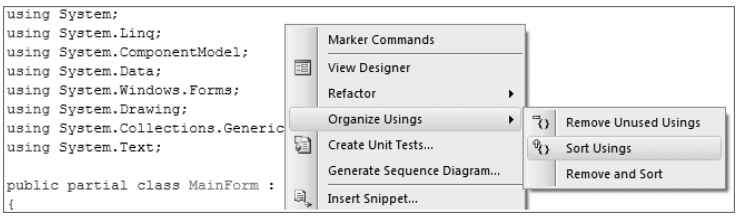


Рис. 8.23

[1] *Комментарии и сообщения в сниппетах оставлены без перевода. – Примеч. ред.*

После рефакторинга кода может выясниться, что в начале файла содержится множество директив `using`, которые больше не используются. Чтобы определить, какие из этих директив используются, а какие нет, вместо метода проб и ошибок можно выполнить операцию, предусмотренную системой Visual Studio. Для этого достаточно щелкнуть правой кнопкой мыши в окне редактора кода и выбрать команду Organize Usings⇨Remove Unused Usings (в языке C#).

Неиспользуемые директивы `using`, их альтернативные имена и внешние альтернативные имена сборок из исходного файла будут удалены.



Разработчики проектов на языке VB не имеют возможности упорядочивать и удалять неиспользуемые инструкции `Imports`. Однако на закладке References в диалоговом окне Project Properties можно пометить пространства имен, которые должны быть импортированы в каждый исходный файл. Это позволит значительно сократить количество инструкций `Imports`. На этой странице можно также удалить ссылки на неиспользуемые сборки.

Очень полезно упорядочивать директивы `using` в алфавитном порядке, чтобы легко управлять пространствами имен, на которые они ссылаются. Для того чтобы не делать это вручную, нужно щелкнуть правой кнопкой мыши в окне редактора кода и выбрать команду Organize Usings⇨Sort Usings.

Директивы `using` для пространства имен `System` указываются первыми, а остальные директивы `using` следуют в алфавитном порядке. Если для пространства имен определено его альтернативное имя, то оно переносится в конец списка, а если в проекте используются альтернативные имена внешних сборок (с помощью ключевого слова `extern` в языке C#), то они перемещаются в начало списка.

Для того чтобы одновременно упорядочить директивы `using` и удалить неиспользуемые, достаточно щелкнуть правой кнопкой мыши в окне редактора кода и выбрать команду Organize Usings⇨Remove and Sort.



Файлы шаблонного кода, предусмотренные в системе Visual Studio по умолчанию, содержат директивы `using` в начале файла за пределами блока пространства имен. Однако, согласно принципам стиля StyleCop, директивы `using` должны содержаться в блоке пространства имен. Операция Organize Usings распознает текущее расположение инструкций `using` в файле и оставляет их на месте.

Резюме

Сниппеты — это ценный инструмент в системе Visual Studio 2010. В главе было описано, как их использовать, как создавать свои собственные сниппеты, включая подстановку переменных (а также инструкции `Imports` и ассоциации ссылок на сниппеты в языке VB). Эта информация позволит читателям создавать свои собственные библиотеки сниппетов для реализации часто используемых функций и сэкономить время на разработку аналогичных конструкций в будущем.

В главе также приведены примеры для каждой операции рефакторинга, предусмотренной в системе Visual Studio 2010. Несмотря на то что разработчики проектов на языке VB не имеют полноценной встроенной поддержки рефакторинга, надстройка CodeRush Xpress предоставляет им широкие возможности для удобного рефакторинга своих проектов.

9

Окно Server Explorer

В ЭТОЙ ГЛАВЕ...

- Запросы на выделение ресурсов аппаратного обеспечения и службы на локальных и удаленных компьютерах
- Использование окна Server Explorer для удобного добавления кода, работающего с компьютерными ресурсами в рамках приложения

Server Explorer — одно из немногих инструментальных окон в системе Visual Studio, не связанных с какими-либо решениями или проектами. Оно позволяет исследовать и запрашивать ресурсы аппаратного обеспечения и службы на локальных или удаленных компьютерах. С его помощью разработчик может решать разные задачи и выполнять многие действия, включая добавление ресурсов в свое приложение.

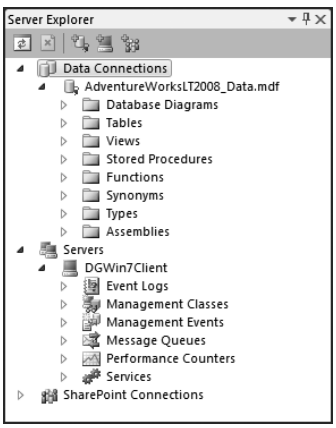


Рис. 9.1

Окно Server Explorer, показанное на рис. 9.1, имеет связь с тремя типами ресурсов. Во-первых, узел Servers обеспечивает доступ к аппаратным ресурсам и службам на локальном или удаленном компьютере. Эта функциональная возможность исследуется в главе очень подробно. Во-вторых, узел Data Connections обеспечивает все аспекты работы с каналами передачи данных, включая создание баз данных, добавление и модификацию таблиц, создание связей между данными и даже выполнение запросов.

Функциональные возможности узла Data Connections подробно описываются в главе 26. В заключение пользователь может добавить связь с сервером SharePoint и просматривать его ресурсы, такие как Content Types, Lists, Libraries и Workflows. Соединения SharePoint подробно описываются в главе 24.

Серверные соединения

Узел Servers лучше было бы назвать Computers, потому что с его помощью можно соединяться с любым компьютером, к которому пользователь имеет доступ, и посылать ему запросы, независимо от того, является ли он сервером или настольной рабочей станцией. Каждый из этих компьютеров представлен в виде отдельного узла, расположенного под узлом Servers. Под узлом компьютера расположен список аппаратного обеспечения, служб и других компонентов, принадлежащих данному компьютеру. Каждый из них предусматривает многочисленные операции или задачи, которые можно выполнить. Некоторые поставщики программного обеспечения поставляют встраиваемые компоненты, которые расширяют функциональные возможности окна Server Explorer.

Для того чтобы открыть окно Server Explorer, необходимо выбрать команду Server Explorer в меню View. Локальный компьютер включается в список Servers по умолчанию. Для того чтобы добавить другой компьютер, следует щелкнуть правой кнопкой мыши на узле Servers и выбрать команду Add Server из контекстного меню.

Ввод имени компьютера или его IP-адреса инициирует попытку связаться с этой машиной, используя мандат пользователя. Если пользователь не имеет достаточных привилегий, он может установить связь, используя другое имя, щелкнув на соответствующем соединении. В исходном состоянии соединение отключено, но если пользователь щелкнет на нем, откроется диалоговое окно, показанное на рис. 9.2, в котором можно ввести альтернативное имя и пароль.



Рис. 9.2



Для доступа к любому серверу необходимы права администратора.

Узел Event Logs

Узел Event Logs открывает доступ к машинным журналам регистрации событий. Для того чтобы запустить утилиту Event Viewer, нужно выбрать соответствующую команду в контекстном меню. Как показано на рис. 9.3, в качестве альтернативы можно открыть список журналов регистрации событий и найти конкретное приложение. Щелкнув на любом из событий, можно вывести на экран информацию о данном событии в окне Properties.

Несмотря на то что окно Server Explorer является полезным инструментом для опрашивания машины при создании программы, его истинная мощь в процессе создания компонентов проявляется, когда пользователь перетаскивает узел ресурса в окно Windows Form. Например, если перетащить узел Application в окно Windows Form, то в область невидимых компонентов проектировщика будет добавлен экземпляр класса `System.Diagnostics.EventLog`. Затем разработчик может сделать запись об этом событии в журнале регистрации событий, используя следующий код.

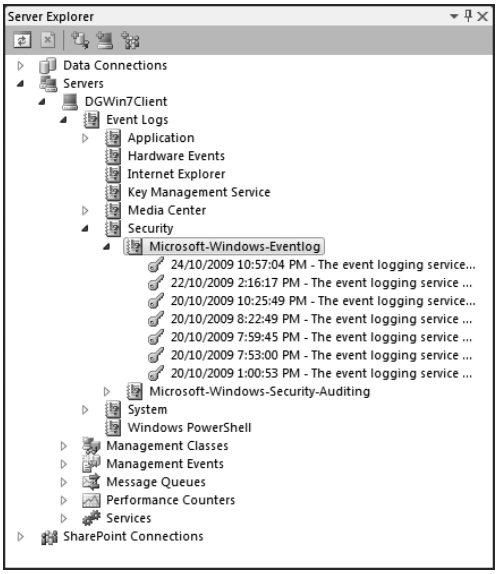


Рис. 9.3

C#

```
this.eventLog1.Source = "My Server Explorer App";
this.eventLog1.WriteEntry("Что-то случилось",
                          System.Diagnostics.EventLogEntryType.Information);
```

Сниппет кода Form1.cs

VB

```
Me.EventLog1.Source = "My Server Explorer App"
Me.EventLog1.WriteEntry("Что-то случилось",
                          System.Diagnostics.EventLogEntryType.Information)
```

Сниппет кода Form1.vb

Поскольку код, приведенный выше, создает новый источник событий Source в журнале Application Event Log, для его выполнения необходимы права администратора. Если при работе под управлением операционной системы Windows Vista или Windows 7 включен механизм управления учетными записями пользователей (User Account Control), то необходимо создать манифест приложения (см. главу 6).

После выполнения этого кода пользователь увидит результаты в окне Server Explorer. Теперь необходимо щелкнуть на кнопке Refresh инструментальной панели Server Explorer, чтобы гарантировать, что новый источник событий (Event Source) будет выведен на экран под узлом Application Event Log.

В качестве альтернативы программисты, работающие на языке Visual Basic, могут добавить в свой код класс `EventLog` и использовать встроенный журнал регистрации событий, принадлежащий пространству имен `My`. Например, можно модифицировать

предыдущий сниппет и создать регистрационную запись с помощью метода My.Application.Log.WriteEntry.

VB

```
My.Application.Log.WriteEntry("Щелчок на кнопке", TraceEventType.Information)
```

Помимо этого, можно также записать информацию об исключении, используя метод `My.Application.Log.WriteException`, который принимает в качестве аргументов исключение и два необязательных параметра, содержащих дополнительную информацию.

Использование пространства имен `My` для записи информации в журнал регистрации событий имеет много дополнительных преимуществ. В следующем файле конфигурации указан объект класса `EventLogTraceListener`, предназначенный для передачи регистрационной информации в журнал регистрации событий. Однако пользователь может задавать других слушателей — например, объекты класса `FileLogTraceListener` , которые записывают *информацию о трассировке* (trace information) в журнал регистрации событий, добавляя их в коллекции `SharedListeners` и `Listeners`.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
     <system.diagnostics>
         <sources>
              <source name="DefaultSource" switchName="DefaultSwitch">
                  <listeners >
                      <add name="EventLog"/>
                  </listeners>
              </source>
         </sources>
         <switches>
              <add name="DefaultSwitch" value="Information"/>
         </switches>
         <sharedListeners>
             <add name="EventLog"
                  type="System.Diagnostics.EventLogTraceListener"
                  initializeData="ApplicationEventLog"/>
         </sharedListeners >
     </system.diagnostics >
</configuration >
```

В этой конфигурации указан также *переключатель* (switch) `DefaultSwitch`. Он связан с источником информации о трассировке посредством атрибута `switchName` и определяет минимальный тип события, которое будет послано перечисленным слушателям. Например, если значение этого переключателя равно `Critical`, то события, имеющие тип `Information`, не будут записаны в журнал регистрации событий. Возможные значения переключателя перечислены в табл. 9.1.

Таблица 9.1 Значения объекта `DefaultSwitch`

Значения по умолчанию	Типы событий, записываемых в журнал
Off	Никакие
Critical	Critical Events
Error	Critical и Error Events
Warning	Critical, Error и Warning Events

Окончание табл. 9.1

Значения по умолчанию	Типы событий, записываемых в журнал
Information	Critical, Error, Warning и Information
Verbose	Critical, Error, Warning, Information и Verbose
ActivityTracing	Start, Stop, Suspend, Resume и Transfer events
All	Все события

Обратите внимание на то, что существуют перегруженные варианты методов `WriteEntry` и `WriteException`, не требующие указания типа событий. В этом случае по умолчанию используются типы `Information` и `Error`.

Классы управления

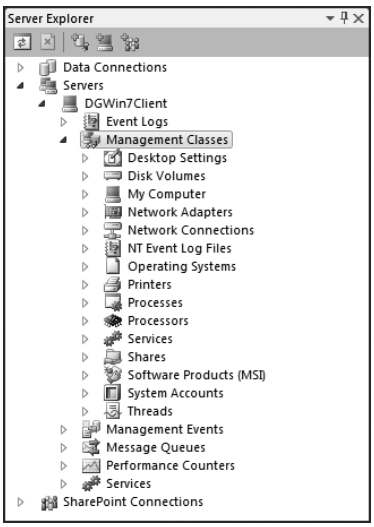


Рис. 9.4

На рис. 9.4 приведен полный список классов управления, доступных в окне Server Explorer. Каждый узел означает набор функциональных возможностей, присущих данному устройству или приложению. Например, щелкнув правой кнопкой мыши на узле Printers, можно добавить новое соединение с принтером, щелкнув правой кнопкой мыши на узле My Computer, можно добавить компьютер в домен или рабочую группу. Все эти узлы обеспечивают строго типизированную оболочку вокруг инфрастуктуры Windows Management Instrumentation (WMI).

В большинстве случаев достаточно перетащить на форму узел, представляющий информацию, которая интересует разработчика. После этого его программа получает доступ к этой информации и возможность ее перерабатывать.

Для того чтобы получить представление о том, как можно использовать эту оболочку, покажем, как с помощью классов управления получить информацию о компьютере. Под узлом My Computer расположен узел с именем локального компьютера. Выбрав этот узел и перетащив его на форму, мы получим компонент `ComputerSystem`, расположенный в невидимой части формы. Теперь добавим элементы управления Label, TextBox, Button и PropertyGrid из закладки All Windows Forms окна Toolbox и разместим их на форме, как показано на рис. 9.5.

Если посмотреть на окно Solution Explorer, то можно увидеть, что в нем тоже появился специальный компонент `root.CIMV2.Win32_ComputerSystem` (или аналогичный, в зависимости от конфигурации компьютера). Он сгенерирован программой Management Strongly Typed Class Generator (`Mgmtclassgen.exe`) и содержит класс `ComputerSystem` и другие классы, позволяющие получить информацию инфраструктуры WMI.

Если щелкнуть на объекте `computerSystem1`, расположенном на форме, в окне Properties отобразится информация об этом компьютере. Однако в данном приложении нас не интересует какой-то конкретный компьютер; этот компьютер был выбран как шаблон для создания класса `ComputerSystem`.

Объект ComputerSystem1 можно удалить, но перед удалением следует обратить внимание на свойство Path этого объекта. Свойство Path в сочетании с именем компьютера внедрено в форму, показанную на рис. 9.5, для того чтобы загрузить информацию об этом компьютере. Это легко увидеть в следующем коде, который является частью обработчика щелчка на кнопке Load.

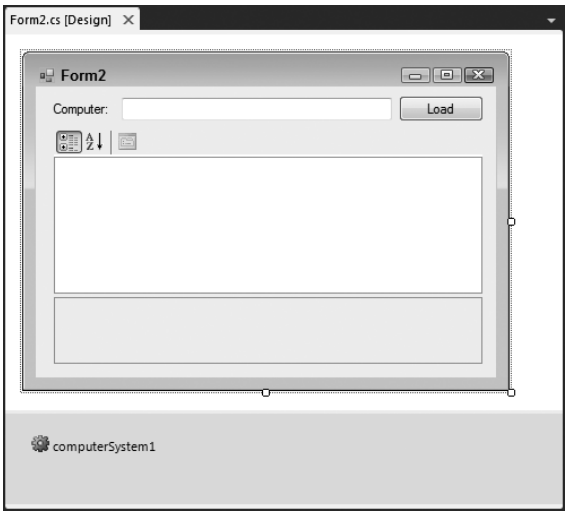


Рис. 9.5

C#

```
const string compPath =
"\\\\{0}\\root\\CIMV2:Win32_ComputerSystem.Name=\"{0}\"";

if (!string.IsNullOrEmpty(this.textBox1.Text))
{
    string computerName = this.textBox1.Text;
    string pathString = string.Format(compPath, computerName);
    var path = new System.Management.ManagementPath(pathString);
    ROOT.CIMV2.ComputerSystem cs = new ROOT.CIMV2.ComputerSystem(path);

    this.propertyGrid1.SelectedObject = cs;
}
```

Сниппет кода Form2.cs

VB

```
Const compPath As String =
"\\{0}\root\CIMV2:Win32_ComputerSystem.Name=""{0}"""

If Not Me.TextBox1.Text = "" Then
    Dim computerName As String = Me.TextBox1.Text
    Dim pathString As String = String.Format(compPath, computerName)
    Dim path As New System.Management.ManagementPath(pathString)
    Dim cs As New ROOT.CIMV2.ComputerSystem(path)

    Me.PropertyGrid1.SelectedObject = cs
End If
```

Сниппет кода Form2.vb

В этом примере свойство `Path`, полученное из объекта `computerSystem1`, использовалось в строковой константе вместе с маркером замены `{0}`, за которым должно следовать имя компьютера. После щелчка на кнопке имя компьютера, вставленное в текстовое поле, объединяется с данным путем с помощью метода `String.Format`, чтобы сгенерировать полный путь WMI. Затем этот путь используется для того, чтобы создать новый объект класса `ComputerAccount`, который в свою очередь передается элементу управления PropertyGrid. Результат этих действий, проявляющийся на этапе выполнения программы, показан на рис. 9.6.

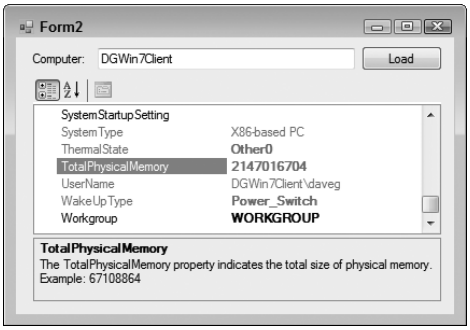


Рис. 9.6

Несмотря на то что большинство свойств являются "только для чтения", для полей, допускающих редактирование, изменения, внесенные в данном элементе PropertyGrid, немедленно передаются компьютеру. Этот сценарий можно изменить, заменив свойство `AutoCommit` в классе `ComputerSystem`.

События управления

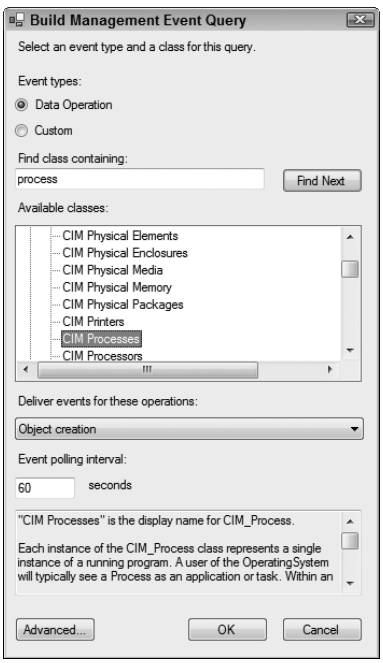


Рис. 9.7

В предыдущем разделе мы видели, как перетащить класс управления из окна Server Explorer на форму и затем работать со сгенерированными классами. Другой способ работы с интерфейсом WMI основывается на использовании узла Management Events. Событие управления позволяет отслеживать любой тип данных интерфейса WMI и генерировать событие при создании, модификации и удалении объектов такого типа. По умолчанию этот узел пуст, но разработчик может создать свой собственный, щелкнув на узле Management Events и выбрав команду Add Event Query, которая открывает окно, показанное на рис. 9.7

С помощью этого окна можно найти тип данных WMI, интересующий разработчика. Поскольку этих типов буквально тысячи, полезно воспользоваться полем Find. На рис. 9.7 показано, что в поле поиска введен термин "`process`" и в результате был найден класс `CIM Processes` в узле `root\CIMV2 node`. Каждый экземпляр этого класса представляет отдельный процесс, выполняемый системой. Разработчик обычно хочет получать информацию, только когда создается новый процесс, поэтому следует выбрать команду Object Creation в выпадающем меню.

После щелчка на кнопке OK к узлу Management Events добавляется узел CIM Processes Event Query. Если разработчик откроет новый экземпляр приложения в своей системе, например Notepad, то увидит события, которые будут последовательно добавлять-

ся в этот узел. В диалоговом окне Build Management Event Query, показанном на рис. 9.7, интервал опроса по умолчанию задан равным 60 секундам, поэтому, для того чтобы увидеть изменения, придется подождать минуту.

Как только событие в конце концов произойдет, оно появится вместе с данными и временем в окне Server Explorer и в окне Output, как показано на рис. 9.8, *снизу*. Если выбрать это событие, то в окне Properties будет содержаться много свойств, не имеющих реального смысла. Однако если разработчик знает, какое свойство искать, то легко найдет его, отфильтрует и сгенерирует в ответ на события системы.

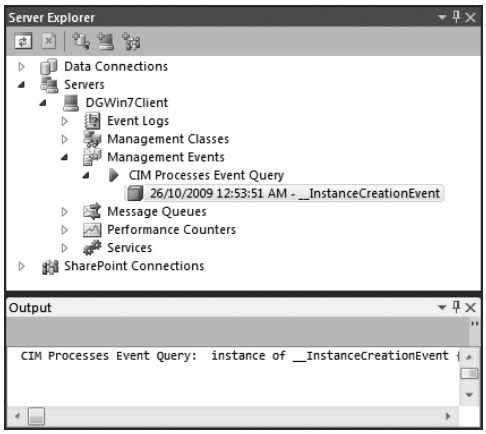


Рис. 9.8

Чтобы продолжить пример, перетащим элементы управления CheckBox и ListBox с панели Toolbox на новую форму типа Windows Form. Затем перетащим узел CIM Processes Event Query из окна Server Explorer на новую форму. В результате будет сгенерирован экземпляр класса `System.Management.ManagementEventWatcher`, свойства которого настроены так, чтобы отслеживать создание нового процесса. Доступ к реальному запросу можно получить с помощью свойства `QueryString` вложенного объекта `ManagementQuery`. Как и большинство классов-наблюдателей, класс `ManagementEventWatch` генерирует событие, когда выполняются конкретные условия, — в данном случае событие `EventArrived`. Для обработки этого события в программу добавляется следующий код.



C#

```
private void managementEventWatcher1_EventArrived(System.Object sender,
                                System.Management.EventArrivedEventArgs e)
{
    foreach (System.Management.PropertyData p in e.NewEvent.Properties)
    {
        if (p.Name == "TargetInstance")
        {
            var mbo = (System.Management.ManagementBaseObject)p.Value;
            string[] sCreatedProcess =
                     {(string)mbo.Properties["Name"].Value,
                      (string)mbo.Properties["ExecutablePath"]. Value };
            this.BeginInvoke(new LogNewProcessDelegate(LogNewProcess),
                       sCreatedProcess);
        }
    }
}
delegate void LogNewProcessDelegate(string ProcessName, string ExePath);
private void LogNewProcess(string ProcessName, string ExePath)
{
    this.listBox1.Items.Add(string.Format("{0} — {1}", ProcessName,
                            ExePath));
}
```

```
private void checkBox1_CheckedChanged(System.Object sender,
System.EventArgs e)
{
    if (this.checkBox1.Checked)
    {
        this.managementEventWatcher1.Start();
    }
    else
    {
        this.managementEventWatcher1.Stop();
    }
}
```

Сниппет кода Form3.cs



VB

```
Private Sub ManagementEventWatcher1_EventArrived(ByVal sender As
System.Object,_
            ByVal e As System.Management.EventArrivedEventArgs)
    For Each p As System.Management.PropertyData In e.NewEvent.Properties
        If p.Name = "TargetInstance" Then
            Dim mbo As System.Management.ManagementBaseObject = _
            CType(p.Value, System.Management.ManagementBaseObject)
            Dim sCreatedProcess As String() =
            {mbo.Properties("Name").Value, _
             mbo.Properties("ExecutablePath").Value}
             Me.BeginInvoke(New LogNewProcessDelegate(AddressOf
             LogNewProcess), _
                           sCreatedProcess)
        End If
    Next
End Sub

Delegate Sub LogNewProcessDelegate(ByVal ProcessName As String, _
                                   ByVal ExePath As String)
Private Sub LogNewProcess(ByVal ProcessName As String,
                          ByVal ExePath As String)
    Me.ListBox1.Items.Add(String.Format("{0} — {1}", ProcessName,
                          ExePath))
End Sub

Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, _
                                     ByVal e As System.EventArgs) _
                                     Handles CheckBox1.CheckedChanged
    If Me.CheckBox1.Checked Then
        Me.ManagementEventWatcher1.Start()
    Else
        Me.ManagementEventWatcher1.Stop()
    End If
End Sub
```

Сниппет кода Form3.vb

Обработчик события должен просмотреть коллекцию `Properties`, связанную с объектом `NewEvent`. Если объект был изменен, то возвращаются два экземпляра: `PreviousInstance` , в котором хранится состояние объекта в начале интервала опроса, и экземпляр `TargetInstance`, в котором хранится состояние в конце этого интервала. В течение интервала опроса состояние объекта может изменяться несколько раз. В этом случае событие может произойти, только если в конце периода оно отличается от состояния в начале периода. Например, если

процесс начался и закончился в течение одного и того же интервала опроса, то событие не произойдет.

Обработчик событий создает новый объект `ManagementBaseObject` по значению, которое передается в качестве аргумента события, чтобы получить отображаемое имя и путь к исполняемому файл, запустившему процесс. Поскольку элементы управления пользовательского интерфейса можно обновить только с помощью потока пользовательского интерфейса, разработчик не может непосредственно обновить объект `ListBox`. Вместо этого он должен вызвать метод `BeginInvoke`, чтобы применить функцию `LogNewProcess` к потоку пользовательского интерфейса. Состояние формы во время этого действия показано на рис. 9.9

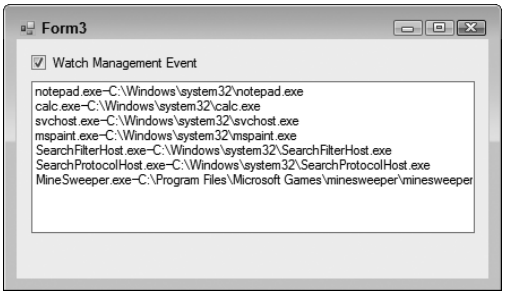


Рис. 9.9

Узел Message Queues

Узел Message Queues, открытый на рис. 9.10, дает разработчику доступ к очередям событий, возможных на его компьютере. Существуют три типа очередей: *закрытая* (private), которая не возникает по внешнему запросу; *открытая* (public), которая возникает по внешнему запросу; и *системная* (system), которая используется для неотправленных сообщений и сообщений о других исключительных событиях.



Для того чтобы использовать узел Message Queues, необходимо убедиться, что на компьютере установлено свойство MSMQ. Это можно сделать с помощью подраздела Programs and Features раздела Control Panel. Для того чтобы включить свойство Microsoft Message Queue (MSMQ) Server, необходимо выполнить команду Turn Windows Features On или Off и установить флажок Microsoft Message Queue (MSMQ) Server.

На рис. 9.10 очередь сообщений `samplequeue` добавлена в узел Private Queues после выбора команды Create Queue из контекстного меню, открывающегося щелчком правой кнопкой мыши. Создав очередь и перетащив ее в новую форму типа Windows Form, можно создать соответственно настроенный экземпляр класса `MessageQueue`. Для того чтобы продемонстрировать функциональные возможности объекта класса `MessageQueue`, добавим в форму два поля ввода и кнопку, как показано на рис. 9.11. Кнопка Send заставляет объект класса `MessageQueue` послать сообщение, введенное в первом поле ввода. Если с формой произойдет событие `Load`, то будет создан фоновый поток, непрерывно опрашивающий очередь в ожидании сообщения, которое будет выведено во втором поле.

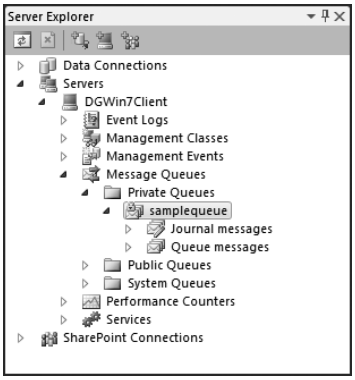


Рис. 9.10

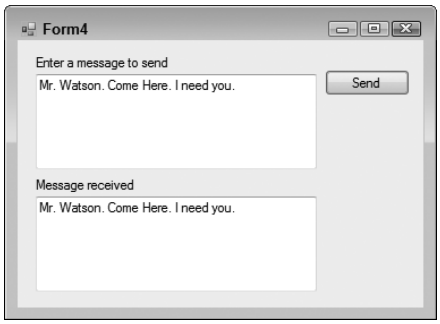


Рис. 9.11

C#

```
public Form4()
{
    InitializeComponent();
    var monitorThread = new System.Threading.Thread(MonitorMessageQueue);
    monitorThread.IsBackground = true;
    monitorThread.Start();
    this.Button1.Click +=new EventHandler(btn_Click);
}

private void btn_Click(object sender, EventArgs e)
{
    this.messageQueue1.Send(this.TextBox1.Text);
}

private void MonitorMessageQueue()
{
    var m = default(System.Messaging.Message);
    while (true)
    {
        try
        {
           m = this.messageQueue1.Receive(new TimeSpan(0, 0, 0, 0, 50));
           this.ReceiveMessage((string)m.Body);
        }
        catch (System.Messaging.MessageQueueException ex)
        {
           if (!(ex.MessageQueueErrorCode ==
                  System.Messaging.MessageQueueErrorCode.IOTimeout))
           {
               throw ex;
            }
        }
        System.Threading.Thread.Sleep(10000);
    }
}

private delegate void MessageDel(string msg);
private void ReceiveMessage(string msg)
{
    if (this.InvokeRequired)
```

```
    {
        this.BeginInvoke(new MessageDel(ReceiveMessage), msg);
        return;
    }
    this.TextBox2.Text = msg;
}
```

Сниппет кода Form4.cs



VB

```
Private Sub Form_Load(ByVal sender As Object, ByVal e
                      As System.EventArgs) _
                      Handles Me.Load
    Dim monitorThread As New Threading.Thread(AddressOf
MonitorMessageQueue)
    monitorThread.IsBackground = True
    monitorThread.Start()
End Sub

Private Sub btn_Click(ByVal sender As System.Object,
                      ByVal e As System.EventArgs)
                      Handles Button1.Click
    Me.MessageQueue1.Send(Me.TextBox1.Text)
End Sub

Private Sub MonitorMessageQueue()
    Dim m As Messaging.Message
    While True
        Try
            m = Me.MessageQueue1.Receive(New TimeSpan(0, 0, 0, 0, 50))
            Me.ReceiveMessage(m.Body)
        Catch ex As Messaging.MessageQueueException
            If Not ex.MessageQueueErrorCode = _
                   Messaging.MessageQueueErrorCode.IOTimeout Then
                Throw ex
            End If
        End Try
        Threading.Thread.Sleep(10000)
    End While
End Sub

Private Delegate Sub MessageDel(ByVal msg As String)
Private Sub ReceiveMessage(ByVal msg As String)
    If Me.InvokeRequired Then
        Me.BeginInvoke(New MessageDel(AddressOf ReceiveMessage), msg)
        Return
    End If
    Me.TextBox2.Text = msg
End Sub
```

Сниппет кода Form4.vb

Обратите внимание на то, что в этом сниппете фоновый поток нигде явно не закрывается. Поскольку свойство `IsBackground` этого потока установлено равным `True`, он автоматически закрывается, когда приложение прекращает работу. Как и в предыдущем примере, поскольку обработка сообщения происходит в фоновом потоке, разработчик должен переключить потоки при обновлении пользовательского интерфейса с помощью метода `BeginInvoke`. Учитывая все вышесказанное, получаем форму, приведенную на рис. 9.11.

Как только сообщение будет послано в очередь сообщений, оно появится в соответствующей очереди в окне Server Explorer. Щелкнув на этом сообщении, можно увидеть его содержимое в окне Properties.

Узел Performance Counters

При создании приложения разработчики очень часто забывают о его поддержке и управлении. Например, рассмотрим приложение, установленное год назад и работавшее без каких-либо проблем. Внезапно запросы стали обрабатываться неприемлемо долго. Очевидно, что приложение стало работать неправильно, но нет никакого способа, чтобы определить причину ошибки. Одна из стратегий, направленных на идентификацию таких проблем, основана на использовании счетчиков производительности. Система Windows имеет много встроенных счетчиков производительности, которые можно использовать для контроля за ее деятельностью. Кроме того, существует множество программного обеспечения, разработанного сторонними организациями, которые инсталлируют администраторов производительности и могут идентифицировать любое неправильное функционирование системы.

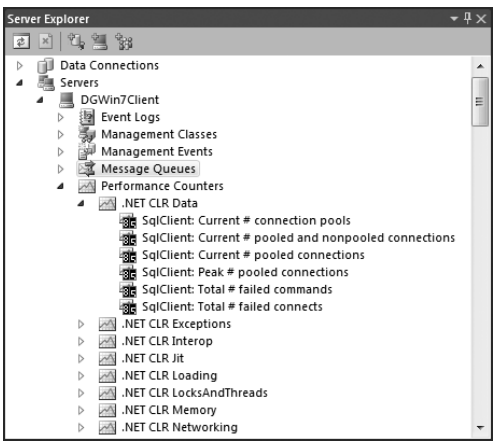


Рис. 9.12

Узел Performance Counters в дереве Server Explorer, продемонстрированный в раскрытом виде на рис. 9.12, имеет две основные функции. Во-первых, он позволяет увидеть и извлечь информацию об инсталлированных счетчиках, создавать новые счетчики производительности, а также редактировать и удалять существующие. Как показано на рис. 9.12, под узлом Performance Counters расположен список категорий, а еще ниже — список счетчиков.



Для того чтобы увидеть узел Performance Counters в окне Server Explorer, необходимо запустить систему Visual Studio, имея права администратора.

Для того чтобы отредактировать категорию или счетчик, следует выбрать команду Edit Category из контекстного меню, которое появляется после щелчка правой кнопкой мыши на имени категории. Для того чтобы добавить новую категорию и связанные с ней счетчики, надо щелкнуть правой кнопкой на узле Performance Counters и выбрать команду Create New Category в контекстном меню. Обе эти операции используют диалоговое окно, показанное на рис. 9.13. Оно демонстрирует, что пользователь создал новую категорию счетчиков, которая будет использоваться для отслеживания событий, связанных с открытием и закрытием формы.

Во-вторых, раздел Performance Counters обеспечивает простой доступ к счетчикам производительности из кода. Перетащив категорию счетчиков производительности на форму, разработчик получает возможность читать информацию из счетчика производительности и записывать информацию в него. Продолжим пример, перетащив на новую форму типа Windows Form новые счетчики производительности My Application, Form Open и Form Close. Добавим также поля ввода и кнопку, с помощью которых будем выводить значения счетчиков производительности. И наконец, переименуем счетчики производительности, чтобы они имели понятные имена. В результате получим форму, показанную на рис. 9.14.

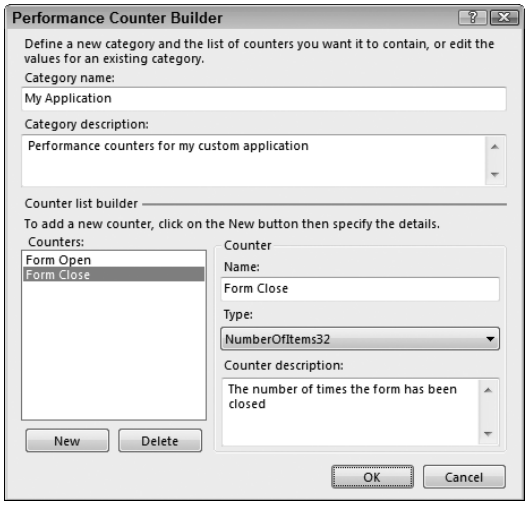


Рис. 9.13

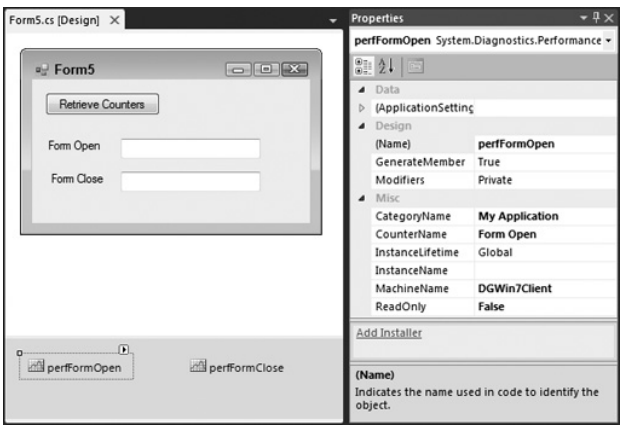


Рис. 9.14

Среди свойств выбранного счетчика производительности можно увидеть категорию, к которой он относится: в данном случае счетчик `Form Close` был выбран из категории My Application. Отметим также свойство `MachineName`, которое содержит информацию о компьютере, от которого счетчик получает информацию, и свойство `ReadOnly`, которое должно быть установлено равным `False`, если счетчик требуется обновить. (По умолчанию свойство `ReadOnly` установлено равным `True`.) Для того чтобы закончить создание формы, добавим на нее кнопку Retrieve Counters для обработчика событий.



C#

```
this.textBox1.Text = this.perfFormOpen.RawValue.ToString();
this.textBox2.Text = this.perfFormClose.RawValue.ToString();
```

Сниппет кода Form5.cs

VB

```
Me.textBox1.Text = Me.perfFormOpen.RawValue
Me.textBox2.Text = Me.perfFormClose.RawValue
```

Сниппет кода Form5.vb

Пользователь может добавить в приложение код, обновляющий счетчики производительности. Например, можно добавить следующий код для обработчиков события `Form Load`.

C#

```
this.perfFormOpen.Increment();
```

VB

```
Me.perfFormOpen.Increment()
```

Во время перетаскивания счетчика производительности на форму можно заметить его интеллектуальный дескриптор (маленькая стрелка, которая появляется в правом верхнем углу при выборе элемента управления), содержащий один элемент: `Add Installer`. Когда пользователь выбирает элемент управления (см. рис. 9.14), имя соответствующей операции появляется в нижней части окна Properties. Щелкнув на имени операции в любом из этих мест, разработчик может добавить в свое решение класс `Installer`, позволяющий инсталлировать счетчик производительности в ходе установки приложения. Разумеется, для того чтобы иметь возможность вызывать объект класса `Installer`, сборка, которой он принадлежит, должна быть добавлена как пользовательская операция в проект развертывания. Более подробная информация о пользовательских операциях содержится в главе 48.

До появления версии Visual Studio 2008 разработчик должен был вручную модифицировать инсталлятор, чтобы создать несколько счетчиков производительности. В текущей версии можно просто выбрать каждый из добавляемых счетчиков производительности и щелкнуть на компоненте Add Installer. Система Visual Studio 2010 вернет разработчика к первому созданному инсталлятору, и в коллекцию `Counters` компонента `PerformanceCounterInstaller` автоматически будет добавлен второй счетчик (рис. 9.15).

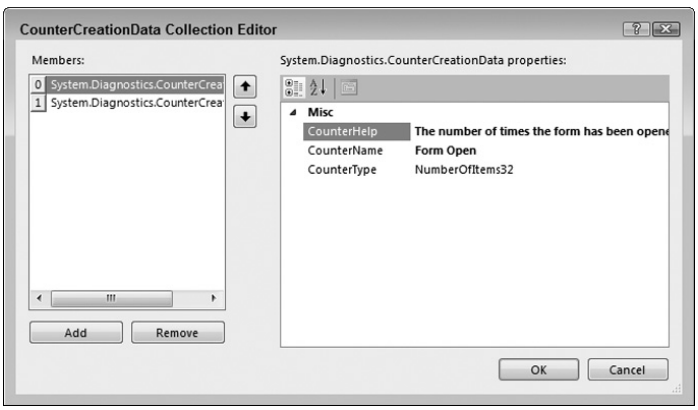


Рис. 9.15

Добавляя компоненты `PerformanceCounterInstaller` в область проектирования, разработчик может добавлять в проект счетчики производительности других категорий. Теперь приложение можно развертывать, зная, что в дальнейшем ее функционирование можно контролировать с помощью таких инструментов, как `PerfMon`.

Узел Services

Узел Services, показанный на рис. 9.16, содержит зарегистрированные службы для компьютера. Каждый узел дерева, корнем которого является узел Services, отображает состояние этой службы в левом нижнем углу пиктограммы. Возможны состояния `Stopped`, `Running` и `Paused`. При выборе службы в окне Properties отображается дополнительная информация о нем, например, о его связях с другими службами.

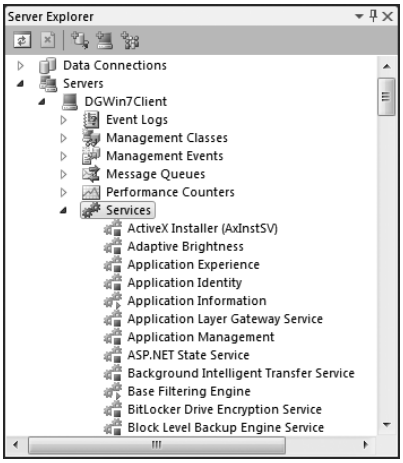


Рис. 9.16

Как и другие узлы в дереве Server Explorer, каждую службу можно перетащить на форму. При этом в невидимой области формы генерируется компонент `ServiceController`. По умолчанию свойство `ServiceName` устанавливается равным имени службы, которая перетаскивается из окна Server Explorer, однако его можно изменить, чтобы получить информацию и управлять любой службой. Аналогично, свойство `MachineName` можно изменить, чтобы связаться с любым доступным компьютером. Следующий код демонстрирует, как можно остановить службу, используя компонент `ServiceController`.



C#

```
this.serviceController1.Refresh();
if (this.serviceController1.CanStop)
{
    if (this.serviceController1.Status ==
              System.ServiceProcess.ServiceControllerStatus.Running)
    {
        this.serviceController1.Stop();
        this.serviceController1.Refresh();
    }
}
```

Сниппет кода Form6.cs

VB

```
Me.ServiceController1.Refresh()
If Me.ServiceController1.CanStop Then
   If Me.ServiceController1.Status = _
         ServiceProcess.ServiceControllerStatus.Running Then
      Me.ServiceController1.Stop()
      Me.ServiceController1.Refresh()
   End If
End If
```

Сниппет кода Form6.vb

Кроме трех основных состояний — `Running`, `Paused` и `Stopped`, — служба может иметь промежуточные состояния: `ContinuePending`, `PausePending`, `StartPending` и `StopPending`. Если необходимо запустить службу, которая может зависеть от другой службы, т.е. пребывает в одном из промежуточных состояний, можно вызвать метод `WaitForStatus`.

Соединения с источниками данных

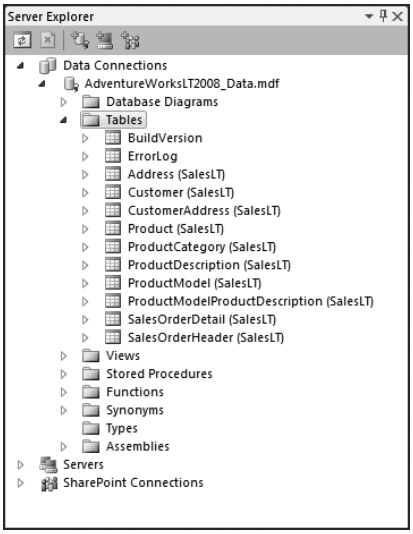


Рис. 9.17

Узел Data Connections позволяет связаться с базой данных и выполнить множество административных функций. Разработчик может связаться с широким спектром источников данных, включая все версии баз данных SQL Server, Microsoft Access, Oracl или ODBC. На рис. 9.17 показана связь программы Server Explorer с файлом базы данных SQL Server.

Программа Server Explorer обеспечивает доступ к набору инструментов Visual Database, позволяющих выполнять разнообразные административные операции над присоединенными базами данных. Разработчик может создавать базы данных, добавлять и модифицировать таблицы, представления и хранимые процедуры, управлять индексами, выполнять запросы и многое другое. Все аспекты узла Data Connections описаны в главе 26.

Соединения SharePoint

Возможность соединения с программой Microsoft Office SharePoint Server с помощью программы Server Explorer является новым свойством системы Visual Studio 2010. Благодаря ему пользователь может перемещаться по многочисленным источникам и компонентам SharePoint и просматривать их.

Программа Server Explorer позволяет только читать источники из библиотеки SharePoint, — пользователи не могут, например, создавать или редактировать определение списка. Но даже в этом режиме может оказаться полезной возможность прочитать эту информацию в системе Visual Studio при разработке приложения, работающего с библиотекой SharePoint. Как и многие компоненты, существующие в окне Servers Node, некоторые компоненты типа SharePoint можно перетаскивать и оставлять прямо в области проектирования приложения, работающего с библиотекой SharePoint.

Использование программы Server Explorer для просмотра ресурсов типа SharePoint подробно описано в главе 24.

Резюме

В главе показано, как с помощью программы Server Explorer управлять компьютерными ресурсами и службами. Более подробное изучение программы Server Explorer и узла Data Connections будет продолжено в главе 22. Углубленному исследованию аспектов управления ресурсами библиотеки SharePoint с помощью программы Server Explorer посвящена глава 24.

10

Моделирование с помощью утилиты Class Designer

В ЭТОЙ ГЛАВЕ...

- Использование утилиты Class Designer для графической визуализации архитектуры пользовательских классов
- Простая генерация и рефакторинг пользовательских классов с помощью утилиты Class Designer
- Использование надстройки Modeling Power Toys системы Visual Studio 2010 для более эффективной работы крупных иерархий классов

Традиционно моделирование программного обеспечения осуществляется отдельно от кодирования. Как правило, это происходит на этапе проектирования, который заканчивается до начала кодирования. В большей части случаев диаграммы моделирования, построенные на этапе проектирования, быстро устаревают в ходе разработки программного обеспечения и теряют свое значение при неизбежном изменении проекта.

Утилита Class Designer системы Visual Studio 2010 позволяет внедрить моделирование в интегрированную среду разработки в виде действий, которые можно выполнить в любое время на всем протяжении разработки проекта. Диаграммы классов динамически конструируются по исходному коду, т.е. они никогда не устаревают. Любые изменения, внесенные в исходном коде, немедленно отражаются на диаграмме классов, а любое изменение диаграммы классов отражается в коде.

В главе детально описывается утилита Class Designer, позволяющая разрабатывать, визуализировать и проводить рефакторинг архитектуры классов.

Создание диаграммы классов

Процесс проектирования приложения обычно подразумевает как минимум создание схемы классов, которые будут созданы, и описание их взаимодействия. Система Visual Studio 2010 имеет область проектирования Class Designer, в которой можно формировать диаграмму классов. Затем в классы можно добавлять поля, свойства и методы, а также устанавливать взаимоотношения между ними.

Несмотря на то что этот объект называется диаграммой классов, он поддерживает классы, структуры, перечисления, интерфейсы, абстрактные классы и делегатов. Существует несколько способов, с помощью которых разработчик может добавлять диаграмму классов в свой проект. Например, он может открыть диалоговое окно Add New Item (рис. 10.1), чтобы создать новую пустую диаграмму классов в проекте.

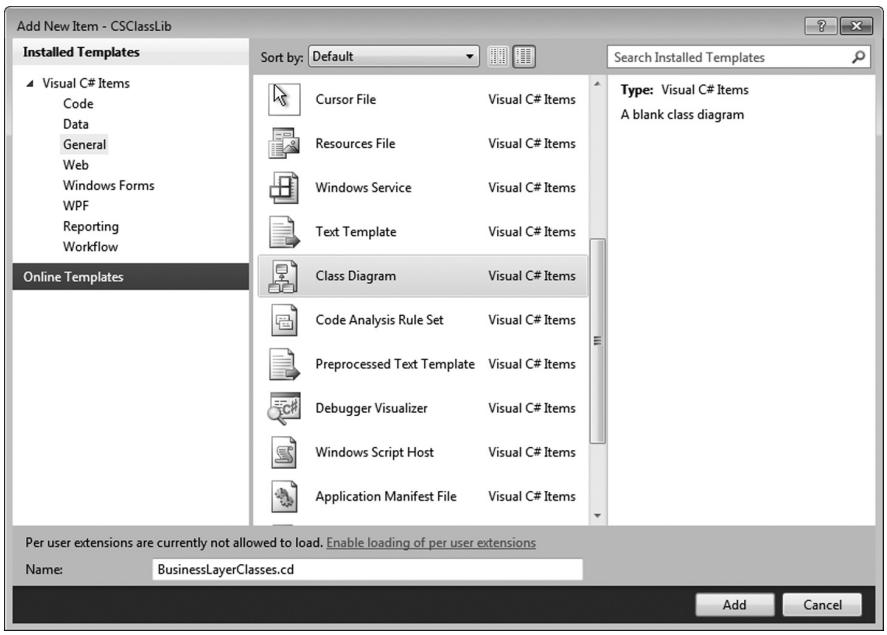


Рис. 10.1

Новую диаграмму классов можно добавить в проект, щелкнув на кнопке View Class Diagram инструментальной панели в окне Solution Explorer или щелкнув правой кнопкой мыши на проекте или классе, а затем выбрав команду View Class Diagram в контекстном меню. Если при этом проект был выделен, то система Visual Studio автоматически добавит все типы, определенные в проекте, в начальную диаграмму классов. Хотя в некоторых случаях это может оказаться удобным, для проекта, содержащего большое количество классов, процесс создания и уточнения такой диаграммы может оказаться слишком долгим.

В отличие от некоторых программных средств, требующих, чтобы все типы, существующие в проекте, принадлежали одной и той же диаграмме классов, в системе Visual Studio диаграмма классов может содержать сколько угодно пользовательских типов. Это позволяет добавлять в одно решение несколько диаграмм классов.



Область действия утилиты Class Designer ограничена одним проектом. Например, нельзя добавлять в диаграмму классов типы, определенные в разных проектах, даже если они являются частями одного и того же решения.

Надстройку Class Designer можно разделить на четыре компонента: *область проектирования* (design surface), инструментальная панель Toolbox, окно Class Details и сетка свойств. Изменения диаграммы классов сохраняются в файле с расширением `.cd`, который в сочетании с исходными файлами используется для генерации схемы, демонстрируемой в области проектирования Class Designer.

Область проектирования

Область проектирования надстройки Class Designer позволяет разработчику взаимодействовать с типами, используя перетаскивание компонентов. Разработчик может добавлять в область проектирования существующие типы, перетаскивая их из любого представления классов или окна Solution Explorer. Если файл, связанный с окном Solution Explorer, содержит несколько типов, то все они добавляются в область проектирования.

На рис. 10.2 показана простая диаграмма классов, содержащая два класса: `Customer` и `Order`, а также перечисление `OrderStatus`. Каждый класс содержит поля, свойства, методы и события. Между классами существует ассоциация, поскольку класс `Customer` содержит свойство `Orders`, представляющее собой список объектов типа `Order`, а класс `Order` реализует интерфейс `IDataErrorInfo`. Вся эта информация содержится на диаграмме классов.

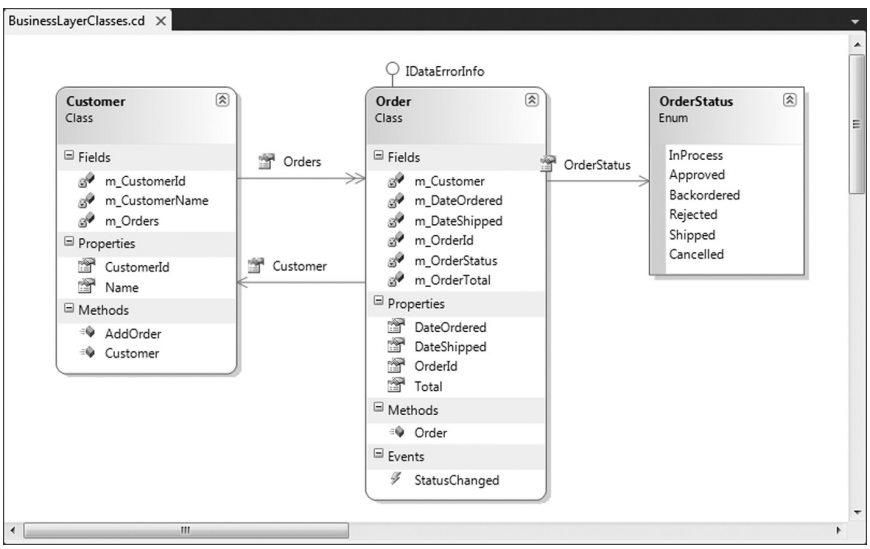


Рис. 10.2

Каждый класс на диаграмме классов изображается как отдельная сущность, которую можно перетаскивать по области проектирования и при необходимости изменять ее размеры. Класс состоит из полей, свойств, методов и событий. На рис. 10.2

эти компоненты сгруппированы в разделы. Разработчик может изменять шаблоны для диаграмм классов, например, перечислять компоненты в алфавитном порядке или группировать их по уровню доступа.

Надстройка Class Designer часто используется для просмотра многочисленных классов, чтобы было понятно, как они связаны друг с другом. В таком случае удобно скрывать компоненты классов, чтобы упростить диаграмму. Для того чтобы скрыть все компоненты одновременно, можно использовать переключатель, расположенный в правом верхнем углу представления класса в области проектирования. Если необходимо скрыть не все, а лишь некоторые компоненты, то их можно скрывать по отдельности или по разделам, щелкая правой кнопкой мыши на соответствующем элементе и выбирая в контекстном меню команду Hide.

Инструментальная панель Toolbox

Для того чтобы облегчить добавление элементов на диаграмму классов, в инструментальной панели Toolbox есть корешок закладки Class Designer. Чтобы создать элемент, следует перетащить его из инструментальной панели Toolbox в область проектирования и дважды щелкнуть на нем. На рис. 10.3 показана инструментальная панель Toolbox с видимой закладкой Class Designer. Элементы на панели Toolbox можно разделить на *сущности* (entities) или *соединители* (connectors). Обратите внимание на то, что элемент Comment можно добавить на закладку Class Designer, но он не появляется ни в одной части кода, а просто добавляется в документацию о диаграмме классов.

Сущности

Сущности, которые можно добавить на диаграмму классов, полностью соответствуют типам в технологии .NET Framework. Добавляя новую сущность в область проектирования, необходимо присвоить ей имя. Кроме того, требуется указать, в какой файл ее следует включить: в новый или существующий.

Для того чтобы удалить сущности из диаграммы, надо щелкнуть на них правой кнопкой мыши и выбрать в контекстном меню команду Remove From Diagram. В этом случае они не удаляются из исходного кода, а исчезают из диаграммы классов. Если разработчик все же хочет удалить связанный с сущностью исходный код, следует выбрать команду Delete Code.

Для того чтобы увидеть код, связанный с сущностью, следует дважды щелкнуть на сущности или выбрать команду View Code в контекстном меню, которое открывается после щелчка правой кнопкой мыши. Сущности, включенные в инструментальную панель Toolbox, описаны ниже.

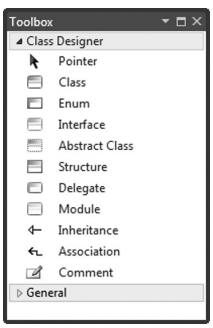


Рис. 10.3

- **Класс.** Поля, свойства, методы, события и константы можно добавлять в класс с помощью контекстного меню, открывающегося после щелчка правой кнопкой мыши или с помощью окна Class Details. Несмотря на то что класс может поддерживать вложенные типы, они не могут добавляться в область проектирования. Класс могут также реализовывать интерфейсы. Например, на рис. 10.2 класс `Order` реализует интерфейс IDataErrorInfo.

- **Перечисление.** Может содержать только список членов с присвоенными им значениями. Каждый метод имеет резюме и комментарии, но они появляются только в виде XML-комментария, расположенного напротив метода.
- **Интерфейс.** Интерфейсы определяют свойства, методы и события, которые должен реализовать класс. Интерфейсы могут также содержать вложенные типы, но напомним, что добавление вложенных типов не поддерживается утилитой Class Designer.
- **Абстрактный класс.** Абстрактные классы не отличаются от обычных классов, за исключением того, что их имена в области проектирования набираются курсивом и снабжаются меткой `Abstract` (в языке C#) или `MustInherit` (в языке VB).
- **Структура.** Это единственная сущность, кроме комментариев, которая появляется в области проектирования в прямоугольнике. Аналогично классу, структура имеет поля, свойства, методы, события и константы. Кроме того, она тоже может содержать вложенные типы. Тем не менее в отличие от класса структура не имеет деструктора.
- **Делегат.** Несмотря на то что делегат появляется на диаграмме классов как сущность, он не может содержать вложенные типы. Он может содержать только параметры, определяющие его сигнатуру.

Соединители

Между сущностями могут быть установлены отношения двух типов. Они иллюстрируются на диаграмме классов с помощью соединителей, описанных ниже.

- **Inheritance (Наследование).** Соединитель `Inheritance` используется для того, чтобы показать отношение между классами, наследующими свойства друг друга.
- **Association (Ассоциация).** Если класс содержит ссылку на другой класс, то между ними существует ассоциация. Она демонстрируется с помощью соединителя `Association`. Если отношение основано на коллекции, — например, на списке объектов типа `Order`, — его можно представить с помощью *ассоциации коллекции* (collection association). Ассоциация коллекции `Orders` показана на рис. 10.2 в виде соединителя, связывающего классы `Customer` и `Order`.

Ассоциацию классов (*class association*) можно представить либо в виде поля или свойства класса, либо в виде ассоциативной связи между классами. Для этого можно использовать контекстное меню, открывающееся после щелчка правой кнопкой мыши на поле, свойстве или линии связи между классами.



Для того чтобы показать свойство как ассоциацию коллекции, необходимо щелкнуть правой кнопкой мыши на классе и выбрать команду Show as Collection Association. В результате свойство будет скрыто в классе и показано в виде соединителя с ассоциированным классом на диаграмме.

Окно Class Details

Компонент можно добавить в сущность, щелкнув правой кнопкой мыши и выбрав нужный компонент. К сожалению, это довольно долгий процесс, не позволяющий добавлять параметры метода или возвращаемые значения. Надстройка Class Designer

в системе Visual Studio 2010 содержит окно Class Details, обеспечивающее пользовательский интерфейс, позволяющий быстро вставлять компоненты. Это окно показано на рис. 10.4 на примере класса `Customer`, ранее продемонстрированного на рис. 10.2.

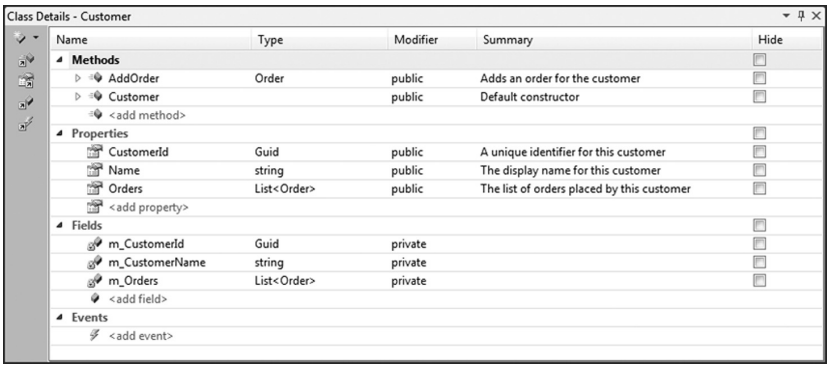


Рис. 10.4

В левой части окна расположены кнопки, которые могут помочь в управлении классами, содержащими большое количество компонентов. Верхнюю кнопку можно использовать для того, чтобы добавлять в класс методы, свойства, поля или события. Остальные кнопки предназначены для перемещения фокуса по группам. Например, вторая кнопка позволяет перейти к списку методов класса. Перемещаться по этому списку можно с помощью клавиш со стрелками, направленными вверх и вниз.

Как показано на рис. 10.4, основная область окна разделяется на четыре списка, упорядоченных по алфавиту: Methods, Properties, Fields и Events. Сущности других типов могут содержать другие компоненты, например Members и Parameters. Каждая строка разделяется на пять столбцов, содержащих имя, тип возвращаемого значения, уровень доступа, резюме, а также индикатор того, скрыт ли данный элемент в области проектирования. В каждом случае поле Summary представляет собой XML-комментарий, расположенный напротив соответствующего компонента. События отличаются от других компонентов тем, что в столбце Type должен быт указан делегат. Перемещаться по столбцам можно с помощью клавиш со стрелками, направленными влево и вправо, клавиши <Tab> (следующий столбец) и комбинации клавиш <Shift+Tab> (предыдущий столбец).

Для того чтобы ввести параметры метода, следует с помощью клавиши со стрелкой, направленной вправо, раскрыть узел метода, чтобы появился список параметров. Выбрав узел Add Parameter, пользователь может добавить новый параметр метода. После этого новый параметр можно найти, перемещаясь с помощью клавиш со стрелками.

Окно Properties

Несмотря на полезность окна Class Details, оно не предоставляет полной информации о сущностях, которыми являются компоненты. Например, свойства могут быть предназначены только для чтения, но этот факт никак не отображается в окне Class

Details. Окно Properties, показанное на рис. 10.5, содержит полный список атрибутов свойства `Orders` класса `Customer`.

На рис. 10.5 показано, что свойство `Orders` предназначено только для чтения и не является статическим. Кроме того, в окне показано, что это свойство определено в классе `Customer.cs`. Класс, состоящий из частичных классов, можно разделить между несколькими файлами. Когда пользователь выбирает частичный класс, свойство `File Name` показывает все файлы, определяющие этот класс, в виде списка, в котором элементы разделены запятыми. Несмотря на то что в этом окне некоторые свойства предназначены только для чтения, их можно уточнять в соответствующем исходном файле.

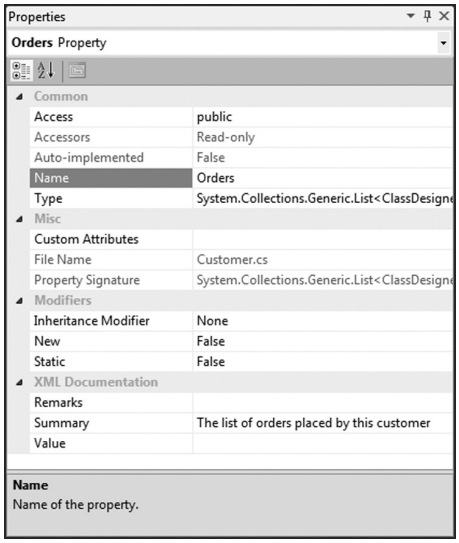


Рис. 10.5

Компоновка



Рис. 10.6

Поскольку диаграмма классов предназначена для визуализации классов и отношений между ними, система Visual Studio предоставляет разработчику несколько инструментальных панелей с элементами управления, позволяющими компоновать сущности в области проектирования. На рис. 10.6 показана инструментальная панель, являющаяся частью области проектирования. Первые три кнопки — Group by Kind, Group by Access и Sort Alphabetically — управляют компоновками сущностей.

Две оставшиеся кнопки используются для автоматического упорядочения сущностей в области проектирования. Левая кнопка — Layout Diagram — автоматически переставляет сущности в области проектирования, а также минимизирует их размеры, скрывая все компоненты. Правая кнопка — Adjust Shapes Width — настраивает размер сущностей так, чтобы все компоненты были полностью видимыми. Если выбран один компонент, то кнопка Adjust Shapes Width настраивает ширину только одного компонента. Если ни один компонент не выбран, то настраивается ширина всех компонентов. Сущности, такие как поля, свойства и методы, можно скрывать с помощью кнопки Hide Member button.

Стиль отображения сущностей можно настроить с помощью трех кнопок. Левая кнопка — Display Name — задает стиль отображения, при котором демонстрируется только имя компонента. Этот стиль можно расширить, используя кнопку Display Name and Type так, чтобы демонстрировались и имя, и тип компонента. Часто это оказывается удобнее, хотя и занимает больше места на экране.

Остальные элементы управления на инструментальной панели позволяют увеличивать или уменьшать область проектирования, а также открывать окно Class Details.

Экспорт диаграмм

Довольно часто процесс выбора классов для архитектуры системы является частью более крупного процесса проектирования или анализа. По этой причине нередко возникает необходимость экспортировать диаграмму класса для ее включения в отчеты.

Экспортировать диаграмму классов можно либо щелчком правой кнопки мыши в любом месте области проектирования, либо с помощью меню Class Diagram. В любом случае, выбрав команду Export Diagram as Image menu, разработчик откроет диалоговое окно, позволяющее выбирать формат изображения и задавать имя файла для сохранения диаграммы.

Изображение можно также непосредственно скопировать и вставить в документ Microsoft Word или графический редактор, например Visio. Для этого необходимо сначала выбрать один или несколько классов на диаграмме. Кроме того, диаграммы классов можно печатать непосредственно из системы Visual Studio с помощью команды File⇨Print.

Генерация и рефакторинг кода

Одна из главных целей системы Visual Studio 2010 и технологии .NET Framework — уменьшить объем кода, который должны писать разработчики. Эта цель достигается двумя путями: либо уменьшается общий объем кода, который должен быть написан, либо уменьшается объем кода, который должен быть написан вручную. Первый подход основан на очень богатом наборе базовых классов, включенных в каркас .NET Framework. Второй подход, нацеленный на уменьшение кода, который должен быть написан вручную, поддерживается средствами генерации кода и рефакторинга, включенными в утилиту Class Designer.

Генерация кода методом “перетащить и отпустить”

Почти каждое действие, выполненное над диаграммой классов, приводит к изменению исходного кода и, по существу, представляет собой определенный уровень генерации кода. Мы уже писали о многих таких изменениях, например, о добавлении свойства или метода в окне Class Details.

Однако некоторые более сложные действия, связанные с генерацией кода, можно выполнить, манипулируя диаграммой классов. Как уже упоминалось ранее, с помощью соединителя Inheritance можно установить отношение наследования между базовым и производным классами. При этом файл кода производного класса обновляется, отражая внесенные изменения. Но если базовый класс является абстрактным, как, например, класс `Product` на рис. 10.7, то надстройка Class Designer может выполнить дополнительный анализ и генерацию кода. Если базовый класс является абстрактным и содержит абстрактные члены, то они автоматически реализуются в производных классах. Это показано на рис. 10.7, *справа*, где абстрактные свойства `Description`, `Price` и `SKU` были добавлены в класс `Book`. Метод `GetInventory()` не был реализован, потому что не был помечен как абстрактный.

Соединитель Inheritance можно использовать еще в одной операции, которая приводит к автоматической генерации кода. На рис. 10.8, *слева*, в диаграмму был добавлен интерфейс ICrudActions. Когда соединитель Inheritance перетаскивается из класса в интерфейс, все члены интерфейса реализуются в классе (см. рис. 10.8, *справа*).

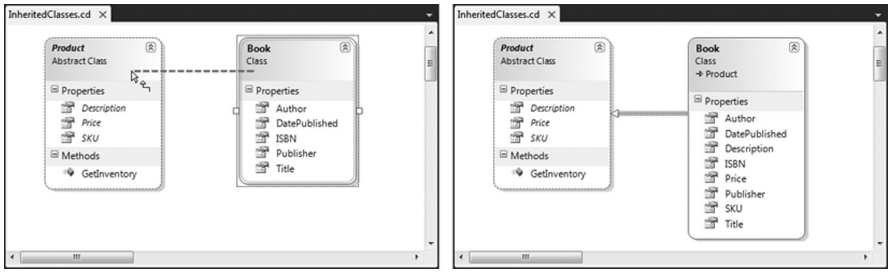


Рис. 10.7

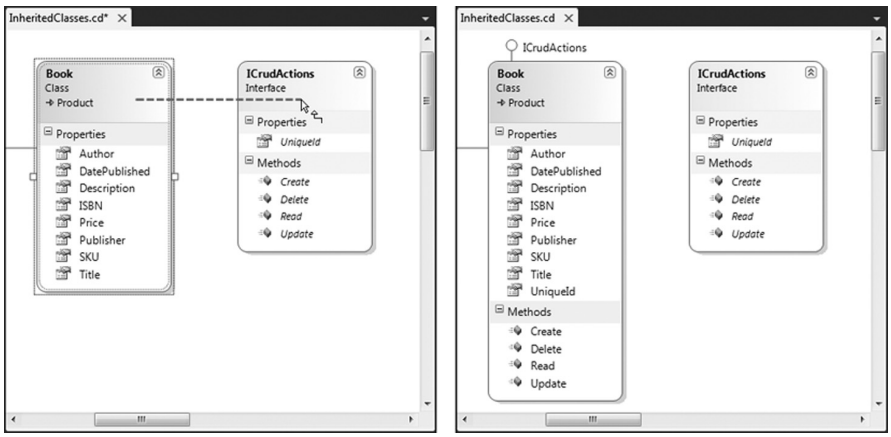


Рис. 10.8

Следующий листинг содержит код, сгенерированный, когда интерфейс ICrudActions был добавлен в класс Book.

C#

```
#region ICrudActions Members
    public Guid UniqueId
    {
        get
        {
            throw new NotImplementedException();
        }
        set
        {
            throw new NotImplementedException();
        }
    }

    public void Create()
    {
        throw new NotImplementedException();
    }

    public void Update()
    {
        throw new NotImplementedException();
    }

    public void Read()
    {
        throw new NotImplementedException();
    }
```

```
public void Delete()
    {
        throw new NotImplementedException();
    }
#endregion
```

Visual Basic

```
#Region ICrudActions Members
    Public Property UniqueId As Guid
        Get
           throw new NotImplementedException()
        End Get
        Set
           throw new NotImplementedException()
        End Set
    End Property

    Public Sub Create()
        throw new NotImplementedException()
    End Sub

    Public Sub Update()
        throw new NotImplementedException()
    End Sub

    Public Sub Read()
        throw new NotImplementedException()
    End Sub

    Public Sub Delete()
        throw new NotImplementedException()
    End Sub
#End Region
```

Генерация кода с помощью технологии IntelliSense

Доступ к остальным функциям, связанным с генерацией кода с помощью надстройки Class Designer, обеспечивает подменю, имеющее довольно странное имя IntelliSense. Поскольку эти функции применяются только к классам, это меню становится видимым только тогда, когда разработчик выбирает какой-нибудь обычный или абстрактный класс на диаграмме.

В этом меню доступны две функции генерации кода: Implement Abstract Class и Override Members. Функция Implement Abstract Class гарантирует, что все абстрактные члены из базового класса будут реализованы в производном классе. Для того чтобы получить доступ к этому классу, необходимо щелкнуть правой кнопкой мыши на производном классе, выбрать команду IntelliSense, а затем команду Implement Abstract Class.

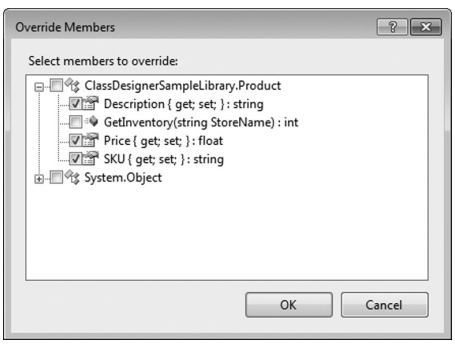


Рис. 10.9

Подобное делает и функция Override Members, которая используется для того, чтобы выбирать подлежащие замещению открытые свойства или методы в базовом классе. Для доступа к этой функции необходимо щелкнуть правой кнопкой мыши на производном классе, выбрать команду IntelliSense, а затем команду Override Members. Диалоговое окно, показанное на рис. 10.9, содержит базовые классы, а также свойства и методы, которые еще не были замещены.

Рефакторинг с помощью надстройки Class Designer

В главе 8 мы видели, как система Visual Studio 2010 поддерживает рефакторинг кода с помощью окна редактирования кода. Надстройка Class Designer тоже имеет много функций, связанных с рефакторингом при работе с сущностями на диаграмме классов.

Для того чтобы получить доступ к функции рефакторинга в надстройке Class Designer, необходимо щелкнуть правой кнопкой мыши на сущности или на любом из ее членов и выбрать команду из меню Refactor. В этом меню доступны следующие функции рефакторинга.

- **Rename Types and Type Members.** Переименовывает тип или его член на диаграмме классов или в окне Properties. Переименование типа или его члена приводит к изменениям всех фрагментов кода, где ранее встречалось его старое имя. Можно даже убедиться, что эти изменения распространяются также на все комментарии и статические строки.
- **Encapsulate Field.** Быстро создает новое свойство на основе существующего поля и аккуратно обновляет код, добавляя в него ссылки на новое свойство.
- **Reorder or Remove Parameters (только для языка C#).** Изменяет порядок следования параметров метода в типе или удаляет параметр из метода.
- **Extract Interface (только в языке C#).** Разработчик может извлекать члены типа в новый интерфейс. Эта функция позволяет выбирать только то подмножество членов, которое необходимо выделить в новый интерфейс.



Для того чтобы копировать и перемещать члены между типами, можно также использовать стандартные операции Cut, Copy и Paste, предусмотренные в системе Windows.

Надстройка Modeling Power Toys для системы Visual Studio

Несмотря на то что надстройка Class Designer является очень полезным инструментом для разработки и визуализации иерархии классов, при работе с крупными диаграммами она иногда оказывается неудобной и громоздкой. Для того чтобы устранить эту проблему, разработчик может либо разделить диаграмму на несколько диаграмм классов, либо инсталлировать надстройку Modeling Power Toys для системы Visual Studio 2010.

Надстройка Modeling Power Toys представляет собой свободную надстройку для системы Visual Studio, расширяющую функциональные возможности утилиты Class Designer в нескольких направлениях. Она включает усовершенствования, позволяющие более эффективно работать с крупными диаграммами, включая *панорамирование* (panning) и изменение *масштаба изображения* (zooming), улучшенную прокрутку и поиск по диаграмме. Она также содержит функции, снимающие некоторые ограничения надстройки Class Designer, например, позволяющие создавать вложенные типы и новые производные классы, а также демонстрировать XML-комментарии.

Надстройка, включая исходный код, доступна по адресу `http://modeling.codeplex.com/`. Ее загрузочный пакет также содержит файл MSI для упрощения инсталляции.

Усовершенствование визуализации

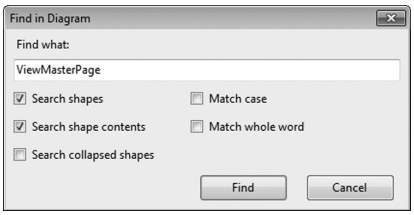


Рис. 10.10

Надстройка Modeling Power Toys для системы Visual Studio 2010 обеспечивает очень полезные усовершенствования для визуализации и работы с крупными диаграммами классов. Одной из наиболее полезных среди них является поиск по диаграмме; это свойство позволяет находить сущности на диаграмме по конкретному запросу. Диалоговое окно поиска, показанное на рис. 10.10, открывается с помощью стандартной команды Find или комбинации клавиш <Ctrl+F>.

Другим полезным инструментом для работы с крупными диаграммами является функция панорамирования, обеспечивающая удобный просмотр всей диаграммы и перемещение по ее разным областям без изменения масштаба изображения. Ее можно вызвать, щелкнув на новой пиктограмме, которая появляется в правом нижнем углу окна. В результате открывается окно панорамирования, показанное на рис. 10.11.

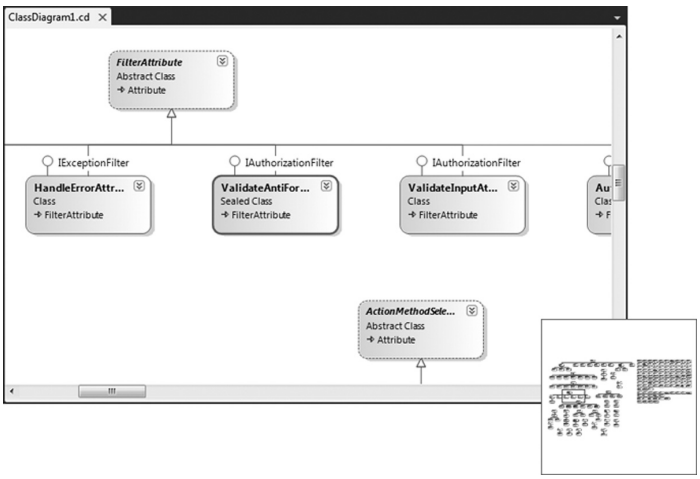


Рис. 10.11

Надстройка Modeling Power Toys позволяет также довольно удобно управлять тем, что должно появиться на диаграмме, с помощью команд фильтрации меню Class Diagram, которые перечислены ниже.

- **Hide Inheritance Lines.** Скрывает все линии наследования на диаграмме.
- **Show All Inheritance Lines.** Отображает все линии наследования на диаграмме.
- **Show All Public Associations.** Демонстрирует все возможные открытые ассоциации на диаграмме.
- **Show All Associations.** Демонстрирует все возможные ассоциации на диаграмме.
- **Show Associations As Members.** Демонстрирует все линии ассоциации в виде членов классов.
- **Hide Private.** Скрывает все закрытые члены класса.

- **Hide Private and Internal.** Скрывает все закрытые и/или внутренние члены классов.
- **Show Only Public.** Скрывает все члены классов, кроме открытых; демонстрирует все открытые члены классов, которые были скрытыми.
- **Show Only Public and Protected.** Скрывает все члены классов, кроме открытых и защищенных; демонстрирует открытые и/или защищенные члены классов, которые были скрыты.
- **Show All Members.** Демонстрирует все скрытые члены классов.

Усовершенствование функциональных свойств

Надстройка Modeling Power Toys содержит большое количество усовершенствований, которые снимают некоторые ограничения, присущие надстройке Class Designer. Несмотря на то что надстройка Class Designer может показывать вложенные типы, она не позволяет разработчику создать их с помощью области проектирования.

В надстройке Modeling Power Toys этого ограничения нет. Разработчик теперь может добавлять любые вложенные типы, включая классы, перечисления, структуры и делегатов. Кроме того, теперь можно легко добавлять несколько новых типов, являющихся членами других типов, например, свойства "только для чтения" и индексаторы.

Работа с интерфейсами также была улучшена. Часто трудно понять, какой член класса был использован для реализации интерфейса. Надстройка Modeling Power Toys упростила эту задачу, добавив команду Select Members в меню, связанное с меткой индикатора типа, напоминающей леденец на палочке. Например, на рис. 10.12 показан вызов команды Select Members для интерфейса IStatus.

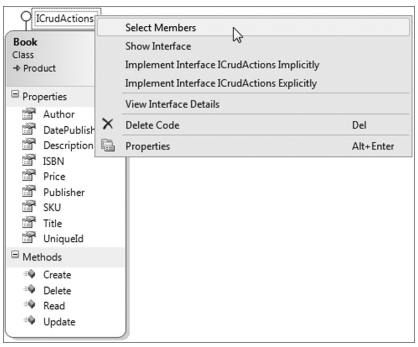


Рис. 10.12

Кроме того, в надстройке Modeling Power Toys реализовано много других небольших улучшений и добавлены разные не столь значительные функциональные возможности, которые делают из нее очень полезный инструмент.

Резюме

Глава посвящена надстройке Class Designer — одному из лучших инструментов генерации и анализа кода, встроенных в систему Visual Studio 2010. Область проектирования, а также инструментальные панели и окна этой надстройки обеспечивают удобный пользовательский интерфейс, позволяющий моделировать и проектировать сложные иерархии классов и ассоциации между классами.

ЧАСТЬ III

Заглянем поглубже

- **ГЛАВА 11.** Модульное тестирование
- **ГЛАВА 12.** Документирование с помощью XML-комментариев
- **ГЛАВА 13.** Инструменты, обеспечивающие согласованность кода
- **ГЛАВА 14.** Генерация кода с помощью генератора T4
- **ГЛАВА 15.** Шаблоны проектов и элементов управления
- **ГЛАВА 16.** Особенности языков программирования

11

Модульное тестирование

В ЭТОЙ ГЛАВЕ...

- Генерация тестовых наборов по существующему коду
- Формальные суждения о функционировании кода
- Выполнение пользовательского кода в ходе тестирования событий жизненного цикла программы
- Создание тестов, ориентированных на данные
- Тестирование закрытых членов и контрактов кода
- Управление списками тестов

Тестирование приложений является одним из наиболее важных компонентов процесса разработки программного обеспечения. Исследование затрат, связанных с сопровождением программного обеспечения, показали, что стоимость исправления ошибки в готовом продукте может в 25 раз превышать стоимость ее исправления в ходе разработки программы. В то же время значительная часть тестирования представляет собой скучную, нудную и подверженную ошибкам работу, которую приходится повторять каждый раз, когда в основной код вносятся изменения. Для того чтобы успешно противостоять этим ошибкам, проще всего разработать автоматизированные тесты, которые можно выполнять на компьютере по требованию разработчика. В настоящей главе рассматриваются специфические тесты автоматизированного тестирования, которые ориентированы на отдельные компоненты или элементы управления системой. Имея набор таких автоматизированных тестов, разработчик может проверить, что все созданные им компоненты системы работают как следует даже после радикальной переработки.

Система Visual Studio 2010 имеет встроенный каркас для авторизации, выполнения и создания отчетов о тестах. Изначально средства тестирования были включены

только в версию Team System Edition системы Visual Studio, но сейчас многие из них доступны и в версии Professional Edition. Это значит, что теперь более широкий круг пользователей может использовать преимущества автоматизированного и надежного тестирования. Данная глава посвящена вопросам создания, конфигурирования и управления набором модульных тестов, а также проблеме создания тестов на основе наборов данных.

Первый тест

Создание тестов нелегко автоматизировать, поскольку они должны отражать функциональные возможности разрабатываемого программного обеспечения. Однако на некоторых этапах этого процесса можно сгенерировать заглушки кода.

Начнем с простого сниппета, который научит читателей писать тесты, полностью выполняющие код программного обеспечения. Рассмотрим класс `Subscription` с открытым свойством `CurrentStatus`, которое возвращает статус текущей подписки в качестве элемента перечисления.



VB

```
Public Class Subscription
    Public Enum Status
        Temporary
        Financial
        Unfinancial
        Suspended
    End Enum

    Public Property PaidUpTo As Nullable(Of Date)

    Public ReadOnly Property CurrentStatus As Status
        Get
            If Not Me.PaidUpTo.HasValue Then Return Status.Temporary
            If Me.PaidUpTo > Now Then
                Return Status.Financial
            Else
                If Me.PaidUpTo > = Now.AddMonths(-3) Then
                    Return Status.Unfinancial
                Else
                    Return Status.Suspended
                End If
            End If
        End Get
    End Property
End Class
```

Сниппет кода Subscriptions\Subscription.vb



C#

```
public class Subscription
{
    public enum Status
    {
        Temporary,
        Financial,
        Unfinancial,
        Suspended
    }
```

```
public DateTime? PaidUpTo { get; set; }

    public Status CurrentStatus
    {
        get
        {
           if (this.PaidUpTo.HasValue == false)
               return Status.Temporary;
           if (this.PaidUpTo > DateTime.Today)
               return Status.Financial;
           else
           {
               if (this.PaidUpTo > = DateTime.Today.AddMonths(-3))
                   return Status.Unfinancial;
               else
                   return Status.Suspended;
           }
       }
    }
}
```

Сниппет кода Subscriptions\Subscription.cs

Как видно из этого сниппета, при тестировании свойства `CurrentStatus` необходимо проверить четыре ветви кода. Для этого мы создали отдельный тестовый класс `SubscriptionTest` в рамках нового тестового проекта и включили в него метод, который должен создавать объект класса `Subscription`, устанавливать свойство `PaidUpTo` и проверять корректность результата, хранящегося в свойства `CurrentStatus`. Затем мы добавляли тестовые методы один за другим, пока не проверили все ветви кода свойства `CurrentStatus`.

К счастью, система Visual Studio автоматизировала процесс создания нового тестового проекта, соответствующего класса `SubscriptionTest` и кода для создания объекта класса `Subscription`. Осталось только выполнить тестовый метод. Кроме того, система Visual Studio содержит механизм, который используется для выполнения тестов, отслеживания процесса их выполнения и создания отчета о результатах тестирования.

Таким образом, мы должны лишь написать код для тестирования нужного свойства. Фактически система Visual Studio генерирует заглушку кода, который исполняет тестируемое свойство. Однако она не генерирует код для проверки того, что объект `Subscription` находится в правильном начальном состоянии; эту часть работы разработчик должен выполнить самостоятельно.

Пользователь может создавать пустые тесты, выбирая команду New Test в меню Test. Эта команда предлагает пользователю выбрать тип создаваемого теста, а затем, после того как будет создан пустой тест, написать соответствующие фрагменты кода. Однако разработчик может также создать новый модельный тест, содержащий бо́льшую часть кода-заглушки, выбрав команду Create Unit Tests в контекстном меню, которое появляется после щелчка правой кнопкой мыши в главном окне редактирования кода. Например, после щелчка правой кнопкой мыши на свойстве `CurrentStatus` и выбора команды Create Unit Tests открывается диалоговое окно Create Unit Tests (рис. 11.1), в котором показаны все члены всех классов в текущем решении. Оно позволяет разработчику выбирать элементы управления, для которых он хочет сгенерировать тестовую заглушку.

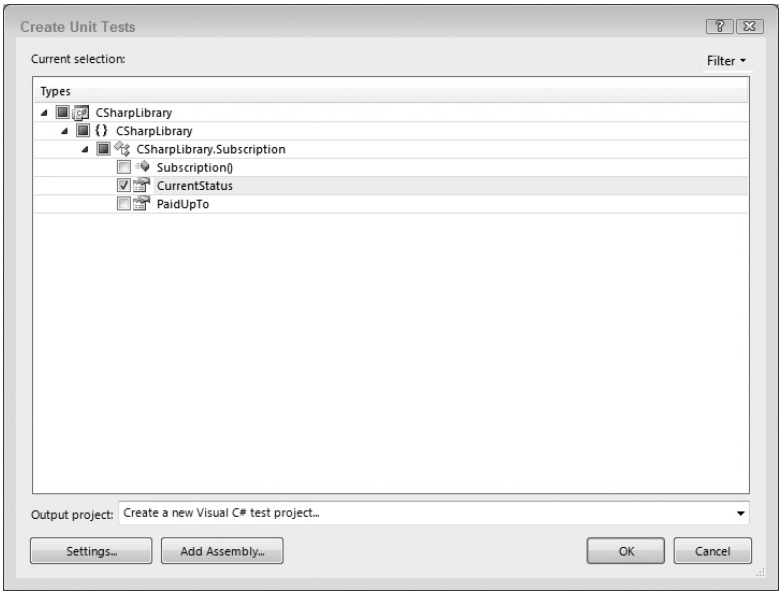


Рис. 11.1

Если проект для модульного тестирования в решении уже существует, то пользователь может сгенерировать свой собственный тестовый класс, выбрав его из раскрывающегося списка Output Project. В противном случае система сделает выбор по умолчанию и сгенерирует новый тестовый проект. В отличие от альтернативных каркасов для модульного тестирования, таких как NUnit, позволяющих размещать тестовые классы в том же проекте, где находится исходный код, каркас для тестирования в системе Visual Studio требует, чтобы все тесты были размещены в отдельном тестовом проекте. После создания тестов с помощью диалогового окна, показанного на рис. 11.1, они именуются с помощью имен членов и классов, которому те принадлежат.



Соглашение о присвоении имен можно изменить с помощью диалогового окна Test Generation Settings, которое открывается после щелчка на кнопке Settings. В нем разработчик найдет много других настроек, позволяющих управлять генерацией тестового кода.

После того как пользователь выберет свойство `CurrentStatus` (см. рис. 11.1) и щелкнет на кнопке OK, система сгенерирует следующий код (некоторые комментарии и закомментированные фрагменты из этого кода были удалены).



VB

```
<TestClass()>
Public Class SubscriptionTest
    Private testContextInstance As TestContext

    Public Property TestContext() As TestContext
        Get
            Return testContextInstance
        End Get
        Set(ByVal value As TestContext)
            testContextInstance = value
```

```
        End Set
    End Property

    <TestMethod()>
    Public Sub CurrentStatusTest()
        Dim target As Subscription = New Subscription()
' TODO: Initialize to an appropriate value
        Dim actual As Subscription.Status
        actual = target.CurrentStatus
        Assert.Inconclusive("Verify the correctness of this test"_
                            "method.")
    End Sub
End Class
```

Сниппет кода SubscriptionTests\SubscriptionTest.vb



C#

```
[TestClass()]
public class SubscriptionTest
{

    private TestContext testContextInstance;

    public TestContext TestContext
    {
        get
        {
           return testContextInstance;
        }
        set
        {
           testContextInstance = value;
        }
    }

    [TestMethod()]
    public void CurrentStatusTest()
    {
        Subscription target = new Subscription();
// TODO: Initialize to an appropriate value
        Subscription.Status actual;
        actual = target.CurrentStatus;
        Assert.Inconclusive("Verify the correctness of this test"_
                            "method.");
    }
}
```

Сниппет кода SubscriptionTests\SubscriptionTest.cs

Тест, сгенерированный для свойства `CurrentStatus`, появляется в последнем методе сниппета. (Первая часть этого класса будет рассмотрена в этой главе позднее.) Как видим, тест был создан с именем, которое соответствует тестируемому свойству (в данном случае `CurrentStatusTest`), и находится в классе, который соответствует классу, содержащему данное свойство (в данном случае `SubscriptionTest`). Одна из трудностей, связанная с тестами, заключается в том, что они быстро выходят из под контроля. Такое простое соглашение об именах гарантирует, что тесты можно будет легко найти и идентифицировать.

Если рассмотреть тесты более внимательно, то можно увидеть, что сгенерированные заглушки содержат код, инициализирующий все объекты, необходимые для тестирования. В частности, создан объект класса `Subscription`, а тестовой переменной `actual` присвоено свойство `CurrentStatus` данного объекта. В этом коде нет лишь реальной проверки правильности данного значения. Прежде чем продвигаться дальше, необходимо выполнить этот тест, чтобы увидеть, что произойдет при открытии окна Test View, показанного на рис. 11.2, из меню Test Windows.



Рис. 11.2

Для того чтобы запустить тест на выполнение, пользователь должен выбрать команду `CurrentStatusTest` и щелкнуть на кнопке Run Selection (первая слева). В результате откроется окно Test Results, которое сначала показывает тест либо в состоянии Pending, либо в состоянии In Progress.



Окно Test View представляет собой лишь один из нескольких способов выбрать и выполнить тест. Если щелкнуть правой кнопкой мыши на самом тесте в окне редактирования кода, то в контекстном меню можно выбрать команду Run Tests. Кроме того, существует инструментальная панель Tests с командами, позволяющими запускать все тесты, доступные в текущем контексте. Она позволяет запускать отдельный тест, целый класс тестов, сборку тестов или все тесты, существующие в решении, в зависимости от текущего выбора пользователя. В заключение пользователь может создать списки тестов, используя надстройку Test List Editor, которая будет рассмотрена в этой главе позднее.

Помимо этого, в каждом из методов можно задать контрольные точки и запустить тесты в режиме отладки, выбрав одну из команд Debug Tests на главной инструментальной панели или в окне Test View.

Как только тест будет выполнен, окно Test Results примет такой вид, как показано на рис. 11.3.

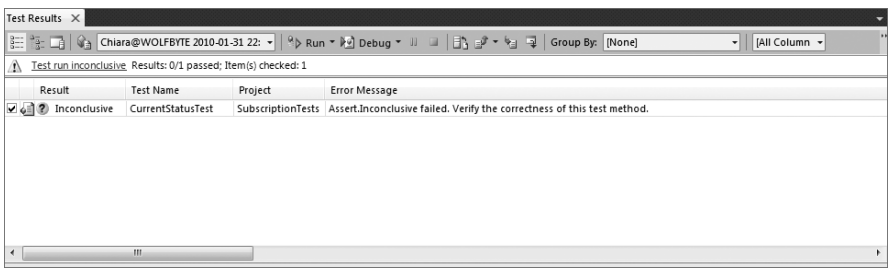


Рис. 11.3

На рис. 11.3 видно, что тест привел к неубедительным результатам. По существу, это значит, что либо тест был неполным, либо результаты не заслуживают доверия из-за того, что внесенные изменения могли нарушить корректность теста.

Дважды щелкнув на результате любого теста, можно получить более подробную информацию о нем. На рис. 11.4 показана информация о незавершенном тесте, которая была получена описанным выше способом. В ней содержатся сведения о самом тесте, его результате, а также другая информация об окружении, например, имя компьютера, продолжительность выполнения теста, время его начала и окончания.



Рис. 11.4



Наряду с информацией о конкретном результате пользователь может получить сведения о завершенном тесте, щелкнув на кнопке Run Details в окне Test Results. По умолчанию в окне Test Results выводятся результаты самого последнего выполнения теста, но система Visual Studio хранит все результаты недавних запусков, а их результаты можно просмотреть с помощью раскрывающегося списка Test Run.

Все тесты, сгенерированные системой Visual Studio, сначала помечаются как незавершенные с помощью инструкции `Assert.Inconclusive`. Кроме того, в зависимости от сгенерированной тестовой заглушки возможны дополнительные инструкции `TODO`, предлагающие пользователю завершить тест.

Вернувшись к сниппету, сгенерированному для метода `CurrentStatusTest`, можно увидеть обе инструкции — `Assert.Inconclusive` и `TODO`. Для того чтобы завершить этот тест, необходимо удалить комментарий из инструкции `TODO` и заменить инструкцию `Assert.Inconclusive` инструкцией `Assert.AreEqual`, как показано в следующем фрагменте кода.



VB

```
<TestMethod()>
    Public Sub CurrentStatusTest()
        Dim target As Subscription = New Subscription
        Dim actual As Subscription.Status
        actual = target.CurrentStatus
        Assert.AreEqual(Subscription.Status.Temporary, actual, _
                        "Subscription.CurrentStatus was not set "_
                        "correctly." )
   End Sub
```

Сниппет кода SubscriptionTests\SubscriptionTest.vb



C#

```
[TestMethod()]
public void CurrentStatusTest()
{
    Subscription target = new Subscription();
```

```
        Subscription.Status actual;
        actual = target.CurrentStatus;
        Assert.AreEqual(Subscription.Status.Temporary, actual,
                        "Subscription.CurrentStatus was not set correctly." );
    }
```

Сниппет кода SubscriptionTests\SubscriptionTest.cs

Каждый из тестов, показанных в окне Test Results, имеет флажок. Выбрав результаты теста и щелкнув на кнопке Run Tests в окне Test Results, пользователь может запустить только выбранные тесты. По умолчанию после выполнения теста все не пройденные тесты помечаются как выбранные. После того как разработчик исправит код, вызвавший сбой при выполнении теста, он должен щелкнуть на кнопке Run Tests, чтобы повторно выполнить тестовые варианты и добиться успешного результата (рис. 11.5).

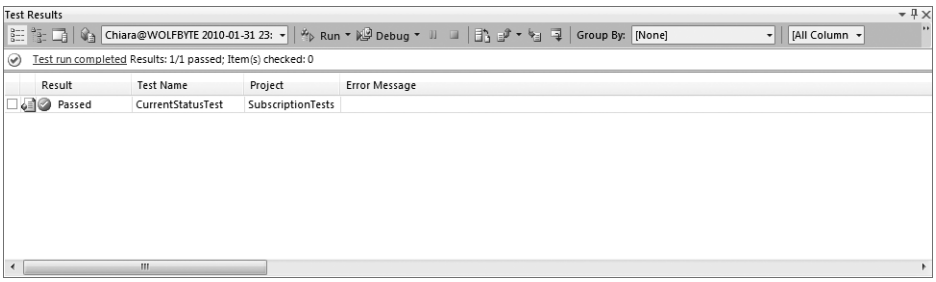


Рис. 11.5



Любой тестовый вариант, не имеющий проверки *значений* (assertions), по умолчанию считается пройденным, поэтому система Visual Studio автоматически добавляет предупреждение `Assert.Inconclusive` в сгенерированные тестовые варианты. Удаляя эту проверку, пользователь указывает, что данный тестовый вариант считается завершенным. В данном примере мы прошли только по одной ветви кода и должны добавить дополнительные тестовые варианты, чтобы проверить остальные три ветви.

Когда мы создавали наш первый модульный тест в начале главы, читатели, возможно, заметили, что кроме нового тестового проекта под папкой нового решения Solutions Item появились два элемента: Chapter11.vsmdi и Local.testsettings.

Файлы с расширением `.vsmdi` представляют собой файлы метаданных, содержащие информацию о тестах в решении. Дважды щелкнув на таком файле в системе Visual Studio, пользователь открывает окно Test List Editor, которое будет рассмотрено в конце главы.

Файлы с расширением `.testsettings` – это файлы конфигурации тестового прогона. Это XML-файлы, в которых хранятся настройки, управляющие наборами тестов, именуемых *тестовыми прогонами* (test runs). Пользователь может создать и сохранить несколько конфигураций, описывающих разные сценарии, а затем создать конкретную активную конфигурацию, выбрав команду Test⇨Select Active Test Run Configuration. Она определяет, какую из конфигураций тестового прогона следует применить при запуске теста.



В версии Ultimate edition системы Visual Studio в момент создания нового тестового проекта создается файл `TraceAndTestImpact.testsettings`. Эта конфигурация тестового прогона используется системой Visual Studio для реализации свойства Test Impact Analysis, описываемого в главе 56.

Когда пользователь дважды щелкает на файле `Local.testsettings`, запускается специальный редактор. В этом редакторе можно задавать конфигурацию тестового прогона, чтобы скопировать требуемые *файлы поддержки* (support files) в каталог развертывания или связаться со сценариями запуска и доводки. Этот редактор также содержит раздел Test Timeouts (рис. 11.6), позволяющий определять лимит времени, по истечении которого тест либо аннулируется, либо помечается как проваленный. Это полезно, когда для приложения установлен глобальный предел производительности (например, если все экраны обновляются в течение пяти секунд).

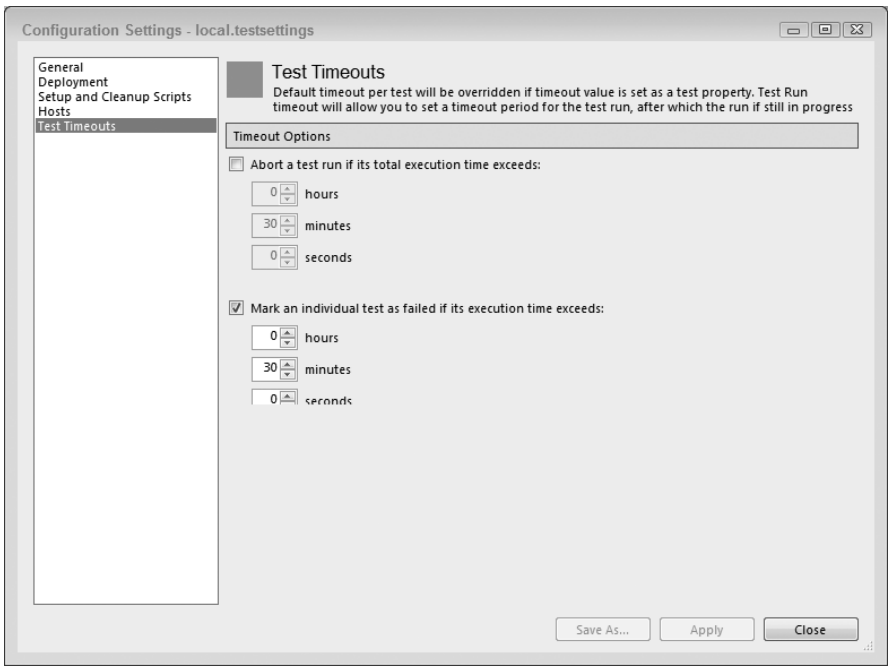


Рис. 11.6

Большинство этих установок могут быть замещены с помощью методов и тестовых атрибутов, которые обсуждаются в следующем разделе.

Идентификация тестов с помощью атрибутов

Прежде чем продолжить наш сценарий, сделаем шаг назад и рассмотрим процесс тестирования в системе Visual Studio 2010. Как указывалось ранее, все тестовые варианты должны находиться в тестовых классах, которые в свою очередь включаются в состав тестового проекта. Но чем же отличаются методы, классы и проекты, содержащие тестовые варианты? Если заглянуть в XML-файл тестового проекта, можно увидеть, что он практически ничем не отличается от обычного файла проекта биб-

лиотеки классов. По существу, этот тип проектов имеет только одно отличие: после компиляции проекта создается стандартная сборка библиотеки классов .NET. Основное отличие заключается в том, что система Visual Studio распознает этот проект как тестовый и автоматически анализирует тестовые варианты по порядку, заполняя разные тестовые окна.

Классы и методы, используемые в процессе тестирования, помечаются соответствующим атрибутом. Эти атрибуты используются механизмом тестирования для нумерации всех тестовых вариантов в конкретной сборке.

Атрибут `TestClass`

Все тестовые варианты должны находиться в тестовых классах, помеченных атрибутом `TestClass`. Несмотря на то что иногда для использования этого атрибута нет почти никаких причин, кроме присоединения тестовых вариантов к тестовому классу и методу, для проверки которого они предназначены, позднее мы увидим некоторые преимущества группирования тестовых вариантов с помощью тестового класса. При тестировании класса `Subscription` создается тестовый класс `SubscriptionTest`, помеченный атрибутом `TestClass`. Поскольку для поиска класса, содержащего тестовые варианты, система Visual Studio использует атрибуты, имя этого класса не имеет значения. Однако определенное соглашение об именах, например добавление суффикса `Test`, облегчает управление многочисленными тестовыми вариантами.

Атрибут `TestMethod`

Некоторые тестовые варианты помечаются атрибутом `TestMethod`, позволяющим системе Visual Studio перечислять тесты, которые могут быть выполнены. Метод `CurrentStatusTest` в классе `SubscriptionTest` помечается атрибутом `TestMethod`. И опять же фактическое имя этого метода значения не имеет, поскольку система Visual Studio использует только атрибуты. Однако имя метода используется в разных тестовых окнах, в которых перечисляются тестовые варианты, поэтому целесообразно давать методам осмысленные имена. Это особенно важно для анализа результатов тестов.

Дополнительные тестовые атрибуты

Как мы видели, для идентификации тестовых вариантов подсистема модульного тестирования в системе Visual Studio использует атрибуты. Для обеспечения более полной информации о тестовом варианте можно задать дополнительные свойства. Затем эта информация станет доступной либо с помощью окна Properties, связанного с тестовым вариантом, или с помощью других тестовых окон. В этом разделе описываются описательные атрибуты, которые можно применять к тестируемому методу.

Атрибут `Description`

Поскольку тестовые варианты перечисляются по именам тестируемых методов, многие тесты могут иметь одинаковые или не вполне информативные имена. Это не позволяет идентифицировать функциональные свойства, которые они тестируют. Для того чтобы обеспечить дополнительную информацию о тестовом варианте, к тестируемому методу можно применить атрибут `Description`, имеющий единственный аргумент типа `String`.

Атрибут Owner

Атрибут Owner, принимающий единственный аргумент типа String, позволяет идентифицировать того, кто владеет тестовым вариантом, кто его написал и кто с ним в данный момент работает.

Атрибут Priority

Для того чтобы идентифицировать относительную важность тестового варианта, к нему можно применить атрибут Priority, принимающий аргумент типа Integer. Несмотря на то что тестовая система не использует этот атрибут для поиска, он полезен для расстановки приоритетов тестовых вариантов при определении порядка, в котором решаются проваленные или незавершенные тестовые варианты.

Атрибут TestCategories

Атрибут TestCategory принимает единственный аргумент типа String, идентифицирующий категорию теста, определенную пользователем. Как и атрибут Priority, атрибут TestCategory, по существу, игнорируется системой Visual Studio, но может быть полезен для сортировки и группировки связанных тестов. Тестовый вариант может принадлежать к разным категориям, но каждый из них должен иметь отдельный атрибут.

Атрибут WorkItem

Атрибут WorkItem можно использовать для связывания тестового варианта с одним или несколькими рабочими элементами в системе отслеживания, например Team Foundation Server. Если применить к тестовому варианту один или несколько атрибутов WorkItem, то пользователь может пересматривать его повторно после внесения изменений в набор существующих функциональных свойств. Система Team Foundation Server описывается в главе 57.

Атрибут Ignore

Выполнение тестового метода можно временно отложить, применив к нему атрибут Ignore. Тестовые методы с атрибутом Ignore не запускаются, а их результаты не отражаются в списке выполненных тестов.



Атрибут Ignore можно применить как к отдельному тестовому классу, так и ко всем тестовым методам одновременно.

Атрибут Timeout

Тестовый вариант может провалиться по разным причинам. Например, тест производительности может потребовать определенную функциональную возможность для его завершения в течение определенного интервала времени. Для того чтобы освободить тестировщика от обязанности писать сложные многопоточные тесты, останавливающие выполнение тестового варианта по истечении определенного времени, разработчик может применить к нему атрибут Timeout, указав продолжительность интервала времени в миллисекундах, как показано в следующем фрагменте кода. Это гарантирует, что тестовый вариант будет провален, если не будет выполнен в течение заданного интервала времени.



VB

```
<TestMethod()>
<Owner("Mike Minutillo")>
<Description("Tests the functionality of the Current Status Property")>
<Priority(3)>
<Timeout(10000)>
<TestCategory("Financial")>
Public Sub CurrentStatusTest()
    Dim target As Subscription = New Subscription
    Dim actual As Subscription.Status
    actual = target.CurrentStatus
    Assert.AreEqual(Subscription.Status.Temporary, actual, _
                    "Subscription.CurrentStatus was
                     not set correctly." )
End Sub
```

Сниппет кода SubscriptionTests\SubscriptionTest.vb



C#

```
[TestMethod()]
[Owner("Mike Minutillo")]
[Description("Tests the functionality of the Current Status Method")]
[Priority(3)]
[Timeout(10000)]
[TestCategory("Financial")]
public void CurrentStatusTest()
{
    Subscription target = new Subscription();
    Subscription.Status actual;
    actual = target.CurrentStatus;
    Assert.AreEqual(Subscription.Status.Temporary, actual,
                    "Subscription.CurrentStatus was
                    not set correctly.");
}
```

Сниппет кода SubscriptionTests\SubscriptionTest.cs

В этом сниппете некоторые атрибуты оригинального метода `CurrentStatusTest` иллюстрируют его использование. Помимо обеспечения дополнительной информации о том, что делает тестовый вариант и кто его написал, этот код присваивает тестовому варианту приоритет `3` и категорию `"Financial"` . Кроме того, этот код означает, что данный тестовый вариант должен считаться проваленным, если не будет выполнен за 10 секунд (10 000 миллисекунд).

Проверка фактов

До сих пор в этой главе мы проверяли, как устроены тесты и как тестовые варианты вкладываются в текстовые классы в рамках тестового проекта. Осталось изучить тело тестового варианта и посмотреть, как тесты выполняются успешно или проваливаются. (После того как тестовый вариант сгенерирован, в конец теста добавляется инструкция `Assert.Inconclusive`, означающая, что тест не завершен.)

Идея, лежащая в основе модульного тестирования, заключается в том, что проверка начинается с системы, компонента или объекта, находящегося в известном состоянии, а затем выполняется метод, модифицируется свойство или провоцируется событие. Этап тестирования начинается в конце разработки, когда необходимо убедиться, что система, компонент или объект пребывает в правильном состоянии.

Кроме того, иногда необходимо проверить, что метод или свойство возвращает правильный результат. В этом случае разработчик должен попытаться проверить конкретное условие. Если это условие не выполняется, то система тестирования сообщает об этом и заканчивает проверку. Естественно, проверяемое условие формулируется с помощью класса `Assert`. Существуют также классы `StringAssert` и `CollectionAssert`, предназначенные для проведения дополнительных проверок, связанных с объектами класса `String` и коллекциями объектов соответственно.

Класс `Assert`

Класс `Assert` принадлежит пространству имен `UnitTesting`, чтобы не путать его с методами `Debug.Assert` и `Trace.Assert` в пространстве имен `System.Diagnostics`. Это основной класс, используемый для формулировки утверждений о тестовом варианте. Основное утверждение имеет следующий формат.

VB

```
Assert.IsTrue(variableToTest, "Output message if this fails")
```

C#

```
Assert.IsTrue(variableToTest, "Output message if this fails");
```

Легко убедиться, что первый аргумент представляет собой проверяемое условие. Если оно выполняется, то тестовый вариант продолжается. Но если условие не выполняется, то система выдает сообщение и тестовый вариант заканчивается провалом.

Эта инструкция имеет много перегруженных вариантов, в которых можно пропускать сообщение об ошибке или задавать параметры форматирования строк. Поскольку проверка какого-то одного позитивного условия производится довольно редко, для упрощения проверки тестовых вариантов предусмотрены дополнительные методы.

- `IsFalse`. Проверяет негативность или ложность условия.
- `AreEqual`. Проверяет, имеют ли два аргумента одинаковые значения.
- `AreSame`. Проверяет, ссылаются ли два аргумента на один и тот же объект.
- `IsInstanceOfType`. Проверяет, является ли аргумент экземпляром конкретного типа.
- `IsNull`. Проверяет, не пуст ли аргумент.

Этот список не полон — существует еще несколько методов, включая негативные эквиваленты перечисленных выше. Кроме того, многие из этих методов перегружены, что позволяет вызывать их разными способами.

Класс `StringAssert`

Класс `StringAssert` не добавляет никакого дополнительного функционального свойства, которое нельзя было бы обеспечить с помощью одного или нескольких утверждений, сделанных с помощью класса `Assert`. Однако он не только упрощает кодирование тестовых вариантов, делая яснее утверждения об объектах класса `String`, но и уменьшает объем работы, связанной с тестированием конкретных условий. Перечислим дополнительные утверждения.

- `Contains`. Проверяет, содержит ли объект класса `String` другой объект класса `String`.

- DoesNotMatch. Проверяет, не совпадает ли объект класса String с регулярным выражением.
- EndsWith. Проверяет, заканчивается ли объект класса String конкретным объектом класса String.
- Matches. Проверяет, совпадает ли объект класса String с регулярным выражением.
- StartsWith. Проверяет, начинается ли объект класса String конкретным объектом класса String.

Класс CollectionAssert

Как и класс StringAssert, класс CollectionAssert является вспомогательным классом, используемым для проверки утверждения о коллекциях элементов. Некоторые из этих утверждений приведены ниже.

- AllItemsAreNotNull. Проверяет, что ни один из элементов в коллекции не является нулевой ссылкой.
- AllItemsAreUnique. Проверяет, что в коллекции элементов нет дубликатов.
- Contains. Проверяет, содержит ли коллекция конкретный объект.
- IsSubsetOf. Проверяет, является ли коллекция подмножеством другой коллекции.

Атрибут ExpectedException

Иногда тестовые варианты должны предусматривать проверку ветвей кода, которые могут генерировать исключения. Несмотря на то что кодирования исключений следует избегать, существуют ситуации, в которых это может быть приемлемым. Вместо того чтобы писать тестовый вариант, содержащий блок Try-Catch с соответствующим проверяемым условием, которое генерирует исключительную ситуацию, можно просто пометить тестовый вариант атрибутом ExpectedException. Например, изменим свойство CurrentStatus так, чтобы оно генерировало исключение, если дата PaidUp предшествует дате открытия подписки, которая в данном тестовом варианте является постоянной.



VB

```
Public Const SubscriptionOpenedOn As Date = #1/1/2000#
Public ReadOnly Property CurrentStatus As Status
    Get
        If Not Me.PaidUpTo.HasValue Then Return Status.Temporary
        If Me.PaidUpTo > Now Then
            Return Status.Financial
        Else
            If Me.PaidUpTo > = Now.AddMonths(-3) Then
                Return Status.Unfinancial
            ElseIf Me.PaidUpTo > SubscriptionOpenedOn Then
                Return Status.Suspended
            Else
                Throw New ArgumentOutOfRangeException( _
            "Paid up date is not valid as it is before the
             subscription opened.")
```

```
            End If
        End If
    End Get
End Property
```

Сниппет кода Subscriptions\Subscription.cs



C#

```
public static readonly DateTime SubscriptionOpenedOn =
                                    new DateTime(2000, 1, 1);
public Status CurrentStatus
{
    get
    {
        if (this.PaidUpTo.HasValue == false)
            return Status.Temporary;
        if (this.PaidUpTo > DateTime.Today)
            return Status.Financial;
        else
        {
            if (this.PaidUpTo > = DateTime.Today.AddMonths(-3))
                return Status.Unfinancial;
            else if (this.PaidUpTo >= SubscriptionOpenedOn)
              return Status.Suspended;
            else
                throw new ArgumentOutOfRangeException(
    "Paid up date is not valid as it is before the subscription opened");
        }
    }
}
```

Сниппет кода Subscriptions\Subscription.vb

Используя аналогичную процедуру, можно создать отдельный тестовый вариант для тестирования ветви кода, как показано в следующем примере.



VB

```
<TestMethod()>
<ExpectedException(GetType(ArgumentOutOfRangeException),
    "Argument exception not raised for invalid PaidUp date.")>
Public Sub CurrentStatusExceptionTest()
    Dim target As Subscription = New Subscription

    target.PaidUpTo = Subscription.SubscriptionOpenedOn.AddMonths(-1)

    Dim expected = Subscription.Status.Temporary

    Assert.AreEqual(expected, target.CurrentStatus, _
                "This assertion should never actually be evaluated")
End Sub
```

Сниппет кода SubscriptionTests\SubscriptionTest.vb



C#

```
[TestMethod()]
[ExpectedException(typeof(ArgumentOutOfRangeException),
    "Argument Exception not raised for invalid PaidUp date.")]
public void CurrentStatusExceptionTest()
{
    Subscription target = new Subscription();
    target.PaidUpTo = Subscription.SubscriptionOpenedOn.AddMonths(-1);

    var expected = Subscription.Status.Temporary;
```

```
        Assert.AreEqual(expected, target.CurrentStatus,
                        "This assertion should never actually be evaluated");
    }
```

Сниппет кода SubscriptionTests\SubscriptionTest.cs

Атрибут `ExpectedException` не только перехватывает любое исключение, генерируемое тестовым вариантом, но и гарантирует, что тип этого исключения совпадает с ожидаемым. Если в тестовом варианте не генерируется ни одно исключение, этот атрибут становится причиной провала теста.

Инициализация и возврат ресурсов

Несмотря на то что система Visual Studio автоматически генерирует заглушки кода для тестовых вариантов вместо разработчика, все же для настройки теста он должен написать довольно много инструкций. Если приложение использует базу данных, то после выполнения каждого теста ее необходимо вернуть в исходное состояние, чтобы тестовые варианты можно было полностью повторить. Это же относится и к приложениям, модифицирующим другие ресурсы, например, файловую систему. Система Visual Studio обеспечивает поддержку для написания методов, которые можно использовать для инициализации и возврата ресурсов после выполнения тестовых вариантов. (Для идентификации соответствующих методов, которые должны инициализировать и возвращать ресурсы после выполнения тестовых вариантов, здесь снова используются атрибуты.)

Атрибуты для инициализации и возврата ресурсов после выполнения тестовых вариантов разделяются на три категории: для индивидуальных тестов, класса тестов и всего тестового проекта.

Атрибуты `TestInitialize` и `TestCleanup`

Как следует из их имен, атрибуты `TestInitialize` и `TestCleanup` идентифицируют методы, которые должны выполняться до и после каждого тестового варианта в конкретном тестовом классе. Эти методы полезны для выделения и дальнейшего освобождения ресурсов, необходимых для всех тестовых вариантов в тестовом классе.

Атрибуты `ClassInitialize` и `ClassCleanup`

Иногда вместо инициализации и освобождения ресурсов после выполнения каждого теста бывает легче проверить, что окружение находится в правильном состоянии перед и после выполнения всех тестовых вариантов из тестового класса. Ранее мы уже объясняли, что тестовые классы представляют собой полезный механизм группировки тестовых вариантов; здесь мы покажем, как можно использовать эти знания. Тестовые варианты можно сгруппировать в тестовые классы, содержащие методы, помеченные атрибутами `ClassInitialize` и `ClassCleanup` соответственно. Эти методы должны быть статическими, причем метод, помеченный атрибутом `ClassInitialize`, должен иметь только один параметр типа `UnitTesting.TestContext`, который будет рассмотрен немного позже.



Если для генерации модульного теста используется меню Create Unit Test, то оно генерирует заглушки для методов TestInitialize, TestCleanup, ClassInitialize и ClassCleanup в закомментированной части кода.

Атрибуты `AssemblyInitialize` и `AssemblyCleanup`

Последняя категория атрибутов инициализации и возврата ресурсов относится к уровню сборок или проекта. Методы, предназначенные для инициализации окружения перед запуском всего тестового проекта и освобождения ресурсов после его завершения, можно пометить атрибутами `AssemblyInitialize` и `AssemblyCleanup` соответственно.

Поскольку эти методы применяются к любому тестовому варианту в тестовом проекте, каждым из этих атрибутов можно пометить только один метод. Как и их эквиваленты уровня класса, оба эти метода должны быть статическими, причем метод, помеченный атрибутом `AssemblyInitialize`, должен принимать параметр типа `UnitTesting.TestContext`.

Для атрибутов уровня сборки и класса важно помнить, что методы, помеченные этими атрибутами, выполняются, даже если запущен только один тестовый вариант.



Рекомендуется поместить методы с атрибутами `AssemblyInitialize` и `AssemblyCleanup` в их собственный тестовый класс, чтобы их можно было легко найти. Если этими атрибутами помечено несколько методов, то запуск любого из тестов в проекте приведет к ошибке во время выполнения программы.

Контекст тестирования

Механизм тестирования имеет множество возможностей, упрощающих разработку тестовых вариантов. К ним относятся управление наборами данных, чтобы применять тестовый вариант только к определенному диапазону значений, а также вывод дополнительной информации о тестовом варианте для облегчения отладки. Эти функциональные возможности обеспечиваются объектом типа `TestContext`, который генерируется в тестовом классе и передается методам `AssemblyInitialize` и `ClassInitialize`.

Данные

Мы проверили только одну ветвь метода `CurrentStatusTest`, сгенерированного в первом разделе главы, — свойство `CurrentStatus`. Для полной поверки метода мы должны написать дополнительные инструкции и *утверждения* (assertions), позволяющие настроить и проверить объект класса `Subscription`. Однако очевидно, что этот процесс носит повторяющийся характер и должен заново выполняться каждый раз, когда изменяется структура свойства `CurrentStatus`. В качестве альтернативы можно предусмотреть объект класса `DataSource` для метода `CurrentStatusTest`, в котором каждая строка данных предназначена для проверки отдельной ветви свойства. Для того чтобы добавить соответствующие данные в метод, следует выполнить следующее.

1. Создать локальную базу данных SQL CE и таблицу базы данных для хранения тестовых данных. В данном случае следует создать базу данных `LoadTest` и таблицу `Subscription_CurrentStatus`. Эта таблица должна содержать столбец со свойством `Identity bigint` и именем `Id`, обнуленный столбец типа `datetime` с именем `PaidUp` и столбец типа `nvarchar(20)` с именем `Status`.
2. Добавить в таблицу значения данных, соответствующие всем ветвям кода. Тестовые значения для свойства `CurrentStatus` показаны на рис. 11.7.

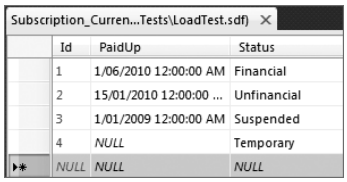


Рис. 11.7

3. Выбрать соответствующий тестовый вариант в окне Test View и открыть окно Properties. Выбрать свойство `Data Connection String` и щелкнуть на эллипсисе, чтобы открыть диалоговое окно Connection Properties.
4. Используя диалоговое окно Connection Properties, установить связь с таблицей данных, созданной на этапе 1. На экране должна появиться строка связи, напоминающая `source=|DataDirectory|\LoadTest.sdf`.
5. Если строка связи корректна, то отобразится раскрывающийся список, в котором следует выбрать свойство `DataTable`. Это позволит выбрать таблицу базы данных, созданной на этапе 1.
6. Для того чтобы открыть тестовый вариант в главном окне, следует вернуться к окну Test View и выполнить команду Open Test из контекстного меню для конкретного тестового варианта. Отметим, что к тестовому варианту при этом добавляется атрибут `DataSource`, который используется тестовым движком для загрузки соответствующих данных из заданной таблицы. Затем эти данные открываются для тестового варианта через объект класса `TestContext`.



Если используется база данных SQL Server CE, то по умолчанию будет добавлен атрибут `DeploymentItem`. Это гарантирует, что данная база данных будет скопирована, если тестовая сборка будет развернута в другом месте.

7. Модифицировать тестовый вариант, чтобы получить доступ к объекту класса `TestContext` и использовать эти данные для управления тестовым вариантом. Это приведет к следующему методу `CurrentStatusTest`.



VB

```
<DataSource("System.Data.SqlServerCe.3.5", _
    "data source=|DataDirectory|\LoadTest.sdf", _
    "Subscription_CurrentStatus", DataAccessMethod.Sequential)> _
<DeploymentItem("SubscriptionTests\LoadTest.sdf")> _
<TestMethod()>_
Public Sub CurrentStatusDataTest()
    Dim target As Subscription = New Subscription
    If Not IsDBNull(Me.TestContext.DataRow.Item("PaidUp")) Then
        target.PaidUpTo = CType(Me.TestContext.DataRow.Item("PaidUp"),
                                Date)
    End If
    Dim val As Subscription.Status = _
        CType([Enum].Parse(GetType(Subscription.Status), _
        CStr(Me.TestContext.DataRow.Item("Status"))),
             Subscription.Status)
```

```
    Assert.AreEqual(val, target.CurrentStatus, _
        "Subscripiton.CurrentStatus was not set correctly.")
End Sub
```

Сниппет кода SubscriptionTests\SubscriptionTest.vb



C#

```
[DataSource("System.Data.SqlServerCe.3.5",
     "data source=|DataDirectory|\\LoadTests.sdf",
     "Subscription_CurrentStatus",
     ataAccessMethod.Sequential)]
[DeploymentItem("SubscriptionTests\\LoadTests.sdf")]
[TestMethod()]
public void CurrentStatusDataTest()
{
    var target = new Subscription();
    var date = this.TestContext.DataRow["PaidUp"] as DateTime?;
    if (date != null)
    {
        target.PaidUpTo = date;
    }

    var val = Enum.Parse(typeof(Subscription.Status),
    this.TestContext.DataRow["Status"] as string);

    Assert.AreEqual(val, target.CurrentStatus,
        "Subscription.CurrentStatus was not set correctly.");
   }
```

Сниппет кода SubscriptionTests\SubscriptionTest.cs

В ходе выполнения тестового варианта метод `CurrentStatusTest` выполняется четыре раза (по одному разу для каждой строки из таблицы базы данных). При этом каждый раз ему передается объект класса `DataRow` с помощью свойства `TestContext.DataRow`. Если логика свойства `CurrentStatus` изменяется, то в таблицу `Subscription_CurrentStatus` можно добавить новую строку, чтобы проверить все вновь возникающие ветви кода.

Прежде чем продолжить изложение, бросим прощальный взгляд на атрибут `DataSource`, который применялся к методу `CurrentStatusTest`. Этот атрибут принимает четыре аргумента, первые три из которых предназначены для идентификации объекта класса `DataTable`, который должен быть извлечен. Последний аргумент— это перечисление `DataAccessMethod`, определяющее порядок, в котором строки возвращаются из объекта класса `DataTable`. По умолчанию оно имеет значение `Sequential`, которое можно заменить значением `Random`, чтобы при каждом новом запуске теста порядок менялся. Это особенно важно, если данные являются репрезентативными, но не предназначены для обработки в каком-то определенном порядке.



Тесты, ориентированные на данные, не ограничиваются только таблицами баз данных, они могут использовать электронные таблицы Excel и даже файлы типа Comma-Separated Values (CSV).

Запись результатов теста

Разработка модульных тестов направлена на автоматизацию процесса тестирования приложения. Следовательно, эти тестовые варианты должны выполняться как часть процесса сборки, возможно, даже на удаленном компьютере. Это значит, что обычные окна для вывода результатов, например консоль, не подходят для демонстрации информации, связанной с тестированием. Очевидно также, что эта информация не долж-

на быть разбросана среди данных об отладке или трассировке, поэтому в системе Visual Studio предусмотрен отдельный канал для записи информации, связанной с тестированием, чтобы ее можно было просматривать вместе с результатами тестирования.

Объект класса `TestContext` содержит метод `WriteLine`, принимающий параметр типа `String` и ряд объектов типа `String.Format`, которые можно использовать для вывода информации о результатах конкретного теста. Например, добавив следующую строку в метод `CurrentStatusDataTest`, можно сгенерировать следующую дополнительную информацию о результатах тестирования.



VB

```
TestContext.WriteLine("No exceptions thrown for test id {0}" , _
    CInt(Me.TestContext.DataRow.Item(0)))
```

Сниппет кода SubscriptionTests\SubscriptionTest.vb



C#

```
TestContext.WriteLine("No exceptions thrown for test id {0}" ,
    this.TestContext.DataRow[0]);
```

Сниппет кода SubscriptionTests\SubscriptionTest.cs



Несмотря на то что разработчик должен использовать метод `TestContext.WriteLine`, чтобы получить подробную информацию о выполнении теста, система тестирования Visual Studio собирает все сообщения о стандартных ошибках и стандартных потоках вывода и выводит эти данные в окне Test Results.

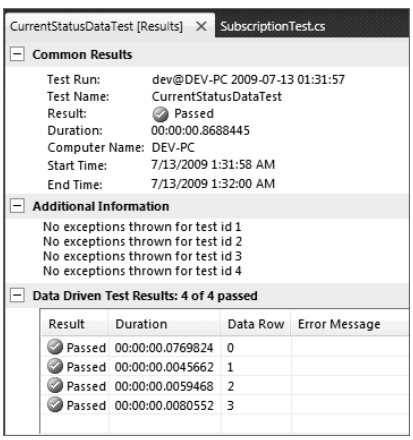


Рис. 11.8

После выполнения теста открывается окно Test Results, в котором перечислены все выполненные тестовые варианты и их результаты. Окно Test Results Details, показанное на рис. 11.8, содержит всю дополнительную информацию, которая была получена при выполнении тестового варианта. Для того чтобы просмотреть это окно, необходимо дважды щелкнуть на тестовом варианте в окне Test Results.

На рис. 11.8 показан раздел Additional Information, содержащий результаты выполнения метода `WriteLine`, добавленного к тестовому методу. Несмотря на то что мы добавили в тестовый метод только одну строку, метод `WriteLine` был выполнен для каждой строки из таблицы базы данных. Раздел Data Driven Test Results, показанный на рис. 11.8, содержит дополнительную информацию о каждой из проверок, причем каждая строка в этом окне соответствует отдельной строке таблицы. Результаты конкретного тестирования могут отличаться от показанных на рис. 11.8, так как это зависит от кода в классе `Subscription`.

Модульное тестирование повышенной сложности

До сих пор мы изучали, как писать и выполнять модульные тесты. В этом разделе мы покажем, как добавить в тестовый вариант дополнительные специальные свойства и использовать тот же каркас для проверки закрытых методов и свойств.

Специальные свойства

Каркас для тестирования содержит большое количество тестовых атрибутов, которые можно применить к методу для записи дополнительной информации о тестовом варианте. Эту информацию можно редактировать с помощью окна Properties. Кроме того, с его помощью можно обновить соответствующие атрибуты тестового метода. Иногда возникает необходимость снабдить тестовые методы специальными свойствами, которые также можно задать с помощью окна Properties. Для этого к тестовому методу следует добавить атрибуты `TestProperty`.

Например, следующий код добавляет в тестовый метод два атрибута, позволяющие задавать произвольную дату и ожидаемое состояние. Для специального тестирования бывает удобно использовать окна Test View и Properties.



VB

```
<TestMethod()>
<TestProperty("SpecialDate", "1/1/2008")>
<TestProperty("SpecialStatus", "Suspended")>
Public Sub SpecialCurrentStatusTest()
    Dim target As New Subscription

    target.PaidUpTo =
CType(Me.TestContext.Properties.Item("SpecialDate"), _
         Date)

    Dim val As Subscription.Status = _
        [Enum].Parse(GetType(Subscription.Status), _
         CStr(Me.TestContext.Properties.Item("SpecialStatus")))

    Assert.AreEqual(val, target.CurrentStatus, _
         "Correct status not set for Paidup date {0}", target.PaidUpTo)
End Sub
```

Сниппет кода SubscriptionTests\SubscriptionTest.vb



C#

```
[TestMethod]
[TestProperty("SpecialDate", "1/1/2008")]
[TestProperty("SpecialStatus", "Suspended")]
public void SpecialCurrentStatusTest()
{
  var target = new Subscription();

    target.PaidUpTo = this.TestContext.Properties["SpecialDate"] as
DateTime?;

    var val = Enum.Parse(typeof(Subscription.Status),
        this.TestContext.Properties["SpecialStatus"] as string);
```

```
        Assert.AreEqual(val, target.CurrentStatus,
              "Correct status not set for Paidup date {0}" , target.PaidUpTo);
    }
```

Сниппет кода SubscriptionTests\SubscriptionTest.cs

Как показано на рис. 11.9, открыв окно Test View, перейдя к заданному тестовому варианту и открыв окно Properties, пользователь может увидеть, что этот код генерирует два дополнительных свойства: `SpecialDate` и `SpecialStatus`.

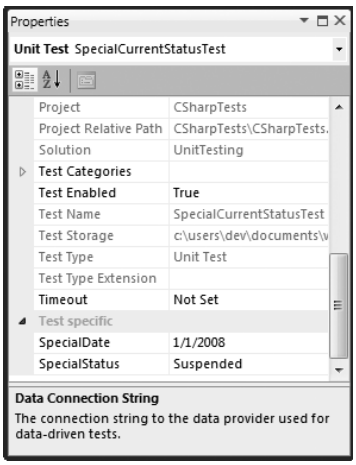


Рис. 11.9

Значения атрибутов `SpecialDate` и `SpecialStatus` можно задать с помощью окна Properties. К сожалению, существуют ограничения, не позволяющие задавать тип для этих значений. В результате сетка свойств выводится и редактируется так, будто все данные имеют тип `String`.



В предыдущей версии системы Visual Studio словарь `TestContext.Properties` не заполнялся автоматически и разработчик должен был делать это с помощью метода `TestInitialize`. В системе Visual Studio 2010 этот недостаток устранен.

Тестирование закрытых членов

Модульное тестирование является особенно эффективным при проверке правильности работы функций-членов класса. Это объясняется тем, что если компоненты корректно работают по отдельности, то возрастает вероятность того, что они будут правильно работать вместе. Действительно, модульное тестирование можно применять для проверки совместной работы нескольких тестовых классов. Однако стоит подчеркнуть, насколько хорошо каркас модульного тестирования справляется с проверкой закрытых методов.

Одной из особенностей платформы .NET является способность динамически создавать объекты любого типа и выполнять любой член независимо от его доступности. Эта функциональная возможность, называемая *отражением* (reflection), достигается за счет снижения производительности, поскольку динамические вызовы методов связа-

ны с дополнительным уровнем переадресации, которые при частом применении приводят к повышенным затратам времени. Тем не менее при тестировании отражение позволяет погружаться во внутреннюю работу класса и не беспокоиться о потенциальном снижении производительности.

Другой, более важный аспект, связанный с использованием механизма отражения для доступа к закрытым членам класса, заключается в том, что получаемый при этом код является несколько запутанным. К счастью, система Visual Studio 2010 очень хорошо генерирует классы-оболочки, облегчающие тестирование даже закрытых методов. Для того чтобы продемонстрировать это, вернемся к свойству `CurrentStatus`, изменим уровень его доступа с открытого на закрытый и переименуем его в `PrivateCurrentStatus`. Теперь сгенерируем модульный тест для этого свойства заново. В результате получим следующий сниппет.



VB

```
<TestMethod(), _
DeploymentItem("Subscriptions.dll")> _
Public Sub PrivateCurrentStatusTest()
    Dim target As Subscription_Accessor = New Subscription_Accessor()
    ' TODO: Initialize to an appropriate value
    Dim actual As Subscription.Status
    actual = target.PrivateCurrentStatus
    Assert.Inconclusive("Verify the correctness of this test method.")
End Sub
```

Сниппет кода SubscriptionTests\SubscriptionTest.vb



C#

```
[TestMethod()]
[DeploymentItem("Subscriptions.dll")]
public void PrivateCurrentStatusTest()
{
    Subscription_Accessor target = new Subscription_Accessor();
    Subscription.Status actual;
    actual = target.PrivateCurrentStatus;
    Assert.Inconclusive("Verify the correctness of this test method.");
}
```

Сниппет кода SubscriptionTests\SubscriptionTest.cs

В предыдущем примере для доступа к свойству `PrivateCurrentStatus` используется экземпляр нового класса `Subscription_Accessor`, который был автоматически сгенерирован и включен в новую сборку системой Visual Studio. Кроме того, в тестовый проект был добавлен новый файл с расширением `.accessor`. Именно это заставляет систему Visual Studio создавать новые *классы доступа* (accessor classes).

Класс доступа к закрытым членам можно добавить в тестовый проект без генерирования модульного теста. Для этого необходимо открыть класс для доступа к закрытым членам и выбрать команду Create Private Accessor из контекстного меню редактора.



Нет необходимости создавать класс доступа к закрытым членам для каждого класса в проекте по отдельности. Каждый файл с расширением *.accessor*, по существу, связан со всем проектом и создает класс доступа для всех классов, включенных в проект.

Тестирование контрактов

Использование нового свойства Code Contracts, описанного в главе 13, позволяет разработчику писать тесты для верификации контрактов. Для того чтобы сделать это, проще всего открыть страницу Code Contracts и снять флажок Assert on Contract Failure. При этом интерфейс Code Contracts API генерирует исключения, а не нарушение *условий* (assertions). Если известен тип ожидаемого исключения, то можно проверить его атрибут `ExpectedException`. По умолчанию система Code Contracts генерирует исключения, которые могут быть переброшены в другие места кода, и их тип в ходе выполнения неизвестен. Многие из контрактных методов имеют перегруженные варианты, принимающие тип исключения как обобщенный параметр.

Рассмотрим простой класс, выполняющий математическую операцию с положительными целыми числами, и модульный тест, проверяющий случай, когда ему передается отрицательное число.



VB

```
Class Calculator
    Public Function Factorial(ByVal n As Integer) As Integer
       Contract.Requires(Of ArgumentOutOfRangeException)(n > = 0, "n")

       If n = 0 Then Return 1
       Return n * Factorial(n-1)
    End Function
End Class

<TestMethod(), ExpectedException(GetType(ArgumentOutOfRangeException))>
Public Sub NegativeTest()

    Dim generator As New Calculator()
    Dim actual = generator.Factorial(-1)

    Assert.Fail("Contract not working")
End Sub
```

Сниппет кода CodeContracts1\CalculatorTests.vb



C#

```
class Calculator
{
    public int Factorial(int n)
    {
       Contract.Requires < ArgumentOutOfRangeException > (n > = 0, "n");

       if (n == 0) return 1;
       return n * Factorial(n-1);
    }
}

[TestMethod, ExpectedException(typeof(ArgumentOutOfRangeException))]
public void NegativeTest()
{
     var generator = new Calculator();
     var actual = generator.Factorial(-1);
     Assert.Fail("Contract not working");
}
```

Сниппет кода CodeContracts1\CalculatorTests.cs

Несмотря на то что этот метод тестирования контракта вполне работоспособен, мы не рекомендуем его использовать, потому что он может замаскировать ошибки, существующие в коде. Лучше вызвать систему Code Contracts и заместить ее поведение, заданное по умолчанию, в самом тестовом проекте. Для этого следует зарегистрировать событие `ContractFailed` для статического класса `Contract` в методе `AssemblyInitialize`. В этом случае обработчик события сообщает интерфейсу `Code Contracts API`, что пользователь распознал нарушение контракта и хочет сгенерировать соответствующее исключение.



VB

```
<AssemblyInitialize()>
Public Shared Sub AssemblyInitialize(ByVal testContext As TestContext)

AddHandler Contract.ContractFailed, Sub(sender As Object,
                   e As ContractFailedEventArgs)
                   e.SetHandled()
                   e.SetUnwind()
               End Sub
End Sub

<TestMethod(), ExpectedException(GetType(Exception),
AllowDerivedTypes:=True)>
Public Sub NegativeTest()

    Dim generator As New Calculator()
    Dim actual = generator.Factorial(-1)

    Assert.Fail("Contract not working")
End Sub
```

Сниппет кода CodeContracts2\CalculatorTests.vb



C#

```
[AssemblyInitialize]
public static void AssemblyInitialize(TestContext testContext)
{
    Contract.ContractFailed += (s, e) = >
    {
         e.SetHandled();
         e.SetUnwind();
    };
}

[TestMethod, ExpectedException(typeof(Exception), AllowDerivedTypes = true)]
public void NegativeTest()
{
    var generator = new Calculator();
    var actual = generator.Factorial(-1);
    Assert.Fail("Contract not working");
}
```

Сниппет кода CodeContracts2\CalculatorTests.cs



Если система Code Contracts настроена на работу с условиями `Asserts`, то исключение будет потеряно, поэтому в рассмотренном примере предусмотрена проверка для любого подкласса класса `Exception`. В действительности генерируется исключение `System.Diagnostics.Contracts.ContractException`, которое является закрытым в каркасе .NET Framework, так что его невозможно выявить непосредственно.

Управление большим количеством тестов

Для перечисления всех тестов, существующих в решении, в системе Visual Studio предусмотрены окна Test View и Test List Editor. Окно Test View, показанное на рис. 11.2, выводит на экран модульные тесты в виде простого списка. Однако, если в решении существуют сотни и даже тысячи модульных тестов, управлять ими с помощью простого списка неудобно. Окно Test List Editor позволяет группировать и организовывать тесты с помощью отдельных списков. Поскольку списки тестов могут содержать как сами тесты, так и другие списки тестов, возникает необходимость в создании логичной иерархической структуры. Все тесты в списках могут выполняться вместе либо системой Visual Studio, либо какой-то отдельной тестовой утилитой.

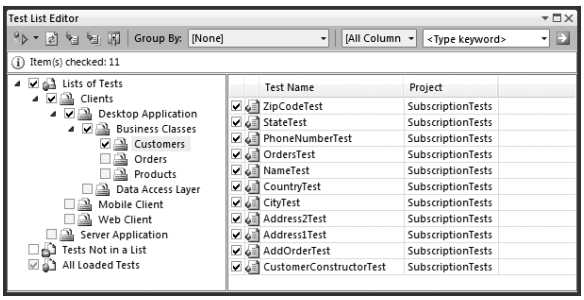


Рис. 11.10

Окно Test List Editor можно открыть с помощью меню Test Windows или дважды щелкнуть на файле тестовых метаданных (с расширением `.vsmdi`). На рис. 11.10 показано окно Test List Editor для решения, содержащего большое количество тестов, организованных в иерархическую структуру.

В левой части окна Test List Editor приведено иерархическое дерево списков тестов, доступных в текущем решении. В нижней части дерева находятся два списка проектов. Один из них содержит все тестовые варианты (All Loaded Tests), а второй — тестовые варианты, не включенные в список (Tests Not in a List). Ниже узла Lists of Tests расположены все списки тестов, созданные для проекта. Для того чтобы создать новый список тестов, необходимо выполнить команду Test⇨Create New Test List. Тестовые варианты можно перетащить из любого существующего списка в новый список. Сначала эта возможность немного сбивает с толку, поскольку тест перемещается в новый список и удаляется из старого. Для того чтобы добавить тестовый вариант сразу в несколько списков, необходимо либо удерживать нажатой клавишу <Ctrl> в процессе перетаскивания тестового варианта, либо скопировать тестовый вариант из первоначального списка в новый.

Создав новый список тестов, можно выполнить все тесты из списка, установив флажок, расположенный сразу за списком в окне Test Manager. Выполнение всего отмеченного списка тестов запускается кнопкой Run. В качестве альтернативы можно запустить список тестов с помощью отладчика, выполнив команду Debug Checked Tests.

Резюме

В главе описано использование модульного тестирования для проверки правильности работы кода. Каркас модульного тестирования в системе Visual Studio является довольно обширным. Он позволяет работать с документацией и управлять тестовыми вариантами.

Для того чтобы минимизировать объем кода, который необходимо написать для того, чтобы полностью выполнить тестирование, можно использовать соответствующий источник данных. Кроме того, каркас тестирования можно расширить и проверить все внутренние элементы приложения.

Версии Premium и Ultimate системы Visual Studio имеют еще больше функциональных возможностей для тестирования, включая способность отслеживать и регистрировать достижимость кода, а также проверку загружаемых приложений и веб-приложений. Более подробно эти возможности рассматриваются в главе 56.

12

Документирование с помощью XML-комментариев

В ЭТОЙ ГЛАВЕ...

- Создание встроенной документации с помощью XML-комментариев
- Использование надстройки GhostDoc для автоматической генерации XML-комментариев
- Создание отдельной документации на основе XML-комментариев с помощью генератора документации Sandcastle
- Использование комментариев Task List для отслеживания незаконченных заданий

Документация — критически важный, хотя и часто недооцениваемый, элемент процесса разработки программного обеспечения. Без документации другие программисты, аналитики и менеджеры сталкиваются со сложностями при анализе и реализации кода. Без документации даже сам автор может запутаться в своем коде, как только он станет слишком сложным. Следовательно, внутренняя документация может существенно облегчить процесс разработки.

XML-комментарии — это способ создания внутренней документации, позволяющий программистам не писать отдельный документ. Вместо этого по мере разработки своего кода они могут включать в начало каждого определения метаданные, объясняющие предназначение кода. Как только эта информация будет включена в код, она будет передана системой Visual Studio подсистемам Object Browser и IntelliSense.

Надстройка GhostDoc, являющаяся внешней по отношению к системе Visual Studio, автоматически вставляет блок XML-комментариев в класс или член класса.

Кроме того, существует набор инструментов Sandcastle, работающий как генератор документации. Эти инструменты позволяют по вставленным разработчиком XML-комментариям легко создать отдельную документацию либо в виде справочного файла HTML по технологии компании Microsoft, либо файл в формате Microsoft Help 2.

Встроенное комментирование

Все языки программирования, поддерживаемые системой Visual Studio, поддерживают создание встроенной документации. По умолчанию все встроенные комментарии выделяются зеленым цветом.

Язык C# поддерживает как однострочные комментарии, так и блоки комментариев. Однострочные комментарии отмечаются символами // в начале комментария. Как показано в следующем коде, блок комментариев обычно занимает несколько строк, открывается символами /* и закрывается символами */.

C#

```
// Вычисляет факториал целого числа
public int Factorial(int number)
{
    /* Эта функция вычисляет факториал,
       используя итерационный метод.
    */
    int intermediateResult = 1;
    for (int factor = 2; factor <= number; factor++)
    {
        intermediateResult = intermediateResult * factor;
    }
    return intermediateResult; //The calculated factorial
}
```

Как показано в следующем фрагменте кода, в языке VB используется одинарная кавычка, означающая, что за ней следует комментарий.

VB

```
' Вычисляет факториал целого числа
Public Function Factorial(ByVal number As Integer) As Integer
    ' Эта функция вычисляет факториал,
    ' используя итерационный метод.
    '
    Dim intermediateResult As Integer = 1
    For factor As Integer = 2 To number
        intermediateResult = intermediateResult * factor
    Next
    Return intermediateResult 'Вычисленное значение факториала
End Function
```

XML-комментарии

XML-комментарии — это специализированные комментарии, включенные в код. Когда проект готов к сборке, система Visual Studio может выполнить дополнительный этап, чтобы сгенерировать XML-файл на основе этих комментариев и предоставить информацию о пользовательских типах, например классах или отдельных членах классов (определенных пользователем или кем-то другим), включая события, функции и свойства.

XML-комментарии могут содержать любое сочетание XML- и HTML-дескрипторов. Как будет показано в этой главе, система Visual Studio выполняет специальную обработку конкретного множества заранее определенных дескрипторов. Все остальные дескрипторы включаются в генерируемый файл документации как есть.

Добавление XML-комментариев

XML-комментарии добавляются непосредственно перед определением свойства, метода или класса. Система Visual Studio автоматически добавляет блок XML-комментариев, когда пользователь набирает символы /// в коде на языке C# перед объявлением члена или класса. В некоторых случаях XML-комментарии уже содержатся в коде шаблонных проектов, как показано на рис. 12.1.

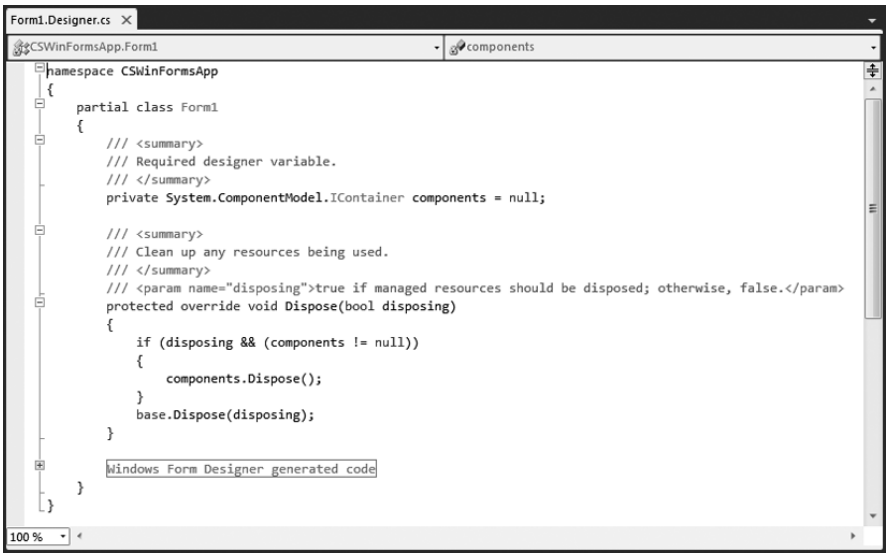


Рис. 12.1



Автоматическое добавление резюме можно включить или выключить с помощью настроек системы Visual Studio. Выберите команду Select Tools⇨Options, а затем Text Editor⇨C#⇨Advanced в навигационном дереве. Для того чтобы отключить автоматическое добавление резюме, сбросьте флажок Generate XML documentation comments for ///.

Добавление блока XML-комментариев в программу на языке VB обеспечивается вставкой символа ‘'’. В остальном язык VB воспроизводит способ генерации документов, принятый в языке C#.

В обоих языках после добавления комментария система Visual Studio автоматически добавляет в левом поле пиктограмму, с помощью которой можно скрывать документацию, если программист хочет сосредоточить свое внимание только на коде. Если оставить курсор мыши в этой области окна, то на экране появится подсказка, содержащая первые несколько строк блока комментариев.

Дескрипторы XML-комментариев

Несмотря на то что пользователь может использовать любой вид комментариев, включая свои собственные XML-дескрипторы, обработчик XML-комментариев в системе Visual Studio распознает только заранее определенные дескрипторы и автоматически выполняет соответствующее форматирование. Генератор документации Sandcastle, который будет рассмотрен ниже, поддерживает много дополнительных дескрипторов, которые могут обогатить XML-схему документа.



Если в тексте комментария желательно использовать угловые скобки, то следует применять ссылки на сущности `<` и `>`.

Поскольку документация играет такую важную роль, в следующем разделе главы мы подробно рассмотрим все заранее определенные дескрипторы, их синтаксис, а также использование в документации.

Дескриптор `<c>`

Дескриптор `<c>` означает, что вложенный текст должен форматироваться как код, а не как обычный текст. Он применяется к коду, включенному в обычный блок текста. Дескриптор `<c>` имеет простую структуру, в которой любой текст, появляющийся между дескрипторами открытия и закрытия, помечается как предназначенный для форматирования в качестве кода.

```
<c> code-formatted text </c>
```

В следующем примере показано, как можно использовать дескриптор `<c>` для описания свойства.

C#

```
/// <summary>
/// Свойство <c>UserId</c> используется в сочетании с другими свойствами
/// для идентификации пользователя. Не забудьте задать поле <c>Password</c>.
/// </summary>
public string UserId { get; set; }
```

VB

```
''' <summary>
''' Свойство <c>UserId</c> используется в сочетании с другими свойствами
''' для идентификации пользователя. Не забудьте задать поле <c>Password</c>.
''' </summary>
Public Property UserId() As String
```

Дескриптор `<code>`

Если большой фрагмент текста в документации, который необходимо отформатировать как код, представляет собой не отдельную фразу в блоке обычного текста, то вместо дескриптора `<c>` можно использовать дескриптор `<code>`. Этот дескриптор помечает весь текст внутри себя как код. Тем не менее он относится к блочным, а не символьным дескрипторам. Синтаксическая структура этого дескриптора состоит из дескрипторов открытия и закрытия с форматируемым текстом внутри.

```
<code>
Текст, форматируемый как код
Текст, форматируемый как код
</code>
```

Дескриптор <code> можно внедрять в любой другой дескриптор XML-комментария. В следующем фрагменте приведен пример использования этого дескриптора в резюме определения свойства.

C#

```
/// <summary>
/// Свойство <c>UserId</c> используется в сочетании с другими свойствами
/// для идентификации пользователя. Не забудьте задать поле <c>Password</c>.
/// Например:
/// <code>
/// myUser.UserId = "daveg"
/// myUser.Password = "xg4*Wv"
/// </code>
/// </summary>
public string UserId { get; set; }
```

VB

```
''' <summary>
''' Свойство <c>UserId</c> используется в сочетании с другими свойствами
''' для идентификации пользователя. Не забудьте задать поле <c>Password</c>.
''' Например:
''' <code>
''' myUser.UserId = "daveg"
''' myUser.Password = "xg4*Wv"
''' </code>
''' </summary>
Public Property UserId() As String
```

Дескриптор <example>

Как правило, внутренняя документация должна содержать пример использования конкретной процедуры или метода. Дескриптор <example> означает, что вложенный блок должен интерпретироваться как отдельный раздел документации, посвященный демонстрации примера для заданного члена класса.

По существу, этот дескриптор лишь помогает организовать документацию, но в сочетании со специально разработанным листом стиля XML или инструкциями обработки он позволяет отформатировать пример должным образом.

Для полноты примера в текст, ограниченный дескрипторами <example>, можно включать другие дескрипторы XML-комментариев, такие как <c> и <code>. Синтаксическая структура этого блочного дескриптора имеет следующий вид.

```
<example>
   Здесь может быть любой текст примера.
</example>
```

Используя предыдущий пример, перенесем текст, отформатированный дескриптором <code>, из раздела <summary> в раздел <example>.

C#

```
/// <summary>
/// Свойство <c>UserId</c> используется в сочетании с другими свойствами
/// для идентификации пользователя. Не забудьте задать поле <c>Password</c>.
/// </summary>
/// <example>
/// <code>
/// myUser.UserId = "daveg"
/// myUser.Password = "xg4*Wv"
```

```
/// </code>
/// </example>
public string UserId { get; set; }
```

VB

```
''' <summary>
''' Свойство <c>UserId</c> используется в сочетании с другими свойствами
''' для идентификации пользователя. Не забудьте задать поле <c>Password</c>.
''' </summary>
''' <example>
''' <code>
''' myUser.UserId = "daveg"
''' myUser.Password = "xg4*Wv"
''' </code>
''' </example>
Public Property UserId() As String
```

Дескриптор <exception>

Дескриптор <exception> используется для определения любых исключений, которые могут передаваться из члена, связанного с текущим блоком XML-документации. Каждое передаваемое исключение должно быть определено в отдельном блоке <exception> с атрибутом cref, идентифицирующим полностью квалифицированное имя типа исключения, которое может быть сгенерировано. Обратите внимание на то, что в системе Visual Studio 2010 процессор XML-комментариев проверяет синтаксис блока исключений, чтобы заставить разработчика вставить в блок указанный атрибут. Он также гарантирует, что в коде не будет нескольких блоков <exception> с одним и тем же значением атрибута. Полный синтаксис этого дескриптора выглядит следующим образом.

```
<exception cref="exceptionName">
   Описание исключения.
</exception>
```

Уточняя пример из предыдущих разделов, добавим два определения исключений в XML-комментарии, связанные со свойством UserId, System.TimeoutException и System.UnauthorizedAccessException.

C#

```
/// <summary>
/// Свойство <c>UserId</c> используется в сочетании с другими свойствами
/// для идентификации пользователя. Не забудьте задать поле <c>Password</c>.
/// </summary>
/// <exception cref="System.TimeoutException">
/// Генерируется, если код не может авторизовать пользователя
/// за разумное время
/// </exception>
/// <exception cref="System.UnauthorizedAccessException">
/// Генерируется, если идентификатор пользователя в текущем контексте
/// является некорректным.
/// </exception>
/// <example>
/// <code>
/// myUser.UserId = "daveg"
/// myUser.Password = "xg4*Wv"
/// </code>
/// </example>
public string UserId { get; set; }
```

VB

```
''' <summary>
''' Свойство <c>UserId</c> используется в сочетании с другими свойствами
''' для идентификации пользователя. Не забудьте задать поле <c>Password</c>.
''' </summary>
''' <exception cref="System.TimeoutException">
''' Генерируется, если код не может авторизовать пользователя
''' за разумное время
''' </exception>
''' <exception cref="System.UnauthorizedAccessException">
''' Генерируется, если идентификатор пользователя в текущем контексте
''' является некорректным.
''' </exception>
''' <example>
''' <code>
''' myUser.UserId = "daveg"
''' myUser.Password = "xg4*Wv"
''' </code>
''' </example>
Public Property UserId() As String
```

Дескриптор <include>

Часто документацию необходимо разделить между несколькими проектами. В других ситуациях один человек может отвечать за документацию, а остальные — за кодирование. Как бы то ни было, в таких случаях полезно использовать дескриптор <include>. Он позволяет ссылаться на комментарии, находящиеся в отдельном XML-файле, так, будто они вставлены непосредственно в документацию. Используя этот метод, реальную документацию можно вынести за пределы кода. Это особенно удобно, когда комментарии становятся слишком обширными.

Синтаксис дескриптора <include> требует, чтобы пользователь указал, какую часть внешнего файла следует использовать в текущем контексте. Атрибут path позволяет идентифицировать путь к XML-узлу и использует стандартную терминологию XPath.

```
<include file="filename" path="XPathQuery" />
```

Путь к внешнему XML-файлу, содержащему дополнительную документацию, должен соответствовать заданному атрибуту, причем конечный узел должен содержать атрибут имени, чтобы однозначно идентифицировать раздел XML-документа, который необходимо включить в документацию.

Включать файлы в программах на языках VB и C# можно с помощью одного и того же дескриптора. В следующем фрагменте свойство принимает примеры, использованные в обсуждении дескриптора <exception>, и переносит документацию во внешний файл.

C#

```
/// <include file="externalFile.xml"
/// path="MyDoc/Properties[@name='UserId']/*" />
public string UserId { get; set; }
```

VB

```
''' <include file="externalFile.xml"
''' path="MyDoc/Properties[@name='UserId']/*" />
Public Property UserId() As String
```

Внешние файлы можно заполнить с использованием следующей структуры XML-документа, чтобы синхронизировать их с результатом, ожидаемым после применения дескриптора <include>.

```
<MyDoc>
  <Properties name="UserId">
    <summary>
        Для идентификации того, кто вызывает процедуру, используется объект
        <c> sender </c>.
    </summary>
    <summary>
        Свойство <c>UserId</c> используется в сочетании с другими свойствами
        для идентификации пользователя. Не забудьте задать поле <c>Password</c>.
    </summary>
    <exception cref="System.TimeoutException">
        Генерируется, если код не может авторизовать пользователя за разумное
    </exception>
    <exception cref="System.UnauthorizedAccessException">
        Генерируется, если идентификатор пользователя в текущем контексте
        является некорректным.
    </exception>
    <example>
      <code>
        myUser.UserId = "daveg"
        myUser.Password = "xg4*Wv"
      </code>
    </example>
  </Procedures>
</MyDoc>
```

Дескриптор `<list>`

Некоторые виды документации содержат списки разных описаний. С помощью дескриптора `<list>` пользователь может сгенерировать нумерованные и ненумерованные списки, а также таблицы, состоящие из двух столбцов. Каждый элемент списка определяется двумя параметрами — термом и описанием, — заданными своими собственными дескрипторами и определяющими процедуру создания документации.

Для создания списка в документе необходимо использовать следующий синтаксис, где параметр `type` может иметь одно из следующих значений: `bullet`, `numbered` или `table`.

```
<list type="type">
   <listheader>
      <term> termName </term>
      <description> description </description>
   </listheader>
   <item>
      <term> myTerm </term>
      <description> myDescription </description>
   </item>
</list>
```

Блок `<listheader>` является необязательным и обычно используется для табулированных или определения списков. Определения списков должны содержать дескриптор `<term>`, но при создании маркированных, нумерованных списков и таблиц этот дескриптор можно пропускать.

Разметку для каждого типа списков можно форматировать по отдельности с помощью листа стиля XML. Пример использования дескриптора `<list>` приведен в следующем фрагменте. Обратите внимание на то, что в этом примере пропущен дескриптор `listheader`, поскольку для маркированных списков он не обязателен.

C#

```
/// <summary>
/// Эта функция изменяет пароль пользователя. Пароль может отклоняться
/// по нескольким причинам:
/// <list type="bullet">
/// <item>
/// <term> Too Short </term>
/// <description> Новый пароль слишком короткий. </description>
/// </item>
/// <item>
/// <term> Not Complex </term>
/// <description> Новый пароль слишком простой. Пароль должен содержать
/// хотя бы один из следующих символов: букву в нижнем регистре,
    букву в верхнем
/// регистре и число.
/// </description>
/// </item>
/// </list>
/// </summary>
public bool ChangePwd(string oldPwd, string newPwd)
    {
      //...код...
      return true;
    }
```

VB

```
''' <summary>
''' Эта функция изменяет пароль пользователя. Пароль может отклоняться
''' по нескольким причинам:
''' <list type="bullet">
''' <item>
''' <term> Too Short </term>
''' <description> Новый пароль слишком короткий. </description>
''' </item>
''' <item>
''' <term> Not Complex </term>
''' <description> Новый пароль слишком простой. Пароль должен содержать
''' хотя бы один из следующих символов: букву в нижнем регистре,
    букву в верхнем
''' регистре и число.
''' </description>
''' </item>
''' </list>
''' </summary>
Public Function ChangePwd(ByVal oldPwd As String, ByVal newPwd As String) _
                          As Boolean
    '...код...
    Return True
End Function
```

Дескриптор <para>

Без использования внутренних XML-комментариев на уровне блока, таких как <list> и <code>, текст, добавляемый в разделы <summary>, <remarks> и <returns>, просто объединяется в одно целое.

Для того чтобы разделить его на удобочитаемые фрагменты, можно использовать дескриптор <para>, который означает, что вложенный текст должен рассматриваться как отдельный абзац. Синтаксис этого дескриптора прост.

```
<para>Этот текст выводится в отдельном абзаце.</para>
```

Дескриптор <param>

Для того чтобы объяснить предназначение всех параметров в объявлении функции, можно использовать дескриптор <param>. Этот дескриптор применяется генератором XML-комментариев системы Visual Studio к каждому экземпляру, атрибут имени которого совпадает с именем какого-то свойства. Текст, который находится между открытием и закрытием дескриптора <param>, представляет собой описание параметра.

```
<param name="parameterName">определение параметра.</param>
```

XML-процессор не позволяет создавать несколько дескрипторов <param> для одного параметра или дескрипторы для несуществующих параметров, выдавая сообщения, которые появляются в списке Error List системы Visual Studio. В следующем примере показано, как дескриптор <param> используется для описания двух параметров функции.

C#

```
/// <param name="oldPwd">Старый пароль должен совпадать с текущим</param>
/// <param name="newPwd">Новый пароль должен быть достаточно сложным</param>
public bool ChangePwd(string oldPwd, string newPwd)
{
    //...код...
    return true;
}
```

VB

```
''' <param name="oldPwd"> Старый пароль должен совпадать с текущим </param>
''' <param name="newPwd"> Новый пароль должен быть достаточно сложным</param>
Public Function ChangePwd(ByVal oldPwd As String, ByVal newPwd As String) _
                          As Boolean
    '...код...
    Return True
End Function
```



Дескриптор <param> особенно полезен для документирования предусловий параметров метода, например, если не допускаются их нулевые значения.

Дескриптор <paramref>

Если возникает необходимость сослаться на параметр метода в каком-либо месте документации, в котором нет дескриптора <param>, для форматирования его значения и даже для установления связи с информацией о параметре можно использовать дескриптор <paramref>. Компилятор не требует, чтобы имя этого параметра существовало, но пользователь может указать текст, который должен использоваться в атрибуте name, следуя следующей синтаксической конструкции.

```
<paramref name="parameterName" />
```

Как правило, дескрипторы <paramref> используются, когда пользователь ссылается на параметры в более крупных разделах, чем разделы <summary> и <remarks>. Рассмотрим пример.

C#

```
/// <summary>
/// Эта функция изменяет пароль пользователя. Она генерирует исключение, если
/// имена <paramref name="oldPwd" /> и <paramref name="newPwd" /> не заданы.
```

```
/// </summary>
/// <param name="oldPwd">Старый пароль должен совпадать с текущим</param>
/// <param name="newPwd">Новый пароль должен быть достаточно сложным</param>
public bool ChangePwd(string oldPwd, string newPwd)
{
    //...код...
    return true;
}
```

VB

```
''' <summary>
''' Эта функция изменяет пароль пользователя. Она генерирует исключение, если
''' имена <paramref name="oldPwd" /> и <paramref name="newPwd" /> не заданы.
''' </summary>
''' <param name="oldPwd">Старый пароль должен совпадать с текущим</param>
''' <param name="newPwd">Новый пароль должен быть достаточно сложным</param>
Public Function ChangePwd(ByVal oldPwd As String, ByVal newPwd As String) _
                          As Boolean
    '...код...
    Return True
End Function
```

Дескриптор <permission>

Для того чтобы описать уровень доступа к коду, требуемый для конкретного метода, используется дескриптор <permission>. Он требует атрибута cref для ссылки на конкретный тип разрешения.

```
<permission cref="permissionName">
   здесь находится описание
</permission>
```

Если функция требует нескольких разрешений, следует использовать несколько блоков <permission>, как показано в следующем примере.

C#

```
/// <permission cref="System.Security.Permissions.RegistryPermission">
/// Требует полного доступа к регистру системы Windows.
/// </permission>
/// <permission cref="System.Security.Permissions.FileIOPermission">
/// Требует полного доступа к файлу.config, содержащему информацию о приложении.
/// </permission>
public string UserId { get; set; }
```

VB

```
''' <permission cref="System.Security.Permissions.RegistryPermission">
''' Требует полного доступа к регистру системы Windows.
''' </permission>
''' <permission cref="System.Security.Permissions.FileIOPermission">
''' Требует полного доступа к файлу.config, содержащему информацию о приложении.
''' </permission>
Public Property UserId() As String
```

Дескриптор <remarks>

Дескриптор <remarks> используется для того, чтобы вставить в документацию дополнительный блок комментариев, связанных с конкретным методом. Обсуждение предыдущих дескрипторов уже демонстрировало действие дескриптора <remarks>. Его синтаксическая конструкция приведена ниже.

```
<remarks>
   Любые дополнительные замечания
</remarks>
```

Как правило, документация должна содержать резюме, в котором кратко описывается метод или тип, а затем — подробную информацию в дескрипторе <remarks>, включающую в себя ожидаемые результаты обращения к методу.

Дескриптор <returns>

С помощью дескриптора <returns> можно описать, что должно произойти, когда метод возвращает значение в точку вызова. Синтаксис дескриптора <returns> похож на синтаксис остальных блочных дескрипторов. Он состоит из дескрипторов открытия и закрытия, а между ними расположена информация, описывающая возвращаемое значение.

```
<returns>
   Описание возвращаемого значения.
</returns>
```

Рассмотрим простую реализацию дескриптора <returns> в следующем коде.

C#

```
/// <summary>
/// Эта функция изменяет пароль пользователя.
/// </summary>
/// <returns>
/// Эта функция возвращает:
/// Значение <c>True</c>, которое означает, что пароль изменен успешно,
/// или значение <c>False</c>, которое означает, что произошел сбой.
/// </returns>
public bool ChangePwd(string oldPwd, string newPwd)
{
    //...код...
    return true;
}
```

VB

```
''' <summary>
''' Эта функция изменяет пароль пользователя.
''' </summary>
''' <returns>
''' Эта функция возвращает:
''' Значение <c>True</c>, которое означает, что пароль изменен успешно,
''' или значение <c>False</c>, которое означает, что произошел сбой.
''' </returns>
Public Function ChangePwd(ByVal oldPwd As String, ByVal newPwd As String) _
                          As Boolean
    '...код...
    Return True
End Function
```



Кроме значения, возвращаемого функцией, дескриптор <returns> особенно полезен при описании любых ожидаемых постусловий.

Дескриптор <see>

Используя дескриптор <see>, можно добавить в проект ссылки на другие элементы. Как и некоторые другие дескрипторы, рассмотренные выше, дескриптор <see> требует атрибута cref со значением, ссылающимся на существующий член, будь то

свойство, метод или определение класса. Дескриптор <see> может использоваться в других областях документации, например <summary> или <remarks>. Синтаксис этого дескриптора выглядит следующим образом.

```
<see cref="memberName" />
```

Когда система Visual Studio обрабатывает дескриптор <see>, она создает полностью квалифицированный адрес, который затем можно использовать в качестве основы для связи в документации при ее трансформации с помощью листов стиля. Например, ссылаясь на приложение в классе, содержащем функцию ChangePwd , получим следующее значение атрибута cref.

```
<see cref="applicationName.className.ChangePwd"/>
```

В следующем примере дескриптор <see> используется для организации связи с другой функцией — CheckUser.

C#

```
/// <remarks>
/// Использует ссылку <see cref="CheckUser" /> для проверки существования
/// пользователя перед вызовом ChangePwd.
/// </remarks>
public bool ChangePwd(string oldPwd, string newPwd)
{
    //...код...
    return true;
}
```

VB

```
''' <remarks>
''' Использует ссылку <see cref="CheckUser" /> для проверки существования
''' пользователя перед вызовом ChangePwd.
''' </remarks>
Public Function ChangePwd(ByVal oldPwd As String, ByVal newPwd As String) _
As Boolean
   '...код...
   Return True
End Function
```



Если член, указанный в атрибуте cref, не существует, то только для проектов на языке VB система Visual Studio для вывода сообщения и ее добавления в список Error List использует технологию IntelliSense.

Дескриптор <seealso>

Дескриптор <seealso> используется для генерации отдельного раздела, содержащего информацию о соответствующих темах, освещенных в документации. В отличие от дескриптора <see>, дескриптор <seealso> определяется за пределами блоков XML-комментариев, причем каждый экземпляр дескриптора <seealso> требует наличия атрибута cref, содержащего имя свойства, метода или класса, с которым требуется установить связь. Полная синтаксическая конструкция этого дескриптора приведена ниже.

```
<seealso cref="memberName" />
```

Модифицируя предыдущий пример, покажем, как можно реализовать дескриптор <seealso> в коде.

C#

```
/// <remarks>
/// Использует ссылку <see cref="CheckUser" /> для проверки существования
/// пользователя перед вызовом ChangePwd.
/// </remarks>
/// <seealso cref="ResetPwd" />
public bool ChangePwd(string oldPwd, string newPwd)
{
    //...код...
    return true;
}
```

VB

```
''' <remarks>
''' Использует ссылку <see cref="CheckUser" /> для проверки существования
''' пользователя перед вызовом ChangePwd.''' </remarks>
''' <seealso cref="ResetPwd" />
Public Function ChangePwd(ByVal oldPwd As String, ByVal newPwd As String) _
                          As Boolean
    '...код...
    Return True
End Function
```

Дескриптор <summary>

Дескриптор <summary> используется для создания краткого описания, которое размещается в начале каждого тематического раздела в документации. Поэтому он обычно размещается перед всеми открытыми и защищенными методами и классами. Кроме того, раздел <summary> используется системой IntelliSense для обработки специального пользовательского кода. Синтаксис дескриптора <summary> выглядит следующим образом.

```
<summary>
   Описание функции или свойства.
</summary>
```

Дескриптор <typeparam>

Дескриптор <typeparam> обеспечивает информацию о параметрах обобщенных типов или параметрах членов класса. Дескриптор <typeparam> получает атрибут name, содержащий параметр типа.

```
<typeparam name="typeName">
   Описание.
</typeparam>
```

Дескриптор <typeparam> можно использовать в коде на языках C# и VB следующим образом.

C#

```
/// <typeparam name="T">
/// Базовый тип элемента (должен реализовывать тип IComparable)
/// </typeparam>
public class myList <T> where T : IComparable
{
    //...код...
}
```

VB

```
''' <typeparam name="T">
/// Базовый тип элемента (должен реализовывать IComparable)
''' </typeparam>
Public Class myList(Of T As IComparable)
    '...код...
End Class
```

Дескриптор <typeparamref>

Для того чтобы сослаться на параметр обобщенного типа за пределами дескриптора <typeparam>, можно использовать дескриптор <typeparamref>, позволяющий форматировать значение параметра и даже устанавливать связь с информацией о параметре в зависимости от способа трансформации XML-кода.

```
<typeparamref name="parameterName" />
```

Как правило, дескрипторы <typeparamref> используются для ссылки на параметры в более крупных разделах документации, например в дескрипторе <summary> или <remarks>.

C#

```
/// <summary>
/// Создает новый список произвольного типа <typeparamref name="T"/>
/// </summary>
/// <typeparam name="T">
/// Базовый тип элемента (должен реализовывать тип IComparable)
/// </typeparam>
public class myList <T> where T : IComparable
{
    //...код...
}
```

VB

```
''' <summary>
''' Создает новый список произвольного типа <typeparamref name="T"/>
''' </summary>
''' <typeparam name="T">
''' Базовый тип элемента (должен реализовывать тип IComparable)
''' </typeparam>
Public Class myList(Of T As IComparable)
    '...код...
End Class
```

Дескриптор <value>

Дескриптор <value> обычно используется для указания предназначения свойства. Это еще один раздел в XML-документе, где можно разместить информацию о соответствующем члене. Дескриптор <value> не используется системой IntelliSense.

```
<value>Отображаемый текст </value>
```

Дескриптор <summary> обычно используется для описания предназначения свойства, а дескриптор <value> — для описания того, что представляет собой свойство.

C#

```
/// <summary>
/// Свойство <c>UserId</c> используется в сочетании с другими свойствами
/// для идентификации пользователя. Не забудьте задать поле <c>Password</c>.
/// </summary>
/// <value>
/// Строка, содержащая значение UserId для текущего пользователя
```

```
/// </value>
public string UserId { get; set; }
```

VB

```
''' <summary>
''' Свойство <c>UserId</c> используется в сочетании с другими свойствами
''' для идентификации пользователя. Не забудьте задать поле <c>Password</c>.
''' </summary>
''' <value>
''' Строка, содержащая значение UserId для текущего пользователя
''' </value>
Public Property UserId() As String
```

Использование XML-комментариев

Вставив XML-комментарии в свой код, пользователь может создать XML-файл, содержащий документацию. В языке VB эта возможность включена по умолчанию, причем путь для записи файла и его имя также задаются по умолчанию. Однако в языке C# имеется возможность отключить эту функциональную возможность, поэтому, если возникнет потребность в такой документации, соответствующую настройку необходимо включить вручную.

Для того чтобы гарантировать, что генерируемый документ будет записан туда, куда следует, необходимо открыть страницы свойства в проекте с помощью контекстного меню, которое открывается после щелчка правой кнопкой мыши в окне Solution Explorer. Далее необходимо выбрать проект, для которого создается документация, щелкнуть правой кнопкой мыши на его имени в окне Solution Explorer и выбрать команду Properties.

Настройки, связанные с XML-документацией, доступны в меню Build (рис. 12.2). Под разделом команд общего предназначения расположен раздел Output, содержащий флажок, позволяющий включить генерацию XML-файлов. Если этот флажок установлен, то следующее за ним текстовое поле позволяет пользователю ввести имя генерируемого XML-файла.

Для приложений на языке VB флажок, позволяющий генерировать XML-файл, расположен на закладке Compile среди свойств проекта.

Если эти настройки сохранены, то в ходе следующей сборки система Visual Studio добавит опцию компилятора `/doc` и обеспечит генерацию XML-документации.



Генерация XML-файла с документацией замедляет процесс компиляции. Если это влияет на процесс разработки или отладки, то пользователь может отключить эту возможность в режиме Debug, оставив ее в режиме Release.

Сгенерированный XML-файл содержит полный XML-документ, который можно подвергнуть XSL-трансформации или передать другому приложению для обработки с помощью объектной модели XML-документов. Все ссылки на исключения, параметры, методы и другие связи включаются в этот файл наряду с полной информацией об их адресе, включая пространство имен, приложение и класс. Ниже в этой главе мы увидим, как использовать этот XML-файл для создания профессиональной документации с помощью надстройки Sandcastle.

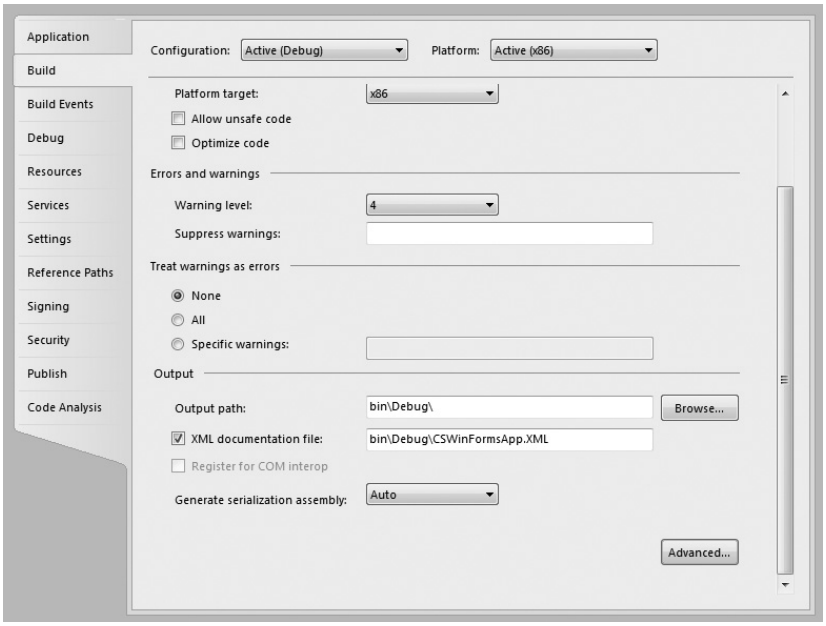


Рис. 12.2

Информация подсистемы IntelliSense

Другим преимуществом XML-комментариев является то, как система Visual Studio использует их в своей подсистеме IntelliSense. После того как пользователь расставит дескрипторы, которые понимает система Visual Studio, она сразу генерирует информацию для подсистемы IntelliSense, после чего пользователь может ссылаться на эту информацию в любом месте своего кода.

Существуют два способа доступа к подсистеме IntelliSense. Если член, на который необходимо сослаться, находится в том же самом или в другом проекте, но в рамках того же самого решения, то для доступа к информации не обязательно создавать или генерировать XML-файл. Однако, даже если проект является внешним по отношению к текущему решению, пользователь все равно может использовать преимущества подсистемы IntelliSense.

Для этого необходимо лишь, чтобы XML-файл, созданный на этапе *сборки* (build), имел то же самое имя, что и *сборка* (assembly), созданная в результате этого процесса. Например, если результатом компиляции является файл `MyApplication.exe`, то связанный с ним XML-файл должен иметь имя `MyApplication.xml`. Кроме того, этот сгенерированный XML-файл должен находиться в том же самом каталоге, что и скомпилированная сборка, чтобы система Visual Studio могла его найти.

Генерация документов с помощью надстройки GhostDoc

Несмотря на то что большинство разработчиков согласны с мнением о важности документации, они воспринимают это как трудоемкое и вынужденное занятие. Золо-

тое правило гласит: “Разработчик стремится делать только то, что проще сделать”. Это значит, что любое усовершенствование процесса разработки документации будет стимулировать все больше разработчиков использовать его в своей работе.



Существует более авторитарный подход к разработке документации, основанный на использовании инструментов анализа кода, таких как StyleCop, которые обеспечивают минимальный объем документации. Программа StyleCop содержит почти 50 встроенных правил, предназначенных для верификации содержания и форматирования XML-документации. Утилита StyleCop более подробно обсуждается в главе 13.

GhostDoc — это надстройка для системы Visual Studio, которая предоставляет пользователю возможность автоматически вставлять блоки XML-комментариев в классы или методы с помощью комбинаций клавиш. Однако истинная мощь надстройки GhostDoc заключается не в способности создавать основные заглушки, а в автоматизации значительной части самой документации.

С помощью ряда списков, указывающих, как следует интерпретировать разные части членов и имена переменных, надстройка GhostDoc генерирует простые фразы, с которых можно начать создание собственной документации. Например, рассмотрим список, показанный на рис. 12.3, где слова определены как точки переключения фраз “Of the”. Если имя переменной или члена класса содержит подстроку “color”, то надстройка GhostDoc попытается создать фразу, которую можно использовать в XML-документации.

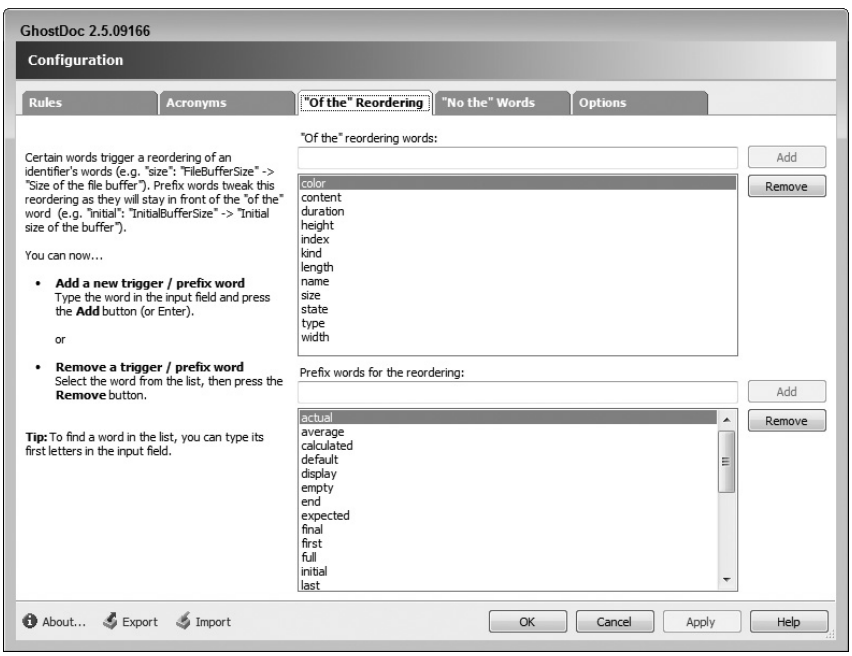


Рис. 12.3

Например, свойство `NewBackgroundColor` приведет к созданию законченной фразы `New color of the background`. Функциональные возможности надстройки GhostDoc позволяют также распознавать имена параметров и их предназначение.

На рис. 12.4 эта особенность надстройки продемонстрирована на примере обработчика события `Click`, заданного по умолчанию для управляющей кнопки. Параметры `sender` и `e` были распознаны как особые типы в контексте обработки события, и этот факт был соответствующим образом отражен в документации, сгенерированной надстройкой GhostDoc.

```
/// <summary>
/// Handles the Click event of the btnClose control.
/// </summary>
/// <param name="sender">The source of the event.</param>
/// <param name="e">The <see cref="System.EventArgs"/> instance containing the event data.</param>
private void btnClose_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Рис. 12.4

Надстройка GhostDoc представляет собой прекрасное средство для тех, кто считает разработку документации слишком трудоемким занятием. Ее можно загрузить с официального сайта `http://submain.com/ghostdoc`.

Генерация документации с помощью набора утилит Sandcastle

Sandcastle — это набор утилит, разработанных компанией Microsoft для компиляции документации. Эти утилиты позволяют легко создавать профессиональную документацию либо в виде HTML-файла справки (`.chm`), либо в формате Microsoft Help 2 (`.hsx`).

Информация о наборе утилит Sandcastle размещена в блоге `http://blogs.msdn.com/sandcastle/`, а также на сайте открытого проекта CodePlex компании Microsoft `http://sandcastle.codeplex.com/`. На этом сайте находятся документация, форум, а также ссылки на свежие пакеты инсталляции набора Sandcastle.

По умолчанию набор утилит инсталлируется в каталог `c:\Program Files\Sandcastle`. В ходе работы утилиты Sandcastle в этом каталоге создается большое количество рабочих файлов, а также окончательный файл. К сожалению, для записи всех файлов и папок в каталог `Program Files` требуются права администратора. При работе под управлением системы Windows Vista и включенной подсистеме UAC это бывает затруднительно. По этой причине мы рекомендуем инсталлировать этот набор утилит в такое место, куда пользователь имеет неограниченный доступ.

Утилиты Sandcastle запускаются только из командной строки. Впрочем, большое количество сторонних поставщиков предлагают графические пользовательские интерфейсы для утилит Sandcastle. Ссылки на них можно найти на сайте Wiki.

Для начала откроем командную строку Command Prompt с помощью команды Start Menu⇨All Programs⇨Microsoft Visual Studio 2010⇨Visual Studio Tools и изменим каталог на `<Sandcastle Install Directory> \Examples\sandcastle\`.



Командная строка Command Prompt в системе Visual Studio 2010 представляет собой эквивалент обычной командной строки, за исключением того, что она к тому же настраивает большое количество переменных окружения, таких как пути поиска каталогов, которые часто требуются средствами Visual Studio 2010.

В этом каталоге содержатся файл с примером класса `test.cs` и файл проекта MSBuild `build.proj`. Файл с примером класса содержит методы и свойства, которые сопровождаются стандартными дескрипторами XML-комментариев, рассмотренными выше, а также дополнительными дескрипторами XML-комментариев, присущими только утилитам Sandcastle. Скомпилируем этот файл и сгенерируем файл с XML-документацией, введя команду

```
csc /t:library test.cs /doc:example.xml
```

Как только эта команда будет выполнена, можно приступать к генерации файла справки. Для этого проще всего выполнить демонстрационный файл проекта MSBuild, который поставляется вместе с утилитами Sandcastle. Этот проект специально предназначен для генерации документации с помощью файлов `test.dll` и `example.xml`. Запустим проект MSBuild, введя команду

```
msbuild build.proj
```

Для создания файла документации в проекте MSBuild вызывается несколько утилит Sandcastle: MRefBuilder, BuildAssembler и XslTransform.



Для того чтобы не вызывать утилиты Sandcastle вручную каждый раз, когда создается окончательная версия приложения, лучше выполнять их запуск в качестве события, *генерируемого после сборки* (post-build event). Создание события сборки описывается в главе 6.

Процесс генерации документации занимает удивительно много времени. Частично это объясняется тем, что для получения информации о всех типах, свойствах и методах, существующих в сборке и зависимых от нее сборках, утилита MRefBuilder применяет механизм отражения. Кроме того, каждый раз, обнаруживая базовый тип платформы .NET Framework, она пытается найти в сети соответствующую документацию MSDN, чтобы вставить в справочную документацию корректные гиперссылки.



При первом запуске проект MSBuild генерирует данные отражения для всех классов, существующих в среде .NET Framework, поэтому его компиляция может затянуться еще больше.

По умолчанию проект MSBuild `build.proj` генерирует документацию в стиле vs2005 (рис. 12.4) и записывает ее в каталог `<Sandcastle Install Directory>\Examples\sandcastle\chm\`. Пользователь может выбрать и другой стиль, добавив в командную строку одну из опций, приведенных ниже.

```
/property:PresentationStyle=vs2005
/property:PresentationStyle=hana
/property:PresentationStyle=prototype
```

В следующем фрагменте кода показан раздел из демонстрационного файла классов `test.cs`, связанного со страницей справочной документации, показанной на рис. 12.5.

```
/// <summary>
/// Переставляет данные типа <typeparamref name="T"/>
/// </summary>
/// <param name="lhs"> левая часть <typeparamref name="T"/> </param>
/// <param name="rhs"> правая часть <typeparamref name="T"/> </param>
/// <typeparam name="T"> Тип переставляемых элементов </typeparam>
public void Swap <T> (ref T lhs, ref T rhs)
```

```
{
    T temp;
    temp = lhs;
    lhs = rhs;
    rhs = temp;
}
```

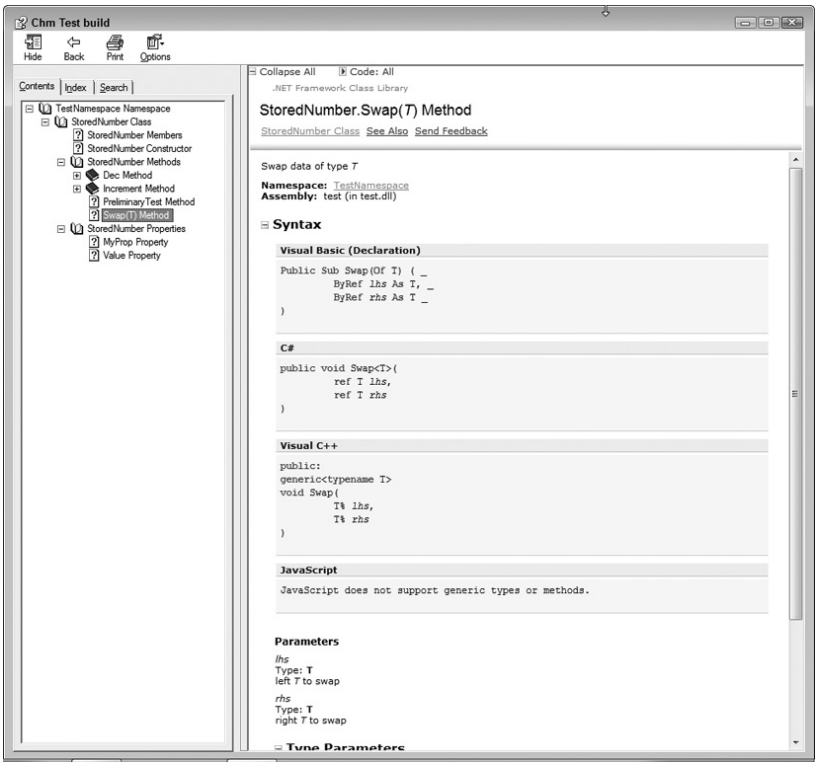


Рис. 12.5

По умолчанию целью проекта MSBuild `build.proj` является файл справки с расширением `.chm` для сборки `test.dll`. Изменить цель проекта можно с помощью опций командной строки.

```
/target:Clean - удаляет все сгенерированные файлы
/target:HxS - создает HxS-файл для системы Visual Studio в дополнение к CHM
```



Microsoft Help 2 (.HxS) — это формат справочного файла для системы Visual Studio. Для того чтобы генерировать .HxS-файлы, необходимо инсталлировать пакет Microsoft Help 2.x SDK, представляющий собой часть пакета Visual Studio 2010 SDK.

Комментарии Task List

Окно Task List — это свойство системы Visual Studio 2010, позволяющее отслеживать любые задачи, связанные с кодированием, а также незавершенные задания. Задачи можно вставить в список вручную в качестве задач категории User Tasks или ав-

томатически, используя встроенные комментарии. Для этого необходимо открыть окно Task List, выбрав команду View⇨Task List или нажав комбинацию клавиш <Ctrl+\>, <Ctrl+T>. На рис. 12.6 показано окно Task List, в которой определены задачи категории User Tasks.



Задачи категории User Tasks сохраняются в файле опций решения с расширением (.suo), содержащем пользовательские настройки. Открывать этот файл для чтения не рекомендуется, поэтому задачи User Tasks невозможно распределить между несколькими пользователями, работающими над одним и тем же решением.

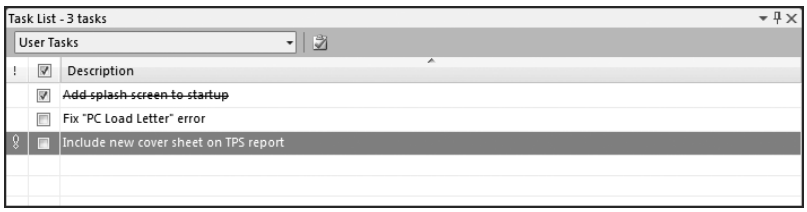


Рис. 12.6



В левом верхнем углу окна Task List есть фильтр, который переключает код из автоматического режима Comment Tasks в ручной режим User Tasks.

Если добавить в свой код комментарий, начинающийся *лексемой комментария* (comment token), то этот комментарий будет добавлен в окно Task List как Comment Task. По умолчанию лексемами комментария в системе Visual Studio 2010 являются `TODO`, `HACK`, `UNDONE` и `UnresolvedMergeConflict`.

В следующем коде продемонстрирована лексема `TODO`. На рис. 12.7 показано, как этот комментарий появляется в виде задачи в окне Task List. Для того чтобы непосредственно ввести строку комментария в свой код, достаточно дважды щелкнуть на соответствующем элементе в окне Task List.

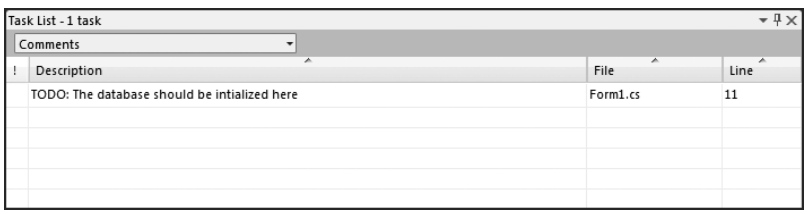


Рис. 12.7

C#

```
using System;
using System.Windows.Forms;

namespace CSWindowsFormsApp
{
    public partial class Form1 : Form
    {
        public Form1()
```

```
        {
            InitializeComponent();
            //TODO: Здесь должна инициализироваться база данных
        }
    }
}
```

Список лексем комментария можно откорректировать на странице опций, выбрав команду Tools⇨Options⇨Environment⇨Task List, как показано на рис. 12.8. Каждой лексеме можно присвоить приоритет — `Low`, `Normal` или `High`. Слово `TODO` является лексемой комментария по умолчанию. Ее нельзя переименовать или удалить. Тем не менее можно изменить ее приоритет.

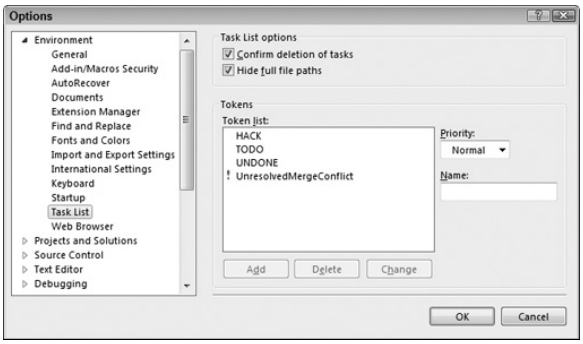


Рис. 12.8

Кроме элементов User Tasks и Comments, с помощью окна Task List в код можно добавить комбинации клавиш. Для создания комбинации клавиш Task List Shortcut следует разместить курсор в соответствующем месте окна редактора кода и выбрать команду Edit⇨Bookmarks⇨Add Task List Shortcut. В результате в левом поле окна редактора кода появится пиктограмма в виде стрелки (рис. 12.9).

Если теперь перейти к окну Task List, то можно увидеть, что в раскрывающемся списке появилась новая категория Shortcuts (рис. 12.10). По умолчанию описание комбинации клавиш содержит строку кода; однако пользователь может отредактировать ее и ввести любой текст. Дважды щелкнув на элементе, пользователь может вернуться к окну редактора кода.

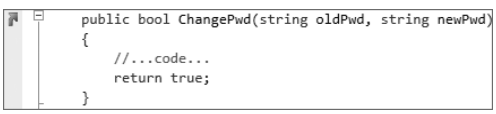


Рис. 12.9



Рис. 12.10

Как и задачи категории User Tasks, комбинации клавиш Shortcuts хранятся в файле с расширением `.suo` и, как правило, не предназначены для чтения с помощью обычных редакторов и разделения между несколькими пользователями. Следовательно, они превосходно подходят для закрытого аннотирования кода с помощью секретных замечаний и пометок.

Резюме

XML-комментарии — это не только чрезвычайно мощный, но и очень удобный для реализации механизм. Он позволяет эффективно использовать возможности системы IntelliSense, включая разработку пользовательских подсказок и кратких справок. Процесс создания XML-комментариев можно автоматизировать с помощью надстройки GhostDoc. Используя набор утилит Sandcastle, пользователь может генерировать отдельную профессиональную документацию для каждого члена и класса, входящего в решение. В заключение отметим, что комментарии категории Task List очень полезны для отслеживания отложенных задач, связанных с кодированием и другими незавершенными действиями.

13

Инструменты, обеспечивающие согласованность кода

В ЭТОЙ ГЛАВЕ...

- Управление исходным кодом Visual Studio 2010
- Создание, добавление и обновление кода в хранилище кодов
- Определение и ввод в действие стандартов кодирования
- Добавление контрактов

При разработке небольшого приложения не составляет труда понять, как связаны друг с другом его составные части, и при необходимости внести изменения. К сожалению, даже в таких проектах хорошо структурированный и организованный код часто превращается в мешанину переменных, методов и классов. Если же приложение большое, сложное и над ним работают много разработчиков, то эта проблема возрастает.

В этой главе описывается, как использовать возможности системы Visual Studio 2010 для создания и сопровождения согласованного кода. Первая часть главы посвящена использованию системы управления исходным кодом, позволяющей отслеживать изменения, происходящие в программе с течением времени. Использование системы управления исходным кодом облегчает распределение кода и отслеживание изменения при коллективной работе над проектом, но, что еще важнее, она позволяет запоминать внесенные изменения.

В оставшейся части главы мы изучим утилиты FxCop и StyleCop, с помощью которых можно устанавливать и проводить в жизнь стандарты кодирования. Соблюдение стандартов и принципов облегчает понимание написанного кода, уменьшает количе-

ство его версий и сокращает время, затрачиваемое на его разработку. Кроме того, мы покажем, как повысить качество кода с помощью контрактов Code Contracts.

Управление исходным кодом

Существует множество разных методов разработки программного обеспечения. Несмотря на то что принципы организации коллективной работы, распределения заданий, проектирования и тестирования часто отличаются друг от друга, все они сходятся в том, что должно существовать хранилище всего исходного кода для приложения.

Управление исходным кодом — это процесс, связанный с записью исходного кода в хранилище (checking code in) и дальнейшим его извлечением (checking code out) для редактирования. Когда мы говорим об исходном коде, мы имеем в виду любые ресурсы, фалы конфигурации и даже документацию, которая требуется для создания и развертывания приложения.

Хранилища исходного кода различаются по структуре и интерфейсу. Как правило, хранилища обеспечивают ограниченный интерфейс, позволяющий записывать файлы в хранилище и извлекать их из него. Механизм хранения может быть простым и сводиться, например, к совместному использованию файлов, и не предусматривать хранение истории изменений. Несмотря на то что такое хранилище обеспечивает доступ всем разработчикам, работающим над проектом, к одному и тому же файлу, риск перезаписи или потери изменений в этом случае сводится к нулю. Более сложные хранилища не только обеспечивают богатый интерфейс для записи кода в хранилище и извлечения его из него, но и помогают объединять файлы и разрешать конфликты. Их также можно использовать в системе Visual Studio для управления исходным кодом. Хранилище исходного кода позволяет также отслеживать версии файлов, осуществлять ветвление и организовывать удаленный доступ.

Большинство организаций уже используют хранилище исходного кода как механизм для его совместного использования участниками проектов. Вместо копирования кода в общую папку и обращения за ним туда же участники проектов могут послать запрос в хранилище и получить самую последнюю версию исходного кода. Когда разработчик завершает свою работу, все изменения могут подвергнуться проверке непосредственно в хранилище. Это гарантирует, что любой член коллектива может получить доступ к самой последней версии кода. Кроме того, проверка исходного когда в хранилище облегчает выполнение регулярного резервного копирования.

Одним из важнейших преимуществ хранилища исходного кода является отслеживание версий, включая хранение полной истории того, где внесены изменения и кто их внес. Несмотря на то что каждый разработчик считает свой код идеальным, в реальности довольно часто изменение в одной части кода может привести к нарушению работы приложения в целом. Возможность просматривать историю изменений, внесенных в проекте, позволяет идентифицировать причину сбоя. Отслеживание изменений, внесенных в проект, также используется для создания отчетов и обзоров, поскольку каждое изменение имеет метку времени и сопровождается информацией об авторе.

Выбор хранилища исходного кода

Хранилище исходного кода не поставляется вместе с системой Visual Studio 2010, но она имеет много средств для записи файлов в хранилище и извлечения их из него, а также для объединения файлов и учета изменений. Для того чтобы использовать

хранилище исходного кода в системе Visual Studio 2010, необходимо указать его тип. Система Visual Studio 2010 обеспечивает глубокую интеграцию с механизмом управления исходными кодами и отслеживания проектов Team Foundation Server (TFS) — основным продуктом компании Microsoft в этой области. Кроме того, система Visual Studio поддерживает любой клиентский механизм, использующий интерфейс Source Code Control (SCC). К продуктам, использующим интерфейс SCC, относятся утилита Visual SourceSafe компании Microsoft, а также свободно распространяемые и открытые хранилища исходного кода Subversion и CVS.



У читателей может возникнуть неверное мнение о том, что утилита Microsoft Visual SourceSafe компании Microsoft больше не доступна, поскольку в прессе упоминается только TFS. Тем не менее утилита Microsoft Visual SourceSafe 2005 по-прежнему доступна и совместима с системой Visual Studio 2010. Однако существует новый лицензированный инструмент для системы TFS, который был специально разработан для небольших коллективов разработчиков взамен утилиты SourceSafe.

Для того чтобы было легче перемещаться по окнам системы Visual Studio 2010, любая недоступная функциональная возможность в меню скрывается. По умолчанию система Visual Studio 2010 не отображает на экране команду для вызова системы управления исходным кодом. Для того чтобы она появилась, пользователь должен задать настройки системы управления исходным кодом, выполнив команду Options из меню Tools. Окно Options, в котором выбрана команда Source Control, показано на рис. 13.1.

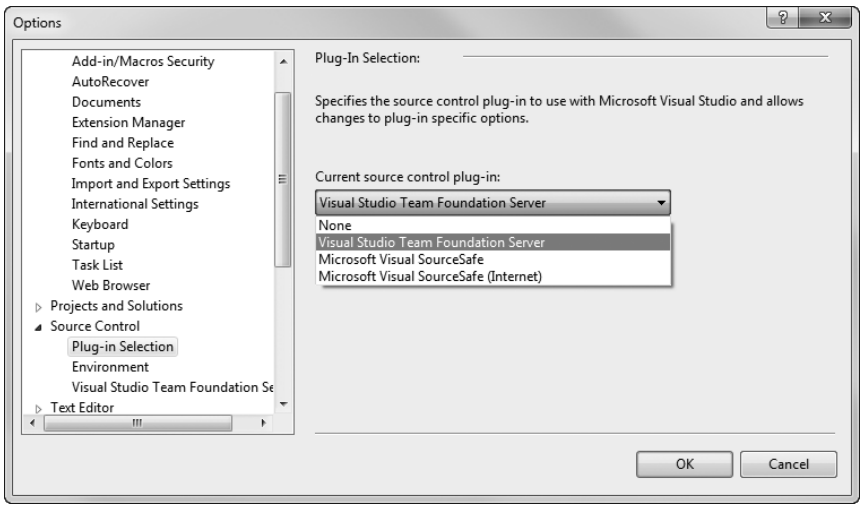


Рис. 13.1

Вначале появляется лишь несколько настроек механизма управления исходным кодом. Но, как только он выбран, на дереве появляются новые узлы, регламентирующие поведение этого механизма. Эти команды относятся именно к тому виду механизма управления исходным кодом, который был выбран пользователем.

В оставшейся части главы мы сосредоточимся на использовании программного продукта Visual SourceSafe в сочетании с системой Visual Studio 2010. В главе 57 мы

рассмотрим использование программного продукта Team Foundation, обеспечивающего более полную интеграцию с системой Visual Studio 2010 и более широкий выбор функциональных возможностей по сравнению с хранилищем исходного кода.



Интернет-ориентированная версия приложения Visual SourceSafe использует клиент-серверную модель, которая реализуется с помощью протокола HTTP или HTTPS, а не с помощью доступа к хранилищу исходного кода посредством совместного использования файлов. При этом требуется отдельная установка серверной части приложения.

Как только хранилище исходного кода будет выбрано в меню, необходимо задать его настройки для конкретного компьютера. В частности, для системы Visual SourceSafe необходимо задать путь к хранилищу пользователя, с которым устанавливается связь, а также настройки для записи файлов в хранилище и извлечения их из него.

Настройки окружения

Большинство хранилищ исходного кода имеет ряд настроек, которые необходимо задать, чтобы система Visual Studio 2010 могла установить связь с хранилищем и обеспечить доступ к информацию, которая в нем хранится. Эти настройки обычно являются уникальными по отношению к конкретному хранилищу, хотя некоторые могут применяться и к большинству других хранилищ.

На рис. 13.2 показана вкладка Environment, содержащая флажки и переключатели, управляющие записью файлов в хранилище и извлечением их из него. Эти настройки относятся к большинству хранилищ. Выпадающее меню в верхней части окна позволяет задать ряд профилей, отражающих предположения о разных категориях разработчиков.

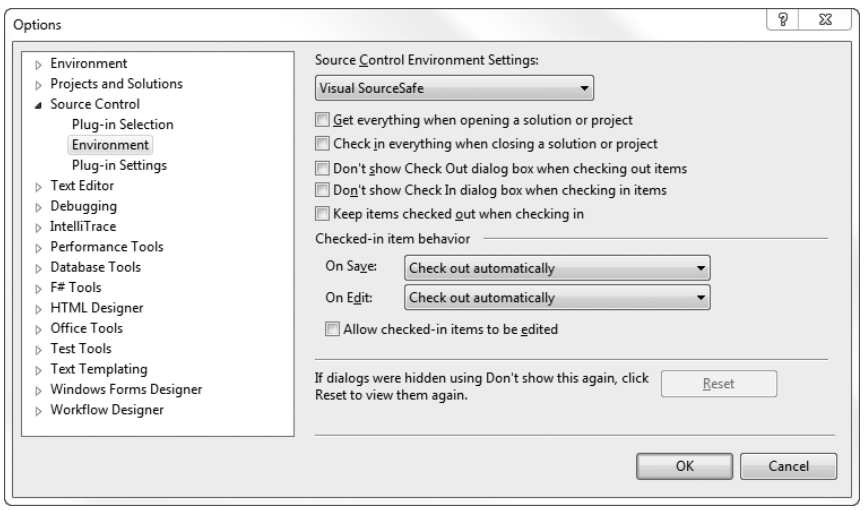


Рис. 13.2

Встроенные настройки

Многие хранилища исходного кода требуют дополнительных настроек в системе Visual Studio 2010, чтобы обеспечить связь с файлами. Эти настройки задаются в окне Plug-in Settings, содержание которого зависит от типа хранилища. Некоторые храни-

лища, такие как SourceSafe, не требуют специальной информации, касающейся местонахождения хранилища, пока решение не будет добавлено в механизм управления исходным кодом. В этот момент система SourceSafe запрашивает местонахождение существующего хранилища или позволяет пользователю создать новое хранилище.

Accessing Source Control

В настоящем разделе мы рассмотрим процесс добавления решения в новое хранилище Visual SourceSafe 2010, хотя те же самые принципы относятся и к любым другим хранилищам. Этот процесс можно применять как к новому, так и к существующему решению, которое еще не находится под контролем системы управления исходным кодом. Кроме того, мы будем предполагать, что система Visual SourceSafe не только уже инсталлирована, но и была выбрана как хранилище исходных кодов в системе Visual Studio 2010.

Создание хранилища

В качестве первого шага при установлении контроля над исходными кодами следует создать хранилище, в котором будут храниться данные. В одном и том же хранилище может храниться сколько угодно решений, хотя при этом намного труднее отделить информацию, касающуюся разных проектов. Более того, если хранилище будет повреждено, это может отразиться на решениях, хранящихся в нем.

Для того чтобы начать добавление решения в систему управления исходным кодом, перейдите в меню File и выберите команду Source Control⇨Add Solution to Source Control, как показано на рис. 13.3. В качестве альтернативы при создании нового решения можно установить флажок Add to Source Control в диалоговом окне New Project.

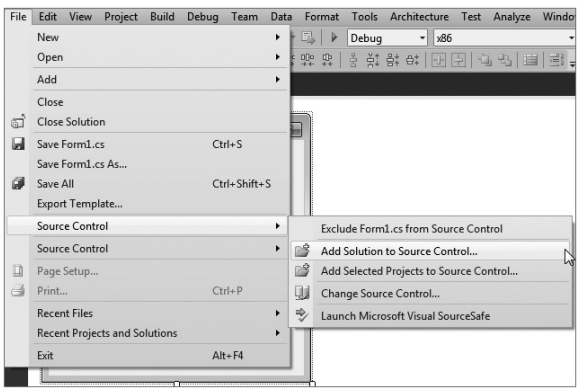


Рис. 13.3

При первом обращении к системе SourceSafe она открывает диалоговое окно, в котором перечислены доступные, пока пустые базы данных. Щелкнув на кнопке Add, можно запустить мастер Add SourceSafe Database Wizard, который предоставляет пользователю возможность сослаться на существующую базу данных, расположенную, возможно, на сервере или в другом месте жесткого диска или создать новую базу данных.

Для того чтобы создать новую базу данных SourceSafe, необходимо указать ее местоположение и имя. Кроме того, необходимо указать тип блокировки, которая будет использоваться при записи файлов в хранилище и извлечении их из него. Выбор мо-

дели Lock-Modify-Unlock позволяет только одному разработчику проверять файл в любое время. Это предотвращает ситуации, в которых два человека одновременно вносят изменения в один и тот же файл и значительно упрощает процесс записи файлов в хранилище. Однако эта модель часто приводит пользователей в отчаяние, если им необходимо использовать один и тот же ресурс. Наиболее типичным примером такого ресурса являются файлы проектов, доступ к которым может понадобиться нескольким разработчикам одновременно. Для того чтобы добавить файл к проекту или удалить его из него, следует извлечь проект из хранилища. Если разработчики не будут аккуратно возвращать файлы проектов назад после добавления нового файла, то работа коллектива будет значительно замедлена.

Альтернативная модель Copy-Modify-Merge позволяет нескольким разработчикам проверять один и тот же файл. Разумеется, если они готовы вернуть проверенный файл назад, должен существовать процесс согласования, позволяющий избежать перезаписи изменений, внесенных другим разработчиком. Объединение изменений может оказаться сложным процессом, приводящим к потере изменений и окончательному коду, который невозможно ни скомпилировать, ни выполнить. Эта модель обеспечивает преимущества параллельного доступа к файлам, но связана с накладными расходами в процессе записи файлов в хранилище.

Добавление решения

Как только будет создано хранилище SourceSafe, откроется диалоговое окно Add to SourceSafe, предлагающее пользователю задать местоположение приложения и присвоить ему имя в хранилище. Система SourceSafe очень похожа на систему распределенного использования файлов — она создает папки под корнем (`$/`), в которые помещает файлы, подвергающиеся контролю.

Окно Solution Explorer

Первое отличие, которое обнаружит пользователь, добавив решение в механизм управления исходным кодом, заключается в том, что система Visual Studio 2010 упорядочивает пиктограммы в окне Solution Explorer, чтобы показать статус файла в этом механизме. На рис. 13.4 показаны три статуса файлов. Когда решение изначально добавляется в хранилище механизма управления исходным кодом, рядом с пиктограммами всех файлов появятся пиктограммы в виде маленького висячего замка. Это означает, что данные файлы были отправлены в хранилище и в текущий момент никем не были извлечены оттуда. Например, эту пиктограмму имеют файлы `Solution` и `Properties`.

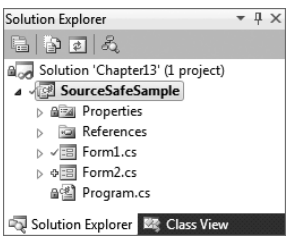


Рис. 13.4

Как только решение поставлено под контроль механизма управления исходным кодом, все изменения записываются, включая добавление и удаление файлов. На рис. 13.4 продемонстрирован процесс добавления файла `Form2.cs` в решение. Знак "плюс", расположенный рядом с именем `Form2.cs`, означает, что этот файл является новым. Галочки рядом с именами проекта `SourceSafeSample` и файла `Form1.cs` означают, что эти файлы в настоящее время извлечены из хранилища. Если два человека одновременно извлекли из хранилища один и тот же файл, то возле соответствующей пиктограммы появляются две галочки.

Запись файлов в хранилище и извлечение их оттуда

Файлы можно отправлять в хранилище и извлекать оттуда с помощью контекстного меню, которое открывается после щелчка правой кнопкой мыши на соответствующем элементе окна Solution Explorer. Если решение находится под контролем механизма управления исходным кодом, то в это меню включаются команды, показанные на рис. 13.5, *слева*.

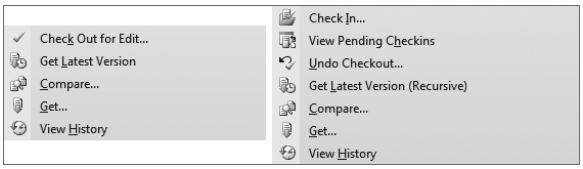


Рис. 13.5

Перед редактированием файла его необходимо извлечь из хранилища с помощью команды Check Out for Edit. Как только файл извлечен, в контекстном меню появляются дополнительные команды, включая Check In, View Pending Checkins, Undo Checkout и др. (см. рис. 13.5).

Откладывание изменений

В большом приложении часто трудно сразу определить, какие файлы были извлечены из хранилища для редактирования, были недавно добавлены или удалены из проекта. Окно Pending Checkins, доступ к которому можно получить из контекстного меню, которое открывается после щелчка правой кнопкой мыши в окне Solution Explorer,
а также из меню View, показано на рис. 13.6. С его помощью очень удобно проверять, какие файлы ожидают записи в хранилище. Оно также содержит поля для ввода комментария. Этот комментарий присоединяется к файлу, когда тот записывается в хранилище, чтобы впоследствии можно было узнать причину внесенных изменений.



Несмотря на то что это больше не требуется для системы SourceSafe, многие коллективы разработчиков по-прежнему согласовывают структуру папок в системе SourceSafe со структурой каталогов в своем компьютере. Это считается целесообразным, потому что стимулирует разработчиков использовать хорошие структуры каталогов и папок.

Интерфейс Source Code Control (SCC) предполагает, что файл решения с расширением `.sln` расположен в той же папке, что и файл проекта, или в ее дочерней папке. Если разместить файл с расширением `.sln` в иерархии папок, которая отличается от иерархии файлов проекта, то при использовании механизма управления кодом могут возникнуть разные “интересные” осложнения.

Для того чтобы вернуть файл обратно в хранилище, следует проверить, есть ли соответствующая отметка в списке файлов, добавить подходящий комментарий в соответствующее поле и щелкнуть на кнопке Check In. В зависимости от установленных настроек система может запросить у пользователя подтверждение, прежде чем записать файл в хранилище. Например, если выбрать вариант Keep All Checked Out, как показано на рис. 13.6, то файлы, которые должны быть записаны в хранилище, будут

считаться извлеченными даже после успешного завершения их записи в хранилище. Это полезно в ситуациях, когда пользователь еще не закончил вносить изменения и хочет сделать текущие изменения доступными другим разработчикам.

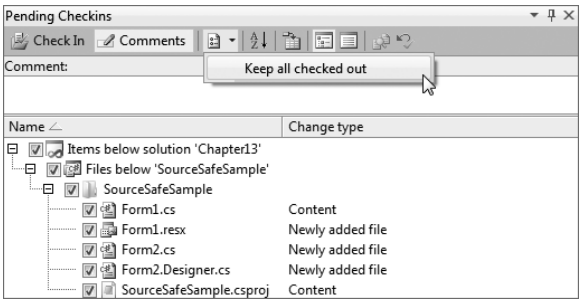


Рис. 13.6

Многие разработчики предпочитают, чтобы система Visual Studio автоматически извлекала файл из хранилища при редактировании. Это позволяет избежать лишнего этапа — извлечения файла перед редактированием. Однако это может привести к преждевременному извлечению файлов: например, если разработчик случайно внесет изменения в неправильный файл. Кроме того, разработчик может решить, что внесенные изменения были не нужны, и вернуться к варианту, записанному в хранилище. На этот случай на инструментальной панели Toolbar в окне Pending Checkins есть кнопка Undo Checkout. Эта команда извлекает из хранилища текущую версию файла и отменяет локальные изменения, внесенные разработчиком. Она также доступна из контекстного меню.

Перед отправкой файла в хранилище целесообразно сделать обзор внесенных изменений. Некоторые организации приняли правила, в соответствии с которыми все изменения перед отправкой файла в хранилище должны быть проанализированы. Выбрав команду Compare Versions, пользователь может запустить интерфейс, который выделяет различия между двумя версиями. На рис. 13.7 показано, что в файл `Form1.vb` был добавлен обработчик события `Form Load`. И хотя на рис. 13.7 этого не видно, разные типы изменений также выделяются разными цветами; добавления (такие как метод `Say_HelloButton_Click`) выделяются зеленым цветом, а удаленные и измененные строки — красным и синим соответственно.

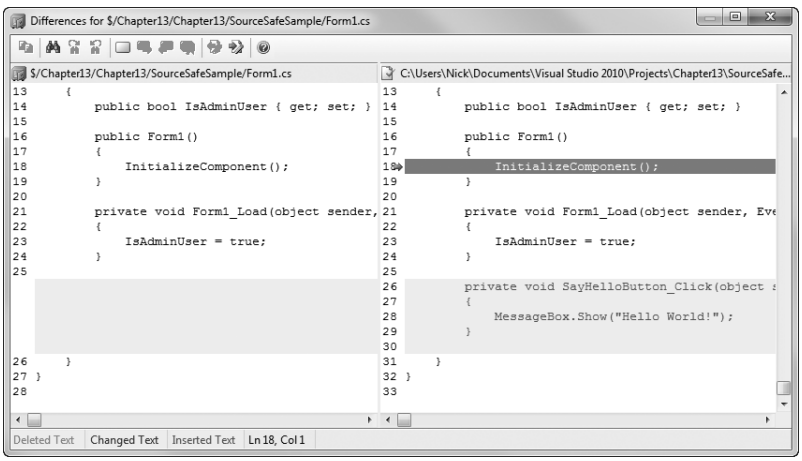


Рис. 13.7

Поскольку исходные файлы часто бывают довольно большими, это окно предусматривает несколько средств для навигации по тексту. Для поиска конкретной строки можно использовать команду Find. Для перемещения вперед и назад в тексте файла можно расставлять закладки. Наиболее полезными являются кнопки Next Difference и Previous Difference, позволяющие разработчику анализировать различия между версиями, не прибегая к прокрутке файла.

Объединение изменений

Иногда необходимо, чтобы один и тот же файл модифицировали несколько разработчиков. В некоторых случаях, когда эти изменения не связаны между собой, их можно учитывать автоматически, например добавление метода в существующий класс. Если же изменения вносятся в один и тот же раздел файла, необходим процесс их согласования, чтобы сохранить корректность кода.

На рис. 13.8 показано диалоговое окно Merge, которое открывается перед разработчиком, пытающимся записать в хранилище файл, измененный другим разработчиком. В верхней части этого окна показаны две версии файла, создающие конфликт. В каждой из областей окна показано, чем эти файлы отличаются от оригинала, который был извлечен из хранилища. В левой части показано то, что сейчас записано в хранилище, а в правой — изменения, внесенные разработчиком. В данном случае обе версии имеют вставку, которая открывает окно сообщения, а разработчик должен решить, какое из сообщений является правильным.

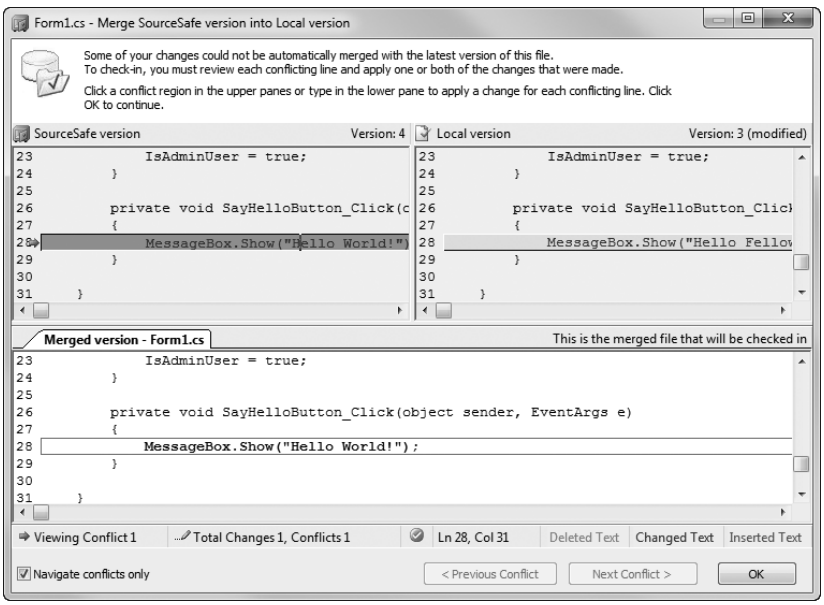


Рис. 13.8

В отличие от диалогового окна Compare Versions окно Merge предназначено для облегчения взаимодействия с пользователем. В верхней части окна пользователь может принять или отклонить изменения, внесенные в файле, просто щелкнув на них. Изменение цвета в нижней части окна свидетельствует о том, что изменения приняты и фрагмент кода вставлен в правильное место кода. Кроме того, в нижней части

окна разработчик может вводить код, хотя при этом не применяется ни технология автоматического дополнения IntelliSense, ни проверка ошибок.

После разрешения конфликта пользователь может щелкнуть на кнопке OK и сохранить изменения в своем локальном файле. Объединенные версии также могут быть отправлены в хранилище.

История

Каждый раз, когда файл отправляется в хранилище SourceSafe и извлекается из него, в историю версии вносится соответствующая запись. Для просмотра этой истории используется команда View History из контекстного меню окна Solution Explorer. На рис. 13.9 показана краткая история файла, который четырежды отправлялся в хранилище. Данное диалоговое окно позволяет разработчикам просматривать предыдущие версии, искать детали (например, комментарии), получать конкретную версию (перезаписывая текущий файл) и извлекать файл из хранилища. Кроме того, пользователь может сравнивать разные версии файла, фиксировать конкретную версию, выполнять откат к предыдущей версии (стирая при этом более свежие версии), а также создавать отчет об истории файла. Выбрав команду View History, пользователь может выполнять фильтрацию списка по диапазону дат и/или по именам пользователей.

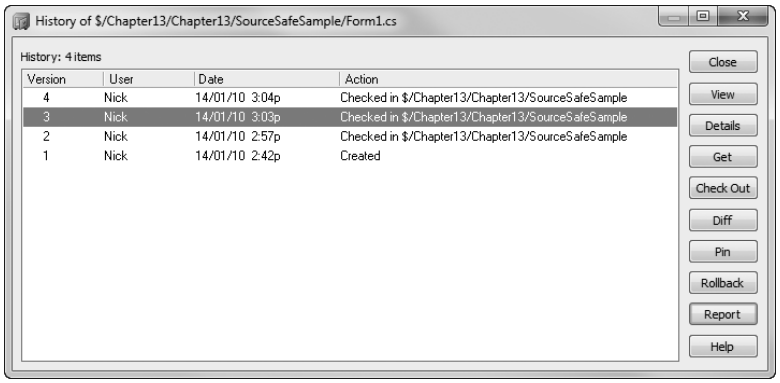


Рис. 13.9

Фиксация версии

Окно History (см. рис. 13.9) можно использовать для фиксации версии файла. Фиксация некоей версии файла делает ее текущей. Когда пользователь извлечет текущий исходный файл из хранилища, он получит зафиксированную версию. Фиксация версии файла также предотвращает ее несанкционированное извлечение из хранилища. Это может оказаться полезным, если внесенные изменения были неполными или вызывают ошибки при работе приложения. В этом случае можно зафиксировать предыдущую версию файла, чтобы другие разработчики могли продолжать работу, пока проблема не будет решена.

Автономная поддержка механизма управления исходным кодом

Если хранилище исходного кода недоступно, то система Visual Studio 2010 самостоятельно обеспечивает встроенную автономную поддержку механизма Visual SourceSafe. Перебой в работе может возникнуть по многим причинам: если сервер вышел из строя,

произошел сбой работы сети либо пользователь решил поработать на изолированном компьютере.

Если открыть решение, извлеченное из хранилища системой Visual SourceSafe, то после обрыва связи с хранилищем система Visual Studio предложит пользователю продолжить работу или выбрать другое хранилище. Помимо этого, система может спросить у пользователя, хочет ли он попытаться установить соединение с помощью протокола HTTP. Если на оба вопроса пользователь ответит отрицательно, то ему будут предложены четыре варианта решения проблемы (рис. 13.10).

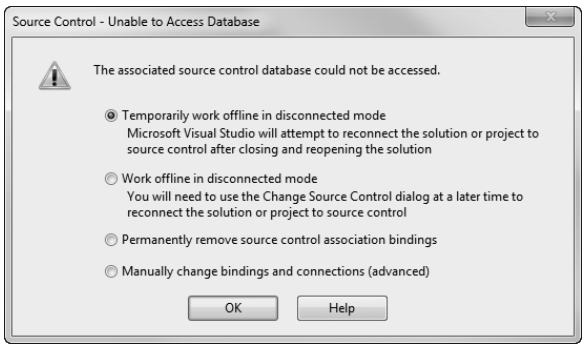


Рис. 13.10

Если сбой носит временный характер, то следует выбрать первый вариант: Temporarily Work Offline in Disconnected Mode. Это позволит загружать файлы из хранилища и продолжить редактирование исходного кода в автономном режиме.



Когда пользователь в первый раз попытается загрузить файлы из хранилища в автономном режиме, откроется очень большое диалоговое окно, содержащее короткое описание. Главная суть этого описания заключается в том, что система Visual Studio фактически имитирует загрузку и пользователь должен вручную объединять изменения при отправке кода обратно в хранилище.

Когда пользователь откроет решение в следующий раз и хранилище будет доступно, система Visual Studio автоматически проверит любые "имитированные" отправления, которые происходили в автономном режиме.

Многие операции, связанные с управлением кодом, в режиме простоя не доступны. К ним относятся операции, которые обычно зависят от непосредственного доступа к серверу, например Check In, Merge Changes, View History и Compare Versions.

Стандарты кодирования

По мере увеличения масштабов проектирования программного обеспечения и коллективов разработчиков возникла тенденция смешения стилей, стандартов и способов кодирования. Это может чрезвычайно усложнить эксплуатацию и сопровождение программ, что часто приводит к неправильному поведению систем из-за большого количества ошибок и проблем. К счастью, в системе Visual Studio 2010 существуют превосходные встроенные инструменты, устанавливаемые в виде надстроек, которые позволяют

установить единообразный стиль именования сущностей, упорядочить методы и гарантировать наличие приемлемых комментариев. В этом разделе мы рассмотрим некоторые инструменты, которые можно использовать для улучшения согласованности кода, написанного пользователем и другими членами команды.

Анализ кода с помощью надстройки FxCop

После нескольких выпусков платформы .NET Framework и системы Visual Studio компания Microsoft решила установить набор стандартов кодирования, которые может выбрать коллектив разработчиков, для того чтобы следовать им в своей работе. Эти стандарты кодирования подробно описаны в разделе Code Analysis for Managed Code Warnings на сайте MSDN (`http://msdn.microsoft.com`) и устанавливаются с помощью надстройки FxCop, которую можно загрузить с сайта компании Microsoft.

Загрузив и инсталлировав надстройку FxCop, пользователь должен запускать ее как отдельный инструмент из меню Start. Для того чтобы выполнять надстройку FxCop в рамках процесса сборки, ее можно вызвать из командной строки с помощью файла `FxCopCmd.exe`, находящегося в папке инсталляции. После запуска надстройка `FxCop` автоматически создает и открывает новый проект. Сначала необходимо сохранить этот проект в одной папке с решением. Затем следует выбрать команду Add Targets из меню Project и выбрать сборки (динамически подгружаемые библиотеки и выполняемые файлы), из которых должно состоять приложение. Для того чтобы начать анализ кода, необходимо щелкнуть на кнопке Analyze; результат должен быть похож на приведенный на рис. 13.11



Выпуск Visual Studio 2010 Premium и более поздние версии включают в себя инструмент Managed Code Analysis, который, по существу, является версией надстройки FxCop, интегрированной в среду разработки. Она обсуждается в главе 55.

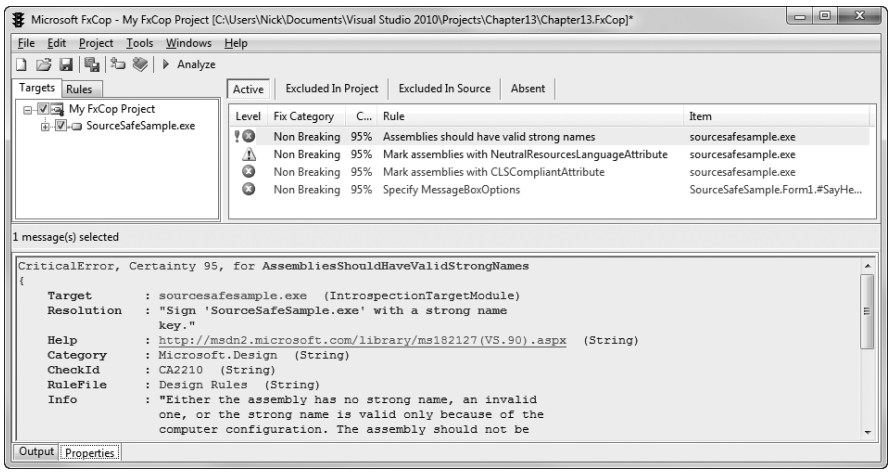


Рис. 13.11

На рис. 13.11 показаны сообщения о трех ошибках (одна из которых помечена как критическая) и одно предупреждение. Несмотря на то что предупреждения можно игнорировать, они часто свидетельствуют о существовании проблемы, связанной с не-

удачной организацией кода или с безопасностью, поэтому целесообразно попытаться по возможности минимизировать или вообще свести к нулю количество предупреждений и сообщений об ошибках. В этом примере исправить первую ошибку несложно: достаточно закодировать подпись в приложении, и ошибка исчезнет. Тем не менее пометить сборку атрибутом `CLSCompliant`, как требует сообщение о второй ошибке, может оказаться невозможно. Для того чтобы сообщение об этой ошибке не появлялось каждый раз при активизации списка ошибок, щелкните правой кнопкой мыши на ней и выберите команду Exclude. Система предложит пользователю добавить комментарий, чтобы он обосновал исключение этой ошибки из списка. После щелчка на кнопке OK исключенная ошибка появится в закладке Excluded In Project tab (см. фон на рис. 13.12). В ответ на двойной щелчок на этой ошибке система откроет окно, содержащее подробное описание ошибки и комментарии пользователя в разделе Notes.

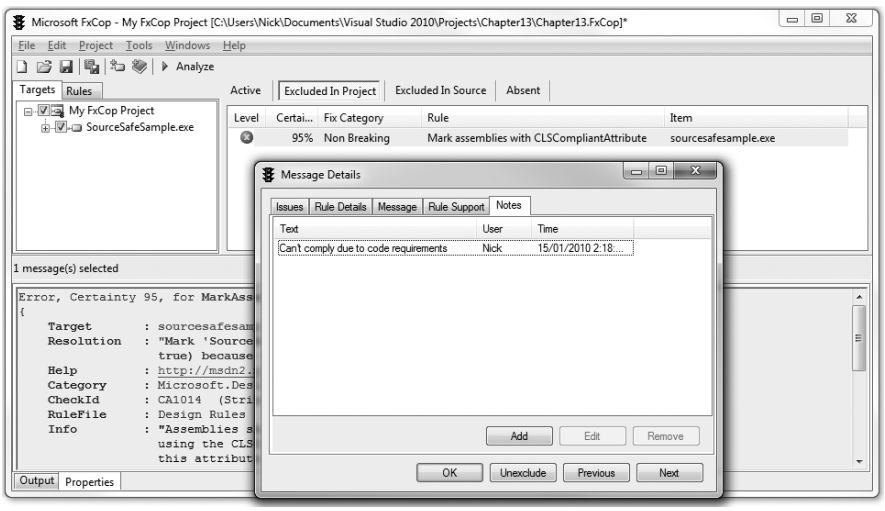


Рис. 13.12

Третья ошибка, показанная на рис. 13.11, свидетельствует о том, что пользователь не указал параметр `MessageBoxOptions`. В данном случае это сделано умышленно, чтобы показать процедуру исключения ошибки, содержащейся в коде. Как показано в следующем фрагменте кода, для этого надо добавить атрибут `SuppressMessage` в метод, вызывающий метод `MessageBox.Show`. При этом передаются параметры `Category`, `CheckId` и `Name`, характеризующие ошибку, найденную в окне Message Details.

C#

```
[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Globalization",
                                                 "CA1300:SpecifyMessageBoxOptions",
                  Justification= "MessageBoxOptions omitted intentionally")]
private void SayHelloButton_Click(object sender, EventArgs e){
    MessageBox.Show("Hello World!" );
}
```

VB

```
<System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Globalization",
                                                 "CA1300:SpecifyMessageBoxOptions",
                Justification:= "MessageBoxOptions omitted intentionally")>
Private Sub SayHelloButton_Click(ByVal sender As System.Object, _
```

```
                    ByVal e As System.EventArgs) Handles SayHelloButton.Click
    MessageBox.Show("Hello World!")
End Sub
```

Для того чтобы надстройка FxCop заметила атрибут `SuppressMessage`, необходимо установить флажок `CODE_ANALYSIS`. Это можно сделать, добавив ключевое слово `CODE_ANALYSIS` в поле ввода Custom Constants диалогового окна Advanced Compile Options (в закладке Compile, находящейся на странице свойств проекта) при работе на языке VB или добавив то же самое ключевое слово в поле ввода Conditional compilation symbols (в закладке Build, расположенной на станице свойств проекта) при работе на языке C#. После сохранения, повторной сборки и повторного выполнения команды Analysis (обратите внимание на то, что повторно выполнять и даже повторно перезагружать проект в надстройку FxCop не обязательно) пользователь увидит, что ошибка была перемещена в закладку Excluded in Source. И снова двойной щелчок на ошибке и переход к закладке Notes позволяет раскрыть содержание параметра `Justification`, представляющего собой часть атрибута `SuppressMessage` (для использования этого атрибута, возможно, понадобится импортировать пространство имен `System.Diagnostic.CodeAnalysis`).

Существуют еще два способа управления применением надстройки FxCop к пользовательскому коду. Первый заключается в использовании окна Targets, позволяющего включать и исключать правила, применяемые к разделам кода. На рис. 13.13, *слева*, показано окно Targets, в котором раскрыты узлы SourceSafeSample и IsAdminUser. В этом примере флажки были сброшены, чтобы отменить применение правил к данному свойству.

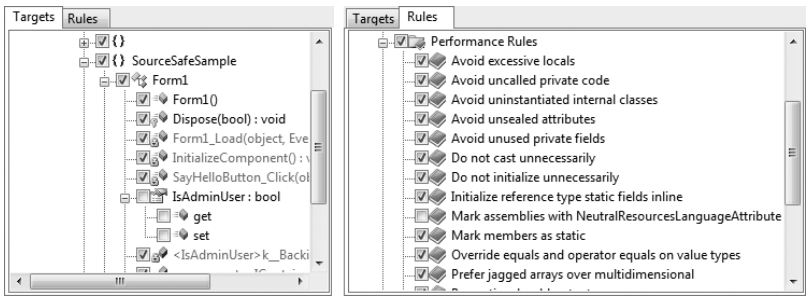


Рис. 13.13

На рис. 13.13, *справа*, приведен список правил Rules. Он раскрыт, чтобы продемонстрировать помеченные сборки с правилом `NeutralResourcesLanguageAttribute`. Именно это правило генерировало предупреждение, показанное на рис. 13.11. Сбросив флажок, мы сняли метку со сборки, чтобы в ходе анализа кода это правило к ней не применялось.



Исключение всего правила в принципе считается плохим тоном, поскольку при этом могут остаться незамеченными ошибки, сделанные позднее. Например, если впоследствии в проект была добавлена сборка, это правило никогда не будет к ней применяться, даже если это очень важно.

Надстройка FxCop сопровождается большим количеством правил, которые можно применять, а можно и не применять при коллективной разработке кода. Если необходимо установить свой стандарт, то пользователь может расширить набор правил, заданных по умолчанию, написав собственные, используя пакет FxCop SDK, который прилагается к надстройке FxCop как справочное руководство.

Проверка стиля с помощью надстройки StyleCop

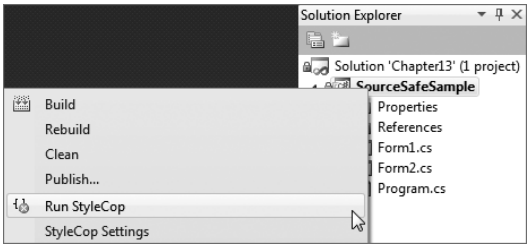


Рис. 13.14

Несмотря на то что надстройка FxCop превосходно решает проблемы, связанные со способом, которым пользователь пишет код, она не позволяет в полной мере поддерживать общий стиль кодирования. Для решения этих задач необходимо загрузить и инсталлировать надстройку StyleCop, доступную в галерее кода на сайте MSDN (`http://code.msdn.microsoft.com`). В отличие от надстройки FxCop, которая выполняется как отдельный инструмент, надстройка StyleCop интегрируется в среду разработки Visual Studio 2010, позволяя выполнять анализ непосредственно в окне Solution Explorer, как показано на рис. 13.14.

После запуска надстройки StyleCop пользователь увидит, что сообщения о любых проблемах появляются в виде предупреждений, как показано на рис. 13.15. Для того чтобы повысить вес сообщений, выдаваемых надстройкой StyleCop, необходимо сообщить системе Visual Studio 2010, чтобы она воспринимала предупреждения как сообщения об ошибках.

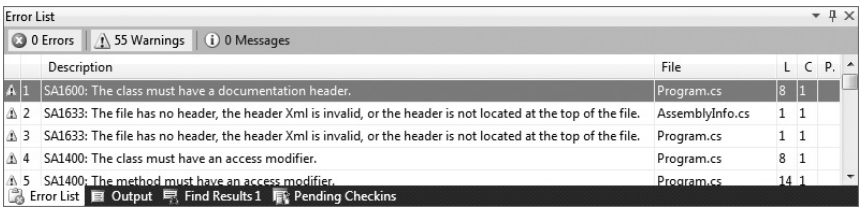


Рис. 13.15

Как и при работе с надстройкой FxCop, пользователь может выбрать правила, которые можно игнорировать, используя флажки, или отменить конкретные правила, добавив специальный атрибут `SuppressMessage`. Чаще всего пользователь использует контекстное меню, щелкая правой кнопкой мыши на сообщениях и выбирая команду Show Error Help для доступа к атрибутам сообщения Category, CheckId и Name, которое мы хотим подавить. Формат аргументов `SuppressMessage` имеет следующий вид `("[Category]","[CheckId]:[Name]")`, например `("Maintainability Rules", "SA1400: AccessModifierMustBeDeclared")`.

Надстройка Code Contracts

Последний инструмент, который мы собираемся описать, — надстройка Microsoft Code Contracts, которую в момент написания книги можно было загрузить с сайта проекта Code Contracts на веб-странице Microsoft DevLabs (`http://msdn.microsoft.com/devlabs`). После загрузки и инсталляции этого инструмента пользователь должен заново запустить систему Visual Studio 2010, чтобы появились расширения интегрированной среды. После повторного открытия проекта понадобится также добавить ссылку на метод `Microsoft.Contracts`, которая должна находиться на закладке .NET диалогового окна Add References (чтобы его открыть, необходимо дважды щелкнуть на проекте в окне Solution Explorer и выбрать команду Add References).

Теперь можно добавить в код контракты, имеющие вид пред- и постусловий. В следующем примере показан набор предусловий для метода Divide, который требует (с помощью члена Contract.Requires), чтобы знаменатель не был равен нулю. Аналогично, существует постусловие, гарантирующее (с помощью члена Contract.Ensure), что метод Add увеличит значение поля currentValue на правильную величину.

C#

```
private double currentValue;
private double Divide(double denominator){
    Contract.Requires(denominator != 0);
    return currentValue / denominator;
}
private void Add(double valueToAdd){
    Contract.Ensures(currentValue==Contract.OldValue(currentValue)+valueToAdd);
    // Ничего не делает в нарушение контракта
}
private void InvokeDivision(){
    currentValue = 7.0;
    double c = Divide(0); // не проходит верификацию, потому что b == 0
}
private void InvokeAddition(){
    currentValue = 13.0;
    Add(6);
}
```

VB

```
Private currentValue As Double
Private Function Divide(ByVal denominator As Double) As Double
    Contract.Requires(denominator < > 0)
    Return currentValue / denominator
End Function
Private Sub Add(ByVal valueToAdd As Double)
    Contract.Ensures(currentValue=Contract.OldValue(currentValue)+valueToAdd)
    ' Ничего не делает в нарушение контракта
End Sub

Private Sub InvokeDivision()
    currentValue = 7.0
    Dim c = Divide(0.0) ' не проходит верификацию, потому что b == 0
End Sub
Private Sub InvokeAddition()
    currentValue = 13.0
    Add(6)
End Sub
```

При наличии контрактов пользователь должен иметь возможность верифицировать их с помощью закладки Code Contracts на странице свойств проекта, как показано на рис. 13.16. Теперь после сборки и запуска приложения пользователь увидит диалоговое окно Assert, которое будет открываться при вызове метода InvokeDivision или InvokeAddition, что свидетельствует о нарушении контракта.

Здесь мы видим, что пользователь установил флажки Perform Runtime Contract Checking и Assert on Contract Failure. Если сбросить эти флажки, то при нарушении контракта будет генерироваться исключение ContractException, которое можно обработать в коде.

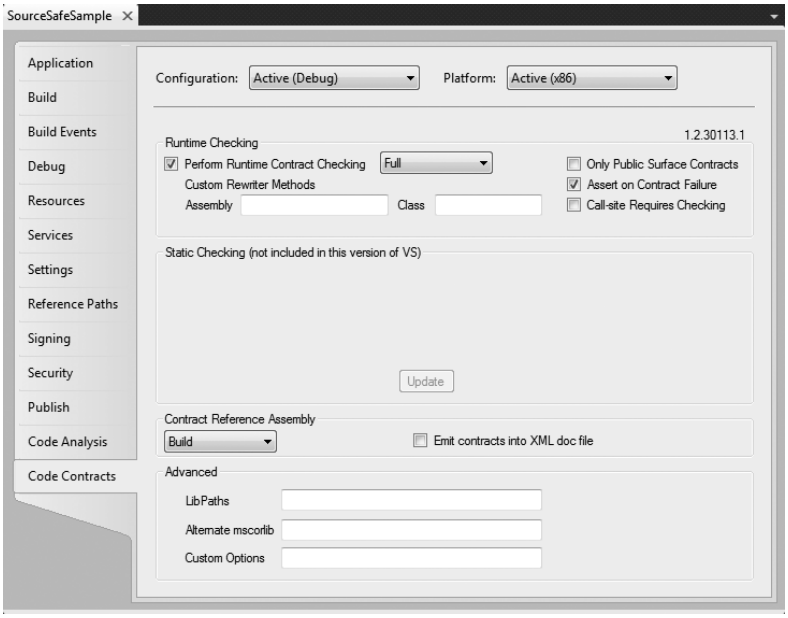


Рис. 13.16



На рис. 13.16 показана область Static Checking, которая отображается, если пользователь установил надстройку Code Contracts для выпуска Visual Studio 2010 Premium и более новых версий. Это открывает возможность выполнять статическую проверку, чтобы предотвратить нарушение контракта еще на стадии проектирования и не ждать, пока они будут нарушены на этапе выполнения приложения.

Резюме

В главе продемонстрирован богатый интерфейс системы Visual Studio 2010, позволяющий использовать хранилище исходного кода для управления файлами, связанными с приложением. Отправление и извлечение файлов можно выполнить из окна Solution Explorer, а доступ к более широким функциональным возможностям можно получить с помощью окна Pending Changes.

Хотя возможностей надстройки SourceSafe вполне достаточно для отдельных разработчиков и небольших коллективов, оно не предназначено для использования в крупномасштабных проектах. Кроме того, она не позволяет отслеживать задания и просматривать комментарии при внесении изменений. В главе 57 мы обсудим преимущества и дополнительные возможности, предоставляемые системой Team Foundation Server, которая относится к механизмам управления исходным кодом в масштабах предприятия.

В главе описаны надстройки FxCop, StyleCop и Code Contracts, которые можно использовать для повышения качества, надежности и согласованности кода. Тесная интеграция с системой Visual Studio 2010 делает их бесценными инструментами для коллективов разработчиков любой численности.

14

Генерация кода с помощью генератора T4

В ЭТОЙ ГЛАВЕ...

- Использование шаблонов генератора T4 для генерации текста и кода
- Проблемы, связанные с шаблонами генератора T4
- Создание предварительно обработанных шаблонов генератора T4 для включения в проекты

Часто при создании программного обеспечения разработчику приходится писать большие фрагменты стандартного кода, состоящего из повторяющихся шаблонов. Работа с такими фрагментами может оказаться очень трудоемкой и утомительной.

Создание такого кода лучше всего поддается автоматизации. Генерация кода — обычная практика при разработке программного обеспечения, при которой автором компонентов программы является не человек, а некий механизм. Инструмент, используемый для автоматического создания кода, называется генератором кода. На рынке существует множество коммерческих и свободно распространяемых генераторов кода — от универсальных до узко специализированных. Система Visual Studio 2010 содержит генератор кода, который может создавать файлы на основе простых шаблонных определений. Этот генератор кода называется Text Template Transformation Toolkit, или просто T4.

Генератор кода T4 был изобретен как часть пакета Domain Specific Languages Toolkit, который представлял собой надстройку для системы Visual Studio 2005. Генератор кода T4 входил также в систему Visual Studio 2008, но был плохо документирован, и в интегрированной среде разработки было всего несколько подсказок на эту тему. В системе Visual Studio 2010 механизм T4 стал наконец полноправным компонентом, так что теперь его можно вызывать с помощью команды File⇨New.

В данной главе рассматриваются вопросы, связанные с созданием, настройкой и выполнением шаблонов генератора T4. Кроме того, в ней рассматриваются возможные проблемы и способы их устранения. В заключение мы создадим шаблон `Preprocessed Text Template`, позволяющий создавать повторно используемые шаблоны T4, которые можно без труда вызывать в своих программах.

Создание шаблона генератора T4

В предыдущих версиях системы Visual Studio создание нового шаблона генератора T4 было скрытой возможностью, связанной с созданием текстового файла, имеющего расширение `.tt`. В системе Visual Studio 2010 разработчик может создавать шаблоны генератора T4, просто выбирая команду Text Template на панели General, находящейся в диалоговом окне Add New Item, показанным на рис. 14.1.

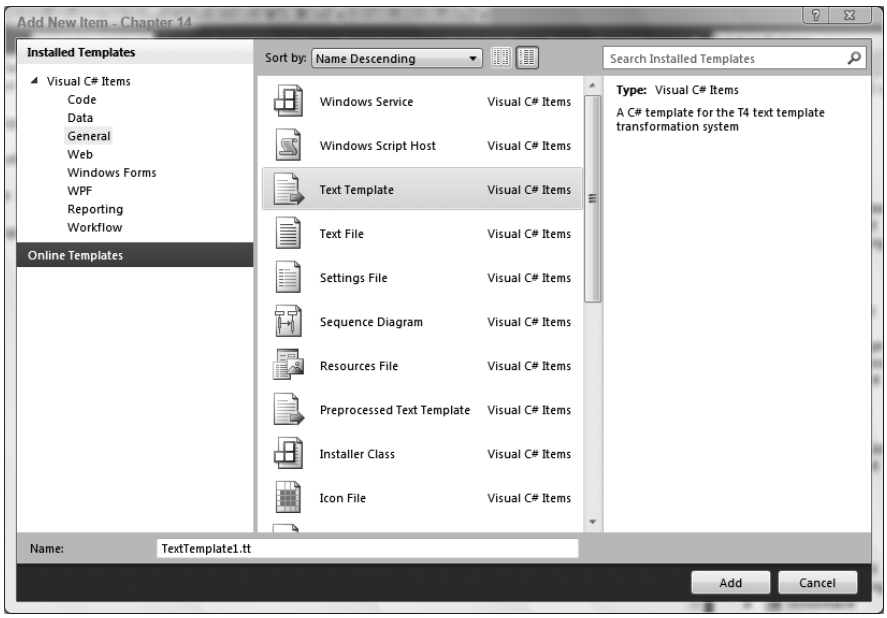


Рис. 14.1

Когда создается или сохраняется новый шаблон генератора T4, система Visual Studio выводит на экран окно с предупреждением, показанное на рис. 14.2. Шаблон генератора T4 выполняет обычный .NET-код, и теоретически его можно использовать для выполнения *любого* .NET-кода. Шаблоны генератора T4 выполняются каждый раз, когда они сохраняются, поэтому разработчик много раз увидит это предупреждение. Существует настройка, позволяющая отменить вывод этих сообщений, но она распространяется на все шаблоны во всех решениях. Если пользователь сбросил этот флажок, а затем решил все же получать сообщения, то он может восстановить их, установив значение Show Security Message равным True с помощью команды Tools⇨Options⇨Text Templating.

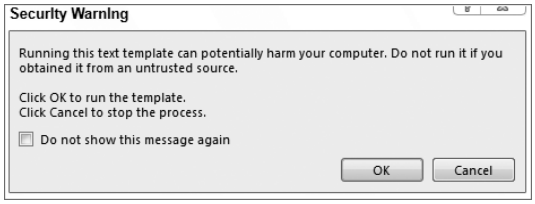


Рис. 14.2

Как только шаблон будет создан, он появляется в окне Solution Explorer как файл с расширением `.tt`. Шаблон можно открыть и увидеть, какой файл он генерирует. Каждый шаблон генерирует отдельный файл, имеющий то же самое имя, что и шаблон, но другое расширение. На рис. 14.3 в окне Solution Explorer показаны шаблонный файл и файл, который он генерирует.



Рис. 14.3



Если программное обеспечение создается на языке VB, то, для того чтобы увидеть сгенерированный файл, необходимо установить флажок Show All Files для текущего проекта.

Сгенерированный файл изначально пуст, поскольку в шаблонном файле никакие данные для вывода не определяются. Однако сам шаблонный файл не пуст. Когда он генерируется в первый раз, он содержит следующие две строки.

```
<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".txt" #>
```

Каждая из этих двух строк является *директивой* генератора T4, контролирующей некий аспект выполнения шаблона. Директивы генератора T4 будут рассмотрены позднее, но в данный момент нас интересует несколько вопросов. Директива `template` содержит атрибут `language`, определяющий язык, который будет использоваться шаблоном. Каждый шаблонный файл может содержать инструкции, выполнение которых приводит к созданию результирующего файла, и этот атрибут сообщает системе Visual Studio, на каком языке должны генерироваться эти инструкции.



Язык шаблона не влияет на генерируемый файл. Можно генерировать файл на языке C# по шаблону, использующему язык VB, и наоборот. По умолчанию используется язык текущего проекта, но его можно изменить. Шаблоны на языках C# и VB поддерживаются в проектах на любом языке.

Вторым интересующим нас аспектом является атрибут `extension` в директиве `output`. Имя генерируемого файла всегда совпадает с именем шаблонного файла, но расширение `.tt` заменяется содержимым этого атрибута. Как только система Visual Studio распознает расширение генерируемого файла, она обрабатывает его точно так же, как если бы разработчик создал его с помощью диалогового окна Add New Item. В частности, если расширение относится к исходному файлу кода, например `.cs` или `.vb`, система Visual Studio включает сгенерированный файл в процесс сборки текущего проекта.



Если расширение результирующего файла изменилось, то при следующем запуске шаблона ранее сгенерированный файл удаляется. Поскольку разработчик не обязан редактировать сгенерированный файл, это не должно вызывать никаких проблем.

Добавим в конце шаблонного файла отдельную строку, содержащую слова `Hello World`, и сохраним шаблон.



C#

```
<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".txt" #>
Hello World
```

Сниппет кода HelloWorld.tt



VB

```
<#@ template debug="false" hostspecific="false" language="VB" #>
<#@ output extension=".txt" #>
Hello World
```

Сниппет кода HelloWorld.tt

Как указывалось выше, шаблоны выполняются каждый раз, когда они сохраняются на диске, поэтому сгенерированный файл будет содержать новое содержание шаблона. Если открыть сгенерированный файл, то мы увидим текст `Hello World`.

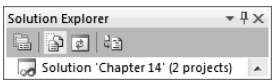


Рис. 14.4

Помимо того что каждый шаблонный файл всегда можно сгенерировать заново, открыв и сохранив его на диске, в верхней части инструментального окна Solution Explorer есть кнопка Transform All Templates (рис. 14.4), щелчок на которой изменяет все шаблоны в решении.

Как указывалось ранее, если директива `output` задает расширение, соответствующее языку текущего проекта, то результирующий сгенерированный файл будет включен в проект. Типы и члены, определенные в сгенерированном файле, позволяют полностью использовать технологию IntelliSense. В следующем сниппете показан шаблон генератора T4 вместе с кодом, который он генерирует. Сгенерированный класс может быть доступен в других частях программы. Этот факт демонстрирует следующее небольшое консольное приложение.



C#

```
<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".cs" #>

namespace AdventureWorks {
  class GreetingManager {
    public static void SayHi() {
      System.Console.WriteLine("Aloha Cousin!");
    }
  }
}
```

Сниппет кода GreetingManager.tt

```
namespace AdventureWorks {
  class GreetingManager {
    public static void SayHi() {
      System.Console.WriteLine("Aloha Cousin!");
    }
  }
}
```

Сниппет кода GreetingManager.cs

```
namespace AdventureWorks {
  class Program {
    static void Main(string[] args) {
```

```
      GreetingManager.SayHi();
    }
  }
}
```

Сниппет кода Program.cs



VB

```
<#@ template debug="false" hostspecific="false" language="VB" #>
<#@ output extension=".vb" #>
Public Class GreetingManager
  Public Shared Sub SayHi
    System.Console.WriteLine("Aloha Cousin!")
End Sub
End Class
```

Сниппет кода GreetingManager.tt

```
Public Class GreetingManager
    Public Shared Sub SayHi()
        System.Console.WriteLine("Aloha Cousin!")
    End Sub
End Class
```

Сниппет кода GreetingManager.vb

```
Module Module1
  Sub Main()
    GreetingManager.SayHi()
  End Sub
End Module
```

Сниппет кода Module1.vb



Несмотря на то что в остальной части приложения можно без ограничений применять технологию IntelliSense к сгенерированному коду, сами шаблонные файлы генератора T4 не используют ни технологию IntelliSense, ни синтаксическую подсветку, принятую в системе Visual Studio 2010. Впрочем, некоторые редакторы и надстройки, созданные сторонними разработчиками, компенсировали этот недостаток и обеспечивают более удобную работу с генератором T4.

Этот пример работает, но он не способен продемонстрировать всю мощь и гибкость генератора T4. Это объясняется тем, что шаблон является полностью статическим. Для того чтобы создать полезные шаблоны, требуются динамические свойства.

Строительные блоки генератора T4

Каждый шаблон генератора T4 состоит из большого количества *блоков*, влияющих на генерируемый файл. Строка `Hello World` из первого примера является блоком текста. Текстовые блоки буквально копируются из шаблонного файла в генерируемый файл и могут содержать любой текст и другие блоки.

Кроме текстовых блоков, существуют еще три типа блоков: блоки выражений (Expression blocks), управляющие блоки (Statement blocks) и блоки элементов класса (Class Feature blocks). Каждый из этих типов блоков окружен специфической разметкой, позволяющей их идентифицировать. Текстовые блоки — это единственный тип блоков, не имеющий специальной разметки.

Блоки выражений

Блоки выражений используются для передачи вычисленного значения в генерируемый файл. Блоки выражений обычно появляются в текстовых блоках и обозначаются дескрипторами <#= и #>. Рассмотрим пример шаблона, выводящего дату и время, когда был сгенерирован файл.



C#

```
<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".txt" #>
This file was generated: <#=System.DateTime.Now #>
```

Сниппет кода Existential.tt



VB

```
<#@ template debug="false" hostspecific="false" language="VB" #>
<#@ output extension=".txt" #>
This file was generated: <#=System.DateTime.Now #>
```

Сниппет кода Existential.tt

Выражение в блоке может быть любым выражением, допустимым в языке шаблона, заданном директивой `template`. При каждом выполнении шаблон вычисляет это выражение, а затем применяет к результату метод `ToString()`. Затем это значение вставляется в генерируемый файл.

Управляющие блоки

Управляющие блоки используются для выполнения произвольных инструкций при запуске шаблона. Код в управляющем блоке может регистрировать выполнение шаблона, создавать временные переменные или удалять файл с диска, поэтому разработчик должен быть осторожным. По существу, код в управляющем блоке может состоять из любых инструкций, допустимых в языке шаблона. Управляющие блоки обычно используются для реализации потока управления в шаблоне, управления временными переменными и взаимодействия с другими системами. Управляющий блок обозначается дескрипторами <# и #>, которые похожи на разделители блоков выражений, но не используют знак равенства. Следующий пример создает файл, содержащий все 99 куплетов популярной застольной песни.



C#

```
<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".txt" #>
<# for(int i = 99; i> = 1; i--)
   { #>
<#=i #> Bottles of Non-alcoholic Carbonated Beverage on the wall
<#=i #> Bottles of Non-alcoholic Carbonated Beverage
Take one down
And pass it around
<# if(i-1 == 0) { #>
There's no Bottles of Non-alcoholic Carbonated Beverage on the wall
<# } else { #>
```

```
There's <#=i-1 #> Bottles of Non-alcoholic Carbonated Beverage on the wall
<# } #>

   <# } #>
```

Сниппет кода DrinkingSong.tt



VB

```
<#@ template debug="false" hostspecific="false" language="VB" #>
<#@ output extension=".txt" #>
<# For i As Integer = 99 To 1 Step -1 #>
<#= i #> Bottles of Non-alcoholic Carbonated Beverage on the wall
<#= i #> Bottles of Non-alcoholic Carbonated Beverage
Take one down
And pass it around
<# If i-1 = 0 Then #>
There's no Bottles of Non-Alcoholic Carbonated Beverage on the wall.
<# Else #>
There's <#= i-1 #> Bottles of Non-alcoholic Carbonated Beverage on the
wall.
<# End If #>

<# Next #>
```

Сниппет кода DrinkingSong.tt



В предыдущем примере управляющий блок содержит текстовый блок, который в свою очередь содержит множество блоков выражений. Даже используя только эти три типа блоков, можно создавать очень мощные шаблоны.

Хотя управляющий блок в данном примере содержит другие блоки, в общем случае это не обязательно. Из управляющего блока можно непосредственно выводить информацию, используя методы `Write()` и `WriteLine()`. Рассмотрим пример использования этих методов.



C#

```
<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".txt" #>
<#
for(int i = 99; i> 1; i--)
{
  WriteLine("{0} Bottles of Non-alcoholic Carbonated Beverage
  on the wall", i);
  WriteLine("{0} Bottles of Non-alcoholic Carbonated Beverage", i);
  WriteLine("Take one down");
  WriteLine("And pass it around");
  if(i-1 == 0) {
    WriteLine(
     "There's no Bottles of Non-alcoholic Carbonated Beverage on the wall.");
  } else {
    WriteLine(
    "There's {0} Bottles of Non-alcoholic Carbonated Beverage
     on the wall.",i-1);
  }
  WriteLine("");
} #>
```

Сниппет кода ImperativeDrinkingSong.tt



VB

```
<#@ template debug="false" hostspecific="false" language="VB" #>
<#@ output extension=".txt" #>
<# For i As Integer = 99 To 1 Step -1
  Me.WriteLine("{0} Bottles of Non-alcoholic Carbonated Beverage on the
wall", i)
  Me.WriteLine("{0} Bottles of Non-alcoholic Carbonated Beverage", i)
  Me.WriteLine("Take one down")
  Me.WriteLine("And pass it around")
  If i-1 = 0 Then
    WriteLine("There's no Bottles of Non-Alcoholic Carbonated Beverage
    on the"
    @_ " wall.")
  Else
    WriteLine("There's {0} Bottles of Non-alcoholic Carbonated Beverage
    on the"
    &_" wall.",i-1)
  End If
  Me.WriteLine("")

Next #>
```

Сниппет кода ImperativeDrinkingSong.tt

Окончательные результаты обоих примеров совпадают. В зависимости от шаблона разработчик может применять те способы, которые он лучше знает. Однако во избежание недоразумений желательно использовать только один выбранный способ к каждому отдельному шаблону.

Блоки элементов класса

Последний тип блоков генератора T4 — блоки элементов класса. Блоки этого типа содержат произвольный код, который можно вызывать из управляющих блоков и блоков выражений, чтобы облегчить создание генерируемого кода. Они часто содержат специальное форматирование кода или повторяющиеся задачи. Блоки элементов класса обозначаются дескрипторами <#+ и #>, которые не отличаются от дескрипторов блоков выражений, за исключением того, что в открывающем дескрипторе вместо символа = используется символ +. Следующий шаблон записывает числа от –5 до 5, используя обычный финансовый формат, в котором каждое число содержит две десятичные цифры и символ доллара перед ними, а отрицательные числа записываются точно так же, как и положительные, но в квадратных скобках.



C#

```
<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".txt" #>

Financial Sample Data
<# for(int i = -5; i <= 5; i++)
   {
     WriteFinancialNumber(i);
         WriteLine("");
    } #>
End of Sample Data

<#+
    void WriteFinancialNumber(decimal amount)
```

```
    {
      if(amount <0)
        Write("(${0:#0.00})", System.Math.Abs(amount));
      else
        Write("${0:#0.00}", amount);
    }
    #>
```

Сниппет кода FinancialData.tt



VB

```
<#@ template debug="true" hostspecific="false" language="VB" #>
<#@ output extension=".txt" #>

Financial Sample Data
<# For i as Integer = -5 To 5
   WriteFinancialNumber(i)
   WriteLine("")
Next #>
End of Sample Data

<#+
Sub WriteFinancialNumber(amount as Decimal)
  If amount <0 Then
    Write("(${0:#0.00})", System.Math.Abs(amount))
  Else
    Write("${0:#0.00}", amount)
  End If
End Sub
#>
```

Сниппет кода FinancialData.tt

Блоки элементов класса могут содержать текстовые блоки и блоки выражений, но не могут содержать управляющие блоки. Кроме того, после появления первого блока элементов класса управляющие блоки использовать нельзя.

Итак, мы рассмотрели четыре разных типа блоков генератора кода T4, которые могут появляться в шаблонном файле. Настало время показать, как система Visual Studio 2010 использует их для генерации результирующего файла.

Как работает генератор кода T4

Процесс генерации файла на основе шаблона генератора T4 состоит из двух основных этапов. На первом этапе файл с расширением `.tt` используется для генерации стандартного класса платформы .NET. Этот класс является производным от класса `abstract (MustInherit)Microsoft.VisualStudio.TextTemplating.TextTransformation` и замещает метод `TransformText()`.

На втором этапе создается и настраивается экземпляр этого класса, а затем вызывается метод `TransformText`. Этот метод возвращает строку, которая используется как содержимое генерируемого файла.

Как правило, пользователь не интересуется содержимым генерируемого файла классов, но, включив режим отладки для шаблонов, он может настроить генератор T4, чтобы тот создавал доступные копии. Для этого достаточно установить атрибут `debug` директивы `template` равным `true` и сохранить шаблонный файл.

После выполнения шаблона генератора T4 в режиме Debug создается много файлов во временной системной папке. Один из этих файлов будет иметь случайное имя и расширение .cs или .vb (в зависимости от языка шаблона). Этот файл содержит реальный класс генератора.



Временную системную папку можно найти, открыв командную строку системы Visual Studio и введя команду эха %TEMP%.

Этот код содержит множество директив препроцессора, поддерживающих отладку, но этот код довольно трудно читать. Рассмотрим содержимое файла, сгенерированного по шаблону FinancialSample.tt, описанному в предыдущем разделе, но отформатированному и без директив препроцессора.



C#

```
namespace
Microsoft.VisualStudio.TextTemplatingBE7601CBE8A6858147D586FD8FC4C6F9
{
  using System;
  public class GeneratedTextTransformation :
         Microsoft.VisualStudio.TextTemplating.TextTransformation
{
    public override string TransformText()
    {
      try
      {
        this.Write("\r\nFinancial Sample Data\r\n");

        for(int i = -5; i <= 5; i++)
        {
           WriteFinancialNumber(i);
           WriteLine("");
        }

        this.Write("End of Sample Data\r\n\r\n ");
      }
      catch (System.Exception e)
      {
        System.CodeDom.Compiler.CompilerError error = new~CA
                    System.CodeDom.Compiler.CompilerError();
        error.ErrorText = e.ToString();
        error.FileName = "C:\\dev\\Chapter 14\\Chapter 14\\Finance.tt";
        this.Errors.Add(error);
      }
      return this.GenerationEnvironment.ToString();
    }

    void WriteFinancialNumber(decimal amount)
    {
      if(amount <0)
        Write("({0:#0.00})", System.Math.Abs(amount));
      else
        Write("{0:#0.00}", amount);
      }
   }
}
```



VB

```
Imports System
Namespace
Microsoft.VisualStudio.TextTemplating2739DD4202E83EF5273E1D1376F8FC4E
  Public Class GeneratedTextTransformation
    Inherits Microsoft.VisualStudio.TextTemplating.TextTransformation

    Public Overrides Function TransformText() As String
      Try
        Me.Write("" & Global.Microsoft.VisualBasic.ChrW(13) _
          & Global.Microsoft.VisualBasic.ChrW(10) _
          & "Financial Sample Data" _
          & Global.Microsoft.VisualBasic.ChrW(13) _
          & Global.Microsoft.VisualBasic.ChrW(10)) _

          For i as Integer = -5 To 5
            WriteFinancialNumber(i)
            WriteLine("")
          Next

          Me.Write("End of Sample Data" _
            & Global.Microsoft.VisualBasic.ChrW(13) _
            & Global.Microsoft.VisualBasic.ChrW(10) _
            & Global.Microsoft.VisualBasic.ChrW(13) _
            & Global.Microsoft.VisualBasic.ChrW(10) & " ")
          Catch e As System.Exception
            Dim [error] As System.CodeDom.Compiler.CompilerError = _
               New System.CodeDom.Compiler.CompilerError()
            [error].ErrorText = e.ToString
            [error].FileName = "C:\\dev\\Chapter 14\\Chapter 14_\\
                               Finance.tt"
            Me.Errors.Add([error])
          End Try
          Return Me.GenerationEnvironment.ToString
    End Function

    Sub WriteFinancialNumber(amount as Decimal)
      If amount <0 Then
        Write("(${0:#0.00})", System.Math.Abs(amount))
      Else
        Write("${0:#0.00}", amount)
      End If
    End Sub

  End Class
End Namespace
```

Обратите внимание на несколько интересных моментов, связанных с этим кодом. Во-первых, этот шаблон выполняется путем вызова метода `TransformText()`. Содержимое этого метода выполняется в контексте блока `try-catch`, в котором перехватываются и хранятся все ошибки. Система Visual Studio 2010 знает, как извлечь эти ошибки и вывести сообщение о них в инструментальном окне.

Следующий интересный аспект связан с использованием метода `Write()`. Легко видеть, что каждый блок типа `Text` был представлен в виде отдельной строки, которая была передана методу `Write()`. При этом она была скрытно добавлена в свойство `GenerationEnvironment`, которое в свою очередь превратилось в строку и было возвращено генератору T4.

Управляющие блоки и блоки элементов класса копируются в генерируемый класс дословно. Разница состоит в том, где они оказываются. Управляющие блоки оказываются в методе `TransformText()`, а блоки элементов класса — после него, хотя находятся в той же области видимости. Это должно подсказать разработчику, какие сущности следует объявлять в блоке элементов класса.

В заключение отметим, что блоки выражений вычисляются, а их результат передается в метод `Microsoft.VisualStudio.TextTemplating.ToStringHelper.ToStringWithCulture()`. Этот метод возвращает строку, которая затем передается обратно методу `Write()`, как будто это текстовый блок. Обратите внимание на то, что класс `ToStringHelper` учитывает особенности локализации при создании строки из выражения. Этот параметр передается как атрибут в директиве `template`.

Когда выполнение метода `TransformText()` заканчивается, строка возвращается в среду выполнения программ, т.е. в систему Visual Studio 2010, которая сама решит, что делать с этой строкой. Для решения этой задачи система Visual Studio использует директиву `output`. Директивы будут рассмотрены в следующем разделе.



Прежде чем продвигаться дальше, отметим, что из предыдущего раздела следует, что генератор T4 не обязательно запускать в системе Visual Studio. Существует инструмент `TextTransform.exe`, который можно вызывать из командной строки. Его можно найти в папке `%CommonProgramFiles%\microsoftshared\TextTemplating\10.0\folder` (на 64-битовых компьютерах см. папку `C:\Program Files(x86)\Common Files\microsoft shared\TextTemplating\10.0\`). Несмотря на то что его тоже можно применять для генерации файлов в процессе сборки, генератор T4 сам по себе гарантирует наличие определенных библиотек, которые должны быть инсталлированы вместе с системой Visual Studio. Это значит, что при использовании отдельного компьютера для сборки на нем необходимо установить систему Visual Studio. В системе Visual Studio файлы с расширением `.tt` обрабатываются инструментом `TextTemplatingFileGenerator`.

Директивы генератора T4

Шаблон генератора T4 может взаимодействовать со средой выполнения программ с помощью директив. Каждая директива должна быть набрана в отдельной строке и обозначена дескрипторами `<#@` и `#>`. В этом разделе рассматриваются пять стандартных директив.

Шаблонные директивы

Директива `template` управляет множеством разных опций, касающихся самого шаблона. Она содержит следующие атрибуты.

- `language`. Определяет язык платформы .NET, используемый в шаблоне в блоках выражений, управляющих блоках и блоках элементов класса. Корректными значениями являются `C#` и `VB`.
- `inherits`. Определяет базовый класс для генерируемого класса, который используется для создания результирующего файла. Методы базового класса могут быть замещены, чтобы обеспечить дополнительные функциональные возможности

в шаблонных классах. Любой новый базовый класс должен быть производным от класса `Microsoft.VisualStudio.TextTemplating.TextTransformation`, который задается для этого атрибута по умолчанию.



Если пользователь хочет вывести свой класс от другого базового класса, он должен использовать директиву `assembly` (подробнее об этом — ниже, в разделе "Директива `assembly`"), чтобы сделать его доступным для генератора T4.

- `culture`. Выбирает локализацию для шаблона. Значения должны быть заданы в стандартном формате xx-XX (en-US, ja-JP и т.д.). По умолчанию это значение представляет собой пустую строку, которая задает свойство `Invariant Culture`.
- `debug`. Включает режим Debug. В результате возникает файл, содержащий класс генератора, который будет формировать дамп во временной системной папке. Может принимать значение `true` или `false`. По умолчанию равен `false`.
- `hostspecific`. Означает, что шаблонный файл предназначен для работы на конкретной хост-системе. Если установлен равным `true`, то свойство `Host` раскрывается в шаблоне. При запуске в системе Visual Studio 2010 это свойство имеет тип `Microsoft.VisualStudio.TextTemplating.VSHost.TextTemplatingService`. По умолчанию равно `false`. Хотя эта тема выходит за рамки рассмотрения книги, но разработчик может написать свою собственную хост-систему для генератора T4 и использовать ее для выполнения шаблонных файлов.

Директива `Output`

Директива `output` используется для управления файлом, генерируемым шаблоном. Она имеет два свойства.

- `extension`. Задает расширение, которое будет добавлено к имени генератора для создания имени результирующего файла. Содержимое этого свойства по существу заменяет расширение `.tt` в имени шаблонного файла. По умолчанию это расширение `.cs`, но оно может содержать любую последовательность символов, допускаемых файловой системой.
- `encoding`. Управляет кодировкой генерируемого файла. Это свойство может быть результатом любой кодировки, возвращенной методом `System.Text.Encoding.GetEncodings()`; иначе говоря, UTF-8, ASCII или Unicode. По умолчанию значение этого свойства равно `Default`, поэтому кодировка, заданная по умолчанию, представляет собой текущую ANSI-страницу кодов операционной системы, под управлением которой должен выполняться шаблон.

Директива `assembly`

Директива `assembly` используется для того, чтобы код в шаблонном файле получил доступ к классам и типам, определенным в других сборках. Это похоже на добавление ссылки в обычный .NET-проект. Директива имеет одно свойство с именем `name`, которое должно содержать один из следующих элементов.

- Имя сборки. Сборка будет загружаться из того же каталога, что и шаблон генератора T4.

- Абсолютный путь к сборке. Сборка будет загружаться из каталога с заданным точным путем.
- Относительный путь к сборке. Сборка будет загружаться из места, определяемого каталогом, в котором расположен шаблон генератора T4.
- Строгое имя сборки. Сборка будет загружаться из кэша Global Assembly Cache (CAG).

Директива `Import`

Директива `import` используется для того, чтобы обеспечить удобный доступ к элементам без указания его пространства имен и типа. Эта директива работает так же, как инструкция `Import` в языке VB или инструкция `using` в языке C#. Имеет единственный атрибут `namespace`. По умолчанию пространство имен `System` уже импортировано. Следующий пример демонстрирует небольшие управляющие блоки с директивой и без директивы `import`.



C#

```
<#
  var myList = new System.Collections.Generic.List < string > ();
  var myDictionary = new System.Collections.Generic.Dictionary < string,
  System.Collections.Generic.List < string > > ();
#>
```

Сниппет кода WithoutImport.tt



VB

```
<#
Dim myList As New System.Collections.Generic.List(Of String)
Dim myDictionary As New System.Collections.Generic.Dictionary(Of
System.String,
System.Collections.Generic.List(Of String))
#>
```

Сниппет кода WithImport.tt



C#

```
<#@ import namespace="System.Collections.Generic" #>
<#
  var myList = new List <string> ();
  var myDictionary = new Dictionary <string, List <string>> ();
#>
```

Сниппет кода WithImport.tt



VB

```
<#@ import namespace="System.Collections.Generic" #>
<#
Dim myList As New List(Of String)
Dim myDictionary As New Dictionary(Of String, List(Of String))
#>
```

Сниппет кода WithImport.tt



Преимущество импортирования кода с помощью директив `import` и `assembly` заключается в том, что этот код выполняется при запуске шаблона T4, но не содержится в окончательном файле. Если необходимо открыть доступ к другим пространствам имен в сгенерированном результирующем файле, то в него необходимо добавить инструкции `using` и `Import`, а также ссылки на свой проект.

Директива `Include`

Директива `include` позволяет копировать содержимое другого файла непосредственно в шаблонный файл. Она имеет единственный атрибут `file`, который должен иметь относительный или абсолютный путь к включаемому файлу. Если другой файл содержит директивы генератора T4 или блоки, то они также будут выполнены. Следующий пример демонстрирует вставку лицензионного соглашения BSD License в комментарий, расположенный в начале генерируемого файла.



```
' Copyright (c) <#=DateTime.Now.Year#> , <#=CopyrightHolder#>
' All rights reserved.

' Redistribution and use in source and binary forms, with or without
...
```

Сниппет кода License.txt

C#

```
<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".generated.cs" #>

<# var CopyrightHolder = "AdventureWorks Inc."; #>
/ *
<#@ include file="License.txt" #>
* /
namespace AdventureWorks {
  // ...
}
```

Сниппет кода IncludeSample.tt



VB

```
<#@ template debug="false" hostspecific="false" language="VB" #>
<#@ output extension=".vb" #>

<# Dim CopyrightHolder = "AdventureWorks Inc." #>

<#@ include file="License.txt" #>

Namespace AdventureWorks
  ' ...
End Namespace
```

Сниппет кода IncludeSample.tt

Проблемы

По мере увеличения и усложнения шаблонного файла вероятность ошибок возрастает. Ситуация усложняется еще и тем, что ошибки могут возникать на разных этапах и каждая из них должна обрабатываться немного иначе, чем другая. Несмотря на то что генератор T4 запускает эти процессы поочередно, следует помнить, что при этом может произойти любая ошибка, а сам шаблонный файл выполняется каждый раз, когда сохраняется на диске.

При внесении любых изменений в шаблонный файл мы настоятельно рекомендуем делать небольшие шаги, часто повторять генерацию и немедленно устранять изменения, которые привели к ошибке.

Ошибки на этапе проектирования

Первые ошибки могут возникнуть, когда система Visual Studio пытается прочитать шаблон генератора T4 и использовать его для создания временного класса платформы .NET. На рис. 14.5 мы умышленно пропустили знак решетки в открывающем дескрипторе для блока выражений. В результате шаблон стал неправильным. Окно Error List (см. рис. 14.5, *снизу*) свидетельствует о том, что система Visual Studio довольно легко идентифицирует такие ошибки. Она даже способна правильно определить номер строки, в которой возникла ошибка.

Другой тип ошибок, которые часто обнаруживаются на этапе проектирования связаны с директивами. Во многих случаях возникают проблемы с атрибутами директив и из значениями, заданными по умолчанию. Если значение, заданное по умолчанию, не имеет смысла, например для директив `import`, `include` и `assembly` , возникает ошибка.

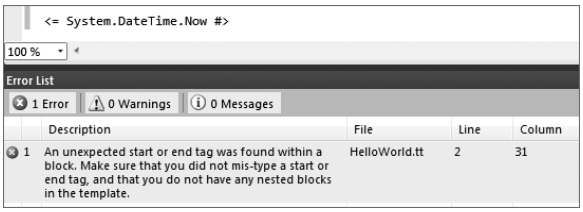


Рис. 14.5



Интересным исключением из правил, по которым система Visual Studio обрабатывает неправильные директивы, является атрибут `extension` директивы `output`. Если значение является неправильным, то система выдает предупреждение, но генерируемый файл не создается вообще. Если существует другой код, зависящий от содержимого генерируемого файла, то процесс фоновой компиляции быстро обнаружит каскад ошибок, который может оказаться огромным. Сначала необходимо убедиться, что файл можно вообще сгенерировать, и лишь потом пытаться исправить шаблон, временно удалив все содержимое шаблонного файла, кроме директив `template` и `output`.

Ошибки преобразования во время компиляции

Следующий этап, на котором могут возникать ошибки в генераторе T4, начинается, когда временный код платформы .NET, содержащий код класса генератора, компилируется в сборку. Ошибки, которые могут возникать на этом этапе, обычно являются результатом небрежного кодирования внутри блоков выражений, управляющих блоков и блоков элементов класса. И вновь система Visual Studio не только отлично находит и распознает сами ошибки, но и определяет файл и номер строки, в которой находится ссылка на генерируемый файл. Каждая ошибка, найденная генератором на этом этапе, сопровождается префиксом `Compiling Transformation`, чтобы их было легко идентифицировать.

В качестве первого шага к исправлению этих ошибок необходимо включить режим Debug в директиве шаблона. Это заставит генератор создать дамп файлов, использованных для компиляции кода, во временной папке. Когда будут созданы дампы этих

файлов, следует дважды щелкнуть на строке ошибки в окне Error List и открыть временный файл, чтобы понять, что случилось. Поскольку этот файл может иметь расширение `.cs` или `.vb`, система Visual Studio способна подсветить синтаксические конструкции и с помощью технологии IntelliSense изолировать проблемный фрагмент. Если понятно, что произошло, то ошибку намного легче найти и исправить.



Среди файлов, генерируемых в режиме отладки, есть файл с расширением `.cmdline`, содержащий аргументы, которые передаются файлу `csc.exe` или `vbc.exe`, когда генератор T4 компилирует шаблон. Этот файл можно использовать для повторения процесса компиляции. Кроме того, существует файл с расширением `.out`, содержащий вызов компилятора из командной строки и результаты его работы.

Ошибки преобразования во время выполнения

Последний этап, на котором могут возникать ошибки генерации, начинается, когда создается экземпляр кода генератора, который выполняется для создания содержимого генерируемого файла. По существу, на этом этапе происходит выполнение кода платформы .NET и, вероятнее всего, могут возникнуть ошибки, связанные с окружением или ошибочной логикой. Как и ошибки преобразования во время компиляции, ошибки, найденные на этом этапе, имеют префикс `Executing Transformation`, который легко обнаружить.

Для того чтобы устранить ошибки преобразования во время выполнения лучше всего использовать защитное программирование. Если обнаружена ошибка, например отсутствие файла или невозможность связаться с базой данных, то для уведомления генератора о возникшей проблеме можно использовать метод `Error()`. Как и все остальные, эти ошибки будут распознаны как ошибки преобразования во время выполнения, только сообщения о них будут более информативными.

```
if( !File.Exists(fileName) ) {
  this.Error("Невозможно найти файл");
}
```

Кроме метода `Error()`, существует эквивалентный метод `Warning()`, генерирующий предупреждения. Если шаблон генератора T4 обнаружит катастрофическую ошибку, например невозможность связаться с базой данных, чтобы получить из нее данные, он может сгенерировать исключение и остановить процесс выполнения. Информация об исключениях собирается и публикуется в инструментальном окне Error List.

Ошибки в сгенерированном коде

Несмотря на то что формально это не относится к процессу, выполняемому генератором T4, генерируемый файл может содержать ошибки времени компиляции или выполнения. Если существуют ошибки времени компиляции, то система Visual Studio просто обнаружит их как обычно. Для ошибок времени выполнения лучше применить метод модульного тестирования сложных типов, даже сгенерированных.

Теперь, когда мы выяснили, что происходит, когда обнаруживаются ошибки, настало время рассмотреть более крупный пример.

Преимущества генерирования кода

При разработке промышленных приложений программисты сталкиваются с данными, которые редко изменяются и представляются в коде как перечисления. Хранение таких данных в базе и их синхронизация со значениями перечислений представляют собой трудоемкую и рутинную задачу, которая является лучшим кандидатом на автоматизацию с помощью шаблонного генератора T4. Шаблон, представленный в этом разделе, связывается с базой данных `AdventureWorks` и создает перечисление, основываясь на содержимом таблицы `Sales.ContactType`.



C#

```
<<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".generated.cs" #>

<#@ assembly name="System.Data" #>
<#@ import namespace="System.Data.SqlClient" #>
<#@ import namespace="System.Text.RegularExpressions" #>

<#
var connectionString = "Data Source=.\\SQLEXPRESS;"
    + "Initial Catalog=AdventureWorks;"
    + "Integrated Security=true;";
var sqlString = "SELECT ContactTypeID, [Name] FROM
    + "[Person].[ContactType]";
#>

// This code is generated. Please do not edit it directly
// If you need to make changes please edit ContactType.tt instead
namespace AdventureWorks {
  public enum ContactType {
<#
using(var conn = new SqlConnection(connectionString))
using(var cmd = new SqlCommand(sqlString, conn))
{
  conn.Open();

  var contactTypes = cmd.ExecuteReader();

  while( contactTypes.Read() )
  {
  #>
    <#= ValidIdentifier( contactTypes[1].ToString() ) #> =
    <#=contactTypes[0]#>,
  <#}

  conn.Close();
}
#>
  }
}

<#+
  public string ValidIdentifier(string input)
  {
    return Regex.Replace(input, @"[^a-zA-Z0-9]", String.Empty );
  }
#>
```

Сниппет кода ContactTypes.tt



VB

```
<#@ template debug="false" hostspecific="false" language="VB" #>
<#@ output extension=".generated.vb" #>

<#@ assembly name="System.Data" #>
<#@ import namespace="System.Data.SqlClient" #>
<#@ import namespace="System.Text.RegularExpressions" #>

<#
Dim ConnectionString as String = "Data Source=.\SQLEXPRESS; " _
& "Initial Catalog=AdventureWorks; Integrated Security=true;"
Dim SqlString as String = "SELECT ContactTypeID,[Name] FROM
[Person].[ContactType]"
#>

' This code is generated. Please do not edit it directly
' If you need to make changes please edit ContactType.tt instead
Namespace AdventureWorks
  Enum ContactType

<#
Using Conn As New SqlConnection(ConnectionString), _
  Cmd As New SqlCommand(SqlString, Conn)

   Conn.Open()

  Dim ContactTypes As SqlDataReader = Cmd.ExecuteReader()

  While ContactTypes.Read()
#>
  <#= ValidIdentifier( contactTypes(1).ToString() ) #> = <#=contactTypes(0)#>
<#
  End While
  Conn.Close()
End Using
#>
  End Enum
End Namespace

<#+
  Public Function ValidIdentifier(Input as String) As String
    Return Regex.Replace(Input, "[^a-zA-Z0-9]", String.Empty )
  End Function
#>
```

Сниппет кода ContactTypes.tt

Первый раздел состоит из директив генератора T4. Первые две директивы задают язык для шаблона и расширение результирующего файла. Третья директива присоединяет сборку к генератору (для того чтобы открыть доступ к пространству имен `System.Data.SqlClient`), а последние две импортируют пространства имен в шаблон, как требует шаблонный код.

Второй раздел представляет собой управляющий блок. Он содержит несколько переменных, которые будут использоваться шаблоном. Для того чтобы впоследствии их было легче найти и изменить, они размещены в начале шаблонного файла. После объявлений переменных следует текстовый блок, содержащий несколько поясняющих комментариев, а также объявления пространства имен и перечисления. Они дословно копируются в генерируемый файл. Обычно целесообразно размещать в гене-

рируемом файле комментарий, объясняющий его происхождение и способ редактирования. Это позволит предотвратить ужасные последствия, которые могут произойти, когда внесенные изменения стираются после повторной генерации файла.

Остальная часть шаблона состоит из управляющего блока. Этот блок устанавливает связь с базой данных `AdventureWorks` с помощью переменных, определенных в первом управляющем блоке. Затем он посылает запрос к базе данных, чтобы получить желаемые данные с помощью механизма чтения данных.

Для каждой записи, извлеченной из базы данных, создается текстовый блок. Этот блок состоит из двух блоков выражений, разделенных знаком равенства. Второе выражение добавляет идентификатор перечисления `Contact Type` в генерируемый файл, а первое вызывает вспомогательный метод `ValidIdentifier`, определенный в блоке элементов класса, создающем корректный идентификатор для каждого типа контакта, удаляя все неправильные символы из их имен.

Генерируемый файл приведен ниже. Окончательный результат выглядит довольно просто по сравнению со сценарием, на основе которого он был создан, но это обманчивое впечатление. Шаблон генератора T4 может оставаться неизменным при добавлении и удалении данных из таблицы `ContactType`. По существу, элементы этой базы данных могут быть полностью переупорядочены, а код будет компилироваться по-прежнему. С небольшими изменениями этот сценарий можно использовать даже для генерации перечислений по нескольким разным таблицам одновременно.



C#

```
// This code is generated. Please do not edit it directly
// If you need to make changes please edit ContactType.tt instead
namespace AdventureWorks {
  public enum ContactType {
    AccountingManager = 1,
    AssistantSalesAgent = 2,
    AssistantSalesRepresentative = 3,
    CoordinatorForeignMarkets = 4,
    ExportAdministrator = 5,
    InternationalMarketingManager = 6,
    MarketingAssistant = 7,
    MarketingManager = 8,
    MarketingRepresentative = 9,
    OrderAdministrator = 10,
    Owner = 11,
    OwnerMarketingAssistant = 12,
    ProductManager = 13,
    PurchasingAgent = 14,
    PurchasingManager = 15,
    RegionalAccountRepresentative = 16,
    SalesAgent = 17,
    SalesAssociate = 18,
    SalesManager = 19,
    SalesRepresentative = 20,
  }
}
```

Сниппет кода ContactTypes.generated.cs



VB

```
' This code is generated. Please do not edit it directly
' If you need to make changes please edit ContactType.tt instead
    Namespace AdventureWorks
  Enum ContactType
```

```
        AccountingManager = 1
        AssistantSalesAgent = 2
        AssistantSalesRepresentative = 3
        CoordinatorForeignMarkets = 4
        ExportAdministrator = 5
        InternationalMarketingManager = 6
        MarketingAssistant = 7
        MarketingManager = 8
        MarketingRepresentative = 9
        OrderAdministrator = 10
        Owner = 11
        OwnerMarketingAssistant = 12
        Generating Code Assets | 283
        ProductManager = 13
        PurchasingAgent = 14
        PurchasingManager = 15
        RegionalAccountRepresentative = 16
        SalesAgent = 17
        SalesAssociate = 18
        SalesManager = 19
        SalesRepresentative = 20
    End Enum
End Namespace
```

Сниппет кода ContactTypes.generated.vb

Генераторы Preprocessed Text Templates

Механизм Text Template Transformation представляет собой очень мощное средство и не должен ограничиваться только этапом проектирования. Система Visual Studio 2010 позволяет легко использовать преимущества генератора T4 для создания пользовательских текстовых шаблонов, предназначенных для использования в собственных проектах. Эти генераторы называются `Preprocessed Text Templates`.

Для того чтобы создать новый шаблон типа `Preprocessed Text Template`, необходимо открыть диалоговое окно Add New Item, выбрать закладку General и пункт Preprocessed Text Template в списке элементов. Вновь созданный файл будет иметь то же расширение `.tt`, что и обычные шаблонные файлы генератора T4, и содержать единственную директиву генератора T4.

C#

```
<#@ template language="C#" #>
```

VB

```
<#@ template language="VB" #>
```

Обратите внимание на то, что это не директива вывода. Генерируемый файл будет иметь то же самое имя, что и шаблонный файл, но расширение `.tt` будет заменено расширением `.vb` или `.cs` в зависимости от языка проекта. Когда этот файл сохраняется, он генерирует результирующий файл наподобие следующего.



C#

```
// ------------------------------------------------------------------------
// <auto-generated>
// This code was generated by a tool.
// Runtime Version: 10.0.0.0
//
// Changes to this file may cause incorrect behavior and will be lost if
```

```
// the code is regenerated.
// </auto-generated>
// ------------------------------------------------------------------------
namespace Chapter_14
{
    using System;

    public partial class NewTemplate
    {
      // region Fields
      // region Properties
      // region Transform-time helpers

     public virtual string TransformText()
     {
       return this.GenerationEnvironment.ToString();
     }
  }
}
```

Сниппет кода NewTemplate.cs



VB

```
Imports System
'------------------------------------------------------------------------
' <auto-generated>
' This code was generated by a tool.
' Runtime Version: 10.0.0.0
'
' Changes to this file may cause incorrect behavior and will be lost if
' the code is regenerated.
' </auto-generated>
'------------------------------------------------------------------------
Namespace My.Templates
  Partial Public Class NewTemplate
    ' Region "Fields"
    ' Region "Properties"
    ' Region "Transform-time helpers"
    Public Overridable Function TransformText() As String
      Return Me.GenerationEnvironment.ToString
    End Function
  End Class
End Namespace
```

Сниппет кода NewTemplate.vb

Он очень похож на промежуточный исходный файл, созданный генератором T4 по нормальному шаблону. Данный сгенерированный класс представляет собой обычный класс в проекте, т.е. его следует конкретизировать, заполнить его свойства и вызвать из него метод `TransformText()`.



Как и при работе с шаблоном `Text Template`, для генерации результирующего файла шаблона `Preprocessed Text Template` система Visual Studio использует надстройку `Custom Tool`. Вместо использования надстройки `TextTemplatingFileGenerator` шаблоны `Preprocessed Text Templates` трансформируются с помощью надстройки `TextTemplatingFilePreprocessor`, которая добавляет в проект класс генератора кода, а не результаты выполнения генератора кода.

Использование шаблона Preprocessed Text Templates

Для того чтобы продемонстрировать использование шаблона Preprocessed Text Template в пользовательском коде, рассмотрим простой сценарий. Проект должен посылать стандартное пригласительное письмо новым членам, когда они вступают в клуб AdventureWorks Cycle. Шаблон Preprocessed Text Template, приведенный ниже, содержит образец такого письма.



C#

```
<#@ template language="C#" #>
Dear <#=Member.Salutation#> <#=Member.Surname#> ,

  Welcome to our Bike Club!

Regards,
The AdventureWorks Team
<#= Member.DateJoined.ToShortDateString() #>
<#+ public ClubMember Member { get; set; } #>
```

Сниппет кода WelcomeLetter.tt



VB

```
<#@ template language="VB" #>
Dear <#=Member.Salutation#> <#=Member.Surname#> ,

  Welcome to our Bike Club!

Regards,
The AdventureWorks Team
<#= Member.DateJoined.ToShortDateString() #>
<#+ Public Member as ClubMember #>
```

Сниппет кода WelcomeLetter.tt

Этот файл генерирует класс WelcomeLetter и использует простой класс данных, который передается в шаблон через его свойство Member.



C#

```
public class ClubMember
{
  public string Salutation { get; set; }
  public string Surname { get; set; }
  public DateTime DateJoined { get; set; }
}
```

Сниппет кода ClubMember.tt



VB

```
Public Class ClubMember
    Public Surname As String
    Public Salutation As String
    Public DateJoined As Date
End Class
```

Сниппет кода ClubMember.tt

В заключение, для того чтобы создать письмо, пользователь должен создать объект класса WelcomeLetter, присвоить свойству Member объект ClubMember и вызвать метод TransformText().



C#

```
// ...
var member = new ClubMember
```

```
{
  Surname = "Fry",
  Salutation = "Mr",
  DateJoined = DateTime.Today
};

var letterGenerator = new WelcomeLetter();

letterGenerator.Member = member;

var letter = letterGenerator.TransformText();
// ...
```

Сниппет кода Program.cs



VB

```
' ...
Dim NewMember As New ClubMember
With NewMember
  .Surname = "Fry"
  .Salutation = "Mr"
  .DateJoined = Date.Today
End With

Dim LetterGenerator As New WelcomeLetter

LetterGenerator.Member = NewMember

Dim Letter = LetterGenerator.TransformText()
' ...
```

Сниппет кода Module1.vb

Это может показаться несколько неудобным, но класс `WelcomeLetter` является разделяемым, поэтому пользователь может изменить прикладной интерфейс в любой момент. Часто программисты делают конструкторы генератора закрытыми и предусматривают несколько статических методов для управления процессом создания и использования экземпляров генератора.



C#

```
public partial class WelcomeLetter
{
  private WelcomeLetter() { }
  public static string Create(ClubMember member)
  {
    return new WelcomeLetter { Member = member }.TransformText();
  }
}
```

Сниппет кода WelcomeLetter.Extensions.cs



VB

```
Namespace My.Templates
  Partial Public Class WelcomeLetter
    Private Sub New()
    End Sub

    Public Shared Function Create(ByVal Member As ClubMember) As String
      Dim LetterGenerator As New WelcomeLetter()
      LetterGenerator.Member = Member
      Return LetterGenerator.TransformText()
    End Function
```

```
    End Class
End Namespace
```

Сниппет кода WelcomeLetter.Extensions.vb



Генератор содержит класс `StringBuilder`, который используется для ввода данных при выполнении метода `TransformText`. Этот класс не исчезает при запуске метода `TransformText`, т.е. при каждом новом запуске его результаты добавляются к результатам предыдущего запуска. Вот почему метод `Create` каждый раз создает новый объект класса `WelcomeLetter`, а не хранит его как статическую переменную (`Shared`), допускающую повторное использование.

Различия между шаблонами

Отвлекаясь от того, какой именно аспект процесса генерации включен в пользовательский проект, отметим несколько важных различий между шаблоном `Preprocessed Text Template` и стандартным шаблоном генератора T4. Во-первых, шаблоны `Preprocessed Text Template` представляют собой полностью самостоятельные классы. Они не являются производными от какого-то базового класса, заданного по умолчанию, и, следовательно, не обращаются для своего выполнения к системе Visual Studio. Метод `TransformText()` класса генератора не выполняется в блоке `try/catch`, поэтому пользователь должен следить за ним и самостоятельно обрабатывать ошибки, возникающие при выполнении генератора.

Не все директивы и атрибуты генератора T4 имеют смысл в шаблоне `Preprocessed Text Template`. Опишем их кратко.

Директива `template` по-прежнему используется, но не все ее атрибуты имеют смысл. Атрибуты `culture` и `language` поддерживаются полностью. Атрибут `language` должен соответствовать языку проекта, иначе генератор класса не будет скомпилирован. Атрибут `debug` игнорируется, потому что пользователь может контролировать статус класса генератора, установив настройки проекта, как для любого другого класса.

Атрибут `inherits` поддерживается и показывает значительное влияние на генерируемый класс. Если пользователь не указал базовый класс, то генерируемый класс будет полностью самостоятельным и будет содержать реализации всех вспомогательных функций, такие как `Write` и `Error`. Если пользователь указал базовый класс, то эти реализации должны содержаться в базовом классе, а генерируемый класс будет вызывать их в процессе генерации.

Атрибут `hostspecific` поддерживается и генерирует свойство `Host` в классе генератора. Это свойство имеет тип `Microsoft.VisualStudio.TextTemplating.ITextTemplatingEngineHost` и находится в сборке `Microsoft.VisualStudio.TextTemplating`. Пользователь должен самостоятельно добавить ссылку на эту сборку в свой проект и приписать члену соответствующий тип перед вызовом метода `TransformText`.

Директива `import` работает как обычно. Соответствующее пространство имен включается в код генератора с помощью инструкций `using` в языке C# и инструкций `Import` в языке VB. Директива `include` также поддерживается полностью.

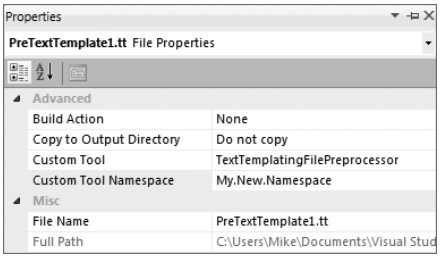


Рис. 14.6

Директивы `output` и `assembly` игнорируются. Для того чтобы добавить сборку в шаблон, пользователь должен просто добавить ссылку в проект, как обычно. Результирующий файл выбирается по имени шаблона и языку программирования. В заключение пользователь может задать пространство имен для класса генератора в окне Properties шаблонного файла, как показано на рис. 14.6. Это пространство имен обычно определяется по настройкам проекта, заданным по умолчанию, и местоположению шаблонного файла в структуре папок данного проекта.

Советы и трюки

Приведем несколько советов, которые помогут полностью использовать преимущества генератора T4.

- Сначала напишите код, который хотите генерировать, как обычный код на языке C# или VB. Как только все будет работать как следует, скопируйте весь код в файл с расширением `.tt`. Теперь понемногу начинайте делать код менее конкретным и более общим, вводя управляющие блоки, блоки выражений и блоки элементов класса.
- Часто сохраняйте измененные файлы. Как только изменение приведет к сбою, выполните откат и повторите попытку.
- Никогда не вносите изменения непосредственно в генерируемый файл. В следующий раз при сохранении шаблона эти изменения будут удалены.
- Делайте генерируемые классы разделяемыми (partial). Это обеспечит их расширяемость, позволит оставить некоторые части класса неизменными, а другие части класса будут генерироваться заново. По существу, это одна из причин, по которым существуют разделяемые классы.
- Используйте расширение, которое содержит слово *generated*, например `.generated.cs` и `.generated.vb`. Этого соглашения придерживается сама система Visual Studio. Кроме того, это отбивает охоту у других пользователей вносить изменения в шаблонные файлы.
- Включайте комментарии в начало генерируемого файла и указывайте в них, что файл был сгенерирован, а также описывайте, как изменить его содержимое и сгенерировать заново.
- Сделайте выполнение шаблона T4 частью вашего процесса сборки. Это гарантирует, что содержимое сгенерированных файлов не будет устаревать по отношению к метаданным, использованным при его генерации.
- Если у вас немного сущностей, зависящих от кода, сгенерированного с помощью обычного шаблона `Text Template`, переключите надстройку так, чтобы шаблон имел тип `Preprocessed Template`, пока вы разрабатываете программу. Это позволит включить в ваш проект генератор кода и даст вам возможность применить к нему модульное тестирование.

- Не используйте генератор T4 для генерации файлов с расширением `.tt`. Если вы попытаетесь использовать генератор кода для генерации шаблонных файлов, то уровень сложности существенно возрастет. В этом случае целесообразно рассмотреть другую стратегию для вашего проекта.
- В заключение отметим чрезвычайно полезный ресурс для любого пользователя, использующего генератор T4: `http://www.olegsych.com`. Его автор — Олег Сыч (Oleg Sych) — является одним из основных разработчиков системы Visual C#. Он поддерживает свой блог, содержащий очень большую коллекцию статей о генераторе T4.

Резюме

Генерация кода может обеспечить фантастическую производительность для вашего проекта, и система Visual Studio 2010 содержит несколько мощных готовых инструментов для управления этим процессом. В этой главы мы описали, как создать и использовать шаблоны генератора T4 для повышения скорости обычного и обобщенного кодирования. Знание того, когда и как применять генератор T4 в своих проектах, увеличивает производительность работы программиста и делает его решения более гибкими.

15

Шаблоны проектов и элементов управления

В ЭТОЙ ГЛАВЕ...

- Создание пользовательских шаблонов элементов
- Создание пользовательских шаблонов проектов
- Добавление мастера в пользовательские шаблоны проектов

Большинство коллективов разработчиков устанавливают стандарты, в соответствии с которыми они создают свои приложения. Это значит, что каждый раз, начиная новый проект или добавляя элемент в существующий проект, разработчик должен выполнить определенную процедуру, чтобы убедиться, что его работа соответствует стандарту. Система Visual Studio 2010 позволяет создавать шаблоны, которые можно использовать повторно без модификации шаблонов элементов, поставляемых вместе с системой Visual Studio 2010. В главе описывается, как создавать простые шаблоны, а затем расширять их с помощью мастера, который может изменять способ генерации проекта с помощью интерфейса `IWizard`.

Создание шаблонов

Существуют два типа шаблонов: те, которые создают новые элементы проекта, и те, которые создают целые проекты. Шаблоны обоих типов имеют практически одинаковую структуру, за исключением того, что они размещаются в разных папках шаблонов. Шаблоны проектов представлены в диалоговом окне `New Project`, а шаблоны элементов — в диалоговом окне `Add New Item`.

Шаблон элемента

Несмотря на то что шаблон элемента проекта можно создать вручную, намного быстрее использовать для этого существующий элемент проекта, а затем внести в не-

го требуемые изменения. Этот раздел начинается с изучения одного шаблона элемента — формы About, содержащей основную информацию, например номер версии приложения и имя автора.

Для начала создадим новое приложение типа Windows Forms (выбрав соответствующий язык программирования) с именем `StarterProject`. Вместо создания формы About с нуля можно настроить шаблон `About Box`, поставляемый вместе с системой Visual Studio. Щелкните правой кнопкой мыши на имени проекта `StarterProject`, выберите команду Add⇨New Item и добавьте новый шаблон `About Box`. Настройте форму About, заданную по умолчанию, удалив логотип и первый столбец элемента управления TableLayoutPanel (для этого необходимо выбрать панель макетов таблиц, перейти в окно Properties, выбрать свойство Columns, щелкнуть на кнопке с эллипсисом (...) и удалить столбец 1). Модифицированная форма About показана на рис. 15.1.

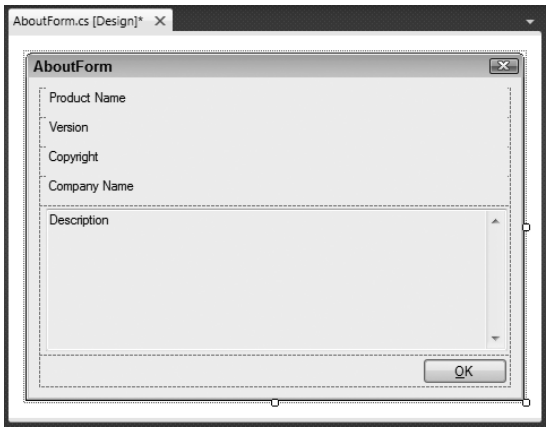


Рис. 15.1

Для того чтобы создать шаблон по форме About, выберите команду Export Template в меню File. Эта команда запускает мастер Export Template Wizard, как показано на рис. 15.2. Если изменения, внесенные в решение, еще не были сохранены, то система предложит сначала сохранить их. На первом этапе определяется тип шаблона, который необходимо создать. В данном случае следует выбрать переключатель Item Template и убедиться, что проект, в котором будет содержаться форма About, выбран в раскрывающемся списке.

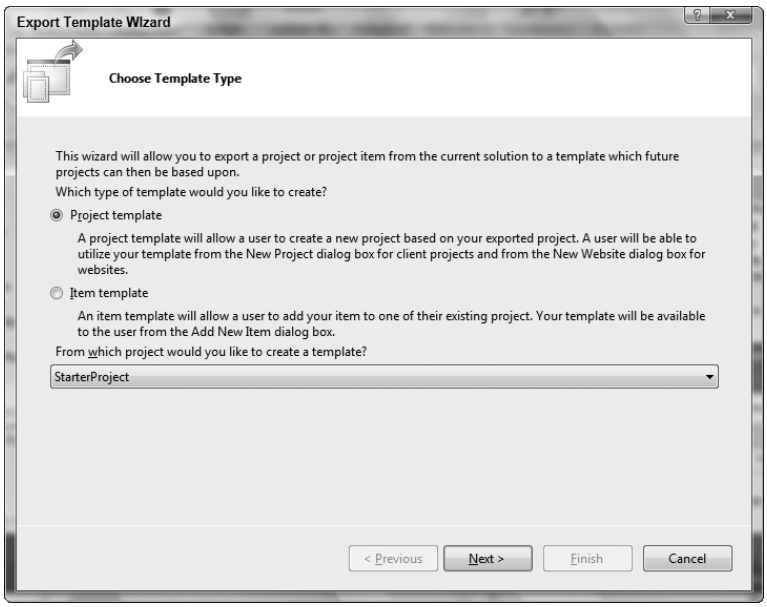


Рис. 15.2

Щелкните на кнопке Next. Система предложит выбрать элемент, на основе которого вы хотите создать свой шаблон. В данном случае следует выбрать форму About. Использование флажков на этом этапе выглядит довольно странно, поскольку пользователь все равно может выбрать только один элемент, на основе которого будет создан шаблон (выбор второго элемента отменяет выбор первого). Выбрав элемент, щелкните на кнопке Next. Диалоговое окно, показанное на рис. 15.3, позволяет включать любые ссылки на сборки, которые могут понадобиться.

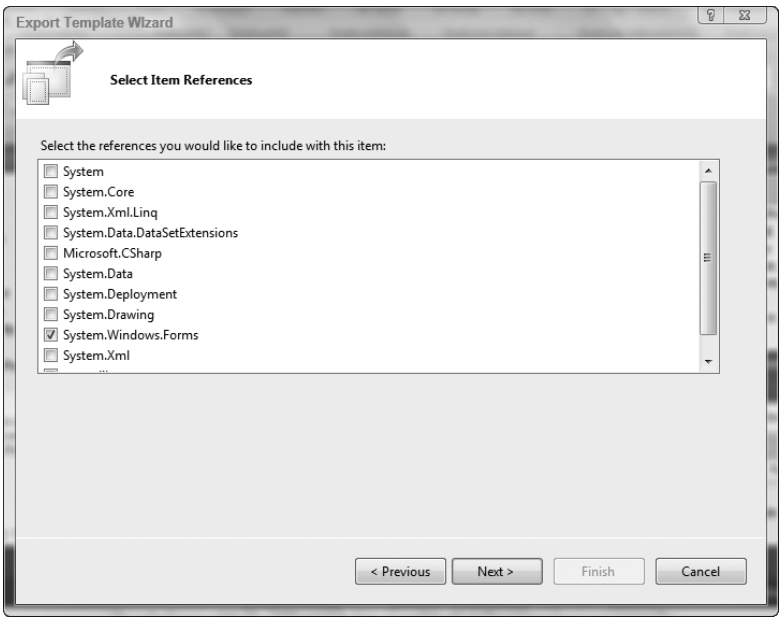


Рис. 15.3

Этот список основан на списке ссылок в проекте, которому принадлежит элемент. Поскольку в данном случае этим элементом является форма, необходимо включить ссылку на библиотеку `System.Windows.Forms`, которая будет добавлена в проект одновременно с новым элементов этого типа (если он еще не был добавлен). В противном случае вполне возможно, что проект не скомпилируется из-за отсутствия ссылки на эту сборку (проекты типа Class Library не имеют ссылки на эту сборку по умолчанию).



Выбрав сборку, пользователь может получить предупреждение, которое выводится под списком и сообщает, что выбранная сборка не была инсталлирована вместе с системой Visual Studio, и может предостерегать пользователя от использования своего шаблона, если эта сборка не доступна на его компьютере. Эту тонкость следует знать и выбирать только те сборки, которые необходимы для данного элемента.

На последнем этапе работы мастера Export Template Wizard необходимо задать некоторые свойства генерируемого шаблона, например имя, описание и пиктограмму, которая будет появляться в диалоговом окне Add New Item.

Последний этап работы мастера показан на рис. 15.4. Как видим, это диалоговое окно содержит два флажка: один для демонстрации папки результатов после завер-

шения работы, а второй — для автоматического импортирования нового шаблона в систему Visual Studio 2010.

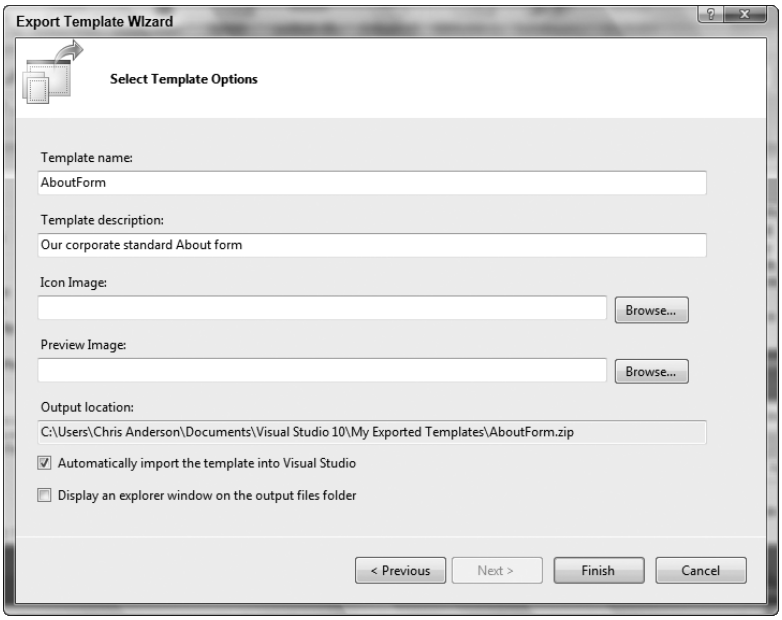


Рис. 15.4

По умолчанию экспортируемые шаблоны создаются в папке My Exported Templates в текущей пользовательской папке `Documents\Visual Studio 2010`. В этой папке существует множество других папок, содержащих настройки пользователя, касающиеся системы Visual Studio 2010 (рис. 15.5).

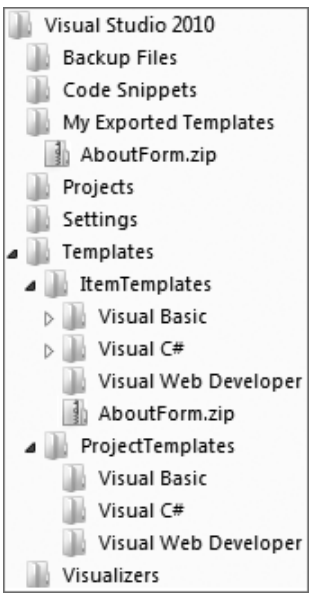


Рис. 15.5

Обратите внимание на папку Templates, показанную на рис. 15.5. Система Visual Studio 2010 ищет в этой папке дополнительные шаблоны, чтобы продемонстрировать их пользователю при создании новых элементов. Две папки под папкой Templates содержат шаблоны элементов и шаблоны проектов соответственно. В свою очередь они разделяются на более мелкие папки в соответствии с языками программирования. Если установить флажок Automatically Import the Template into Visual Studio на последней странице мастера Export Template Wizard, то новый шаблон будет не только размещен в папке результатов, но и скопирован в соответствующее место (в зависимости от языка и типа шаблона) в папке Templates. В следующий раз система Visual Studio 2010 автоматически выведет этот шаблонный элемент в диалоговом окне Add New Item, как показано на рис. 15.6

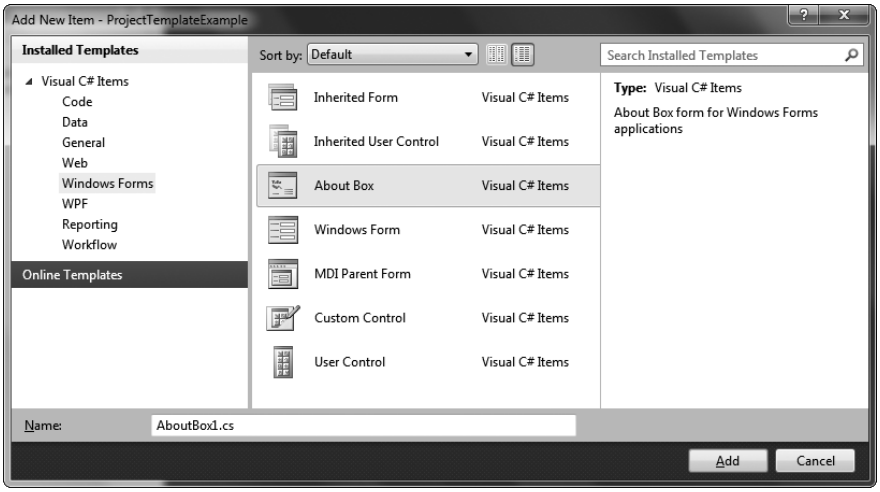


Рис. 15.6



Если нужно, чтобы шаблон элемента или проекта появился в существующей категории (системной или пользовательской) в окне Add New Item/New Project (например, в категории Windows Forms), достаточно создать папку с этим именем и поместить в нее шаблон (в соответствующее место, отведенное этому шаблону). В следующий раз при открытии окна Add New Item/New Project шаблон появится в категории, соответствующей имени папки (или в виде новой категории, если для нее нет соответствующей папки).

Шаблон проекта

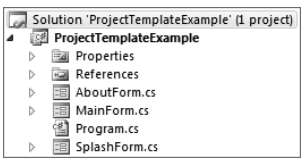


Рис. 15.7

Шаблон проекта создается точно так же, как и шаблон элемента, за одним исключением. В то время как шаблон элемента основывается на существующем элементе, шаблон проекта должен быть основан на целом проекте. Например, может существовать простой проект `ProjectTemplateExample` (рис. 15.7), имеющий главную форму, форму About и заставку.

Для того чтобы сгенерировать шаблон на основе такого проекта, необходимо выполнить те же действия, что и при генерации шаблона элемента, за исключением того, что пользователь должен выбрать команду Project Template в ответ на запрос, какой тип шаблона следует генерировать, и теперь нет этапа, на котором выбираются включаемые элементы (все элементы, находящиеся в проекте, включаются в шаблон безусловно). После завершения работы мастера Export Template Wizard в окне New Project, показанном на рис.15.8, появляется новый шаблон проекта.

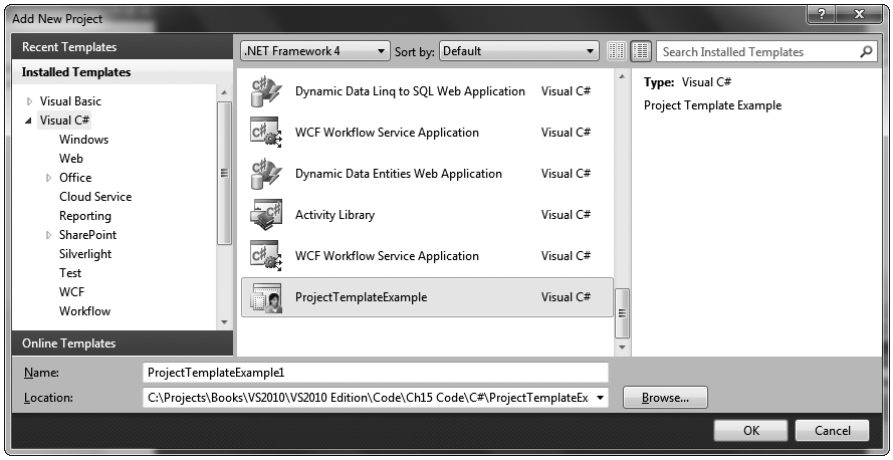


Рис. 15.8

Структура шаблона

Прежде чем рассмотреть процедуру создания более сложных шаблонов, необходимо понять, что именно создается мастером Export Template Wizard. Если заглянуть в папку My Exported Templates, мы увидим, что все шаблоны экспортируются в виде единственного сжатого файла с расширением `.zip`. Этот файл может содержать любое количество файлов или папок, в зависимости от того, создаются ли шаблоны для отдельных файлов или для целых проектов. Однако для всех шаблонных zip-файлов характерно то, что они содержат файл с расширением `.vstemplate`. Этот файл представляет собой XML-документ, в котором хранится конфигурация шаблона. Ниже приведен листинг файла с расширением `.vstemplate`, экспортированный ранее как часть шаблона проекта.

```
<VSTemplate Version="2.0.0"
 xmlns="http://schemas.microsoft.com/developer/vstemplate/2005" Type="Project">
  <TemplateData>
    <Name> Project Template Example </Name>
    <Description> Project Template Example </Description>
    <ProjectType> CSharp </ProjectType>
    <ProjectSubType>
    </ProjectSubType>
    <SortOrder> 1000 </SortOrder>
    <CreateNewFolder> true </CreateNewFolder>
    <DefaultName> Project Template Example </DefaultName>
    <ProvideDefaultName> true </ProvideDefaultName>
    <LocationField> Enabled </LocationField>
    <EnableLocationBrowseButton> true </EnableLocationBrowseButton>
    <Icon> __TemplateIcon.ico </Icon>
</TemplateData>
<TemplateContent>
 <Project TargetFileName="ProjectTemplateExample.csproj"
   File="ProjectTemplateExample.csproj" ReplaceParameters="true">
   <ProjectItem ReplaceParameters="true" TargetFileName="AboutForm.cs">
     AboutForm.cs </ProjectItem>
   <ProjectItem ReplaceParameters="true" TargetFileName="AboutForm.Designer.cs">
     AboutForm.Designer.cs </ProjectItem>
   <ProjectItem ReplaceParameters="true" TargetFileName="AboutForm.resx">
     AboutForm.resx </ProjectItem>
```

```
    <ProjectItem ReplaceParameters="true" TargetFileName="MainForm.cs">
      MainForm.cs </ProjectItem>
    <ProjectItem ReplaceParameters="true" TargetFileName="MainForm.Designer.cs">
      MainForm.Designer.cs </ProjectItem>
    <ProjectItem ReplaceParameters="true" ="Program.cs"> Program.cs </ProjectItem>
    <Folder Name="Properties" TargetFolderName="Properties">
      <ProjectItem ReplaceParameters="true"
        TargetFileName="AssemblyInfo.cs"> AssemblyInfo.cs </ProjectItem>
      <ProjectItem ReplaceParameters="true"
        TargetFileName="Resources.resx"> Resources.resx </ProjectItem>
      <ProjectItem ReplaceParameters="true"
        TargetFileName="Resources.Designer.cs"> Resources.Designer.cs
      </ProjectItem>
      <ProjectItem ReplaceParameters="true"
         TargetFileName="Settings.settings"> Settings.settings </ProjectItem>
      <ProjectItem ReplaceParameters="true"
         TargetFileName="Settings.Designer.cs"> Settings.Designer.cs
      </ProjectItem>
    </Folder>
    <ProjectItem ReplaceParameters="true" TargetFileName="SplashForm.cs">
      SplashForm.cs </ProjectItem>
    <ProjectItem ReplaceParameters="true"
      TargetFileName="SplashForm.Designer.cs">
      SplashForm.Designer.cs </ProjectItem>
    <ProjectItem ReplaceParameters="true" TargetFileName="SplashForm.resx">
      SplashForm.resx </ProjectItem>
  </Project>
</TemplateContent>
```

В начале этого файла узел `VSTemplate` содержит атрибут `Type`, который указывает, что создается: шаблон элемента (`Item`), шаблон проекта (`Project`) или шаблон нескольких проектов (`ProjectGroup`). Остальная часть файла состоит из блоков `TemplateData` и `TemplateContent`. Блок `TemplateData` содержит информацию о самом шаблоне, например его имя, описание и пиктограмму, которая будет использована для его представления в окне New Project, а блок `TemplateContent` определяет файловую структуру шаблона.

В предыдущем примере содержимое файла начиналось с узла `Project`. Это значит, что используется файл проекта. Файлы, содержащиеся в этом шаблоне, перечислены с помощью узлов `ProjectItem`. Каждый узел содержит атрибут `TargetFileName`, который можно использовать для указания имени файла, который появится в проекте, создаваемом на основе данного шаблона. Если создается шаблон элемента, то узел `Project` отсутствует, а в узле `TemplateContent` содержатся узлы `ProjectItem`.



Можно создать шаблоны для решения, содержащего несколько проектов. Эти шаблоны содержат отдельный файл с расширением `.vstemplate` для каждого проекта в решении. Кроме того, эти шаблоны содержат глобальный файл с расширением `.vstemplate`, который описывает весь шаблон и содержит ссылки на другие файлы с расширением `.vstemplate`, существующие в каждом проекте. Однако этот файл создается вручную, поскольку система Visual Studio в настоящее время не имеет функции для экспорта шаблона решения.

Более подробную информацию о файле с расширением `.vstemplate` можно найти в файле `%programfiles%\Microsoft Visual Studio 10.0\Xml\Schemas\1033\vstemplate.xsd`.

Параметры шаблона

Как шаблоны элементов, так и шаблоны проектов поддерживают подстановку параметров, позволяющую заменять ключевые параметры при создании проекта или элемента на основе шаблона. В некоторых случаях они подставляются автоматически. Например, когда форма About экспортировалась как шаблон элемента, имя класса было удалено и заменено шаблонным параметром, как показано ниже.

```
Public Class $safeitemname$v
```

В табл. 15.1 описаны четырнадцать зарезервированных параметров шаблонов, которые можно использовать в любом проекте.

Таблица 15.1. Параметры шаблонов

Параметр	Описание
clrversion	Текущая версия общеязыковой выполняющейся среды
GUID[1-10]	Идентификатор GUID, использованный для замены проектного идентификатора GUID в файле проекта. Пользователь может задать до десяти уникальных идентификаторов GUID (например, GUID1, GUID2 и т.д.)
itemname	Имя, заданное пользователем в диалоговом окне Add New Item
machinename	Текущее имя компьютера (например, computer01)
projectname	Имя, указанное пользователем в диалоговом окне New Project
registeredorganization	Ключи и значения из регистра, в которых хранится зарегистрированное название организации
rootnamespace	Базовое пространство имен текущего проекта. Этот параметр используется для замены пространства имен в элементе, добавляемом в проект
safeitemname	Имя, заданное пользователем в диалоговом окне Add New Item, из которого удалены все небезопасные символы и пробелы
safeprojectname	Имя, заданное пользователем в диалоговом окне New Project, из которого удалены все небезопасные символы и пробелы
time	Текущее время на локальном компьютере
userdomain	Текущий домен пользователя
username	Текущее имя пользователя
webnamespace	Имя текущего веб-сервера. Используется в любом шаблоне веб-формы для обеспечения уникальности имен классов
year	Текущий год в формате YYYY

Кроме зарезервированных параметров, пользователь может создать собственные шаблонные параметры. Для этого достаточно добавить раздел <CustomParameters> в файл с расширением .vstemplate, как показано ниже.

```
<TemplateContent>
   ...
   <CustomParameters>
        <CustomParameter Name="$timezoneName $" Value="(GMT+8:00) Perth"/>
        <CustomParameter Name="$timezoneOffset $" Value="+8"/>
   </CustomParameters>
</TemplateContent>
```

Ссылаться на эти шаблонные параметры в своем коде можно следующим образом.

```
string tzName = "$timezoneName$";
string tzOffset = "$timezoneOffset$";
```

Когда на основе шаблона создается новый элемент или проект, содержащий пользовательский параметр, система Visual Studio автоматически выполняет шаблонную подстановку как для пользовательских, так и для зарезервированных параметров.

Расположение шаблона

По умолчанию шаблоны элементов и проектов пользователя хранятся в персональной папке `Documents\Visual Studio 2010\Templates`, но он может задать другое место для их хранения (например, общий каталог в сети, чтоб сделать его доступным своим коллегам), используя диалоговое окно Options. Для того следует выбрать команду Tools⇨Options и узел Projects and Solutions. Затем пользователь может задать другое место для хранения своих шаблонов.

Расширение шаблонов

Создание шаблонов на основе существующих элементов и проектов ограничивает возможности пользователя. Это предполагает, что каждый проект или сценарий должен состоять из одних и тех же элементов. Вместо создания нескольких шаблонов для каждого отдельного сценария (например, один шаблон может иметь главную форму с черным фоном, а другой – с белым) пользователь может настроить несколько шаблонов на основе одного сценария. По этой причине в данном разделе рассматривается шаблон проекта, созданный ранее, и настраивается так, чтобы пользователи могли самостоятельно задавать цвет фона. Кроме того, мы создадим программу установки для шаблона и мастера, с помощью которого можно взаимодействовать с пользователем.

Для того чтобы включить в шаблон взаимодействие с пользователем, необходимо реализовать интерфейс `IWizard` в библиотеке классов, а затем подписать его и разместить в глобальном кэше сборок (Global Assembly Cache – GAC) на компьютере, на котором будет выполняться шаблон. По этой причине для разворачивания шаблона, использующего мастер, пользователь должен иметь права на развертывание сборки мастера в кэше GAC.

Установка шаблона проекта

Перед реализацией интерфейса `IWizard` следует выполнить этапы установки решения, чтобы все фрагменты находились в одном и том же месте. Это облегчит внесение изменений, сборку и выполнение программы инсталляции.

1. Создайте новый проект на основе шаблона проекта Project Template Example, созданного ранее в этой главе, и назовите его `ExtendedProjectTemplate-Example`. Убедитесь, что решение собрано и правильно выполняется. На более поздних этапах будет намного труднее обнаружить ошибки, потому что сообщения об ошибках, которые будут появляться при использовании шаблона, сложно понять.
2. Добавьте в это решение проект типа Class Library, назвав его `WizardClass-Library`. В этот проект мы включим реализацию интерфейса `IWizard`.

3. Добавьте в проект `WizardClassLibrary` новый пустой файл классов `MyWizard` и пустую форму типа Windows Form с именем `ColorPickerForm`. Этот файл и форму мы настроим позднее.
4. Для доступа к интерфейсу `IWizard` добавьте в проект типа Class Library ссылки на библиотеки `EnvDTE.dll` и `Microsoft.VisualStudio.TemplateWizardInterface.dll`, расположенные по адресу `%programfiles%\Microsoft Visual Studio 10.0\Common7\IDE\PublicAssemblies\`.
5. В заключение необходимо добавить в решение проект `Setup`. Для этого выберите команду File⇨Add⇨New Project, откройте категорию Other Project Types и выберите команду Setup and Deployment. Выберите шаблон Setup Wizard и примите приглашение включить файлы типа Primary Output и Content Files из библиотеки `WizardClassLibrary`.

В результате получим решение, показанное на рис. 15.9.

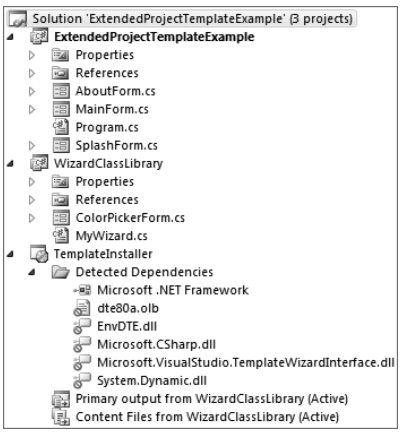


Рис. 15.9

Как показано на рис. 15.9, при включении в инсталляционную программу файлов для вывода *первичной информации* (primary output) и *содержимого* (content) из проекта типа Class Library в ней возникает большое количество зависимостей. Поскольку шаблон будет использоваться только на компьютере с системой Visual Studio 2010, ни одна из этих зависимостей нам не нужна. Их следует исключить, выбрав в контекстном меню команду Exclude. Теперь выполните следующие этапы, чтобы завершить конфигурацию проекта `Installer`.

1. При включении в инсталляционную программу файлов для вывода первичной информации (primary output) и содержания (content) из проекта они добавляются в папку `Application`. Однако первичная информация из библиотеки классов должна размещаться в кэше GAC, а файлы содержимого должны поступать в пользовательскую папку `Templates`. Для перемещения этих файлов необходимо щелкнуть правой кнопкой мыши на проекте Installer и выбрать команду View⇨File System из контекстного меню, чтобы открыть окно File System.
2. По умолчанию окно File System содержит папки `Application` (которую невозможно удалить), `User's Desktop` и `User's Programs Menu`. Удалите две последние пользовательские папки, выбрав команду Delete из контекстного меню.
3. Добавьте папки `Global Assembly Cache (GAC)` и `User's Personal Data` в файловую систему, щелкнув правой кнопкой мыши на узле File System on Target Machine и выбрав эти папки в списке.
4. Добавьте в папку `User's Personal Data` папку `Visual Studio 2010` (щелкните правой кнопкой мыши и выберите команду Add⇨Folder), затем папки `Templates`, `ProjectTemplates` и папку `Visual C#` (если проект создается на языке C#) или `Visual Basic` (если проект создается на языке VB). Результат будет выглядеть так, как показано на рис. 15.10.

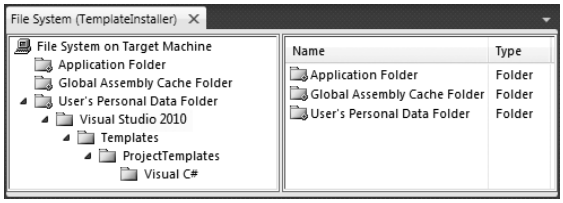


Рис. 15.10

5. Для завершения инсталляционной программы переместите файлы первичной информации из папки `Application` в папку `Global Assembly Cache`, а затем переместите файлы содержимого из папки `Application` в папку `ProjectTemplates`. (Для этого достаточно перетащить эти файлы в окно File System.)

Интерфейс IWizard

Итак, мы закончили программу инсталляции и начинаем работать над библиотекой классов мастера. Как показано на рис. 15.9, у нас есть форма (`ColorPickerForm`) и класс (`MyWizard`). Форма имеет простой вид и предназначена для задания цвета фона в главной форме. В эту форму необходимо добавить элемент управления Color Dialog с именем `ColorDialog1`, элемент управления Panel с именем `ColorPanel`, элемент управления `Button` с именем `PickColorButton` (и надписью "`Pick Color`"), а также элемент управления Button с именем `AcceptColorButton` (и надписью "`Accept Color`").

Вместо пиктограммы, которую система Visual Studio по умолчанию использует для форм, мы можем выбрать более подходящую форму из библиотеки `Image Library` системы Visual Studio 2010. Эта библиотека представляет собой коллекцию стандартных пиктограмм, образов и мультфильмов, используемых в программном обеспечении Windows, Office и других программных продуктах компании Microsoft. Пользователь может бесплатно использовать эти образы, обеспечивая тем самым совместимость с программным обеспечением компании Microsoft.

Библиотека `Image Library` инсталлируется в систему Visual Studio как сжатый файл `VS2010ImageLibrary.zip`. По умолчанию его можно найти в папке `%programfiles%\Microsoft Visual Studio 10.0\Common7\VS2010ImageLibrary\1033\` и извлечь содержимое архива в более удобное место, например в каталог своего профиля.

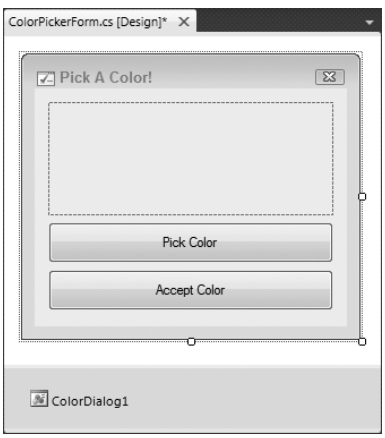


Рис. 15.11

Для того чтобы заменить пиктограмму на форме, сначала перейдите к окну Properties и выберите элемент Form в раскрывающемся списке. Щелкните на эллипсисе (`...`) свойства Icon, чтобы загрузить диалоговое окно для выбора файла. Выберите файл с пиктограммой, которую хотите использовать, и щелкните на кнопке OK (в данном примере мы выбрали файл `VS2010ImageLibrary\Objects\ico_format\WinVista\Settings.ico`).

После завершения форма `ColorPickerForm` должна выглядеть так, как показано на рис. 15.11.

В эту форму можно добавить код, приведенный ниже. Логика этой формы реализуется при обработке события, связанного с кнопкой Pick Color, которая открывает диалоговое окно ColorDialog, предназначенное для выбора цвета.

VB

```
Public Class ColorPickerForm
    Public ReadOnly Property SelectedColor() As Drawing.Color
        Get
             Return ColorPanel.BackColor
        End Get
End Property

Private Sub PickColorButton_Click(ByVal sender As System.Object, _
                                  ByVal e As System.EventArgs) Handles
                                  PickColorButton.Click
    ColorDialog1.Color = ColorPanel.BackColor
    If ColorDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
        ColorPanel.BackColor = ColorDialog1.Color
    End If
End Sub

Private Sub AcceptColorButton_Click(ByVal sender As System.Object, _
                                    ByVal e As System.EventArgs) Handles
                                    AcceptColorButton.Click
    Me.DialogResult = Windows.Forms.DialogResult.OK
    Me.Close()
End Sub
End Class
```

C#

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

namespace WizardClassLibrary
{
    public partial class ColorPickerForm : Form
    {
        public ColorPickerForm()
        {
            InitializeComponent();

            PickColorButton.Click += PickColorButton_Click;
            AcceptColorButton.Click += AcceptColorButton_Click;
        }

    public Color SelectedColor
    {
        get { return ColorPanel.BackColor; }
    }

    private void PickColorButton_Click(object sender, EventArgs e)
    {
        ColorDialog1.Color = ColorPanel.BackColor;

        if (ColorDialog1.ShowDialog() == DialogResult.OK)
        {
            ColorPanel.BackColor = ColorDialog1.Color;
        }
    }
```

```
    private void AcceptColorButton_Click(object sender, EventArgs e)
    {
        this.DialogResult = DialogResult.OK;
        this.Close();
    }
  }
}
```

Класс `MyWizard` реализует интерфейс `IWizard`, который предоставляет массу возможностей для взаимодействия с пользователем в шаблонном процессе. Добавьте код в метод `RunStarted`, который будет вызываться сразу после начала процесса создания проекта. Этот код обеспечивает отличную возможность для выбора и применения нового цвета фона главной формы.

VB

```
Imports Microsoft.VisualStudio.TemplateWizard
Imports System.Collections.Generic
Imports System.Windows.Forms

Public Class MyWizard
    Implements IWizard

    Public Sub BeforeOpeningFile(ByVal projectItem As EnvDTE.ProjectItem) _
                                 Implements IWizard.BeforeOpeningFile

    End Sub

    Public Sub ProjectFinishedGenerating(ByVal project As EnvDTE.Project) _
                                  Implements IWizard.ProjectFinishedGenerating
    End Sub

    Public Sub ProjectItemFinishedGenerating _
                                    (ByVal projectItem As EnvDTE.ProjectItem) _
                               Implements IWizard.ProjectItemFinishedGenerating
    End Sub

    Public Sub RunFinished() Implements IWizard.RunFinished

    End Sub

    Public Sub RunStarted(ByVal automationObject As Object, _
                          ByVal replacementsDictionary As _
    Dictionary(Of String, String), _
                          ByVal runKind As WizardRunKind, _
                          ByVal customParams() As Object) _
    Implements IWizard.RunStarted
        Dim selector As New ColorPickerForm
        If selector.ShowDialog = DialogResult.OK Then
            Dim c As Drawing.Color = selector.SelectedColor
            Dim colorString As String = "System.Drawing.Color.FromArgb(" & _
    c.R.ToString & "," & _
    c.G.ToString & "," & _
    c.B.ToString & ")"
             replacementsDictionary.Add _
                           ("Me.BackColor = System.Drawing.Color.Silver", _
                            "Me.BackColor = " & colorString)
        End If
    End Sub
```

```
    Public Function ShouldAddProjectItem(ByVal filePath As String) As Boolean _
                                        Implements IWizard.ShouldAddProjectItem
        Return True
    End Function
End Class
```

C#

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Collections.Generic;
using Microsoft.VisualStudio.TemplateWizard;

namespace WizardClassLibrary
{
    public class MyWizard : IWizard
    {
         public void BeforeOpeningFile(EnvDTE.ProjectItem projectItem)
         {
         }

         public void ProjectFinishedGenerating(EnvDTE.Project project)
         {
         }

         public void ProjectItemFinishedGenerating(EnvDTE.ProjectItem
projectItem)
         {
         }

          public void RunFinished()
          {
          }

          public void RunStarted(object automationObject, Dictionary
          <string, string>
              replacementsDictionary, WizardRunKind runKind, object[]
              customParams)
          {
              ColorPickerForm selector = new ColorPickerForm();

              if (selector.ShowDialog() == DialogResult.OK)
              {
                   Color c = selector.SelectedColor;
                   string colorString = "Color.FromArgb(" +
                         c.R.ToString() + "," +
                         c.G.ToString() + "," +
                         c.B.ToString() + ")";
                   replacementsDictionary.Add
                                      ("this.BackColor =
                                       System.Drawing.Color.Silver",
                                       "this.BackColor = " + colorString);
              }
          }

          public bool ShouldAddProjectItem(string filePath)
          {
              return true;
          }
    }
}
```

В методе `RunStarted` пользователю предлагается выбрать новый цвет, затем на основе его ответа в словарь замен добавляется новый элемент. В данном случае заменяем `"Me.BackColor = System.Drawing.Color.Silver"` (VB) или `"this.BackColor = System.Drawing.Color.Silver"` (C#) конкатенированной строкой, состоящей из значений RGB для цвета, указанного пользователем. Словарь замен используется при создании файлов для нового проекта, потому что именно в них будет производиться поиск ключей замены. Как только будут найдены какие-нибудь экземпляры этих ключей, они будут заменены соответствующими значениями замены. В данном случае мы ищем строку, в которой указано, что значение `BackColor` равно `Silver`, и заменяем это значение новым цветом, указанным пользователем.

Библиотека классов, содержащая реализацию интерфейса `Iwizard`, должна содержать строго именованные сборки, которые можно хранить в кэше GAC. Для того чтобы гарантировать это, необходимо сгенерировать новый ключ подписи с помощью закладки Signing в диалоговом окне Project Properties, как показано на рис. 15.12.

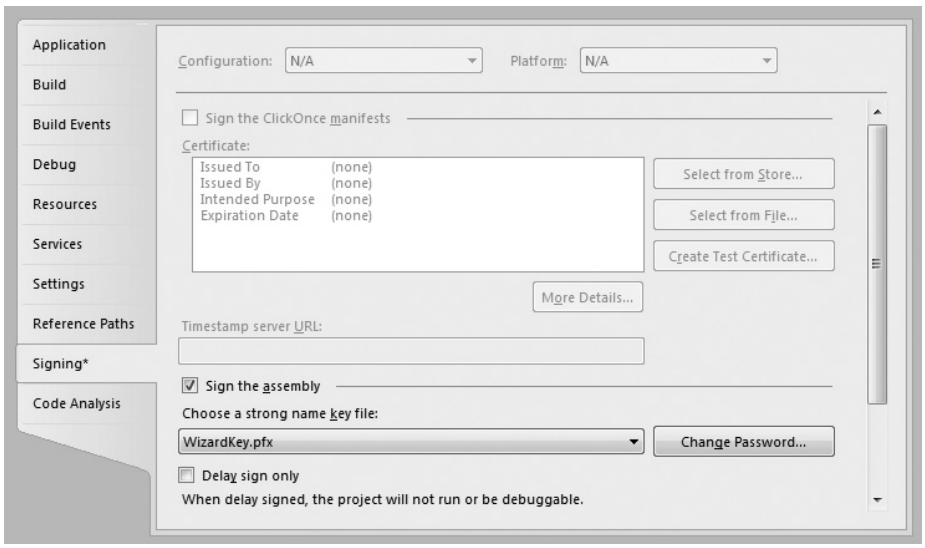


Рис. 15.12

Если установить флажок Sign the Assembly, то файл ключей не будет задан по умолчанию. Для того чтобы создать новый ключ, необходимо выбрать команду `<New ...>` из раскрывающегося списка. В качестве альтернативы можно использовать существующий файл ключей, используя элемент `<Browse ...>` в раскрывающемся списке.

Генерация шаблона Extended Project Template

Шаблон в рассматриваемом примере основан на проекте `ExtendedProjectTemplateExample`. Нам нужно лишь внести небольшие изменения, чтобы созданный нами мастер работал правильно. В предыдущем разделе мы добавили новый элемент в словарь замен. Этот словарь производит поиск экземпляров, в которых свойство `BackColor` имеет значение `Silver`. Если форма `MainForm` должна иметь свойство `BackColor`, заданное с помощью мастера, необходимо гарантировать, чтобы эта за-

мена была найдена. Для этого достаточно установить значение свойства BackColor формы MainForm равным Silver. В результате в файл MainForm.Designer.vb (VB) будет добавлена строка "Me.BackColor = System.Drawing.Color.Silver", а в файл MainForm.Designer.cs — строка "this.BackColor = System.Drawing.Color.Silver", так что эти файлы будут найдены на этапе замены.

Теперь необходимо связать мастер с шаблоном проекта, чтобы мастер вызывался при создании нового проекта по этому шаблону. К сожалению, вначале это придется делать вручную, но при последующих сборках проекта этот процесс можно автоматизировать. Начнем с экспорта проекта ExtendedProjectTemplateExample как нового шаблона проекта в соответствии с ранее приведенными инструкциями. Найдите файл .zip для этого шаблона в окне Windows Explorer и распакуйте его. Найдите файл с расширением .vstemplate и файл пиктограмм и поместите их в папку, содержащую проект ExtendedProjectTemplateExample. Другие файлы из распакованного шаблона можно игнорировать — это просто те же самые файлы из папки проекта, которые будут использоваться для вывода результатов работы шаблона, поэтому теперь в нашем распоряжении есть все необходимые файлы. Следует убедиться, что вы *не* включили их в сам проект ExtendedProjectTemplateExample; они должны появляться отдельно, как показано на рис. 15.13.

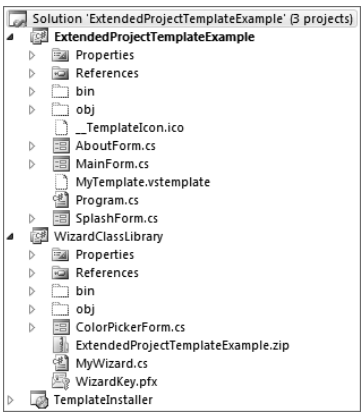


Рис. 15.13

Обратите внимание на то, что файл с расширением .zip в проекте WizardClassLibrary — это шаблонный файл, экспортированный системой Visual Studio (и который при настройке проекта должен компилироваться). Пока возьмите шаблон проекта .zip, который создан системой Visual Studio, и скопируйте его в папку проекта WizardClassLibrary. Отобразите все файлы проекта (как на рис. 15.13), щелкните правой кнопкой мыши на файле и выберите команду Include In Project. В окне Properties установите для свойства Build Action значение Content. Это необходимо для программы инсталляции, созданной вами ранее. В результате файлы типа Content из библиотеки классов будут включены в файл настройки и будут находиться в папке Templates как часть процесса инсталляции.

Для переключения мастера при создании проекта на основе данного шаблона добавьте в файл MyTemplate.vstemplate несколько строк.

```
<VSTemplate Version="2.0.0"
xmlns="http://schemas.microsoft.com/developer/vstemplate/2005" Type="Project">
  <TemplateData>
  ...
  </TemplateData>
  <TemplateContent>
   ...
  </TemplateContent>
  <WizardExtension>
    <Assembly>WizardClassLibrary, Version=1.0.0.0, Culture=neutral,
       PublicKeyToken=022e960e5582ca43, Custom=null</Assembly>
     <FullClassName>WizardClassLibrary.MyWizard</FullClassName>
   </WizardExtension>
</VSTemplate>
```

Узел <WizardExtension>, добавленный в этот пример, задает имя класса мастера и строгое имя сборки, которой он принадлежит. Мы уже подписали сборку мастера, поэтому осталось лишь определить свойство PublicKeyToken, открыв сборку с помощью утилиты Reflector for .NET, разработанную Лутцем Редером (Lutz Roeder's) (см. http://www.red-gate.com/products/reflector/). Если проект WizardLibrary еще не собран, то это необходимо сделать с помощью утилиты Reflector. Открыв созданную сборку с помощью утилиты Reflector (выбрав команду File⇨Open, а затем сборку), можно увидеть значение PublicKeyToken данной сборки, выбрав его в дереве, как показано на рис. 15.14. Значение PublicKeyToken в файле с расширением .vstemplate необходимо заменить значением, найденным с помощью утилиты Reflector.

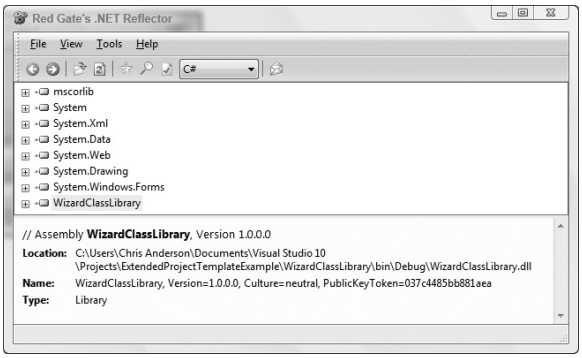


Рис. 15.14

Осталось только добавить в проект ExtendedProjectTemplateExample событие, *генерируемое после сборки* (post-build event command), которое будет архивировать проект в шаблон проекта с расширением .zip (данный пример использует утилиту 7-zip, доступную на сайте www.7-zip.org, но вместо нее можно использовать любой архиватор, запускаемый из командной строки). Мы сделаем вызов утилиты 7-zip выполняемым. Она будет архивировать содержимое папки ExtendedProjectTemplateExample (рекурсивно, но исключая папки, содержащие бинарные и объектные файлы) в файл ExtendedProjectTemplateExample.zip и записывать его в папку WizardClassLibrary. Обратите внимание на то, что путь к архиватору, возможно, придется изменить, поэтому следует включить в код следующую команду (в одной строке) в качестве события, генерируемого после сборки.

```
"C:\Program Files\7-Zip\7z.exe" a -tzip ..\..\..\WizardClassLibrary\
ExtendedProjectTemplateExample.zip ..\..\*.* -r -x!bin -x!obj
```

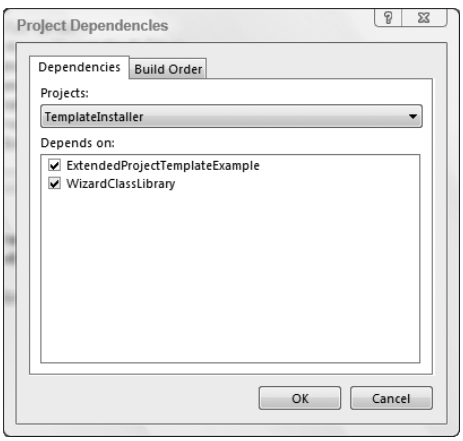


Рис. 15.15

Итак, мы завершили отдельные проекты, необходимые для создания шаблона проекта (ExtendedProjectTemplateExample), добавили мастер для модификации проекта в процессе его создания (WizardClassLibrary) и собрали проект инсталляции для развертывания шаблона на других компьютерах. На последнем этапе необходимо исправить список зависимостей решения, чтобы проект ExtendedProjectTemplateExample мог быть собран заново (а значит, шаблонный архивный файл был создан заново) до сборки проекта инсталляции. Поскольку между проектами Installer и ExtendedProjectTemplateExample нет прямой зависимости, необходимо открыть свойства решения и указать зависимость, как показано на рис. 15.15.

Наше решение закончено и может использоваться для инсталляции проекта `ExtendedProjectTemplateExample` и связанной с ним реализации интерфейса `IWizard`. После инсталляции решения можно создавать новые проекты на основе проекта `ExtendedProjectTemplateExample`.

Комплект Starter Kits

Комплект Starter Kit по существу аналогичен шаблону, но с точки зрения предназначения несколько отличается от него. В то время как шаблоны проектов создают лишь основную оболочку приложения, комплект Starter Kits создает простое приложение целиком вместе с документацией о ее настройках. Комплект Starter Kits появляется в окне New Project точно так же, как шаблоны проектов. Комплект Starter Kits может оказать большую помощь в начале работы над проектом (когда выбираются его общие контуры), причем пользователь может создавать проекты, которые можно использовать совместно с другими коллегами, точно так же, как и шаблоны проектов, описанные выше.

Интерактивные шаблоны

Система Visual Studio 2010 прекрасно интегрируется с коллекцией Visual Studio Gallery (`http://www.visualstudiogallery.com`), позволяющей найти шаблоны, созданные и загруженные другими разработчиками. Пользователь может просматривать эту коллекцию и инсталлировать выбранные шаблоны в системе Visual Studio двумя способами: с помощью окна Open Project или надстройки Extension Manager.

Когда пользователь открывает окно New Project в системе Visual Studio, он ищет шаблоны, инсталлированные на его компьютере; однако он может также искать шаблоны, доступные в сети, выбрав команду Online Templates в панели инструментов. Система Visual Studio позволяет искать шаблоны в Интернете. Выбранный шаблон загружается из сети и инсталлируется на компьютере, а затем с его помощью создается новый проект.

Система Visual Studio 2010 включает в себя новое свойство Extension Manager (рис. 15.16), которое можно вызвать с помощью команды Tools⇨Extension Manager. Надстройка Extension Manager интегрирует коллекцию Visual Studio Gallery (`http://www.visualstudiogallery.com`) прямо в систему Visual Studio. Она также позволяет просматривать коллекцию Visual Studio Gallery и загружать и инсталлировать шаблоны, а также элементы управления и инструменты.

Резюме

В главе описан процесс создания шаблонов элементов и проектов с помощью системы Visual Studio 2010. Существующие проекты и элементы можно экспортировать в шаблоны, которые можно развертывать на других компьютерах. В качестве альтернативы можно создавать шаблоны вручную и добавлять пользовательский интерфейс с помощью интерфейса `IWizard`. Сведения, приведенные в главе, позволяют собирать шаблоны решений и шаблоны проектов, собирать и интегрировать интерфейс мастера и, наконец, собирать проект инсталляции пользовательского шаблона.

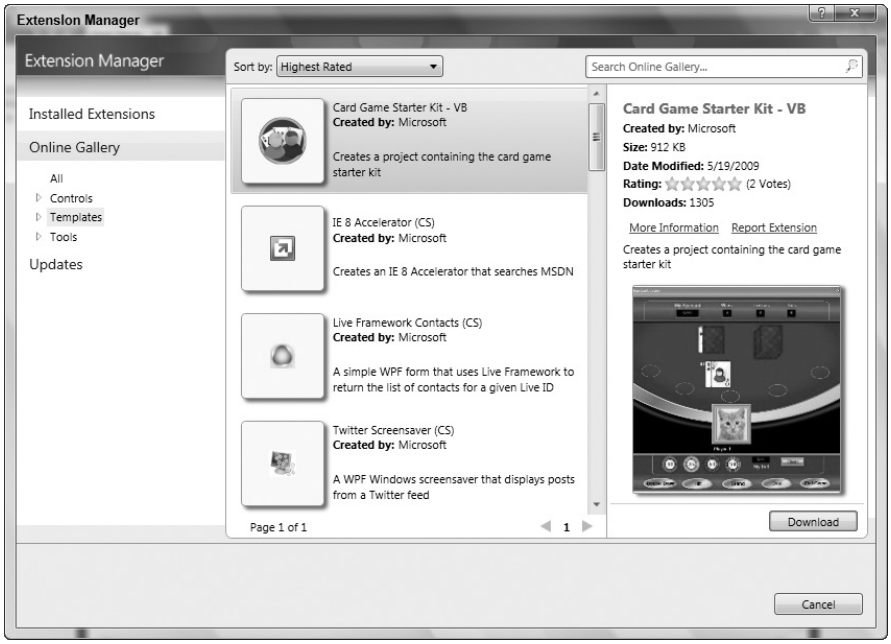

Extension Manager
Extension Manager
Installed Extensions
Online Gallery
All
Controls
Templates
Tools
Updates
Sort by: Highest Rated
Search Online Gallery...
Card Game Starter Kit - VB
Created by: Microsoft
Creates a project containing the card game starter kit
IE 8 Accelerator (CS)
Created by: Microsoft
Creates an IE 8 Accelerator that searches MSDN
Live Framework Contacts (CS)
Created by: Microsoft
A simple WPF form that uses Live Framework to return the list of contacts for a given Live ID
Twitter Screensaver (CS)
Created by: Microsoft
A WPF Windows screensaver that displays posts from a Twitter feed
Page 1 of 1
1
Card Game Starter Kit - VB
Created by: Microsoft
Size: 912 KB
Date Modified: 5/19/2009
Rating: (2 Votes)
Downloads: 1305
More Information
Report Extension
Creates a project containing the card game starter kit
Download
Cancel

Рис. 15.16

16

Особенности языков программирования

В ЭТОЙ ГЛАВЕ...

- Выбор правильного языка программирования для работы
- Работа с новыми возможностями языков C# и VB
- Освоение языка Viausl F#

Языковая экосистема платформы .NET процветает. В настоящее время существуют сотни языков программирования, предназначенных для платформы .NET Framework (почти полный список этих языков можно найти на сайте `www.dotnetpowered.com/languages.aspx`), поэтому разработчики приложений для платформы .NET имеют в своем распоряжении огромный арсенал возможностей. Поскольку платформа .NET Framework изначально задумывалась как многоязычная, эти языки допускают взаимодействие друг с другом и перекрестное обогащение языков при решении задач программирования. Благодаря этому разработчик может выбрать действительно оптимальный язык для выполнения своего задания.

В главе рассматриваются новейшие парадигмы программирования в технологии .NET, а также их особенности и преимущества, облегчающие решение задач. Обсудив парадигмы программирования, мы опишем некоторые новые языковые возможности, появившиеся в системе Visual Studio 2010, включая последнюю новинку в списке языков, поддерживаемых компанией Microsoftt: язык функционального программирования F#.

Гвозди надо забивать правильными молотками

Программисты должны быть гибкими и творческими людьми. Программирование требует элегантности, эффективности и долговечности. Прошли те дни, когда программист был привязан к одному языку и платформе и был вынужден, образно говоря, выворачиваться наизнанку, чтобы удовлетворить требования, которые выдвигала проблемная область. Для разных гвоздей нужны разные молотки.

На платформе .NET существуют сотни языков. Чем же они отличаются друг от друга? По правде говоря, большинство из них представляют собой результат небольшой эволюции других языков и не очень полезны в промышленном производстве. Однако эти языки легко классифицировать в соответствии с парадигмами программирования.

Языки программирования можно классифицировать несколькими способами, но мы предпочитаем крупномасштабный подход, в соответствии с которым языки разделяются на четыре категории: императивные, декларативные, динамические и функциональные. В этом разделе мы кратко рассмотрим эти категории и языки, которые в них входят.

Императивные языки

Классические императивные языки описывают "как", а не "что". Императивные языки были разработаны для того, чтобы перейти на новый уровень абстракции, удаляясь от машинного кода. Говорят, что когда Грейс Хоппер (Grace Hopper) изобрела первый компилятор — систему A-0, — ее коллеги-программисты жаловались, что она может оставить их без работы.

К этой категории относятся языки программирования, в которых инструкции манипулируют в основном состоянием программы. Классическими манипуляторами состояниями являются объектно-ориентированные языки программирования, предназначенные для создания и изменения объектов. Языки C и C++ хорошо соответствуют критериям императивных языков программирования, впрочем, как и наши любимые языки VB и C#.

Эти языки отлично описывают реалистичные сценарии, используя для этого систему типов и объектов. Они являются строгими, т.е. компилятор выполняет большую работу, связанную с проверкой безопасности типов. Безопасность типов (или надежность типов) означает, что программисту трудно изменить тип `Cow` (Корова) на тип `Sheep` (Овца), поэтому, если необходимо объявить тип `Cow` в сигнатуре метода, компилятор проверит, чтобы вместо аргумента типа `Cow` методу не передали аргумент типа `Sheep`. Кроме того, эти языки имеют фантастический механизм повторного использования кода: код, написанный с использованием полиморфизма, можно легко абстрагировать и сделать доступным для совершенно других проектов. К тому же императивные языки очень популярны. Они являются действительно хорошим выбором для работы в коллективе.

Декларативные языки

Декларативные языки описывают "что", а не "как" (в отличие от императивных языков, описывающих, как с помощью инструкций программы манипулировать ее состоянием). Классическим декларативным языком программирования является HTML. Он описывает макет страницы: какие требуются шрифты, текст и элементы декора-

ции, а также какие изображения должны появляться на экране. Другим классическим декларативным языком является SQL. Он описывает, что программисту требуется от реляционной базы. Относительно свежим примером декларативных языков является XAML (eXtensible Application Markup Language), который возглавляет длинный список декларативных языков, основанных на языке XML.

Декларативные языки превосходны для описания и преобразования данных. В этой роли они многие годы использовались в программах, написанных на императивных языках, для извлечения и обработки данных.

Динамические языки

Динамическими называют языки, имеющие "динамические" свойства, например, поздние связывание и вызовы, циклы REPL (Read Eval Print Loops), "утиную" типизацию (т.е. нестрогую типизацию, когда объект, который крякает и ходит, как утка, должен быть уткой) и т.д.

Динамические языки, как правило, откладывают выполнение статических операций на как можно более поздний момент — на этап выполнения программы. В то время как типичный вызов метода в языке C# "`Console.WriteLine()`" проходит статическую проверку и редактирование связей на этапе компиляции, динамический язык отложил бы все это на этап выполнения программы. Вместо этого он искал бы метод "`WriteLine()`" типа "`Console`" в момент выполнения программы, и если бы нашел такой метод, то вызвал бы его. Если же он не нашел бы такой метод или тип, то программист должен был бы как-то отреагировать на сбой и попытаться сделать что-то еще.

К другим свойствам динамических языков относятся расширение объектов, классов и интерфейсов на этапе выполнения программы (т.е. модификация системы типов на ходу), динамическое управление областью видимости (например, переменная, определенная в глобальной области видимости, может быть доступной закрытым и вложенным методам) и многое другое.

Методы компиляции, подобные описанным выше, имеют интересный побочный эффект. Если нет необходимости полностью определять пользовательские типы заранее (поскольку система типов обладает очень большой гибкостью), можно написать код, получающий строгие интерфейсы (такие как COM или другие .NET-сборки, например), и сделать его устойчивым к сбоям и изменению интерфейса. Если интерфейс, который получает код на языке C# от внешней сборки, изменяется, то его, как правило, необходимо перекомпилировать, согласовать с внутренним кодом и выполнить снова. В динамических языках в таких ситуациях можно использовать "механизм отсутствующих методов", и когда конкретный интерфейс изменится, проанализировать его и решить, стоит ли его вызывать снова. Это значит, что программист может написать фантастический код, объединяющий интерфейсы, которые нельзя изменять согласованно.

Динамические языки программирования превосходно решают задачу ускоренного создания программ. Возможность не определять пользовательские типы заранее (что-то можно сделать прямо в коде на языке C#) позволяет сконцентрироваться на коде, решающем задачу, а не на ограничениях типов при их реализации. Циклы REPL (Read Eval Print Loop) позволяют писать прототипы строку за строкой и сразу видеть, как изменения отражаются на программе, не тратя время на компиляцию и отладку.

Если читателей интересуют динамические языки для платформы .NET, то им сильно повезло. Компания Microsoft выпустила язык IronPython (`www.codeplex.com/IronPython`), представляющий собой реализацию языка Python для платформы .NET

Framework. Python — это классический пример динамического языка. Он очень популярен в научных вычислениях, системном администрировании и универсальном программировании. Если язык Python не поразил вашего воображения, то можно загрузить и испытать версию IronRuby (`www.ironruby.net/`), представляющую собой реализацию языка Ruby для платформы .NET Framework. Ruby — это динамический язык программирования, популярный у разработчиков веб-приложений. Несмотря на то что он выпущен относительно недавно, он получил огромное распространение.

Функциональные языки

Функциональными называют языки, которые выполняют вычисления так, как будто это математические функции. Они стараются избегать манипуляций состоянием, вместо этого интерпретируя результаты вычисления функций как строительные блоки для решения задач. Если читатели выполняли ранее какие-нибудь вычисления, то теоретически уже знакомы с функциональным программированием.

Поскольку функциональное программирование обычно не имеет дело с манипулированием состоянием, поле для появления побочных эффектов, генерируемых программой, значительно сужается. Это значит, что функциональное программирование прекрасно подходит для реализации параллельных и конкурентных алгоритмов. Для систем параллельного программирования чрезвычайно важно избежать перекрытия "непреднамеренных" манипуляций состояниями. Взаимные блокировки, состязания и нарушенные инварианты — все это классические проявления нарушения синхронизации в коде, манипулирующем состояниями. Конкурентное программирование, синхронизацию с помощью потоков, общую память и блокировки невероятно трудно реализовать, так почему же не избежать их всех сразу? Поскольку функциональное программирование поощряет программистов писать алгоритмы, не имеющие состояний, компилятор может проанализировать возможность автоматического распараллеливания кода. Это значит, что программист может использовать мощь многопроцессорных систем, не утруждаясь управлением потоками, блокировками и общей памятью.

Функциональные программы очень лаконичны. Как правило, они короче своих императивных аналогов. Более короткий код обычно содержит меньше ошибок и легче тестируется.

Что все это значит?

Перечисленные выше категории очень широки: языки программирования могут иметь свойства, которые являются общими для нескольких категорий одновременно. По этой причине их следует использовать для установления связей между свойствами языков, которые позволяют решить поставленную задачу наилучшим образом.

Языки наподобие C# и VB.NET позволяют использовать преимущества динамических и функциональных языков программирования. Библиотека LINQ (Language Integrated Query) представляет собой яркий пример заимствованной парадигмы.

Рассмотрим следующий пример на языке C# 3.0 с использованием библиотеки LINQ.

```
var query = from c in customers
            where c.CompanyName == "Microsoft"
            select new { c.ID, c.CompanyName };
```

В этом фрагменте кода есть несколько заимствованных свойств. Ключевое слово `var` означает "вывести тип указанного запроса" и выглядит как элемент динамического языка программирования. Сам запрос `from c in ...` выглядит и действует как

элемент декларативного языка SQL, а инструкция `select new { c.ID ...` создает новый анонимный тип, который снова выглядит как нечто динамическое. Код, полученный в результате генерации этих инструкций, особенно интересен: они на самом деле не компилируются в классический промежуточный язык программирования, а компилируются в сущность, которая называется деревом выражений и интерпретируется на этапе выполнения программы, как это принято в динамических языках программирования.

По существу, сами категории языков программирования не имеют большого значения при выборе инструмента для решения поставленной задачи. Главным является перекрестное обогащение возможностей языков разных категорий, которое увеличивает производительность работы программиста, заимствующего лучшие свойства у языков разных категорий. В настоящее время существует тенденция использовать для разработки приложений смесь императивных и динамических языков, а функциональные языки применяются для решения специфических задач из разных предметных областей.

Если вы работаете на платформе .NET, то у вас есть повод радоваться. Взаимодействие языков с помощью спецификации CLS (Common Language Specification) осуществляется практически идеально: вы можете использовать свой любимый императивный язык для решения большинства задач, а затем применить функциональный язык для обработки данных или воспользоваться математическими библиотеками.

Рассказ о двух языках

С момента создания платформы .NET Framework идут непрерывные споры о том, какие языки программирования должны использовать разработчики при создании своих приложений. В большинстве случаев коллективы разработчиков выбирают между C# и VB, опираясь на свои знания о языках C/C++, Java и VB6. Однако несходство между этими языками сильно затрудняет выбор. В прошлом коллективы разработчиков языков в компании Microsoft вносили новшества в свои языки независимо друг от друга. В результате многие свойства появлялись в одном языке, но отсутствовали в другом. Например, язык VB имел интегрированную языковую поддержку для работы с XML-литералами, в то время как в языке C# существовали анонимные методы и итераторы. Несмотря на то что эти возможности приносили пользу программистам, писавшим программы на этих языках, организациям было трудно выбрать один из них.

Фактически в некоторых случаях организации использовали смесь этих языков, пытаясь использовать их лучшие свойства. К сожалению, при этом коллективы разработчиков были вынуждены читать и писать программы на обоих языках или разбивать приложения на фрагменты, одни из которых были написаны на языке C#, а другие — на языке VB.

При разработке системы Visual Studio 2010 и платформы .NET Framework 4.0 в компании Microsoft было принято решение о согласованной эволюции двух основных языков для платформы .NET — C# и VB. Благодаря этому разница между возможностями этих языков должна была быть сведена к нулю (этот феномен получил название *равенство возможностей* (feature parity)).

Однако это не было попыткой объединить два языка — совсем наоборот. Компания Microsoft ясно указала, что каждый язык должен реализовывать свои свойства по-разному, чтобы обеспечить удобства для разработчиков, которые уже писали на этом языке.

В следующих разделах мы опишем языковые свойства, которые были добавлены в систему Studio 2010. Начнем со свойств, общих для обоих языков, а затем перейдем к различиям между ними, большинство из которых будет рассмотрено в контексте равенства возможностей, а также покажем, как эти новшества согласуются со свойствами, которыми уже обладал другой язык.

Компиляция без сборок PIA

Система Visual Studio 2010 обеспечивает превосходную поддержку создания надстроек как на уровне документов, так и на уровне основных приложений пакета Office, таких как Word и Excel. При автоматизации этого процесса хотелось бы иметь возможность заходить в открытые COM-интерфейсы. Для этого пользователь должен сослаться на основные сборки взаимодействия (Primary Interop Assemblies — PIA), чтобы иметь возможность работать с моделью Microsoft Office Object Model.

Кроме того, в прошлом была принята концепция *зависимости развертывания* (deployment dependency), в соответствии с которой сборки PIA должны не просто существовать заранее, но и их версия должна совпадать с заданной. В результате размер надстроек и сложность их развертывания увеличивались без особой необходимости.

В языки VB и C# включена поддержка развертывания приложений как в виде надстроек, так и в виде отдельных приложений, использующих средства автоматизации пакета Office и не требующих предварительной инсталляции сборок PIA на компьютерах пользователей. На рис. 16.1 видно, что в системе Visual Studio 2010 существует новое свойство Embed Interop Types, которое указывает, должен ли компилятор внедрять типы технологии Interop.

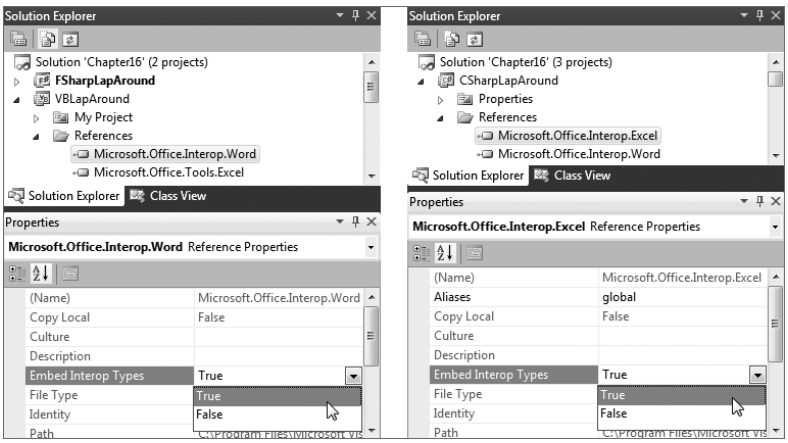


Рис. 16.1

После компиляции приложения любые типы технологии Interop, на которые существуют ссылки, клонируются из сборок PIA в скомпилированное приложение. На рис. 16.2 показано, что интерфейс `Microsoft.Office.Interop.Word.Application` является исполняемым, поскольку он создан в сборке `CShaperLapAround`.

В ходе выполнения этого приложения платформа .NET Framework 4.0 использует новое свойство — *эквивалентность типов*, — позволяющее передавать объекты COM между управляемыми сборками, чтобы привести их к соответствующему типу в сборке-получателе.

Это значит, что в двух сборках могут быть объявлены управляемые типы, которые служат оболочкой для COM-объектов и должны иметь эквивалентные типы, как если бы они были определены с одинаковыми типами.

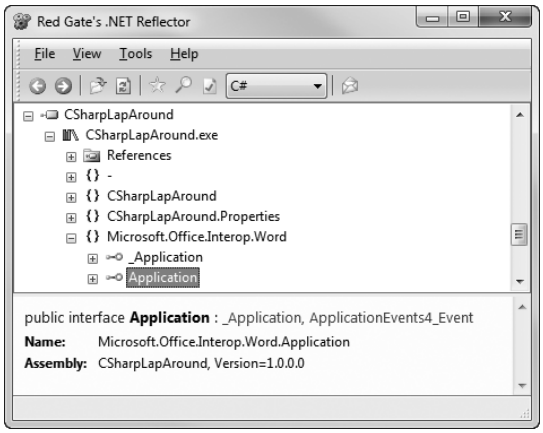


Рис. 16.2



Поскольку компиляция без сборок PIA основана на принципе эквивалентности типов, принятом на платформе .NET Framework 4.0, свойство Embed Interop Types доступно только для проектов, компилируемых на платформе .NET Framework 4.0. В связи с тем, что система Visual Studio 2010 позволяет использовать разные версии .NET Framework, существует опасность случайно создать проект, предназначенный для более ранней версии этой платформы, в которой данное свойство не доступно.

Обобщенная вариантность

Одним из наиболее запутанных свойств механизма обобщенного программирования является наследование. Рассмотрим в качестве примера цепочку наследования класса `Tortoise`, который является производным от класса `Animal`, который, в свою очередь, является производным от класса `Object`. Можно предположить, что, имея тип для списка объектов класса `Tortoise` (т.е. `List <Tortoise>` в языке C# или `List(of Tortoise)` в языке VB), мы могли бы преобразовать его обратно в тип для списка объектов класса `Animal`. Следующий код показывает, что это невозможно.



C#

```
private void InvalidGenericCast(){
    List <Tortoise> tortoiseList = new List <Tortoise> ();
    List <Animal> animalList = tortoiseList;
    animalList.Add(new Lion());
    var notATortoise = tortoiseList[0];
}
```

Сниппет кода MainForm.cs



VB

```
Public Sub InvalidGenericCast()
    Dim tortoiseList As New List(Of Tortoise)
```

```
    Dim animalList As List(Of Animal) = tortoiseList
    animalList.Add(New Lion)
    Dim notATortoise As Tortoise = tortoiseList(0)
End Sub
```

Сниппет кода MainForm.vb

Этот код пытается привести тип для списка объектов класса `Tortoise` в тип для списка объектов класса `Animal`. Если бы это было возможно, то в этот список можно было бы добавить объект класса `Lion`, поскольку он тоже является производным от класса `Animal`. В этом случае последняя инструкция была бы противоречивой, так как список объектов класса `Tortoise` больше не содержал бы только объекты класса `Tortoises`.

Несмотря на то что этот пример демонстрирует недостатки, присущие приведению обобщенных типов, в некоторых случаях они все же разрешаются. Например, следующий фрагмент кода показывает, как тип для списка объектов класса `Tortoise` можно привести к типу для коллекции (`IEnumerable`) объектов класса `Animal`. Поскольку интерфейс `IEnumerable` не допускает модификацию коллекции, это преобразование кажется более безопасным.



C#

```
private void ValidGenericCast(){
    List <Tortoise> tortoiseList = new List <Tortoise> ();
    tortoiseList.Add(new Tortoise());
    IEnumerable <Animal> animalList = tortoiseList;
    var firstAnimal = animalList.First();
}
```

Сниппет кода MainForm.cs



VB

```
Public Sub ValidGenericCast()
    Dim tortoiseList As New List(Of Tortoise)
    tortoiseList.Add(New Tortoise)
    Dim animalList As IEnumerable(Of Animal) = tortoiseList
    Dim firstAnimal As Animal = animalList.First()
End Sub
```

Сниппет кода MainForm.vb

Способность преобразовывать обобщенные типы подобным образом называется *обобщенной вариантностью* (generic variance). В некоторых ситуациях тип переменной желательно сузить, как в предыдущем примере, а в некоторых— расширить. Эти свойства называются *ковариантностью* (covariance) и *контравариантностью* (contravariance) соответственно.

Ковариантность

В предыдущем примере мы видели, как тип для коллекций `IEnumerable` объектов класса `Tortoise` был приведен к типу для коллекции `IEnumerable` объектов класса `Animal`. Эта функциональная возможность называется ковариантностью и является допустимой, потому что интерфейс `IEnumerable` объектов класса `T` был модифицирован ключевым словом `out` .

C#

```
public interface IEnumerable <out T> : IEnumerable{
    IEnumerator <T> GetEnumerator();
}
```

VB

```
Interface IEnumerable(Of Out T) : Inherits IEnumerable
    Function GetEnumerator() As IEnumerator(Of T)
End Interface
```

С помощью ключевого слова out можно также объявлять интерфейсы и делегаты, имеющие переменный параметр типа. Например, в следующем фрагменте кода интерфейс IAnimalCreator допускает расширение типа, заданного параметром, и преобразование интерфейса IAnimalCreator для типа Lion (который реализует метод MainForm) в интерфейс IAnimalCreator для класса Animal, который ожидает метода DoAnimalAction.



C#

```
public interface IAnimalCreator <out T> where T:Animal{
    T CreateAnimal();
}

public partial class MainForm : Form, IAnimalCreator <Lion> {
    public MainForm(){
        InitializeComponent();
        var animal = DoAnimalAction(this);
        MessageBox.Show(animal.GetType().Name);
    }

    private Animal DoAnimalAction(IAnimalCreator <Animal> action) {
        return action.CreateAnimal();
    }

    Lion IAnimalCreator <Lion> .CreateAnimal(){
        return new Lion();
    }
}
```

Сниппет кода MainForm.cs

VB

```
Public Interface IAnimalCreator(Of Out T As Animal)
    Function CreateAnimal() As Animal
End Interface

Public Class MainForm
        Implements IAnimalCreator(Of Lion)

    Public Sub New()

        InitializeComponent()

        Dim animal = DoAnimalAction(Me)
        MessageBox.Show(animal.GetType().Name)

        End Sub

     Public Function DoAnimalAction(ByVal action
         As IAnimalCreator(Of Animal)) _As Animal
         Return action.CreateAnimal()
    End Function

     Public Function CreateAnimal() As Animal _
        Implements IAnimalCreator(Of Lion).CreateAnimal
        Return New Lion
    End Function
End Class
```

Сниппет кода MainForm.vb

Этот код демонстрирует, что данное преобразование является безопасным, поскольку метод CreateAnimal не принимает никакого параметра типа (т.е. параметра типа, модифицированного ключевым словом in). Вместо этого параметр типа определяет тип возвращаемого значения с помощью ключевого слова out. Это делает интерфейс контравариантным по отношению к типу T.

Контравариантность

Существуют ситуации, в которых желательно расширить параметр типа. Эта функциональная возможность называется контравариантностью и используется интерфейсом IComparer. Как можно представить, коллекция IComparer объектов класса Animal одновременно является коллекцией IComparer объектов класса Lion, поскольку если мы можем сравнивать объекты класса Animals, то можем сравнивать и объекты класса Lion. Это преобразование возможно, поскольку интерфейс IComparer был модифицирован с помощью ключевого слова in.

C#

```
public interface IComparer <in T> {
    public int Compare(T left, T right);
}
```

VB

```
   Interface IComparer(Of In T)
    Function Compare(left As T, right As T) As Integer
End Interface
```

И вновь мы можем использовать ключевое слово in для создания контравариантных интерфейсов или делегатов. В примере для класса Animal мы можем определить метод DoAnotherAnimalAction, который будет принимать коллекции IAnimalAction объектов класса Tortoise. Однако на самом деле метод MainForm реализует коллекцию IAnimalAction для класса Animal.



C#

```
public interface IAnimalAction <in T> where T : Animal{
    void Action(T animal);
}

public partial class MainForm : Form, IAnimalAction <Animal> {
    public MainForm(){
        InitializeComponent();
        DoAnotherAnimalAction(this);
    }

     private void DoAnotherAnimalAction(IAnimalAction <Tortoise> action){
        action.Action(new Tortoise());
    }

    void IAnimalAction <Animal> .Action(Animal animal){
        MessageBox.Show("This could be any animal.... " +
        animal.GetType().Name);
    }
}
```

Сниппет кода MainForm.cs



VB

```
Public Interface IAnimalAction(Of In T As Animal)
    Sub Action(ByVal animal As T)
End Interface

Public Class MainForm
    Implements IAnimalCreator(Of Lion), IAnimalAction(Of Animal)

    Public Sub New()
        InitializeComponent()
           DoAnotherAnimalAction(Me)
    End Sub

    Public Sub DoAnotherAnimalAction(ByVal action
                                          As IAnimalAction(Of Tortoise))
        action.Action(New Tortoise)
    End Sub

    Public Function CreateAnimal() As Animal _
              Implements IAnimalCreator(Of Lion).CreateAnimal
        Return New Lion
       End Function

    Public Sub Action(ByVal animal As Animal) _
                     Implements IAnimalAction(Of Animal).Action
        MessageBox.Show("This could be any animal.... " &
                        animal.GetType().Name)
    End Sub
End Class
```

Сниппет кода MainForm.vb

Возвращаясь к этому примеру, мы можем спросить: почему преобразование между интерфейсами `IAnimalAction` для класса `Animal` и `IAnimalAction` для класса `Tortoise` является безопасным? Эта операция является безопасной, потому что компилятор настаивает, что контравариантный параметр типа `T` можно использовать только как входной.

Поскольку класс `Tortoise` всегда можно преобразовать в класс `Animal`, можно совершенно безопасно использовать объект класса `Tortoise` как входной параметр метода, ожидающего объект класса `Animal`.

Язык Visual Basic

Новая версия языка Visual Basic (VB) содержит много новшеств, обеспечивающих более полное выполнение принципа равенства возможностей с языком C#. Она также содержит множество специфических языковых средств, облегчающих инициализацию коллекций и массивов.

Лямбда и анонимные методы

Одним из важнейших недостатков предыдущих версий языка VB было полное отсутствие лямбда и анонимных методов. Анонимный метод — это метод, определенный без имени, а лямбда — это особый вариант выражения, которое может использоваться для генерации делегата (как в большинстве анонимных методов), или дерево выражений (эту тему мы обсудим в главе 29, посвященной языку LINQ). В языке VB теперь существует возможность объявлять одно- и многострочные анонимные методы.

Следующий сниппет кода иллюстрирует множество функциональных возможностей, касающихся анонимных методов в языке VB. В первой строке анонимной функции объявляется, что она принимает параметр `name` и возвращает значение типа `Boolean`. Для определения того, что параметр `name` должен быть строкой, используется вывод типа и тот факт, что в данном случае выражение `As Boolean` можно пропустить, потому что тип возвращаемого значения тоже выводится. Если выведенный входной тип является неправильным или пользователь желает сделать свой код более читабельным, пользователь может также указать тип входного параметра.



VB

```
Dim exp = Function(name) As Boolean
              Console.WriteLine("Hello " & name)
              Return name.length> 10
          End Function
Dim exp2 = Sub(name)
              If exp(name) Then
                  Console.WriteLine(name &_
                                    " is longer than 10 characters")
              End If
           End Sub
exp("Fred")
Dim names = {"Fred", "Joe", "Sandra"}
Array.ForEach(names, exp2)
Array.ForEach(names, Sub(name As String) Console.WriteLine(name))
```

Сниппет кода MainForm.vb

Вторая строка кода иллюстрирует тот факт, что пользователь может создать анонимный метод, не указывая тип возвращаемого значения, известный в языке VB как `Sub`. Обе строки возвращают делегата, который можно вызвать, передав ему параметр, как показано в следующих двух строках кода. Последняя строка иллюстрирует тот факт, что пользователь может определить анонимный метод в строке, содержащей вызов метода. Обратите внимание на то, что здесь была использована сокращенная форма, которая, как и метод `Sub`, состоит из одной строки. Если метод состоит из нескольких строк, необходимо использовать полную нотацию, которая включает инструкцию `End Sub` или `End Function`, в зависимости от того, есть ли возвращаемое значение или нет.

Неявное продолжение строки

Там, где это возможно, компилятор языка VB автоматически осуществляет продолжение строки. Например, программист может теперь писать, как показано ниже, не указывая символ продолжения строки.



VB

```
Public Function LongMethodDeclaration(ByVal parameterOne As Integer,
                                      ByVal parameterTwo As Integer,
                                      ByVal parameterThree As Integer,
                                      ByVal parameterFour As Integer,
                                      ByVal parameterFive As Integer,
                                      ByVal parameterSix As Integer)
                                                          As Integer
    Return parameterOne + parameterTwo + parameterThree +
                parameterFour + parameterFive + parameterSix
End Function
```

Сниппет кода MainForm.vb

Однако в некоторых случаях символ продолжения строки по-прежнему необходим, например, если автор хочет, чтобы список параметров начинался с новой строки.

VB

```
Public Function LongMethodDeclaration _
                                    (ByVal parameterOne As Integer,
                                     ByVal parameterTwo As Integer,
```

Неявное продолжение строки намного облегчает создание выражений LINQ на языке VB, поскольку теперь выражение можно разделить на несколько строк, не добавляя каждый раз символ продолжения строки.

Автоматические свойства с начальными значениями

При определении класса рекомендуется использовать инкапсуляцию, чтобы скрыть его функциональные возможности. Идея заключается в том, чтобы при изменении его реализации программист мог сделать это, не касаясь других частей кода, которые используют данный класс. В соответствии с принципом инкапсуляции открывать поле для доступа рекомендуется с помощью *свойства* (property). Свойством могут быть как обычные методы, просто получающие и отправляющие данные, так и методы, обладающие дополнительными функциональными возможностями, например, генерирующие события при изменении значения свойства.

Из-за такой практики в программах возникает большое количество повторяющегося кода, в которых объявляется свойство и зарезервированное поле. Несмотря на помощь, которую оказывают сниппеты, это приводит к появлению слишком громоздких программ.

В системе Visual Studio 2010 язык VB не только имеет автоматические свойства, т.е. свойства, автоматически резервирующие поле, но и позволяет объявлять начальное значение для свойства точно так же, как это сделано для поля. Начальное значение устанавливается с помощью вызова настройщика свойства после инициализации экземпляра объекта, но до вызова конструктора. Следующий код иллюстрирует работу этого механизма на примере свойства `MaximumWordCount`.



VB

```
Public Property MaximumWordCount As Integer = 10
Public Sub New()
    MessageBox.Show("The maximum word count is " & MaximumWordCount)
    MessageBox.Show("The maximum word count is " & _MaximumWordCount)
```

Сниппет кода MainForm.vb

В этом сниппете кода поле `_MaximumWordCount` нигде не определено. Язык VB открывает зарезервированное поле, используемое автоматическим свойством, просто приписывая перед именем свойства символ подчеркивания. Хотя это и не рекомендуется, можно получить доступ к этому полю непосредственно из класса, поскольку оно объявлено как `Private` .

Инициализация коллекций и литералы массивов

Теперь в языке VB есть компактное обозначение для массивов. Проиллюстрируем его с помощью следующего сниппета кода.



VB

```
' Single dimension arrays
Dim a = {56, 34, 29, 12, 35, 872, 34, 12, 66} 'Integer()
Dim b = {1, 5, 3.54, 3.5} 'Double()
Dim c = {"Betty", "Frank"} 'String()
Dim d = {1, "123", New Animal} 'Object()

'Multi-dimension arrays
Dim e = {{1, 2, 3},
         {4, 5, 6}} 'non-jagged array (ie Integer(,))
Dim f = {({1, 2, 3}),
         ({4}),
         ({5, 6, 7})} 'jagged array (ie Integer()())
```

Сниппет кода MainForm.vb

Обратите внимание на то, что тип этого массива выводится из типа составляющих его элементов. Следовательно, массив `b` имеет тип `Double`, поскольку это позволяет хранить первые два числа, имеющие тип `Integer`, наряду с остальными числами. Если в настройках установлен флажок Option Strict, то массив типа `Object` вызовет ошибку.

Списки и словари можно инициализировать с помощью аналогичных компактных обозначений.



VB

```
Private listOfNames As New List(Of String) From {"Nick",
                                                 "Dave", "Mike", "Chris"}
Private cityLookup As New Dictionary(Of String, String) _
                                        From {{"Nick", "Sydney"},
                                              {"Dave", "Perth"},
                                              {"Mike", "Perth"},
                                              {"Chris", "Sydney"}}
```

Сниппет кода MainForm.vb

Инициализация объектов классов `List` и `Dictionary` выполняется с помощью вызова метода `Add` из вновь созданного экземпляра. Аналогичный компактный синтаксис позволяет добавлять в объекты класса `List` объекты своих собственных типов, создавая метод расширения `Add`, который будет преобразовывать набор начальных значений в экземпляр пользовательского класса.



VB

```
Private listOfWeirdObjects As New List(Of MyListClass)
                                                  From {{"Boo", 45, 67, 4.5},
                                                        {"Foo", 29, 34, 7.4}}
Public Module Extensions
    <Extension()>
    Sub Add(ByVal list As List(Of MyListClass),
            ByVal Name As String,
            ByVal Height As Integer,
            ByVal Weight As Integer,
            ByVal Width As Double)

list.Add(New MyListClass With {
                                .Name = Name,
                                .Height = Height,
                                .Weight = Weight,
                                .Width = Width
                               })
```

```
    End Sub
End Module
```

Сниппет кода MainForm.vb

Обнуляемые необязательные параметры

Прежде обнуляемые параметры было невозможно сделать необязательными. Теперь это стало возможным, и пользователь может определять следующие методы.



VB

```
Public Sub New()
    MethodWithOptionalParameters(5, parameterThree:=6)
End Sub

    Public Function MethodWithOptionalParameters _
                      (Optional ByVal parameterOne As Integer? = Nothing,
                       Optional ByVal parameterTwo As Integer? = 0,
                       Optional ByVal parameterThree As Integer? = Nothing) _
                                                      As Integer
```

Сниппет кода MainForm.vb

Как видим, для необязательных параметров необходимо определить значение по умолчанию. Кроме того, при вызове этого метода иногда необходимо использовать именование параметра (как в выражении `parameterThree:=`), если требуется пропустить необязательный параметр.

Надстройки Visual Basic PowerPacks

Одна из проблем, с которыми часто сталкивались разработчики, работавшие в системе VB6, заключалась в том, что выполнение заданий на платформе .NET требовало намного больше этапов и было намного сложнее, чем в системе VB6. Для того чтобы стимулировать разработчиков переходить от системы VB6 на платформу .NET Framework, в языке VB было введено пространство имен `My`, содержащее сокращенные методы для выполнения часто повторяющихся задач.

Кроме того, коллектив разработчиков языка VB также модифицировал надстройки Visual Basic PowerPacks из предыдущих версий системы Visual Studio, добавив в них большое количество полезных элементов управления и другие классы для программистов, пишущих программы на языке VB.

Система Visual Studio 2010 поставляется вместе с надстройками Visual Basic PowerPacks. Как показано на рис. 16.3, дополнительная закладка в инструментальном окне Toolbox содержит множество элементов для управления рисованием, например Line, Oval и Rectangle.

Все это можно использовать для создания простых графических изображений, одно из которых показано на рис. 16.3, *справа*.

Несмотря на то что надстройки Visual Basic PowerPacks по умолчанию доступны разработчикам, пишущим программы на языке VB, нет никаких причин, запрещающих использовать их при разработке приложений на языке C#. Для использования этих элементов управления в проекте на языке C#, необходимо добавить ссылку на сборку PowerPacks, а затем добавить элементы управления в свое инструментальное окно Toolbox. Теперь эти элементы управления можно использовать в любом приложении типа Windows Forms.

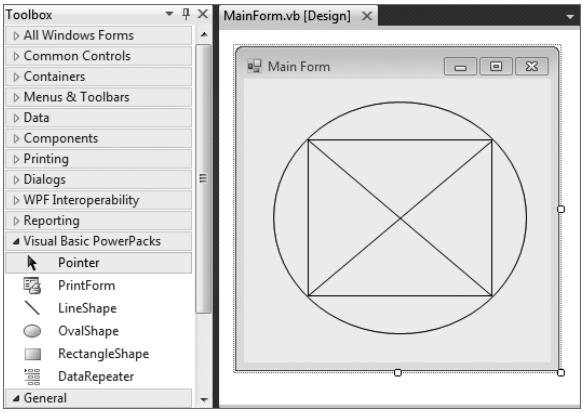


Рис. 16.3

Язык C#

В новой версии языка C# предусмотрено лишь несколько новых функциональных возможностей, касающихся взаимодействия *машинно-ориентированных* (native) и динамических языков.

Позднее связывание с динамическим просмотром

Взаимодействие с другими языками и технологиями часто бывает трудным, особенно с динамическими языками, в которых не всегда можно знать заранее, какие методы может содержать тот или иной класс. В прошлом существовала возможность выполнять такие вызовы, но для этого требовалось глубокое понимание работы механизма отражения и много вызовов для обращения к одному-единственному методу. Новое ключевое слово `dynamic` позволяет выполнять позднее связывание с такими методами.



C#

```
public class DynamicClass{}
public class MoreDynamic : DynamicClass{
    public void SimpleMethod(){
            MessageBox.Show("Dynamic Invoked");
    }
}

public MainForm(){
    InitializeComponent();

    dynamic lateBound = CreateDynamic();
    lateBound.SimpleMethod();
}

public object CreateDynamic(){
    return new MoreDynamic();
}
```

Сниппет кода MainForm.cs

В этом примере кода метод `CreateDynamic` возвращает объект класса `MoreDynamic` как обычный объект. Объявив объект `lateBound` с помощью ключевого слова `dynamic`,

можно вызывать метод SimpleMethod так, будто он был объявлен в самом объекте. Без этого ключевого слова будет вызван механизм контроля статических типов и сгенерирована ошибка компиляции, поскольку метод SimpleMethod не объявлен в объекте класса object (который возвращается методом CreateDynamic, а значит, должен выводиться из типа переменной lateBound).



Одна из проблем, связанных с ключевым словом dynamic, заключается в том, что теперь программист должен сам правильно задавать имя, тип и количество параметров метода, чтобы во время выполнения программы не произошло сбоя. Ключевое слово dynamic блокирует механизм контроля статических типов на этапе компиляции, откладывая проверку на этап выполнения программы.

Именованные и необязательные параметры

В языке C# ощущается недостаток двух важных функциональных возможностей: определять необязательные параметры и задавать параметры, используя их имена. Обе эти возможности повышают практичность и читабельность кода, поскольку они позволяют избежать ненужного раздувания программы (так как обнуляя параметры, программист не обязан указывать или определять дополнительные перегруженные варианты методов, чтобы на самом деле определить необязательные параметры) и позволяют именовать значения параметров.



C#

```
public MainForm(){
    InitializeComponent();
    var output = MethodWithOptionalParameters(parameterTwo: 15);
}

public int MethodWithOptionalParameters(int? parameterOne = null,
     int parameterTwo = 5){
     return (parameterOne ?? 0) + parameterTwo;
}
```

Сниппет кода MainForm.cs

В этом коде оба параметра являются необязательными: первый параметр — это обнуляемое целое число, значение по умолчанию которого равно нулю, а второй — обычное целое число, значение по умолчанию которого равно 5. Параметры становятся необязательными, когда их значение по умолчанию указывается как часть сигнатуры метода. При вызове метода с необязательными параметрами они просто игнорируются. Независимо от того, являются ли параметры необязательными, программист может именовать каждое значение параметра, чтобы читатель кода мог легко понять, чему соответствует тот или иной параметр. Это особенно важно, если значение по умолчанию является константой, как в предыдущем сниппете кода, где смысл константы сразу не ясен.

Использование именованных и необязательных параметров особенно полезно при работе с COM-интерфейсами. Они довольно часто имеют много необязательных параметров, которые прежде должны были задаваться. Теперь это не требуется, поскольку эти параметры можно просто пропускать. Использование именованных значений параметров позволяет разъяснить, какие параметры мы передаем.



Вызов методов с именованными параметрами может усложнить дальнейшее изменение кода. Например, допустим, что мы обращаемся к элементу управления, созданному посторонним лицом и имеющему метод с единственным параметром `height`. Если поставщик элемента управления обновит версию и изменит имя параметра на `controlHeight`, то, даже если сигнатура метода не изменится, наш код больше не сможет найти параметр `height` и произойдет сбой.

Язык F#

F# (читается как "Ф-шарп") — это относительно новый язык, изобретенный Доном Саймом (Don Syme), сотрудником подразделения Microsoft Research в Кэмбридже, Англия (Cambridge, England), участвовавшим в разработке обобщенных типов для платформы .NET Framework. Подразделение Developer Division компании Microsoft недавно включило язык F# в набор языков программирования, поддерживаемых системой Visual Studio 2010 и поставляемых вместе с ней. Язык F# является мультипарадигменным функциональным языком. Это значит, что он является функциональным, но поддерживает и другие стили программирования, например императивный и объектно-ориентированный.

Наша первая программа на языке F#

Запустим систему Visual Studio 2010 и создадим новый проект на языке F#. Как показано на рис. 16.4, шаблон приложения F# Application находится в узле Visual F#, расположенном в диалоговом окне New Project. Присвойте ему имя и щелкните на кнопке OK.

Шаблон F# Application просто создает проект на языке F# с одним исходным файлом `Program.fs`, который не содержит ничего, кроме ссылки на центр разработки языка F# (F# Developer Center) — `http://fsharp.net`. Если вы хотите глубже изучить язык F#, то лучше всего начинать с шаблона F# Tutorial. Он создает обычный проект на языке F#, в котором главный исходный файл `Tutorial.fs` содержит около 280 строк документации, описывающей начало работы с языком F#. Чтение этого файла и проверка доступных свойств языка само по себе является интересным упражнением. Пока вернемся к файлу `Program.fs`, создадим канонический пример "Hello World" и запустим его с разными опциями компиляции и настройками для интерактивной работы. Добавьте код

```
#light

printfn "Hello, F# World!"
```

Первая инструкция, `#light`, — это флажок компилятора, который сигнализирует о том, что код написан с использованием *упрощенного синтаксиса* (lightweight syntax). В правилах упрощенного синтаксиса большая роль отводится пробелам, благодаря чему можно избежать лексем наподобие `in` или `;;`. Вторая инструкция выводит на консоль строку `"Hello, F# World!"`.

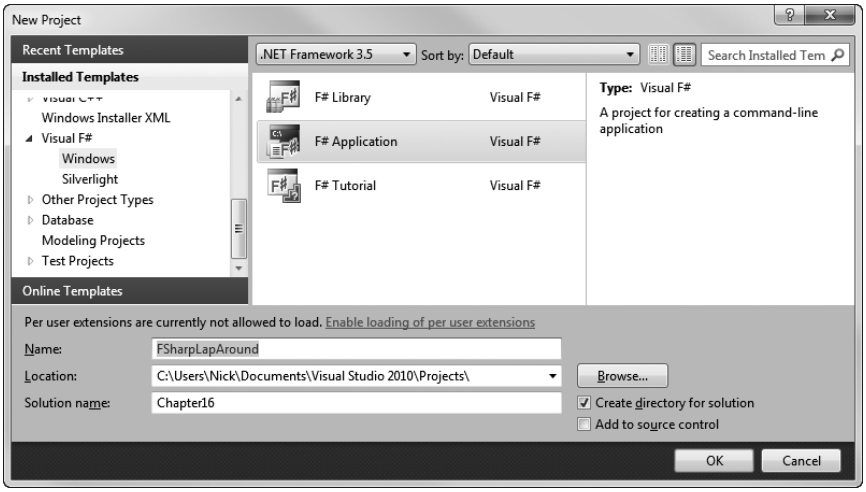


Рис. 16.4



Читатели, работавшие с ранними версиями языка F#, могут обратить внимание на то, что теперь их код вызывает ошибки на этапе компиляции. Язык F# появился в рамках исследовательского проекта и только теперь был превращен в коммерческий продукт. По этим причинам он был переработан, и некоторые операции были перемещены из пространства имен `FSharp.Core` во вспомогательные сборки. Например, команда `print_endline` была перемещена в сборку `FSharp.PowerPack.dll`. Библиотеку F# Powerpack можно загрузить с веб-сайта центра разработки языка F# `http://fsharp.net`..

Программу, написанную на языке F#, можно выполнить двумя способами. Во-первых, можно просто запустить приложение, как обычно (нажав клавишу <F5> для начала отладки). Система скомпилирует и запустит программу так, как показано на рис. 16.5.

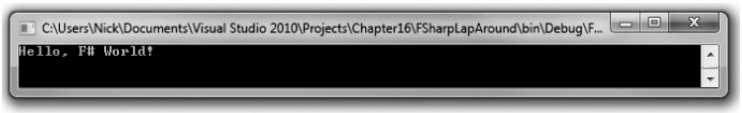


Рис. 16.5

Во-вторых, можно запустить программу F# с помощью окна F# Interactive в системе Visual Studio. Это позволяет подсветить и выполнить код в интегрированной среде разработки Visual Studio и немедленно увидеть результаты работы программы. Кроме того, выполняемую программу можно модифицировать на ходу!

Окно F# Interactive открывается с помощью команды View⇨Other Windows⇨F# Interactive или комбинации клавиш <Ctrl+Alt+F>, как показано на рис. 16.6.

В окне F# Interactive пользователь может приступить к взаимодействию с компилятором языка F# с помощью приглашения REPL (Read Eval Print Loop). Это означает, что каждая набранная программистом строка на языке F# будет немедленно скомпилирована и выполнена. Приглашение REPL является превосходным средством, если пользователь хочет быстро проверить идеи и модифицировать свою программу на ходу. Оно позволяет быстро экспериментировать с алгоритмами и незамедлительно создавать прототипы.

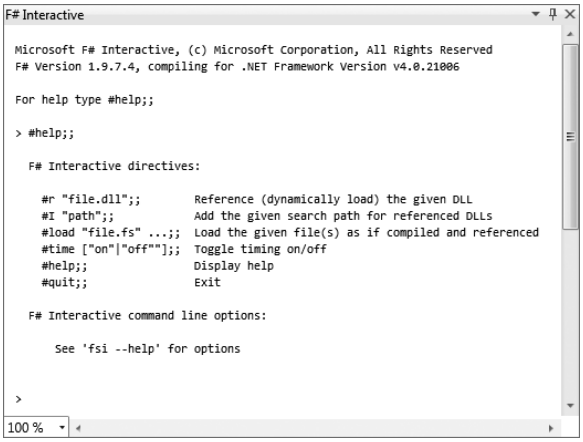


Рис. 16.6

Однако приглашение REPL в окне F# Interactive не позволяет разработчику воспользоваться остальными преимуществами системы Visual Studio, которые она обеспечивает благодаря технологии IntelliSense, сниппетам кода и другим механизмам. Лучше всего использовать преимущества обеих подсистем: применить текстовый редактор системы Visual Studio для создания программы, а затем пропустить результат через механизм интерактивного приглашения. Для этого необходимо нажать комбинацию клавиш <Alt+Enter>, когда курсор находится на любой подсвеченной части исходного кода на языке F#. В качестве альтернативы можно использовать контекстное меню, которое открывается после щелчка правой кнопкой мыши, чтобы открыть окно F# Interative, как показано на рис. 16.7.

Нажав комбинацию клавиш <Alt+Enter> или выбрав команду Send To Interactive, программист посылает выделенный фрагмент кода сразу в механизм Interactive Prompt, который немедленно выполняет этот код (рис. 16.8).

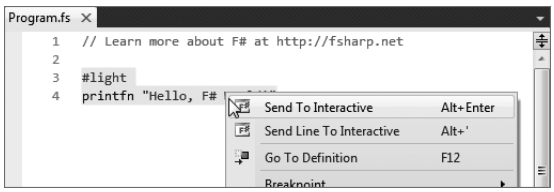


Рис. 16.7

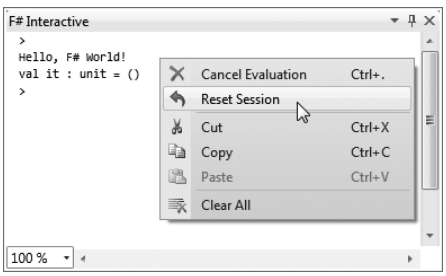


Рис. 16.8

На рис. 16.8 показано также контекстное меню для окна F# Interactive, в котором есть команды Cancel Evaluation (для долго выполняемых операций) и Reset Session (для отмены всего, сделанного ранее).

Исследование свойств языка F#

Введение в язык F# выходит за рамки рассмотрения нашей книги, но мы считаем целесообразным обсудить некоторые из наиболее важных свойств этого языка. В лю-

бом случае это должно вызвать у читателей интерес к языку F# и стимулировать их к его дальнейшему изучению.

Самым общим типом данных в языке F# является список, представляющий собой простую коллекцию с набором выразительных операторов. Программист может создать пустые или многомерные списки и даже классический простой список. Список в языке F# является неизменяемым, т.е. его невозможно модифицировать после того, как он был создан; его можно только скопировать. Язык F# имеет механизм под названием List Comprehensions, позволяющий создавать и манипулировать списками еще проще и выразительнее. Рассмотрим пример.

```
#light

let countInFives = [ for x in 1 .. 20 do if x % 5 = 0 then yield x ]

printf "%A" countInFives
System.Console.ReadLine()
```

Выражение в квадратных скобках представляет собой классический цикл “for”, применяемый к списку, содержащему элементы от 1 до 20 (выражение `".."` представляет собой сокращение, означающее создание списка с элементами от 1 до 20). Выражение `"do"` является сокращением, означающим, что цикл “for” применяется к каждому элементу списка. В данном случае в результате этого действия программа “вычисляет” `x`, для которого выполняется условие, “если `x` по модулю 5 равно 0”. Квадратные скобки представляют собой сокращение, означающее “создать новый список, содержащий возвращенные элементы”. Это очень выразительный способ определения нового списка на ходу с помощью всего лишь одной строки.

Механизм языка F# под названием Pattern Matching реализует гибкий и мощный способ создания потока управления. В языке C# программисты пользуются переключателями (или наборами операторов “if else”), но всегда ограничены типами, к которым их можно применять. Механизм переключения в языке F# похож на своих предшественников, но является более гибким, позволяя не ограничивать проверяемые типы и значения. Например, рассмотрим функцию, вычисляющую числа Фибоначчи с помощью механизма Pattern Matching.

```
let rec fibonacci x =
    match x with
    | 0 | 1 -> x
    | _ -> fibonacci (x-1) + fibonacci (x-2)

printfn "fibonacci 15 = %i" (fibonacci 15)
```

Оператор | указывает, что программист хочет сравнить результат работы функции с результатом вычисления выражения, стоящего в правой части. Первая строка этой инструкции требует, чтобы функция возвращала входное значение `x`, если `x` равен `0` или `1` . Вторая строка требует рекурсивно вызвать функцию `Fibonacci` с входным параметром `x-1` и суммировать этот результат с результатом рекурсивного вызова для параметра `x-2` . Последняя строка выводит результат работы функции `Fibonacci` на консоль.

Сопоставление образцов (pattern matching) в функциях имеет интересный побочный эффект — оно упрощает диспетчеризацию потока управления в зависимости от разных типов получаемых параметров. В языках C#/VB.NET программисты традиционно пишут ряд перегруженных методов, основываясь на типах параметров, а в языке F# это не обязательно, поскольку синтаксис сопоставления образцов позволяет достичь того же с помощью одной-единственной функции.

Еще одной важной особенностью функциональных языков программирования, которой обладает язык F#, являются ленивые вычисления. Ленивые вычисления означают, что компилятор может составить расписание вычисления функции или выражения только тогда, когда это необходимо, а не заранее. Иначе говоря, на этапе выполнения код содержит только необходимые вычисления, а меньшее количество циклов и операторов ускоряет его работу.

Как правило, когда переменной присваивается некоторое выражение, оно сразу выполняется в соответствии с приоритетами операторов, а результат присваивается переменной. Исходя из того, что функциональное программирование не имеет побочных эффектов, теперь можно не вычислять этот результат немедленно (поскольку вычисление выражения по порядку не является обязательным). Выражение следует вычислить только тогда, когда эта переменная действительно понадобится. Рассмотрим простой пример.

```
let lazyDiv = lazy (10 / 2)
printfn "%A" lazyDiv
```

В первой строке ключевое слово `lazy` означает, что функция или выражение вычисляется только по прямому указанию. Вторая строка выводит содержимое переменной `lazyDiv` на консоль. При выполнении этого примера результатом будет строка `"(unevaluated)"`. Это объясняется тем, что входные параметры для функции `printfn` аналогичны входным параметрам для делегата. На самом деле программист должен дать прямое указание или вызвать выражение, чтобы оно вернуло результат, как показано в следующем примере.

```
let lazyDiv2 = lazy (10 / 2)
let result = lazyDiv2.Force()
print_any result
```

Функция `lazyDiv2.Force()` вынуждает вычисление выражения `lazyDiv2`.

Эта концепция является очень мощным средством повышения быстродействия программы. Сокращение объема работы или памяти, необходимой приложению, чрезвычайно важно для повышения скорости как запуска, так и выполнения программы. Кроме того, ленивые вычисления эффективны при работы с большими объемами данных. Если необходимо перебрать терабайты данных, хранящихся на диске, можно легко написать функцию ленивого перебора, чтобы загружать данные только тогда, когда они действительно нужны. Группа Applied Games Group в подразделении Microsoft Research написала прекрасную статью о механизме ленивых вычислений в языке F#, которые выполняются точно по описанному выше сценарию (см. `http://blogs.technet.com/apg/archive/2006/11/04/dealing-with-terabytes-with-f.aspx`).

Резюме

В главе рассмотрены разные стили программирования, их преимущества и недостатки. Система Visual Studio 2010 объединяет в одно целое два основных языка для платформы .NET — C# и VB, — чтобы достичь равенства возможностей. Согласованная эволюция этих языков позволяет сократить стоимость разработки проектов и дает разработчикам возможность более просто переключаться с одного языка на другой. В главе рассмотрено последнее новшество среди языков, поддерживаемых компанией Microsoft, — язык Visual F#. По мере увеличения масштаба решаемых задач увеличивается сложность, связанная с необходимостью разработки параллельных приложений. Язык Visual F# можно использовать для решения этих проблем с помощью параллельных операций без увеличения сложности приложения.

ЧАСТЬ IV

Расширенные клиентские приложения

- **ГЛАВА 17.** Проекты Windows Forms Application
- **ГЛАВА 18.** Подсистема Windows Presentation Foundation (WPF)
- **ГЛАВА 19.** Бизнес-приложения для пакета Office

17

Проекты Windows Forms Applications

В ЭТОЙ ГЛАВЕ...

- Создание нового приложения Windows Forms
- Проектирование макетов форм и элементов управления с помощью инструментов системы Visual Studio и свойств управления
- Использование контейнерных управляющих элементов и свойств управления для автоматического изменения размеров элементов управления при изменении размеров приложения

С первых дней своего существования система Visual Studio обеспечивала превосходную визуальную среду для быстрой разработки Windows-приложений. Конструктор, встроенный в систему Visual Studio 2010, предоставляет пользователю широчайшие возможности для создания интерфейса, не прибегая к кодированию вручную: от простых процедур перетаскивания до размещения на форме графических элементов управления и тонкой настройки их макетов и функционирования.

В главе рассматриваются мощные средства поддержки графического проектирования и полный набор элементов управления, позволяющих достичь оптимальной производительности при создании приложения Windows Forms.

Начало

Начнем с создания нового проекта типа Windows Forms. Выберите команду File⇨New⇨Project, чтобы создать проект в новом решении. Если необходимо добавить проект в уже существующее решение, следует выполнить команду File⇨Add⇨ New Project.

Приложения Windows Forms можно создать как на языке VB, так и на языке C#. В обоих случаях проект Windows Forms Application является выбором по умолчанию в диалоговом окне New Project. Выберите категорию Windows, как показано на рис. 17.1

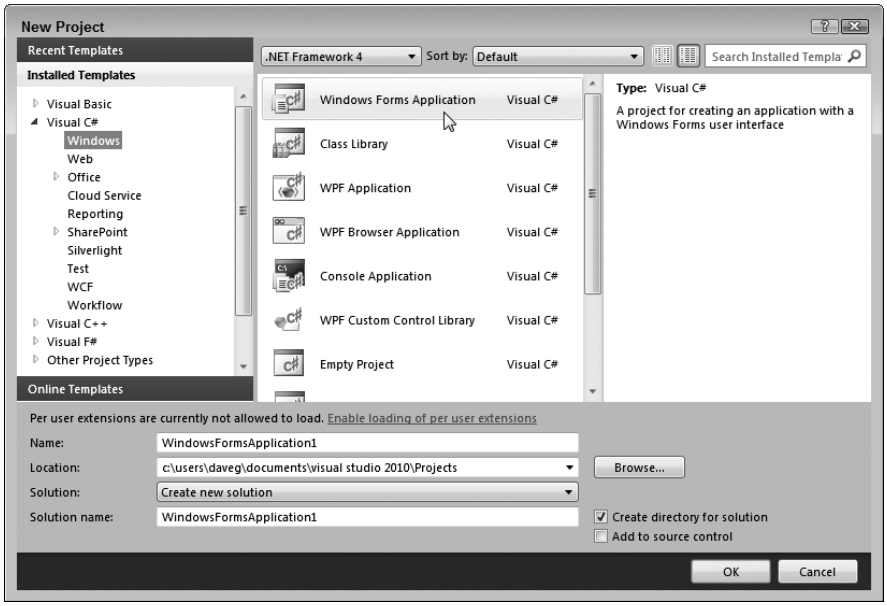


Рис. 17.1

Диалоговое окно New Project позволяет выбрать версию платформы .NET, для которой создается приложение. В отличие от приложений WPF проекты Windows Forms можно было создавать на платформе .NET Framework 1.0, и они остаются в списке доступных проектов для любой версии этой платформы. Введя приемлемое имя проекта, щелкните на кнопке OK, чтобы создать новый проект Windows Forms Application.

Проект Windows Form

При создании проекта для Windows-приложения система Visual Studio 2010 автоматически создает отдельную пустую форму, готовую для проектирования пользовательского интерфейса (рис. 17.2). Визуальный дизайн проекта Windows Form можно создать двумя способами: используя мышь для изменения размеров и позиционирования элементов или изменяя значения свойств управляющих элементов в окне Properties.

Размер почти каждого визуального элемента управления, включая саму форму Windows Form, можно изменять с помощью мыши. *Захваты* (grippers) для изменения размеров появляются, когда форма или элемент контроля находится в фокусе в режиме Design. Для проекта Windows Form они видны только внизу, справа и в правом нижнем углу. С помощью мыши можно взять эти захваты и тащить, изменяя размеры. В процессе изменения размеров они выводятся в правом нижнем углу статусной строки в виде чисел.

Кроме того, размеры и позиции форм и элементов управления можно задавать с помощью соответствующего свойства. Как указывалось в главе 2, в окне Properties (см. рис. 17.2, *справа*) выводятся текущие значения многих атрибутов формы, в частности, свойства `Size`, которое состоит из свойств `Height` и `Width`. Для того чтобы увидеть значения индивидуальных свойств, образующих составное свойство, необходимо щелкнуть на пиктограмме разворачивания. Задать размеры формы в пикселях можно,

введя конкретные значения в поля отдельных свойств `Height` и `Width` или в поле составного свойства `Size` в формате "ширина, высота".

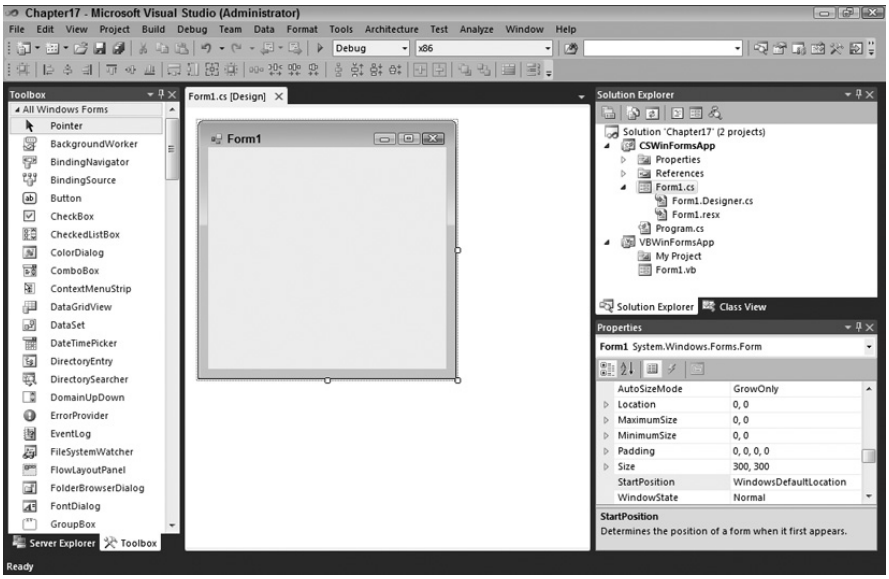


Рис. 17.2

Окно Properties, показанное на рис. 17.3, содержит некоторые значения доступных свойств для настройки внешнего вида и функционирования элементов управления и форм.

Свойства выводятся в одном из двух видов: либо объединенные вместе по категориям, либо в алфавитном порядке. Представление свойств управляется первыми двумя пиктограммами в инструментальной панели окна Properties. Следующие две пиктограммы позволяют переключать список атрибутов с представления свойств на представление событий (Properties и Events).

Внешний вид и стиль формы описывается тремя категориями свойств: Appearance, Layout и Window Style. Многие свойства из этих категорий доступны для элементов управления Windows.

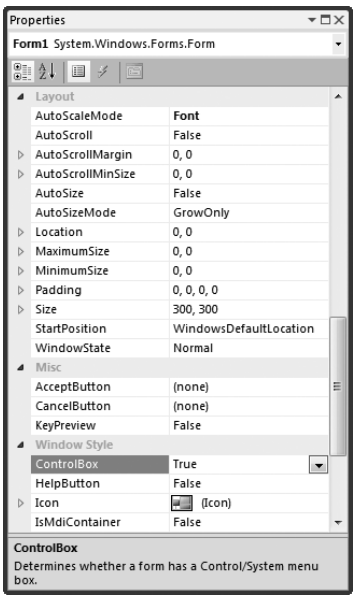


Рис. 17.3

Свойства Appearance

Категория Appearance описывает цвета, шрифты и стиль границы формы. Как правило, разработчики приложений типа Windows Forms не изменяют значения большинства свойств, заданные по умолчанию. Чаще всего изменяется свойство `Text`, поскольку оно управляет надписью на кнопке, расположенной на форме.

Если разработчик создает форму с необычным поведением, то он может зафиксировать ее размеры или задать специальный стиль границы, как это часто делается при создании специальных инструментальных окон. Свойство `FormBorderStyle` управляет обработкой аспекта внешнего представления формы.

Свойства Layout

Кроме свойства `Size`, рассмотренного ранее, в категорию Layout входят свойства `MaximumSize` и `MinimumSize`, задающие максимально и минимально допустимые размеры окна. Свойства `StartPosition` и `Location` можно использовать для управления положением формы на экране. С помощью свойства `WindowState` форма может появляться на экране с минимальными, максимальными или нормальными размерами, заданными по умолчанию.

Свойства Window Style

Категория Window Style содержит свойства, определяющие элементы строки заголовка формы, включая кнопки максимизации и минимизации, помощи и пиктограмму формы. Свойство `ShowInTaskbar` определяет, будет ли форма указана на панели задач Windows. Другим замечательным свойством из этой категории является свойство `TopMost`, используемое для того, чтобы гарантировать, что форма всегда будет появляться поверх всех других окон, даже если она не находится в фокусе, а также свойство `Opacity`, которое можно использовать для создания полупрозрачных форм.

Настройки конструктора форм

Некоторые настройки интегрированной среды разработки Visual Studio можно изменять, чтобы упростить пользовательский интерфейс на этапе конструирования формы. В диалоговом окне Options, показанном на рис. 17.4, системы Visual Studio 2010 две страницы настроек взаимодействуют с конструктором Windows Forms Designer.

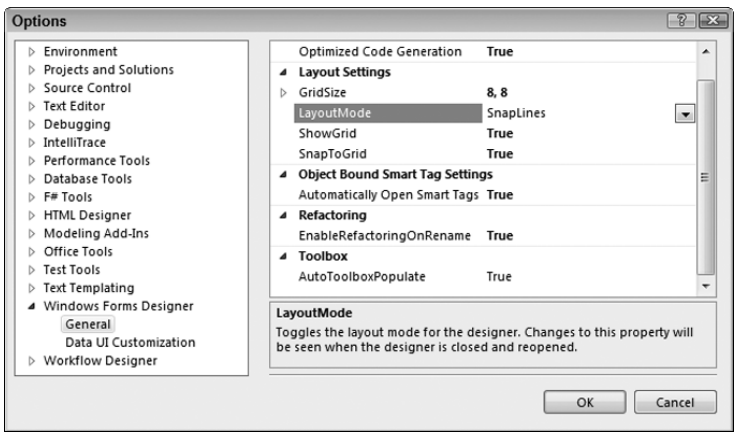


Рис. 17.4

Основными настройками, влияющими на проектирование формы, являются настройки макета. По умолчанию система Visual Studio 2010 использует шаблон макета под названием SnapLines. Вместо размещения видимых компонентов на форме, размеченной невидимой сеткой, макет SnapLines помогает позиционировать компоненты, используя контекст окружающих элементов управления и границы формы. Сейчас мы покажем, как использовать этот новый режим, но если разработчик предпочитает старый стиль проектирования форм, впервые предложенный в системе Visual Basic 6 и использовавшийся в двух первых версиях системы Visual Studio .NET, то он может изменить значение свойства LayoutMode на `SnapToGrid`.



Режим макета SnapToGrid будет по-прежнему использоваться, даже если значение свойства LayoutMode установлено равным `SnapLines`. Режим SnapLines становится активным только тогда, когда изменяется положение одного элемента управления относительно другого. В других ситуациях включается режим SnapToGrid, позволяющий позиционировать элементы управления по узлам сетки.

Свойство GridSize используется для позиционирования и изменения размера элементов управления на форме. При перемещении элементов управления на форме они привязываются к конкретным точкам на основе заданных разработчиком значений. В большинстве случаев разработчики считают, что сетка с ячейками 8×8 (заданная по умолчанию) слишком грубая для точной настройки, и изменяют ее на сетку с ячейками 4×4.



Режимы SnapToGrid и SnapLines позволяют проектировать пользовательский интерфейс с помощью мыши. Как только приблизительное положение элемента управления установлено, его можно уточнить, "подталкивая" элемент управления с помощью навигационных клавиш.

В режиме ShowGrid на плоскости формы демонстрируется сеть, состоящая из точек, а в режиме SnapToGrid разработчику легче определить, где находится элемент управления при его перемещении. Для того чтобы увидеть результаты изменения настроек, необходимо закрыть конструктор формы и вновь его открыть. В заключение, задав свойство SnapToGrid равным `False`, можно отключить систему помощи макета, доступную в режиме SnapToGrid, и перейти к проектированию формы в свободном формате.

Просматривая эту панель, разработчик может пожелать установить значение Automatically Open Smart Tags равным `False`. Значение `True`, заданное по умолчанию, открывает список интеллектуальных дескрипторов, связанных с каждым элементом управления, добавленным на форму. Это может отвлекать внимание на первых этапах разработки формы. Интеллектуальные дескрипторы будут рассмотрены в этой главе позднее.

Другая страница предпочтений, которую можно настроить для конструктора Windows Forms Designer, – Data UI Customization (рис. 17.5). Она используется для автоматического связывания разных элементов управления с разными типами данных при соединении с базой данных.

Как видно на рисунке, тип данных `String` связан с пятью широко используемыми элементами управления, среди которых элемент управления TextBox задается по умолчанию. Когда поле базы данных, имеющее тип String, будет добавлено на форму, система Visual Studio автоматически сгенерирует элемент управления TextBox для хранения этого значения.

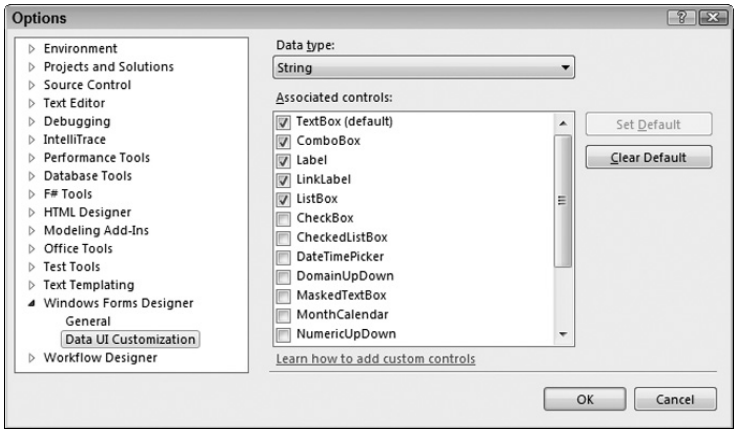


Рис. 17.5

Другие элементы управления, помеченные как элементы, связанные с типами данных (ComboBox, Label, LinkLabel и ListBox), можно использовать при редактировании источника данных и стиля.



При этом следует еще раз проанализировать элементы управления, связанные с каждым типом данных, и убедиться, что эти связи установлены правильно. Например, все переменные типа `DateTime` автоматически представляются с помощью элемента управления DateTime Picker, но их можно также связать с элементом управления MonthCalendar.

Работа с элементами управления, связанными с типами данных, рассматривается в главе 27.

Добавление и позиционирование элементов управления

На форму типа Windows Form можно добавлять элементы управления двух типов: графические компоненты, которые действительно находятся на форме, и компоненты, не имеющие конкретного визуального представления.

Графические элементы управления можно добавить на форму двумя способами. Первый метод — найти требуемый элемент управления в окне Toolbox и дважды щелкнуть на нем. В этом случае система Visual Studio 2010 разместит его на форме в позиции, заданной по умолчанию, — первый элемент управления будет расположен в левом верхнем углу формы, а последующие — снизу и справа.



Если окно Toolbox закрыто, то оно не откроется автоматически в следующий раз, когда будет открыт конструктор Windows Forms Designer. Для того чтобы оно снова открылось, необходимо выбрать команду View⇨Toolbox.

Второй способ — щелкнуть мышью и перетащить элемент из списка на форму. В процессе перетаскивания над допустимыми областями формы курсор мыши изме-

няется, показывая, где будет размещен элемент управления. Это позволяет непосредственно позиционировать элемент управления в нужном месте, а не добавлять его сначала на форму, а затем перетаскивать в требуемую позицию. В любом случае, как только элемент управления будет размещен на форме, его можно перемещать сколько угодно раз, поэтому на самом деле не имеет значения, каким образом разработчик поместит элемент управления на проектируемую форму.



На самом деле существует и третий способ добавления элементов управления на форму — скопировать и вставить элемент управления или набор элементов управления с другой формы. При одновременной вставке группы элементов управления можно сохранить их относительное положение и выравнивание относительно друг друга. При этом будут сохранены все другие свойства, хотя имена элементов управления могут оказаться измененными, потому что они должны быть уникальными.

При проектировании макета формы в режиме SnapLines (см. предыдущий раздел) и перемещении элементов управления по экрану появляется множество ориентиров. Они представляют собой хорошо зарекомендовавшие себя на практике маркеры позиционирования и размера и позволяют разработчику относительно легко позиционировать элементы управления относительно друг друга и границ формы.

На рис. 17.6 показан элемент управления Button, перемещаемый в левый верхний угол формы. Как только он окажется возле рекомендованной позиции, он будет установлен на точно заданное расстояние от левой и верхней границ формы, и на экране появятся небольшие синие ориентиры.

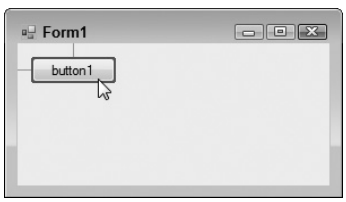


Рис. 17.6

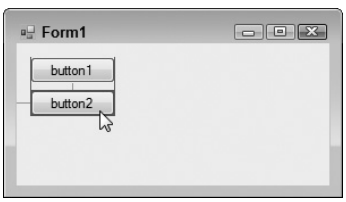


Рис. 17.7

Эти ориентиры влияют и на позиционирование, и на размеры элемента управления, позволяя привязывать его к любой из четырех границ формы, но это лишь вершина айсберга под названием SnapLines. Когда на форме окажутся дополнительные компоненты, при их перемещении на ней будут появляться все новые и новые ориентиры.

На рис. 17.7 показано перемещение второго элемента управления Button. Ориентир, размещенный слева, совпадает с ориентиром, описанным ранее. Он задает идеальное расстояние от левой границы формы до кнопки.

Однако теперь на форме видны три дополнительных ориентира. Две синие вертикальные линии с каждой из сторон элемента управления показывают выравнивание его левой и правой сторон по отношению к другой кнопке, уже размещенной на форме (вполне естественно ожидать, что обе кнопки должны иметь одинаковые размеры). Оставшаяся вертикальная линия задает идеальное расстояние между двумя кнопками.

Вертикальное выравнивание текстовых элементов управления

Одна из проблем, связанных с выравниванием текста, заключается в том, что вертикальное выравнивание текста в элементе TextBox отличается от вертикального выравнивания текста в элементе Label. Дело в том, что текст в каждом элементе управления имеет разное вертикальное расстояние от верней границы элемента. Если просто выровнять эти разные элементы управления, ориентируясь по их границам, то текст, размещенный внутри них, может оказаться не выровненным.

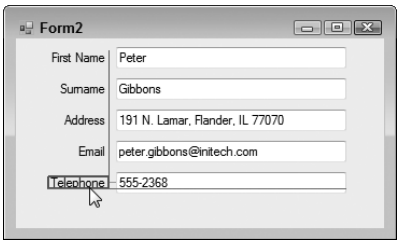


Рис. 17.8

Как показано на рис. 17.8, при выравнивании элементов управления, имеющих текст, на форме появляются дополнительные ориентиры. В данном примере метка Telephone выравнивается по отношению к полю ввода, содержащему реальный номер телефона. При этом на форме появляется линия, по умолчанию имеющая пурпурный цвет, которая указывает правильное место для элемента управления. Метку все еще можно выравнивать по верхней или нижней границе поля ввода, слегка сдвигая и фиксируя ее по ориентирам, но эти новые ориентиры позволяют избежать сложностей при выравнивании.

Обратите внимание на то, что другие ориентиры показывают, как правильно выровнять метку в горизонтальном направлении по метке, расположенной над ней, причем так, чтобы она находилась на рекомендованном расстоянии от поля ввода.

Автоматическое позиционирование нескольких элементов управления

Система Visual Studio 2010 предоставляет в распоряжение разработчика дополнительные средства для автоматического форматирования внешнего вида элементов управления одновременно с их расположением в заданном месте формы. Меню Format, показанное на рис. 17.9, обычно доступно только тогда, когда разработчик включает режим Design. В этом режиме интегрированная среда разработки позволяет автоматически выравнивать, изменять размеры и позиционировать группы элементов управления, а также задавать порядок следования элементов управления, если они перекрываются. Эти команды доступны также на инструментальной панели проектирования, а также после нажатия соответствующих комбинаций клавиш.

Форма, показанная на рис. 17.9, содержит несколько элементов управления TextBox разной ширины. Они выглядят небрежно и должны иметь одинаковую ширину, равную максимальной из них. Меню Format предоставляет разработчику возможность автоматически уравнивать размеры элементов управления с помощью команды Make Same Size⇨Width.



Команды в меню Make Same Size используют первый элемент управления в качестве шаблона для выравнивания размеров. Разработчик может сначала выбрать элемент управления, который следует использовать в качестве шаблона, а затем добавлять элементы один за другим, удерживая нажатой клавишу <Ctrl>. В качестве альтернативы, если все элементы управления имеют одинаковые размеры, можно выбрать их и изменить размер группы одновременно с помощью мыши.

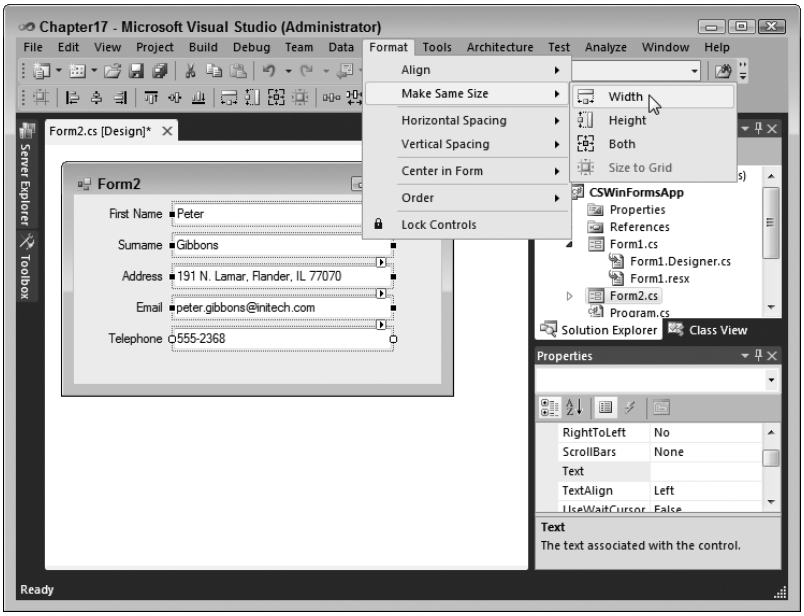


Рис. 17.9

Аналогично можно автоматически выравнивать несколько элементов управления. Сначала следует выбрать элемент, граница которого должна использоваться в качестве образца, а затем выбрать все остальные элементы, которые необходимо выровнять по заданному образцу. Потом надо выбрать команду Format⇨Align и указать, какое выравнивание необходимо выполнить. В данном примере элементы управления Label должны быть выровнены по правой границе. Это можно сделать, используя ориентиры, но, как правило, проще использовать возможность группового выравнивания.

Две другие полезные функции — команды горизонтальной и вертикальной разрядки. Они автоматически выравнивают расстояния между элементами управления в соответствии с выбранными настройками.

Порядок перебора и перекрытия элементов управления

Многие пользователи считают, что при работе с приложениями клавиатура удобнее, чем мышь, особенно если приходится иметь дело с большими объемами данных. Следовательно, важно, чтобы при нажатии клавиши <Tab> курсор перемещался от одного поля к другому в ожидаемом порядке.



Если установить значение свойства TabStop равным `False`, то элемент управления будет пропускаться при нажатии клавиши <Tab> и пользователь не сможет установить на него фокус, не используя мышь.

Некоторые элементы управления никогда не попадают в фокус, например Label. Они также имеют свойство TabIndex; однако при нажатии клавиши <Tab> эти элементы управления пропускаются.

По умолчанию порядок перебора элементов управления совпадает с порядком, в котором они добавлялись на форму. С самого начала свойству TabIndex каждого элемента

управления приписывается некое значение. Чем меньше это значение, тем ближе к началу списка находится элемент управления.

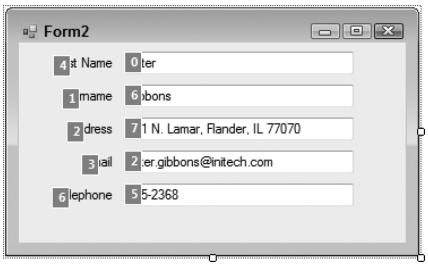


Рис. 17.10

Система Visual Studio имеет полезную функциональную возможность для просмотра и настройки порядка перебора всех элементов управления, существующих на форме, при нажатии клавиши <Tab>. Если выбрать команду View⇨Tab Order, то на форме высветятся значения свойства TabIndex для каждого элемента управления, как показано на рис. 17.10. В данном примере значения свойства `TabIndex`, присвоенные элементам управления, следуют не по порядку. Это может привести к "метанию" фокуса по форме при нажатии клавиши <Tab>.

Для того чтобы задать новый порядок перебора, пользователь может щелкнуть на каждом элементе управления. Закончив, следует нажать клавишу <Esc> и скрыть порядок перебора с экрана.

Если на одной и той же форме несколько элементов управления имеют одно и то же значение свойства `TabIndex`, то для определения порядка перебора элементов управления при нажатии клавиши <Tab> используется *z-порядок*. Этот порядок определяет перекрытие элементов управления на форме вдоль оси z (в глубину) и обычно вступает в силу, только если элементы управления должны накрывать друг друга.

Z-порядок элементов управления можно изменить с помощью команд Bring to Front и Send to Back в меню Format⇨Order.

Фиксация дизайна элементов управления

Если пользователь доволен дизайном формы, то он может применять внесенные изменения к разным элементам управления и их свойствам. Однако в процессе выбора элементов управления на форме пользователь может нечаянно сдвинуть их с правильного положения, особенно, если при этом не используются методы сноса на сетку или если на форме расположено много элементов управления, которые необходимо выровнять относительно друг друга.

К счастью, в системе Visual Studio 2010 предусмотрено решение этой проблемы в виде команды Lock Controls, доступной в меню Format. При блокировке элемента управления пользователь может изменять его свойства, но не может использовать мышь для перемещения или изменения его размеров, а также изменения самой формы.

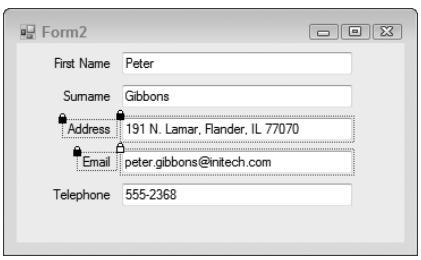


Рис. 17.11

Местоположение элементов управления может быть по-прежнему изменено с помощью сетки Properties.

На рис. 17.11 показано, что при выполнении команды Lock Controls возле элементов управления появляются небольшие пиктограммы в виде висячего замка.



Заблокировать элементы управления по отдельности можно в окне Properties, присвоив свойству Locked значение True.

Установка свойств элементов управления

Свойства элементов управления задаются с помощью окна Properties, точно так же, как это происходит при настройке форм. Кроме обычных текстовых свойств, в системе Visual Studio 2010 существует множество других свойств, которые можно изменять с помощью редактора, позволяющего выбирать значения только из ограниченного допустимого подмножества.

Многие сложные свойства имеют набор подчиненных свойств, которые можно настроить отдельно, развернув соответствующий пункт в окне Properties. На рис. 17.12, *слева*, показано окно Properties для элемента управления Label, у которого развернуто свойство Font, чтобы показать его доступные подчиненные свойства.

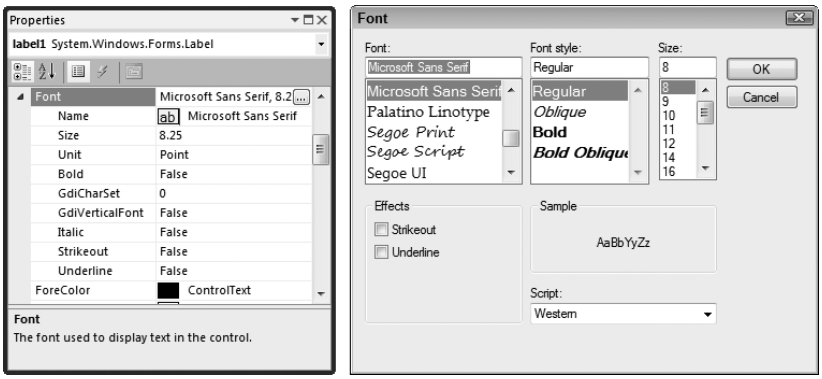


Рис. 17.12

Кроме того, многие свойства допускают расширенное редактирование, как, например, свойство Font. На рис. 17.12, *справа*, показано, что после щелчка на кнопке расширенного редактора в поле свойства `Font` открывается диалоговое окно Font.

Некоторые расширенные редакторы вызывают полноценных мастеров, как, например, редактор свойства Data Connection у некоторых компонентов, связанных с данными, а другие используют встроенные средства редактирования. Примером может служить свойство `Dock`, для которого можно выбрать визуальное представление стыковки элемента с контейнером или формой.

Служебные компоненты

Как указывалось ранее в этой главе, на форму Windows Form можно добавлять два вида компонентов — имеющие визуальное представление и не имеющие такового. Служебные компоненты, такие как таймеры, диалоговые окна, подсказки и сообщения об ошибках, позволяют расширить функциональные возможности приложения.

Для того чтобы добавить служебные компоненты, их не обязательно выбирать в инструментальном окне Toolbox двойным щелчком мыши и перетаскивать на рабочую поверхность. Система Visual Studio 2010 создает под рабочей поверхностью вспомогательную область и помещает туда новые экземпляры служебных компонентов, как показано на рис. 17.13.

Для того чтобы отредактировать свойства этих компонентов, нужно найти соответствующий компонент во вспомогательной области и открыть окно Properties.

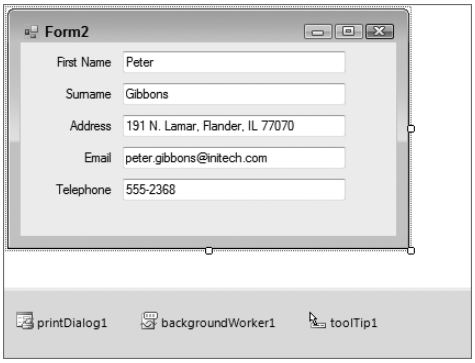


Рис. 17.13



Как известно, пользователь может создавать собственные визуальные компоненты, выполняя наследование от класса `System.Windows.Forms.Control`. Точно так же можно создавать невизуальные служебные компоненты, наследуя класс `System.ComponentModel.Component`. Фактически класс `System.ComponentModel.Component` является базовым классом для класса `System.Windows.Forms.Control` .

Интеллектуальные дескрипторы

Технология интеллектуальных дескрипторов впервые была внедрена в пакете Microsoft Office, в котором предусматривались встроенные сокращения, позволяющие быстро выполнить действия, предусмотренные для конкретного элемента. В текстовом процессоре Microsoft Word этим элементом могло быть слово или фраза, а в программе Microsoft Excel — ячейка рабочего листа. Система Visual Studio 2010 также поддерживает применение интеллектуальных дескрипторов на этапе проектирования для большого количества элементов управления.

Если выбранный элемент управления имеет интеллектуальный дескриптор, то в правом верхнем углу этого элемента отображается стрелка, направленная вправо. Щелкнув на этом дескрипторе, можно открыть меню Tasks, связанное с данным элементом управления.

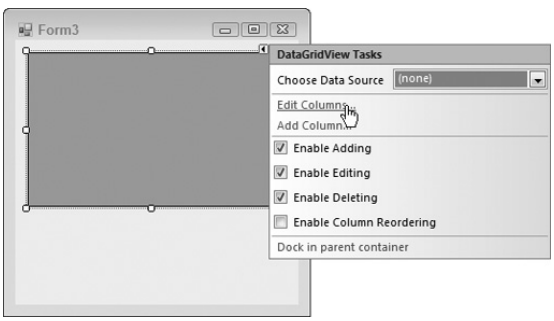


Рис. 17.14

На рис. 17.14 показаны действия, доступные для только что добавленного элемента управления DataGridView. Действия, допустимые для этого элемента, обычно соответствуют свойствам, доступным в окне Properties (например, свойству Multiline для элемента управления TextBox), но иногда они обеспечивают быстрый доступ к более сложным настройкам данного компонента.

Команды Edit Columns и Add Column (см. рис. 17.14) не перечислены в списке свойств элемента Data-

GridView, а настройки Data Source и Enable прямо соответствуют индивидуальным свойствам (например, команда Enable Adding эквивалентна свойству AllowUserToAddRows).

Контейнерные элементы управления

Некоторые элементы управления, называемые *контейнерными* (container controls), разработаны специально для того, чтобы помочь разработчику при проектировании формы и ее внешнего вида. Они не имеют собственного внешнего вида, а просто содержат другие элементы управления. Если контейнер содержит набор элементов управления, их больше не нужно перемещать по отдельности, потому что теперь можно переместить весь контейнер целиком. Используя комбинацию значений свойств Dock и Anchor, пользователь может автоматически переделывать целые разделы формы в ответ на изменение ее размеров, а также размеров элементов управления, содержащихся в контейнере.

Элементы управления Panel и SplitContainer

Элемент управления Panel используется для группировки компонентов, связанных друг с другом. Когда он помещается на форму, он может принимать любые размеры и позиционироваться в любом месте рабочей области. Поскольку этот элемент управления является контейнерным, щелчок на его границах выбирает все, что находится у него внутри. Для того чтобы его можно было перемещать, система Visual Studio 2010 размещает специальную пиктограмму в левом верхнем углу элемента управления. Щелкнув и перетащив эту пиктограмму, можно изменить местоположение элемента управления Panel.

Элемент управления SplitContainer, показанный на рис. 17.15, автоматически создает два элемента управления Panel, а затем добавляет их на форму или в другой контейнерный элемент управления. Он разделяет рабочую область на два раздела, каждым из которых можно управлять отдельно.

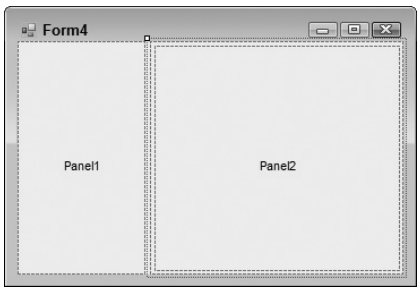


Рис. 17.15

На этапе выполнения программы пользователи могут изменять размеры этих областей, перетаскивая разделительную полосу. Элемент управления SplitContainer может быть вертикальным (см. рис. 17.15) и горизонтальным. Кроме того, он может содержаться в других элементах управления SplitContainer, образуя сложные макеты, которые конечный пользователь может легко настраивать, не заглядывая в код.



Иногда трудно выбрать реальный элемент управления, содержащий другие компоненты, как, например, в случае, когда элемент управления SplitContainer содержит два элемента управления Panel. Для того чтобы получить прямой доступ к самому элементу управления SplitContainer, пользователь может найти его в раскрывающемся списке в окне Properties или щелкнуть правой кнопкой мыши на элементах управления Panel и выбрать команду Select, соответствующую элементу SplitContainer. Это контекстное меню содержит команду Select для каждого контейнерного элемента управления из входящих в иерархию контейнеров, а также для самой формы.

Элемент управления FlowLayoutPanel

Элемент управления FlowLayoutPanel позволяет создавать проекты форм с функциональными возможностями веб-браузеров. Вместо точного позиционирования каждого элемента управления в конкретном контейнере система Visual Studio размещает каждый вновь добавляемый компонент в следующее доступное место. По умолчанию элементы управления следуют слева направо и сверху вниз, но с помощью свойства FlowDirection этот порядок можно изменить в зависимости от приложения.

На рис. 17.16 показана та же самая форма с шестью кнопками, помещенными в контейнер FlowLayoutPanel. Свойство Dock элемента FlowLayoutPanel позволяет заполнить всю рабочую поверхность формы, поэтому при изменении размера формы размеры контейнера также будут автоматически изменены. Как только форма расширяется и появляется свободное место, элементы управления перераспределяются слева направо, а потом сверху вниз, заполняя форму.

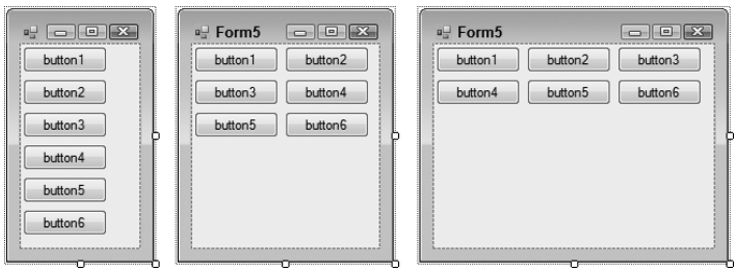


Рис. 17.16

Элемент управления TableLayoutPanel

Альтернативой ранее описанным контейнерным элементам управления является контейнер TableLayoutPanel. Этот элемент управления функционирует как таблица в редакторе Microsoft Word или типичный веб-браузер, в котором каждая ячейка представляет собой отдельный контейнер для одного элемента управления.



В отдельную ячейку невозможно непосредственно поместить несколько элементов управления. Однако в нее можно поместить другой контейнерный элемент управления, например Panel, а затем поместить требуемые компоненты в этот вложенный контейнер.

Помещение элемента управления непосредственно в ячейку приводит к его автоматическому позиционированию относительно левого верхнего угла этой ячейки. Для того чтобы отменить это свойство и разместить элемент по-своему, можно воспользоваться свойством Dock, которое обсуждается далее.

Контейнер TableLayoutPanel позволяет легко создавать структурированные макеты форм со сложными свойствами, например, с возможностью автоматического увеличения за счет добавления строк при вставке новых дочерних элементов управления.

На рис. 17.17 показана форма, на которую был добавлен элемент управления TableLayoutPanel. Затем были открыты интеллектуальные дескрипторы и выполнена команда Edit Rows and Columns. В результате открылось диалоговое окно Column and Row Styles, позволяющее настраивать индивидуальные параметры столбцов и строк. Это диалоговое окно содержит несколько подсказок для выбора макетов форм, вклю-

чая охват нескольких строк и столбцов, а также выравнивание элементов управления в ячейке. Здесь пользователь может изменить способ изменения размеров ячейки, а также добавить или удалить столбцы и строки.

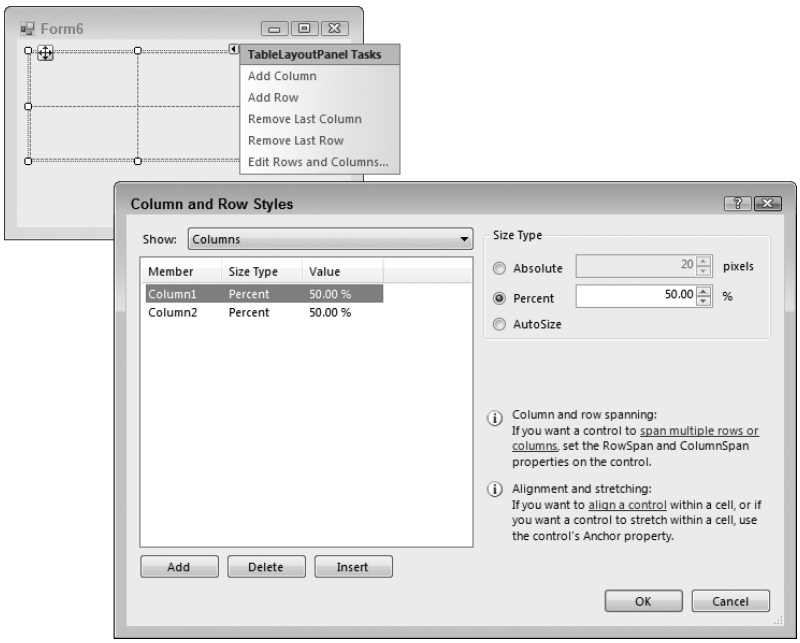


Рис. 17.17

Элементы управления стыковкой и закреплением

Недостаточно создать макет, точно настроенный на этапе проектирования. На этапе выполнения пользователь может захотеть изменить размер формы, и в идеале размеры элементов управления должны автоматически измениться в соответствии с модифицированным пространством. Самыми важными свойствами элементов управления, влияющими на эту возможность, являются Dock и Anchor. На рис. 17.18 показано, как на форме Windows Form изменяются размеры элементов управления при указании правильных значений свойств Dock и Anchor.

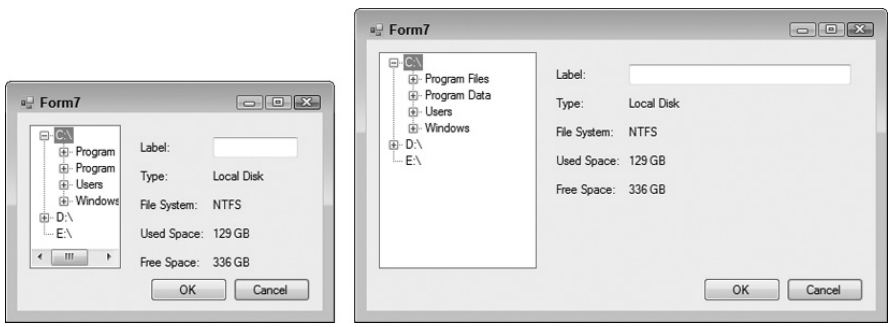


Рис. 17.18

Свойство Dock задает границы элемента управления, которые должны быть связаны с контейнером. Например, на рис. 17.18, *слева*, для свойства Dock элемента управления TreeView установлено значение `Fill`, чтобы заполнить левую панель компонента SplitContainer, что приводит к его стыковке со всеми четырьмя границами. Следовательно, независимо от того, насколько мала или велика левая граница элемента управления SplitContainer, размеры элемента управления TreeView будут всегда автоматически изменяться так, чтобы заполнять все свободное пространство.

Свойство Anchor определяет края контейнера, к которым должен привязываться элемент управления. На рис. 17.18, *слева*, в правом нижнем углу формы закреплены две кнопки. Как показано на рис. 17.18, *справа*, при изменении размеров формы расстояние от кнопок до правого нижнего угла остается постоянным. Аналогично элемент управления TextBox закреплен слева и справа, т.е. будет автоматически сужаться и расширяться при изменении размеров формы.

Резюме

В главе продемонстрировано, как система Visual Studio может помочь быстро разработать макет приложений Windows Forms. Разнообразные элементы управления и их свойства позволяют быстро и легко создавать сложные формы, обеспечивающие взаимодействие с пользователем самыми разными способами. В последующих главах будет описана специфика проектирования пользовательских интерфейсов для других платформ, включая надстройки для пакета Office, а также приложения WPF, Web и Silverlight.

18

Подсистема Windows Presentation Foundation (WPF)

В ЭТОЙ ГЛАВЕ...

- Основы языка XAML
- Создание WPF-приложения
- Настройка стиля WPF-приложения
- Внедрение WPF-контента в проект Windows Forms
- Внедрение контента проекта Windows Forms в WPF-проект
- Использование визуализатора WPF

Начиная разработку нового клиентского приложения для системы Windows с помощью интегрированной среды Visual Studio, пользователь может выбрать одну из двух технологий: Windows Forms или Windows Presentation Foundation (WPF). Это совершенно разные интерфейсы прикладного программирования для управления слоем презентации пользовательского приложения. Технология WPF чрезвычайно мощная и гибкая. Она была разработана для преодоления недостатков и ограничений, присущих технологии Windows Forms. Во многих аспектах технология WPF является преемником Windows Forms. Однако овладение этой мощной и гибкой подсистемой требует от разработчика больших усилий, поскольку она заметно отличается от своего предшественника.

В главе описывается процесс создания простого WPF-приложения с помощью интегрированной среды Visual Studio 2010. Подробное описание технологии WPF выходит за рамки рассмотрения данной книги, так как для этого потребовалось бы слишком много места. Вместо этого мы приводим обзор возможностей системы Visual Studio 2010, помогающих быстро создавать пользовательские интерфейсы с помощью языка XAML.

Что такое WPF

Windows Presentation Foundation — это презентационный функциональный слой для системы Windows. Что же делает его уникальным и почему его следует считать достойной альтернативой подсистеме Windows Forms? В то время, как подсистема Windows Forms для рендеринга использует растровые механизмы GDI/GDI+, подсистема WPF имеет свой собственный векторный механизм рендеринга. По этой причине она не создает окна и элементы управления стандартным способом и в стандартном виде, присущем системе Windows. Технология WPF радикально отличается от технологии Windows Forms. В графической подсистеме Windows Forms разработчик обычно определял пользовательский интерфейс с помощью визуального конструктора, автоматически генерируя код (на языке проекта) в файле с расширением `.designer`. Таким образом, по существу, пользовательский интерфейс определялся и управлялся кодом на языке C# или VB. В то же время пользовательский интерфейс, созданный по технологии WPF, фактически определяется языком разметки Extensible Application Markup Language, который обычно называют XAML (произносится как "zammel"). Этот язык основан на языке XML и специально разработан компанией Microsoft для использования в системе WPF. Именно язык XAML, лежащий в основе технологии WPF, обеспечивает ее мощь и гибкость, позволяя разрабатывать намного более богатые и особенные пользовательские интерфейсы по сравнению с технологией Windows Forms. Определение пользовательского интерфейса с помощью языка XAML является одинаковым для любого языка проекта. По этой причине наряду с новыми возможностями управления пользовательским интерфейсом появилось большое количество новых вспомогательных концепций, влияющих на код. Например, возникли свойства зависимостей, значениями которых являются выражения, что часто требуется во многих сценариях связывания для поддержки сложных возможностей связывания, предоставляемых языком XAML. Тем не менее код в WPF-приложении почти не отличается от кода стандартного приложения, созданного по технологии Windows Forms, — основное внимание разработчику следует уделить языку XAML.

При разработке WPF-приложений необходимо думать совершенно иначе, чем при разработке приложений по технологии Windows Forms. Разработчик должен сконцентрировать свое внимание на преимуществах новых возможностей связывания, предоставляемых языком XAML, и рассматривать свой код не как контроллер пользовательского интерфейса, а как обслуживающий механизм. Теперь код не должен "проталкивать" данные в пользовательский интерфейс и говорить ему, что делать. Вместо этого пользовательский интерфейс должен спрашивать у кода, что делать, и посылать ему запросы на данные (т.е. "выталкивать" их из него). Разница почти незаметна, но способ определения презентационного слоя приложения изменяется радикально. Представьте себе, что пользовательский интерфейс теперь начальник. Код может (и должен) принимать решения, но больше не должен инициировать действия.

Это "новое мышление" порождает новые шаблоны проектирования, описывающие взаимодействие кода и элементов пользовательского интерфейса, такие как популярный шаблон Model-View-ViewModel (MVVM), позволяющий намного лучше проводить модульное тестирование кода, обслуживающего пользовательский интерфейс и поддерживающего явное разделение между элементами проектировщика и разработчика в рамках проекта. В результате изменяется способ написания кода, и в итоге — способ проектирования приложения. Такое явное разделение упрощает работу проектировщика и разработчика, позволяя проектировщику работать с помощью

программы Expression Blend над той же частью проекта, над которой работает разработчик (с помощью системы Visual Studio), и при этом избегать столкновения.

Благодаря гибкости языка XAML система WPF позволяет разрабатывать уникальные пользовательские интерфейсы и способы *интерактивного взаимодействия* (user experiences). В их основе лежат стили и шаблонные функциональные возможности системы WPF, которая отделяет внешний вид элементов управления от их поведения. Это позволяет легко изменять внешний вид элементов управления без изменения самого элемента.

Хорошая новость заключается в том, что благодаря применению языка XAML технология WPF реализует лучший способ определения пользовательских интерфейсов по сравнению с Windows Forms, а также порождает большое количество дополнительных понятий. Плохая новость состоит в том, что эта гибкость и мощь языка XAML требует значительного времени для обучения даже опытных разработчиков. У разработчиков, достигших приемлемого уровня производительности труда с помощью технологии Windows Forms, система WPF несомненно вызовет чувство неудовлетворения, пока они не освоят новые концепции и не испытают реальной потребности изменить свой образ мышления. Многие простые задачи вначале выглядят намного более сложными, чем они могли бы быть, если их реализовать с помощью технологии Windows Forms. Однако со временем разработчики оценят преимущества новых возможностей, которые им предоставляют технология WPF и язык XAML. Поскольку графическая система Silverlight имеет много общего с системой WPF (обе основаны на языке XAML, так что Silverlight, по существу, — это подсистема системы WPF), изучение технологии WPF позволит разработчиками одновременно научиться создавать приложения с помощью системы Silverlight.



Если читатели уже знакомы с более ранними версиями системы WPF (поставляемыми с платформами .NET Framework 3.0 и 3.5), то они могли заметить, что текст, прорисованный в системе WPF, часто выглядел слегка расплывчатым, а не четким и контрастным. Это приводило к многочисленным жалобам разработчиков. К счастью, в версии .NET Framework 4.0 прорисовка текста была значительно улучшена, и теперь разочарованные разработчики могут вернуться к использованию этой технологии. Компания Microsoft продемонстрировала свою приверженность технологии WPF, переписав с ее помощью редактор кода в среде Visual Studio 2010, чтобы показать ее мощь и гибкость.

Начало работы с системой WPF

Открыв диалоговое окно New Project, разработчик видит множество встроенных шаблонов проектов для технологии WPF, поставляемых вместе с системой Visual Studio 2010: WPF Application, WPF Browser Application, WPF Custom Control Library и WPF User Control Library (рис. 18.1).

Обратите внимание на то, что эти проекты большей частью являются эквивалентами проектов для графической системы Windows Forms. Исключением является шаблон WPF Browser Application, генерирующий XBAP-файл, использующий браузер как контейнер для пользовательского приложения “толстого” клиента (это похоже на то, что делает программа Silverlight, за исключением того, что XBAP-приложение предназначено для полноценной платформы .NET Framework, которая должна быть инсталлирована на компьютере клиента).

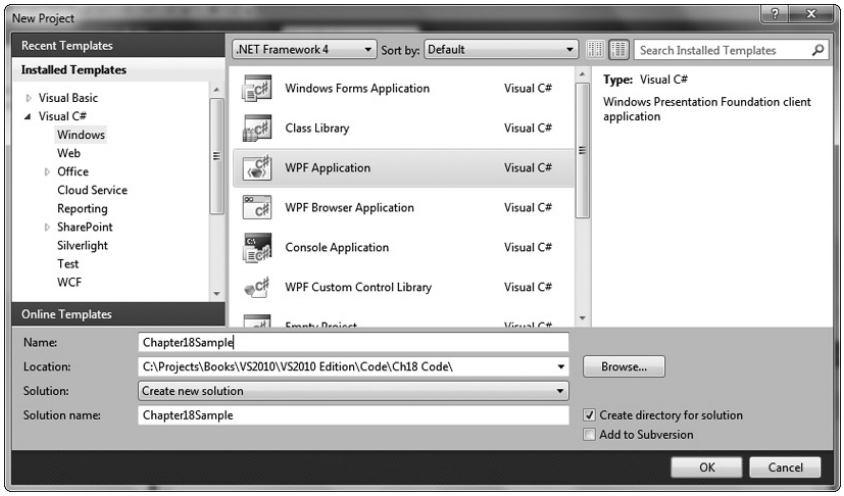


Рис. 18.1

В качестве примера создадим проект, используя шаблон WPF Application, хотя большинство функциональных возможностей системы Visual Studio 2010, обсуждаемых здесь, в одинаковой степени относится и к другим типам проектов. Генерируемая структура проекта выглядит так, как показано на рис. 18.2.

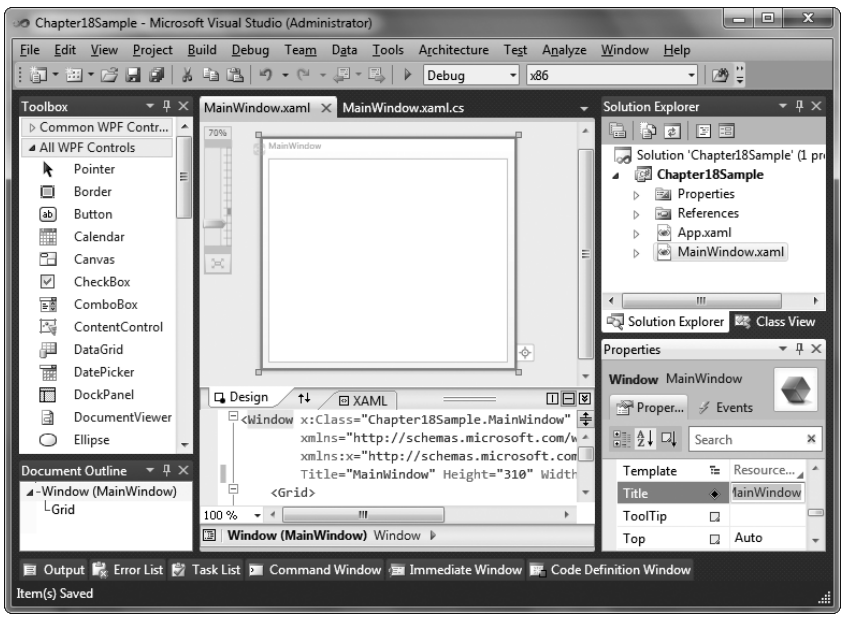


Рис. 18.2

Здесь видно, что структура проекта состоит из файлов `App.xaml` и `MainWindow.xaml`, каждому из которых соответствует свой исходный файл (`.cs` или `.vb`), которые можно увидеть, развернув соответствующие узлы проекта. На этом этапе файл `App.xaml`

содержит элемент Application XAML, имеющий атрибут `StartupUri`, который используется для определения первоначального загружаемого XAML-файла (по умолчанию — `MainWindow.xaml`). В технологии Windows Forms он соответствует начальной форме. Итак, если мы захотим изменить имена файла `MainWindow.xaml` и его соответствующего класса на более информативные, то должны будем выполнить следующие действия.

- Изменить имя файла с расширением `.xaml`. Исходный файл будет переименован автоматически и согласованно.
- Изменить имя класса в исходном файле вместе с именем конструктора, а также значение атрибута `x:Class` элемента Window в файле с расширением `.xaml`, для того, чтобы сослаться на новое имя класса (полностью квалифицированное своим пространством имен). Обратите внимание на то, что два последних этапа выполняются автоматически, если сначала изменить имя класса в исходном файле и использовать интеллектуальные дескрипторы, которые появляются после этого, для того, чтобы переименовать объект во всех местах, где на него есть ссылки.
- Изменить атрибут `StartupUri` элемента Application в файле `App.xaml` и указать новое имя файла с расширением `.xaml` (поскольку он является объектом запуска).

Как видим, для переименования файла в WPF-проекте требуется внести больше изменений, чем в стандартном проекте Windows Forms, однако эти изменения вполне очевидны, если вы знаете, что делаете (и используете интеллектуальные дескрипторы, чтобы уменьшить количество требуемых действий).

Изучая окна системы Visual Studio на рис. 18.2, можно увидеть, что уже известное нам инструментальное окно Toolbox, расположенное вдоль левого края экрана, содержит элементы управления WPF, похожие на элементы управления, которые используются при создании приложений по технологии Windows Forms. Под этим окном, все там же слева, расположено инструментальное окно Document Outline. И в проектах Windows Forms, и в проектах Web Applications оно содержит иерархическое представление элементов текущего окна. Выбрав один из этих узлов, можно выделить соответствующий элемент управления в основном окне редактора. Это упрощает навигацию по более сложным документам. Интересной особенностью окна Document Outline при работе с системой WPF является тот факт, что если курсор мыши "зависает" над элементом, то пользователь может увидеть его миниатюрное изображение. Это помогает идентифицировать выбираемый элемент управления.



Инструментальное окно Document Outline можно свернуть у одного из краев рабочей области Visual Studio. И наоборот, его можно открыть, выбрав команду View⇨Other Windows.

На рис. 18.2, *справа*, показано инструментальное окно Properties. И внешним видом, и функционированием оно очень похоже на инструментальное окно Properties в конструкторе форм в системе Windows Forms. Однако на самом деле это окно относится к конструктору системы WPF и имеет дополнительные возможности для редактирования окон и элементов управления в этой графической системе. Посередине экрана расположена основная область редактирования и предварительного просмотра, которая в данный момент разделена, чтобы продемонстрировать как визуальный макет окна (в верхней части), так и код на языке XAML, который этот макет определяет (в нижней части).

Основы языка XAML

Если читатели знакомы с языком XML (или хотя бы с языком HTML), то они заметят, что синтаксис языка XAML относительно прост, поскольку он основан на языке XML. Язык XAML имеет только один корневой узел, и элементы вкладываются один в другой, определяя внешний вид и содержимое пользовательского интерфейса. Каждому элементу языка XAML соответствует класс платформы .NET, а именам атрибутов — свойства/события этого класса. Отметим, что имена элементов и атрибутов чувствительны к регистру.

Посмотрим на XAML-файл, по умолчанию созданный для класса `MainWindow`.

```
<Window x:Class="Chapter18Sample.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="300" Width="300">
    <Grid>

        </Grid>
</Window>
```

Здесь `Window` — корневой узел, а `Grid` — элемент, который в нем содержится. Для того чтобы понять их смысл, следует поразмышлять о них в терминах "окна, содержащего сетку". Корневому узлу соответствует класс, определенный атрибутом `x:Class` и содержащий объявления префиксов пространств имен (вскоре мы вернемся к этой теме), а также некоторые атрибуты, используемые для задания значений свойств (`Title`, `Height` и `Width`) класса `Window`. Значения всех атрибутов (независимо от типа) должны быть заключены в кавычки.

В корневом узле определены два префикса пространств имен. Оба эти префикса объявлены с помощью атрибута `xmlns` (XML-атрибута, используемого для объявления пространств имен). Объявления префиксов пространств имен в языке XAML можно рассматривать как аналоги инструкций `using`/`Imports` в начале классов в языках C#/VB, но без кавычек.

Эти объявления присваивают пространствам имен уникальные префиксы, используемые в XAML-файле, а также префиксы, используемые для квалификации пространств имен при ссылке на классы, находящиеся в них (т.е. для задания местоположения классов). Префиксы сокращают многословность языка XAML, позволяя при ссылке на класс в файле XAML указывать только префикс, а не все пространство имен. Префикс определяется сразу после двоеточия в выражении `xmlns`. Первое определение на самом деле не задает префикс, потому что оно определяет пространство имен, заданное по умолчанию (пространство имен системы WPF). В то же время второе объявление задает `x` как префикс пространства имен языка XAML. Оба определения отображаются в унифицированные идентификаторы ресурса URI, а не в конкретные пространства имен, поскольку они представляют собой консолидированные пространства имен (т.е. охватывают несколько пространств имен) и поэтому ссылаются на единый идентификатор URI, используемый для определения данной консолидации. Однако разработчику не следует беспокоиться об этом — нужно просто оставить эти определения как есть и добавлять свои определения после них. Собственные определения пространств имен, добавляемые разработчиком, почти всегда начинаются с ключевого слова `clr-namespace`, которое ссылается на пространство имен `CLR` и содержащую его сборку. Рассмотрим пример.

```
xmlns:wpf="clr-namespace:Microsoft.Windows.Controls;assemblty=WPFToolkit"
```

Префиксы могут быть любыми, но лучше всего делать их по возможности короткими. Как правило, пространства имен определяются в корневом узле XAML-файла. Это не обязательно, поскольку префикс пространства имен может быть определен на любом уровне в XAML-файле. И все же стандартной практикой стало собирать префиксы вместе в корневом узле, чтобы за ними было легче следить.

При необходимости сослаться на элемент управления в коде или связать его с другим элементом управления в XAML-файле (например, при связывании элемента ElementName) нужно приписать этому элементу управления какое-то имя. Многие элементы управления используют для этой цели свойство Name, но присвоить имя элементу управления можно с помощью атрибута `x:Name`. Он определен в пространстве имен XAML (поэтому имеет префикс `x:`) и может применяться к любому элементу управления. Если свойство Name реализовано (чаще всего так и есть, потому что это свойство определено в базовых классах, которые наследуются большинством элементов управления), этот атрибут просто отображается в него и служит той же цели. Рассмотрим пример. Инструкция

```
<Button x:Name="OKButton" Content="OK" />
```

эквивалентна инструкции

```
<Button Name="OKButton" Content="OK" />
```

Оба варианта являются правильными с формальной точки зрения (хотя в приложении Silverlight большинство элементов управления не поддерживают атрибут `Name` и должны использовать вместо него атрибут `x:Name`). Как только эти свойства заданы, генерируется поле (в автоматически генерируемом коде, который разработчику недоступен), которое можно использовать для ссылки на элемент управления.

Элементы управления системы WPF

Система WPF содержит множество элементов управления, предназначенных для использования в пользовательском интерфейсе. Эти элементы управления примерно соответствуют стандартным элементам управления в системе Windows Forms. Если взглянуть на предыдущие версии системы WPF, то можно заметить много элементов управления (таких как Calendar, DatePicker, DataGrid и др.), которые были включены в стандартный набор для системы Windows Forms, но не вошли в стандартный набор для системы WPF. Для того чтобы воспользоваться этими элементами управления, пользователь должен подключить свободно распространяемый пакет WPF Toolkit, размещенный на сайте CodePlex. Этот пакет был разработан компанией Microsoft, чтобы компенсировать указанный недостаток первой версии системы WPF за счет недостающих компонентов. Однако с появлением системы WPF 4.0 многие элементы управления из пакета WPF Toolkit уже вошли в стандартный набор. Разумеется, пользователь может использовать пакеты элементов управления, созданные другими разработчиками в тех случаях, когда стандартных средств недостаточно, но в настоящее время для этого нужны веские основания.

Несмотря на то что набор элементов управления в системе WPF сопоставим с набором элементов управления в системе Windows Forms, по своим свойствам они сильно отличаются от своих аналогов. Например, у многих элементов управления больше нет свойства Text, а вместо него появилось свойство Content, которое используется для присвоения элементу управления определенного содержимого (а следовательно, и имени). В большинстве случаев это свойство можно интерпретировать как прямой аналог свойства Text в элементах управления системы Windows Forms и просто присваивать ему какую-то строку, которая должна прорисовываться на экране. Однако на самом деле свойство Content может принимать не только текстовое значение, но и

любой элемент системы WPF. Это открывает практически безграничные возможности для настройки существующих элементов управления и освобождает пользователя от необходимости создавать свои собственные элементы управления. При разработке сложных пользовательских интерфейсов это очень полезно. Кроме того, многие элементы управления больше не имеют удобных свойств, которые существовали в системе Windows Forms. Это может показаться несколько странным. Например, у элемента управления Button в системе WPF нет свойства Image, позволявшего присваивать кнопке определенное изображение, как это было в системе Windows Forms. На первый взгляд, это сильно ограничивает возможности системы WPF, но это впечатление ошибочно, поскольку теперь у кнопки есть свойство Content. Так как свойство Content позволяет присваивать элементу управления системы WPF определенное содержимое, пользователь может присвоить ему элемент StackPanel (который будет рассмотрен в следующем разделе), содержащий как элемент управления Image, так и элемент управления TextBlock, обеспечивающие тот же самый эффект. Это требует от пользователя чуть больше усилий, чем при работе с системой Windows Forms, но позволяет ему проще моделировать содержимое кнопки на любой форме (а не подбирать элементы управления, которые он может реализовать) и обеспечивает невероятную гибкость системы WPF и языка XAML. Код на языке XAML для кнопки, показанной на рис. 18.3, имеет следующий вид.

```
<Button HorizontalAlignment="Left" VerticalAlignment="Top" Width="100"
 Height="30">
    <Button.Content>
         <StackPanel Orientation="Horizontal">
              <Image Source="/Chapter18Sample;component/Images/save.png"
               Width="16"
               Height="16" />
              <TextBlock Margin="5,0,0,0" Text="Save"
VerticalAlignment="Center" />
         </StackPanel>
    </Button.Content>
</Button>
```



Рис. 18.3

Другими замечательными свойствами, которые отличаются от свойств системы Windows Forms, являются свойства IsEnabled (которое в системе Windows Forms называлось Enabled) и Visibility (которое в системе Windows Forms называлось Visible). На примере свойства IsEnabled видно, что имена большинства булевых свойств имеют префикс Is (например, IsTabStop, IsHitTestVisible и т.д.), что соответствует стандартной схеме именования. Однако свойство Visibility больше не имеет булевого значения — теперь оно представляет собой перечисление, принимающее значение Visible, Hidden или Collapsed.



Не забывайте о пакете WPF Toolkit, доступном на веб-сайте `http://wpf.codeplex.com`, поскольку в него постоянно включаются новые элементы управления для системы WPF, которые могут оказаться полезными.

Компоновочные элементы управления в системе WPF

В системе Windows Forms для размещения элементов управления на рабочей поверхности использовались абсолютные координаты (т.е. каждый элемент управления имел явно заданные координаты *x* и *y*). Со временем появились элементы управления TableLayoutPanel и FlowLayoutPanel, которые могут содержать другие элементы управления. Это позволило создавать более сложные схемы размещения элементов

управления на форме. Однако концепция позиционирования элементов управления в системе WPF немного отличается от системы Windows Forms. Наряду с элементами управления, имеющими специальное предназначение (например, кнопками, полями ввода и т.п.), в системе WPF есть множество элементов управления, используемых специально для определения компоновки пользовательского интерфейса.

Компоновочные элементы управления являются невидимыми и предназначены для позиционирования других элементов управления на своей поверхности. В системе WPF нет поверхности, по умолчанию предназначенной для позиционирования элементов управления, — поверхность, на которой работает пользователь, определяется компоновочными элементами управления, образующими иерархию. Компоновочные элементы управления, как правило, находятся в этой иерархии непосредственно под корневым узлом XAML-файла и определяют метод компоновки, принятый по умолчанию для данного XAML-файла. Наиболее важными компоновочными элементами управления в системе WPF являются Grid, Canvas и StackPanel, поэтому мы рассмотрим каждый из них. Например, в XAML-файле, созданном по умолчанию для класса MainWindow, рассмотренного ранее, элемент Grid располагался непосредственно под корневым узлом Window, а значит, должен был по умолчанию служить для этого окна рабочей поверхностью. Разумеется, пользователь может заменить любой компоновочный элемент управления, а также использовать любые дополнительные элементы управления для создания дополнительных поверхностей и иного размещения элементов управления.

В следующем разделе мы рассмотрим, как скомпоновать форму с помощью рабочей поверхности конструктора, но сначала заглянем в код XAML, чтобы увидеть, как используются элементы управления.

Если пользователь хочет размещать элементы управления, используя абсолютную систему координат (как в системе Windows Forms), то он может в качестве поверхности выбрать элемент управления Canvas, предусмотренный в системе WPF.

Определение элемента Canvas в языке XAML выглядит очень просто.

```
<Canvas>

    </Canvas>
```

Для того чтобы разместить элемент управления (например, TextBox) на этой поверхности, используя координаты *x* и *y* (по отношению к левому верхнему углу элемента Canvas), необходимо использовать концепцию *присоединенных свойств* (attached properties), принятую в языке XAML. Элемент управления TextBox на самом деле не имеет свойств, определяющих его местоположение, поскольку его позиция в компоновочном элементе управления полностью зависит от типа этого элемента. Следовательно, свойства, которых недостает элементу управления TextBox, чтобы определить его положение в компоновочном элементе, должны быть позаимствованы у самого компоновочного элемента, поскольку именно он определяет, где в нем должен находиться элемент TextBox. Именно в этот момент на первый план выходят присоединенные свойства. Коротко говоря, присоединенными называют свойства, которые присваивают элементу управления какое-то значение, но определяются и принадлежат другому элементу управления, стоящему выше в иерархии элементов. При использовании такого свойства его имя уточняется именем элемента управления, в котором оно на самом деле определено, за которым следует точка, а затем имя элемента управления, в котором оно используется (например, Canvas.Left). Присвоив это значение другому элементу управления, который содержится внутри (как, например, наше поле ввода TextBox), элемент Canvas хранит это значение в себе и с его помощью управляет позицией поля ввода. Например, следующий фрагмент кода на языке тре-

бует разместить поле ввода в точке 15, 10, используя свойства Left и Top, определенные в элементе управления Canvas.

```
<Canvas>
     <TextBox Text="Hello" Canvas.Left="15" Canvas.Top="10" />
</Canvas>
```

В то время как абсолютное позиционирование в системе Windows Forms принято по умолчанию, в системе WPF для компоновки лучше всего использовать элемент управления Grid. Элемент управления Canvas следует использовать редко и только при крайней необходимости, поскольку элемент управления Grid представляет собой намного более мощное средство для определения компоновки форм и лучше работает в большинстве сценариев. Одним из самых заметных преимуществ элемента управления Grid является возможность автоматически изменять размер его содержимого при изменении его собственного размера. Поэтому пользователь может легко проектировать формы, автоматически изменяющие свои размеры, чтобы заполнить всю доступную область,— иначе говоря, размер и положение элементов управления в этой форме определяются динамически. Элемент управления Grid позволяет разделять свою область на подобласти (ячейки), в которые можно поместить другие элементы управления. Эти ячейки определяются путем задания строк и столбцов сетки с помощью значений свойств RowDefinitions и ColumnDefinitions. Пересечения строк и столбцов становятся ячейками, в которые можно помещать элементы управления.

Для определения строк и столбцов пользователь должен знать только, как определять сложные значения в языке XAML. До сих пор мы могли присваивать элементам управления лишь простые значения, которые отображались либо в элементарные типы данных платформы .NET, либо в имя элемента перечисления. Кроме того, мы могли конвертировать строку в соответствующий объект. Значения этих простых свойств использовались как атрибуты при определении элемента управления. Однако сложные значения нельзя присваивать таким способом, потому что они отображаются в объекты (т.е. необходимо выполнить присваивание нескольких свойство объекта) и должны определяться с помощью *синтаксиса "свойство–элемент"* (property element syntax). Поскольку свойства RowDefinitions и ColumnDefinitions элемента Grid представляют собой коллекции, они принимают сложные значения, которые должны быть определены с помощью синтаксиса "свойство–элемент". Например, рассмотрим сетку, состоящую из двух строк и трех столбцов, определенных с помощью синтаксиса "свойство–элемент".

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="100" />
        <ColumnDefinition Width="150" />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
</Grid>
```

Для того чтобы задать свойство RowDefinitions с помощью синтаксиса "свойство–элемент", необходимо создать и определить дочерний элемент компонента Grid. Ставя имя Grid перед именем свойства, мы указываем, что оно принадлежит элементу управления, который относится к более высокому уровню иерархии (как и в случае присоединенных свойств), а задание свойства в виде элемента XAML-файла означает, что мы присваиваем сложное значение указанному свойству элемента управления Grid.

Свойство RowDefinitions получает коллекцию объектов RowDefinitions. Таким образом мы создаем несколько экземпляров объектов RowDefinition, образующих эту

коллекцию. Соответственно свойству ColumnDefinitions присваивается коллекция объектов ColumnDefinition. Для того чтобы продемонстрировать, что ColumnDefinition (как и RowDefinition) — это действительно объект, свойство Width объекта ColumnDefinition было задано в двух первых строках определения столбца.

Для того чтобы поместить элемент управления в заданную ячейку, необходимо снова использовать присоединенные свойства, на этот раз сообщив контейнеру, какие столбец и строку он должен разместить внутри себя.

```
<CheckBox Grid.Column="0" Grid.Row="1" Content="A check box" IsChecked="True" />
```

Еще одним важным контейнерным элементом управления, используемым для компоновки, является StackPanel. Он расставляет элементы управления, содержащиеся в нем, в горизонтальном или вертикальном направлении (в зависимости от значения своего свойства Orientation). Например, если в одной и той же ячейке сетки определены две кнопки (без элемента StackPanel), сетка может разместить вторую кнопку прямо поверх первой. Если же поместить эти кнопки в элемент управления StackPanel, то он выровняет их и разместит одну после другой.

```
<StackPanel Orientation="Horizontal">
    <Button Content="OK" Height="23" Width="75" />
    <Button Content="Cancel" Height="23" Width="75" Margin="10,0,0,0" />
</StackPanel>
```

Конструктор WPF и редактор XAML

По сравнению с версией Visual Studio 2008 конструктор WPF и редактор XAML приобрели множество усовершенствований, включая повышенную надежность (конструктор WPF в системе Visual Studio 2008 очень неустойчив), и, что самое важное, теперь конструктор поддерживает связывание с помощью метода “перетащить и опустить”.

Конструктор WPF по внешнему виду напоминает конструктор Windows Form, но поддерживает больше уникальных свойств. Для того чтобы ближе ознакомить с ними читателей, на рис. 18.4 мы выделили это окно, чтобы рассмотреть его компоненты более подробно.

Во-первых, окно разделено на область визуального конструктора вверху и окно для редактирования кода внизу. Если пользователь предпочитает другое представление, он может щелкнуть на пиктограмме со стрелками, направленными вверх и вниз, которая расположена между корешками вкладок Design и XAML. Вторая пиктограмма на рис. 18.4, *справа*, подсвечена. Это означает, что окно разделено горизонтально. Выбрав пиктограмму, расположенную слева от нее, можно разделить окно по вертикали.



Работать с конструктором WPF в режиме разделения экрана очень удобно, потому что у пользователя есть возможность непосредственно и регулярно модифицировать код XAML, без проблем используя конструктор для решения общих задач.

Если же пользователь не хочет работать в режиме разделения экрана, то он может дважды щелкнуть на корешках вкладок Design или XAML. В этом случае соответствующая вкладка полностью заполнит окно редактирования, как показано на рис. 18.5, и для переключения между ними пользователь сможет поочередно щелкать на корешках вкладок. Для того чтобы вернуться в режим разделения экрана, нужно щелкнуть на пиктограмме Expand Pane, которая расположена на панели разделения правее всех.

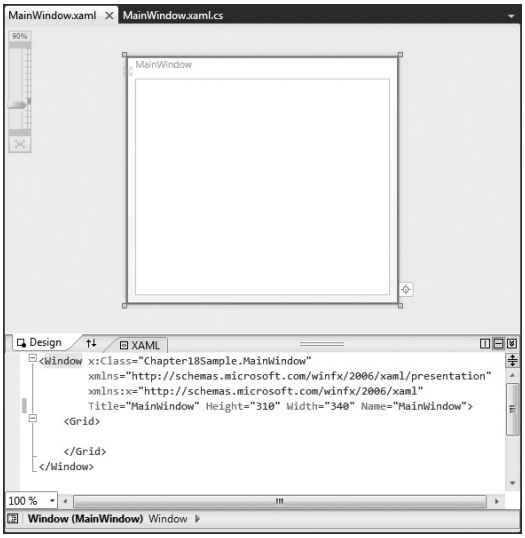


Рис. 18.4



Рис. 18.5

В области конструктора WPF можно заметить элемент управления для изменения масштаба изображения. Он позволяет легко увеличивать или уменьшать масштаб окна или редактируемого элемента управления, что очень удобно при кропотливой настройке мелких деталей, а также для получения общего представления о всем XAML-макете. В данном случае масштаб экрана уменьшен до 90%. Кроме того, на шкале масштабирования существует метка, соответствующая 100%-ному представлению, а также кнопка внизу экрана, позволяющая легко изменять размер XAML-макета, чтобы разворачивать его на всю область конструктора или, наоборот, сворачивать.

Последняя деталь, заслуживающая упоминания, — крошечный трекер, расположенный внизу окна визуального конструктора справа от корешков вкладок Design и XAML. В данном случае форма содержит только один элемент Window, но при добавлении новых элементов в окно этот трекер станет очень полезным для поиска требуемого элемента и перемещения по иерархии элементов управления.

Работа с редактором XAML

Работа с редактором XAML чем-то похожа на работу с редактором HTML в системе Visual Studio. По сравнению с версией Visual Studio 2008 в систему внесено много усовершенствований технологии IntelliSense. Благодаря этому работа с редактором XAML стала очень быстрой и удобной.

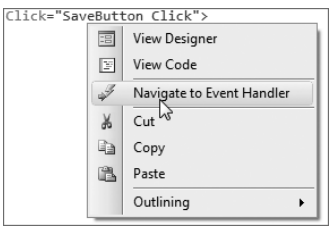


Рис. 18.6

Одной из важнейших особенностей редактора XAML является возможность легко переходить к обработчику события, как только он был назначен для элемента управления. Для этого достаточно щелкнуть правой кнопкой мыши на инструкции присваивания обработчика события в коде XAML и выбрать команду Navigate to Event Handler из раскрывшегося меню, как показано на рис. 18.6.

Работа с конструктором WPF

Несмотря на то что разработчик все же должен владеть языком XAML и уметь работать с редактором XAML, система VS2010 имеет очень хороший конструктор WPF, сравнимый с конструктором Windows Forms, а в некоторых отношениях даже превосходящий его. В этом разделе мы рассмотрим некоторые свойства конструктора WPF.

На рис. 18.7 показаны линии разметки, ориентиры и глифы, которые появляются на экране, когда пользователь выбирает, перемещает или изменяет размер элемента управления. Обратите внимание на глиф, расположенный в правом нижнем углу первого изображения, из представленных на рис. 18.7. Щелкнув на нем, можно легко переключаться из режима, в котором окна имеют фиксированную ширину и высоту, в режим, в котором окна автоматически изменяют размеры, чтобы разместить все свое содержимое. После щелчка глиф изменяется (это означает, что включился режим изменения размеров), а для свойства SizeToContent устанавливается соответствующее значение.

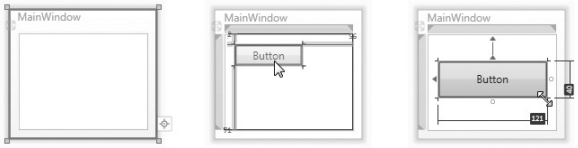


Рис. 18.7

Щелчок на глифе возвращает окно в режим фиксированных размеров. Эта возможность существует только в корневом узле.



Когда пользователь щелкает на глифе, чтобы окно изменило свой размер и вместило содержимое, размер окна не изменяется, потому что его свойства Height и Width заменяются соответствующими свойствами конструктора, так что свойство SizeToContent на проектирование формы не влияет. При возврате в режим фиксированных размеров эти свойства заменяются стандартными свойствами Height и Width.

Второе изображение на рис. 18.7 демонстрирует линии разметки, появляющиеся, когда пользователь перемещает элемент управления по форме (или изменяет его размеры). Аналогичные линии разметки есть в конструкторе Windows Forms. Они помогают выравнивать элементы управления относительно друг друга, а также относительно границ контейнерного элемента управления. Если пользователь не хочет, чтобы отображались линии разметки, но в то же время желает сохранить возможность выравнивания элементов, то при перемещении элементов управления следует удерживать нажатой клавишу <Alt>.

Третье изображение на рис. 18.7 демонстрирует линейки, появляющиеся при изменения размера элемента управления. Эта функциональная возможность позволяет легко увидеть и настроить новые размеры изменяемого элемента управления.

На рис. 18.7 можно заметить некоторые точки привязки (т.е. две стрелки, направленные влево и вверх от кнопки к соответствующим границам контейнерной сетки). Эти стрелки означают, что кнопка будет иметь поля, и задают ее положение в ячейке сетки. В данном случае стрелки означают, что кнопка будет иметь поля слева и сверху, фактически “привязывая” ее верхнюю и левую стороны к левой и верхней сторонам содержащей ее сетки. Однако верхнюю точку привязки легко переключить

так, чтобы она фиксировала кнопку по отношению к нижней границе, а левую точку привязки перенести на правую сторону. Если щелкнуть на стрелке в верхней точке привязки, то она будет перенесена на нижнюю сторону, а если щелкнуть на стрелке в левой точке привязки, то она будет перенесена на правую сторону. Стрелки привязки меняют позицию после каждого щелчка. Кроме того, точки привязки можно устанавливать на обеих сторонах элемента управления сразу (т.е. на левой и правой, а также на верхней и нижней соответственно), так что при изменении размера содержащей его ячейки сетки он будет растягиваться или сжиматься. Например, если левая сторона поля ввода привязана к ячейке сетки, пользователь может также привязать ее к правой стороне, щелкнув на маленьком кружке, расположенном на правой стороне поля ввода. Для того чтобы удалить привязку с какой-то стороны, достаточно щелкнуть на стрелке привязки с этой стороны.

Как указывалось ранее, наиболее важным элементом управления, влияющим на компоновку формы, является Grid. Рассмотрим некоторые специальные возможности конструктора WPF для работы с этим элементом управления. По умолчанию файл `MainWindow.xaml` создается с одним элементом сетки без строк и столбцов. Прежде чем добавлять элементы управления, следует определить некоторые строки и столбцы, с помощью которых можно управлять их компоновкой на форме. Для этого следует начать с выбора сетки, щелкнув на пустой области в середине окна, а затем выбрать соответствующий узел в инструментальном окне Document Outline или установить курсор на соответствующий элемент сетки в самом XAML-файле (в режиме разделения экрана).

При выборе элемента сетки возле левого и верхнего краев сетки появится граница, выделяющая реальную область, занятую сеткой, и относительные размеры каждой строки и столбца, как показано на рис. 18.8. В данном случае на рисунке показана сетка, состоящая из двух строк и двух столбцов.

Добавлять строки и столбцы можно, просто щелкая на границе. После добавления строки или столбца их маркеры можно выделить и перетащить, чтобы задать правильный размер. При первичном размещении маркеров на экране, к сожалению, не высвечивается никакой информации о размерах строки или столбца; однако после создания маркера они появляются.

Когда пользователь перемещает курсор поверх размера строки или столбца, вдоль верхнего края сетки появляются пиктограммы (рис. 18.9).

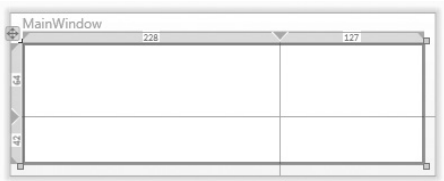


Рис. 18.8

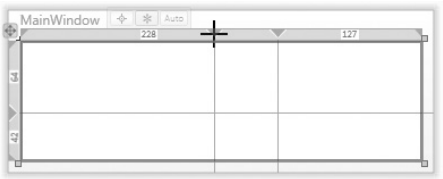


Рис. 18.9

Они позволяют легко указать, что размер строки или столбца должен быть фиксированным (#), пропорциональным (*) или определяться в зависимости от содержимого (Auto).

Для того чтобы удалить строку или столбец, необходимо щелкнуть на них и перетащить за пределы сетки. В этом случае строка или столбец будут удалены, а элементы управления в окружающих ячейках будут изменены соответствующим образом.



Пропорциональный размер (weighted proportion) означает, что пользователь должен задать процент доступного пространства (по сравнению с другими столбцами). После того как строки и столбцы с фиксированными и автоматическими размерами займут свои места, остальное пространство будет разделено между строками и столбцами с пропорциональными размерами. Если пользователь не указал после префикса в виде звездочки числовой коэффициент, то свободное место разделяется между строками поровну. Допустим, например, что пользователь задал сетку с шириной, равной 1000 пикселей, и двумя столбцами. Если ширина каждого из них задана символом `*`, то каждый из них будет иметь ширину, равную 500 пикселям. Однако, если ширина одного из них задана символом `*`, а ширина другого — `3*`, то 1000 пикселей будут разделены на порции размером 250 пикселей, причем одна порция будет выделена первому столбцу (т.е. его ширина будет равна 250 пикселям), а остальные три — второму (т.е. его ширина будет равна 750 пикселям).



При создании элемента управления путем перетаскивания его в ячейку сетки следует "привязать" его к левому и верхнему краям ячейки (т.е. перетаскивать его до тех пор, пока он не будет зафиксирован в этой позиции). В противном случае поля элемента управления будут определяться его фактическим положением в ячейке, а это иногда не соответствует желаниям пользователя.

Инструментальное окно Properties

Помещая элемент управления на форму, необязательно возвращаться к редактору XAML, чтобы задать значения свойств и назначить обработчики событий. Как и в системе Windows Forms, в системе WPF есть окно Properties, которое, впрочем, имеет некоторые особенности (рис. 18.10).

По сравнению с системой Visual Studio 2008 окно Properties приобрело массу новый функциональных возможностей. В системе Visual Studio 2008 его возможности были очень ограничены. В результате во многих случаях пользователь был вынужден непосредственно модифицировать код XAML. Однако в системе Visual Studio 2010 возможности окна Properties были расширены, уменьшив необходимость редактировать XAML-файлы.

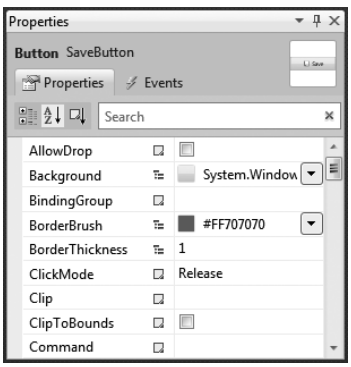


Рис. 18.10

Инструментальное окно Properties для разработки форм по технологии Windows Forms позволяет выбирать элемент управления и задавать его свойства с помощью раскрывающегося списка, расположенного над списком свойств и событий. Однако в окне Properties системы WPF этого списка нет. Теперь пользователь должен выбирать элементы управления в конструкторе, выделяя их в окне Document Outline или устанавливая курсор в область определения элемента управления в XAML-представлении. При этом в левом верхнем углу окна появляется маленький эскиз элемента управления (и всех элементов управления, которые в нем содержатся), за которым следует его квалифицированный тип.



Окно Properties можно использовать, работая с редактором XAML и конструктором одновременно. Однако, если пользователь работает с редактором XAML, конструктор должен быть загружен (если открыть файл непосредственно в редакторе XAML, может возникнуть необходимость переходить в рабочую область конструктора и назад). Если же XAML-файл является некорректным, то сначала придется исправить все ошибки.

Свойство Name элемента управления не входит в список свойств, но для него предусмотрено отдельное поле ввода, расположенное над этим списком. Обратите внимание на то, что если элемент управления еще не имеет имени, то оно присваивается как значение свойства Name (а не x:Name). Однако, если атрибут x:Name для элемента управления уже определен и пользователь изменяет его имя с помощью окна Properties, этот атрибут будет обновлен.

Элементы управления могут иметь много свойств и событий, так что их поиск в списках свойств и событий в системе Windows Forms может оказаться утомительным. Для того чтобы облегчить поиск конкретного свойства, в окне Properties системы WPF предусмотрена функция поиска, которая динамически фильтрует список свойств по тексту, который пользователь вводит в поле ввода. Поисковая строка, которую вводит пользователь, не обязательно должна начинаться с имени свойства или события. Свойство или событие будет найдено, даже если в поисковой строке указана только часть его имени. К сожалению, эта поисковая функция не поддерживает поиск имен, набранных в "верблюжьем" стиле.

Список свойств в конструкторе WPF (аналогично системе Windows Forms) может выводиться как в алфавитном порядке (Alphabetical order), так и по категориям (Categorized order). Отметим, что ни одно свойство, являющееся объектом (например, Margin), не может быть развернуто для демонстрации или редактирования (как это делалось в системе Windows Forms). Если список выводится по категориям, то пользователь может использовать новую и уникальную функциональную возможность окна свойств системы WPF: редакторы категорий. Например, если выбрать элемент управления Button и перейти к категории Text, можно найти специальный редактор для свойств, относящихся к категории Text, и задать их новые значения, как показано на рис. 18.11

В списке свойств также выводятся присоединенные свойства элементов управления, как показано на рис. 18.12.

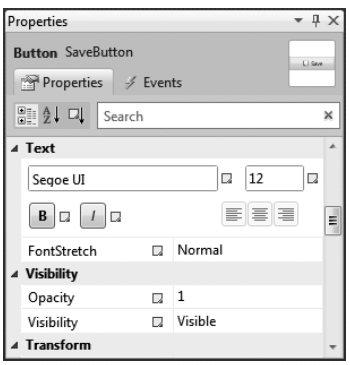


Рис. 18.11

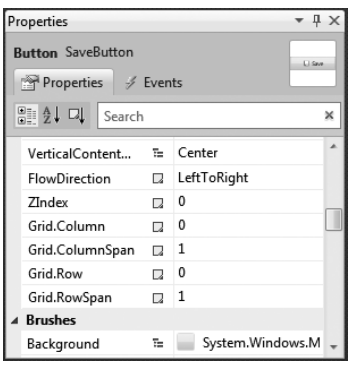


Рис. 18.12

Легко заметить, что справа от имени каждого свойства располагается маленькая пиктограмма. Эта новая функциональная возможность системы Visual Studio 2010 называется *маркерами свойств* (property markers). Эти маркеры указывают, что является источником значения свойства. Когда курсор мыши находится над маркером, на экране появляется подсказка, описывающая смысл свойства. Если источник значения изменяется, то маркер изменяется вместе с ним. Опишем пиктограммы, показанные на рис. 18.13.

- Светло-серая пиктограмма означает, что для свойства еще не задано никакого значения и поэтому оно будет использовать значение, заданное по умолчанию.
- Черный ромбик означает, что для свойства установлено локальное значение (т.е. конкретное значение).
- Желтый цилиндр означает, что для свойства задано выражение, связывающее данные (это понятие будет рассмотрено далее в этой главе).
- Кисточка (с зеленой краской) означает, что для свойства установлен какой-то ресурс.
- Пурпурная иерархия в виде дерева означает, свойство наследовало свое значение у другого элемента управления, который расположен дальше по иерархии.

Щелчок на маркере свойства открывает меню, в котором есть дополнительные команды для присваивания значений данному свойству (рис. 18.14).



Рис. 18.13

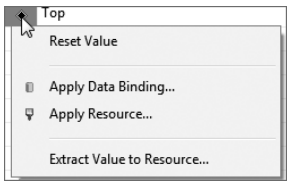


Рис. 18.14

Команда Reset Value просто устанавливает значение свойства равным значению, заданному по умолчанию (удаляя атрибут, присваивающий значения в XAML-файле).

Команда Apply Data Binding вызывает редактор, позволяющий выбирать разные варианты связывания для создания выражений, связывающих данные и используемых в качестве значения свойства. Система WPF поддерживает множество вариантов связывания. Соответствующие окна будут рассмотрены в следующем разделе.

Команда Apply Resource позволяет выбирать ресурс, созданный пользователем или определенный системой WPF, и присваивать его в качестве значения указанному свойству. Ресурсы — это повторно используемые объекты и значения. Эта концепция близка понятию констант в коде. На рис. 18.15 показано всплывающее окно, которое появляется при выборе этой команды.

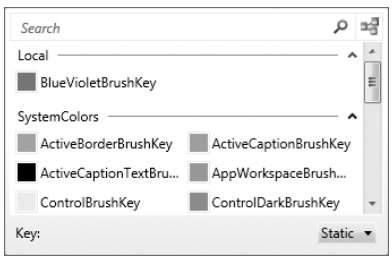


Рис. 18.15

Ресурсами являются все ресурсы, доступные для данного свойства (т.е. находящиеся в его области видимости и имеющие одинаковые типы) и сгруппированные по словарям ресурсов. Обратите внимание на пиктограмму, расположенную в правом верхнем углу всплывающего окна. Щелчок на этой пиктограмме

также группирует ресурсы по XAML-файлам, в которых они определены. Эту опцию можно включать и выключать.

На рис. 18.15 показан ресурс, тип которого совпадает с типом свойства (BlueVioletBrushKey), определенного в текущем XAML-файле (в группе Local). Поскольку это свойство имеет тип SolidColorBrush, в окне продемонстрированы все ресурсы для цвета кисточки, определенные заранее в системе WPF и доступные для выбора.

Вернемся к другим командам из меню, показанного на рис. 18.14. Команда Extract Value to Resource принимает значение данного свойства и возвращает его ресурсу. Этот ресурс создается как ресурс корневого узла в XAML-файле, так что его можно повторно использовать в файле с помощью присвоенного ему уникального ключа. Значение свойства автоматически обновляется так, чтобы использовать этот ресурс. Например, применяя эту команду к свойству Background элемента управления, в котором значение `#FF8888B7` определяет следующий ресурс в элементе `Window.Resources` с именем `BlueVioletBrushKey`.

```
<SolidColorBrush x:Key="BlueVioletBrushKey"> #FF8888B7 </SolidColorBrush>
```

Элемент управления будет ссылаться на этот ресурс следующим образом.

```
Background="{StaticResource BlueVioletBrushKey}"
```

Затем этот ресурс можно применять к другим элементам управления, используя те же самые средства языка XAML или указывая элемент управления и свойство, к которому применяется ресурс, а затем выбирая команду Apply Resource из меню, описанного ранее.

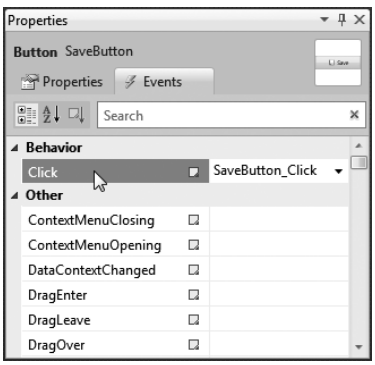


Рис. 18.16

В конструкторе WPF (как и в системе Windows Forms) двойной щелчок на элементе управления автоматически создает обработчик события, заданного для данного элемента по умолчанию в исходном коде. Кроме того, можно создать обработчик события для любых событий, связанных с элементом управления, используя окно Properties, как в системе Windows Forms. Щелкнув на пиктограмме, подсвеченной в окне Properties, можно перейти в вкладку Events, как показано на рис. 18.16. На ней выводится список событий, которые могут происходить с элементом управления. Дважды щелкнув на событии, пользователь может автоматически создать обработчик события в исходном коде.



Разработчики проектов на языке VB.NET, дважды щелкая на элементе управления Button или создавая события с помощью окна Properties, связывают их с помощью синтаксической конструкции Handles. Следовательно, в этом случае обработчик события не присваивается событию как атрибут. Пользователь, применяющий этот метод обработки события, не может видеть определения обработчика события в XAML-файле данного элемента управления, а значит, не может использовать меню Navigate to Event Handler (см. рис. 18.6) в редакторе XAML.

Привязка данных

Привязка данных — одна из очень важных и сильных концепций в системе WPF. Вначале синтаксис привязки данных может отпугнуть, но система Visual Studio 2010 делает создание форм для привязки данных очень легким делом. Система Visual Studio 2010 помогает привязывать данные двумя способами: с помощью команды Apply Data Binding, применяемой к свойству в инструментальном окне Properties, и с помощью перетаскивания средств поддержки привязки данных из окна Data Sources. В этом разделе мы по очереди рассмотрим обе эти возможности.

В системе WPF с элементами управления можно связывать объекты (к которым относятся наборы данных, сущности технологии ADO.NET Entity Framework и т.п.), ресурсы и даже свойства других элементов. Система WPF имеет очень богатые возможности для привязки, позволяющие связывать свойства с чем угодно. Ручное программирование сложных выражений для привязки данных на языке XAML может показаться слишком утомительным, но редактор Apply Data Binding позволяет строить такие выражения с помощью интерфейса "укажи и щелкни".

Для того чтобы связать свойство с элементом управления, сначала необходимо найти требуемый элемент управления в конструкторе и соответствующее свойство в окне Properties. Затем нужно щелкнуть на маркере свойства и выбрать команду Apply Data Binding. Окно, которое откроется после этого, показано на рис. 18.17.

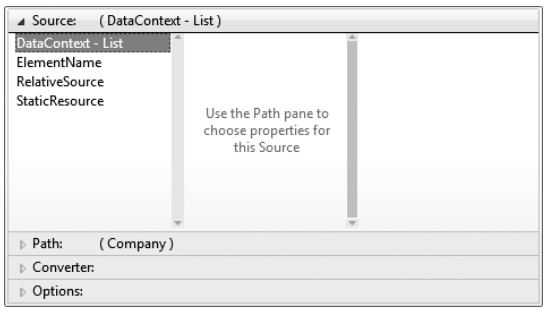


Рис. 18.17

Это окно содержит длинный список этапов (аналогично мастеру), которые необходимо пройти, чтобы осуществить привязку, — Source, Path, Converter и Options, в зависимости от стиля компоновки. Для того чтобы начать новый этап, необходимо щелкнуть на его заголовке.

Обычно, когда пользователь открывает это окно, система предлагает ему выполнить этап Source, позволяющий выбрать источник привязки (иначе говоря, источник данных, с которым необходимо связать элемент управления). Обратите внимание на то, что этот этап можно автоматически пропустить и перейти на этап Path (рис. 18.18), если источник данных уже был связан с элементом управления ранее (или находится выше в иерархии). Если пользовать желает выбрать один из типов привязки (например, ElementName), достаточно выбрать заголовок этапа Source, чтобы изменить ранее заданный источник привязки. Затем необходимо выполнить другие команды привязки (т.е. выбрать команду и перейти к следующему окну).



Рис. 18.18

В примере, показанном на рис. 18.17, на вершине иерархии находится элемент управления Grid, у которого свойству DataContext был присвоен ресурс CollectionViewSource (ссылающийся на объект ViewModel как на источник данных). Значение свойства DataContext наследуется элементами управления, находящимися на более низких уровнях иерархии, поэтому, когда к текстовому полю внутри сетки применяется привязка данных, пользователь может указать, что источником привязки является свойство DataContext текстового поля (в окне это выглядит как присваивание элемента List этому свойству). Выбрав источник привязки, можно переходить на этап Path.

Этап Path позволяет выбрать путь к источнику привязки, содержащему искомое значение. Например, на рис. 18.18 выбрано свойство Company (относящееся к объекту ViewModel, на который ссылается источник связывания).

Если свойство само по себе является объектом, то в нем можно выбрать свойство, к которому следует применить привязку, и т.д. Как показано на рис. 18.18, свойство Company (строка) имеет свойство Length, с которым по желанию можно связать элемент управления.

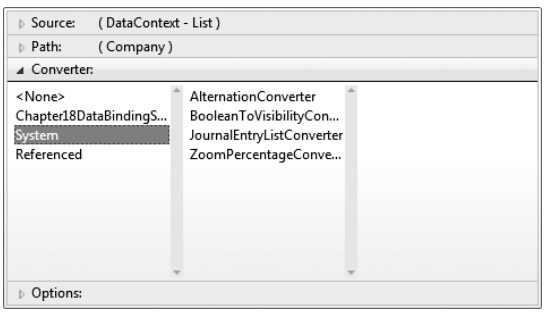


Рис. 18.19

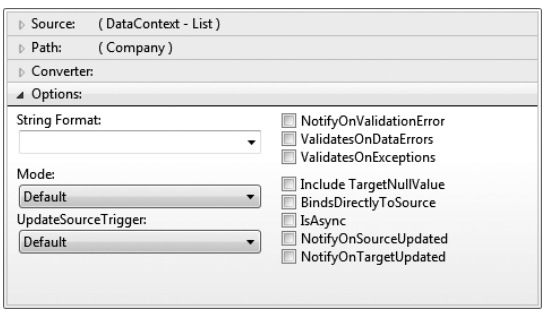


Рис. 18.20

Для того чтобы закрыть редактор, следует дважды щелкнуть на своем последнем выборе. По необходимости можно вызвать конвертер (рис. 18.19), который преобразует связываемое значение перед присваиванием его выбранному значению и перед тем, как значение будет возвращено обратно связываемому свойству (очень мощное средство привязки данных в системе WPF).

Команда Options позволяет также задать другие варианты привязки, показанные на рис. 18.20.

Как видим, этот конструктор намного облегчает процесс создания выражений для привязки, не требуя от пользователя изучения синтаксиса привязки данных. Впрочем, овладеть этим синтаксисом очень полезно, поскольку тогда можно понять выражения, создаваемые в XAML-файле.

Рассмотрим возможности привязки данных с помощью механизма перетаскивания в системе Visual Studio 2010. На первом этапе необходимо создать то, с чем будут связываться данные. Это может быть объект, набор данных или сущность технологии ADO.NET Entity Framework, а также многое другое. Для примера создадим объект. Затем создадим новый класс в проекте с именем `ContactViewModel` и несколько свойств этого класса: FirstName, LastName, Company, Phone, Fax, Mobile и Email (все они являются строками).



Объект назван `ContactViewModel`, потому что он действует как объект `ViewModel`, который имеет отношение к шаблону Model-View-ViewModel (MVVM), упомянутому ранее. Впрочем, этот шаблон проектирования не будет реализован в полном объеме, чтобы не усложнять пример.

Теперь скомпилируем проект (это важно, иначе класс на следующем этапе не появится). Вернитесь к конструктору формы и выберите команду Add New Data Source в меню Data. Затем выберите Object как тип источника данных, щелкните на кнопке Next и выберите класс ContactViewModel из дерева (для того, чтобы найти этот класс, необходимо развернуть узлы иерархии пространства имен). Щелкните на кнопке Finish. Откроется инструментальное окно Data Sources, в котором указан объект ContactViewModel, а ниже перечисляются его свойства, как показано на рис. 18.21.

Перетащите на форму весь объект или его отдельные свойства. В результате возникнет один или несколько элементов управления для демонстрации данных. По умолчанию для демонстрации данных будет создан элемент управления DataGrid, но если пользователь выберет команду ContactViewModel, то появится кнопка, которая в ответ на щелчок откроет меню (как показано на рис. 18.22), предоставляя пользователю возможность выбрать команды DataGrid, List и Details.

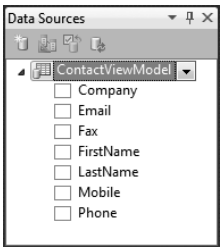


Рис. 18.21

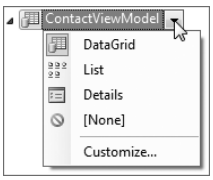


Рис. 18.22

- Команда DataGrid создает элемент управления DataGrid, в котором для каждого свойства объекта предусмотрен отдельный столбец.
- Команда List создает элемент управления List с шаблоном данных, содержащих поля для каждого свойства.
- Команда Details создает элемент управления Grid с двумя столбцами: один для меток, а другой для полей. Для каждого свойства объекта будет создана строка и элемент управления Label, демонстрирующий имя поля (с пробелами, вставленными перед прописными буквами) в первом столбце и само поле (тип которого зависит от типа данных свойства) во втором столбце.

В свойстве Resources элемента управления Window создается ресурс, ссылающийся на объект ContactViewModel, который можно использовать как контекст данных или источник элементов управления, связанных с объектом. Если пользователь планирует задавать источник данных в коде, то на последующих стадиях этот ресурс можно удалить. Элементам управления необходимо выражение для привязки данных. Тип элемента управления, который будет создан на форме для демонстрации данных, зависит от выбора элемента ContactViewModel.

Тип элемента управления, созданного для каждого свойства, по умолчанию является производным от типа данных этого свойства, но, как и в случае элемента Con-

tactViewModel, щелкнув на кнопке, пользователь может выбрать в меню другой тип управляющего элемента (рис. 18.23). Если требуемого типа элемента в списке нет (например, если необходимо использовать управляющий элемент, разработанный сторонним поставщиком), то можно выбрать команду Customize и добавить в список соответствующий тип данных. Если же пользователь не хочет создавать поле для свойства, то необходимо выбрать команду None.

В данном примере мы создадим *форму сведений* (details form), поэтому в окне Data Sources в узле ContactViewModel выберите команду Details. По желанию можно изменить элемент управления, генерируемый для каждого свойства, но пока мы будем генерировать для каждого свойства поле ввода и сгенерируем свойства в форме сведений. Выберите узел ContactViewModel в окне Data Sources и перетащите его на форму. Вместе с полями, соответствующими каждому свойству, будет создана сетка, показанная на рис. 18.24

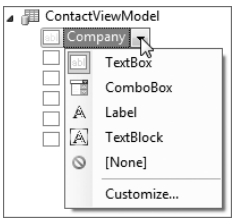


Рис. 18.23



Рис. 18.24

К сожалению, в окне Data Sources нет возможности определить порядок следования полей в форме, поэтому элементы управления придется упорядочивать на сетке вручную (либо с помощью визуального конструктора, либо непосредственно редактируя XAML-файл).

Просмотрев XAML-файл, сгенерированный при осуществлении привязки данных с помощью перетаскивания, легко убедиться, что мы сэкономили много времени и сделали все намного проще и быстрее.



Если в проекте есть элементы управления, имеющие свойства, которым можно присваивать выражения для привязки данных, необходимо создать их свойства зависимости. Свойства зависимости — это особая сущность в технологиях WPF и Silverlight, значением которой может служить вычисляемое выражение (например, выражение для привязки данных). Определение свойств зависимости должно отличаться от определения стандартных свойств. Обсуждение этой концепции выходит за рамки рассмотрения данной главы, но, по существу, только свойствам, определенным как свойства зависимости, можно присваивать выражения для привязки данных.

Выбор стиля приложения

До сих пор наше приложение выглядело слишком простым — проще, чем обычные приложения, создаваемые с помощью системы Windows Forms. Однако огромным преимуществом технологии WPF является легкость, с которой можно изменять внешний вид элементов управления. Это позволяет полностью изменять стиль приложения. Часто используемые изменения можно хранить в виде специальных элементов управления — стилей (коллекции значений свойств элементов управления, хранящейся в виде ресурса, который можно определить один раз и применять ко многим элементам управления) — или полностью переопределять в XAML-файлах, соответствующих элементам управления, создавая новые шаблоны элементов управления. Эти ресурсы можно определить в свойстве Resources любого элемента управления вместе с ключом, который можно применять к любому элементу управления, расположенному на более низких уровнях иерархии, и ссылаться на них с помощью этого ключа. Например, если требуется определить ресурс, доступный для любого элемента управления, существующего в XAML-файле `MainWindow`, то его можно определить в элементе Window.Resources. Если же пользователь хочет иметь возможность использовать его во всем приложении, то он может определить его в свойстве Application.Resources элемента Application в файле `App.xaml`.

Более того, в словаре ресурсов можно определить несколько шаблонов и стилей элементов управления и использовать их как *тему* (theme). Эту тему можно применять ко всему приложению и автоматически настраивать стили пользовательского интерфейса, обеспечивая уникальный и согласованный внешний вид своего приложения. Именно этому посвящен данный раздел. Впрочем, пользователь может не создавать свои собственные темы, а загрузить их на сайте CodePlex со страницы, посвященной проекту WPF Themes (`http://www.codeplex.com/wpfthemes`).

Эти темы изначально были разработаны (*б*ольшая часть — компанией Microsoft) для использования в приложениях технологии Silverlight, но затем они были преобразованы (так, где это было необходимо) для применения к приложениям технологии WPF. Используем одну из этих тем для создания совершенно иного внешнего вида нашего приложения. Создадим новое приложение и добавим на форму несколько разных элементов управления, как показано на рис. 18.25.

Как видите, элементы выглядят очень невыразительно, поэтому мы применим тему и легко полностью изменим внешний вид формы. Загружая тему с веб-страницы проекта WPF Themes, можно заметить, что он содержит решение, состоящее из двух проектов: один реализует темы, а второй демонстрирует результаты использования первого. Однако мы используем темы несколько иначе. Запустим простое приложение и выберем тему, которая нам нравится. Для демонстрации возьмем тему Shiny Blue. В проекте `WPF.Themes`, находящемся в папке `ShinyBlue`, можно найти файл `Theme.xaml`. Скопируйте его в корневой узел нашего проекта (чтобы гарантировать, что он будет включен в наш проект в системе Visual Studio).

Откройте файл `App.xaml` и добавьте следующий код в раздел `Application.Resources`.

```
<ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="Theme.xaml"/>
    </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

Этот код на языке XAML просто объединяет ресурсы из файлов темы с ресурсами приложения. Теперь эти ресурсы будут применяться ко всему приложению, замещая стили элементов управления, заданные по умолчанию, соответствующими стилями из файла тем.

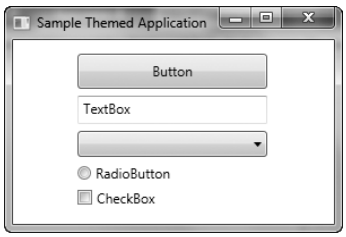


Рис. 18.25

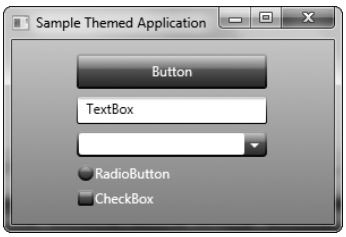


Рис. 18.26

Последнее изменение касается стиля фона наших окон на стиль из файла тем (он не присваивается автоматически). Добавьте в элемент Window атрибут

```
Background="{StaticResource WindowBackgroundBrush}"
```

Запустите проект. Теперь все элементы управления выглядят совершенно иначе, как показано на рис. 18.26.

Для того чтобы изменить тему, достаточно заменить файл `Theme.xaml` другим файлом из проекта WPF.Themes и перекомпилировать проект.



При частом изменении стилей и шаблонов элементов управления в приложениях намного проще использовать программу Expression Blend — инструмент, специально разработанный для разработчиков графических приложений, работающих с языком XAML. Программа Expression Blend намного лучше приспособлена для разработки графики и анимации, чем система Visual Studio. Программа Expression Blend может открывать решения, созданные в системе Visual Studio, а также просматривать, редактировать и компилировать проекты, хотя сама она намного лучше подходит для задач проектирования. Эта интеграция системы Visual Studio и программы Expression Blend позволяет поддерживать технологический процесс проектирования и разработки. Обе программы могут одновременно открывать одно и то же решение или проект (даже на одном и том же компьютере), позволяя пользователю быстро переключаться между ними по мере необходимости. Если файл был открыт одной из этих программ, то при сохранении изменений в файле, выполняемом другой программой, откроется диалоговое окно, в котором пользователю будет задан вопрос о том, не хочет ли он перезагрузить файл. Для того чтобы легко открыть решение, созданное в программе Expression Blend, находясь в системе Visual Studio, пользователь должен щелкнуть правой кнопкой мыши на XAML-файле и выбрать команду Open in Expression Blend.

Взаимодействие с системой Windows Forms

До сих пор мы описывали процесс создания приложения в графической системе WPF, однако вполне вероятно, что пользователь уже имеет большой опыт в разработ-

ке приложений с помощью системы Windows Forms и вряд ли захочет немедленно перейти на новую технологию. У него есть возможность сохранить свои наработки и не переписывать их в погоне за новыми технологиями. Для того чтобы облегчить переход на новую технологию, компания Microsoft сохранила возможность одновременно использовать технологии WPF и Windows Forms при создании одного и того же приложения. Обоюдное взаимодействие поддерживается как приложениями WPF, так и приложениями Windows Forms, причем элементы управления системы WPF могут использоваться в приложениях, созданных по технологии Windows Forms, и наоборот. В настоящем разделе мы покажем, как реализовать эти сценарии.

Использование элементов управления WPF Control в приложениях Windows Forms

Для начала создадим новый проект в своем решении, чтобы внедрить в него элемент управления WPF. Этим элементом (в целях демонстрации) будут поля для ввода имени и пароля пользователя. В диалоговом окне Add New Project (рис. 18.27) выберите шаблон проекта WPF User Control Library. Он уже содержит XAML-файл и исходный файл, необходимые для элемента управления в системе WPF. Если проверить XAML-код, соответствующий данному элементу управления, то можно увидеть, что он, по существу, совпадает с исходным XAML-кодом для окна, рассмотренного в начале главы, за исключением того, что корневым XAML-элементом является элемент UserControl, а не Window.

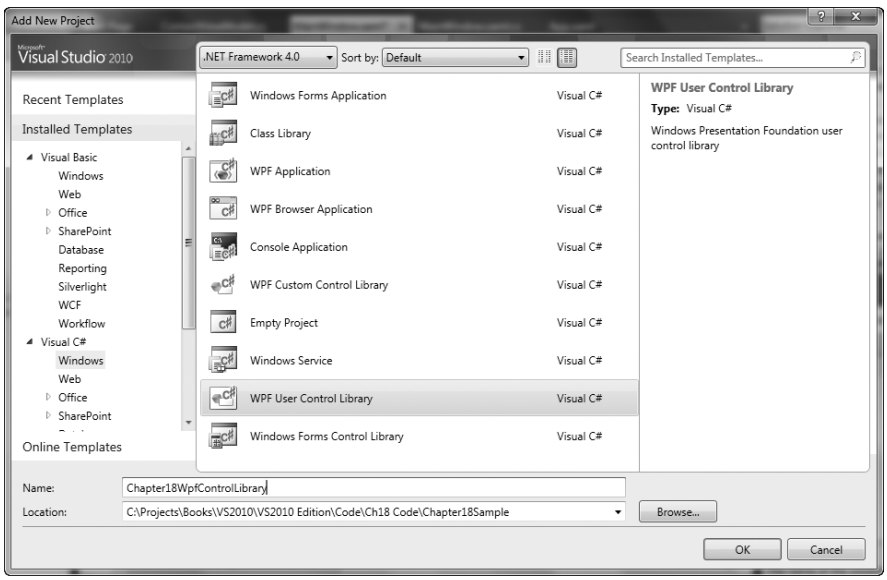


Рис. 18.27

Переименуйте элемент управления в UserLoginControl и добавьте сетку, два текстовых блока и два поля для ввода, как показано на рис. 18.28.



Рис. 18.28

В исходный файл добавьте несколько простых свойств, чтобы открыто обнародовать содержимое текстовых блоков (получатель и отправитель).

VB

```
Public Property UserName As String
    Get
        Return txtUserName.Text
    End Get
    Set(ByVal value As String)
        txtUserName.Text = value
    End Set
End Property

Public Property Password As String
    Get
        Return txtPassword.Text
    End Get
    Set(ByVal value As String)
        txtPassword.Text = value
    End Set
End Property
```

C#

```
public string Username
{
    get { return txtUserName.Text; }
    set { txtUserName.Text = value; }
}

public string Password
{
    get { return txtPassword.Text; }
    set { txtPassword.Text = value; }
}
```

Теперь, когда у вас есть свой собственный элемент управления, созданный по технологии WPF, соберите проект и создайте новый проект типа Windows Forms, который будет содержать ваш элемент управления. Создайте проект и добавьте в него ссылку на WPF-проект, содержащий элемент управления (используйте команду Add Reference из меню, которое открывается после щелчка правой кнопкой мыши на заголовке References).

Откройте форму, которая будет содержать элемент управления WPF. Поскольку библиотека создаваемых пользователем элементов управления WPF находится в том же самом решении, элемент UserLoginControl появится в окне Toolbox и может быть просто перетащен на форму. В результате будет автоматически добавлен элемент управления ElementHost (который может содержать другие элементы управления WPF), содержащий элемент UserLoginControl.

Однако по необходимости это можно сделать вручную. Для этого необходимо выполнить следующие действия. В инструментальном окне Toolbox содержится вкладка WPF Interoperability, в которой находится отдельный элемент ElementHost. Перетащите его на форму, как показано на рис. 18.29, и вы увидите интеллектуальные дескрипторы, предлагающие пользователю выбрать требуемый элемент управления. Если элемент управления не появляется в раскрывающемся списке, то пользователю придется создать свое решение.

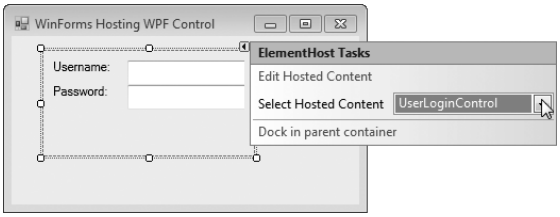


Рис. 18.29

Элемент управления можно загрузить в элемент управления ElementHost. При этом он автоматически получит имя в коде, который будет изменен с помощью свойства HostedContentName.

Использование элементов управления Windows Forms Control в приложениях WPF

Рассмотрим теперь противоположный сценарий — использование элементов управления Windows Forms Control в приложениях WPF. Создайте новый проект `Chapter 18 WinFormsControlLibrary`, используя шаблон проекта Class Library. Удалите класс `Class1` и добавьте в этот проект новый элемент управления User Control с именем `UserLoginControl`.

Откройте этот элемент в визуальном конструкторе и добавьте два текстовых блока и два поля ввода, как показано на рис. 18.30.



Рис. 18.30

В исходный код добавьте несколько простых свойств, чтобы продемонстрировать содержимое текстовых блоков и открыто обнародовать содержимое текстовых блоков (получатель и отправитель).

VB

```
Public Property UserName As String
    Get
        Return txtUserName.Text
    End Get
    Set(ByVal value As String)
        txtUserName.Text = value
    End Set
End Property

Public Property Password As String
    Get
        Return txtPassword.Text
    End Get
    Set(ByVal value As String)
        txtPassword.Text = value
    End Set
End Property
```

C#

```
public string Username
{
    get { return txtUserName.Text; }
    set { txtUserName.Text = value; }
}
```

```
public string Password
{
    get { return txtPassword.Text; }
    set { txtPassword.Text = value; }
}
```

Теперь, когда у вас есть свой собственный элемент управления, созданный по технологии Windows Forms, соберите проект и создайте новый проект типа WPF, который будет содержать наш элемент управления. Создайте проект и добавьте в него ссылку на проект Windows Forms, содержащий элемент управления (используйте команду Add Reference из меню, которое открывается после щелчка правой кнопкой мыши на заголовке References).

Откройте форму, которая будет содержать элемент управления Windows Forms, в визуальном конструкторе. Выберите элемент управления WindowsFormsHost в инструментальном окне Toolbox и перетащите его на форму. К сожалению, при этом визуальный конструктор не может оказать пользователю никакой помощи, и приходится вносить изменения в редакторе XAML editor. Необходимо добавить префикс пространства имен в определение корневого элемента.

```
xmlsn:wfapp="clr-namespace:Chapter18WinFormsControlLibrary;
assembly=Chapter18WinFormsControlLibrary"
```

Модифицируйте элемент WindowsFormsHost, который будет содержать наш элемент управления и перерисовывать его при выполнении приложения, как показано на рис. 18.31.

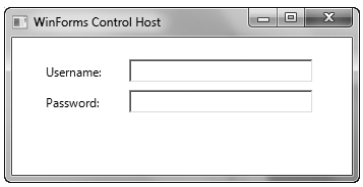


Рис. 18.31

```
<WindowsFormsHost x:Name="windowsFormsHost">
    <wfapp:UserLoginControl x:Name="userLoginDetails" />
</WindowsFormsHost>
```

Отладка с помощью визуализатора WPF

Найти причину ошибки, возникшей на этапе управления, просматривая дерево XAML или визуальное дерево, очень сложно, но, к счастью, в системе VS2010 появилась новая функциональная возможность под названием WPF Visualize, которая помогает отлаживать визуальное дерево WPF-приложения. Например, элемент может оказаться невидимым, хотя должен быть видимым, может появиться не там, где задумано, или иметь неправильный стиль. Визуализатор WPF Visualizer помогает отслеживать такие проблемы, позволяя пользователю просматривать визуальное дерево, просматривать значения свойств выделенного элемента, а также отслеживать источники стилей для свойств.

Для того чтобы открыть визуализатор WPF Visualizer, перейдите в режим прерывания. Затем, используя инструментальное окно Autos, Local или Watch, найдите переменную, которая содержит ссылку на элемент, в XAML-документе для отладки. После этого щелкните на пиктограмме с изображением увеличительного стекла, распо-

ложенной рядом с элементом пользовательского интерфейса WPF, указанным в инструментальном окне, и откройте визуализатор (рис. 18.32). В качестве альтернативы можно поместить курсор на переменную, соответствующую элементу пользовательского интерфейса WPF, чтобы открыть окно DataTip, и щелкнуть на пиктограмме с изображением увеличительного стекла.

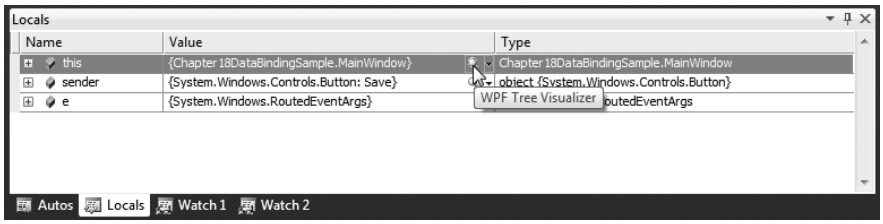


Рис. 18.32

Визуализатор WPF Visualizer показан на рис. 18.33. В левой части окна мы видим визуальное дерево для текущего XAML-документа и прорисовку выбранного элемента, расположенного в дереве под ним. В правой части приведен список всех свойств выбранного элемента дерева, его текущие значения и другая информация о каждом свойстве.

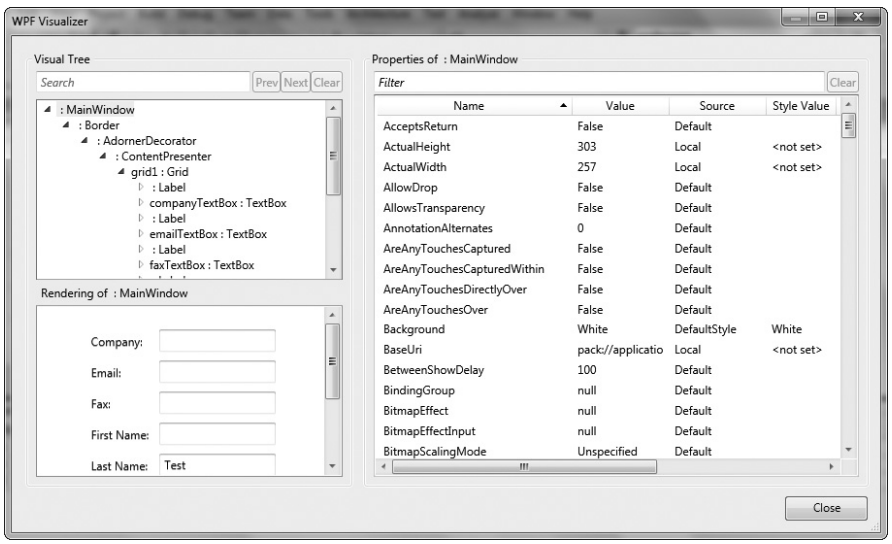


Рис. 18.33

Поскольку визуальное дерево может содержать тысячи элементов, поиск заданного элемента путем перебора может оказаться слишком трудным делом. Если известно имя или тип искомого элемента, то можно ввести его в поле поиска, расположенное над деревом, и пройти по соответствующим узлам, используя кнопки Next и Prev. Можно также отфильтровать список свойств, введя часть имени, значения или типа искомого элемента.

К сожалению, в системе Visual Studio 2010 нет средств для редактирования значения свойства или модификации дерева свойств, но анализ элементов визуального де-

рева и значений их свойств (а также источников этих значений) должен помочь решить проблемы, связанные с XAML, намного проще, чем в предыдущих версиях системы Visual Studio.

Резюме

В этой главе показано, как с помощью системы Visual Studio 2010 создать приложения по технологии WPF. Мы описали несколько важных концепций языка XAML, показали, как использовать уникальные свойства конструктора WPF, а также продемонстрировали способы изменения внешнего вида приложений и возможности взаимодействия систем WPF и Windows Forms.

19

Бизнес-приложения для пакета Office

В ЭТОЙ ГЛАВЕ...

- Изучение способов расширения пакета Microsoft Office
- Создание средств настройки документов Microsoft Word
- Создание надстроек для программы Microsoft Outlook
- Запуск и отладка приложений для пакета Office
- Упаковка и разворачивание приложений для пакета Office

Приложения пакета Microsoft Office всегда допускали расширение с помощью надстроек и других многочисленных методов автоматизации. Даже язык Visual Basic for Applications (VBA), печально известный своими ограничениями, касающимися доступа к файловой системе, позволял писать приложения, использовавшие экземпляр приложения из пакета Office для выполнения некоторых заданий, например, для проверки правописания с помощью средств редактора Word.

После выпуска системы Visual Studio.NET в 2002 году компания Microsoft вскоре выпустила первую версию пакета Visual Studio Tools for Office (известного под аббревиатурой VSTO, которая читается как *visto*). Первая версия пакета VSTO на самом деле не делала ничего нового, за исключением более простого создания приложений, способных использовать программы Microsoft Word и Microsoft Excel. Однако последующие версии пакета VSTO быстро эволюционировали и стали более мощными, позволяя создавать более функциональные приложения, чем приложения на платформе Office.

Последняя версия пакета VSTO поставляется как часть системы Visual Studio 2010. Она имеет несколько усовершенствований по сравнению с предыдущими версиями, включая поддержку пакета Office 2010, расширенную поддержку пользовательского интерфейса Ribbon и улучшенные средства упаковки и развертывания.

Данная глава начинается с обзора типов приложений, которые можно создать с помощью VSTO. Затем в ней рассматривается процесс настройки документов Word, в частности, содержащих специальную панель Actions Pane. В главе также проведен критический анализ процесса создания надстройки для программы Outlook с областью Outlook Form. В заключение приводится важная информация, касающаяся отладки и развертывания приложений для пакета Office.

Выбор типа проекта для пакета Office

Как и следовало ожидать, типы приложений, которые можно создать с помощью пакета VSTO в рамках системы Visual Studio, по сравнению с предыдущей версией значительно обновлены. Теперь существует возможность создавать приложения, ориентированные как на пакет Microsoft Office 2010, так и на Microsoft Office 2007.

Как и в предыдущей версии, надстройки можно создавать почти для каждого приложения, входящего в пакет Office, включая Excel, InfoPath, Outlook, PowerPoint, Project, Visio и Word. Для программ Excel и Word эти решения можно либо присоединить к отдельному документу или загружать каждый раз при запуске приложения.

Новое приложение для пакета Office можно создать, выбрав команду File⇨New⇨Project. Затем необходимо выбрать предпочитаемый язык (Visual Basic или Visual C#) и категорию проекта для пакета Office, как показано на рис. 19.1. Для приложения пакета Office существуют два шаблона проекта: настройка документа и надстройка для приложения.

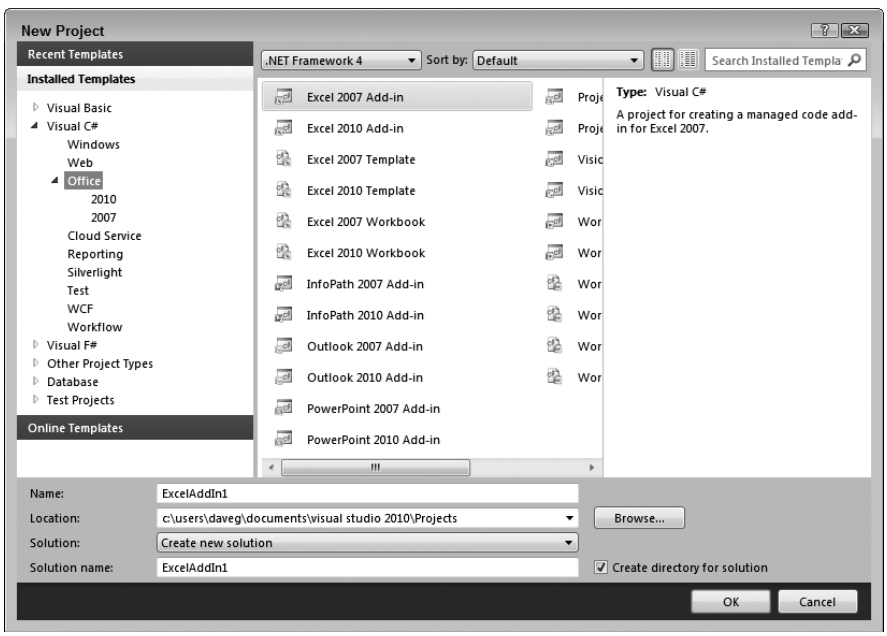


Рис. 19.1

Настройка документа

Настройка документа (document-level customization) — это решение, основанное на отдельном документе. Для того чтобы загрузить настройку, конечный пользователь должен открыть конкретный документ. События, происходящие с документом, например загрузка документа, щелчок на кнопке или команде меню, могут вызывать методы обработки событий из присоединенной сборки. Настройки документов можно включать в шаблоны пакета Office, чтобы загружать их при создании нового документа.

Система Visual Studio 2010 позволяет создавать настройки для следующих типов документов.

- Microsoft Excel Workbook.
- Microsoft Excel Template.
- Microsoft Word Document.
- Microsoft Word Template.

Используя настройку документа, можно модифицировать пользовательский интерфейс приложений Word или Excel, чтобы предоставить конечному пользователю уникальное решение. Например, можно добавлять новые элементы управления в интерфейс Office Ribbon или открывать специальную панель Actions Pane.

Программы Microsoft Word и Microsoft Excel также включают технологию под названием *интеллектуальные дескрипторы* (smart tags), позволяющую разработчику отслеживать пользовательский ввод и распознавать текст в специальном формате. Решение пользователя может использовать технологию, обеспечивая обратную связь, и даже действия, которые пользователь может выполнить в ответ на некоторые распознанные термы, такие как номер телефона или адрес.

Система Visual Studio включает в себя также набор специальных элементов управления, характерных для программы Microsoft Word. Эти *элементы управления контентом* (content controls) оптимизированы как для ввода, так и для вывода данных. Эти элементы управления будут рассмотрены в главе позднее.

Надстройки для приложений

В отличие от настройки документов *надстройка* (add-in) для приложений всегда загружается независимо от документа, который открыт в данный момент. Фактически надстройки для приложений выполняются даже тогда, когда приложение работает без открытых документов.

В предыдущих версиях пакета VSTO существовали значительные ограничения, касавшиеся надстроек для приложений. Например, пользователь мог создавать только надстройки для программы Microsoft Outlook, но и тогда не мог осуществлять полноценную настройку интерфейса.

К счастью, в системе Visual Studio 2010 таких ограничений больше нет, и пользователь может создавать надстройки для практически каждого приложения, входящего в состав пакета Microsoft Office, включая Excel, InfoPath, Outlook, PowerPoint, Project, Visio и Word. Они в равной степени применяются к версиям пакета Office 2007 и 2010. Пользователь может создавать такие же усовершенствования пользовательского интерфейса, как и при настройке документов, например, добавлять новые элементы управления в интерфейс Office Ribbon.

Кроме того, пользователь может создавать специальную панель Task Pane в качестве надстройки. Панель Task Pane очень похожа на панель Actions Pane, доступную в настройках документов. Однако пользовательская панель Task Pane связана с приложением, а не с документом, и поэтому может создаваться на уровне надстройки для приложения.

С другой стороны, панель Actions Pane представляет собой разновидность панели Task Pane, которую можно настраивать и присоединять к конкретному документу Word или рабочей книге Excel. В надстройках для приложений панель Actions Pane создать невозможно.

Кроме того, в системе Visual Studio 2010 существует возможность создавать настраиваемые области форм Outlook в надстройках для программы Outlook. Область формы – это окна, которые появляются после выбора команды приложения Outlook, например Contact или Appointment. Пользователь может как расширять существующие области форм, так и создавать полностью настраиваемые формы Outlook. Позднее в этой главе мы рассмотрим процесс создания надстройки для программы Outlook 2010, включающей настраиваемую область формы Outlook.

Создание настройки документа

Рассмотрим процесс создания настройки документа Word. В частности, покажем, как создать настройку документа с элементами управления Word Content Control и специальной панелью Actions Pane.



Пример из данного раздела использует программу Word 2010, которая должна быть инсталлирована заранее, чтобы проект можно было отлаживать. Если на компьютере инсталлирована только версия Word 2007, то необходимо следовать указанным инструкциям и создать настройку документа, используя шаблон проекта Word 2007. Разница между программами Word 2010 и Word 2007 описана в инструкциях.

Первый проект VSTO

При создании настройки документа с помощью системы Visual Studio 2010 можно либо создать документ с нуля, либо начать разработку на основе существующего документа или шаблона. Прекрасным источником шаблонов, в частности, для бизнес-форм, являются свободно распространяемые шаблоны, которые можно загрузить с веб-сайта Office Online компании Microsoft по адресу `http://office.microsoft.com/templates/`.



Все шаблоны, доступные для загрузки с веб-сайта Office Online, имеют старый формат Word 97–2003 (`.dot`). К сожалению, некоторые свойства, например элементы управления Word Content Controls, доступны только для документов, созданных в новом формате Open XML format (`.dotx`). Следовательно, сначала необходимо убедиться в том, что шаблон документа имеет новый формат, чтобы получить доступ ко всем имеющимся возможностям.

В этом примере используется сообщение Employee, доступное в категории Forms, Employment. При загрузке шаблона с веб-сайта Office Online с помощью браузера In-

ternet Explorer пользователь получит предложение сохранить его в папке, предусмотренной для хранения шаблонов по умолчанию. После того как шаблон будет сохранен, программа Microsoft Word сможет открыть новый документ, используя этот шаблон. Рекомендуем сохранить этот новый документ в удобной папке на компьютере в виде шаблона в формате Open XML format (`.dotx`), как показано на рис. 19.2.

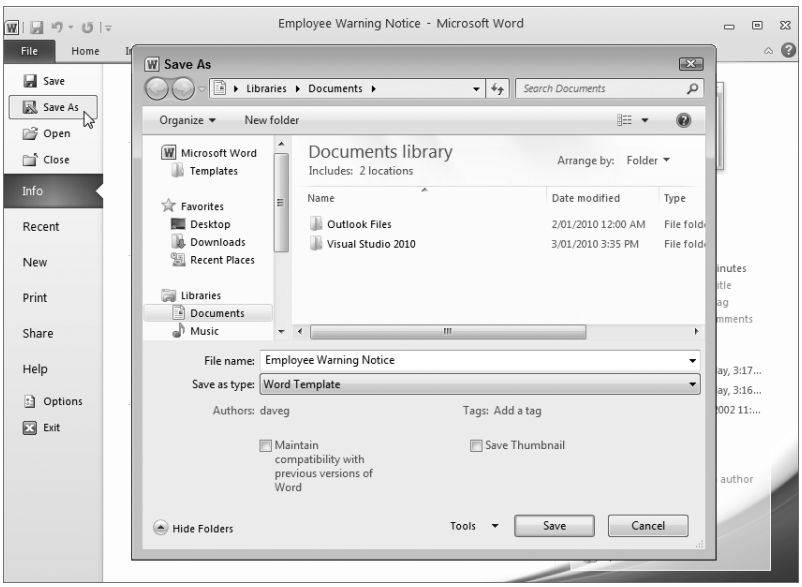


Рис. 19.2

Затем запустите программу Visual Studio 2010 и выберите команду File⇨New⇨Project. Отфильтруйте типы проектов, выбирая последовательно предпочитаемый язык (C# или Visual Basic), тип приложения Office и новый шаблон Word 2010 Template. На экране появится приглашение создать новый документ или скопировать существующий. Скопируйте существующий документ, а затем выберите шаблон документа, сохраненного ранее. Щелкнув на кнопке OK, завершите создание проекта и откройте документ в визуальном редакторе, как показано на рис. 19.3.



Пакет VSTO требует доступа к средствам языка Visual Basic for Applications (VBA), даже если проект не использует этот язык. По этой причине при создании первого приложения для пакета Office система предлагает пользователю подключить язык доступа к средствам языка VBA. Этот доступ должен быть открыт даже в том случае. когда проект создается на языке C#.

На рис. 19.3 заслуживают внимания несколько моментов. Во-первых, вдоль верхнего края рабочей области визуального редактора расположен интерфейс Office Ribbon. Это тот же самый интерфейс, который используется в программе Word. С его помощью можно модифицировать компоновку и внешний вид документа Word. Во-вторых, в окне Solution Explorer, расположенном справа, показано, что файл, открытый в данный момент, называется `ThisDocument.cs` (или `ThisDocument.vb`, если используется язык Visual Basic). Пользователь может щелкнуть на этом файле правой

кнопкой мыши и выбрать либо визуальный редактор View Designer для вывода в рабочей области для документа (см. рис. 19.3), либо редактор кода View Code, чтобы открыть исходный файл для документа. В заключение в инструментальном окне Toolbox, показанном слева, имеется группа вкладок Word Controls, содержащая набор элементов управления, позволяющих создавать полноценные пользовательские интерфейсы для ввода данных и вывода их на экран.

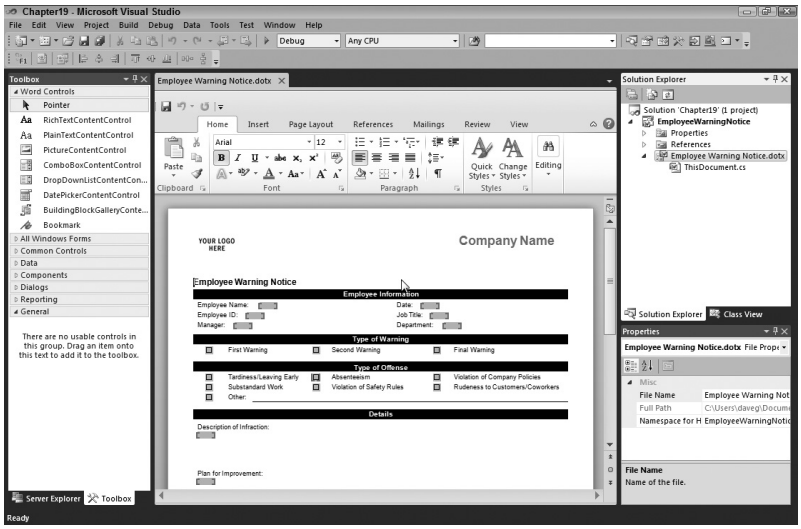


Рис. 19.3

Для настройки этой формы сначала перетащите четыре элемента управления PlainTextContentControl в рабочую область визуального редактора для вывода полей Employee Name, Employee ID, Job Title и Manager. Переименуйте эти элементы управления в `txtEmpName`, `txtEmpID`, `txtJobTitle` и `txtManager` соответственно.

Затем перетащите элемент DatePickerContentControl поля Date и переименуйте его в `dtDate`. Перетащите элемент управления DropDownListContentControl для поля Department и переименуйте его в `ddDept`.

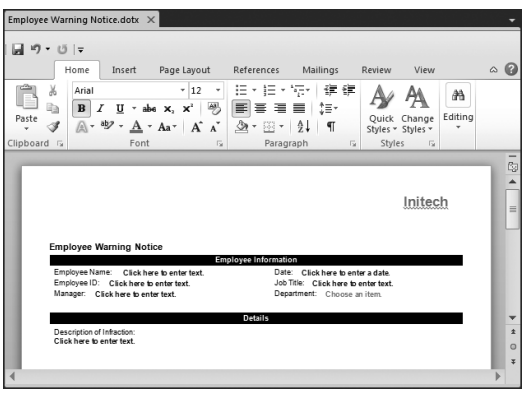


Рис. 19.4

Перетащите элемент управления RichTextContentControl в раздел Details документа и поместите его над меткой Description of Infraction.

В заключение немного очистите документ, удалив разделы Type of Warning и Type of Offense, а также весь текст, расположенный под добавленным элементом управления RichTextContentControl. После этого форма примет вид, показанный на рис. 19.4.

Перед запуском проекта необходимо заполнить раскрывающийся список Department. Несмотря на то что это можно сделать декларативно с помощью поля

Properties, в качестве упражнения выполните это программным способом. Щелкните правой кнопкой мыши на файле ThisDocument в окне Solution Explorer и выберите команду View Code для вывода управляемого кода, лежащего в основе документа. В нем есть два заранее определенных метода: функция, которая выполняется при запуске, когда документ открывается, и функция, которая выполняется по завершении работы, когда документ закрывается.

Для того чтобы заполнить список Department, добавьте в метод ThisDocument_Startup следующий код.



C#

```
private void ThisDocument_Startup(object sender, System.EventArgs e)
{
    ddDept.PlaceholderText = "Select your department";
    ddDept.DropDownListEntries.Add("Finance", "Finance", 0);
    ddDept.DropDownListEntries.Add("HR", "HR", 1);
    ddDept.DropDownListEntries.Add("IT", "IT", 2);
    ddDept.DropDownListEntries.Add("Marketing", "Marketing", 3);
    ddDept.DropDownListEntries.Add("Operations", "Operations", 4);
}
```

Сниппет кода ThisDocument.cs



VB

```
Private Sub ThisDocument_Startup() Handles Me.Startup
    ddDept.PlaceholderText = "Select your department"
    ddDept.DropDownListEntries.Add("Finance", "Finance", 0)
    ddDept.DropDownListEntries.Add("HR", "HR", 1)
    ddDept.DropDownListEntries.Add("IT", "IT", 2)
    ddDept.DropDownListEntries.Add("Marketing", "Marketing", 3)
    ddDept.DropDownListEntries.Add("Operations", "Operations", 4)
End Sub
```

Сниппет кода ThisDocument.vb

Теперь можно запустить проект в режиме отладки, нажав клавишу <F5>. При этом проект скомпилируется и документы откроются в программе Microsoft Word. Для того чтобы протестировать приложение, введите данные в разные поля и проследите за реакцией программы.

Защита компоновки документа

Открыв документ, пользователь может не только вводить текст в добавленные поля, но и редактировать окружающий текст и даже удалять некоторые элементы управления. Очевидно, что это не идеальный сценарий. К счастью, пакеты Office и VSTO позволяют защитить документ от нежелательного редактирования. Для этого необходимо открыть вкладку Developer.

При работе с программой Word 2010 необходимо щелкнуть на корешке вкладки File, а затем на кнопке Options. Затем в окне Word Options следует выбрать команду Customize Ribbon и установить флажок Developer под списком Main Tabs.

При работе с программой Word 2007 необходимо щелкнуть на кнопке Office, а затем на кнопке Word Options, расположенной внизу экрана. В диалоговом окне Word Options необходимо установить переключатель Show Developer Tab в положение Ribbon.

После остановки отладки и возвращения в систему Visual Studio пользователь увидит новую вкладку на инструментальной панели, расположенной над интерфейсом Ribbon, как показано на рис. 19.5. Эта вкладка содержит полезные функции для задач, связанных с пакетом Office.

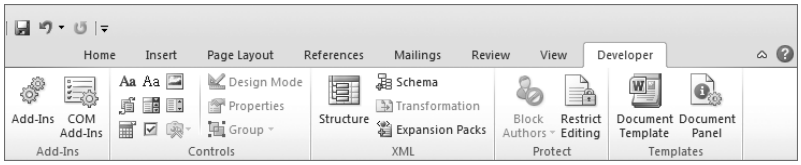


Рис. 19.5

Для предотвращения нежелательного редактирования документа пользователь должен выполнить ряд действий. Во-первых, убедиться, что визуальный редактор открыт, и нажать комбинацию клавиш <Ctrl+A>, чтобы выделить все содержимое документа (текст и элементы управления). На вкладке Developer необходимо выполнить команду Group⇨Group. В этом случае все содержимое документа будет интерпретироваться как одно целое, и его свойства можно изменять за один шаг.

Выбрав новую группу, откройте окно Properties и установите для свойства LockContentControl значение True. Теперь, когда проект будет запущен, стандартный текст документа нельзя будет редактировать или удалять. Можно будет только вводить данные в элементы управления, добавленные пользователем.

Добавление панели Actions Pane

В заключение добавим к документу панель Actions Pane. Эта панель обычно прикрепляется к одной из сторон окна в редакторе Word и может использоваться для отображения сопутствующей информации или получения доступа к дополнительной информации. Например, к форме заявления на отпуск можно добавить панель Actions Pane, которая вычисляет количество дней для отпуска работника и выводит его на экран.



В панелях Actions Pane (в настройках документов) и Task Pane (в надстройках для приложений) используются только стандартные элементы управления. Правда, на панели Actions Pane есть шаблон элемента управления, однако, по существу, это не более чем возможность добавить в проект стандартный элемент управления из импортируемого пространства имен Office. В надстройках для приложений на панели Task Panes нет настраиваемого шаблона элемента управления, поэтому пользователь может просто добавлять в проект стандартные элементы управления.

Для того чтобы добавить панель Actions Pane к настройке документа, щелкните правой кнопкой мыши на проекте в окне Solution Explorer, выберите команду Add⇨New Item⇨Select Actions Pane Control, введите осмысленное имя и щелкните на кнопке Add. Панель Actions Pane откроется в новом окне визуального редактора. Добавьте на панель кнопку для команды, которая будет определять имя текущего пользователя и добавлять его в документ. Перетащите командную кнопку на форму и назовите ее `btnGetName`. Затем дважды щелкните на ней, чтобы зарегистрировать обработчик событий, и добавьте для события, связанного с щелчком на кнопке, следующий код.



C#

```
private void btnGetName_Click(object sender, EventArgs e)
{
    var myIdent = System.Security.Principal.WindowsIdentity.GetCurrent();
    Globals.ThisDocument.txtEmpName.Text = myIdent.Name;
}
```

Сниппет кода GetUserName.cs

VB

```
Private Sub btnGetName_Click(ByVal sender As System.Object, _
                             ByVal e As System.EventArgs) _
                             Handles btnGetName.Click
    Dim myIdent = System.Security.Principal.WindowsIdentity.GetCurrent()
    Globals.ThisDocument.txtEmpName.Text = myIdent.Name
End Sub
```

Сниппет кода GetUserName.vb

Компоненты панели Actions Pane не добавляются в документ автоматически, потому что пользователь может пожелать открыть другую панель Actions Panes, если выяснится, что в этот момент документ редактируют другие пользователи. Однако, если существует только один компонент Actions Pane и нужно немедленно добавить его в открытый документ, можно добавить этот компонент в коллекцию `ActionsPane.Controls` в момент загрузки документа, как показано в следующем фрагменте кода.

C#

```
private void ThisDocument_Startup(object sender, System.EventArgs e)
{
    this.ActionsPane.Controls.Add(new NameOfActionsPaneControl ());
}
```

Сниппет кода ThisDocument.cs

VB

```
Private Sub ThisDocument_Startup() Handles Me.Startup
    Me.ActionsPane.Controls.Add(new NameOfActionsPaneControl ())
End Sub
```

Сниппет кода ThisDocument.vb

При создании надстройки для приложения следует добавить пользовательский элемент управления в коллекцию `CustomTaskPanes`. В момент следующего запуска проекта он выведет на экран документ в программе Word с панелью Actions Pane, как показано на рис. 19.6.

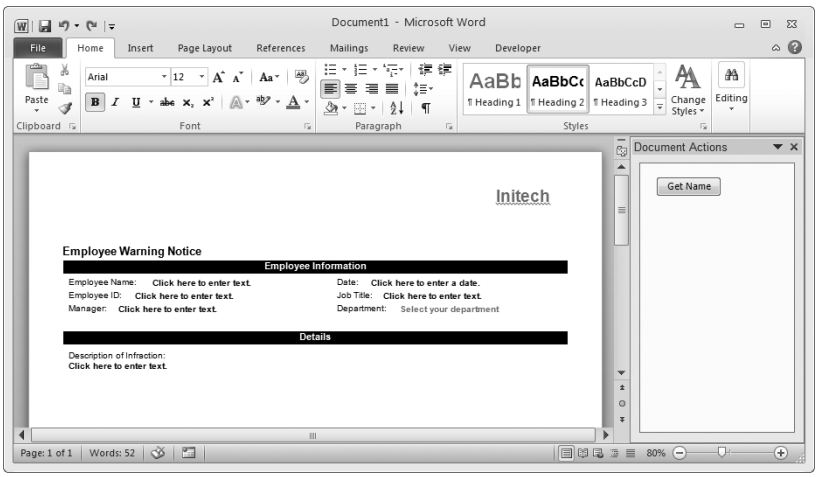


Рис. 19.6

Создание надстройки для приложения

В этом разделе рассматривается процесс создания надстройки для программы Microsoft Outlook 2010. Это позволит показать, как создать надстройку уровня приложения, включающую в себя настраиваемую форму Outlook с элементом управления Contact.



Никогда не разрабатывайте надстройки для программы Outlook, используя свою учетную запись! *Слишком велик риск, что вы случайно сделаете то, о чем впоследствии будете сожалеть, например, удалите все сообщения из папки Inbox. Работая с программой Outlook, можно создавать отдельные почтовые профили – один для обычного почтового ящика, а второй для тестового.*

Некоторые концепции программы Outlook

Прежде чем создавать надстройку для программы Outlook, целесообразно разобраться в основных концепциях, характерных для программы Outlook. Несмотря на разумное сходство с другими программными продуктами из пакета Office, программа Outlook всегда имела особую модель программирования.

Объектная модель программы Outlook в значительной степени опирается на прикладной интерфейс, использующий коллекции. На высшем уровне иерархии классов находится класс `Application`, представляющий приложение Outlook. К нему можно непосредственно обращаться из кода как к свойству надстройки: `this.Application` в языке C# и `Me.Application` в языке Visual Basic. Посредством класса `Application` можно получить доступ к классам, представляющим окна Explorer и Inspector.

Окно Explorer в приложении Outlook — это главное окно, которое выводится на экран, когда программа открывается впервые, и демонстрирует содержимое папки, например Inbox или Calendar. На рис. 19.7, *слева*, показано окно Calendar в окне Explorer. Это окно представляется классом Explorer, включающем свойства, методы и события, которые можно использовать для доступа к этому окну и реагирования на действия.

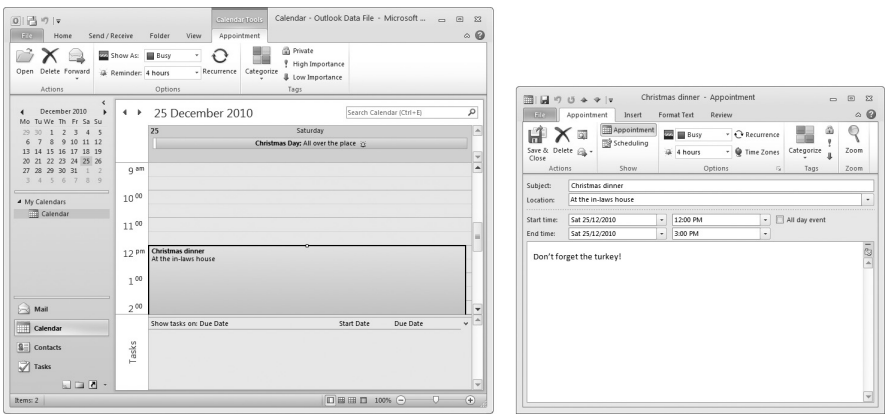


Рис. 19.7

Окно Inspector содержит отдельные элементы, такие как электронное сообщение, контакт или напоминание. На рис. 19.7, *справа*, показано окно Inspector с напоминанием. Класс Inspector содержит свойства и методы доступа к окну, а также события,

которые можно обрабатывать, когда в окне выполняются какие-то действия. Область формы Outlook содержится в окне Inspector.

Класс `Application` содержит также объект `Session`, представляющий все, что происходит в ходе текущей сессии программы Outlook. Этот объект обеспечивает пользователю доступ к спискам адресов, хранилищам, папкам и другим объектам программы Outlook. Почтовые папки, такие как Inbox или Calendar, представляются классом `MAPIFolder` и содержат коллекции элементов. В программе Outlook каждый элемент имеет свойство класса сообщений, описывающее, каким образом почтовое сообщение должно быть представлено в приложении. Например, электронное сообщение относится к классу сообщений `IPM.Note`, а напоминание — к классу сообщений `IPM.Appointment`.

Создание области формы Outlook

Освоив основы объектной модели Outlook, вы сможете создать свою первую надстройку. Выберите команду File⇨New⇨Project в системе Visual Studio 2010. Отфильтруйте типы проектов, сначала выбрав тип Visual C#, затем Office и Outlook 2010 Add-in.



При работе с программой Outlook 2007 все равно следует выбрать тип проекта Outlook 2010 Add-In project.

В отличие от настроек документов, надстройки для приложения существенно опираются на код. При создании надстроек для программы Word или Excel при первом запуске приложения может даже не открываться ни один документ. Надстройка для приложения Outlook основана на простых соображениях: при первом запуске надстройки для приложения Outlook она состоит только из одного невизуального класса `ThisAddIn.cs` (или `ThisAddIn.vb`). В него можно добавить код, выполняющий некоторые действия в начале и по завершении работы приложения.

Для того чтобы настроить пользовательский интерфейс программы Outlook на конкретного пользователя, в него можно добавить область формы Outlook. Это элемент управления, принадлежащий окну Outlook Inspector, когда на экран выводится элемент определенного сообщения.

Для того чтобы добавить новую область формы Outlook, щелкните правой кнопкой мыши на проекте в окне Solution Explorer и выберите команду Add⇨New Item. Затем в списке доступных элементов выберите Outlook Form Region, присвойте ему осмысленное имя и щелкните на кнопке Add. После этого система Visual Studio запустит программу New Outlook Form Region Wizard, которая определит основные свойства, необходимые для создания нового элемента.

На первом этапе работы мастер запрашивает у пользователя, что ему делать: проектировать новую форму или импортировать файл хранилища Outlook Form Storage (с расширением `.ofs`), содержащий форму, разработанную в программе. Выберите команду Select Design a New Form Region и щелкните на кнопке Next.

На втором этапе мастер позволяет выбрать тип создаваемой области формы. Для этого он предлагает удобное визуальное представление каждого типа, как показано на рис. 19.8. Выберите переключатель Separate и щелкните на кнопке Next.

На следующем этапе мастер позволяет ввести имя области формы и, в зависимости от выбранного типа области формы, заголовок и описание. На данном этапе пользователь может также выбрать режим отображения области формы. *Режим сочинения* (compose mode) включается, когда элемент управления создается впервые, например,

при создании нового электронного сообщения. *Режим чтения* (read mode) включается, когда пользователь открывает ранее созданное или полученное электронное сообщение. Убедитесь, что оба этих флажка установлены, введите имя `Custom Details` и щелкните на кнопке Next.

На последнем этапе работы мастера пользователь может выбрать класс сообщения для отображения области формы. Можно выбрать любой из стандартных классов сообщения, например почтовое сообщение или напоминание, или указать пользовательский класс сообщений. Выберите класс сообщений Contact, как показано на рис. 19.9, и щелкните на кнопке Finish, чтобы закрыть мастер.

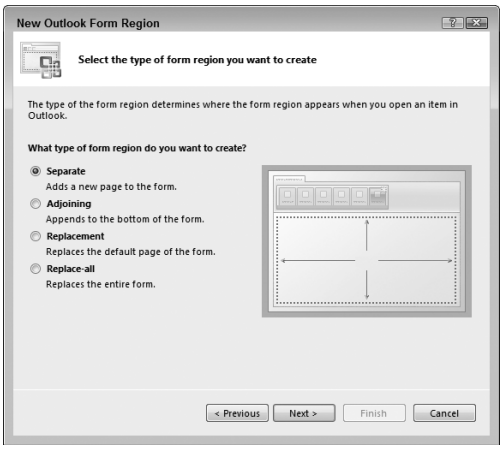


Рис. 19.8



Рис. 19.9

После завершения работы мастера создается новая область формы, которая открывается визуальным редактором. Как указывалось ранее, область формы Outlook, как и панели Actions Pane и Task Pane, представляет собой всего лишь элемент управления. Но в отличие от панели Actions Pane она содержит встроенный манифест, определяющий внешний вид области формы в приложении Outlook. Для того чтобы получить доступ к этому манифесту, следует выбрать форму в визуальном редакторе и открыть окно Properties. В этом окне будет показано свойство Manifest, под которым скрыто множество разных свойств, описывающих внешний вид области формы. Кроме того, к этому свойству можно обратиться посредством кода в ходе выполнения приложения.

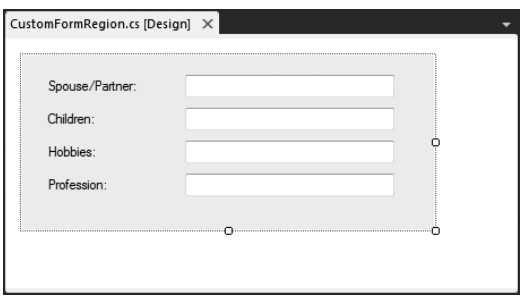


Рис. 19.10

В этом сценарии область формы Outlook используется для демонстрации некоторой дополнительной полезной информации об элементе Contact. Компоновка области формы Outlook создается точно так же, как и компоновка любого другого элемента управления. Перетащите на окно визуального редактора четыре метки элементов управления и четыре поля ввода, а затем выровняйте их, как показано на рис. 19.10. Переименуйте поля ввода, присвоив им имена

txtPartner, txtChildren, txtHobbies и txtProfession, и измените текст меток, чтобы они соответствовали этим полям.

Класс ContactItem содержит удивительно много свойств, которые скрыты в стандартной форме Contact приложения Outlook. По существу, он содержит более ста полей для разных видов контактов, и существует высокая вероятность, что внешний вид контакта, который может пожелать реализовать пользователь, уже определен в этом списке. В данном случае поля, которые выводятся в форме (супруг/партнер, дети, хобби и профессия), доступны, потому что такие свойства уже существуют. Пользовательское свойство элемента управления можно сохранить, добавив его в коллекцию UserProperties.

Код, лежащий в основе области формы, уже имеет заглушки для обработчиков событий FormRegionShowing и FormRegionClosed. Для того чтобы получить доступ к текущему элементу Contact, необходимо добавить к этим свойствам следующий код, а затем найти и сохранить эти пользовательские свойства.



C#

```
private void CustomFormRegion_FormRegionShowing(object sender,
                                                System.EventArgs e)
{
    var myContact = (Outlook.ContactItem)this.OutlookItem;
    this.txtPartner.Text = myContact.Spouse;
    this.txtChildren.Text = myContact.Children;
    this.txtHobbies.Text = myContact.Hobby;
    this.txtProfession.Text = myContact.Profession;
}

private void CustomFormRegion_FormRegionClosed(object sender,
                                                System.EventArgs e)
{
    var myContact = (Outlook.ContactItem)this.OutlookItem;
    myContact.Spouse = this.txtPartner.Text;
    myContact.Children = this.txtChildren.Text;
    myContact.Hobby = this.txtHobbies.Text;
    myContact.Profession = this.txtProfession.Text;
}
```

Сниппет кода CustomFormRegion.cs



VB

```
Private Sub CustomFormRegion_FormRegionShowing(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
                            Handles MyBase.FormRegionShowing
    Dim myContact = CType(Me.OutlookItem, Outlook.ContactItem)
    myContact.Spouse = Me.txtPartner.Text
    myContact.Children = Me.txtChildren.Text
    myContact.Hobby = Me.txtHobbies.Text
    myContact.Profession = Me.txtProfession.Text
End Sub

Private Sub CustomFormRegion_FormRegionClosed(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
                            Handles MyBase.FormRegionClosed
    Dim myContact = CType(Me.OutlookItem, Outlook.ContactItem)
    myContact.Spouse = Me.txtPartner.Text
    myContact.Children = Me.txtChildren.Text
    myContact.Hobby = Me.txtHobbies.Text
    myContact.Profession = Me.txtProfession.Text
End Sub
```

Сниппет кода CustomFormRegion.vb

Для сборки и выполнения надстройки в режиме отладки нажмите клавишу <F5>. Если решение скомпилировано правильно, приложение Outlook откроет его как зарегистрированную надстройку. Откройте папку `Contacts` и создайте новый элемент управления Contact. Для того чтобы увидеть свою область формы Outlook, щелкните на кнопке Custom Details вкладке Show, расположенной в группе Office Ribbon. На рис. 19.11 показана область формы Outlook, которая должна появиться в окне Contact Inspector.

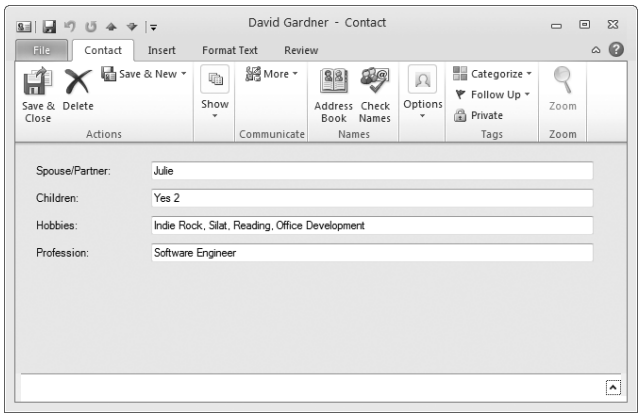


Рис. 19.11

Отладка приложений для пакета Office

Отладка приложения для пакета Office очень похожа на отладку любого другого приложения для системы Windows. При отладке приложения для пакета Office можно использовать все стандартные функциональные возможности отладки, существующие в системе Visual Studio, например, возможность вставлять точки прерывания и задавать наблюдаемые переменные.

Исполняющая среда VSTO, выполняющая загрузку надстроек в соответствующие приложения, может вывести сообщение о любой ошибке, возникшей при запуске, или сделать запись в журнале регистрации. По умолчанию эти возможности отключены, но их можно включить с помощью переменных окружения.

Для того чтобы вывести сообщение об ошибке в отдельном окне, создайте переменную окружения с именем `VSTO_SUPPRESSDISPLAYALERTS` и присвойте ей нулевое значение. Задав ее равной единице или вообще удалив, вы предотвратите появление сообщений об ошибках.

Для того чтобы записывать ошибки в журнал регистрации, создайте переменную окружения с именем `VSTO_LOGALERTS` и присвойте ей единицу. Исполняющая среда система VSTO создаст журнал регистрации `<manifestname>.manifest.log` в папке, где находится манифест приложения. Установив эту переменную равной нулю или удалив ее вообще, вы сможете отменить регистрацию ошибок в журнале.

Отмена регистрации надстройки

При компиляции надстройки для приложения система Visual Studio 2010 автоматически регистрирует ее для данного приложения. Автоматическая отмена регистра-

ции надстройки не состоится, пока пользователь не выполнит команду Build⇨Clean Solution. Следовательно, надстройка будет загружаться каждый раз, когда будет запускаться приложение, для которого она была зарегистрирована. Для того чтобы этого не происходило, вместо повторного открытия решения в системе Visual Studio можно отменить регистрацию надстройки непосредственно из пакета Office.

Для того чтобы отменить регистрацию приложения, необходимо открыть окно Add-Ins. В программе Outlook 2010 для этого следует выполнить команду File⇨Options⇨Add-ins, которая открывает окно, показанное на рис. 19.12. В программе Outlook 2007 следует выполнить команду Tools⇨Trust Center из меню и щелкнуть на кнопке Add-ins. Во всех остальных приложениях пакета Microsoft Office необходимо открыть меню File или Office и щелкнуть на кнопке Options, расположенной внизу окна, в котором размещено меню.

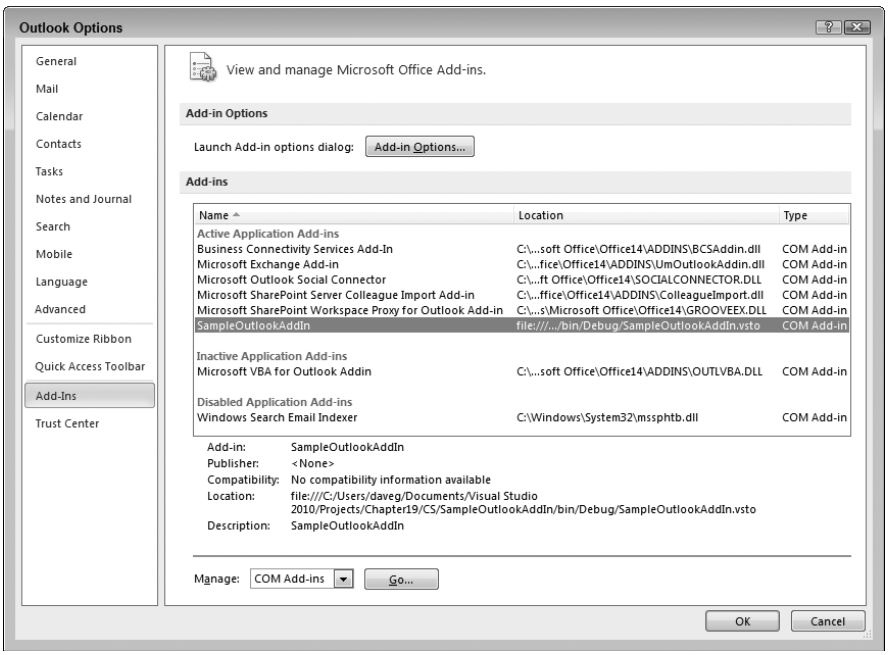


Рис. 19.12

Если пользовательское приложение зарегистрировано и загружено, то оно будет указано в списке Active Application Add-ins. Необходимо выбрать пункт COM Add-ins в раскрывающемся списке внизу окна и щелкнуть на кнопке Go. В результате откроется диалоговое окно COM Add-Ins, показанное на рис. 19.13, которое позволяет удалять надстройки из приложения.

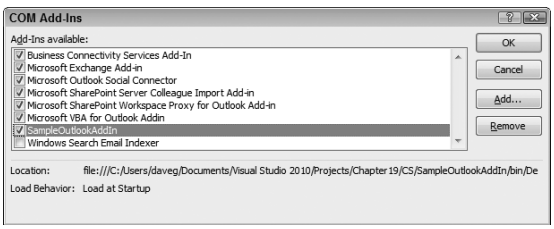


Рис. 19.13

Кроме того, пользователь может просто отключить надстройку, сбросив флажок в этом окне.

Отключение надстроек

Разрабатывая приложения для пакета Office, пользователь непременно сделает нечто, что вызовет генерацию необрабатываемого исключения и крах надстройки. Если крах надстройки произошел при загрузке, то приложение пакета Office отключит ее. Это называется *мягким отключением* (soft disabling).

Мягко отключенная надстройка не будет загружена и появится в окне Trust Center (см. рис. 19.12) в списке Inactive Application Add-ins. После повторной компиляции система Visual Studio 2010 подключит мягко отключенную надстройку автоматически. Кроме того, для повторного подключения надстройки можно использовать окно COM Add-Ins, показанное на рис. 19.13, в котором необходимо установить флажок, расположенный рядом с именем надстройки.

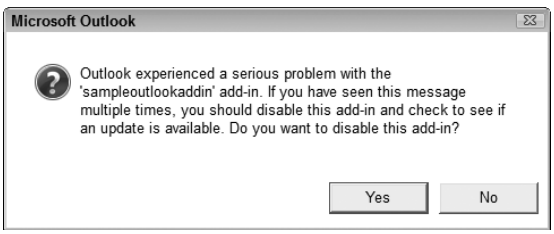


Рис. 19.14

Если крах потерпела не надстройка, а все приложение или пользователь остановил отладку во время работы конструктора или обработчика запуска, то надстройка будет помечена как подлежащая *жесткому отключению* (hard disabled). В следующий раз при запуске приложения из пакета Office пользователь увидит диалоговое окно, показанное на рис. 19.14. Выбрав команду Yes, он жестко отключит надстройку.

Если надстройка была отключена жестко, то ее невозможно снова подключить с помощью системы Visual Studio. При попытке отладить жестко отключенную надстройку, пользователь получит предупреждение о том, что надстройка была добавлена в список Disabled Items и не может быть загружена.

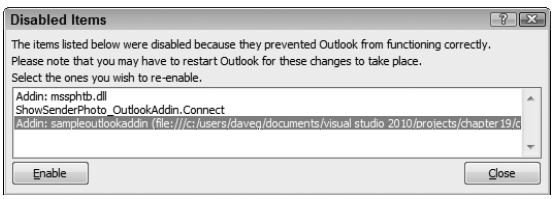


Рис. 19.15

Для того чтобы удалить надстройку из списка Disabled Items, необходимо запустить приложение из пакета Office и открыть окно Add-Ins, показанное на рис. 19.12. (Для этого следует выполнить команды File⇨Options⇨Add-ins в программе или Tools⇨Trust Center в программе Outlook 2007.) Выберите пункт Disabled Items в раскрывающемся списке внизу окна и щелкните на кнопке Go. В результате откроется окно Disabled Items, показанное на рис. 19.15. Выберите надстройку и щелкните на кнопке Enable, чтобы удалить ее из списка. Для того чтобы эти изменения были приняты, перезагрузите приложение.

Развертывание приложения из пакета Office

Существуют два способа развертывания приложений из пакета Office: с помощью традиционного проекта инсталляции MSI или технологии развертывания ClickOnce, встроенной в систему Visual Studio 2010.

В предыдущих версиях пакета VSTO настройка уровня доступа к коду была ручным процессом. Несмотря на то что пакет VSTO скрывает от пользователя много деталей реализации, для взаимодействия с пакетом Office ему по-прежнему необходимо вызывать код COM+. Поскольку исполняющая среда Common Language Runtime (CLR) не может установить уровень доступа к неуправляемому коду, она требует, чтобы любые приложения, вызывающие компоненты COM+, имели уровень безопасности Full Trust.

К счастью, технология ClickOnce для приложения из пакета Office, встроенная в систему Visual Studio 2010, автоматически разворачивается с уровнем безопасности Full Trust. Как и все другие приложения технологии ClickOnce, при каждом вызове она автоматически проверяет наличие обновлений.

Когда приложение Office развернуто, его необходимо упаковать вместе с необходимыми системами. Для приложений из пакета Office к ним относятся следующие программы.

- Windows Installer 3.1.
- .NET Framework 4, .NET Framework 4 Client Profile или .NET Framework 3.5.
- Visual Studio 2010 Tools for Office Run Time.

При работе на платформе NET Framework 3.5 необходимо также упаковать основные сборки взаимодействия Microsoft Office (PIA). Сборка PIA — это сборка, содержащая определения типов, реализуемых с помощью технологии COM. Сборки PIA для приложений Office 2007 и Office 2010 поставляются вместе с пакетом Visual Studio Tools for Office и автоматически включаются как ссылки при создании проекта. На рис. 19.16, *слева*, показана ссылка на сборку `Microsoft.Office.Interop.Outlook`, которая является сборкой PIA для приложения Outlook 2010.

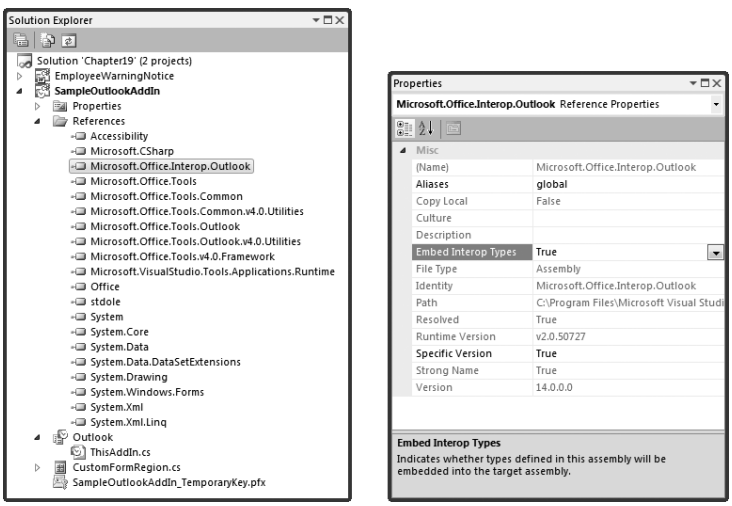


Рис. 19.16

При работе на платформе .NET Framework 4 сборки PIA разворачивать необязательно, потому что существует новая функциональная возможность — Type Equivalence. Если возможность Type Equivalence включена, то система Visual Studio будет встраивать необходимые сборки в целевую сборку как новое пространство имен, а система CLR гарантирует, что эти типы при выполнении приложения будут считаться эквивалентными.

Функциональная возможность Type Equivalence становится доступной для ссылки после того, как свойство Embed Interop Types будет установлено равным значению True, как показано на рис. 19.16, *справа*. Вместо включения всей сборки взаимодействия система Visual Studio будет встраивать только те их части, которые действительно использует. Это позволяет уменьшить и упростить пакеты развертывания.

Более подробно технология ClickOnce и проекты инсталляции описываются в главе 48.

Резюме

В главе рассмотрены основные функциональные возможности пакета Visual Studio Tools for Office. Теперь можно очень просто создавать полноценные приложения, используя приложения из пакета Microsoft Office, поскольку инструменты развертывания полностью интегрированы в систему Visual Studio 2010. Пользователь может создавать .NET-решения, добавляя к пользовательскому интерфейсу приложений Office свои собственные компоненты как на уровне приложения, так и на уровне документа. Это предоставляет пользователю беспрецедентный контроль над всеми программными продуктами из пакета Microsoft Office.

ЧАСТЬ V
Веб-приложения

- ГЛАВА 20. Веб-формы ASP.NET
- ГЛАВА 21. Платформа ASP.NET MVC
- ГЛАВА 22. Технология Silverlight
- ГЛАВА 23. Механизм Dynamic Data
- ГЛАВА 24. Коллекция программных продуктов SharePoint
- ГЛАВА 25. Платформа Windows Azure

20

Веб-формы ASP.NET

В ЭТОЙ ГЛАВЕ...

- Разница между проектами Web Site и Web Application
- Использование средств проектирования HTML и CSS для управления компоновкой веб-страниц
- Простая генерация высокофункциональных веб-приложений с серверным веб-управлением
- Добавление полноценных клиентских взаимодействий с веб-страницами с помощью языка JavaScript и библиотеки ASP.NET AJAX

Когда компания Microsoft выпустила первую версию платформы ASP.NET, одной из наиболее обсуждаемых стала возможность создавать полноценные веб-приложения точно так же, как обычное Windows-приложение. Абстракции, воплощенные в технологии ASP.NET, в сочетании с богатым инструментарием системы Visual Studio позволяют программистам быстро создавать высокофункциональные приложения, выполняемые в веб, не выходя из интегрированной среды разработки.

Платформа ASP.NET 2.0, выпущенная в 2005 году, содержала много новых функциональных возможностей, в частности, модель провайдера, позволяющую реализовать что угодно: от навигации по меню до аутентификации пользователя, более пятидесяти новых серверных элементов управления, каркас веб-портала и встроенные средства администрирования для веб-сайта. Эти усовершенствования еще больше упростили и ускорили процесс создания сложных веб-приложений.

Бо́льшая часть новых функциональных возможностей, реализованных в последних версиях платформы ASP.NET и системы Visual Studio, сосредоточены на усовершенствовании процесса разработки клиентской стороны. К этим возможностям относятся средства редактирования HTML Designer и CSS, улучшенная поддержка языка JavaScript с помощью технологии IntelliSens, сниппеты кодов на языках HTML и JavaScript, а также новые шаблоны проектов.

В этой главе мы рассмотрим процесс создания проектов типа Web Applications с помощью технологии ASP.NET в системе Visual Studio 2010, а также множество функциональных возможностей и компонентов, которые компания Microsoft включила в систему облегчения разработки веб-приложений (в некоторых случаях небольшого, но иногда существенного).

Проекты Web Application и Web Site

В версии Visual Studio 2005 появился совершенно новый проект — Web Site. Основанием для введения нового типа проекта было предположение, что веб-сайты кардинально отличаются от других типов приложений, а их разработчики отличаются от разработчиков обычных проектов, и, следовательно, они извлекут выгоду из появления новой модели. Несмотря на то что компания Microsoft проделала огромную работу, восхваляя новый тип проекта, многие разработчики обнаружили, что работать с ним непросто, и ясно выразили компании свое недовольство.

К счастью, компания Microsoft услышала сигналы обратной связи и вскоре выпустила свободно распространяемую надстройку для системы Visual Studio, которая поддерживала новый тип проекта — Web Application. Эта надстройка была также включена в пакет Service Pack 1 для системы Visual Studio 2005.

Основные различия между этими двумя типами проекта весьма значительны. Фундаментальное изменение заключается в том, что проект Web Site не содержит файла проекта системы Visual Studio (файла с расширением `.csproj` или `.vbproj`), в то время как проект Web Application содержит такой файл. Благодаря этому не существует главного файла, содержащего список всех файлов в проекте Web Site. Вместо этого файл решения в системе Visual Studio содержит ссылку на корневую папку проекта Web Site, а содержимое и компоновка прямо выводятся из файлов и папок нижележащего уровня. Если скопировать новый файл в папку нижележащего уровня в проекте Web Site, используя программу Windows Explorer, то этот файл по определению будет принадлежать проекту. В проекте Web Application пользователь должен явно добавить все файлы в проект, находясь в среде Visual Studio.

Другое основное отличие заключается в способе компиляции проекта. Проекты Web Application компилируются практически так же, как любой другой проект в системе Visual Studio. Код компилируется в единую сборку, которая хранится в каталоге `\bin`, соответствующем веб-приложению. Как и для всех остальных проектов в системе Visual Studio, пользователь может управлять процессом сборки с помощью страниц свойств, задавать имя выходной сборки, а также добавлять правила для действий, выполняемых до и после процесса сборки.

В противоположность этому в проекте Web Site все классы, которые не относятся к странице или пользовательскому элементу управления, компилируются в общую сборку. Страницы и элементы управления компилируются динамически по мере необходимости в набор отдельных сборок.

Основное преимущество более подробной детализации сборок заключается в том, что пользователь не обязан каждый раз перекомпилировать весь веб-сайт, когда изменяется хотя бы одна страница. Вместо этого перекомпилируются только те сборки, в которых внесены изменения (или зависящие от этих изменений). Это экономит массу времени.

Компания Microsoft пообещала и впредь поддерживать оба типа проектов, Web Site и Web Application, во всех будущих версиях системы Visual Studio.

Итак, какой же тип проекта выбрать? Официальная позиция компании Microsoft звучит так: "по ситуации". Это очень прагматичный, но не очень полезный ответ. Все сценарии отличаются друг от друга, и пользователь всегда должен тщательно взвешивать все альтернативы в контексте своих требований и существующего окружения. Однако сообщения, поступающие от разработчиков, использующих технологию .NET в течение последних нескольких лет, и опыт авторов показывают, что в большинстве случаев наилучшим выбором является проект Web Application.

Создание веб-проектов

Кроме стандартных проектов ASP.NET Web Application и Web Site, система Visual Studio 2010 обеспечивает поддержку и реализует шаблоны нескольких специализированных сценариев, касающихся веб-приложений. К ним относятся веб-сервисы, WCF-сервисы, библиотеки серверных элементов управления и приложения для создания отчетов. Но, прежде чем перейти к их обсуждению, мы должны понять, как создаются обычные проекты.

Создание проекта Web Site

Как указывалось ранее, процесс создания проекта Web Site в системе Visual Studio 2010 немного отличается от процесса создания обычного проекта для системы Windows. При создании обычных Windows-приложений и служб пользователь выбирает тип проекта, имя решения и щелкает на кнопке OK. Каждому языку соответствует свой набор проектных шаблонов, и у пользователя нет полной свободы выбора при создании проекта. Проект Web Site отличается тем, что пользователь может создать проект в разных местах, от локальной файловой системы до разных адресов FTP и HTTP, определенных в настройках системы, включая локальный сервер Internet Information Services (IIS) или удаленные папки FTP.

Из-за этой очень важной особенности процесса создания таких проектов компания Microsoft выделила проектному шаблону Web Site отдельные командную строку и диалоговое окно. Команда New Web Site из подменю File⇨New открывает диалоговое окно New Web Site, в котором можно выбрать желаемый тип проектного шаблона (рис. 20.1).



Если веб-проект не слишком большой и не состоит из сотен страниц, то в принципе не сложно перейти от проекта Web Site к проекту Web Application, и наоборот. Поэтому особенно долго над этим решением раздумывать не стоит. Выберите тип проекта, и если в будущем возникнут трудности, переходите к альтернативе.

Чаще всего пользователи выбирают шаблон проекта ASP.NET Web Site. С его помощью можно создать веб-сайт, наполненный веб-приложением запуска, которое гарантирует, что начальное приложение имеет правильную структуру. Этот шаблон создает проект, демонстрирующий, как использовать главную страницу, меню, счетчики, каскадные таблицы стилей CSS и библиотеку jQuery JavaScript.

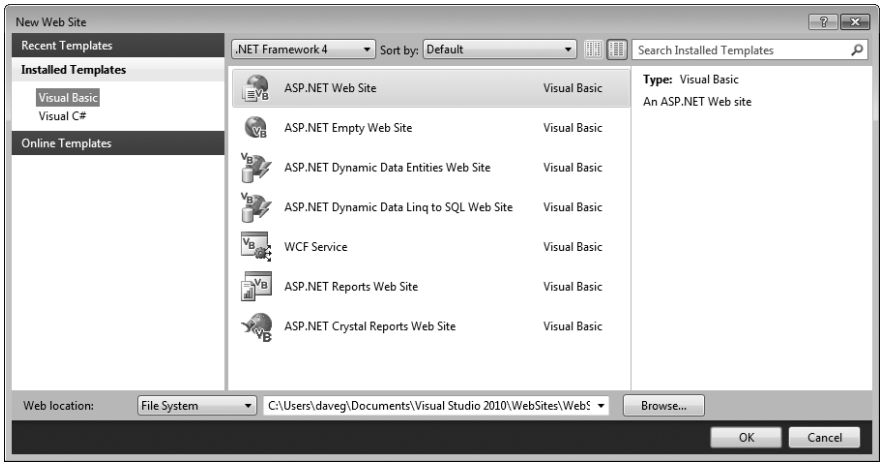


Рис. 20.1

Кроме проектного шаблона ASP.NET Web Site, существует шаблон Empty Web Site, не создающий ничего, кроме пустой папки и ссылки на файл решения. Остальные шаблоны, по большей части представляющие собой варианты шаблона Web Site, будут рассмотрены в этой главе позднее. Независимо от типа создаваемого веб-проекта, в нижней части диалогового окна пользователь может выбрать, где создавать проект.

По умолчанию система Visual Studio ожидает, что пользователь будет разрабатывать веб-сайт или сервис локально, т.е. используя обычную файловую систему. Для текущего пользователя по умолчанию выделяется папка `My Documents/Visual Studio 2010/WebSites`, но этот адрес можно изменить, выбрав альтернативное место в раскрывающемся списке или щелкнув на кнопке Browse.

Раскрывающийся список Location также содержит варианты HTTP и FTP. Выбор варианта HTTP или FTP изменяет имя файла в поле ввода на префикс `http://` или `ftp://`, который предваряет адрес URL. Пользователь может либо ввести корректный адрес, либо щелкнуть на кнопке Browse, чтобы изменить местоположение проекта.

Диалоговое окно Choose Location (рис. 20.2) открывается, когда пользователь щелкает на кнопке Browse. Оно позволяет ему указать, где должен храниться проект. Обратите внимание на то, что этот адрес не обязан совпадать с адресом развертывания проекта, поскольку при разработке проекта разработчик может еще не знать, где именно он будет развернут.

Команда File System позволяет просматривать структуру папок системы, включая папки My Network Places. По мере необходимости она дает пользователю возможность создавать папки нижележащего уровня. Это простейший способ указать место для файлов веб-проектов и найти их впоследствии.



Несмотря на то что пользователь может указать, где создать файлы проекта, по умолчанию файл решения для нового пользователя создается в новой папке, вложенной в папку `My Documents/Visual Studio 2010/Projects`. Файл решения можно переместить в любую другую папку без какого-либо вреда для проекта.

При использовании локального сервера IIS для отладки своего проекта Web Site пользователь может выбрать вариант File System и указать папку `wwwroot` для создания веб-сайта. Однако намного лучше использовать тип Local IIS и углубляться в поисках требуемого места внутрь папок Default Web Site. Этот интерфейс позволяет пользователю просматривать виртуальную папку, ссылающуюся на веб-сайт. Физически виртуальная папка внутри папки `wwwroot` не существует, но фактически является синонимом любой папки в файловой системе или в сети. Пользователь может создать собственное приложение в папке Web Application или создать новый виртуальный каталог, в котором можно искать физический файл или указывать синоним для списка веб-сайтов.

На рис. 20.2 показан тип FTP Site, позволяющий анонимно или открыто регистрироваться на удаленной FTP-сайте. Когда пользователь щелкает на кнопке Open, система Visual Studio сохраняет настройки FTP, которые будут использоваться при создании проекта. Следует помнить, что эти настройки не проверяются, пока пользователь не попытается создать проектные файлы и сохранить их в указанном месте.

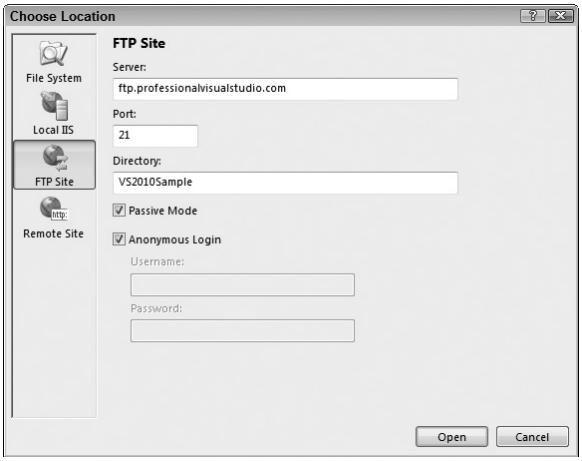


Рис. 20.2



Пользователь может сохранить свои проектные файлы на FTP-сервере, к которому он имеет доступ, даже если на этом FTP-сайте не инсталлирована платформа .NET. Однако пользователь не сможет запустить эти файлы без платформы .NET, поэтому такие сайты можно использовать только как хранилища файлов.

Последним типом для местоположения проекта является Remote Site, позволяющий связываться с удаленным сервером, на котором установлены расширения программы FrontPage. Если такой сайт существует, пользователь может просто указать, где он хочет сохранить новый проект, и система Visual Studio 2010 подтвердит, что она может создать эту папку с помощью расширений программы FrontPage.



Как только пользователь выбрал место для хранения проекта, он должен щелкнуть на кнопке OK, чтобы заставить систему Visual Studio 2010 создать файлы проекта и сохранить их в желательном месте. После завершения инициализации веб-приложения система Visual Studio откроет страницу `Default.aspx` и заполнит инструментальное окно компонентами, доступными для проектирования веб-приложений.

Проект Web Site имеет лишь небольшое подмножество настроек, которые можно установить на страницах свойств проектов других типов, как показано на рис. 20.3. Для того чтобы получить доступ к этим настройкам, необходимо щелкнуть правой кнопкой мыши на проекте и выбрать команду Property Pages.

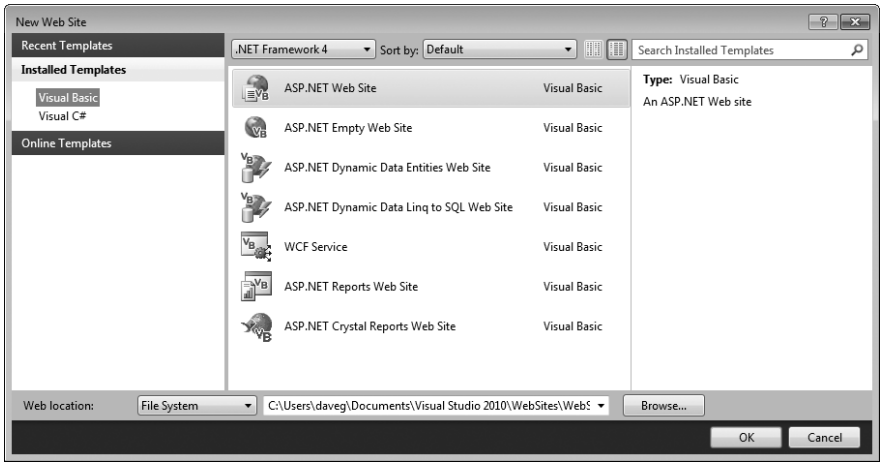


Рис. 20.3

Страница свойств References, показанная на рис. 20.3, позволяет пользователю определить ссылки на внешние сборки или веб-службы. Если добавить бинарную ссылку в сборку, которой нет в кэше Global Assembly Cache (GAC), то эта сборка будет скопирована в папку `\bin` пользовательского веб-проекта вместе с файлом, имеющим расширение `.refresh`, который представляет собой небольшой текстовый файл, содержащий путь к первоначальному местоположению сборки. Каждый раз при создании веб-сайта система Visual Studio сравнивает текущую версию сборки из папки `\bin` с ее версией, находящейся в исходном месте, и по необходимости обновляет ее. Если в проекте много внешних ссылок, то компиляция существенно замедляется. По этой причине мы рекомендуем удалять файлы с расширением `.refresh`, ассоциированные с любыми ссылками на сборку, что, к сожалению, случается довольно часто.

Окна свойств Build, Accessibility и Start Options частично определяют способ сборки и запуска веб-сайта для отладки. Настройки доступа будут рассмотрены в этой главе позднее. Остальные настройки на данном этапе дополнительных пояснений не требуют.

Страница свойств MSBuild Options содержит ряд интересных и сложных настроек для веб-приложений. Если установить переключатель Allow This Precompiled Site в положение Updatable, то все содержимое страниц с расширениями `.aspx` и `.ascx` будет скомпилировано в сборку вместе с основным кодом. Это может оказаться полезным, если пользователь хочет защитить пользовательский интерфейс от модификации. Флажок Use Fixed Naming and Single Page Assemblies указывает, что каждая страница должна компилироваться в отдельную сборку, которая не совпадает со стандартными.

Страница свойств Silverlight Applications позволяет пользователю добавлять проект платформы Silverlight или ссылаться на него, чтобы встроить его в веб-сайт. Более подробно она будет рассмотрена в главе 22.

Создание проекта Web Application

Процесс создания проекта Web Application в системе Visual Studio 2010 почти не отличается от процесса создания любого другого проекта. Сначала необходимо выполнить команду File⇨New⇨Project, чтобы открыть диалоговое окно New Project, показанное на рис. 20.4. Отфильтровав типы проектов по языкам, а затем по категориям веб, следует выбрать шаблоны, напоминающие шаблоны, доступные для проектов Web Site.

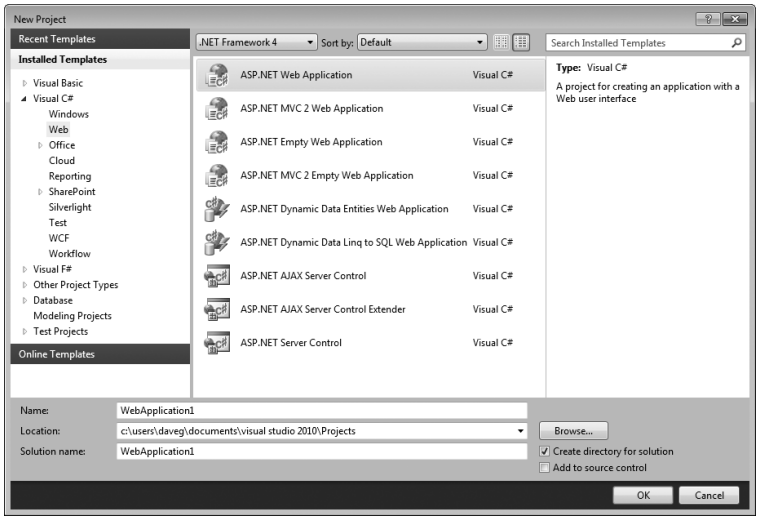


Рис. 20.4

Примечательным является то, что среди доступных шаблонов для создания отчетов нет проекта Web Application. Однако проектный шаблон Web Application содержит шаблоны для создания нескольких разных типов серверных элементов управления.

Щелкнув на кнопке OK, пользователь создает новый проект Web Application, имеющий немного больше компонентов, чем проекты Web Site. Он содержит файл `AssemblyInfo`, папку References и узел My Project, расположенный на уровень ниже узла Visual Basic, или узел Properties, расположенный на уровень ниже узла C#.

Страницы свойств проекта Web Application можно просмотреть, дважды щелкнув на узле Properties или My Project. Страницы свойств включают в себя дополнительную страницу Web, показанную на рис. 20.5.

Все настройки веб-страниц связаны с отладкой веб-приложения на платформе ASP.NET и описываются в главах 42 и 43.

Другие веб-проекты

Кроме стандартных проектных шаблонов ASP.NET Web Site и Web Application, существуют решения, предназначенные для реализации более сложных сценариев.

- ASP.NET MVC 2 Web Application. Создает веб-приложение с помощью архитектуры Model-View-Controller (MVC). Она доступна только для проектов Web Application и описывается в главе 21.
- ASP.NET Web Service. Создает обычный веб-сервис с именем `Sevice.asmx`, содержащий обычный веб-метод. На платформе .NET Framework 3.5 и в предшествующих ей версиях эта возможность доступна только для проектов Web Application.
- ASP.NET Reports Web Site. Создает веб-сайт на платформе ASP.NET с отчетом (`.rdlc`) и элементом управления ReportViewer, связанным с этим отчетом. Эта возможность доступна только для проектов проект Web Site и описывается в главе 30.

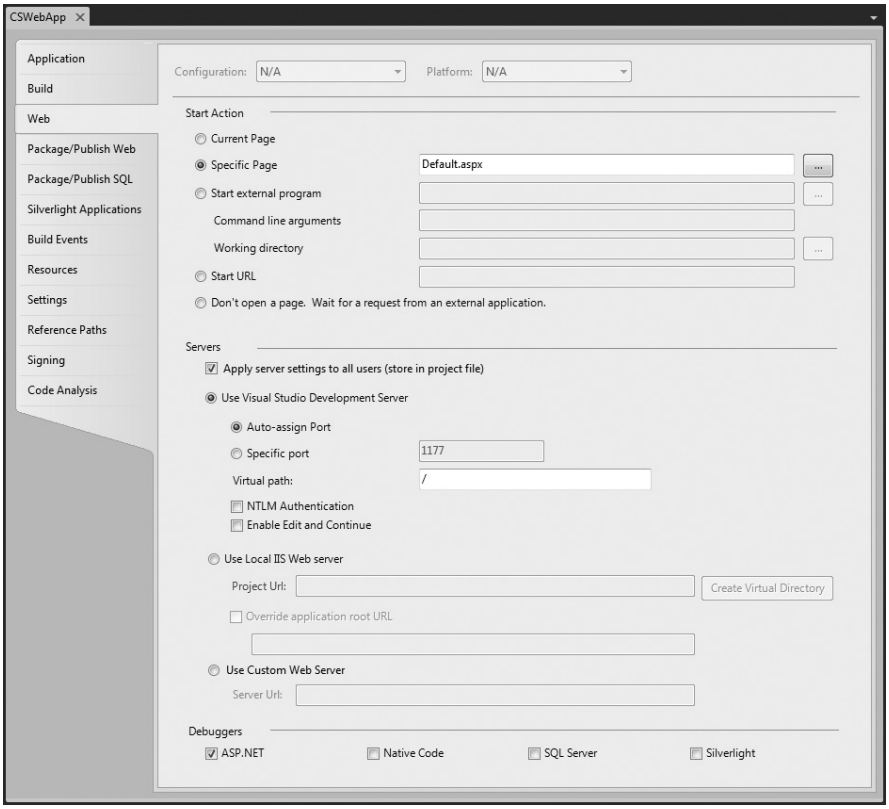


Рис. 20.5

- ASP.NET Crystal Reports Web Site. Создает веб-сайт на платформе ASP.NET с образцом отчета Crystal Report. Эта возможность доступна только для проектов Web Site.
- ASP.NET Server Control. К серверным элементам управления относятся стандартные элементы, такие как кнопки и поля ввода, а также элементы управления специального назначения, такие как календарь, меню и компоненты для отображения данных в виде иерархических структур. Этот шаблон доступен только как проект Web Application.
- ASP.NET AJAX Server Control. Содержит элементы управления веб-сервера, позволяющие добавлять на веб-страницы для платформы ASP.NET функциональные возможности библиотеки AJAX. Эта возможность доступна только для проектов Web Application.
- ASP.NET AJAX Server Control Extender. Расширяющие элементы управления AJAX для платформы ASP.NET улучшают функциональные возможности клиента и свойства стандартных серверных элементов управления. Эта возможность доступна только для проектов Web Application.
- Dynamic Data Web Site и Dynamic Data Web Application. Эти шаблоны обеспечивает быстрое создание веб-приложений, связанных с базой данных и использующих компонент LINQ to SQL или технологию Entity Framework. Они доступны для проектов Web Site и Web Application и обсуждаются в главе 23.

Время от времени компания Microsoft выпускает дополнительные проектные шаблоны, которые можно загружать отдельно. Например, в системе Visual Studio 2008 были выпущены проекты типов ASP.NET MVC и Silverlight 2.0.

Наборы для начинающих, общественные проекты и приложения, и общедоступные приложения

Для того чтобы освоить новую технологию, лучше всего разобрать простое приложение. Веб-сайт Microsoft ASP.NET содержит список наборов для начинающих и общественных проектов на веб-странице `http://www.asp.net/community/projects`. Эти веб-приложения являются превосходной коллекцией программ, демонстрирующих самые лучшие методы использования компонентов и шаблонов для платформы ASP.NET.

К сожалению, многие пакеты для начинающих не поддерживаются старыми версиями платформы .NET Framework. Тем не менее они очень полезны, поскольку демонстрируют широкий спектр современных методов и технологий для платформы ASP.NET, включая множество тем CSS, страницы мастер–деталь и пользовательские средства управления.

Сайт Microsoft ASP.NET содержит также список популярных общедоступных проектов, построенных на платформе ASP.NET. Одним из наиболее полных проектов является приложение DinnerNow.net, доступное на веб-сайте `http://www.dinnernow.net/`. Несмотря на то что оно относится к категории приложений с открытым исходным кодом, в нем действительно реализованы новейшие технологии компании Microsoft.

Приложение DinnerNow.net — это виртуальный рынок, на котором потребители могут заказывать еду из местных ресторанов для доставки себе домой или в офисы. Кроме новейших компонентов платформы ASP.NET, оно демонстрирует использование технологий IIS7, ASP.NET AJAX Extensions, LINQ, Windows Communication Foundation, Windows Workflow Foundation, Windows Presentation Foundation, Windows Powershell и .NET Compact Framework.

Другим превосходным местом, где можно найти множество отличных примеров с открытым исходным кодом, является веб-сайт CodePlex, на котором компания Microsoft хранит общедоступные проекты. Веб-сайт CodePlex расположен по адресу `http://www.codeplex.com/` и представляет собой поистине неиссякаемый источник хороших, плохих и ужасных проектов компании Microsoft с открытым исходным кодом.

Проектирование веб-форм

Одним из самых важных преимуществ системы Visual Studio 2010 с точки зрения веб-разработчиков является визуальное проектирование веб-приложений. Редактор HTML Designer позволяет изменять положение, заполнять и задавать поля в окне Design, используя средства компоновки. Он также поддерживает режим разделения экрана, дающий пользователю возможность одновременно работать с проектом и размечать веб-форму. Система Visual Studio 2010 поддерживает также множество средств редактирования каскадных таблиц стилей CSS для проектирования макетов и управления стилем веб-контента.

Редактор HTML Designer

Редактор HTML Designer в системе Visual Studio — это одно из главных средств, благодаря которым разработка приложения для технологии ASP.NET стала такой легкой. Поскольку этот редактор знает, как визуализировать элементы HTML и серверные элементы управления на платформе ASP.NET, пользователь может просто перетаскивать компоненты из инструментального окна Toolbox в рабочую область редактора HTML Designer и быстро создавать пользовательский веб-интерфейс. Кроме того, он может быстро переключаться с просмотра разметки на языке HTML на визуальное представление веб-страницы или пользовательского элемента управления, и наоборот.

Модификации, внесенные в меню View интегрированной среды разработки, являются прекрасным примером того, как система Visual Studio учитывает контекст и предоставляет пользователю нужные средства в зависимости от того, что он в данный момент делает. Если пользователь редактирует веб-страницу в окне конструктора, то в меню появятся дополнительные команды, позволяющие уточнять внешний вид рабочей области (рис. 20.6).

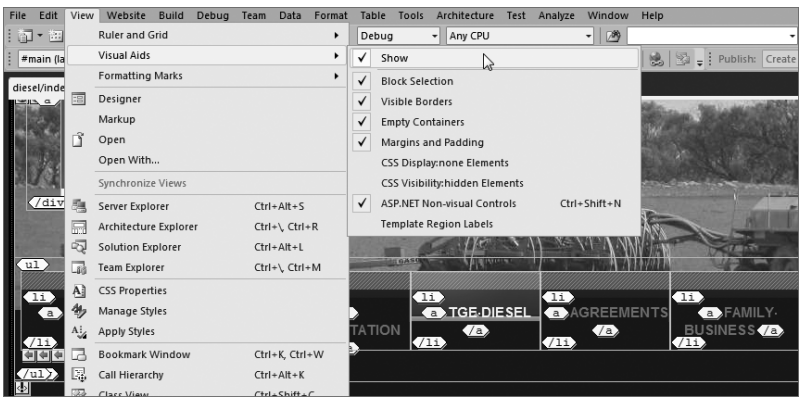


Рис. 20.6

Три подменю, расположенные в верхней части меню View, — Ruler and Grid, Visual Aids и Formatting Marks — предоставляют полный набор полезных инструментов, облегчающих компоновку элементов управления и элементов HTML на веб-странице.

Например, если в подменю Visual Aids установлен переключатель Show, вокруг контейнерных элементов управления и дескрипторов HTML, таких как `<table>` и `<div>`, рисуется серый контур, и пользователю легко определить их местоположение на форме. Кроме того, это подменю позволяет рисовать цветные тени, чтобы обозначить поля и заполнение вокруг элементов HTML и серверных элементов управления. Аналогично, в подменю Formatting Marks можно включить переключатель, чтобы отобразить на экране имена дескрипторов HTML, обрывы строки, пробелы и многое другое. Реакцию редактора HTML Designer на положение этих переключателей можно увидеть на рис. 20.6.

Кроме того, редактор HTML поддерживает режим разделения экрана (рис. 20.7), который позволяет одновременно видеть и разметку HTML, и визуальную компоновку формы. Этот режим включается, когда пользователь открывает окно в режиме проектирования и щелкает на кнопке Split, расположенной в левом нижнем углу окна редактора HTML Designer.



Рис. 20.7

Когда пользователь выбирает элемент управления или элемент HTML в рабочей области, редактор HTML Designer подсвечивает их в разметке HTML. Аналогично, если внутри разметки переместить курсор в другое место, редактор выделит соответствующий элемент управления в рабочей области.

Если пользователь изменил какой-нибудь элемент в рабочей области, то это изменение немедленно отразится на HTML-разметке. Однако изменения разметки не всегда непосредственно отражаются на рабочей области редактора HTML Designer. Вместо этого в информационной строке, расположенной в верхней части окна Design, появляется сообщение, что визуальное представление в данный момент не синхронизировано с панелью Source (рис. 20.8). Пользователь может либо щелкнуть на информационной строке, либо нажать комбинацию клавиш <Ctrl+Shift+Y>, чтобы синхронизировать эти панели. Кроме того, панели синхронизируются при сохранении изменений.

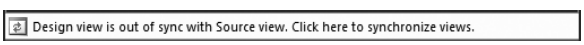


Рис. 20.8



При работе с широким монитором можно включить режим вертикального разделения экрана, чтобы воспользоваться преимуществом высокого разрешения. Для этого необходимо выбрать команду Tools⇨Options и щелкнуть на узле HTML Designer на панели TreeView. Для конфигурирования поведения редактора HTML Designer можно использовать много настроек, в частности переключатель Split Views Vertically.

Другой заслуживающей внимания особенностью редактора HTML Designer является *навигатор по дескрипторам с помощью навигационных цепочек* (tag navigator bread crumb), появляющийся в нижней части окна визуального редактора. Этот механизм, существующий также в визуальных редакторах, использующих технологии Silverlight и WPF, отображает иерархию текущего элемента управления или элемент управления и его наследников. Механизм навигационных цепочек отображает на экране тип эле-

мента управления, идентификатор или CSS-класс, если они были определены. Если путь дескриптора слишком длинный и не помещается в окне редактора HTML Designer, то список укорачивается и на экране появляется группа стрелок, с помощью которых можно прокручивать путь дескриптора.

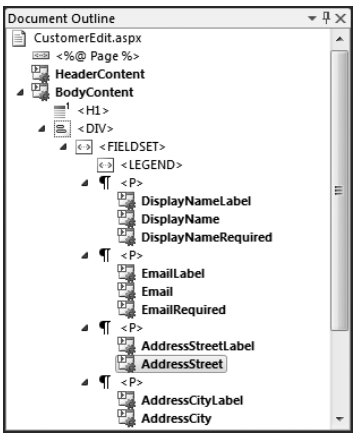


Рис. 20.9

Механизм навигации по дескрипторам с помощью навигационных цепочек выводит на экран путь, который начинается только на текущем элементе управления и заканчивается на его родительском элементе высшего уровня. Он не содержит никаких элементов, находящихся за пределами этого пути. Если пользователь хочет видеть все элементы в текущем документе, то он должен использовать окно Document Outline, показанное на рис. 20.9. Для того чтобы открыть это окно, необходимо выбрать команду Select View⇨Other Windows⇨Document Outline. Когда пользователь выбирает компонент или элемент правления в окне Document Outline, тот подсвечивается на панелях Design и Source редактора HTML Designer. Однако при выборе элемента в редакторе HTML Designer он не выделяется в окне Document Outline.

Позиционирование элементов управления и элементов HTML

Одним из наиболее сложных аспектов создания веб-страниц является позиционирование HTML-элементов. Для того чтобы задать местоположение элемента управления, необходимо задать несколько атрибутов, в частности, какой вид позиционирования (относительное или абсолютное) использует элемент, плавающий он или нет, какой z-index он имеет, а также указать ширину заполнения и поля.

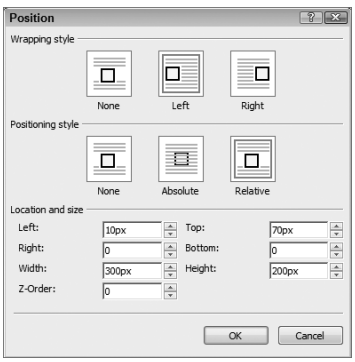


Рис. 20.10

К счастью, пользователю не обязательно изучать синтаксические конструкции и имена всех этих атрибутов и вручную вводить их в разметку. Как обычно, на помощь приходит интегрированная среда разработки Visual Studio. Выберите компонент или элемент управления, который хотите разместить на панели Design. Затем выберите команду Format⇨Position из меню и откройте окно Position, показанное на рис. 20.10.

Щелкните на кнопке OK, чтобы сохранить в атрибуте стиля элемента HTML выбранный стиль упаковки и позиционирования, а также все значения, касающиеся его местоположения и размера.

Если элемент имеет относительную или абсолютную позицию, то его местоположение на панели Design можно изменить. Однако следует очень осторожно перетаскивать элементы в окне редактора HTML Designer, потому что может произойти неприятность! Как только вы выберете компонент или элемент управления на панели Design, в верхнем левом углу элемента появится белый дескриптор. Он отображает на экране тип элемента, а также его идентификационный номер и имя класса, если они были определены.

Если возникла необходимость передвинуть элемент, имеющий относительную или абсолютную позицию, его можно перетащить в новое место, используя белый управляющий дескриптор. Если перетаскивать элемент, используя его самого, то его позиция в тексте программы на языке HTML не изменится, а сам он будет просто переставлен в новую строку кода в исходном файле.

На рис. 20.11 показана кнопка, имеющая относительную позицию, местоположение которой было изменено. Фактическое положение этого элемента в документе показано в виде пустого голубого прямоугольника. Однако этот элемент управления был смещен на 45 пикселей вниз и на 225 пикселей вправо от своей исходной позиции. Настоящий элемент управления показан в его новой позиции, а голубые горизонтальные и вертикальные линии означают, что его позиция является относительной. Эти линии и исходная позиция элемента демонстрируются на экране только при его выборе.

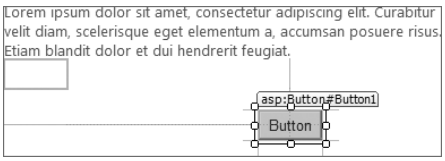


Рис. 20.11



Если элемент управления использует абсолютное позиционирование, то подсвечивается позиционный контейнер и на экране появляются две дополнительные линии, проходящие от нижнего и правого края элемента до границы контейнера.

Последним методом компоновки, который мы обсудим, является настройка заполнения и полей элемента HTML. Многие разработчики веб-приложений сначала путают эти атрибуты. Ситуация усугубляется еще и тем, что разные браузеры по-разному прорисовывают элементы с этими атрибутами. Несмотря на то что не все HTML-элементы имеют границу, мы, *как правило*, рассматриваем заполнение как пространство, расположенное внутри границы, а полями считаем пространство, расположенное за пределами границы.

Если внимательно присмотреться к редактору HTML Designer, то можно заметить серые короткие линии, проходящие от всех четырех углов элемента управления в горизонтальном и в вертикальном направлениях. Они видимы, только когда элемент выделен на панели Design. Называются они *манипуляторами поля* (margin handles) и позволяют задавать ширину полей. Для того чтобы увеличить или уменьшить ширину поля, нужно установить курсор мыши над манипулятором, подождать, пока он не изменится, а потом передвинуть его в желательном направлении (рис. 20.12).

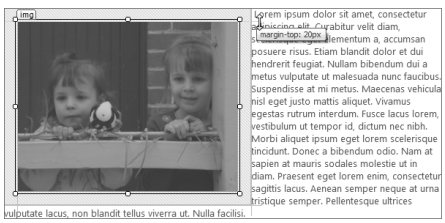


Рис. 20.12

В заключение отметим, что в редакторе HTML Designer можно задать параметры заполнения вокруг элемента. Если выбрать элемент и затем нажать клавишу <Shift>, то манипуляторы полей станут манипуляторами заполнения. Удерживая нажатой клавишу <Shift>, можно перетащить эти манипуляторы, чтобы увеличить или уменьшить ширину заполнения. После того как клавиша <Shift> будет отпущена, они снова превратятся в манипуляторы поля. На рис. 20.12 показано, как выглядит HTML-элемент изображения в редакторе HTML Designer, когда ширина полей и заполнения задана для всех четырех сторон.

На первый взгляд эти настройки полей и заполнения могут выглядеть странно, поскольку они не очень согласованы. Для того чтобы увеличить верхнее и левое поля, необходимо перетащить манипуляторы на элемент, а для того чтобы увеличить верхнее и левое заполнение, необходимо перетащить манипуляторы за пределы элемента. Просто чтобы еще больше все запутать, отметим, что перетаскивание нижнего и правого манипуляторов за пределы элемента увеличивает ширину как полей, так и заполнения.

Создав компоновку HTML и разместив элементы в требуемых местах, можно применить новые средства CSS, чтобы переместить компоновку со страницы и сохранить ее как внешний стиль. Эти инструменты будут рассмотрены в этой главе позднее.

Форматирование элементов управления и HTML-элементов

Кроме диалогового окна Position, рассмотренного в предыдущем разделе, в системе Visual Studio 2010 есть инструментальная панель и ряд диалоговых окон, позволяющих форматировать элементы управления и HTML-элементы на веб-странице.

Инструментальная панель Formatting, показанная на рис. 20.13, обеспечивает легкий доступ к большинству возможностей форматирования. Список, расположенный в верхнем левом углу, определяет порядок применения параметров форматирования к элементу управления и содержит опции, позволяющие определять стили или новые правила CSS. Следующий раскрывающийся список содержит все HTML-элементы. которые можно применить к тексту, включая заголовки от `<h1>` до `<h6>`, `<ul>`, `<ol>` и `<blockquote>`.

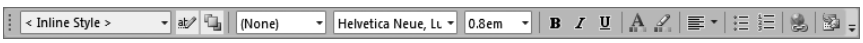


Рис. 20.13

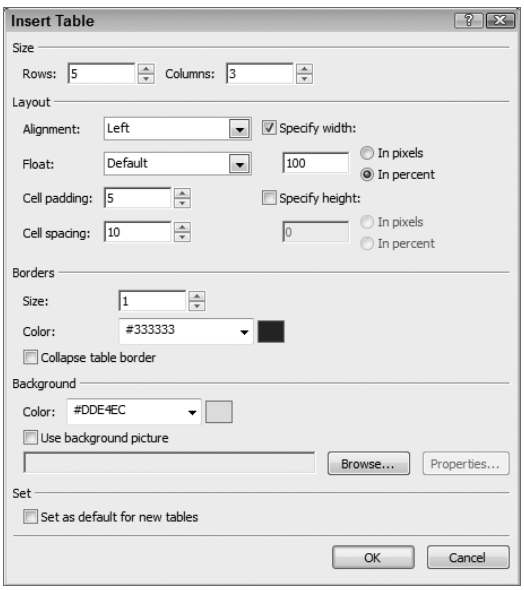


Рис. 20.14

Большинство других диалоговых окон для форматирования перечислены в меню Format. К ним относятся окна для настройки цветов переднего плана и фона, шрифта, выравнивания, маркер и нумерация. Эти диалоговые окна похожи на окна, существующие в любой версии текстового процессора Word или в интерфейсе, построенном по принципу WYSIWYG, а их использование вполне очевидно.

Диалоговое окно Insert Table, показанное на рис. 20.14, позволяет легко определять компоновку и внешний вид новой HTML-таблицы. Для того чтобы создать ее, необходимо указать курсором мыши на требуемое место в рабочей области визуального редактора и выбрать команду Table⇨Insert Table.

Последний и довольно полезный элемент диалоговых окон Insert Table и Font расположен под цветовой палитрой. Кроме списка стандартных цветов Standard Colors, существует также список Document Colors, показанный на рис. 20.15. Он содержит все цвета, которые так или иначе можно применить к текущей странице, например, цвет переднего плана, фона или границ. Это позволяет пользователю не запоминать значения RGB для цветовой схемы, выбранной для своей страницы.

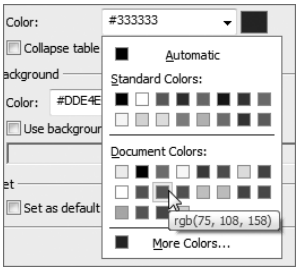


Рис. 20.15

Инструменты CSS

Когда-то разметка HTML на типичной веб-странице представляла собой смесь разметок содержимого и представления. Веб-страницы допускали свободное использование дескрипторов HTML, которые определяли, *как* должно быть прорисовано содержимое, например `<font>`, `<center>` и `<big>`. В настоящее время такое проектирование считается недопустимым, так как желательно, чтобы HTML-документы определяли только содержимое веб-страницы, упакованное в семантические дескрипторы, такие как `<h1>`, `<ul>` и `<div>`. Элементы, требующие специальных правил представления, должны иметь атрибут `class`, а вся информация о стилях должна храниться во внешней каскадной таблице стилей CSS.

Система Visual Studio 2010 имеет несколько функциональных возможностей, обеспечивающих множество вариантов редактирования CSS в интегрированной среде. Как было показано в предыдущем разделе, пользователь может выполнить бо́льшую часть работы по компоновке и стилизации содержимого с помощью панели Design. Она поддерживается окнами Manage Styles, Apply Styles и CSS Properties, доступными из меню View, когда открыт редактор HTML Designer.

Окно Manage Styles содержит список всех стилей CSS, представляющих собой CSS-файлы внутренних, внедренных или внешних стилей, связанных с текущей страницей. Цель этого инструментального окна заключается в предоставлении пользователю полного обзора правил CSS для конкретной страницы и возможности редактировать и управлять этими CSS-классами.

Все стили перечисляются на панели TreeView вместе с листами стилей, образуя верхние уровни иерархии, показанной на рис. 20.16. Стили перечисляются в порядке их следования в файле листа стилей. Их можно перетаскивать и даже перемещать с одного листа стилей на другой.

Если курсор мыши задержать над стилем, появится подсказка с описанием CSS-свойств данного стиля (рис. 20.16). Раскрывающееся меню Options позволяет фильтровать лист стилей, чтобы видеть только те из них, которые можно применить к элементам на текущей странице, или, если какой-то элемент выделен в редакторе HTML Designer, только те стили, которые относятся к выделенному элементу.

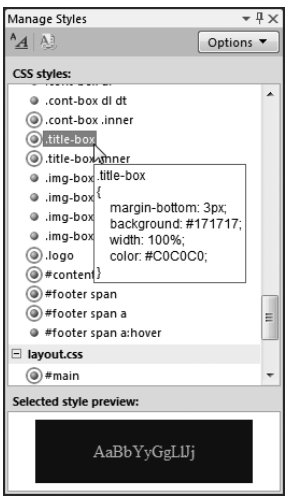


Рис. 20.16

Окно Manage Styles использует набор пиктограмм, позволяющих предоставить пользователю дополнительную визуальную информацию о типе каждого стиля. Пиктограммы, следующие за именами стилей, имеют разные цвета: красная точка означает стиль, основанный на атрибуте ID, зеленая точка — стиль, основанный на классе, синяя точка — стиль, основанный на элементе, а желтая точка — внедренный стиль.

Кружок вокруг точки означает, что стиль используется для текущей страницы. Например, на рис. 20.16 легко увидеть, что CSS-класс `title-box` используется на активной веб-странице, а класс `img-box` на ней не используется. В заключение отметим, что символ @ обозначает импортированную внешнюю таблицу CSS.

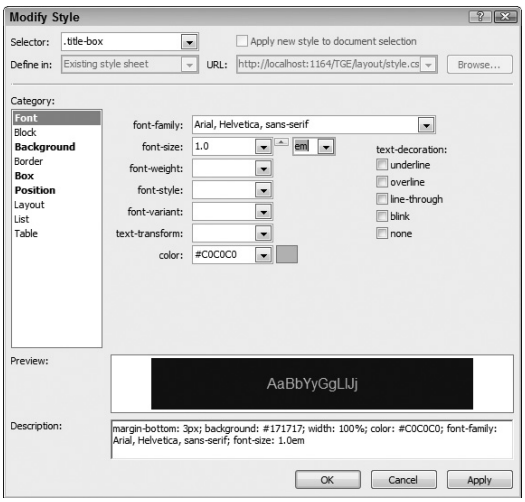


Рис. 20.17

Когда пользователь щелкает правой кнопкой мыши в окне Manage Styles, он может создать новый стиль с нуля, создать новый стиль на основе выбранного стиля или модифицировать выбранный стиль. После выбора любого из этих вариантов открывается диалоговое окно Modify Style, показанное на рис. 20.17. Это диалоговое окно реализует интуитивно понятный способ определения или модификации CSS-стиля. Свойства стиля группируются в хорошо известные категории: Font, Border и Position, а предварительный вид стиля демонстрируется внизу окна.



Рис. 20.18

Второе из окон CSS называется Apply Styles. Несмотря на то что это окно очень похоже на окно Manage Styles, оно имеет несколько иное предназначение — упростить применение стилей к элементам, расположенным на веб-странице. Выберите команду View⇨Apply Styles, чтобы открыть окно, показанное на рис. 20.18. Как и окно Manage Styles, окно Apply Styles содержит все доступные стили, которые можно фильтровать, чтобы видеть только те из них, которые можно применить к текущей странице или выделенному элементу. В этом окне для обозначения стилей, основанных на атрибуте ID, классах, элементе, а также внедренных стилей используются те же пиктограммы. Кроме того, оставляя курсор на стиле, пользователь может увидеть все его свойства в виде CSS-правила.

Однако окно Apply Styles намного аккуратнее демонстрирует стили, чем окно Manage Styles. Оно содержит цвет шрифта и его плотность, цвета фона, изображений, границ, а также параметры выравнивания текста.



Предварительный вид стиля, показанный в нижней части окна Manage Styles, обычно не совпадает с его видом, реализованным веб-браузером. Это объясняется тем, что при предварительном просмотре совершенно не учитываются правила наследования CSS, которые могут приводить к замещению свойств стиля.

Когда пользователь выбирает HTML-элемент на панели Designer, стили, которые применяются к этому элементу, в окне Apply Styles выделяются голубым контуром. Это можно увидеть на рис. 20.18, где стиль `.phone` является активным для выделенного элемента. Если задержать курсор мыши над любым из стилей, на нем появляется кнопка, открывающая доступ к контекстному меню. Это меню содержит команды для применения стиля к выделенному элементу или, если стиль уже был применен, для его удаления. Для того чтобы применить стиль к текущему HTML-элементу, достаточно щелкнуть на стиле.

Третье новое окно CSS в системе Visual Studio 2010 называется CSS Properties (рис. 20.19). В нем отображается сетка свойств, содержащая все стили, используемые HTML-элементом, выделенным в редакторе HTML Designer. Кроме того, это окно содержит исчерпывающий список всех доступных свойств CSS. Это позволяет добавлять свойства в существующий стиль, модифицировать свойство, которое уже было задано, а также создавать новые внедренные стили.

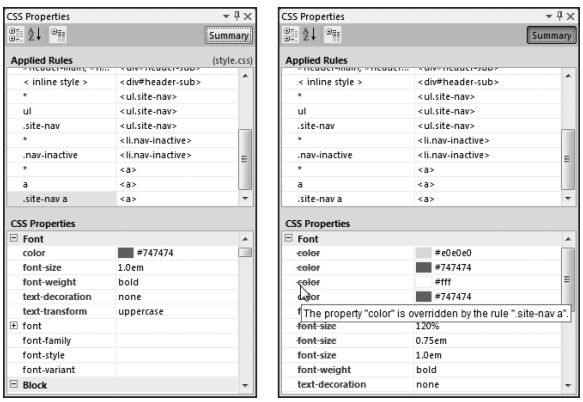


Рис. 20.19

Вместо демонстрации деталей индивидуального стиля, как это принято в окнах Apply Styles и Manage Styles, окно CSS Properties показывает сводную информацию о всех стилях, применимых к текущему элементу, учитывая порядок их приоритетов. В верхней части окна CSS Properties находится раздел Applied Rules, в котором перечислены стили CSS в порядке их применения. Стили, расположенные ниже в этом списке, замещают стили, расположенные сверху.

Когда пользователь выбирает стиль в разделе Applied Rules, в сетке свойств, расположенной ниже, демонстрируются все CSS-свойства этого стиля. На рис. 20.19, *слева*, выбрано CSS-правило `.site-nav a`, содержащее определения для CSS-свойств `color`, `font-size`, `font-weight`, `text-decoration` и `text-transform`. Пользователь может редактировать эти правила или определять новые непосредственно в этой сетке свойств.

В окне CSS Properties имеется кнопка Summary, после щелчка на которой отображаются все CSS-свойства, применимые к текущему элементу (см. рис. 20.19, *справа*). CSS-свойства, которые были замещены, зачеркнуты красной линией. Если задержать курсор мыши над свойством, на экране появляется подсказка, в которой указана причина замещения.

Система Visual Studio 2010 содержит также переключатель Target Rule в инструментальной панели Formatting, показанной на рис. 20.20. Этот переключатель позволяет указать, где следует сохранить изменения стиля, внесенные с помощью инструментальной панели форматирования и диалоговых окон. К ним относятся инструментальная панель Formatting и диалоговое окна, входящие в меню Format, т.е. Font, Paragraph, Bullets and Numbering, Borders and Shading и Position.

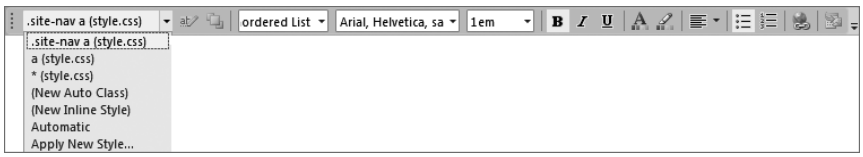


Рис. 20.20

Переключатель Target Rule имеет два режима: Automatic и Manual. В режиме Automatic система Visual Studio автоматически выбирает, где применить новый стиль. В режиме Manual пользователь может свободно выбирать, где создавать результирующие свойства CSS. По умолчанию система Visual Studio 2010 устанавливает режим Manual, и любые переключения режима запоминаются.

Переключатель Target Rule связан со списком стилей, которые уже применялись к текущему выделенному элементу. Внедренные стили выделяются записью `<inline style>`. К именам стилей, внедренных на текущую страницу, добавляется окончание `(Current Page)`, а к именам стилей, определенных на внешнем листе стилей, дописывается имя файла.

В заключение отметим, что в системе Visual Studio 2010 реализована поддержка технологии IntelliSense для редакторов CSS и HTML. Редактор CSS, который по умолчанию открывается, когда пользователь дважды щелкает на CSS-файле, реализует приглашения IntelliSense для ввода всех CSS-атрибутов и корректных значений (рис. 20.21). После определения стилей CSS редактор HTML последовательно определяет и выводит на экран список корректных имен CSS-классов, доступных для веб-страницы, когда пользователь добавляет атрибут `class` к элементу HTML.

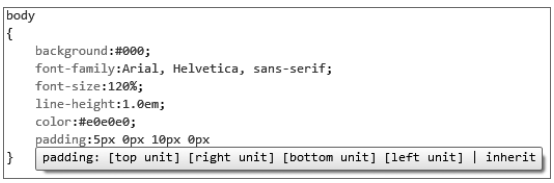


Рис. 20.21

Инструменты проверки корректности

Веб-браузеры отлично скрывают плохо сформированный HTML-код от конечных пользователей. Неправильная синтаксическая конструкция, способная вызвать фатальную ошибку в XML-документе, например нарушение порядка или пропущенный закрывающий дескриптор, часто остается незаметной в веб-браузере. Однако, если

открыть плохой HTML-код в другом браузере, веб-страница может выглядеть совершенно иначе. Это достаточная причина для того, чтобы требовать от HTML-кода соответствия стандарту.

На первом этапе проверки корректности следует установить целевую схему проверки. Для этого можно использовать инструментальную панель HTML Source Editing, показанную на рис. 20.22.

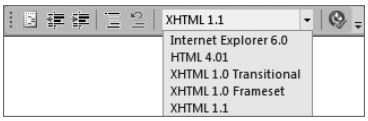


Рис. 20.22

HTML-разметка будет сравниваться с выбранной схемой. Процесс проверки напоминает фоновую проверку правописания и выполняется при вводе разметки. Элементы или атрибуты, не соответствующие текущей схеме, подчеркиваются зеленой волнистой линией. Как показано на рис. 20.23, когда курсор находится над помеченным элементом, появляется подсказка, объясняющая причину ошибки. Кроме того, в окне Error List появляется запись об ошибке.

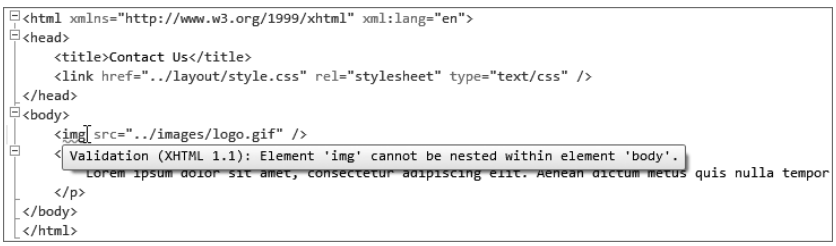


Рис. 20.23

Схема проверки гарантирует, что пользовательская веб-страница будет выглядеть одинаково в любом браузере. Однако она не гарантирует, что пользовательский сайт будет доступен для всех. Есть большая группа людей с ограниченными физическими возможностями, которым может оказаться трудно работать с сайтом из-за особенностей его HTML-разметки.

По оценкам Всемирной организации здравоохранения в мире насчитывается около 314 миллионов людей с ограниченным зрением (World Health Organization, 2009). Только в США более 21 млн. человек имеют очень плохое зрение (National Center for Health Statistics, 2006). Это очень большая группа людей, особенно если учесть, что в нее не входят люди с другими физическими ограничениями.

Кроме уменьшения потенциальной базы пользователей, веб-приложение, не учитывающее особенности людей с ограниченными физическими возможностями, порождает риск судебного преследования. Во многих странах приняты законы, требующие от разработчиков веб-сайтов и других форм коммуникации учитывать интересы людей с ограниченными физическими возможностями.

К счастью, система Visual Studio 2010 содержит инструмент для проверки доступности приложения, которое проверяет разметки HTML в рамках разных сценариев. Программа Web Content Accessibility Checker, которая запускается командой Tools⇨ Check Accessibility, позволяет проверять индивидуальные страницы на соответствие нескольким критериям доступа, включая принципы Web Content Accessibility Guidelines (WCAG) версии 1.0 и закон Americans with Disabilities Act Section 508 Guidelines, кратко называемый Section 508.

Пользователь должен выбрать принципы для проверки и щелкнуть на кнопке Validate. После проверки веб-страницы все сообщения, интерпретируемые как ошибки или предупреждения, выводятся в окне Error List, как показано на рис. 20.24.

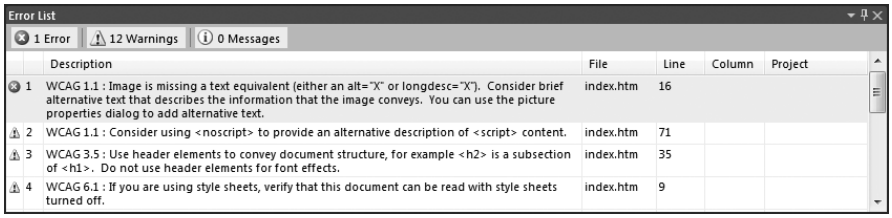


Рис. 20.24



В предыдущих версиях платформы ASP.NET визуальное представление элементов управления для веб-приложений обычно не соответствовало HTML-разметке или стандартам доступности. К счастью, большинство из этих недостатков было исправлено в версии ASP.NET 4.0.

Элементы управления для веб-приложений

С появлением технологии ASP.NET 1.0 разработчики, использующие программные продукты компании Microsoft, получили совершенно новое средство для создания веб-приложений. Вместо использования HTML-элементов, смешанных с серверным языком сценариев, как это принято в классических языках ASP, JSP и Perl, технология ASP.NET ввела концепцию полноценных элементов управления для веб-страниц, которые действуют примерно так же, как их аналоги в системе Windows.

Элементы управления для веб-приложений, например кнопка и поле ввода, имеют схожие свойства, такие как `Text`, `Left` и `Width`, а также распознаваемые методы и события, такие как `Click` и `TextChanged`. Кроме них, платформа ASP.NET 1.0 содержала некоторое количество компонентов, специфичных для веб-приложений. Одни из них были предназначены для работы с базами данных, например элемент управления DataGrid, а другие решали общие задачи, характерные для веб. Например, элемент управления ErrorProvider обеспечивал обратную связь с пользователями и сообщал о проблемах, возникших в связи с информацией, введенной пользователем в веб-форму.

Последующие версии платформы ASP.NET содержали более пятидесяти элементов управления для веб-приложений, включая компоненты навигации, аутентификации пользователей, веб-части и улучшенные элементы управления для обработки данных. Сторонние поставщики также выпустили множество серверных элементов управления и компонентов, обеспечивающих еще более сложные функциональные возможности.

К сожалению, объем данной книги не позволяет очень подробно описать все элементы управления, доступные для веб-приложений. По существу, многие из этих компонентов, например TextBox, Button и Checkbox, представляют собой прямые веб-аналоги давно и хорошо известных элементов управления, образующих пользовательский интерфейс. Однако может оказаться полезным дать обзор более специализированных и функциональных элементов управления, входящих в пакет разработчиков веб-приложений на платформе ASP.NET.

Навигационные компоненты

Платформа ASP.NET позволяет достаточно просто добавлять в пользовательские веб-приложения элементы навигации по сайту, включая провайдер карты сайта и ассоциированные с ним элементы управления. Для того чтобы реализовать возможность создавать карту сайта в своем проекте, необходимо вручную сформировать данные о сайте, по умолчанию хранящиеся в файле `Web.sitemap`, и обновлять их при добавлении и удалении страниц сайта. Файлы карты сайта можно использовать как источник данных для многих элементов управления, включая элемент SiteMapPath, который автоматически отслеживает место пользователя в иерархии сайта, а также элементы Menu и TreeView, которые могут быть использованы для специального представления подмножества карты сайта.

После того как иерархия сайта определена в файле `Web.sitemap`, проще всего использовать ее путем перетаскивания элемента управления SiteMapPath на рабочую область проектирования веб-страницы (рис. 20.25). Этот элемент управления автоматически связывается с провайдером карты сайта, заданным по умолчанию в соответствии с информацией, записанной в файле `Web.config`, и генерирует узлы для вывода на экран.

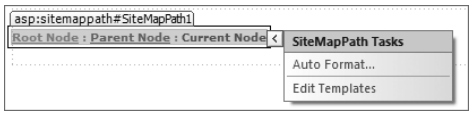


Рис. 20.25

Несмотря на то что элемент управления SiteMapPath выводит на экран только навигационные цепочки, прямо ведущие к текущей странице, просматриваемой пользователем, иногда пользователь хочет увидеть список всех страниц, существующих на сайте. Для этого можно использовать элемент управления Menu, существующий в платформе ASP.NET. При этом информацию можно просматривать как в вертикальном, так и в горизонтальном направлении. Аналогично, с картой сайта можно связать элемент управления TreeView и использовать его для визуализации иерархического меню страниц, существующих на веб-сайте. На рис. 20.26 показана веб-страница с элементами SiteMapPath, Menu и TreeView, каждый из которых отформатирован в соответствии с одним из внедренных стилей.

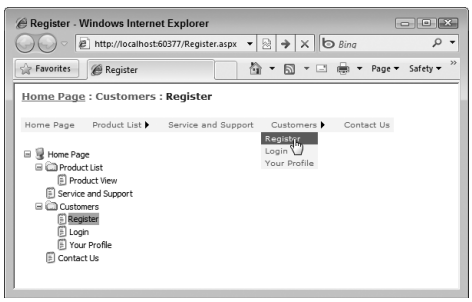


Рис. 20.26

Аутентификация пользователей

Возможно, наиболее значительным дополнением в наборе компонентов платформы ASP.NET 2.0 были новые компоненты для аутентификации и регистрации пользователей. С помощью этих компонентов можно быстро и легко создать части своего веб-приложения, ориентированные на пользователя, не заботясь о том, как их форматировать или какие элементы управления использовать.

Каждое веб-приложение имеет источник данных, заданный по умолчанию и добавляемый в конфигурацию ASP.NET, когда приложение создается в первый раз. Таким источником является база данных SQL Server Express, имя которой по умолчанию

ссылается на определенное место в локальной файловой системе. Этот источник данных используется как место для аутентификации пользователя, в котором хранится информация о пользователях и их текущих настройках.

Преимущество, обеспечиваемое этим автоматическим хранилищем данных, генерируемым для каждого веб-приложения, состоит в том, что система Visual Studio может использовать массив веб-компонентов для работы с пользователем, в котором информация о пользователях записывается без изменения исходного кода.

Прежде чем зарегистрироваться как пользователь на конкретном сайте, необходимо создать учетную запись. В принципе это можно сделать с помощью средств администрирования и конфигурирования платформы ASP.NET, которые будут рассмотрены в этой главе далее, но можно разрешить пользователям сайта самостоятельно создавать свои учетные записи. Именно это делает компонент CreateUserWizard. Он состоит из двух страниц мастера с информацией о создании учетной записи и выводит сообщение, если создание учетной записи прошло успешно.



Рис. 20.27

Как только пользователь создаст свою учетную запись, он должен иметь возможность зарегистрироваться на сайте. Для этого необходим элемент управления Login. Когда разработчик добавляет компонент Login на свою страницу, создается небольшая форма, содержащая поля User Name и Password, а также флажок, позволяющий запомнить регистрационные данные, и кнопка Log In (рис. 20.27).

Для того чтобы все это заработало правильно, необходимо отредактировать файл `Web.config` и изменить тип аутентификации на `Forms`. По умолчанию в системе используется тип аутентификации `Windows`, и без его изменения веб-сайт будет распознавать всех как Windows-пользователей.

Очевидно, что некоторые веб-приложения требуют аутентификации типа Windows, но для простого веб-сайта, который мы планируем развернуть в Интернете, достаточно внести только маленькое изменение, чтобы элемент управления Login стал работать корректно.

Можно также использовать несколько элементов управления, которые будут распознавать, зарегистрирован ли пользователь, и выводить зарегистрированным и анонимным пользователям разную информацию. Элемент управления LoginStatus представляет собой простой компонент, имеющий два состояния. Они выводят на экран один набор данных, если пользователь зарегистрирован, и другой, если нет. Компонент LoginName также прост: он лишь выводит имя зарегистрированного пользователя.

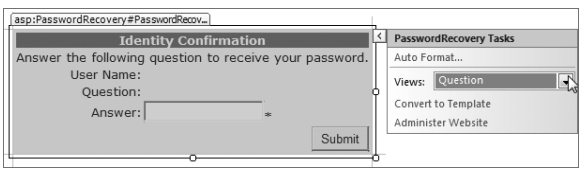


Рис. 20.28

Кроме того, существуют элементы управления, позволяющие конечным пользователям управлять своими паролями. Компонент ChangePassword работает в сочетании с другими автоматическими компонентами для работы с конечными пользователями и позволяет им изменять свои пароли. Иногда пользователи забывают свои пароли. В этих ситуациях на помощь приходит элемент управления PasswordRecovery (рис. 20.28), который имеет три состояния: UserName, Question и Success.

Идея состоит в том, что пользователи сначала вводят свои имена, а приложение находит и выводит на экран секретный вопрос, а затем ожидает ответа. Если ответ

правильный, то компонент переходит на страницу Success и посылает электронное сообщение на зарегистрированный адрес электронной почты.

Последний компонент в группе Login на панели Toolbox называется LoginView. Он позволяет создавать целые разделы веб-страниц, которые становятся видимыми только при определенных условиях, связанных с личностью регистрируемого пользователя. По умолчанию этот компонент имеет два состояния: AnonymousTemplate, используемое, когда ни один пользователь не зарегистрирован, и LoggedInTemplate, используемое, когда все пользователи зарегистрированы. Оба шаблона имеют область редактирования, которая в начальный момент совершенно пуста.

Но поскольку разработчик может определять специальные роли и присваивать их пользователям, он может также создавать шаблоны для каждой роли, определенной для веб-сайта (рис. 20.29). Команда Edit RoleGroups в списке задач интеллектуального дескриптора, ассоциированного с компонентом LoginView, открывает обычный редактор коллекций и позволяет разработчику создавать группы ролей, содержащих одну или несколько ролей.

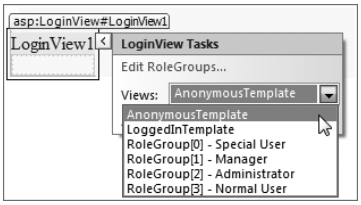


Рис. 20.29

Когда сайт распознает пользователя, которому присвоена определенная роль, область компонента LoginView заполняется содержимым конкретного шаблона.



Как создавать роли и управлять ими, описывается ниже, в разделе “Средства ASP.NET Web Site Administration”.

Любопытно, что, внеся вручную лишь несколько исправлений в свойствах всех этих элементов управления и сделав несколько дополнительных записей в файле `Web.config`, можно создать полную систему аутентификации пользователей в своем веб-приложении. По существу, как будет показано позднее в разделе “Средства ASP.NET Web Site Administration”, все эти настройки можно изменить, даже не редактируя файл `Web.config` непосредственно. Это действительно эффективное кодирование!

Компоненты для работы с данными

Компоненты для работы с данными были изобретены для разработчиков веб-приложений, работающих с программами компании Microsoft, и появились еще в первой версии системы Visual Studio .NET. Затем они эволюционировали и с каждым новым выпуском системы становились мощнее. Каждый элемент управления имеет интеллектуальный дескриптор Tasks list, связанный с ним. Это позволяет редактировать отдельные шаблоны для каждой части видимой области. Например, элемент DataList имеет семь шаблонов, каждый из которых может быть настроен по-своему (рис. 20.30).

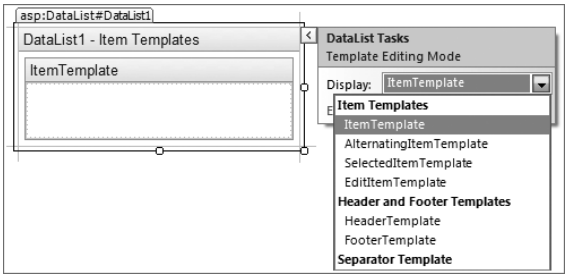


Рис. 20.30

Элементы управления источниками данных

Архитектура элементов управления источниками данных на платформе ASP.NET обеспечивает действительно простой способ связывания с данными посредством пользовательского интерфейса. Элементы управления источниками данных появились в версии ASP.NET 2.0. Среди них были компоненты SqlDataSource и AccessDataSource, предназначенные для связывания с базами данных SQL Server или Access, ObjectDataSource — для связывания с обобщенным классом, XmlDataSource — для связывания с XML-файлами и SiteMapDataSource — для реализации навигационного дерева сайта.

В версии ASP.NET 3.5 появился элемент управления LinqDataSource, позволяющий непосредственно связывать элементы управления пользовательского интерфейса с источниками данных, используя язык Language Integrated Query (LINQ). Элемент управления EntityDataSource, появившийся в версии ASP.NET 3.5 SP1, поддерживает связывание данных с помощью технологии ADO.NET Entity Framework. Эти элементы управления позволяют автоматически генерировать бо́льшую часть кода, необходимого для взаимодействия с данными.

Все элементы управления источниками данных работают примерно одинаково. Для примера возьмем элемент управления LinqDataSource.

Перед тем как использовать элемент управления LinqDataSource, необходимо создать класс `DataContext`. Контекст данных служит оболочкой для связи с базой данных, чтобы обеспечивать сервис жизненного цикла объектов. Как создать новый класс `DataContext` в своем приложении, описано в главе 28.

Затем можно создать экземпляр элемента управления LinqDataSource, перетащив его из панели Toolbox в рабочую область проектирования. Для настройки элемента управления необходимо запустить мастер Configure Data Source Wizard, пиктограмма которого расположена под интеллектуальным дескриптором. Затем следует выбрать класс контекста данных и детали выбора данных, которые будут использоваться. На рис. 20.31 показан экран мастера Configure Data Source Wizard, на котором можно выбрать таблицы и столбцы для генерации запроса LINQ к базе SQL. Осталось только связать этот источник данных с элементом управления из пользовательского интерфейса, например с элементом управления ListView, чтобы обеспечить доступ данных "только для чтения".

Разработчик может легко воспользоваться преимуществами более мощных функциональных возможностей, связанных с доступом к данным, поддерживаемых технологией LINQ, например разрешить вставки, обновления и удаления, задав свойства `EnableInsert`, `EnableUpdate` и `EnableDelete` элемента LinqDataSource равными `true`. Это можно сделать, редактируя код, или с помощью сетки свойств.

Более подробная информация о технологии LINQ приведена в главе 28.

Элементы управления для просмотра данных

Указав источник данных, можно легко просматривать их содержимое с помощью одного из элементов управления. Платформа ASP.NET содержит восемь встроенных элементов управления для веб-приложений: Chart, DataList, DetailsView, FormView, GridView, ListView и Repeater. Элемент управления Chart используется для графической визуализации данных, например для построения диаграмм или графиков. Он обсуждается в главе 30.

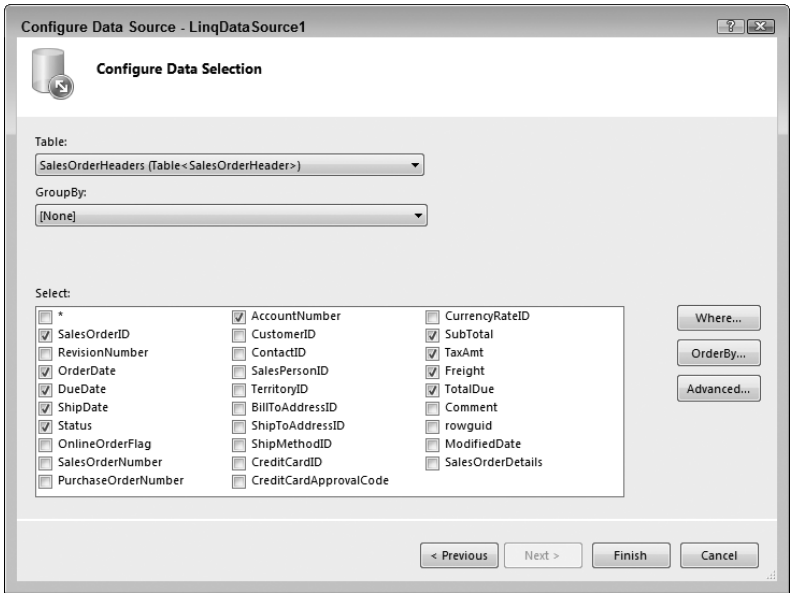


Рис. 20.31

Недостатком серверных элементов управления на платформе ASP.NET является то, что разработчики очень слабо контролируют HTML-разметку, которую они генерируют. Это особенно справедливо по отношению ко многим элементам управления, таким как GridView, который всегда использует таблицу HTML для форматирования выводимых данных, несмотря на то, что в некоторых ситуациях более удобным было бы выводить данные по порядку.

Элемент управления ListView представляет собой хорошее решение, компенсирующее отсутствие других аналогичных элементов управления данными. Вместо окружения визуализированной разметки излишними элементами `<table>` или `<span>` он позволяет разработчику указать в коде HTML, что именно будет выведено на экран. Разметка HTML определена в одиннадцати шаблонах, поддерживаемых элементом управления ListView.

- AlternatingItemTemplate
- EditItemTemplate
- EmptyDataTemplate
- EmptyItemTemplate
- GroupSeparatorTemplate
- GroupTemplate
- InsertItemTemplate
- ItemSeparatorTemplate
- ItemTemplate
- LayoutTemplate
- SelectedItemTemplate

Наиболее полезными являются шаблоны LayoutTemplate и ItemTemplate. Шаблон LayoutTemplate определяет разметку HTML, которая окружает вывод, а шаблон ItemTemplate задает разметку HTML, которая форматирует каждую запись, связанную с элементом ListView.

Добавляя элемент управления ListView на рабочую область, разработчик может связать его с источником данных, а затем открыть диалоговое окно Configure ListView, показанное на рис. 20.32, с помощью команд интеллектуального дескриптора. Таким образом, возникает механизм генерации кода, который автоматически создает HTML-код, основанный на небольшом количестве заранее определенных компоновок и стилей.

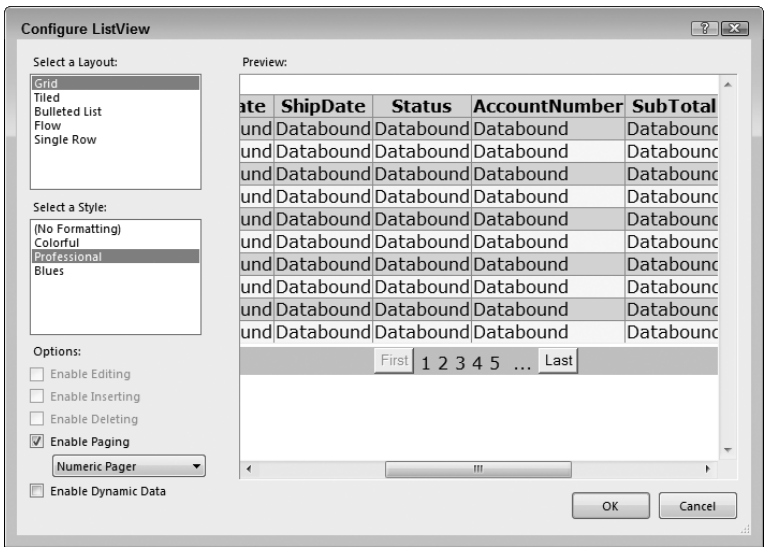


Рис. 20.32



Поскольку разработчик имеет полный контроль над HTML-разметкой, диалоговое окно Configure ListView даже не делает попытки сделать анализ любой из существующих разметок. Вместо этого, когда разработчик открывает окно снова, оно просто показывает настройки, заданные по умолчанию.

Вспомогательные элементы управления для работы с данными

Элемент управления DataPager используется, чтобы разделить данные, отображаемые компонентом пользовательского интерфейса, на несколько страниц. Это необходимо при работе с большими наборами данных. Данный элемент управления поддерживает разделение на страницы либо с помощью объекта NumericPagerField, позволяющего пользователям выбирать номер страницы, либо объекта NextPreviousPagerField, позволяющего перемещаться на предыдущую или следующую страницу. Как и в случае элемента управления ListView, разработчик может также написать свою собственную HTML-разметку для разделения данных на страницы, используя объект TemplatePagerField.

В заключение отметим, что элемент управления QueryExtender, впервые появившийся в версии ASP.NET 4.0, обеспечивает способ фильтрации данных, поступающих из элемента EntityDataSource или LinqDataSource декларативным способом. Это особенно полезно для сценариев поиска.

Веб-части

Другой превосходной особенностью технологии ASP.NET является возможность создавать элементы управления и страницы Web Parts. Это позволяет разделять определенные страницы на сайте на фрагменты, которые разработчик или пользователь может перемещать, показывать и скрывать, создавая свое собственное визуальное представление. Элементы Web Parts для платформы ASP.NET слабо связаны с пользовательскими элементами управления веб-приложениями, но своим включением в технологию ASP.NET они обязаны своей огромной популярности в технологии SharePoint Portals.

Как и в случае страницы Web Parts, сначала разработчик должен создать компонент WebPartManager, который находится на странице, чтобы следить за любой ее частью, определенной как веб-часть. Затем необходимо использовать контейнеры WebPartZone, чтобы задать место для размещения настраиваемого содержимого страницы, и поместить реальное содержимое в контейнер WebPartZone.

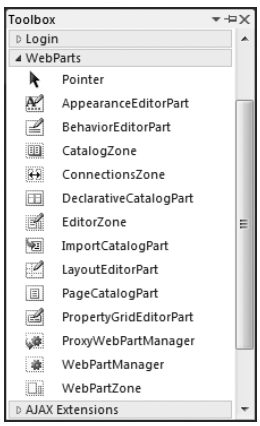


Рис. 20.33

Хотя эти два контейнера являются ядром компонентов Web Parts, на панели Toolbox достаточно найти группу WebParts, чтобы обнаружить весь массив дополнительных компонентов (рис. 20.33). Эти дополнительные компоненты используются для того, чтобы дать пользователям возможность настраивать внешний вид своего веб-сайта.

К сожалению, в книге недостаточно места для того, чтобы подробнее описать все элементы управления веб-приложениями на платформе ASP.NET. Если читатели хотят узнать о них побольше, рекомендуем массивную книгу Bill Evjen, Scott Hanselman, and Devin Rader *Professional ASP.NET 4 in C# and VB.*

Страницы мастера

Очень полезной особенностью разработки веб-приложений в среде Visual Studio является возможность создавать *страницы мастера* (master pages), определяющие настраиваемые разделы. Это позволяет определять отдельные страницы, содержащие общие элементы, которые должны совместно использоваться всеми компонентами сайта, определять области, в которых хранится индивидуализированное содержимое, а также наследовать его для каждой страницы сайта.

Для того чтобы добавить страницу мастера в свой проект Web Application, следует выбрать команду Add New Item из меню веб-сайта или контекстного меню в окне Solution Explorer. В открывшемся диалоговом окне Add New Item, показанном на рис. 20.34, содержится множество шаблонов для элементов, которые можно добавить в веб-приложение. Обратите внимание на то, что, кроме страниц Web Forms (`.aspx`) и элементов Web User Controls, в свой проект можно добавить обычные HTML-файлы, листы стилей и другие файлы, связанные с веб. Для того чтобы добавить страницу мастера, необходимо выбрать шаблон Master Page, указать имя файла и щелкнуть на кнопке Add.

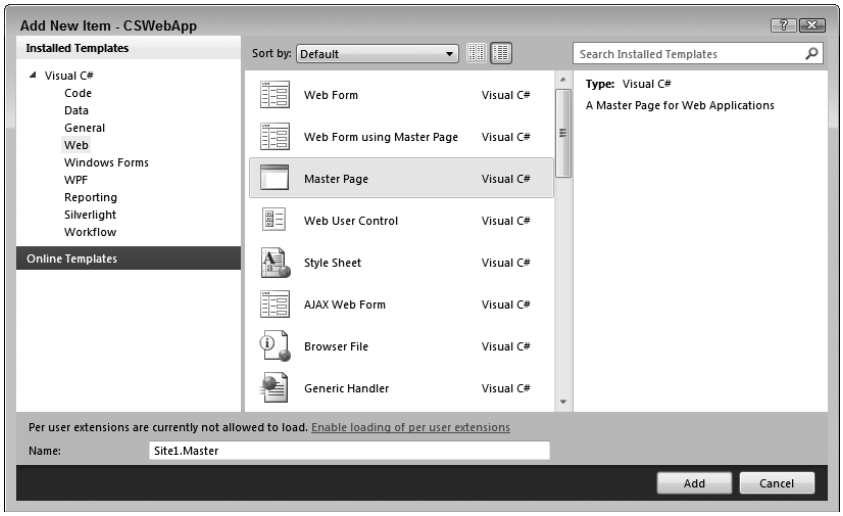


Рис. 20.34

Страница мастера, добавленная на пользовательский веб-сайт, запускается как минимальная веб-страница с двумя пустыми компонентами ContentPlaceHolder: один — для тела веб-страницы, другой — для ее заголовка. Именно здесь можно разместить подробную информацию о каждой отдельной странице. Страницу мастера можно создавать так, будто она представляет собой обычную форму для веб-страницы, наполняя ее элементами ASP.NET и HTML, таблицами CSS и темами.

Если требуется дополнительная область для размещения информации, на страницу можно перетащить новый элемент управления ContentPlaceHolder из инструментальной панели Toolbox или переключить представление в режим Source и добавить следующие дескрипторы в те места, где необходима дополнительная область.

```
<asp:ContentPlaceHolder id="aUniqueid" runat="server">
</asp:ContentPlaceHolder>
```

После завершения проектирования страниц мастера их можно использовать как страницы деталей для создания новых форм в рамках проекта.

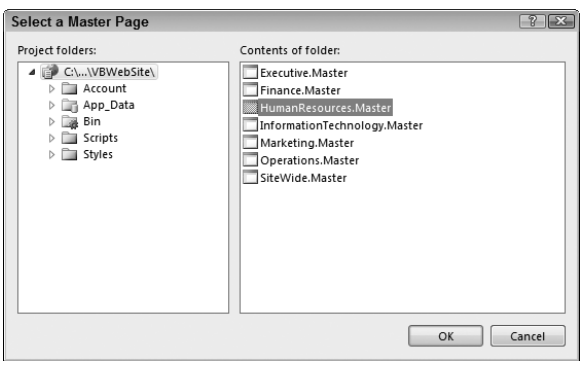


Рис. 20.35

К сожалению, процесс добавления формы, использующей страницу мастера, зависит от того, какой проект используется для создания веб-приложения — Web Application или Web Site. Для проекта Web Application вместо команды Web Form следует выбирать команду Master Page, которая открывает диалоговое окно Select a Master Page (рис. 20.35). Для проекта Web Site окно Add New Item содержит флажок Select Master Page. Если его установить, то откроется диалоговое окно Select a Master Page.

Пользователь должен выбрать страницу мастера, которая будет применена к странице деталей, и щелкнуть на кнопке OK. Новая веб-форма, добавляемая в проект, будет содержать один или несколько элементов управления Content, которым соответствуют элементы управления ContentPlaceHolder на странице мастера.

Несложно заметить преимущества страниц мастера и понять, почему они стали очень популярными. Однако еще более полезно создавать вложенные страницы мастера.

Работа с вложенными страницами мастера не очень отличается от работы с обычными страницами мастера. Для того чтобы добавить вложенную страницу мастера, необходимо выбрать команду Nested Master Page в окне Add New Item. После этого пользователь получит приглашение выбрать родительскую мастер-страницу в окне Select a Master Page, как показано на рис. 20.35. Когда впоследствии пользователь станет добавлять новую веб-страницу, все вложенные страницы мастера будут показаны в окне Select a Master Page.

Создание "толстого" клиента

В последние годы индустрия программного обеспечения существенно сместила акценты на взаимодействие конечных пользователей в разработке приложений. Особенно ярко это проявилось в разработке веб-приложений. Появление технологии AJAX и все возрастающее признание языка JavaScript позволяют надеяться на то, что в ближайшем будущем появятся веб-приложения, функциональные возможности которых будут близки к возможностям их настольных эквивалентов.

Компания Microsoft четко осознает это и поэтому включила в систему Visual Studio 2010 много новых инструментов и функциональных возможностей, поддерживающих создание "толстых" клиентов. В системе существует режим интегрированной отладки, а также поддержка языка JavaScript с помощью технологии IntelliSense. Вместе с системой Visual Studio 2010 поставляется библиотека ASP.NET AJAX. Кроме того, в интегрированной среде разработки существует поддержка пакета AJAX Control Extenders. Эти инструменты позволяют значительно облегчить разработку, создание и отладку клиентского кода, обеспечивающего сложные взаимодействия с пользователями.

Разработка приложений с помощью языка JavaScript

Разработка клиентского кода на языке JavaScript долгое время считалась трудной работой, несмотря на то, что сам по себе язык довольно простой. Поскольку язык JavaScript является динамическим и слабо типизированным языком программирования — и в силу этого сильно отличается от строго типизированных языков Visual Basic и C#, — его репутация среди некоторых разработчиков программ, использующих платформу .NET, невысока.

Таким образом, поддержка технологии IntelliSense для языка JavaScript является одной из наиболее полезных функциональных возможностей системы Visual Studio. Проявление технологии IntelliSense можно заметить сразу, как только пользователь начнет набирать текст: на экране появятся приглашения ввести функции и ключевые слова языка JavaScript, такие как `var`, `alert` и `eval`.

Более того, подсистема JavaScript IntelliSense в системе Visual Studio 2010 автоматически оценивает и выводит типы переменных, чтобы приглашения были как можно более точными. Например, на рис. 20.36 можно увидеть, что подсистема IntelliSense рас-

познала, что слово `optSelected` является HTML-объектом, поскольку вызов функции `document.getElementByID` возвращает именно этот тип.



Рис. 20.36

Помимо веб-форм, система Visual Studio обеспечивает поддержку технологии IntelliSense во внешних файлах JavaScript. Кроме того, она обеспечивает помощь технологии IntelliSense в сценарных и библиотечных файлах, например в библиотеке Microsoft AJAX.

Компания Microsoft расширила систему комментирования с помощью языка XML в системе Visual Studio, и теперь она распознает комментарии к функциям языка JavaScript. Подсистема распознает комментарии на языке XML и выводит резюме, параметры и информацию о типе возвращаемого значения для данной функции.

Несколько ограничений препятствуют выводу информации системой JavaScript IntelliSense в некоторых ситуациях.

- Синтаксическая или другая ошибка во внешнем сценарном файле.
- Вызов функции, специфичной для браузера, или объекта. Большинство веб-браузеров имеют группу объектов, характерных только для данного браузера. Эти объекты можно использовать по-прежнему, и многие популярные реализации языка JavaScript так и делают; однако технология IntelliSense эти объекты не поддерживает.
- Ссылка на файлы, не входящие в текущий проект.

Система Visual Studio постоянно отслеживает файлы в проекте и обновляет систему IntelliSense, как только в проект включается новый файл. Если по какой-то причине система Visual Studio не выводит самую свежую информацию, пользователь может обновить систему IntelliSense, выбрав команду Edit⇨IntelliSense⇨Update JScript IntelliSense.

Одной из новых функциональных возможностей последней версии технологии ASP.NET является свойство ClientIDMode, которое можно добавлять к элементам управления веб-серверов. В предыдущих версиях было трудно ссылаться из программы на языке JavaScript на эти элементы управления из-за значения, которое генерировалось для атрибута id генерируемых элементов управления HTML. Свойство ClientIDMode уст-

раняет этот недостаток, определяя два новых режима (`Static` и `Predictable`), чтобы генерация этих идентификаторов стала более простой и предсказуемой.

Усовершенствованная поддержка технологии JavaScript IntelliSense в сочетании с улучшенной отладкой клиентского кода и лучшим контролем клиентских идентификаторов значительно облегчает разработку кода на языке JavaScript в системе Visual Studio 2010.

Работа с платформой ASP.NET AJAX

Платформа ASP.NET AJAX реализует хорошо известный подход к программированию серверных элементов управления для реализации полноценного взаимодействия с клиентом на основе технологии AJAX.

Платформа ASP.NET AJAX состоит из серверных и клиентских компонентов. Серверные компоненты, включая популярные элементы управления UpdatePanel и UpdateProgess, позволяют обновлять часть страницы без внесения изменений в существующий код этой страницы. Клиентская часть библиотеки Microsoft AJAX представляет собой каркас JavaScript, который можно использовать в любом веб-приложении, например PHP на сервере Apache, а не только ASP.NET или IIS.

Следующий пример показывает, как улучшить существующую веб-страницу, добавив на нее элемент управления UpdatePanel из платформы ASP.NET AJAX для обновления частей страницы. В этом сценарии используется очень простая веб-форма с серверным элементом управления DropDownListl, имеющим компонент AutoPostBack для подключения сервера. Эта веб-форма обрабатывает событие `DropDownList.SelectedIndexChanged` и сохраняет значение, выбранное в элементе DropDownList, в серверном элементе управления TextBox на этой странице. Код этой страницы приведен ниже.

AjaxSampleForm.aspx

```
<%@ Page Language="vb" AutoEventWireup="false"
   CodeBehind="AjaxSampleForm.aspx.vb"
   Inherits="ASPNetWebApp.AjaxSampleForm" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ASP.NET AJAX Sample</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
        Select an option:
        <asp:DropDownList ID="DropDownList1" runat="server"
AutoPostBack="True">
            <asp:ListItem Text="Option 1" Value="Option 1" />
            <asp:ListItem Text="Option 2" Value="Option 2" />
            <asp:ListItem Text="Option 3" Value="Option 3" />
            </asp:DropDownList>
            <br />
            Option selected:
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        </div>
        </form>
</body>
</html>
```

AjaxSampleForm.aspx.vb

```
Public Partial Class AjaxSampleForm
    Inherits System.Web.UI.Page
    Protected Sub DropDownList1_SelectedIndexChanged(ByVal sender As Object, _
                                                     ByVal e As EventArgs) _
                                        Handles DropDownList1.SelectedIndexChanged
        System.Threading.Thread.Sleep(2000)
        Me.TextBox1.Text = Me.DropDownList1.SelectedValue
    End Sub
End Class
```

Обратите внимание на то, что в метод DropDownList1_SelectedIndexChanged добавлена двухсекундная задержка. Это позволит заметить время обработки события на сервере, чтобы явственнее выделить внесенные изменения. Когда пользователь запустит эту страницу и изменит выбор пункта в раскрывающемся списке, в браузере будет обновлена вся страница.

Первый элемент управления из библиотеки AJAX, который необходимо добавить на свою веб-страницу, называется ScriptManager. Это невизуальный элемент управления, играющий главную роль в технологии ASP.NET AJAX и выполняющий такие задачи, как пересылка клиенту библиотек и файлов, а также генерация любых проксиклассов, запрашиваемых клиентом. Даже если на каждой веб-странице, созданной по технологии ASP.NET, помещено по одному элементу управления ScriptManager, может возникнуть проблема, если разработчик использует страницы мастера и пользовательские элементы управления. В такое случае необходимо добавить элемент управления ScriptManager на страницу, находящуюся на вершине иерархии, и элементы управления ScriptManagerProxy на каждую дочернюю страницу.

После добавления элемента управления ScriptManager разработчик может добавлять любые другие элементы управления ASP.NET AJAX. В данном случае мы добавили на веб-страницу элемент управления UpdatePanel, как показано в следующем коде. Обратите внимание на то, что элемент TextBox1 теперь содержится в новом компоненте UpdatePanel.

```
<%@ Page Language="vb" AutoEventWireup="false"
   CodeBehind="AjaxSampleForm.aspx.vb"
   Inherits="ASPNetWebApp.AjaxSampleForm" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>ASP.NET AJAX Sample</title>
</head>
<body>
   <form id="form1" runat="server">
   <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
   <div>
       Select an option:
       <asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True">
           <asp:ListItem Text="Option 1" Value="Option 1" />
           <asp:ListItem Text="Option 2" Value="Option 2" />
           <asp:ListItem Text="Option 3" Value="Option 3" />
           </asp:DropDownList>
           <br />
           Option selected:
           <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                     <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
                </ContentTemplate>
```

```
                <Triggers>
                    <asp:AsyncPostBackTrigger ControlID="DropDownList1"
                                              EventName="SelectedIndexChanged" />
                </Triggers>
            </asp:UpdatePanel>
        </div>
    </form>
</body>
</html>
```

Данная веб-страница использует технологию AJAX, обеспечивающую ее частичное обновление. Когда пользователь запустит эту страницу и выберет другой пункт в раскрывающемся списке, страница не будет обновлена целиком. Вместо этого будет изменен только текст, содержащийся в текстовом поле. Впрочем, запустив страницу, можно заметить, что технология AJAX является слишком мощной, чтобы ограничиться только обновлением частей страницы. Пока у нашей веб-страницы нет обратной связи, поэтому пользователю может показаться, что ничего не произошло. Для того чтобы устранить этот недостаток, полезно добавить на страницу элемент управления UpdateProgress. В этом случае, пока будет выполняться запрос к системе AJAX, HTML-элемент в разделе ProgressTemplate этого элемента управления будет перерисовываться. Пример элемента управления UpdateProgress продемонстрирован в следующем коде.

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server">
    <ProgressTemplate>
        Loading.
    </ProgressTemplate>
</asp:UpdateProgress>
```

Последний элемент управления ASP.NET AJAX, еще не упомянутый нами, — Timer. Он позволяет выполнить синхронный или асинхронный обратный вызов формы (postback) в определенном интервале времени. Это может быть полезным в некоторых сценариях, например, при проверке, видит ли сервер, что значение было изменено.

Добавив основные функциональные возможности технологии AJAX в свое веб-приложение, разработчик может еще больше усовершенствовать взаимодействие с клиентом, включив один или несколько элементов из пакета AJAX Control Toolkit, который будет рассмотрен в следующем разделе.

Использование элементов AJAX Control Extenders

Элементы AJAX Control Extenders позволяют наделить стандартные серверные элементы управления ASP.NET функциональными возможностями технологии AJAX. Наиболее широко известным набором средств расширения элементов управления является библиотека клиентских функций AJAX Control Toolkit с открытым исходным кодом, содержащая почти сорок средств расширения элементов. Они либо наделяют существующие веб-элементы управления ASP.NET новыми функциональными возможностями, либо обеспечивают совершенно новые элементы пользовательского интерфейса “толстого” клиента. На рис. 20.37 показан элемент Calendar Extender, связанный с элементом управления TextBox.

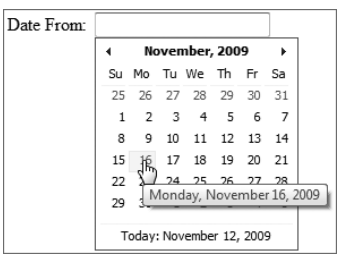


Рис. 20.37

Библиотеку ASP.NET AJAX Control Toolkit можно загрузить с сайта `http://ajaxcontroltoolkit.codeplex.com`. Бинарная версия загрузки включает в себя сборку `AjaxControlToolkit.dll`. Ее следует скопировать в каталог, из которого ее невозможно удалить случайно.

Для того чтобы добавить элементы управления в инструментальное окно Visual Studio Control Toolbox, сначала необходимо создать новую закладку, в которой они будут содержаться. Щелкните правой кнопкой мыши в окне Toolbox, выберите команду Add Tab, а затем присвойте новой вкладке осмысленное имя, например `AJAX Control Toolkit`. Затем щелкните правой кнопкой на новой закладке и выберите команду Choose Items. Щелкните на кнопке Browse и найдите библиотеку `AjaxControlToolkit.dll`, чтобы добавить элементы управления AJAX в список доступных компонентов .NET Framework Components. Щелкните на кнопке OK, и закладка будет заполнена всеми элементами управления из пакета AJAX Control Toolkit.

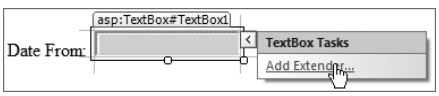


Рис. 20.38

Система Visual Studio 2010 предоставляет поддержку проектирования для всех элементов AJAX Control Extenders, включая библиотеку AJAX Control Toolkit. После того как разработчик добавит элементы управления в инструментальное окно Toolbox, система Visual Studio добавит в список интеллектуального дескриптора Tasks list все веб-элементы управления вместе с их расширениями, как показано на рис. 20.38.

Когда разработчик выбирает задачу Add Extender, он запускает мастер Extender Wizard (рис. 20.39). Теперь необходимо выбрать расширение в списке и щелкнуть на кнопке OK, чтобы добавить его в веб-форму. В большинстве случаев мастер Extender Wizard будет автоматически добавлять ссылку на библиотеку AJAX Control Toolkit. Однако, если он не может это сделать, разработчик может вручную добавить бинарную ссылку на сборку `AjaxControlToolkit.dll`.



Поскольку элементы Extender Controls созданы поверх платформы ASP.NET AJAX, разработчик должен убедиться, что в его веб-форме есть элемент управления ScriptManager.

Как показано на рис. 20.40, система Visual Studio 2010 включает все свойства расширения элемента управления в сетку свойств под элементом управления, с которым он связан.

Поскольку библиотека AJAX Control Toolkit содержит открытый код, разработчик может настраивать или еще больше совершенствовать любое расширение элемента управления, который она содержит. Система Visual Studio 2010 поставляется вместе с шаблонами проектов на языках C# и Visual Basic, позволяющих создавать собственные средства расширения AJAX Control Extenders и элементы управления ASP.NET AJAX Controls. Это облегчает написание полноценных веб-приложений с функциональными возможностями пользовательского интерфейса, которые можно повторно использовать на веб-страницах и в проектах.

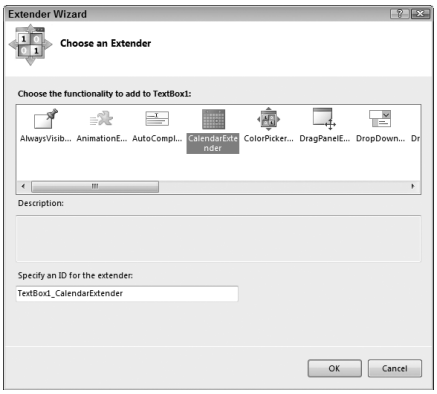


Рис. 20.39

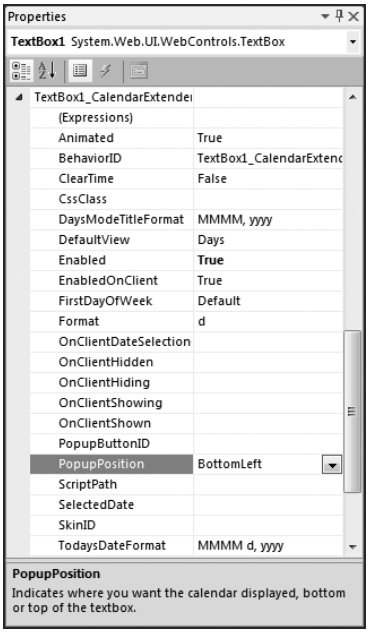


Рис. 20.40

Средства ASP.NET Web Site Administration

Несмотря на то что в большинстве ситуаций веб-приложения пользователей работают как положено, иногда возникает необходимость управлять настройками приложения, а не просто задавать параметры свойств компонентов и элементов страницы. Пакет Web Site Administration Tool представляет собой веб-приложение, позволяющее пользователям определять разные настройки, связанные с безопасностью, например имена пользователей и их роли, а также изменять существующие настройки, такие как страница ошибок, заданная по умолчанию, и глобальные настройки электронной почты SMTP, используемые разными компонентами, такими как PasswordRecovery.

Для того чтобы запустить приложение Administration Tool, необходимо выбрать команду Project⇨ASP.NET Configuration для проектов Web Application или команду Website⇨ASP.NET Configuration для проектов Web Site. После запуска этого приложения система Visual Studio 2010 создает экземпляр временного веб-сервера на уникальном порту и открывает веб-браузер на домашней странице Administration Tool для приложения, которое подлежит администрированию.

Пользователь может определить, является ли веб-сервер активным, просмотрев область сообщений своей инструментальной панели и найдя пиктограмму сервера, соединенного с портом, который система Visual Studio 2010 выделила при запуске. Пользователь может остановить активный веб-сервер, щелкнув правой кнопкой мыши на этой пиктограмме в области сообщений и выбрав команду Show Details. Когда на экране появится информация о сервере (рис. 20.41), пользователь должен щелкнуть на кнопке Stop, чтобы остановить конкретный экземпляр веб-сервера разработки.

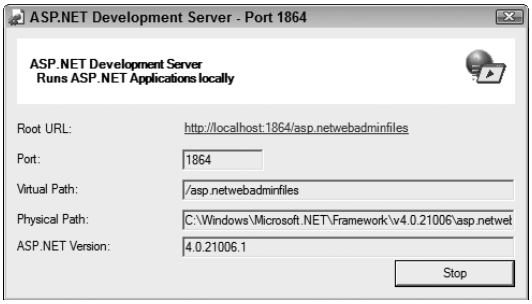


Рис. 20.41



Остановка активного веб-сервера не влияет на другие серверы разработки, которые запущены в текущий момент.

Когда приложение Administration Tool отображается в веб-браузере пользователя, на экран выводится имя приложения, которое сопровождается именем текущего аутентифицированного Windows-пользователя. Приложение имеет три основных раздела: уровень безопасности для создания и поддержки пользователей, ролей и аутентификации; конфигурация приложения для управления парами "ключ–значение", настройками SMTP и отладки; а также конфигурация провайдера для управления способом хранения на сайте административных данных о пользователях.

Безопасность

Раздел безопасности содержит информацию о пользователях и их ролях, определенных на сайте, а также о режиме аутентификации. Пользователь может изменить индивидуальные настройки на этой странице, щелкнув на ассоциированных ссылках или используя мастер Security Setup Wizard для поочередного прохода по разделам настроек безопасности.

Режим аутентификации управляется страницей метода доступа (рис. 20.42). Если пользователь устанавливает переключатель Select Access Method в положение `From the Internet`, то приложение устанавливает режим аутентификации `Forms`, в то время как положение `From a Local Area Network` включает режим аутентификации системы Windows. Наиболее полезной особенностью приложения является возможность добавлять и редактировать роли. В мастере пользователь должен сначала включить механизм управления ролями, установив переключатель Web Site в положение `Enable Roles`. После активизации ролей пользователь может определить их либо с помощью мастера, либо страницы резюме. Каждая роль определяется одной строкой, а задача управления ролью в веб-приложении возлагается на пользователя (за исключением правил доступа, которые мы рассмотрим в свое время).

На следующем этапе мастер создает учетную запись пользователя. Информация на этой странице является повторением компонента CreateUserAccount и позволяет создавать первого пользователя, который будет администратором веб-сайта.

Страница правил доступа (рис. 20.43) позволяет пользователю ограничивать доступ к определенным частям своего сайта для некоторой роли или пользователя или предоставлять доступ только зарегистрированным пользователям. Как показано на рис. 20.43, по умолчанию существует единственное правило (которое, по существу, определено неявно и наследуется у сервера), определяющее полный доступ всем пользователям ко всему сайту.

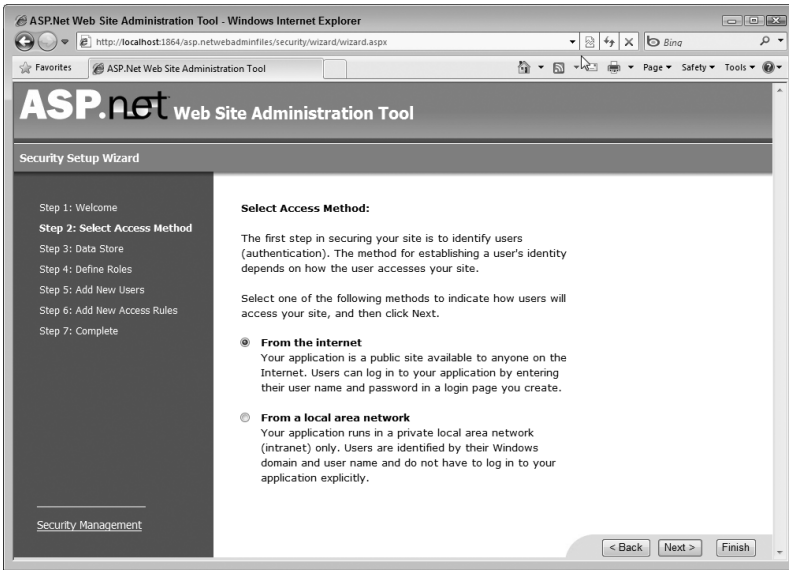


Рис. 20.42

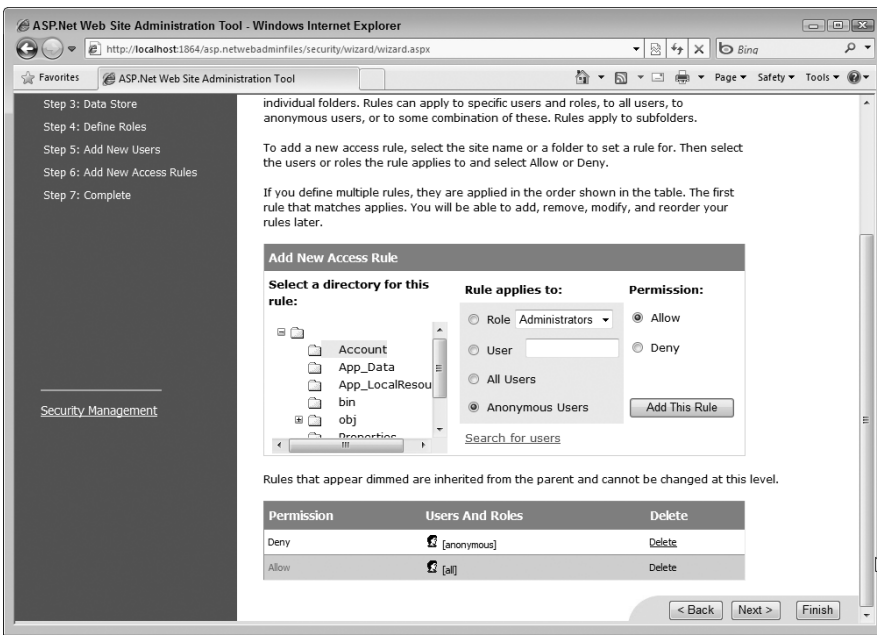


Рис. 20.43

Веб-сайт просматривает эти правила по порядку их определения, останавливаясь на первом правиле, которое можно применить в конкретном контексте. Например, если пользователь определил сначала правило, открывающее доступ к папке Admin для всех, кому присвоена роль Administrator, а затем определил последующую роль, которая закрывает доступ к этой папке для всех пользователей, он, по существу, блокирует доступ к папке Admin для всех пользователей, кому не назначена роль Administrator.

Определив пользователей, роли и правила, определенные на своем сайте, пользователь может запустить приложение, применяя правила доступа, щелкая на ссылке Manage Users, расположенной на странице безопасности. Она содержит также список всех пользователей, определенных в системе. Для того чтобы указать роли, которые назначены пользователям, необходимо щелкнуть на ссылке Edit User или Edit Roles.

Эту информацию можно использовать для настройки содержимого своих веб-страниц с помощью компонента LoginView, который был рассмотрен ранее.

Настройки приложения

Раздел приложения программы Web Site Administration Tool позволяет пользователю определять и редактировать настройки приложения в виде пар "ключ–значение", а также конфигурировать настройки электронной почты SMTP, включая почтовый сервер SMTP, заданный по умолчанию, и электронный адрес отправителя.

Разработчик может также указать, какой уровень отладки он хочет применить к приложению, и настроить отладочную информацию, которая должна храниться в ходе выполнения приложения.

Конфигурация ASP.NET в службе IIS

Если приложение ASP.NET уже развернуто на сервере производства, разработчик может редактировать настройки конфигурации непосредственно в службе Internet Information Services (IIS), расположенной в подразделе Administrative Tools раздела Control Panel. Когда платформа ASP.NET инсталлируется на машине, выясняется, что каждый веб-сайт (включая виртуальные каталоги) имеет набор средств конфигурации в службе IIS на страницах свойств (рис. 20.44).

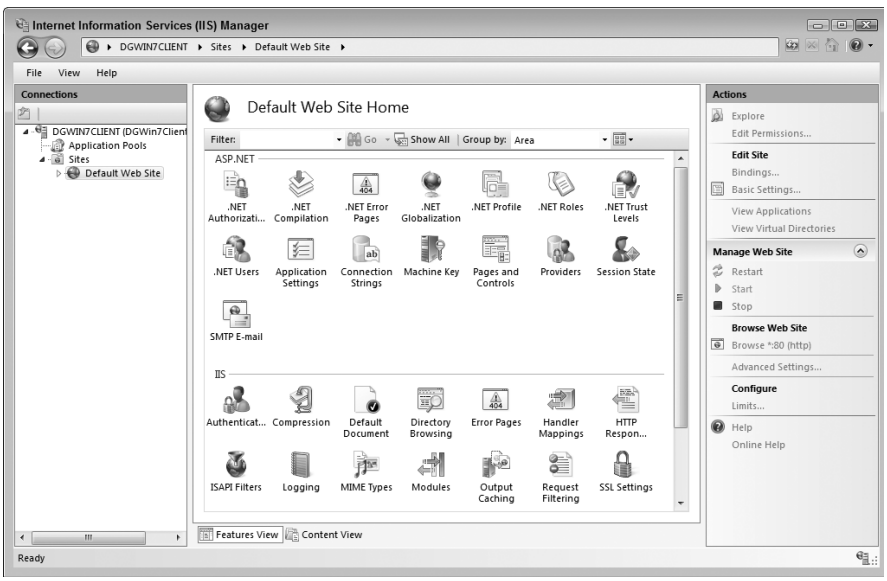


Рис. 20.44

Инструменты, включенные в службу IIS, позволяют управлять всеми настройками, включая настройки создания и управления пользователями, ролями, а также настройки приложения и SMTP. Разработчик также имеет доступ к более мощным средствам администрирования, позволяющим задавать сложные настройки, такие как поведение компилятора в системе .NET, уровень доверия в системе .NET и конфигурацию состояния сессии. Эти инструменты позволяют также поддерживать выполнение веб-приложения на любом IIS-сервере без редактирования конфигурационного файла `Web.config`.

Резюме

В этой главе мы показали, как создавать приложения ASP.NET с помощью проектов Web Site и Web Application. Улучшенный инструмент HTML Designer и новые средства CSS в системе Visual Studio 2010 предоставляют разработчику большую власть над компоновкой и визуальным проектированием веб-страниц. Огромное количество веб-элементов управления, включая элементы платформы ASP.NET, позволяют быстро собирать высокофункциональные веб-страницы. Благоразумное использование языка JavaScript, библиотеки ASP.NET AJAX и расширений элементов управления из пакета AJAX Control Toolkit обеспечивает очень сложные сценарии работы пользователей веб-приложений.

Разумеется, мы изложили не всю информацию, относящуюся к разработке веб-приложений. В главах 21 и 22 мы продолжим обсуждение методов создания полноценных веб-приложений и обсудим новейшие веб-технологии компании Microsoft: ASP.NET MVC и Silverlight. В главе 42 изложена подробная информация об инструментах и методах эффективной отладки веб-приложений. В заключение отметим, что в главе 49 мы рассмотрим вопросы, связанные с развертыванием веб-приложений. Более подробную информацию можно найти в книге Bill Evjen, Scott Hanselman, and Devin Rader *Professional ASP.NET 4 in C# and VB*. Эта книга содержит более 1 600 страниц и представляет собой наилучшее и наиболее полное руководство для веб-разработчиков, желающих создавать приложения с помощью новейшей версии ASP.NET.

21

Платформа ASP.NET MVC

В ЭТОЙ ГЛАВЕ...

- Анализ шаблона проектирования Model-View-Controller
- Разработка приложений на платформе ASP.NET MVC
- Разработка маршрутизаторов URL
- Проверка корректности информации, введенной пользователем
- Настройка шаблонов View на платформе ASP.NET MVC
- Интеграция с библиотекой jQuery

Выпустив первую версию платформы .NET Framework в 2002 году, компания Microsoft предложила новую абстракцию для разработки веб-приложений: технологию ASP.NET Web Forms. Традиционная технология Active Server Pages (ASP) была основана на простых шаблонах, представляющих собой комбинацию HTML-разметки и серверного кода. В отличие от нее, технология Web Forms была призвана приблизить разработку веб-приложений к моделям программирования обычных настольных приложений. Эта модель предлагала разработчику перетаскивать компоненты из инструментального окна на рабочую область, а затем настраивать их, задавая значения свойств и программируя обработку конкретных событий.

Несмотря на то что технология Web Forms была и остается очень успешной, у нее есть недостатки. Если не придерживаться строгой дисциплины, то бизнес-логика и вопросы доступа к данным в рамках этой технологии могут внедриться в пользовательский интерфейс. В результате тестировать такие приложения без браузера очень трудно. Кроме того, эта технология не учитывает природу веб, основанную на запросах и ответах и не имеющую состояний, так что отладка таких приложений становится весьма сложной. Она слишком сильно зависит от визуального представления элементов управления на основе их HTML-разметки. В итоге бывает трудно понять, каким будет окончательный вид каждой страницы.

В 2004 году выпуск простой платформы Ruby on Rails с открытым кодом для построения веб-приложений стал предвестником возрождения интереса к архитектурному шаблону Model-View-Controller (MVC). Шаблон MVC разделяет части пользовательского интерфейса на три категории с очень точно определенными ролями, что облегчает тестирование и эксплуатацию приложений.

Впервые компания Microsoft анонсировала платформу ASP.NET MVC на конференции ALT.NET в конце 2007 года. Эта платформа позволяет создавать приложения, основанные на архитектуре MVC, сохраняя преимущества использования множества библиотек и языковых возможностей платформы .NET. Платформа ASP.NET MVC была разработана как открытый проект, и многие ее функциональные возможности были созданы на основе обратной связи с широкими кругами программистов. В апреле 2009 года весь исходный код платформы был распространен под защитой лицензии Ms-PL.



Компания Microsoft проявила большую осторожность и заявила, что технология ASP.NET MVC не является заменой технологии Web Forms. Это просто альтернативный способ создания веб-приложений, который некоторые разработчики могут счесть более привлекательным. Компания Microsoft очень четко заявила, что будет и в будущем поддерживать обе технологии: ASP.NET Web Forms и ASP.NET MVC.

Шаблон Model-View-Controller

Если читатели до сих пор ничего не слышали об этом шаблоне, то могут очень удивиться, узнав, что "новый" архитектурный шаблон Model-View-Controller был впервые описан в 1979 году Тригве Ринскаугом (Trygve Reenskaug), исследователем, занимавшимся реализацией языка SmallTalk.

В архитектуре MVC приложения разделяются на несколько компонентов.

- **Модель.** Модель состоит из классов, реализующих предметную логику приложения. Несмотря на то что сама по себе архитектура MVC не содержит слоя доступа к данным, совершенно ясно, что эта модель должна инкапсулировать любой код, реализующий доступ к данным. В принципе эта модель назначает отдельные классы для доступа к данным, отвечающие за поиск и хранение информации в базе данных.
- **Представление.** Представления — это классы, получающие модель и представляющие ее в определенном формате, допускающем взаимодействие с пользователем.
- **Контроллер.** Отвечает за объединение компонентов в одно целое. Контроллер обрабатывает и реагирует на события, например, когда пользователь щелкает на кнопке. Контроллер отображает эти события в модель и вызывает соответствующее представление.

Эти описания ничего не дают, если не понятно, как взаимодействуют указанные части. Жизненный цикл запроса в приложении ASP.NET MVC, как правило, состоит из следующих этапов.

1. Пользователь выполняет некое действие, запускающее событие, например вводит адрес URL или щелкает на кнопке. В результате генерируется запрос к контроллеру.
2. Контроллер получает запрос и сообщает модели, какое действие она должна выполнить. Часто, хотя и не всегда, это приводит к изменению состояния модели.
3. Контроллер извлекает из модели все необходимые данные и вызывает соответствующее представление, передавая ему данные, полученные от модели.
4. Представление визуализирует данные и посылает их обратно пользователю.

Наиболее важный аспект этой архитектуры заключается в том, что и представление, и контроллер зависят от модели. Однако модель не имеет зависимостей. Именно в этом состоит основное преимущество данной архитектуры. Такое разделение обеспечивает более простое тестирование и упрощает управление.



Реализации архитектуры MVC на разных платформах немного отличаются структурой жизненного цикла. Например, в некоторых случаях представление посылает запрос модели о ее текущем состоянии, а не получает его от контроллера.

Разобравшись в архитектурном шаблоне Model-View-Controller, мы можем приступить к созданию первого ASP.NET MVC приложения.

Начало работы с платформой ASP.NET MVC

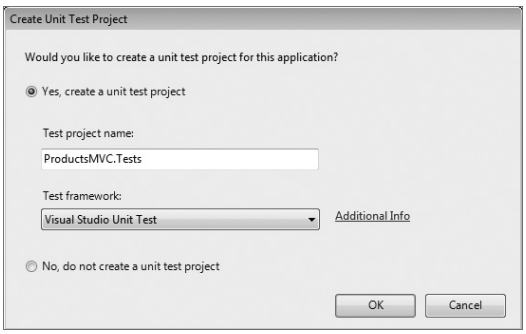


Рис. 21.1

В этом разделе описываются детали создания нового приложения по технологии ASP .NET MVC и некоторые стандартные компоненты. Для того чтобы создать новое приложение по шаблону MVC, необходимо выбрать команду File⇨New Project и пункт ASP.NET MVC 2.0 Application в разделе Web. После того как разработчик даст имя приложению и щелкнет на кнопке OK, система Visual Studio спросит у него, следует ли создать проект для модульного тестирования приложения (рис. 21.1). Несмотря на то что это не обязательно, мы настоятельно рекомендуем это сделать, потому что улучшенное тестирование является одним из главных преимуществ платформы MVC. Впрочем, по желанию тестовый проект можно добавить и позднее.



Система Visual Studio 2010 может создавать тестовые проекты для приложений MVC, используя множество каркасов для модульного тестирования. По умолчанию (см. рис. 21.1) разработчику предлагается использовать встроенные средства модульного тестирования системы Visual Studio. Если разработчик хочет использовать другие технологии модульного тестирования, он должен обратиться за инструкциями к поставщику и включить их в список доступных вариантов.

Когда приложение ASP.NET MVC создается в первый раз, оно генерирует большое количество файлов и папок. Фактически приложение MVC, которое генерируется по проектному шаблону, представляет собой законченное приложение, которое можно немедленно выполнить.

Структура папок, автоматически генерируемая системой Visual Studio, показана на рис. 21.2. Она включает следующие папки.

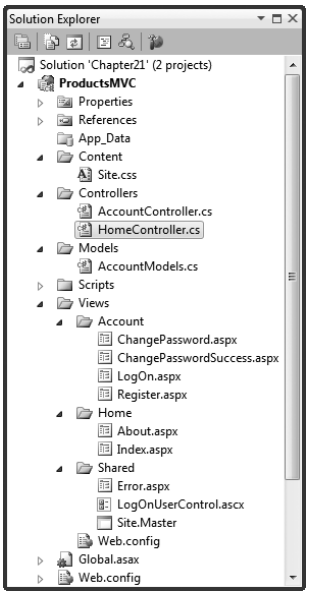


Рис. 21.2

- **`Content`.** Место, где хранятся файлы статического содержимого, например файлы CSS и изображения.
- **`Controllers`.** Папка, содержащая файлы контроллера. В соответствии с проектным шаблоном создаются два контроллера: `HomeController` и `AccountController`.
- **`Models`.** Папки, содержащие файлы модели. Это также подходящее место для хранения любых классов для доступа к данным, инкапсулированным в модель. Проектный шаблон MVC не создает примеров модели.
- **`Scripts`.** Папка, содержащая файлы на языке JavaScript. По умолчанию она содержит файлы сценариев для библиотек JQuery и Microsoft AJAX вместе с несколькими вспомогательными сценариями, обеспечивающими интеграцию с платформой MVC.
- **`Views`.** Папка, содержащая файлы представлений. Проектный шаблон MVC создает большое количество папок и файлов в папке `Views`. Папка нижнего уровня, `Home`, содержит два примера представлений, вызываемых контроллером `HomeController`. Папка нижнего уровня, `Shared`, содержит страницу мастера, используемую этими представлениями.

Система Visual Studio также создает файл `Default.aspx`, который является заглушкой, необходимой для правильной загрузки приложения MVC на сервер IIS. Существует также файл `Global.asax`, который используется для настроек правил маршрутизации (эту тему мы обсудим позднее).

И наконец, если разработчик решит создать тестовый проект, то он будет создан в папке `Controllers`, содержащей две заглушки для модульного тестирования контроллеров — `HomeController` и `AccountController` соответственно.

Несмотря на то что сделано немного, приложение MVC уже можно выполнить, нажав клавишу <F5>. Когда оно откроется в браузере Internet Explorer, он сначала визуализирует представление Index, содержащее ссылку, позволяющую перейти к представлению About. Ни одно из этих представлений особого интереса не представляет, поскольку они просто визуализируют статическое содержимое.

Выбор модели

В предыдущем разделе было указано, что проектный шаблон MVC не создает пример модели, предоставляя это разработчику. Фактически приложение может быть

выполнено вообще без модели. И хотя на практике приложения всегда имеют полную модель, технология MVC не имеет правил, указывающих, как их использовать. Это обеспечивает бо́льшую гибкость при разработке приложений.

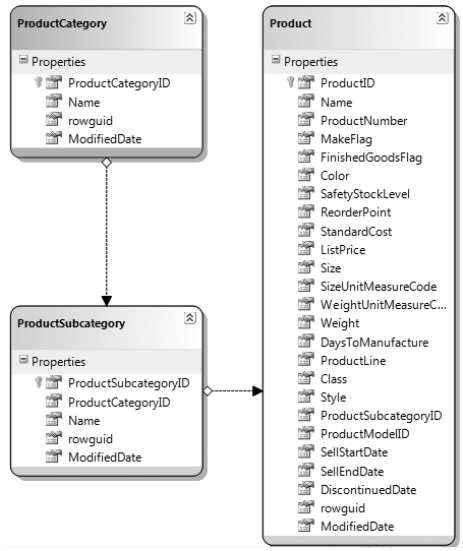


Рис. 21.3

Модель приложения — это абстракция его функциональных возможностей. Если разработчик создает приложение для обработки заказов или организовывает расписание отпусков, модель должна выражать эти концепции. Это не всегда просто. Часто бывает заманчиво внедрить эти детали в контроллер View своего приложения.

Примеры, приведенные в главе, используют простую модель LINQ to SQL, основанную на подмножестве примерной базы данных AdventureWorksDB, показанной на рис. 21.3. Эту базу данных можно загрузить с сайта `http://msftdbprodsamples.codeplex.com/`.

В следующем разделе мы объясним, как создать свой собственный контроллер, управляющий работой интересных представлений, визуализирующих динамический пользовательский интерфейс.

Контроллеры и методы действия

Контроллер — это класс, отвечающий за какое-то действие пользователя. Обычно он вызывает обновление модели и выполняет организацию представления для визуализации содержимого перед пользователем. Каждый контроллер способен отслеживать множество действий пользователей и реагировать на них. Каждая из этих реакций представляется в коде как обычный метод, называемый *методом действия* (action method).

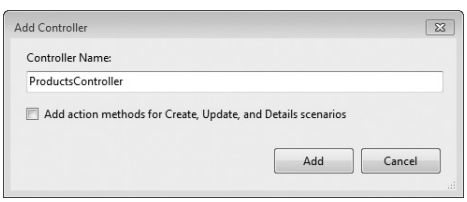


Рис. 21.4

Начните с щелчка правой кнопкой мыши на папке `Controllers` в окне Solution Explorer и выберите команду Add⇨Controller, открывающую диалоговое окно Add Controller (рис. 21.4). Это простое диалоговое окно позволяет выбрать имя для нового контроллера. По соглашению платформа MVC требует, чтобы все классы контроллеров имели имена, заканчивающиеся словом “Controller”, поэтому часть имени заполняется автоматически. Существует также флажок, позволяющий добавлять некоторые функциональные возможности. Пока мы их проигнорируем и вернемся к ним позднее. Присвойте новому контроллеру имя `ProductsController` и щелкните на кнопке Add.



Разработчик может быстро добавить контроллер в свой проект, используя комбинацию клавиш <Ctrl+M>, <Ctrl+C>.

Новые классы контроллеров являются производными от базового класса System.Web.Mvc.Controller, выполняющего всю тяжелую работу, связанную с определением релевантного метода для вызова в ответ на действие и отображение адреса URL и значений параметра POST. Это значит, что разработчик может сконцентрировать свое внимание на деталях реализации своих действий, которые обычно сводятся к вызову метода из класса моделей и выбору представления для визуализации.

Вновь созданный контроллер по умолчанию будет заполнен методом действия Index. Разработчик может добавить новое действие, добавив открытый метод в этот класс. Если метод является открытым, он будет видимым для любого действия над контроллером. Разработчик может предотвратить интерпретацию открытого метода как действия, добавив в него атрибут System.Web.Mvc.NonAction. Следующий код содержит класс контроллера с действием, заданным по умолчанию, которое визуализирует представление Index, и открытым методом, который остается невидимым как действие.

C#

```
public class ProductsController : Controller
{
  //
  // GET: /Products/

  public ActionResult Index()
  {
    return View();
  }

  [NonAction]
  public void NotAnAction()
  {
    // Этот метод не отмечен как метод действия.
  }
}
```

VB

```
Public Class ProductsController
    Inherits System.Web.Mvc.Controller

    '
    ' GET: /Products/

       Function Index() As ActionResult
        Return View()
    End Function

      <NonAction()>
    Sub NotAnAction()
        ' Этот метод не отмечен как метод действия.
    End Sub

End Class
```

Результатом метода Index является объект класса, производного от абстрактного класса System.Web.Mvc.ActionResult. Этот объект отвечает за определение того, что произойдет после возврата управления из метода действия. Существует большое количество стандартных классов, производных от класса ActionResult, позволяющих выполнять множество стандартных заданий, включая перенаправление на другой указатель URL, генерацию простого содержимого в разных форматах или, как в данном случае, визуализацию представления.



Комментарий, появляющийся перед методом Index, — это соглашение, указывающее способ активизирования события. Каждый метод действия размещается в указателе URL, представляющем собой комбинацию имени контроллера и имени метода действия в формате /controller/action. Данный комментарий не влияет на данное соглашение, а используется для указания, где можно найти этот метод действия. В данном случае он утверждает, что действие Index запускается при выполнении HTTP-запроса GET к URL /Products/. Это имя контроллера, поскольку, если в указателе URL ни одно действие не указано, то подразумевается действие Index. Данное соглашение обсуждается в разделе, посвященном маршрутизации.



Метод View в базовом классе Controller — это простой метод, создающий и настраивающий объект System.Web.Mvc.ViewResult. Этот объект отвечает за выбор представления и передачу всей информации, необходимой для визуализации его содержимого.

Следует отметить, что метод Index — это обычный метод платформы .NET, а класс ProductsController — обычный класс платформы .NET. В них нет ничего особенного. Это означает, что можно легко создавать экземпляры класса ProductsController в тестовом наборе, вызывать их метод Index, а затем делать предположения об объекте ActionResult, которые он возвращает.

Прежде чем идти далее, модифицируйте метод Index так, чтобы он находил список объектов класса Products и передавал их представлению, как показано в следующем коде.



C#

```
public ActionResult Index()
{
    List <Product> products;

    using (var db = new ProductsDataContext())
    {
        products = db.Products.ToList();
    }

    return View(products);
}
```

Сниппет кода Controllers\ProductsController.cs



VB

```
Function Index() As ActionResult
  Dim products As New List(Of Product)

  Using db As New ProductsDataContext
    products = db.Products.ToList()
  End Using

  Return View(products)
End Function
```

Сниппет кода Controllers\ProductsController.vb

Теперь, создав модель и контроллер, достаточно создать представление для визуализации пользовательского интерфейса.

Визуализация пользовательского интерфейса с помощью представлений

В предыдущем разделе мы создали метод действия, собиравший список продуктов и передававший его представлению. Каждое представление принадлежит отдельному контроллеру и хранится в папке нижнего уровня в папке `Views`. Имя представления указывается после имени контроллера, которому оно принадлежит. Кроме того, существует папка `Shared`, содержащая большое количество совместно используемых представлений, доступных из множества контроллеров. Когда механизм представления ищет конкретное представление, он сначала проверяет область хранения контроллеров, а затем совместно используемую область.



Если необходимо сослаться на представление, которое хранится где-то в другом месте, пользователь может задать полный путь к этому представлению.

Каждое представление очень похоже на стандартные файлы `Page` и `Control` на стандартной платформе ASP.NET Web Forms, имеющие расширения `.aspx` и `.ascx`. Они содержат комбинацию HTML-разметки и блоков кода. Они даже могут содержать страницы мастера и визуализировать некоторые стандартные элементы управления. Однако следует подчеркнуть, что между ними существует много различий.

Во-первых, представление не имеет страниц, основанных на коде. По этой причине не существует возможности добавлять обработчики событий для любого элемента управления, который визуализируется представлением, включая те из них, которые обычно находятся за сценой. Вместо этого ожидается, что за пользовательские события отвечает контроллер, а представление демонстрирует способы, которыми пользователь может запускать методы действия. Во-вторых, вместо наследования от класса `System.Web.Page` представление является производным классом от класса `System.Web.Mvc.ViewPage`. Этот базовый класс содержит множество полезных свойств и методов, которые можно использовать для облегчения визуализации разметки HTML. Одно из этих свойств содержит словарь объектов, которые передаются представлению от контроллера. В заключение заметим, что в разметке нет элемента управления формой с атрибутом `runat="server"`. Отсутствие серверной формы означает, что ни одно состояние `View State` не эмитируется вместе со страницей. Большинство серверных элементов управления на платформе ASP.NET должно быть размещено в серверной форме. Некоторые элементы управления, такие как Literal и Repeater, прекрасно работают вне формы; однако, если попытаться использовать элемент управления Button или DropDownList, то на этапе выполнения страница сгенерирует исключение. Разработчик может создать представление разными способами, но проще всего щелкнуть правой кнопкой мыши на заголовке метода действия и выбрать команду Add View, открывающую диалоговое окно Add View (рис. 21.5).



Для того чтобы открыть окно Add View, можно также использовать комбинацию клавиш <Ctrl+M>, <Ctrl+V>.

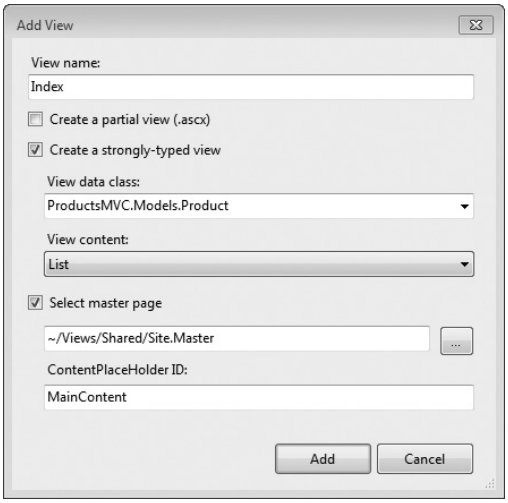


Рис. 21.5

Это диалоговое окно содержит множество опций. По умолчанию имя представления должно совпадать с именем метода действия. Если его изменить, то придется изменить вызов метода `View`, чтобы включить имя представления как параметр. Для того чтобы создать строго типизированное представление, необходимо установить соответствующий флажок и выбрать пункт `Models.Product` из раскрывающегося списка View Data Class. Если пункта Product в списке нет, разработчик должен создать приложение перед тем, как добавить в него представление.



Если пользователь предпочитает не создавать строго типизированное представление, оно будет содержать словарь объектов, которые перед использованием необходимо преобразовать обратно в их реальные типы. Рекомендуется всегда использовать строго типизированные представления. Если пользователю, работающему на языке C#, все же нужны слабо типизированные представления, то следует создать строго типизированное представление, имеющее новый динамический тип, и передать его экземплярам класса `ExpandoObject`.

В заключение измените список `View Content` на `List`. В этом случае система Visual Studio сгенерирует страницу списка для объектов класса `Product`. Когда пользователь щелкнет на кнопке Add, должно генерироваться представление, открывающееся в главном окне редактора. Этот код приведен ниже.



C#

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage <IEnumerable
                                   <CSProductsMVC.Models.Product>> " %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
                runat="server">
    Index
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
                runat="server">

    <h2>Index</h2>

    <table>
        <tr>
            <th></th>
            <th>ProductID</th>
            <th>Name</th>
            <th>ProductNumber</th>
```

```
            <th>MakeFlag</th>
            <th>FinishedGoodsFlag</th>
            <th>Color</th>
            <th>SafetyStockLevel</th>
            <th>ReorderPont</th>
            <th>StandardCost</th>
            <th>ListPrice</th>
            <th>Size</th>
            <th>SizeUnitMeasureCode</th>
            <th>WeightUnitMeasureCode</th>
            <th>Weight</th>
            <th>DaysToManufacture</th>
            <th>ProductLine</th>
            <th>Class</th>
            <th>Style</th>
            <th>ProductSubcategoryID</th>
            <th>ProductModelID</th>
            <th>SellStartDate</th>
            <th>SellEndDate</th>
            <th>DiscontinuedDate</th>
            <th>rowguid</th>
            <th>ModifiedDate</th>
        </tr>

    <% foreach (var item in Model) { %>

        <tr>
            <td>
            <%= Html.ActionLink("Edit", "Edit", new
            { id=item.ProductID }) %> |
            <%= Html.ActionLink("Details", "Details", new
            { id=item.ProductID })%>
            </td>
            <td><%= Html.Encode(item.ProductID) %></td>
            <td><%= Html.Encode(item.Name) %></td>
            <td><%= Html.Encode(item.ProductNumber) %></td>
            <td><%= Html.Encode(item.MakeFlag) %></td>
            <td><%= Html.Encode(item.FinishedGoodsFlag) %></td>
            <td><%= Html.Encode(item.Color) %></td>
            <td><%= Html.Encode(item.SafetyStockLevel) %></td>
            <td><%= Html.Encode(item.ReorderPoint) %></td>
            <td><%= Html.Encode(String.Format("{0:F}",
                                              item.StandardCost)) %></td>
            <td><%= Html.Encode(String.Format("{0:F}",
                                              item.ListPrice)) %></td>
            <td><%= Html.Encode(item.Size) %></td>
            <td><%= Html.Encode(item.SizeUnitMeasureCode) %></td>
            <td><%= Html.Encode(item.WeightUnitMeasureCode) %></td>
            <td><%= Html.Encode(String.Format("{0:F}",
                                              item.Weight)) %></td>
            <td><%= Html.Encode(item.DaysToManufacture) %></td>
            <td><%= Html.Encode(item.ProductLine) %></td>
            <td><%= Html.Encode(item.Class) %></td>
            <td><%= Html.Encode(item.Style) %></td>
            <td><%= Html.Encode(item.ProductSubcategoryID) %></td>
            <td><%= Html.Encode(item.ProductModelID) %></td>
            <td><%= Html.Encode(String.Format("{0:g}",
                                              item.SellStartDate)) %></td>
            <td><%= Html.Encode(String.Format("{0:g}",
                                              item.SellEndDate)) %></td>
```

```
            <td><%= Html.Encode(String.Format("{0:g}",
                                              item.DiscontinuedDate)) %></td>
            <td><%= Html.Encode(item.rowguid) %></td>
            <td><%= Html.Encode(String.Format("{0:g}",
                                              item.ModifiedDate)) %></td>
        </tr>

    <% } %>

    </table>

    <p>
        <%= Html.ActionLink("Create New", "Create") %>
    </p>

</asp:Content>
```



VB

```
<%@ Page Title="" Language="VB" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage(Of IEnumerable
                                        (Of ProductsMVC.Product))" %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
runat="server">
    Index
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
runat="server">

    <h2>Index</h2>

    <p>
        <%=Html.ActionLink("Create New", "Create")%>
    </p>

    <table>
        <tr>
            <th></th>
            <th>ProductID</th>
            <th>Name</th>
            <th>ProductNumber</th>
            <th>MakeFlag</th>
            <th>FinishedGoodsFlag</th>
            <th>Color</th>
            <th>SafetyStockLevel</th>
            <th>ReorderPoint</th>
            <th>StandardCost</th>
            <th>ListPrice</th>
            <th>Size</th>
            <th>SizeUnitMeasureCode</th>
            <th>WeightUnitMeasureCode</th>
            <th>Weight</th>
            <th>DaysToManufacture</th>
            <th>ProductLine</th>
            <th>Class</th>
            <th>Style</th>
            <th>ProductSubcategoryID</th>
            <th>ProductModelID</th>
            <th>SellStartDate</th>
            <th>SellEndDate</th>
            <th>DiscontinuedDate</th>
```

```
            <th>rowguid</th>
            <th>ModifiedDate</th>
        </tr>

<% For Each item In Model%>

        <tr>
            <td>
                <%=Html.ActionLink("Edit", "Edit", New With
                {.id = item.ProductID})%> |
                <%=Html.ActionLink("Details", "Details", New With
                {.id = item.ProductID})%>
            </td>
            <td><%= Html.Encode(item.ProductID) %></td>
            <td><%= Html.Encode(item.Name) %></td>
            <td><%= Html.Encode(item.ProductNumber) %></td>
            <td><%= Html.Encode(item.MakeFlag) %></td>
            <td><%= Html.Encode(item.FinishedGoodsFlag) %></td>
            <td><%= Html.Encode(item.Color) %></td>
            <td><%= Html.Encode(item.SafetyStockLevel) %></td>
            <td><%= Html.Encode(item.ReorderPoint) %></td>
            <td><%= Html.Encode(String.Format("{0:F}",
                                    item.StandardCost)) %></td>
            <td><%= Html.Encode(String.Format("{0:F}",
                                    item.ListPrice)) %></td>
            <td><%= Html.Encode(item.Size) %></td>
            <td><%= Html.Encode(item.SizeUnitMeasureCode) %></td>
            <td><%= Html.Encode(item.WeightUnitMeasureCode) %></td>
            <td><%= Html.Encode(String.Format ("{0:F}",
                                    item.Weight)) %></td>
            <td><%= Html.Encode(item.DaysToManufacture) %></td>
            <td><%= Html.Encode(item.ProductLine) %></td>
            <td><%= Html.Encode(item.Class) %></td>
            <td><%= Html.Encode(item.Style) %></td>
            <td><%= Html.Encode(item.ProductSubcategoryID) %></td>
            <td><%= Html.Encode(item.ProductModelID) %></td>
            <td><%= Html.Encode(String.Format("{0:g}",
                                    item.SellStartDate)) %></td>
            <td><%= Html.Encode(String.Format("{0:g}",
                                    item.SellEndDate)) %></td>
            <td><%= Html.Encode(String.Format("{0:g}",
                                    item.DiscontinuedDate)) %></td>
            <td><%= Html.Encode(item.rowguid) %></td>
            <td><%= Html.Encode(String.Format("{0:g}",
                                    item.ModifiedDate)) %></td>
        </tr>

    <% Next%>

    </table>

</asp:Content>
```

Сниппет кода Views\Products\Index.aspx

Данное представление визуализирует список объектов класса `Products` в виде простой таблицы. Основная работа выполняется в цикле “for each”, который проходит по списку продуктов и визуализирует строки HTML-таблицы одну за другой.

C#

```
<% foreach (var item in Model) { %>

    <tr>
       <!-- ... -->
       <td><%= Html.Encode(item.ProductID) %></td>
       <td><%= Html.Encode(item.Name) %></td>
       <!-- ... -->
    </tr>

<% } %>
```

VB

```
<% For Each item In Model%>

       <tr>
          <!-- ... -->
          <td><%= Html.Encode(item.ProductID) %></td>
          <td><%= Html.Encode(item.Name) %></td>
          <!-- ... -->

       </tr>

<% Next%>
```



Система Visual Studio способна выводить тип модели, поскольку пользователь создал строго типизированное представление. В директиве страницы можно видеть, что это представление не является наследником класса `System.Web.Mvc.Page`. Вместо этого оно наследует свойства обобщенной версии, которая утверждает, что модель является коллекцией `IEnumerable` объектов класса `Product`. В свою очередь, они имеют свойство `Model` с этим типом. Обратите внимание на то, что пользователь попрежнему может заставлять контроллер передавать представлению неправильный тип. В случае строго типизированного представления это приведет к генерации исключения во время выполнения приложения.

Перед визуализацией продуктов с помощью метода `Encode` вспомогательного свойства класса `Html` каждое из свойств продуктов кодируется на языке HTML. Это предотвращает злонамеренное внедрение в приложение нежелательного кода под видом корректных данных. Платформа ASP.NET MVC способна упростить это кодирование благодаря новой разметке `<%: … %>`, в которой используется двоеточие вместо знака равенства, принятого в версии ASP.NET 4 и более ранних версиях этой платформы. Рассмотрим тот же самый сниппет кода, используя преимущества этой технологии.

C#

```
    <% foreach (var item in Model) { %>

       <tr>
           <!-- ... -->
           <td><%: item.ProductID %></td>
           <td><%: item.Name %></td>
       <!-- ... -->

    </tr>

<% } %>
```

VB

```
<% For Each item In Model%>

    <tr>
        <!-- ... -->
        <td><%: item.ProductID %></td>
        <td><%: item.Name %></td>
        <!-- ... -->

    </tr>

<% Next%>
```

Кроме метода Encode, в данном представлении используется еще один вспомогательный метод класса Html. Этот метод называется ActionLink. Он будет эмитировать стандартный якорный дескриптор HTML, предназначенный для запуска конкретного действия. Здесь использованы две формы. Простейшая из них создает новую запись о продукте.

C#

```
<p>
  <%= Html.ActionLink("Create New", "Create") %>
</p>
```

VB

```
<p>
  <%=Html.ActionLink("Create New", "Create")%>
</p>
```

Первый параметр является текстом, который будет визуализирован в якорном дескрипторе. Именно этот текст будет представлен пользователю приложения. Второй параметр — это имя действия, которое должно быть активизировано. Поскольку в программе не указан ни один контроллер, подразумевается текущий контроллер.

Более сложная форма использования метода ActionLink предназначена для визуализации ссылки редактирования и удаления для каждого продукта.

C#

```
<td>
  <%= Html.ActionLink("Edit", "Edit", new { id=item.ProductID }) %> |
  <%= Html.ActionLink("Details", "Details", new { id=item.ProductID })%>
</td>
```

VB

```
<td>
  <%=Html.ActionLink("Edit", "Edit", New With {.id = item.ProductID})%> |
  <%=Html.ActionLink("Details", "Details", New With {.id =
item.ProductID})%>
</td>
```

Первые два параметра совпадают с предыдущими и представляют собой текстовую ссылку и имя действия соответственно. Третий параметр — это анонимный объект, содержащий данные, которые должны передаваться методу действия при его вызове.

Если запустить приложение и ввести в строке адреса путь /products/, на экране откроется страница, показанная на рис. 21.6. При попытке щелкнуть на какой-нибудь из этих ссылок возникнет исключение, поскольку целевое действие пока не существует.

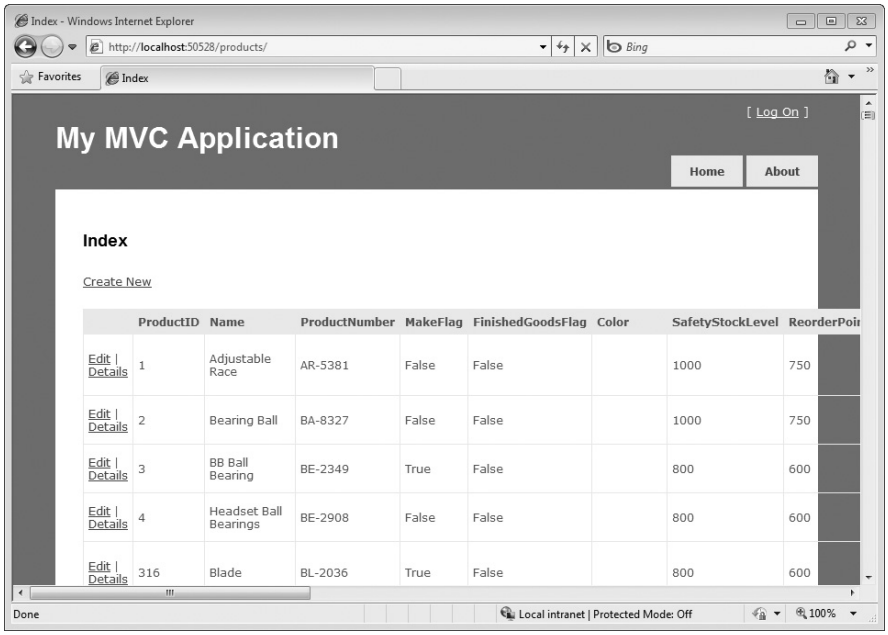


Рис. 21.6

Продвинутые возможности технологии MVC

В настоящем разделе приводится обзор более сложных возможностей технологии ASP.NET MVC.

Маршрутизация

Просматривая сайт, созданный по технологии MVC, в своем веб-браузере, можно заметить, что его указатели URL довольно сильно отличаются от указателей URL на обычных сайтах, созданных по технологии ASP.NET. Они не содержат расширений файлов и не соответствуют базовой структуре папок. Эти указатели URL отображаются в методы действия и контроллеры с набором классов, принадлежащих механизму маршрутизации, расположенному в сборке `System.Web.Routing`.



Для переключения между представлением и контроллером можно использовать комбинацию клавиш <Ctrl+M>, <Ctrl+G>. Механизм маршрутизации изначально был разработан как часть проекта ASP.NET MVC, но был выпущен в виде отдельной библиотеки еще до выпуска самой платформы MVC. Несмотря на то что он в книге не описывается, его можно использовать в проектах ASP.NET Web Forms

В предыдущем примере мы создали просто список представления для продуктов. Этот список был основан на стандартном шаблоне List, который визуализирует следующий сниппет для каждого объекта класса `Product`, находящегося в базе данных, записи которой подлежат выводу на дисплей.

C#

```
<td>
  <%= Html.ActionLink("Edit", "Edit", new { id=item.ProductID }) %> |
  <%= Html.ActionLink("Details", "Details", new { id=item.ProductID })%>
</td>
```

VB

```
<td>
  <%=Html.ActionLink("Edit", "Edit", New With {.id = item.ProductID})%> |
  <%=Html.ActionLink("Details", "Details", New With {.id = item.ProductID})%>
</td>
```

Если посмотреть на сгенерированную HTML-разметку, то можно увидеть следующее.

HTML

```
<td>
  <a href="/Products/Edit/2">Edit</a> |
  <a href="/Products/Details/2">Details</a>
</td>
```

Эти указатели URL состоят из трех частей.

- "`Products`" — это имя контроллера. В проекте существует соответствующий класс `ProductsController`.
- "`Edit`" и "`Details`" — имена действий контроллера. Класс `ProductsController` имеет методы `Edit` и `Details`.
- "`2`" — параметр с именем "`id`".

Каждый из этих компонентов определен в *маршрутизаторе* (route), который описывается в файле `Global.asax.cs` (или в файле `Global.asax.vb` для языка VB) в методе `RegisterRoutes`. При первом запуске приложения оно вызывает этот метод и переходит в статическую коллекцию `System.Web.Routing.RouteTable.Routes`.

Эта коллекция содержит все маршрутизаторы для всего приложения.



C#

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        "Default",
        "{controller}/{action}/{id}",
        new { controller = "Home", action = "Index", id = "" }
    );
}
```

Сниппет кода Global.asax.cs



VB

```
Shared Sub RegisterRoutes(ByVal routes As RouteCollection)
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}")

    routes.MapRoute(_
        "Default",
        "{controller}/{action}/{id}", _
        New With {.controller = "Home", .action = "Index", .id = ""} _
    )

End Sub
```

Сниппет кода Global.asax.vb

Первый вызов метода сообщает механизму маршрутизации, что он должен игнорировать все запросы на файлы с расширением .axd. Если входящий указатель URL соответствует этому критерию, то механизм маршрутизации полностью игнорирует его и позволяет обработать его другим частям приложения. Данный метод может быть очень удобным, если необходимо интегрировать платформы Web Forms и MVC в рамках одного приложения. Все, что требуется для этого, — попросить механизм маршрутизатора игнорировать файлы с расширениями .aspx и .asmx.

Второй вызов метода определяет новый объект Route и добавляет его в коллекцию. При этом происходит замещение метода MapRoute, получающего три параметра. Первый параметр — это имя, которое позднее используется как имя маршрутизатора. Второй параметр — это шаблон указателя URL. Этот параметр может представлять собой обычный текст, содержащий специальные лексемы в фигурных скобках. Эти лексемы будут использоваться как заглушки, которые заполняются при условии, что маршрутизатор соответствует указателю URL. Некоторые лексемы резервируются и будут использоваться механизмом маршрутизации MVC для выбора контроллера и выполнения правильного действия. Последний параметр — это словарь значений, заданных по умолчанию. Как видим, маршрутизатор "Default" соответствует любому указателю URL, имеющему вид /controller/action/id, где контроллером по умолчанию является контроллер с именем “Home”, действие по умолчанию называется “Index”, а параметр “id” по умолчанию является пустой строкой.

Когда поступает новый запрос HTTP, каждый маршрутизатор в коллекции RouteCollection пытается сравнить указатель URL со своим шаблоном URL в порядке, в котором они добавлялись. Первый маршрутизатор, способный это сделать, заполняет все значения, задаваемые по умолчанию, если они не были переданы. Как только будут собраны все эти значения, создается объект Controller и вызывается метод действия.

Маршрутизаторы также используются для генерации указателей URL в представлениях. Когда вспомогательному методу требуется указатель URL, он проверяет каждый маршрутизатор (также по порядку), чтобы узнать, способен ли он создать указатель для заданного контроллера, действие и значения параметров. Первый маршрутизатор, соответствующий этому критерию, сгенерирует правильный указатель. Если маршрутизатор обнаруживает значение параметра, которое ему не известно, то оно становится строкой запроса в генерируемом указателе URL.

Следующий сниппет кода объявляет новый маршрутизатор для интерактивного магазина с двумя параметрами: категория и подкатегория. Предполагая, что приложение MVC развернуто в корневом каталоге веб-сервера, запрос на указатель http://servername/Shop/Accessories/Helmets поступит в метод действия “List” в контроллере “Products” с параметром Category, равным “Accessories”, и параметром Subcategory, равным “Helmets”.



C#

```
public static void RegisterRoutes(RouteCollection routes)
{
  routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

  routes.MapRoute(
      "ProductsDisplay",
      "Shop/{category}/{subcategory}",
      new {
```

```
            controller = "Products",
            action = "List",
            category = "",
            subcategory = ""
        }
    );

    routes.MapRoute(
        "Default",
        "{controller}/{action}/{id}",
        new { controller = "Home", action = "Index", id = "" }
    );
}
```

Сниппет кода Global.asax.cs



VB

```
Shared Sub RegisterRoutes(ByVal routes As RouteCollection)
  routes.IgnoreRoute("{resource}.axd/{*pathInfo}")

    routes.MapRoute(_
      "ProductsDisplay", _
      "Shop/{category}/{subcategory}", _
      New With { _
      .controller = "Products", .action = "List", _
      .category = "", .subcategory = "" _
    })

    routes.MapRoute(_
      "Default", _
      "{controller}/{action}/{id}", _
      New With {.controller = "Home", .action = "Index", .id = ""} _
    )

End Sub
```

Сниппет кода Global.asax.vb



Как только маршрутизатор `Route` из коллекции `RouteCollection` находит соответствие с указателем URL, ни один другой маршрутизатор больше не имеет этой возможности. По этой причине порядок, в котором маршрутизаторы добавляются в коллекцию, может иметь довольно большое значение. Если бы предыдущий сниппет был помещен в коллекцию после маршрутизатора `Default`, то он никогда бы не смог сравнивать входящие запросы, потому что запрос на `/Shop/Accessories/Helmets` искал бы метод действия `Accessories` в объекте `ShopController` с идентификаторами "`id`", равными "`Helmets`". Поскольку такого объекта `ShopController` не существует, запрос не был бы выполнен. Если приложение не обращается к заданному методу действия за указателем URL, пользователь может пожелать добавить более конкретный маршрутизатор `Route` в коллекцию `RouteCollection` до более общего маршрутизатора или удалить более общие маршрутизаторы, пока проблема не будет решена.

В заключение отметим, что разработчик может также добавлять ограничения для маршрутизатора `Route`, которые будут предотвращать сопоставление с указателем URL, пока не будет выполнено другое условие. Это может оказаться хорошей идеей, если параметры должны быть преобразованы в сложные типы данных, например, даты требуют очень специфичного формата. Основным ограничением является строка.

Она интерпретируется как регулярное выражение, которое параметр должен сравнивать с маршрутизатором, чтобы действие было выполнено. Эта технология используется в следующем определении маршрутизатора, чтобы гарантировать, что параметр `zipCode` имеет ровно пять цифр.

C#

```
routes.MapRoute(
  "StoreFinder",
  "Stores/Find/{zipCode}",
  new { controller = "StoreFinder", action = "list" },
  new { zipCode = @"^\d{5}$" }
);
```

VB

```
routes.MapRoute(_
  "StoreFinder", _
  "Stores/Find/{zipCode}", _
  New With {.controller = "StoreFinder", .action = "list"}, _
  New With {.zipCode = "^\d{5}$"} _
)
```

Другим типом ограничений является класс, реализующий интерфейс `IrouteConstraint`. Данный интерфейс определяет единственный метод `Match`, возвращающий значение типа `Boolean`, которое означает, соответствует ли входящий запрос заданному ограничению. За пределами блока `HttpMethodConstraint` существует одна реализация интерфейса `IRouteConstraint`. Это ограничение можно использовать, чтобы гарантировать, что будет использован правильный HTTP-метод, например `GET`, `POST`, `HEAD` и `DELETE`. Следующий маршрутизатор принимает только запросы HTTP POST.

C#

```
routes.MapRoute(
  "PostOnlyRoute",
  "Post/{action}",
  new { controller = "Post" },
  new { post = new HttpMethodConstraint("POST") }
);
```

VB

```
routes.MapRoute(
  "PostOnlyRoute", _
  "Post/{action}", _
  New With {.controller = "Post"}, _
  New With {.post = New HttpMethodConstraint("POST")} _
)
```

Классы маршрутизаторов URL — очень мощное и гибкое средство. Они позволяют легко создавать "красивые" указатели URL. Это может помочь пользователям перемещаться по сайту и даже улучшить его рейтинг в поисковых машинах.

Параметры методов действия

Все методы действия из предыдущих примеров не получали извне никаких данных для решения своей задачи; они целиком полагались на состояние модели. В реальных приложениях этот сценарий мало вероятен. Технология ASP.NET MVC позволяет очень легко параметризовать методы действия, чтобы они получали информацию из разных источников.

Как указывалось в предыдущей главе, маршрутизатор "Default" имеет параметр "id", который по умолчанию является пустой строкой. Для того чтобы получить значение параметра "id" в методе действия, можно просто добавить его в сигнатуру, как показано в следующем сниппете.



C#

```
public ActionResult Details(int id)
{
  using (var db = new ProductsDataContext())
  {
    var product = db.Products.SingleOrDefault(x => x.ProductID == id);

    if (product == null)
      return View("NotFound");

    return View(product);
  }
}
```

Сниппет кода Controllers\ProductsController.cs



VB

```
Public Function Details(ByVal id As Integer) As ActionResult
  Using db As New ProductsDataContext
    Dim product = db.Products.FirstOrDefault(Function(p As Product)
    p.ProductID = id)

    Return View(product)
  End Using
End Function
```

Сниппет кода Controllers\ProductsController.vb

Когда платформа MVC выполняет метод действий Details, она просматривает параметры, извлеченные из указателя URL маршрутизатором, выполнявшим сравнение. Эти параметры сравниваются с параметрами метода действий по имени, а затем передаются туда, откуда поступил вызов метода.

Как показывает пример метода Details, платформа способна даже преобразовывать тип параметра на лету. С помощью этой же технологии методы действия могут также извлекать параметры из строки запроса, содержащейся в указателе URL, а также из данных HTTP POST.



Если преобразование невозможно, генерируется исключение.

Кроме того, метод действия может получать параметр типа FormValues, агрегирующий все данные HTTP POST в одном параметре. Если данные в коллекции FormValues представляют свойства объекта, то можно просто добавить параметр этого типа и при вызове метода действия будет создан его новый экземпляр. Действие Create, показанное в следующем сниппете, использует эту технологию для создания и сохранения нового экземпляра класса Product.



C#

```
public ActionResult Create()
{
  return View();
}
```

```
[HttpPost]
public ActionResult Create([Bind(Exclude="ProductId")]Product product)
{
  if (!ModelState.IsValid)
     return View();

  using (var db = new ProductsDataContext())
  {
     db.Products.InsertOnSubmit(product);
     db.SubmitChanges();
  }
  return RedirectToAction("List");
}
```

Сниппет кода Controllers\ProductsController.cs



VB

```
<HttpPost()>
Function Create(<Bind(Exclude:="id")> ByVal product As Product)

  If (Not ModelState.IsValid) Then
    Return View()
  End If

  Using db As New ProductsDataContext
    db.Products.InsertOnSubmit(product)
    db.SubmitChanges()
  End Using
  Return RedirectToAction("List")
End Function
```

Сниппет кода Controllers\ProductsController.vb



Здесь есть два метода действия `Create`. Первый метод визуализирует представление "`Create`". Второй размечен с помощью атрибута `HttpPostAttribute`, т.е. он будет выбран, только если HTTP-запрос использует слово `POST`. Это обычная практика при проектировании веб-сайтов с помощью технологии ASP.NET MVC. Кроме атрибута `HttpPostAttribute`, существуют соответствующие атрибуты для действий `GET`, `PUT` и `DELETE`.

Средство привязки модели

Процесс создания нового экземпляра класса `Product` выполняется *средством привязки модели* (model binder). Средство привязки модели сравнивает свойства данных HTTP POST со свойствами типа, который он пытается создать. В данном примере это срабатывает, потому что шаблон, использованный для генерации представления "`Create`", визуализирует поля HTML INPUT с правильным именем, что подтверждает следующий сниппет кода.

HTML

```
<p>
  <label for="ProductID">ProductID:</label>
  <input id="ProductID" name="ProductID" type="text" value="" />
</p>
<p>
  <label for="Name">Name:</label>
  <input id="Name" name="Name" type="text" value="" />
</p>
```

Существует множество способов управлять поведением средства привязки модели, включая атрибут `BindAttribute`, который используется в методе `Create`, показан-

ном ранее. Этот атрибут используется для включения или исключения определенных свойств и для указания префикса для значений HTTP POST. Это может быть полезным, если необходимо связать несколько объектов в коллекции POST.

Средства связывания моделей можно также использовать в методе действий для обновления существующих экземпляров классов модели, используя методы `UpdateModel` и `TryUpdateModel`. Основная разница между ними заключается в том, что метод `TryUpdateModel` возвращает булево значение, служащее индикатором, смог ли он создать успешную модель, а метод `UpdateModel` генерирует исключение, если не смог выполнить задание. Эта технология продемонстрирована на примере метода действия `Edit`.



C#

```
[HttpPost]
public ActionResult Edit(int id, FormCollection formValues)
{
  using (var db = new ProductsDataContext())
  {
    var product = db.Products.SingleOrDefault(x => x.ProductID == id);

    if (TryUpdateModel(product))
    {
      db.SubmitChanges();
      return RedirectToAction("Index");
    }
    return View(product);
  }
}
```

Сниппет кода Controllers\ProductsController.cs



VB

```
<HttpPost()>
Function Edit(ByVal id As Integer, ByVal formValues As FormCollection)
  Using db As New ProductsDataContext
    Dim product = db.Products.FirstOrDefault(Function(p As Product)
    p.ProductID = id)

    If TryUpdateModel(product) Then
      db.SubmitChanges()
      Return RedirectToAction("Index")
    End If
    Return View(product)
  End Using
End Function
```

Сниппет кода Controllers\ProductsController.vb

Области

Область (area) — это изолированная часть приложения MVC, управляющая своими собственными моделями, контроллерами и представлениями. В области можно даже определить свои собственные маршрутизаторы. Для того чтобы создать новую область, необходимо выбрать команду Add⇨Area из контекстного меню проекта в окне Solution Explorer. В открывшемся диалоговом окне Add Area, показанном на рис. 21.7, будет предложено ввести имя области.

После того как разработчик щелкнет на кнопке Add, в проект будут добавлены многочисленные файлы, нужные для создания области. На рис. 21.8 показан проект с двумя областями, имеющими имена `Shop` и `Blog` соответственно.



Рис. 21.7



Рис. 21.8

Кроме собственных контроллеров и представлений, каждая область имеет класс `AreaNameAreaRegistration`, являющийся производным от абстрактного базового класса `AreaRegistration`. Этот класс содержит абстрактное свойство для имени собственной области и абстрактный метод для ее интеграции с остальной частью приложения. По умолчанию реализация регистрирует стандартные маршруты.



C#

```
public class BlogAreaRegistration : AreaRegistration
{
  public override string AreaName
  {
    get
    {
      return "Blog";
    }
}

public override void RegisterArea(AreaRegistrationContext context)
{
    context.MapRoute(
      "Blog_default",
      "Blog/{controller}/{action}/{id}",
      new { action = "Index", id = "" }
    );
  }
}
```

Сниппет кода Areas\Blog\BlogAreaRegistration.cs



VB

```
Public Class BlogAreaRegistration
  Inherits AreaRegistration

Public Overrides ReadOnly Property AreaName() As String
    Get
```

```
      Return "Blog"
    End Get
End Property

Public Overrides Sub RegisterArea(ByVal context As
AreaRegistrationContext)
    context.MapRoute(_
       "Blog_default", _
       "Blog/{controller}/{action}/{id}", _
       New With {.action = "Index", .id = ""} _
     )
  End Sub
End Class
```

Сниппет кода Areas\Blog\BlogAreaRegistration.vb



Метод `RegisterArea` класса `BlogAreaRegistration` определяет маршрут, в котором по соглашению каждый указатель имеет префикс `/Blog/`. Это может быть полезно для отладки, но не является необходимым, поскольку область маршрутов не пересекается ни с одним другим маршрутом.

Для того чтобы связаться с контроллером, расположенным за пределами области, необходимо использовать перегруженный вариант метода `Html.ActionLink`, который принимает параметр `routeValues`. Свойство `area` объекта, созданного для этого параметра, должно содержать имя области, содержащей контроллер, к которому вы стремитесь подключиться.

C#

```
<%= Html.ActionLink("Blog", "Index", new { area = "Blog" }) %>
```

VB

```
<%= Html.ActionLink("Blog", "Index", New With {.area = "Blog"})%>
```

Довольно часто при разделении проекта на области фабрика контроллеров обнаруживает, что в проекте существует несколько контроллеров с одинаковыми именами, и перестает работать. Для того чтобы этого избежать, можно ограничить пространства имен так, чтобы маршрут, используемый для поиска контроллера, удовлетворял любые запросы. Следующий сниппет ограничивает пространства имен глобальными маршрутами пространством `MvcApplication.Controllers`, которое не соответствует ни одной области контроллеров.



C#

```
routes.MapRoute(
  "Default",
  "{controller}/{action}/{id}",
  new { controller = "Home", action = "Index", id = "" },
  null,
  new[] { "MvcApplication.Controllers" }
);
```

Сниппет кода Global.asax.cs



VB

```
routes.MapRoute(_
  "Default",
  "{controller}/{action}/{id}", _
  New With {.controller = "Home", .action = "Index", .id = ""}, _
  Nothing, _
  New String() {"MvcApplication.Controllers"} _
)
```

Сниппет кода Global.asax.vb



Область `AreaRegistrationContext` автоматически включает в себя пространство имен области, если оно используется для указания маршрута, поэтому необходимо лишь указать пространства имен для глобальных маршрутов.

Подтверждение корректности модели

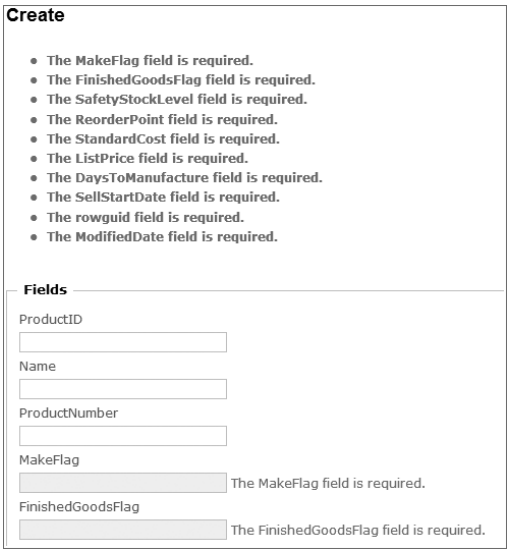


Рис. 21.9

Средство привязки модели может не только создавать и изменять модели, но и проверять их корректность. Результат этой проверки хранится в свойстве `ModelState`. Средства привязки моделей могут обнаруживать некоторые ошибки по умолчанию, особенно если речь идет о неправильных типах. На рис. 21.9 показан результат попытки сохранить объект класса `Product` с пустой формой. Большинство ошибок, выявляемых при проверке корректности моделей, основаны на факте, что их свойства являются необнуляемыми типами и должны иметь ненулевые значения.

Пользовательский интерфейс для этой ошибки обеспечивается вызовом метода `Html.ValidationSummary` из представления. Этот вспомогательный метод проверяет свойство `ModelState` и, если есть ошибки, визуализирует их в виде списка, сопровождаемого сообщением.

В свойство класса модели можно добавить дополнительные подсказки, касающиеся проверки корректности, пометив их с помощью атрибутов в сборке `System.ComponentModel.DataAnnotations`. Поскольку класс `Product` создан с помощью компонента LINQ to SQL, его нельзя обновлять непосредственно. Классы, генерируемые компонентом LINQ to SQL, определяются как частичные, поэтому их можно расширять, но к их генерируемым свойствам нелегко присоединять метаданные. Вместо этого необходимо создать *прокси-класс метаданных* (metadata proxy) со свойствами, которые вы собираетесь пометить, снабдить их правильными атрибутами, содержащими аннотацию данных, а затем пометить частный класс атрибутом `MetadataTypeAttribute`, идентифицирующим прокси-класс. Эта технология демонстрируется в следующем сниппете, в котором в класс `Product` добавляются метаданные о корректности модели.



C#

```
[MetadataType(typeof(ProductValidationMetadata))]
public partial class Product
{
}

public class ProductValidationMetadata
```

```
{
    [Required, StringLength(256)]
    public string Name { get; set; }

    [Range(0, 100)]
    public int DaysToManufacture { get; set; }
}
```

Сниппет кода Models\Product.cs



VB

```
Imports System.ComponentModel.DataAnnotations

<MetadataType(GetType(ProductMetaData))>
Partial Public Class Product

End Class

Public Class ProductMetaData
    <Required(), StringLength(256)>
    Property Name As String

    <Range(0, 100)>
    Property DaysToManufacture As Integer
End Class
```

Сниппет кода Models\Product.vb

Теперь при попытке создать новый объект класса `Product` без имени и с отрицательным числом в поле `Days to Manufacture` возникнет ошибка, показанная на рис. 21.10.

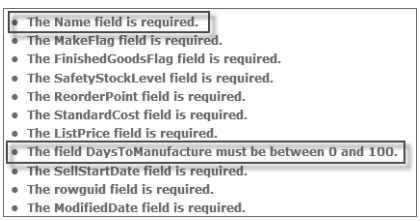


Рис. 21.10



Обратите внимание на то, что в верхней части экрана выводится отчет об ошибках, а поля ввода, соответствующие неправильным полям, окрашиваются в красный цвет и в конце ставится звездочка. Первый эффект вызывается вспомогательным методом `Html.TextBox`, принимающим значение свойства, к которому он присоединен. Если в состоянии модели для этого свойства обнаружена ошибка, этот метод добавляет CSS-класс `input-validation-error` в визуализируемый элемент управления INPUT. По умолчанию в листе стилей определен красный фон. Второй эффект вызывается вспомогательным методом `Html.ValidationMessage`. Он также ассоциирован со свойством и визуализирует содержимое своего второго параметра, если обнаруживает, что в присоединенном свойстве возникла ошибка, имеющая к нему отношение.

Частичные представления

Иногда у разработчиков возникает желание повторно использовать большие фрагменты разметки пользовательского интерфейса. На платформе ASP.NET MVC повторно используемый раздел представления называется частичным представлением. Частичные представления действуют почти как обычные представления, за исключением того, что они имеют расширение `.ascx` и являются производными от класса `System.Web.Mvc.ViewUserControl`. Для того чтобы создать частное представление, необходимо установить флажок Create a Partial View в том же самом диалоговом окне Add View, которое используется для создания других представлений.

Для визуализации частичного представления можно использовать метод `Html.RenderPartial`. Наиболее распространенный перегруженный вариант этого метода принимает имя представления и объект модели. Как и в случае обычного представления, частичное представление может быть привязано к конкретному контроллеру или быть совместно используемым. После визуализации частичного представления его HTML-разметка вставляется в основное представление. Этот сниппет кода визуализирует частичное представление "`Form`" для текущей модели.

C#

```
<% Html.RenderPartial("Form", Model); %>
```

VB

```
<% Html.RenderPartial("Form", Model) %>
```



Частное представление можно вызвать непосредственно из метода действия, используя обычный метод `View`. При этом в ответ HTML вставляется только разметка HTML, визуализированная частным представлением. Это может быть очень полезным, если данные возвращаются библиотеке jQuery.

Пользовательские шаблоны представлений

Когда пользователь использует диалоговое окно Add View, чтобы добавить элементы в свой проект, система Visual Studio 2010, по существу, выполняет шаблон T4, который определяет генерируемый код. Какой именно шаблон будет выполнен, определяется в раскрывающемся списке View Content. По желанию этот список можно отредактировать. Его можно найти в папке `C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\ItemTemplates\language\Web\MVC\v2\CodeTemplates\AddView`, где вместо слова `language` стоит либо `Csharp`, либо `VisualBasic`.

Непосредственное редактирование этих файлов приведет к изменениям всех проектов MVC на локальном компьютере, но этот метод нельзя применять для создания проектных шаблонов. Кроме того, очень трудно осуществлять эффективный контроль версий этих файлов из остальной части приложения. Обойти эти камни преткновения можно, создав папку `CodeTemplates\AddView` в корневой папке проекта и скопировав туда шаблоны T4, как показано на рис. 21.11. Когда окно Add View будет заполнять список View Content, оно будет использовать эти шаблоны вместо глобальных.

Кроме того, добавив в эту папку шаблонные файлы T4, можно создать свои собственные шаблоны. На рис. 21.11 показан шаблонный файл `MyViewTemplate.tt`, а на рис. 21.12 — представление "`MyViewTemplate`", демонстрирующее раскрывающийся список.

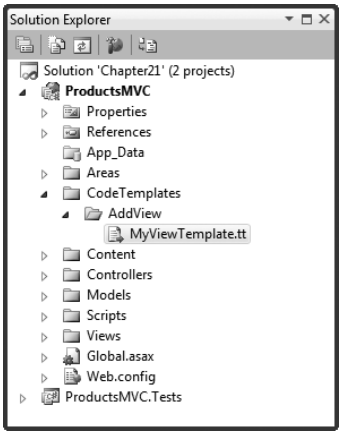


Рис. 21.11

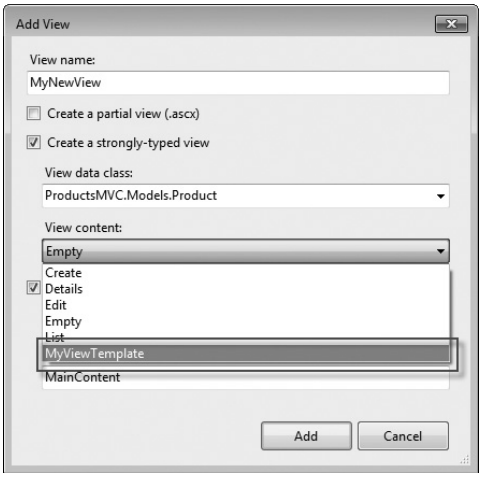


Рис. 21.12

Пользовательские шаблоны могут учитывать специфику хоста. Объект `Host` имеет тип `MvcTextTemplateHost`, содержащий все свойства, определенные в диалоговом окне Add View. Более подробную информацию о создании шаблонов T4 см. в главе 14.

Шаблоны Dynamic Data

Dynamic Data — это свойство технологии ASP.NET Web Forms, позволяющее визуализировать графический интерфейс с учетом метаданных, связанных с моделью. Несмотря на то что платформа ASP.NET MVC не интегрирована со свойством Dynamic Data, много новых функциональных возможностей в платформе ASP.NET MVC 2.0 по духу очень похожи на него. Шаблоны в платформе ASP.NET MVC 2.0 способны визуализировать части модели разными способами, как маленькие и простые, такие как отдельное строковое свойство, так и крупные и сложные, такие как класс продуктов. Представление шаблонов обеспечивается с помощью вспомогательных методов `Html`. Существуют шаблоны для отображения информации и шаблоны для ее редактирования.

Шаблоны визуализации

Представление `Details`, созданное диалоговым окном `Add View`, содержит код, визуализирующий каждое свойство. Как выглядит разметка для этих целей, показано ниже.

C#

```
<p>
  ProductID:
  <%= Html.Encode(Model.ProductID) %>
</p>
<p>
  Name:
  <%= Html.Encode(Model.Name) %>
</p>
```

VB

```
<p>
  ProductID:
```

```
  <%= Html.Encode(Model.ProductID) %>
</p>
<p>
  Name:
  <%= Html.Encode(Model.Name) %>
</p>
```

Используя новые возможности шаблонов, этот код можно изменить следующим образом.

C#

```
<p>
  <%= Html.LabelFor(x => x.ProductID) %>
  <%= Html.DisplayFor(x => x.ProductID) %>
</p>
<p>
  <%= Html.LabelFor(x => x.Name) %>
  <%= Html.DisplayFor(x => x.Name) %>
</p>
```

VB

```
<p>
  <%: Html.LabelFor(Function(x As ProductsMVC.Product) x.ProductID)%>
  <%: Html.DisplayFor(Function(x As ProductsMVC.Product) x.ProductID) %>
</p>
<p>
  <%: Html.LabelFor(Function(x As ProductsMVC.Product) x.Name)%>
  <%: Html.DisplayFor(Function(x As ProductsMVC.Product) x.Name) %>
</p>
```

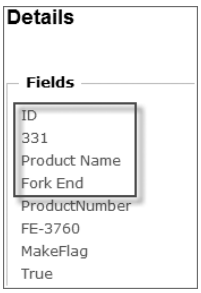


Рис. 21.13

Такой подход имеет массу явных преимуществ. Во-первых, метку больше не трудно кодировать в представлении. Поскольку метка теперь является строго типизированной, она будет обновлена, если разработчик решит переработать свой класс модели. Кроме этого, разработчик может применить атрибут `System.ComponentModel.DisplayName` к своей модели (или к модели прокси метаданных), чтобы изменить текст, который выводится на экран. Это позволяет сохранить согласованность всего приложения. В следующем сниппете кода показан прокси-класс метаданных для класса `Product` с атрибутами `DisplayNameAttribute`, а на рис. 21.13 продемонстрирована визуализация результата.

C#

```
public class ProductValidationMetadata
{
    [DisplayName("ID")]
    public int ProductID { get; set; }

    [Required, StringLength(256)]
    [DisplayName("Product Name")]
    public string Name { get; set; }

    [Range(0, 100)]
    public int DaysToManufacture { get; set; }
}
```

VB

```
Public Class ProductMetaData
    <DisplayName("ID")>
    Property ProductID As Integer
```

```
    <Required(), StringLength(256)> _
    <DisplayName("Product Name")>
    Property Name As String

    <Range(0, 100)>
    Property DaysToManufacture As Integer
End Class
```

Вспомогательный метод `DisplayFor` также обеспечивает массу возможностей. Он выбирает шаблон, основываясь на типе свойства, которое он демонстрирует на экране. Разработчик может заменить каждое из представлений, связанных с типом, создав частичное представление, имя которого совпадает с названием типа в папке `Shared\DisplayTemplates`. На рис. 21.14 показан шаблон `String`, а на рис. 21.15 — вывод результатов.

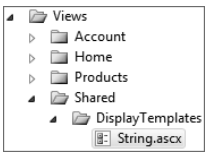


Рис. 21.14

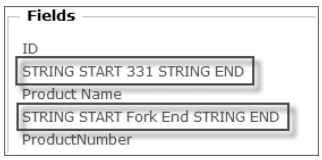


Рис. 21.15

C#

```
<%@ Control Language="C#" Inherits="System.Web.Mvc.ViewUserControl" %>

STRING START
<%= Html.Encode(ViewData.TemplateInfo.FormattedModelValue) %>
STRING END
```

VB

```
<%@ Control Language="VB" Inherits="System.Web.Mvc.ViewUserControl" %>

STRING START
<%= Html.Encode(ViewData.TemplateInfo.FormattedModelValue) %>
STRING END
```



Можно также создать шаблоны, связанные с контроллерами, поместив их в папке нижнего уровня, `DisplayTemplates`, в соответствующей папке Views.

Несмотря на то что шаблон визуализации по умолчанию выбирается на основе типа свойства, разработчик может замещать его, либо передавая имя шаблона вспомогательному методу `DisplayFor`, либо применяя к свойству метод `System.ComponentModel.DataAnnotations.UIHintAttribute`. Этот атрибут принимает строку, которая идентифицирует тип используемого шаблона. Когда платформе требуется визуализировать свойство, она пытается найти шаблон визуализации, описанный в системе подсказок пользовательского интерфейса. Если он не найден, то платформа будет искать шаблон, связанный с типом. Если и этот шаблон не будет найден, будет выполнен метод, предусмотренный по умолчанию.

Если просто применять методы `LabelFor` и `DisplayFor` к каждому свойству модели, то можно использовать вспомогательный метод `Html.DisplayForModel`. Этот метод визуализирует метку и шаблон визуализации для каждого свойства в классе модели. Визуализацию свойства с помощью этого вспомогательного метода можно от-

менить, указав, что свойство System.ComponentModel.DataAnnotations.ScaffoldColumnAttribute передает ему значение false.



Если необходимо изменить способ визуализации для метода DisplayForModel, можно создать для него шаблон, связанный с типом. Если требуется вообще изменить способ визуализации, необходимо создать шаблон визуализации Object.

Существует множество доступных для использования встроенных шаблонов визуализации. Следует только помнить, что, желая изменить поведение одного из них, придется переделывать его с самого начала.

- String. Никаких сюрпризов. Просто визуализирует содержимое самой строки. Впрочем, этот шаблон кодирует значение свойства на языке HTML.
- Html. То же, что и шаблон String, но без кодирования на языке HTML. Это самая простая форма визуализации, которая вообще существует. Этот шаблон следует применять с осторожностью, потому что он представляет собой мишень для злонамеренного встраивания кода, например с помощью атак Cross Site Scripting Attacks (XSS).
- EmailAddress. Визуализирует адрес электронной почты в виде "mailto: ссылка".
- Url. Визуализирует указатель URL в виде якоря HTML.
- HiddenInput. Не визуализирует свойство вообще, пока свойство ViewData.ModelMetaData.HideSurroundingHtml имеет значение false.
- Decimal. Визуализирует свойство с двумя цифрами после десятичной точки.
- Boolean. Визуализирует флажок, предназначенный только для чтения, для ненулевых значений и раскрывающиеся списки, предназначенные только для чтения, с вариантами True, False и Not Set для обнуляемых свойств.
- Object. Визуализирует сложные объекты и нулевые значения.

Шаблоны редактирования

Вероятно, нет ничего удивительного в том, что существуют соответствующие вспомогательные методы EditorFor и EditorForModel, определяющие способ визуализации свойств и объектов, предназначенных для редактирования. Шаблоны редактирования могут быть замещены частичными представлениями в папке EditTemplates. Шаблоны редактирования могут использоваться той же системой подсказок пользовательского интерфейса, которая использует шаблоны визуализации. Так же как и в случае шаблонов визуализации, разработчик может использовать многочисленные встроенные шаблоны.

- String. Визуализирует стандартное поле ввода, вначале заполненное значением, если оно было задано, и имеющее имя свойства. Это позволяет средствам привязки модели правильно его использовать для повторного создания объекта.
- Password. Аналогичен шаблону String, но визуализирует ввод строки HTML PASSWORD вместо поля ввода.

- `MultilineText`. Создает многострочное поле ввода. Задать количество строк и столбцов непосредственно невозможно. Предполагается, что для этого используется таблица CSS.
- `HiddenInput`. Аналогичен шаблону визуализации, визуализирует строку ввода HTML `HIDDEN`.
- `Decimal`. Аналогичен шаблону визуализации, но визуализирует поле ввода для редактирования значений.
- `Boolean`. Если тип свойства является необнуляемым, то этот шаблон визуализирует флажок. Если тип свойства является обнуляемым, этот шаблон визуализирует раскрывающийся список, содержащий те же три элемента, что и в шаблоне визуализации.
- `Object`. Визуализирует сложные редакторы.



Бред Вильсон (Brad Wilson) разместил хороший, многотомный и содержательный учебник по шаблонам ASP.NET MVC 2.0 Templates в своем блоге по адресу `http://bradwilson.typepad.com/blog/2009/10/aspnet-mvc-2-templates-part-1-introduction.html`.

Библиотека jQuery

Библиотека jQuery представляет собой открытый каркас на языке JavaScript, по умолчанию содержащий платформу ASP.NET MVC. Основным элементом библиотеки jQuery является функция `$()`. Эта функция может получать DOM-элемент языка JavaScript или строку, описывающую элементы с помощью переключателя CSS. Функция `$()` возвращает jQuery-объект, имеющий множество функций, влияющих на ее элементы. Большинство этих функций также возвращают такие же jQuery-объекты, поэтому из них можно создавать цепочки. Например, следующий сниппет кода выбирает все дескрипторы `H2` и добавляет в конец каждого из них слово “section”.

JavaScript

```
$("h2").append(" section");
```

Для того чтобы использовать библиотеку jQuery, необходимо создать ссылку на библиотеку jQuery в папке `/Scripts`, добавив в раздел заголовка своей страницы следующий код.

HTML

```
<script type="text/javascript" src="/Scripts/jquery-1.3.2.js"> </script>
```

С помощью библиотеки jQuery можно создать HTTP-запрос, используя методы `$.get` и `$.post`. Эти методы принимают указатель URL и могут по необходимости иметь функции обратного вызова. Ниже приведены представления, визуализирующие время в двух дескрипторах `div`: `server` и `client` соответственно. Кроме того, в представлении существует кнопка `Update`, после щелчка на которой создается `GET`-запрос на указатель `/time`. Получив результат, она обновляет значение, выведенное в клиентском дескрипторе `div`, но не изменяет его в серверном дескрипторе. Кроме того, она использует функции `slideUp` и `slideDown` для анимации времени в клиентском пользовательском интерфейсе.



C#

```
<%@ Page Language="C#" Inherits="System.Web.Mvc.ViewPage<System.String>" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Index</title>
  <script type="text/javascript" src="/Scripts/jquery-1.3.2.js"></script>
  <script type="text/javascript">
    $(document).ready(function () {
      $('#updater').click(UpdateNow);
    });

    function UpdateNow() {
      $.get('/time', function (data) {
        $('#clientTime').slideUp('fast', function () {
          $('#clientTime').empty().append(data).slideDown();
        });
     });
    }
  </script>
</head>
<body>
     <div>
     <h2>
           Server</h2>
     <div id="serverTime">
           <%:Model %></div>
     <h2>
           Client</h2>
     <div id="clientTime">
           <%:Model %></div>
      <input type="button" value="Update" id="updater" />
   </div>
</body>
</html>
```

Сниппет кода Views\Time\Index.aspx

Рассмотрим метод, управляющий предыдущим представлением. Он использует метод расширения `IsAjaxRequest`, чтобы определить, поступил ли запрос от библиотеки jQuery. Если это так, он возвращает время в виде строки, в противном случае возвращает полное представление.



C#

```
public ActionResult Index()
{
  var now = DateTime.Now.ToLongTimeString();
  if (Request.IsAjaxRequest())
    return Content(now);
return View(now as object);
}
```

Сниппет кода Controllers\Time.cs



VB

```
Function Index() As ActionResult
  Dim timeNow = Now.ToString()
  If Request.IsAjaxRequest() Then
    Return Content(timeNow)
  End If
```

```
    Return View(CType(timeNow, Object))
End Function
```

Сниппет кода Controllers\Time.vb

Библиотека jQuery представляет собой средство программирования "толстых" клиентов. У него много приверженцев и масса модулей plug-in. Более подробную информацию о библиотеке jQuery, включая множество учебников и демонстрационных модулей, можно найти на сайте `http://jquery.com`.

Резюме

Платформа ASP.NET MVC облегчает создание интенсивно тестируемых и слабо связанных веб-приложений, воплощающих идеи языка HTTP. Версия 2.0 позволяет достичь высокой производительности и допускает интеграцию шаблонов и системы Visual Studio. Более полную информацию о платформе ASP.NET MVC можно найти на веб-сайте `http://asp.net/mvc`.

22

Технология Silverlight

В ЭТОЙ ГЛАВЕ...

- Создание первого приложения по технологии Silverlight
- Использование средства навигации Navigation Framework
- Задание темы для приложения по технологии Silverlight
- Запуск приложения по технологии Silverlight вне браузера

Несмотря на то что технология Silverlight появилась относительно недавно, благодаря своему огромному потенциалу она получила широкое признание среди пользователей продукции компании Microsoft и разработчиков. Новые версии этой технологии выпускаются очень регулярно (между выпусками второй и третьей версий прошло только девять месяцев), что свидетельствует о быстром прогрессе. В момент написания книги была выпущена четвертая версия платформы Silverlight, которая продемонстрировала большую зрелость технологии.

В прошлом настройка системы Visual Studio 2008 для разработки приложения по технологии Silverlight представляла собой довольно нудное занятие, для которого требовалось предварительно инсталлировать пакеты Service Pack 1 и Silverlight. Система Visual Studio 2010 поставляется уже настроенной для разработки приложений по технологии Silverlight, что значительно облегчает начало работы. Кроме того, система Visual Studio 2008 не имела визуального проектировщика для создания пользовательских интерфейсов по технологии Silverlight (в первых версиях была возможность для предварительного просмотра, но позднее от нее отказались), поэтому пользователь должен писать код на языке XAML и запускать свои приложения для просмотра результатов или использовать настройку Expression Blend, если есть такая возможность (в ней есть визуальный проектировщик). Этот недостаток в системе Visual Studio 2010 был исправлен. В новой версии существует визуальный проектировщик, облегчающий работу разработчиков пользовательских интерфейсов по технологии Silverlight.

Поскольку платформа Silverlight, по существу, является частью платформы Windows Presentation Foundation (WPF), многие функциональные возможности системы Visual Studio, относящиеся к технологии WPF, описанные в главе 18, применимы и к технологии Silverlight, так что мы не будем повторяться. Разумеется, технология Silverlight не обладает такой способностью к взаимодействию, как технология Windows (поэтому она запускается в ограниченной среде ("песочнице") и не использует полностью платформу .NET Framework), но другие возможности системы, относящиеся к разработке приложений по технологии WPF, также можно использовать для разработки приложений по технологии Silverlight. В этой главе мы рассмотрим функциональные возможности системы Visual Studio, которые являются специфичными для технологии Silverlight, но неприменимы к платформе WPF.

Что такое технология Silverlight

Начиная разработку приложения по технологии Silverlight, легко заметить ее сходство с технологией WPF. Обе технологии основаны на использовании языка XAML для определения слоя представления и очень похожи друг на друга по способу работы. Однако они значительно отличаются по своему предназначению. Платформа Silverlight рассматривается как усеченный вариант платформы WPF, разработанный для развертывания через веб и предназначенный для запуска в веб-браузере. Такое приложение называется *многофункциональным интернет-приложением* (Rich Internet Application — RIA). С другой стороны, платформа WPF предназначена для разработки многофункциональных клиентских приложений (настольных). Можно отметить, что приложения, созданные по технологии WPF, могут компилироваться в приложения XBAP (XAML Browser Application) и развертываться точно так же, как и приложения, разработанные по технологии Silverlight, но для этого необходимо, чтобы на клиентском компьютере была инсталлирована платформа .NET Framework, причем приложение может быть запущено только в системе Windows, — все это к приложениям Silverlight не относится.

Одним из основных преимуществ технологии Silverlight является то, что она не требует, чтобы на клиентском компьютере была инсталлирована платформа .NET Framework (если она не инсталлирована, то придется загружать довольно большой пакет). В то же время система выполнения платформы Silverlight относительно невелика (около 5 Мбайт) и инсталлируется как сменный модуль браузера. Если пользователь просматривает веб-страницу, содержимое которой создано по технологии Silverlight, а на клиентском компьютере система выполнения платформы Silverlight не инсталлирована, то пользователь получит предложение загрузить и инсталлировать ее. Если пользователь согласен, то инсталлирование происходит автоматически и после этого открывается приложение Silverlight. При таком маленьком объеме загрузки выполняемой системы инсталляция сменного модуля платформы Silverlight и запуск приложения Silverlight занимает не более двух минут. Это значительно упрощает развертывание пользовательского приложения. Хотя платформа Silverlight еще не достигла ровня популярности платформы Adobe Flash, ее база инсталляции постоянно расширяется и вскоре сравняется с базой платформы Flash.

Одно из преимуществ приложений Silverlight (и приложений RIA в целом) над приложениями ASP.NET состоит в том, что они позволяют писать многофункциональные приложения, которые выполняются только на клиентском компьютере и взаимодействуют с сервером только тогда, когда это необходимо (обычно при посылке или получении данных). По существу, пользователь может писать веб-приложения

почти так же, как обычные настольные приложения. В частности, он может писать код на языках C# и VB.NET, выполняемый на клиентском компьютере. Это позволяет повторно использовать программы и не изучать новые языки (такие как JavaScript).

Другое большое преимущество платформы Silverlight заключается в том, что ее приложения выполняются на всех основных веб-браузерах и, что наиболее важно, как в системе Mac, так и в Windows. Это позволяет очень легко создавать кроссплатформенные приложения, которые могут выполняться на разных браузерах. Поддержка для системы Linux обеспечивается технологией Moonlight (разработанной подразделением Mono компании Novell), хотя ее качество несколько уступает версиям, поставляемым компанией Microsoft. Это значит, что технология Silverlight может оказаться идеальным средством для создания кроссплатформенных приложений, развертываемых с помощью веб. Приложения Silverlight на разных браузерах выглядят одинаково, поэтому пользователи могут не беспокоиться о том, как будет выглядеть приложение на другом браузере.

К недостаткам платформы Silverlight относится то, что она является лишь подмножеством платформы .NET Framework. Это позволило минимизировать размер загружаемого модуля и в то же время заставило выполнять приложения в ограниченной среде, чтобы предотвратить доступ к клиентскому компьютеру (что хорошо для безопасности, но сужает сферу применения технологии). Существуют определенные критерии, позволяющие сделать выбор между платформами WPF и Silverlight. Если пользователь выбрал платформу Silverlight, он должен быть готов к ограничениям.

И наконец, можно отменить, что приложения Silverlight представляют собой мостик между многофункциональными клиентскими приложениями и веб-приложениями, вобравший в себя лучшие свойства обоих.

Начало работы с платформой Silverlight

Система Visual Studio 2010 изначально содержит основные компоненты, необходимые для разработки приложений по технологии Silverlight. Платформа Silverlight полностью поддерживается системой Visual Studio 2010, но при выпуске новой версии Silverlight, возможно, понадобится загрузить пакет нового инструментария. Лучше всего загружать все требуемые компоненты с веб-сайта `http://www.silverlight.net/getstarted`.

Создайте новый проект и выберите категорию Silverlight (рис. 22.1). Перед нами откроется широкий выбор шаблонов для платформы Silverlight.

Приложение Silverlight application, по существу, представляет собой пустую форму, которая служит основой для разработки проекта (например, простого гаджета). В то же время приложение Silverlight Navigation Application имеет намного более сложную структуру, позволяющую создавать приложения. Оно содержит несколько экранов, пользовательский интерфейс и несколько простых представлений. Проектный шаблон Silverlight Class Library генерирует точно такие же классы, как и стандартный проектный шаблон Class Library, но предназначен для системы выполнения платформы Silverlight, а не для полной платформы .NET Framework.

Для демонстрационного проекта мы будем использовать шаблон Silverlight Navigation Application, поскольку он создает хорошую базу для подготовки к настоящей работе. При создании проекта перед пользователем открывается экран мастера шаблонов, позволяющий задать конфигурацию проекта (рис. 22.2).

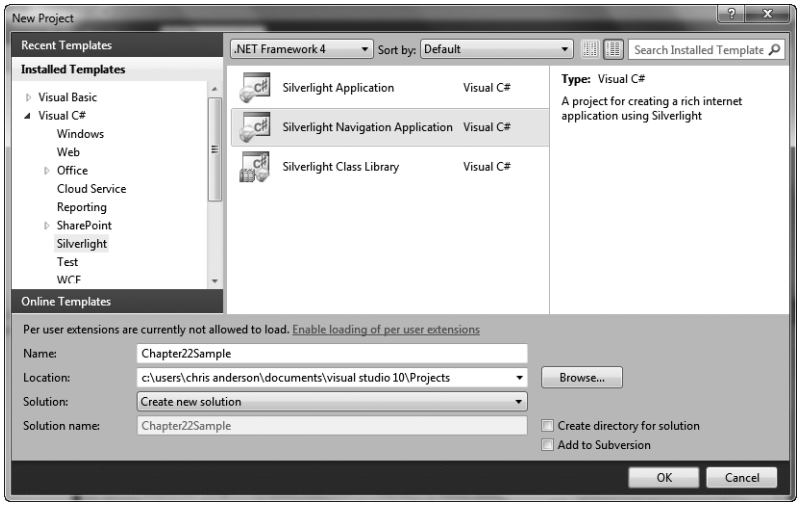


Рис. 22.1

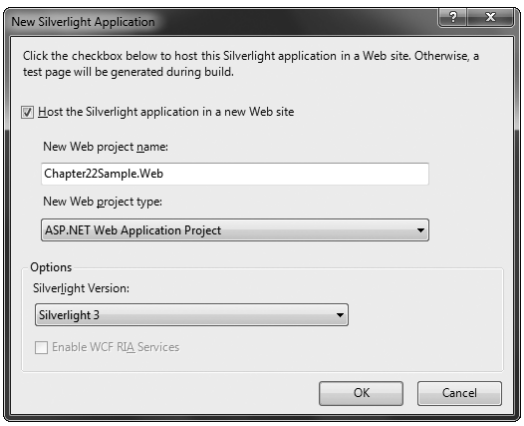


Рис. 22.2

Большинство возможностей, представленных в этом окне, предназначены для задания конфигурации веб-проекта, который будет сгенерирован в том же решении, что и проект для платформы Silverlight. Доступ к приложениям Silverlight в основном обеспечивается через веб-браузер, поэтому они должны быть внедрены в веб-страницы. Следовательно, необходимо иметь отдельный веб-проект со страницей, которая будет содержать приложение Silverlight в браузере.

Поскольку мастер генерирует веб-проект для внедрения в него приложения Silverlight, установите флажок Host the Silverlight application in the new Web. Если проект Silverlight добавляется в решение с уже существующим веб-проектом, в который будет внедряться приложение, можете сбросить этот флажок и вручную задать связи проекта среди свойств проекта (для приложения Silverlight). По умолчанию веб-проект получает заранее заданное имя в поле ввода New Web Project Name, но его можно изменить по желанию. В последнюю очередь пользователю предлагается выбрать тип веб-проекта. Существуют три варианта.

- ASP.NET Web Application Project.
- ASP.NET Web Site Project.
- ASP.NET MVC Web Project.

Выбор каждого из этих типов зависит от намерений пользователя и совершенно не влияет на проект Silverlight. В демонстрационном приложении мы использовали проект ASP.NET Web Application Project, но выбор проекта в итоге зависит от того, как пользователь собирается разрабатывать веб-сайт, в который будет внедрено приложение.

В группе Options доступны настройки, относящиеся собственно к приложению Silverlight. Раскрывающийся список Silverlight Version позволяет выбрать версию платформы Silverlight, для которой предназначается проект. Версии, перечисленные в списке, зависят от того, какой именно пакет Silverlight SDK инсталлирован на компьютере. По умолчанию предполагается, что на компьютере установлена новейшая версия инструментального пакета. Поскольку средства RIA Services обсуждаются в главе 35, будем пока игнорировать флажок Enable WCF RIA Services и оставим его сброшенным.



Свойства проекта в группе Options можно изменить позднее с помощью страниц свойств проекта Silverlight (рис. 22.4).

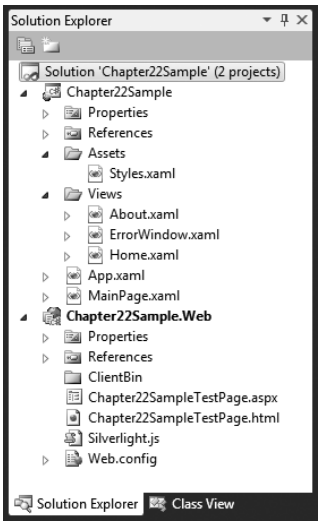


Рис. 22.3

Рассмотрим структуру сгенерированного решения (рис. 22.3). Как указывалось ранее, существуют два проекта: проект Silverlight и отдельный веб-проект, в который будет внедряться приложение Silverlight. Веб-проект активизируется при запуске решения, поскольку именно он на самом деле открывается в браузере, а затем загружает приложение Silverlight.

Веб-проект связан с проектом Silverlight, поэтому, после того, как приложение Silverlight будет скомпилировано (т.е. создан файл с расширением `.xap`), оно автоматически будет скопировано в веб-проект (в папку ClientBin), где станет доступным через веб-браузер. Скомпилируем решение, и в результате в папке ClientBin появится файл с расширением `.xap`.

Веб-проект содержит две разные страницы, которые можно использовать для внедрения приложения Silverlight: стандартную HTML-страницу и ASPX-страницу. Обе эти страницы делают одно и то же, поэтому пользователь может выбрать одну из них и удалить другую.

Теперь, посмотрев на проект Silverlight, мы видим файлы `App.xaml` и `MainPage.xam`, очень напоминающие первоначальную структуру проекта WPF. Файл `MainPage.xaml` будет заполнять окно браузера, демонстрировать заголовок и кнопки для навигации по приложению, а также содержать разные представления в элементе управления Frame. Таким образом, файл `MainPage.xaml` можно рассматривать как оболочку для содержимого в приложении.

Проектный шаблон по умолчанию содержит два представления содержимого: Home и About. Вопросы, связанные с модификацией и добавлением новых представлений, будут рассматриваться в следующем разделе. Эта папка также содержит файл

ErrorWindow.xaml, описывающий окно, которое наследуется у элемента управления ChildWindow (который по существу представляет собой модальное диалоговое окно в платформе Silverlight) и появляется, если возникает необработанное исключение (необработанное исключение обрабатывается в коде, лежащем в основе файла App.xaml и демонстрирующем этот элемент управления).

Папка Assets содержит файл Styles.xaml, в котором хранятся тематические стили, используемые приложением. Они обсуждаются ниже, в разделе "Темы".

Теперь посмотрим, какие настройки доступны на страницах свойств проекта Silverlight. Страница свойств Silverlight является уникальной для приложений Silverlight (см. рис. 22.4).

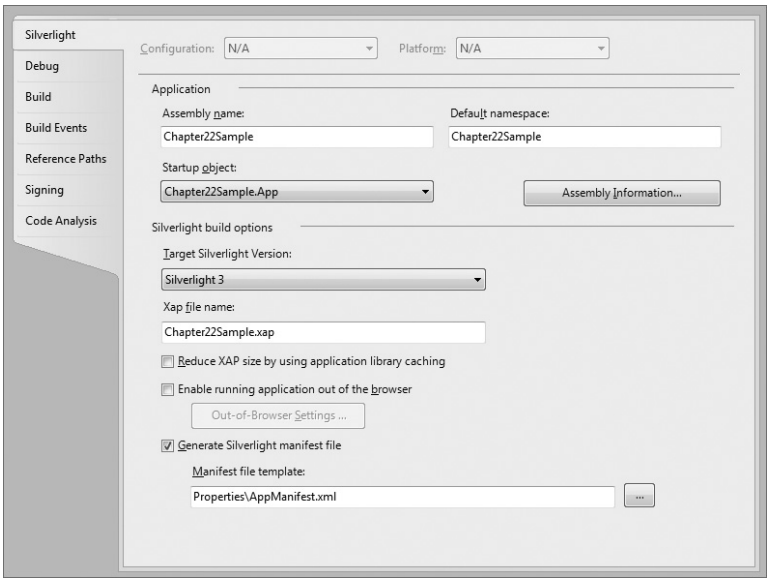


Рис. 22.4

Многие свойства на этой странице представляют особый интерес. Настройка имени файла с расширением xap позволяет задать имя файла с расширением .xap, который будет скомпилирован в проекте Silverlight и во всех связанных с ним модулях (библиотечные сборки, сборки элементов управления и т.д.). Файл .xap представляет собой zip-файл с разными расширениями. Для того чтобы просмотреть его содержимое, нужен архиватор. Если проект несложен (т.е. создается с помощью шаблона Silverlight Application и не содержит ссылок на библиотеки элементов управления), то, скорее всего, он будет содержать только сборку и манифестный файл. Если же в нем есть ссылки на другие сборки (например, если используется элемент управления DataGrid, принадлежащий динамической библиотеке System.Windows.Controls.Data.dll), то файл с расширением .xap резко увеличится в размерах (поскольку эти сборки уже содержатся в других файлах с расширением .xap). Это значит, что каждый раз, когда разработчик будет вносить даже самое незначительное изменение в свой проект и разворачивать его, пользователи будут вынуждены повторно загружать сборки (например, сборку, содержащую элемент управления DataGrid), так как они не изменяются только потому, что снова включаются в файл с расширением .xap. К счастью, существует возможность улучшить этот сценарий и кэ-

ширование библиотеки приложений. Эту возможность очень легко подключить, установив флажок Reduce XAP size by using application library caching. При следующей компиляции проекта используемые сборки будут отделены друг от друга в разных файлах и загружены по отдельности из файла xap-приложения.

Сборки, подлежащие кэшированию, должны иметь XML-карту, которая включается в файл с расширением .xap и ссылается на zip-файл, содержащий сборку. Большинство элементов управления от компании Microsoft уже имеют такую XML-карту, поэтому пользователь может об этом не беспокоиться. Теперь скомпилируем проект снова и заглянем в папку ClientBin, относящуюся к веб-проекту. Там мы найдем один или несколько файлов с расширением .zip — по одному для каждой внешней сборки, на которую ссылается проект Silverlight и которые не включаются в систему выполнения платформы Silverlight. Файл с расширением .xap теперь будет намного меньше, поскольку он больше не содержит сборки. При первом выполнении приложения все требуемые части будут загружены через веб. После обновления проекта и компиляции повторно загружать нужно будет только файл с расширением .xap. При этом используется более узкая полоса пропускания канала связи между сервером и клиентом (так как загружаемые модификации будут намного меньше), и обновление происходит намного быстрее, т.е. пользователь меньше ждет, пока сможет повторно использовать свое приложение.

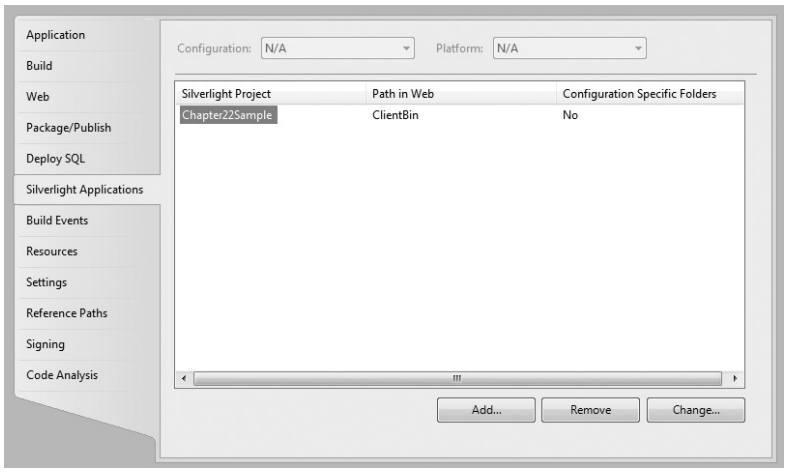


Рис. 22.5



К сожалению, кэширование библиотеки приложений невозможно применить в приложениях, которые настроены в режиме Out Of Browser (детали этого режима описаны далее в главе), поскольку он требует, чтобы все сборки находились в файле с расширением .xap. При попытке использовать обе возможности на экране появляется предупреждение.

Посмотрим, как связываются проект Silverlight и веб-проект. Связью между проектами управляет веб-проект, причем ее можно настраивать с его страницы свойств. Откроем свойства этого проекта и выберем корешок закладки Silverlight Applications, чтобы увидеть проекты Silverlight, которые в данный момент связаны с веб-проектом (рис. 22.5).

Если в веб-проект внедрено несколько приложений Silverlight, то, скорее всего, разработчику придется использовать только страницу свойств или добавлять проект Silverlight в существующее решение, уже содержащее веб-проект. Связи между проектами можно только добавлять или удалять (но не модифицировать), поэтому страница свойств обычно используется только тогда, когда проект Silverlight добавляется в решение или удаляется из него.

Эта страница свойств отображает в виде списка существующие связи между веб-проектом и проектами Silverlight, включенными в решение. Имеются три возможности: можно добавить другую ссылку на проект Silverlight, удалить ссылку на проект или изменить ссылку на проект (впрочем, как мы вскоре убедимся, это изменение представляет собой не то, что можно было ожидать).

Щелкните на кнопке Add, чтобы установить связь веб-проекта с другим проектом Silverlight. На рис. 22.6 показано окно, которое используется для настройки новой связи.

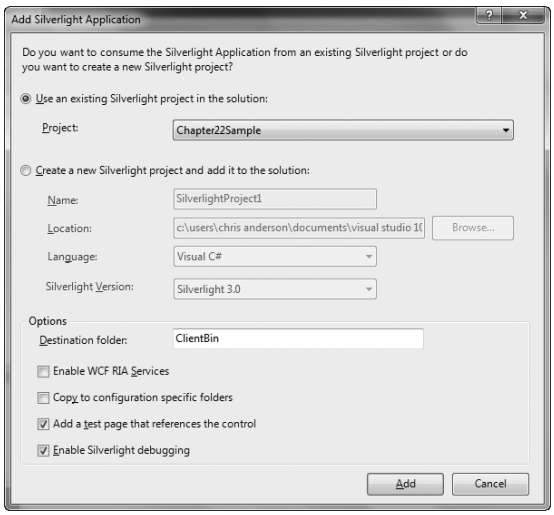


Рис. 22.6

При добавлении ссылки на проект Silverlight пользователь имеет две возможности. Во-первых, можно установить связь с проектом Silverlight, который уже существует в решении, где его можно просто выбрать из списка. Кроме того, можно создать новый проект Silverlight и автоматически установить его связь с текущим веб-проектом. К сожалению, пользователь не может выбрать шаблон проекта, поэтому генерируется только новый проект, основанный на шаблоне Silverlight Application, с некоторыми ограничениями, касающимися его использования.

Текстовое поле Destination Folder позволяет указать папку в веб-проекте, в которую при компиляции будет скопирован проект Silverlight. Именно на эту папку будут ссылаться генерируемые тестовые страницы (если они будут выбраны для создания), в которые должно внедряться приложение Silverlight.

Если в окне установлен флажок Copy to configuration specific folders, то приложение Silverlight не будет скопировано непосредственно в указанную папку. Вместо этого в ней будет создана дополнительная папка с именем текущей конфигурации (Debug, Release и т.д.), и приложение Silverlight будет скопировано в нее. Если этот флажок установлен, то тестовые страницы будут по-прежнему ссылаться на исходную целевую папку, а не на папку нижнего уровня с именем текущей конфигурации, в которой будет находиться приложение Silverlight. Если пользователь хочет использовать эту возможность, то он должен вручную изменить тестовые страницы, чтобы они ссылались на путь, существующий в текущей конфигурации, и обновлять его каждый раз, когда происходит переключение настроек. По умолчанию этот флажок сброшен, и, возможно, лучше его не использовать без крайней необходимости.

Установив флажок Add a test page that references the control option, пользователь добавляет в веб-проект страницы HTML и ASPX, уже настроенные для внедрения проекта Silverlight, с которым устанавливается связь (из этих страниц используется только одна, другую можно удалить).

Флажок Enable Silverlight debugging включает возможность выполнять отладку приложения Silverlight (т.е. останавливаться в точках прерывания, выполнять код шаг за шагом и т.д.). Обратной стороной этой возможности является то, что она отключает отладку JavaScript для веб-проекта, поскольку выполнять отладку обоих проектов одновременно невозможно.

Вернемся к списку проектов Silverlight, с которыми устанавливается связь (рис. 22.5). Кнопка Remove удаляет связь (этого и следовало ожидать), но кнопка Change означает не то, что можно было бы предположить. Она используется для переключения между использованием и отказом от использования папок конфигурации (описанных выше).

Итак, изучив структуру проекта, можно попытаться выполнить его. Проектный шаблон Silverlight Navigation Application представляет собой хорошую отправную точку для создания пользовательского приложения и может стать основой для создания приложения (как показано на рис. 22.7).



Рис. 22.7

Средство навигации Navigation Framework

Поскольку мы использовали для своего проекта шаблон Silverlight Navigation Application, мы должны сделать беглый обзор механизма навигации платформы Silverlight под названием Navigation Framework. Механизм Navigation Framework появился в версии Silverlight 3 и упростил создание приложений с несколькими представлениями и переходами между ними. Файл `MainPage.xaml` содержит элемент управления Frame (часть механизма Navigation Framework), который используется для хранения отдельных представлений, подлежащих демонстрации.

Для того чтобы их можно было внедрить во фрейм, представления должны быть производными от элемента управления Page. Если заглянуть в файл `Home.xaml`, то можно заметить, что корневым элементом является `navigation:Page`, а не `UserControl`. Для того чтобы создать новое представление, щелкните правой кнопкой мыши на папке Views и выберите команду Add⇨New Item. Затем выберите шаблонный элемент Silverlight Page, дайте ему имя (например, `Test.xaml`) и щелкните на кнопке OK. По необходимости представление можно наполнить содержимым.

Для каждого представления необходим указатель URI, который ссылался бы на него. Этот указатель URI используется, когда пользователь хочет перейти на конкретное представление. В соответствующем файле представления пользователь может за-

дать отображение выбранного указателя URI в соответствующий путь (внутри проекта). Эти отображения определяются свойством `UriMapper` в элементе управления Frame (в файле `MainPage.xaml`). Они допускают использование шаблонных символов. Отображение для шаблонных символов создано заранее и позволяет просто использовать имя XAML-файла (не указывая расширение `.xaml`). При этом выполняется поиск XAML-файла с указанным именем и расширением `.xaml` в папке `Views`. Это значит, что пользователь не обязан задавать это отображение, если хочет искать файл `Test.xaml`, используя строку `/Test` в качестве указателя URI.

Теперь необходимо добавить кнопку, позволяющую переходить к новому представлению. Элементы управления HyperlinkButton (с именами Link1 и Link2) можно найти в файле `MainPage.xaml`. Скопируйте один из них и вставьте под ним новую строку (скопировав в файл существующий элемент, можно создать любой другой разделяющий элемент). Измените свойство NavigateUri так, чтобы оно отображалось на требуемое представление (в данном случае оно должно быть равным `/Test`), дайте элементу управления новое имя и введите текст, который должен появиться на кнопке (в свойстве Content). Запустите проект. В области заголовка приложения появится новая кнопка, щелкнув на которой можно перейти к новому представлению.



Обратите внимание на то, как изменяется закладка на указателе URL (часть после символа # в указателе URL в адресной строке браузера) при переключении между страницами. Для перехода вперед и назад между представлениями, которые пользователь уже просматривал, можно использовать кнопки браузера Back и Next. Кроме того, можно использовать глубокое связывание, при котором представление имеет уникальный указатель URL, который открывается автоматически. Все эти функциональные возможности предоставляются механизмом Navigation Framework.

Темы

Как и платформа WPF, платформа Silverlight имеет множество стилей и тем, хотя их модели стилей реализованы немного по-разному. В технологии Silverlight используется механизм Visual State Manager (VSM), которого сначала не было в технологии WPF (до версии WPF 4). Этот механизм позволяет явно определять контракт элемента управления, регламентирующий его функциональные возможности (т.е. код), и его вид (т.е. файл XAML). Это позволяет осуществить строгое разделение между ними. Этот контракт определяет модель для шаблонов элементов управления, Parts and States, состоящую из частей, состояний, переходов и групп состояний. Дальнейшее обсуждение этой модели выходит за рамки главы, однако отметим, что в технологии Silverlight этой моделью управляет механизм VSM. Данный механизм управления стилями считается намного лучше, чем оригинальный метод триггеров, использованный в технологии WPF. По этой причине механизм VSM был встроен в версию WPF 4. Однако до появления версии Silverlight 4 технология Silverlight не поддерживала неявную стилизацию (в отличие от технологии WPF, которая это делала), с помощью которой можно было бы указать, что все элементы управления должны использовать определенный стиль. Это затрудняло применение тем для проекта. Для того чтобы определение тем стало более простым, компания Microsoft создала элемент управления ImplicitStyleManager, поставляемый в библиотеке Silverlight Toolkit. В результате

версия Silverlight 4 все же включила возможность неявной стилизации, что сделало элемент управления ImplicitStyleManager в некотором смысле излишним, но в нашей книге мы по-прежнему его используем, чтобы продемонстрировать определение тем в разных версиях платформы Silverlight.



Библиотеку Silverlight Toolkit можно свободно загрузить в веб-сайта CodePlex: `http://silverlight.codeplex.com`. Она также содержит многочисленные полезные элементы управления, не включенные в пакет Silverlight SDK (например, диаграммы, закладки, TreeView и т.д.).

Несмотря на различия между ними, технологии WPF и Silverlight имеют элементы управления, имеющие похожие стили и темы. Рассмотрим теперь применение другой темы к проекту, чтобы полностью изменить внешний вид его элементов управления. Платформа Silverlight имеет темы, перечисленные в главе 18 (фактически эти темы сначала были разработаны для платформы Silverlight, а затем перенесены на платформу WPF). Их можно найти в пакете Silverlight Toolkit. Назовем их *темами элементов управления* (control themes), чтобы отличать их от *тем приложения* (application themes), которые мы вскоре обсудим.

Существует множество способов использования тем элементов управления. Например, можно взять тематические XAML-файлы из пакета Silverlight Toolkit, скопировать их в корневую папку своего проекта и включить их в проект (одновременно установив их свойство Build Action равным значению Content). Для нашего примера мы использовали тематический файл `System.Windows.Controls.Theming.ExpressionDark.xaml`. Теперь добавим ссылку на библиотеку `System.Windows.Controls.Theming.Toolkit.dll` из пакета Silverlight Toolkit (позволяющую использовать элемент управления ImplicitStyleManager). К сожалению, мы не можем задать тему на уровне приложения, потому что элемент управления ImplicitStyleManager имеет определенные ограничения, наложенные технологией Silverlight. Следовательно, в корневых элементах всех представлений, которые должны иметь новую тему, необходимо добавить следующий префикс пространства имен и задать значения свойств.

```
xmlsn:theming="clr-namespace:System.Windows.Controls.Theming; assembly=
System.Windows.Controls.Theming.Toolkit"

theming:ImplicitStyleManager.ResourceDictionaryUri=
"System.Windows.Controls.Theming.ExpressionDark.xaml"
theming:ImplicitStyleManager.ApplyMode="OneTime"
```

Скомпилировав и выполнив наш проект, мы убедимся, что все элементы управления в представлениях, для которых задана новая тема, выглядят иначе и соответствуют теме из указанного файла.



Проект может не скомпилироваться из-за путаницы со ссылками на тематический файл. Поскольку тематический файл содержит стили для многих элементов управления, он ссылается на сборки, содержащие эти элементы управления. Пользователь может либо добавить ссылку на требуемую сборку, либо удалить из тематического файла стили для указанных элементов управления (а также связанные с ними определения префиксов пространств имен).

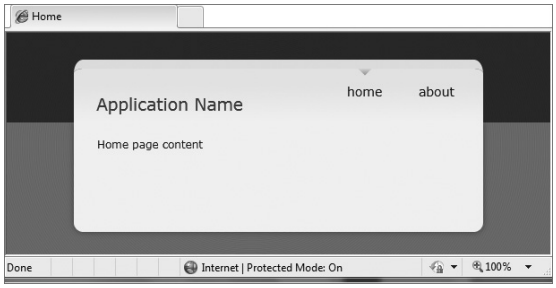


Рис. 22.8

Если создать проект на основе шаблона Silverlight Navigation Application или Silverlight Business Application, то можно использовать альтернативные *темы приложения*, определяющие совершенно новый внешний вид приложения в целом. Стили тем приложений можно найти в файле `Styles.xaml` в папке Assets проекта Silverlight. Если проект создан на основе шаблона Silverlight Navigation Application, то в файл `App.xaml` добавляются стили из файла `Styles.xaml`. Для того чтобы определить компоновку и внешний вид основной страницы, в файле `MainPage.xaml` используются стили, определенные в файле `Styles.xaml`. Следовательно, чтобы полностью изменить внешний вид приложения, достаточно заменить этот файл файлом с теми же стилями, но с другими значениями. Компания Microsoft и сообщество разработчиков создали множество тем приложений, основанных на проектном шаблоне Silverlight Navigation Application. Их можно загрузить с вебсайта `http://gallery.expression.microsoft.com` (см. категорию Themes). Например, заменив файл `Styles.xaml` для проекта, показанного на рис. 22.7, тематическим файлом из галереи "Frosted Cinnamon Toast", можно полностью изменить внешний вид приложения (рис. 22.8).

Режим запуска вне браузера

Несмотря на то что платформа Silverlight изначально была спроектирована как сменный модуль для браузера, в версии Silverlight 3 появилась возможность выполнять приложения Silverlight за пределами браузера. В этом случае больше нет необходимости запускать приложение Silverlight в браузере. По существу, нет даже необходимости работать с приложением Silverlight в интерактивном режиме, если оно было инсталлировано в режиме Out-Of-Browser. Такие приложения сначала доставляются с помощью браузера, а затем инсталлируются на компьютере (если это допускается разработчиком). Процесс инсталляции можно запустить из контекстного меню, которое открывается после щелчка правой кнопкой мыши, или из кода. Единственное условие, которое обязательно должно быть выполнено, — процесс должен инициироваться пользователем (чтобы случайные приложения не могли самостоятельно инсталлироваться на компьютере пользователя без его ведома).

По умолчанию приложение Silverlight не настроено на режим Out-Of-Browser, и пользователь должен явно установить это свойство приложения. Проще всего это сделать, используя свойства проекта для приложения Silverlight, как показано на рис. 22.4. Если установить флажок Enable Running the Application Out of the Browser, то появится кнопка Out-of-Browser Settings, после щелчка на которой откроется окно, показанное на рис. 22.9



Рис. 22.9

Это окно позволяет задавать множество настроек для режима Out-Of-Browser. Большинство этих настроек понятно без дополнительных объяснений. Пользователь может задать заголовок окна и его первоначальные размеры (напомним, что размеры окна можно изменять). Можно также задать комбинации клавиш для начального меню, названия этих комбинаций, текст, который должен появиться на экране, когда курсор мыши задерживается над пиктограммой (описание приложения), а также пиктограммы переменного размера для использования в качестве сокращенной команды. Эти пиктограммы должны быть представлены в виде PNG-файлов и заранее добавлены в проект Silverlight. Выберите соответствующее изображение для каждого размера пиктограммы. Если оставить одну из пиктограмм пустой, то она будет использоваться как пиктограмма режима Out-Of-Browser, заданная по умолчанию. Два флажка, расположенные внизу окна, Use GNU Acceleration и Show install menu, позволяют указать, должен ли режим Out-Of-Browser использовать ускорение графического пользовательского интерфейса (для приложений Silverlight запуск в браузере автоматически означает, что приложение устанавливается в виде сменного модуля), и может ли пользователь инсталлировать приложение с помощью контекстного меню (в противном случае процесс инсталляции должен активизироваться из кода).



При запуске вне браузера обратите внимание на то, что приложение Silverlight по-прежнему выполняется в ограниченной среде ("песочнице") и не имеет больше прав доступа к пользовательскому компьютеру, чем приложение, которое запускается в браузере. Несмотря на то что приложение Silverlight вне браузера запускается как обычное стандартное приложение, оно по-прежнему ограничено определенной моделью безопасности, как если бы было запущено в барузере. Однако в версии Silverlight 4 появилась возможность давать приложениям в режиме Out-Of-Browser все больше привилегий, включая автоматизацию элементов COM и доступ к локальным файлам.

Задав установки режима Out-Of-Browser, можно запустить проект на выполнение, затем щелкнуть правой кнопкой мыши и выбрать команду Install XXXX onto your computer, как показано на рис. 22.10, чтобы инициировать процесс инсталляции (здесь XXXX — имя приложения).

Окно, показанное на рис. 22.11, позволяет пользователю выбирать типы комбинаций для установки приложения. Оно инсталлирует приложение локально (в профиле пользователя), настраивает выбранные комбинации клавиш и автоматически запускает приложение в режиме Out-Of-Browser.

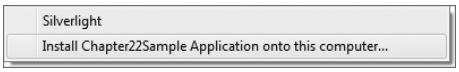


Рис. 22.10

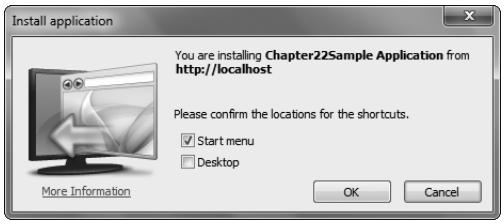


Рис. 22.11



Для того чтобы деинсталлировть приложение, достаточно щелкнуть правой кнопкой мыши и выбрать команду Remove this Application.

Разумеется, в какой-то момент придется обновить приложение и заново инсталлировать его на пользовательских компьютерах. К счастью, это очень просто сделать, но для этого придется писать программу. Этот код можно использовать в любом месте приложения, но мы поместим его в файл `App.xaml` и начнем обновление приложения.

VB

```
Private Sub Application_Startup(ByVal o As Object, ByVal e As StartupEventArgs) _
                                                              Handles Me.Startup
     Me.RootVisual = New MainPage()

     If Application.Current.IsRunningOutOfBrowser Then
         Application.Current.CheckAndDownloadUpdateAsync()
     End If
End Sub

Private Sub App_CheckAndDownloadUpdateCompleted(ByVal sender As Object, _
                            ByVal e As _

System.Windows.CheckAndDownloadUpdateCompletedEventArgs) _
                            Handles Me.CheckAndDownloadUpdateCompleted
    If e.UpdateAvailable Then
       MessageBox.Show("A new version of this application is available and " &
                       "has been downloaded. Please close the application and " &
                       "restart it to use the new version.",
                       "Application Update Found", MessageBoxButton.OK)
    End If
End Sub
```

C#

```
private void Application_Startup(object sender, StartupEventArgs e)
{
    this.RootVisual = new Page();

    if (Application.Current.IsRunningOutOfBrowser)
    {
        Application.Current.CheckAndDownloadUpdateCompleted +=
             Current_CheckAndDownloadUpdateCompleted;
        Application.Current.CheckAndDownloadUpdateAsync();
```

```
    }
}

private void Current_CheckAndDownloadUpdateCompleted(object sender,
                    CheckAndDownloadUpdateCompletedEventArgs e)
{
    if (e.UpdateAvailable)
    {
        MessageBox.Show("A new version of this application is available and " +
            "has been downloaded. Please close the application and restart " +
            "it to use the new version.", "Application Update Found",
            MessageBoxButton.OK);
    }
}
```

Легко видеть, что если приложение запускается в режиме Out-Of-Browser, необходимо проверить, существуют ли его обновления. Это приводит к асинхронному возвращению к указателю URL, ссылающемуся на компьютер, с которого инсталлировалось приложение, и проверке, есть ли новая версия (одновременно продолжается загрузка и выполнение приложения). Если есть новая версия, она автоматически загружается. Независимо от того, есть ли новая версия или нет, при выполнении проверки обновления возникает событие `CheckAndDownloadUpdateCompleted` (и происходит потенциальная загрузка новой версии). Остается только проверить, найдено ли обновление, и уведомить об этом пользователя. При следующем запуске приложения будет выполнена автоматическая инсталляция обновления, поэтому при запуске новой версии пользователь должен закрыть приложение и открыть его снова.

Для того чтобы выполнить тестирование процесса обновления приложения, начнем с включения кода проверки в свое приложение. Запустите приложение и инсталлируйте его так, как описано выше. Закройте его и экземпляр приложения, запущенный в браузере, и вернитесь к системе Visual Studio. Внесите в приложение изменения (чтобы заметить, изменилось ли приложение) и перекомпилируйте его. Запустите ранее инсталлированную версию (из меню Start или пиктограммы на рабочем столе). Приложение запустится, и вскоре появится сообщение о том, что была загружена новая версия и приложение должно быть выполнено заново. Когда вы откроете приложение снова, вы должны заметить, что запущена действительно новая версия.

Резюме

В этой главе мы показали, как в системе Visual Studio 2010 создавать приложение с помощью технологии Silverlight и запускать его в браузере и за его пределами. Более подробная информация о взаимодействии клиента и сервера изложена в главе 35.

23

Механизм Dynamic Data

В ЭТОЙ ГЛАВЕ...

- Создание веб-приложения, управляемого данными, с помощью скаффолдинга, обеспечиваемого механизмом Dynamic Data
- Настройка модели данных и слоя представления в приложении Dynamic Data
- Добавление возможностей механизма Dynamic Data в существующее веб-приложение

Большинство разработчиков проводят чрезвычайно много времени, создавая код для обработки данных. Из-за того, что занятие это настолько важное и настолько рутинное, для такого кода появилась специальная аббревиатура — *CRUD* (*Create, Read, Update, Delete* — создать, прочитать, обновить, удалить), т.е. четыре основные операции над данными.

Например, рассмотрим простое приложение для работы со списком `Tasks` или `To Do`. Это приложение должно по меньшей мере обеспечивать следующие функциональные возможности.

- `Create`. Создавать новые задания и сохранять их в базе данных.
- `Read`. Извлекать список заданий из базы данных и выводить их на экран. Извлекать и выводить на экран все свойство отдельного задания.
- `Update`. Модифицировать свойства существующего задания и сохранять изменения в базе данных.
- `Delete`. Удалять ненужные задачи из базы данных.

Многие рутинные задачи, необходимые для написания CRUD-кода, решает механизм ASP.NET Dynamic Data. Он может выявить модель данных и автоматически сгенерировать полнофункциональный веб-сайт, ориентированный на обработку данных. Это позволяет разработчику сосредоточиться на бизнес-процессе, усовершенствовании интерфейса приложения или решении сложных и важных задач.

ЛУЧШЕ МЕНЬШЕ, ДА ЛУЧШЕ: СКАФФОЛДИНГ И СОГЛАШЕНИЕ О КОНФИГУРАЦИИ

Скаффолдинг (*scaffolding*) — это название механизма, который используется технологией ASP.NET Dynamic Data для динамической генерации веб-страниц на основе базы данных. Генерируемые страницы имеют все функциональные возможности, которые можно ожидать от любого приличного приложения, управляемого данными, включая разбиение на страницы и сортировку. Кроме удобства, которое получает разработчик, освобождаясь от необходимости писать низкоуровневый код для доступа к данным, этот механизм обеспечивает встроенную проверку корректности, используя схему базы данных, и полную поддержку внешних ключей и отношений между таблицами.

Скаффолдинг получил популярность благодаря технологии Ruby on Rail, предназначенной для разработки веб-сайтов. Помимо скаффолдинга, технология ASP.NET Dynamic Data использует еще несколько принципов и методов, явно навеянных технологией Ruby on Rails. Один из этих принципов формулируется как "Convention over Configuration" ("Соглашение по конфигурации") и означает, что некоторые вещи предполагаются неявно по стандартному соглашению. Предположим, что на этапе выполнения приложения механизм Dynamic Data обнаружит файл `List.aspx` в папке Products. В этом случае он использует его для визуализации пользовательской веб-страницы, предназначенной для таблицы базы данных `Product`. Поскольку имя папки представляет собой имя базы данных во множественном числе, нет необходимости явно сообщать механизму Dynamic Data о том, что этот файл существует или связан с таблицей `Product`.

Чем меньше кода, тем меньше возможностей сделать ошибку.

В этой главе показано, как использовать скаффолдинг механизма Dynamic Data для создания веб-приложения, управляемого данными, с минимумом кодирования или вообще без него. Кроме того, мы покажем, насколько гибко настраивается механизм Dynamic Data на модели данных и веб-страницы.

Несмотря на то что механизмы Dynamic Data и скаффолдинга часто ассоциируют только с разработкой совершенно новых веб-приложений, управляемых данными, в конце главы мы покажем, что разработчик может получить массу преимуществ, добавив функциональные возможности механизма Dynamic Data в существующее веб-приложение.

Создание веб-приложения с помощью механизма Dynamic Data

Для того чтобы создать и запустить веб-приложение, используя механизм Dynamic Data, нам необходима база данных. В качестве примера в главе используется база данных SQL Server 2008 AdventureWorksLT, которую можно загрузить с веб-сайта CodePlex `http://msftdbprodsamples.codeplex.com/`.

Загрузив нашу базу данных, откройте интегрированную среду Visual Studio и выберите команду File⇨New⇨Project. В категории Web project, которая существует как для языка Visual Basic, так и для языка C#, есть два проектных шаблона для механизма Dynamic Data, отражающих две основные возможности для доступа к данным, поддерживаемых компанией Microsoft. Первый шаблон — Dynamic Data Linq to SQL Web Application — поддерживает способ доступа к данным LINQ to SQL. Второй шаблон — Dynamic Data Entities Web Application — поддерживает технологию ADO.NET Entity Framework.

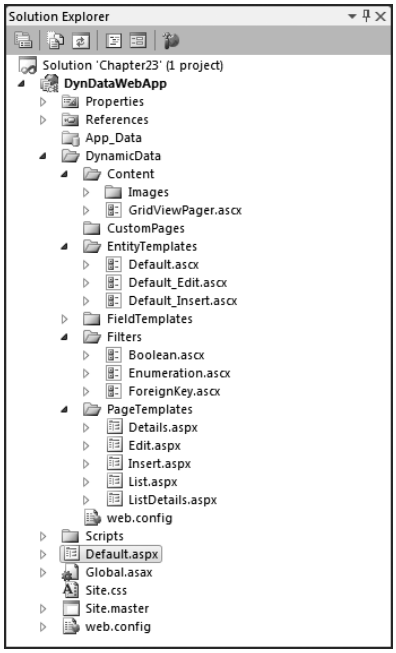


Рис. 23.1

Выберите проектный шаблон Dynamic Data Linq to SQL Web Application и щелкните на кнопке OK. Когда будет создан новый проект, он сгенерирует множество файлов и папок, показанных на рис. 23.1. Большинство этих файлов являются шаблонами, которые можно изменять, чтобы настроить пользовательский интерфейс. Они загружаются в корневую папку механизма DynamicData, и их мы рассмотрим в главе немного позднее.

Проектный шаблон также создаст стандартную веб-форму `Default.aspx`, представляющую собой стартовую страницу веб-приложения. Как и стандартный проект ASP.NET Web Application, данное приложение позволяет использовать наилучшие приемы работы, используя страницу мастера и внешний CSS-файл, а также подключает библиотеку JQuery JavaScript. Более подробно об этих функциональных возможностях см. в главе 20.



Если пользователь предпочитает работать с проектами Web Site, а не Web Application, он также может использовать механизм Dynamic Data. В диалоговом окне New Web Site есть два эквивалентных шаблона для создания проекта LINQ to SQL или Entities Dynamic Data Web Site.

Добавление модели данных

Создав новый проект, вы должны указать базу данных и создать новую модель данных. Щелкните правой кнопкой мыши на папке `App_Data` и выберите команду Add Existing Item. Затем найдите файл `AdventureWorksLT2008_Data.mdf`, загруженный ранее, и щелкните на кнопке Add.

ТЕХНОЛОГИЯ LINQ TO SQL ПРОТИВ ТЕХНОЛОГИИ ADO.NET ENTITY FRAMEWORK

Обе технологии, LINQ to SQL и ADO.NET Entity Framework, предназначены для доступа к данным и поддерживаются компанией Microsoft. У них есть свои преимущества и недостатки, и обе они превосходно работают в наиболее распространенных ситуациях.

Технология LINQ to SQL работает только с базой данных Microsoft SQL Server и поддерживает только прямое отображение отдельной таблицы базы данных в отдельный класс платформы .NET. Поскольку технология LINQ to SQL тесно связана с базами данных SQL Server, она генерирует очень эффективный код T-SQL.

С другой стороны, технология ADO.NET Entity Framework позволяет модели данных отличаться от базовой схемы базы данных. Разработчик может отображать несколько баз данных в отдельный класс платформы .NET или отдельную таблицу базы данных в несколько классов платформы .NET. Технология Entity Framework также поддерживает множество разных баз данных, включая Oracle, MySQL и DB2.

Более подробную информацию о технологии LINQ to SQL можно найти в главе 28, а технологии ADO.NET Entity Framework посвящена глава 29.

На следующем этапе необходимо создать модель данных. Щелкните правой кнопкой мыши в окне Solution Explorer и выберите команду Add⇨New Item. Выберите в категории Data компонент LINQ to SQL Classes и назовите его `AdventureWorksDM.dbml`. Если разработчик на предыдущем этапе выбрал проект Entities Dynamic Data, то следует выбрать элемент ADO.NET Entity Data Model.

Щелкнув на кнопке Add, откройте новый компонент в окне конструктора Object Relational Designer. Дважды щелкните на файле базы данных AdventureWorksLT, добавленном ранее, чтобы открыть его в окне Server Explorer, а затем разверните узел Tables. Выберите все таблицы, за исключением двух первых, и перетащите их в окно конструктора. Тем самым вы заполнили модель данных LINQ to SQL, как показано на рис. 23.2.

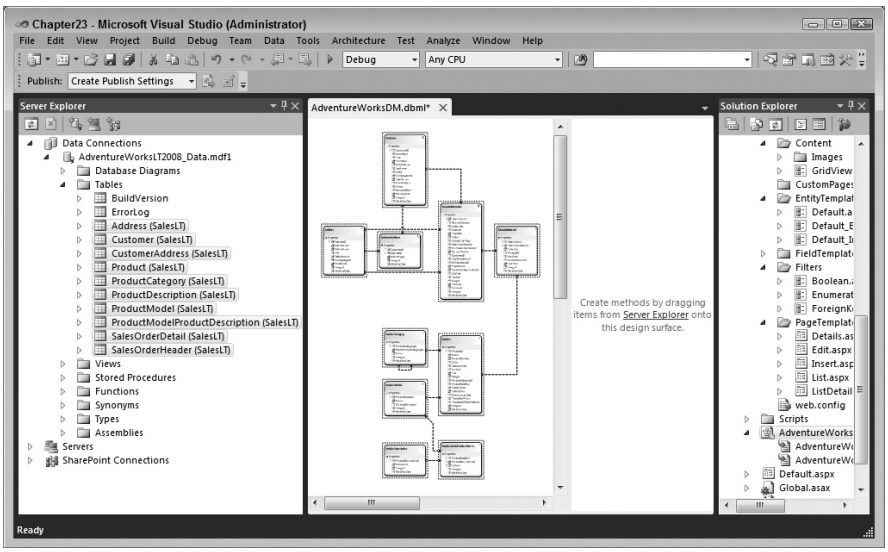


Рис. 23.2

В заключение необходимо зарегистрировать нашу модель данных в механизме Dynamic Data и включить механизм скаффолдинга. Откройте файл Global.asax.cs (или `Global.asax.vb`, если основная программа написана на языке Visual Basic) и найдите строку кода, указанную ниже. Снимите знак комментария с этой строки и измените имя `YourDataContextType` на `AdventureWorksDMDataContext`. И наконец, измените значение свойства `ScaffoldAllTables` на `true`.



C#

```
DefaultModel.RegisterContext(typeof(AdventureWorksDMDataContext),
                             new ContextConfiguration()
                             { ScaffoldAllTables = true });
```

Сниппет кода Global.asax.cs



VB

```
DefaultModel.RegisterContext(GetType(AdventureWorksDMDataContext), _
                             New ContextConfiguration() _
                             With {.ScaffoldAllTables = True})
```

Сниппет кода Global.asax.vb

Это все, что необходимо для того, чтобы получить веб-приложение, управляемое данными, с полной поддержкой операций CRUD и запустить его на выполнение.

Исследование приложения, созданного с помощью механизма Dynamic Data

При запуске приложения открывается главное окно `Default.aspx`, которое содержит список гиперссылок на все таблицы, добавленные в модель данных (рис. 23.3). Обратите внимание на то, что имена, перечисленные в этой таблице, представляют собой версии соответствующего имени таблицы во множественном числе.

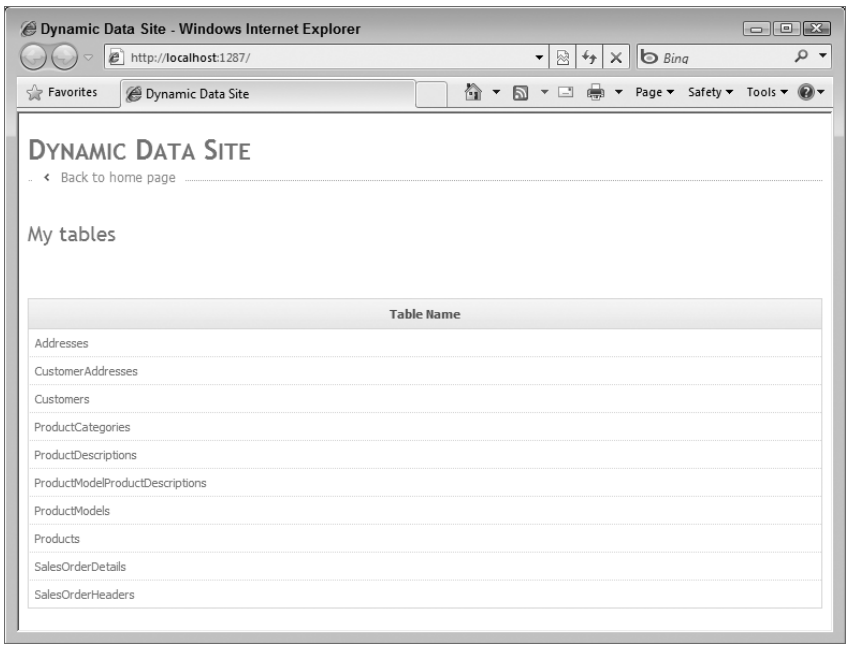


Рис. 23.3

Щелкнув на одной из этих ссылок, можно открыть страницу `List.aspx`, показанную на рис. 23.4, соответствующую выбранной таблице. Эта страница, совместно со

страницей `Details.aspx`, соответствующей отдельной записи, реализует функцию "Read" для данного CRUD-приложения и включает поддержку разделения на страницы и фильтрации записей по внешнему ключу. Эти страницы также содержат гиперссылки для представления деталей, редактирования или удаления записи. Любые внешние ключи демонстрируются как ссылки на страницы деталей, соответствующие данной записи внешнего ключа.



Легко заметить, что на веб-странице нет некоторых полей базы данных, например `ProductID` и `ThumbNailPhoto`. По умолчанию механизм Dynamic Data не выполняет скаффолдинг столбцов `Identity`, а также бинарных и вычисляемых столбцов. Однако, как будет показано далее, это обстоятельство можно преодолеть.

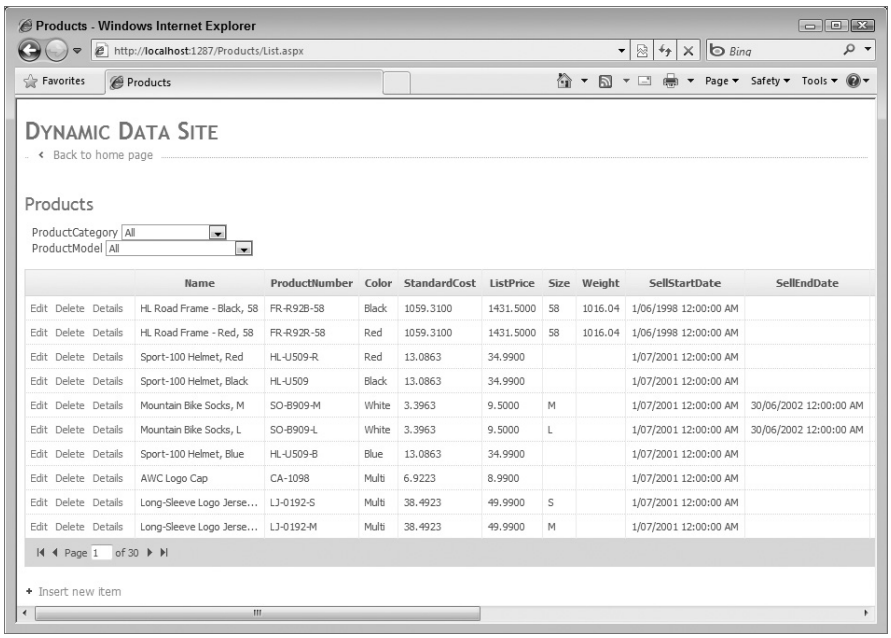


Рис. 23.4

CRUD-функция "Update" активизируется после щелчка на ссылке Edit, расположенной напротив записи. При этом открывается страница `Edit.aspx`, показанная на рис. 23.5. Легко заметить, что поля ввода имеют разную ширину, которая определяется по длине поля базы данных. Эта страница также содержит элементы управления ASP.NET для проверки корректности на основе информации, содержащейся в поле базы данных. Например, корректность поля `ProductNumber` проверяется элементом управления RequiredFieldValidator, потому что соответствующее поле данных в базе данных не может содержать нули. Аналогично поле `Weight` использует элемент управления CompareValidator, который следит за тем, чтобы введенное значение было десятичным.

Внешние ключи также обрабатываются с помощью раскрывающихся списков. Например, на рис. 23.5 поля `ProductCategory` и `ProductModel` являются внешними ключами.

Таблицы, использующие выбранную таблицу как внешний ключ, изображаются в виде гиперссылок. Примером этого служит поле `SalesOrderDetails` на рис. 23.5.

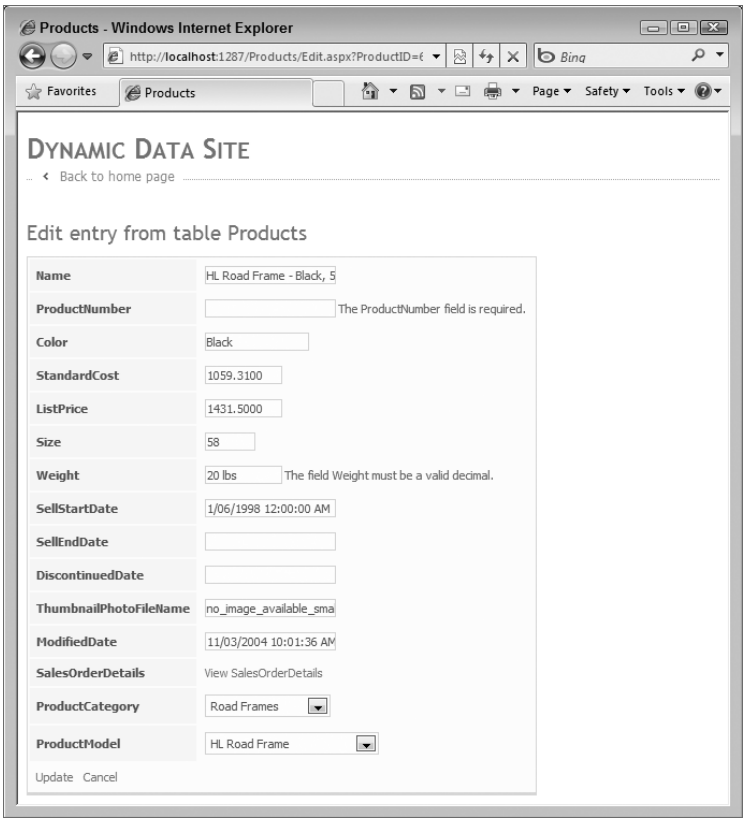


Рис. 23.5

Настройка модели данных

Несмотря на то что скаффолдинг всей базы данных выглядит впечатляюще, вряд ли в каждом приложении необходимо демонстрировать конечным пользователям каждую таблицу и поле базы данных. К счастью, механизм Dynamic Data позволяет настроить модель данных, чтобы решить эту и многие другие проблемы.

Скаффолдинг отдельных таблиц

Перед тем как настраивать модель данных, необходимо отключить автоматический скаффолдинг всех таблиц. Откройте файл `Global.asax.cs` и измените значение свойства ScaffoldAllTables на `false`.

На следующем этапе мы будем избирательно применять скаффолдинг к отдельным таблицам. Начнем с добавления нового файла классов в проект `Product.cs`. Этот класс должен быть частичным, поскольку класс `Product` уже был определен в модели данных LINQ to SQL. Для того чтобы выполнить скаффолдинг таблицы `Product`,

снабдите этот класс атрибутом `ScaffoldTable`. После этого класс должен выглядеть следующим образом.



C#

```
using System.ComponentModel.DataAnnotations;

namespace DynDataWebApp
{
    [ScaffoldTable(true)]
    public partial class Product
    {
    }
}
```

Сниппет кода Product.cs



VB

```
Imports System.ComponentModel.DataAnnotations

<ScaffoldTable(True)> _
Partial Public Class Product
End Class
```

Сниппет кода Product.vb

Если запустить приложение сейчас, то просматривать и редактировать можно будет только таблицу `Product`.



Тот же самый результат можно получить, установив значение свойства ScaffoldAllTables равным `true` и избирательно скрывая таблицы, снабжая соответствующие классы атрибутом `ScaffoldTable`, равным `false`.

Настройка отдельных полей данных

Во многих ситуациях необходимо, чтобы те или иные поля в таблице можно было только читать или вообще скрыть. Это особенно полезно, если таблица содержит важную информацию, например номер кредитной карточки.

Например, при редактировании записи в таблице `Products` приложение показывает ссылку на таблицу `SalesOrderDetails`. Эта ссылка блокирована, потому что таблица `SalesOrderDetails` недоступна для скаффолдинга. Следовательно, демонстрация этого поля не дает пользователю никакой полезной информации. Кроме того, поле `ModifiedDate`, которое может заинтересовать пользователя, не относится к полям, которые следует разрешать редактировать. Таким образом, было бы лучше показывать это поле только для чтения и разрешить базе данных модифицировать его с помощью триггера `Update`.

Эти требования выполняются механизмом Dynamic Data путем добавления в класс модели данных *класса метаданных* (metadata class). В самый конец файла `Product.cs` добавляется новый класс — `ProductMetadata`. Этот класс можно сделать ассоциированным, применив к классу `Product` атрибут `MetadataType`.

В классе `ProductMetadata` создайте открытые поля, имена которых совпадают с именами каждого из настраиваемых полей данных. Поскольку механизм Dynamic Data будет считывать тип каждого поля из класса модели данных, а не из метакласса, эти поля могут иметь тип `object`.

Добавьте атрибут ScaffoldColumn в поле SalesOrderDetails и установите его равным false, чтобы скрыть поле. Для того чтобы поле ModifiedDate стало доступным только для чтения, снабдите его атрибутом Editable, равным false .

Эти изменения отражены в следующем фрагменте кода.



C#

```
namespace DynDataWebApp
{
    [ScaffoldTable(true)]
    [MetadataType(typeof(ProductMetadata))]
    public partial class Product
    {
    }
    public class ProductMetadata
    {
        [ScaffoldColumn(false)]
        public object SalesOrderDetails;

        [Editable(false)]
        public object ModifiedDate;
    }
}
```

Сниппет кода Product.cs



VB

```
<ScaffoldTable(True)> _
<MetadataType(GetType(ProductMetadata))> _
Partial Public Class Product
End Class
Public Class ProductMetadata
    <ScaffoldColumn(False)> _
    Public SalesOrderDetails As Object

    <Editable(False)> _
    Public ModifiedDate As Object
End Class
```

Сниппет кода Product.vb

Результаты этих изменений продемонстрированы на рис. 23.6. В левой части рисунка показано исходное окно редактирования таблицы Product. В правой части рисунка показано новое окно редактирования после настройки модели данных.

Добавление правил проверки корректности

Как указывалось выше, механизм Dynamic Data содержит встроенные правила проверки корректности, которые выводятся на основе схемы базы данных. Например, для поля таблицы базы данных, которое помечено как необнуляемое, на страницу Update можно добавить элемент управления RequiredFieldValidator.

Однако в некоторых ситуациях существуют бизнес-правила, регламентирующие формат данных, которые не входят в состав встроенных правил проверки корректности. Например, в таблице Product значения, сохраняемые в поле ProductNumber, должны иметь определенный формат: запись должна начинаться с двух прописных букв, за которыми следует символ подчеркивания. Этот формат можно сделать обязательным, снабдив поле ProductNumber атрибутом RegularExpression, как показано в следующем фрагменте кода.

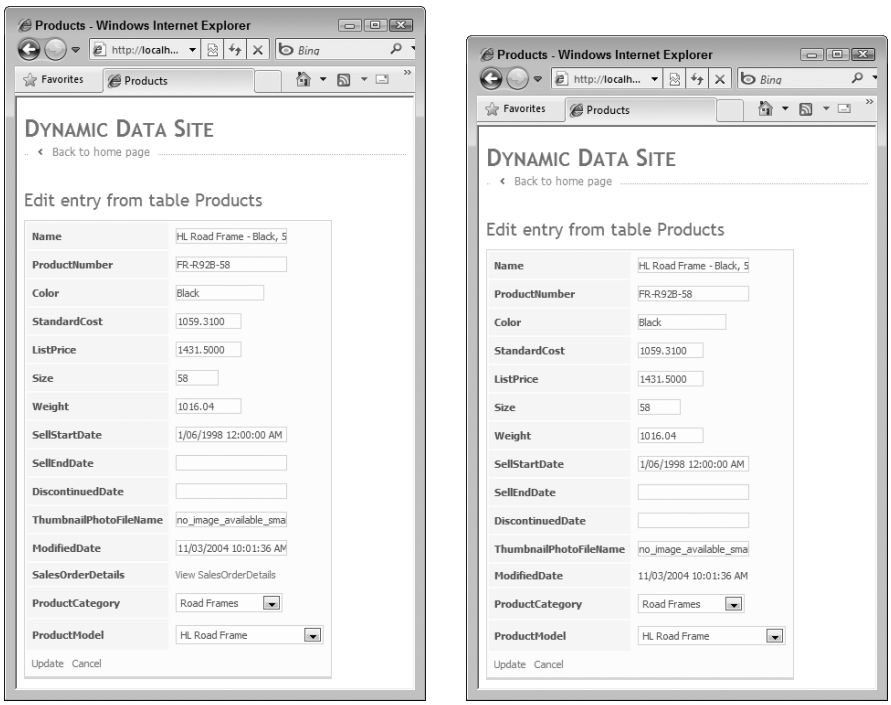


Рис. 23.6

C#

```
[ScaffoldTable(true)]
[MetadataType(typeof(ProductMetadata))]
public partial class Product
{
}

public class ProductMetadata
{
    [RegularExpression("^[A-Z]{2}-[A-Z0-9]{4}(-[A-Z0-9]{1,2})?$",
    ErrorMessage="Product Number must be a valid format")]
    public object ProductNumber;
}
```

Сниппет кода Product.cs

VB

```
<ScaffoldTable(True)> _
<MetadataType(GetType(ProductMetadata))> _
Partial Public Class Product
End Class

Public Class ProductMetadata
    <RegularExpression("^[A-Z]{2}-[A-Z0-9]{4}(-[A-Z0-9]{1,2})?$", _
    ErrorMessage:="Product Number must be a valid format")> _
    Public ProductNumber As Object
End Class
```

Сниппет кода Product.vb

Кроме того, существует атрибут `Range`, позволяющий задавать минимальное и максимальное допустимые значения для числового поля. Если необходимо установить эти ограничения на поле в модели данных, не указывая их в соответствующей базе данных, то можно применить атрибут `Required` или `StringLength`.

Несмотря на свою полезность, проверка корректности с помощью установки атрибутов не учитывает все возможные сценарии. Например, пользователь может попытаться ввести в поле SellEndDate дату, которая предшествует дате, указанной в поле SellStartDate. Из-за ограничений, наложенных на это поле, возникнет ошибка на этапе выполнения программы, а не ошибка, связанная с проверкой корректности.

Для каждого поля в модели данных технология LINQ to SQL определяет два метода, которые вызываются в процессе редактирования, — метод `On`*FieldName*`Changing`, который вызывается непосредственно перед изменением поля, и метод `On`*FieldName*`Changed`, который вызывается сразу после изменения. Для того чтобы выполнить сложные правила проверки корректности, нужно завершить частичное объявление метода в модели данных.

Следующий фрагмент кода иллюстрирует правило проверки корректности, которое гарантирует, что значение, введенное в поле SellEndDate, не превосходит значение поля SellStartDate.



C#

```
[ScaffoldTable(true)]
[MetadataType(typeof(ProductMetadata))]
public partial class Product
{
    partial void OnSellEndDateChanging(DateTime? value)
    {
        if (value.HasValue && value.Value <this._SellStartDate)
        {
            throw new ValidationException(
                     "Sell End Date must be later than Sell Start Date");
        }
    }
}
```

Сниппет кода Product.cs



VB

```
<ScaffoldTable(True)> _
<MetadataType(GetType(ProductMetadata))> _
Partial Public Class Product
    Private Sub OnSellEndDateChanging(ByVal value_
                                      As Nullable(Of DateTime))
        If value.HasValue AndAlso value.Value <Me._SellStartDate Then
            Throw New ValidationException( _
                      "Sell End Date must be later than Sell Start Date")
        End If
    End Sub
End Class
```

Сниппет кода Product.vb

На рис. 23.7 показано, как это правило устанавливается механизмом Dynamic Data.

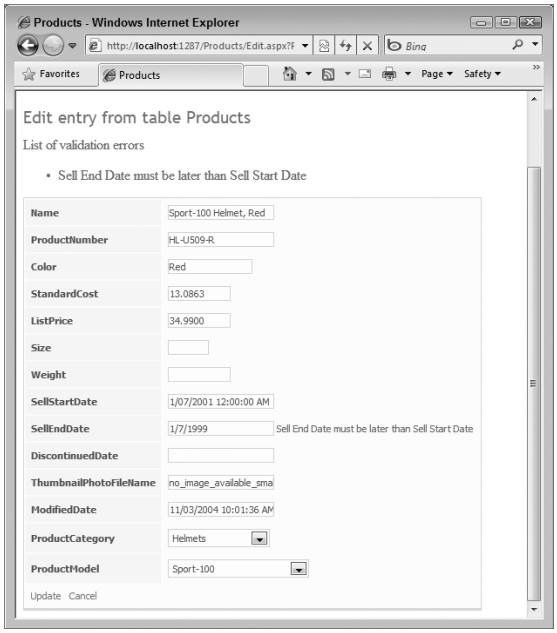


Рис. 23.7

Настройка формата представления

Формат представления, принятый для некоторых типов данных по умолчанию, не является идеальным. Например, поля StandardCost и ListPrice, использующие тип данных SQL `money`, изображаются как числа с четырьмя знаками после десятичной точки. Кроме того, поля SellStartDate и SellEndDate fields, имеющие тип данных SQL `datetime`, отформатированы так, что показывают и дату, и время, хотя часто время на самом деле является излишней информацией.

Формат представления этих ячеек можно настроить двумя способами: глобально для конкретного типа данных, настроив шаблон поля, или индивидуально для каждого поля, настроив модель данных. Настройка шаблона поля обсуждается далее в главе.

Во-первых, для того чтобы задать формат представления поля в пользовательском интерфейсе, необходимо включить в модель данных атрибут `DisplayFormat`. Этот атрибут имеет свойство `DataFormatString`, значением которого является строка форматирования .NET. Атрибут также содержит множество дополнительных параметров, позволяющих управлять визуализацией, включая параметр `HtmlEncode`, который служит индикатором того, что поле должно быть закодировано на языке, и атрибут `NullDisplayText`, задающий текст, который должен выводиться на экран, если поле содержит нуль. Применение атрибута `DisplayFormat` иллюстрирует следующий фрагмент кода.



C#

```
[DisplayFormat(DataFormatString="{0:C}")]
public object ListPrice;
```

```
[DisplayFormat(DataFormatString="{0:MMM d, yyyy}",
               NullDisplayText="Not Specified")]
public object SellEndDate;
```

Сниппет кода Product.cs



VB

```
<Display(Name:="List Price")> _
<DisplayFormat(DataFormatString:="{0:C}")> _
Public ListPrice As Object

<Display(Name:="Sell End Date")> _
<DisplayFormat(DataFormatString:="{0:MMM d, yyyy}",
               NullDisplayText:="Not Specified")> _
Public SellEndDate As Object
```

Сниппет кода Product.vb



По умолчанию формат представления может быть применен только к представлению Read. Для того чтобы применить это форматирование к представлению Edit, необходимо установить свойство `ApplyFormatInEditMode` атрибута `DisplayFormat` равным `true`.

Во-вторых, маловероятно, что разработчик захочет использовать имена полей базы данных в пользовательском интерфейсе. Намного лучше придумать описательные имена для всех полей. Атрибут `Display` позволяет управлять визуализацией меток полей. Этот атрибут принимает множество параметров, включая `Name` для задания реальной метки и `Order` для управления порядком перечисления полей. В следующем коде поле ProductNumber на экране имеет имя "Product Code" и по порядку имеет номер 1, т.е. выводится первым.



C#

```
[Display(Name="Product Code", Order=1)]
public object ProductNumber;
```

Сниппет кода Product.cs



VB

```
<Display(Name:="Product Code", Order:=1)> _
Public ProductNumber As Object
```

Сниппет кода Product.vb

На рис. 23.8 показано, как эти изменения формата визуализируются механизмом by Dynamic Data.

Настройка представления

Часто механизм Dynamic Data по умолчанию визуализирует не совсем то, что хотелось бы. В предыдущем разделе мы показали, сколько аспектов модели данных необходимо настроить, чтобы добиться желательной визуализации таблиц баз данных и полей. Однако существуют ограничения, не позволяющие решить все проблемы путем настройки модели данных. К счастью, механизм Dynamic Data имеет богатую систему шаблонов, которые легко настраиваются и позволяют полностью управлять пользовательским интерфейсом.

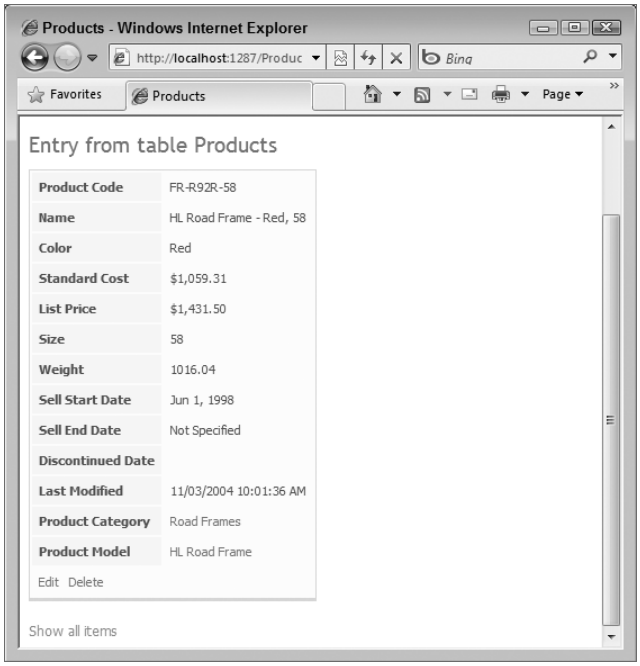


Рис. 23.8

Файлы шаблонов механизма Dynamic Data хранятся в разных папках, вложенных в папку `DynamicData`, которая является корневым узлом веб-приложения. В соответствии с принципом соглашения по конфигурации эти файлы шаблонов не нужно вручную регистрировать в механизме Dynamic Data. Вместо этого каждый тип шаблона должен храниться в своей папке, а платформа будет использовать эти папки и имена файлов шаблонов для того, чтобы определить, какой из них следует загрузить в ходе выполнения программы.

Шаблоны страниц

Шаблоны страниц используются для визуализации таблиц баз данных по умолчанию. Мастер шаблонов страниц хранится в папке `DynamicData\PageTemplates`. Механизм Dynamic Data содержит пять шаблонов страниц, предназначенных для представления и редактирования данных.

- `Details.aspx`. Визуализирует представление существующей записи таблицы, предназначенное только для чтения.
- `Edit.aspx`. Выводит на экран редактируемое представление существующей записи в таблице.
- `Insert.aspx`. Выводит на экран представление, позволяющее пользователю добавлять новую запись в таблицу.
- `List.aspx`. Визуализирует всю таблицу в виде сетки, поддерживающей разделение на страницы и сортировку.

- `ListDetails.aspx`. Используется, когда механизм Dynamic Data находится в режиме комбинированной страницы, в котором задания Detail, Edit, Insert и List выполняются на одной и той же странице. Этот режим можно включить, следуя инструкциям, записанным в файле `Global.asax`.

Для того чтобы изменить визуализацию всех таблиц по умолчанию, эти шаблоны можно модифицировать. Кроме того, можно заменить шаблоны страниц, заданные по умолчанию, множеством шаблонов для таблицы, созданных разработчиком. Эти шаблоны страниц хранятся в папке `DynamicData\CustomPages`.

Хорошим кандидатом для создания пользовательского шаблона является таблица `SalesOrderHeader` в базе данных `AdventureWorksLT`. Перед тем как создать шаблон, необходимо включить скаффолдинг для данной таблицы. Создайте новый частичный класс модели данных для таблицы SalesOrderHeader и включите скаффолдинг, как показано в следующем листинге.



C#

```
using System.ComponentModel.DataAnnotations;

namespace DynDataWebApp
{
    [ScaffoldTable(true)]
    public partial class SalesOrderHeader
    {
    }
}
```

Сниппет кода SalesOrderHeader.cs



VB

```
Imports System.ComponentModel.DataAnnotations

<ScaffoldTable(True)> _
Partial Public Class SalesOrderHeader
End Class
```

Сниппет кода SalesOrderHeader.vb

Затем создайте папку нижнего уровня с именем SalesOrderHeaders в папке `DynamicData\CustomPages`. Эта папка будет содержать пользовательские шаблоны для таблицы `SalesOrderHeader`. Скопируйте существующий шаблон `List.aspx` из папки `DynamicData\PageTemplates` в папку `DynamicData\CustomPages\SalesOrderHeaders`.



Имя папки для пользовательских шаблонов страницы обычно представляет собой имя таблицы во множественном числе. Исключение из этого правила — если модель данных использует платформу ADO.NET Entity Framework версии 3.5 или разработчик изменил положение переключателя Pluralize or Singularize Generated Object Names. В этом случае имя папки совпадет с именем таблицы.

Поскольку шаблон был скопирован, а значит, создан дубликат класса, приложение больше не будет компилироваться. Проще всего исправить эту проблему, присвоив пространству имен в файлах разметки и исходного кода какое-нибудь уникальное имя, как показано ниже.



C#

```
<%@ Page Language="C#" MasterPageFile="~/Site.master"
CodeBehind="List.aspx.cs"
    Inherits="DynDataWebApp._SalesOrderHeaders.List" %>
```

Сниппет кода DynamicData\CustomPages\SalesOrderHeaders\List.aspx

```
namespace DynDataWebApp._SalesOrderHeaders
{
    public partial class List : System.Web.UI.Page
    {
        // Code snipped
    }
}
```

Сниппет кода DynamicData\CustomPages\SalesOrderHeaders\List.aspx.cs



VB

```
<%@ Page Language="VB" MasterPageFile="~/Site.master"
CodeBehind="List.aspx.vb"
    Inherits="DynDataWebApp._SalesOrderHeader.List" %>
```

Сниппет кода DynamicData\CustomPages\SalesOrderHeaders\List.aspx

```
Namespace _SalesOrderHeader
    Class List
        Inherits Page
        ' Code Snipped
    End Class
End Namespace
```

Сниппет кода DynamicData\CustomPages\SalesOrderHeaders\List.aspx.vb

Теперь шаблон можно настраивать как угодно. Например, можно уменьшить количество столбцов в представлении List, оставив все поля данных в представлениях Insert и Edit. Эта степень настройки возможна только для шаблонов страниц, связанных с таблицами.

Внесите эти изменения, найдя элемент управления GridView в шаблоне `List.aspx`. Включите автоматическую визуализацию всех полей данных, добавив свойство `AutoGenerateColumns="False"`. Затем вручную задайте поля, которые хотите видеть на экране, добавив множество элементов управления DynamicField, как показано в следующем коде.



```
<asp:GridView ID="GridView1" runat="server" DataSourceID="GridDataSource"
              EnablePersistedSelection="True" AllowPaging="True"
              AllowSorting="True" CssClass="DDGridView"
              AutoGenerateColumns="False" RowStyle-CssClass="td"
              HeaderStyle-CssClass="th" CellPadding="6">
   <Columns>
      <asp:TemplateField>
         <ItemTemplate>
             <asp:DynamicHyperLink runat="server" Text="Details" />
         </ItemTemplate>
      </asp:TemplateField>
      <asp:DynamicField DataField="AccountNumber" HeaderText="Account No" />
      <asp:DynamicField DataField="PurchaseOrderNumber"
                        HeaderText="PO Number" />
      <asp:DynamicField DataField="OrderDate"
                        DataFormatString="{0:d-MMM-yyyy}"
                        HeaderText="Order Date" />
```

```
        <asp:DynamicField DataField="ShipDate"
                          DataFormatString="{0:d-MMM-yyyy}"
                          HeaderText="Ship Date" />
        <asp:DynamicField DataField="SubTotal" DataFormatString="{0:c}"
                          HeaderText="Sub Total" />
        <asp:DynamicField DataField="TaxAmt" DataFormatString="{0:c}"
                          HeaderText="Tax Amount" />
        <asp:DynamicField DataField="Freight" DataFormatString="{0:c}"
                          HeaderText="Freight" />
    </Columns>

    <HeaderStyle CssClass="th" />

    <PagerStyle CssClass="DDFooter"/>
    <PagerTemplate>
        <asp:GridViewPager runat="server" />
    </PagerTemplate>
    <EmptyDataTemplate>
        There are currently no items in this table.
    </EmptyDataTemplate>
    <RowStyle CssClass="td" />
</asp:GridView>
```

Сниппет кода DynamicData\CustomPages\SalesOrderHeaders\List.aspx

На рис. 23.9 показано настроенное представление List таблицы SalesOrderHeader с уменьшенным количеством столбцов.

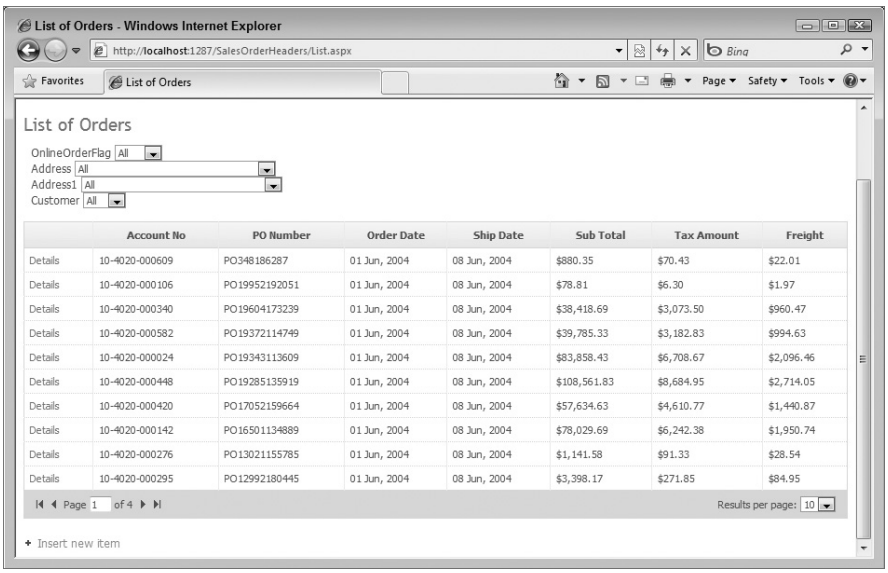


Рис. 23.9

Шаблоны полей

Шаблоны полей используются для визуализации отдельных полей данных в пользовательском интерфейсе. Они разделяются на шаблоны представления и редактирования полей. Шаблоны полей именуются в соответствии с именем типа данных. При-

чем, если шаблон предназначен для визуализации редактируемого поля, после его имени добавляется суффикс _Edit. Например, шаблон представления поля Text называется Text.ascx. Он визуализирует поле, используя элемент управления ASP.NET Literal. Соответствующий шаблон для редактирования поля называется Text_Edit.ascx. Он визуализирует поле, используя элемент управления ASP.NET TextBox. Шаблон редактирования также содержит несколько элементов управления для проверки корректности данных, что позволяет правильно выполнять визуализацию и обрабатывать любые исключения, генерируемые моделью данных.

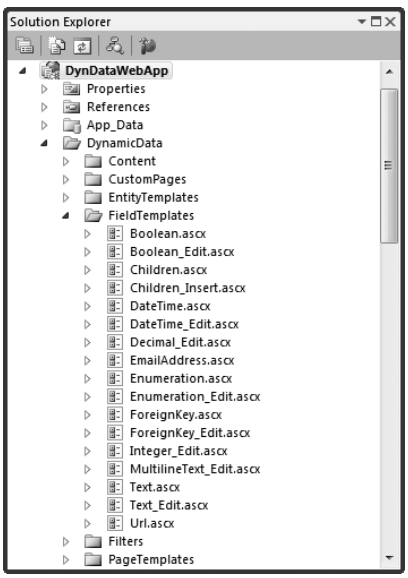


Рис. 23.10

Механизм Dynamic Data имеет много шаблонов полей, как показано на рис. 23.10. Как и в случае шаблонов страниц, шаблоны полей, заданные по умолчанию, можно настраивать или создавать новые. Все шаблоны полей, включая все новые шаблоны, которые созданы пользователем, хранятся в папке DynamicData\FieldTemplates.

Некоторые поля данных в таблице SalesOrderHeader базы данных AdventureWorksLT содержат как дату, так и время, даже если время не имеет значения. Шаблон поля DateTime в механизме Dynamic Data выводит на экран элемент управления TextBox в соответствующем представлении Edit. Если поле данных разрешает ввод только даты, а не времени, то было бы правильным выводить на экран элемент представления Calendar, а не TextBox.

Начните работу с создания копии шаблона DateTime.ascx и переименуйте его в DateCalendar.ascx. Затем откройте файлы разметки и исходный файл для шаблона DateCalendar.ascx и переименуйте класс DateTimeField в DateCalendarField, как показано ниже.



C#

```
<%@ Control Language="C#" CodeBehind="DateCalendar.ascx.cs"
            Inherits="DynDataWebApp.DateCalendarField" %>
```

Сниппет кода DynamicData\FieldTemplates\DateCalendar.ascx

```
namespace DynDataWebApp
{
    public partial class DateCalendarField : FieldTemplateUserControl
    {
        // Code snipped
    }
}
```

Сниппет кода DynamicData\FieldTemplates\DateCalendar.ascx.cs



VB

```
<%@ Control Language="VB" CodeBehind="DateCalendar.ascx.vb"
            Inherits="DynDataWebApp.DateCalendarField" %>
```

Сниппет кода DynamicData\FieldTemplates\DateCalendar.ascx

```
Class DateCalendarField
    Inherits FieldTemplateUserControl
    ' Code Snipped
End Class
```

Сниппет кода DynamicData\FieldTemplates\DateCalendar.ascx.vb

Затем создайте копию шаблона `DateTime_Edit.ascx` и переименуйте его в `DateCalendar_Edit.ascx`. Как и прежде, откройте файл разметки и исходный файл для шаблона `DateCalendar_Edit.ascx` и переименуйте класс `DateTime_EditField` в `DateCalendar_EditField`. Ниже показано, как это будет выглядеть.



C#

```
<%@ Control Language="C#" CodeBehind="DateCalendar_Edit.ascx.cs"
            Inherits="DynDataWebApp.DateCalendar_EditField" %>
```

Сниппет кода DynamicData\FieldTemplates\DateCalendar_Edit.ascx

```
namespace DynDataWebApp
{
    public partial class DateCalendar_EditField :
FieldTemplateUserControl
    {
        // Code snipped
    }
}
```

Сниппет кода DynamicData\FieldTemplates\DateCalendar_Edit.ascx.cs



VB

```
<%@ Control Language="VB" CodeBehind="DateCalendar_Edit.ascx.vb"
            Inherits="DynDataWebApp.DateCalendar_EditField" %>
```

Сниппет кода DynamicData\FieldTemplates\DateCalendar.ascx

```
Class DateCalendar_EditField
    Inherits FieldTemplateUserControl
    ' Code Snipped
End Class
```

Сниппет кода DynamicData\FieldTemplates\DateCalendar.ascx.vb

Теперь можно было бы заменить элемент управления TextBox в файле `DateCalendar_Edit.ascx` на стандартный серверный веб-элемент управления Calendar. Однако это потребовало бы внести множество исправлений в исходном файле. Намного проще воспользоваться элементом управления Calendar из библиотеки AJAX Control Toolkit. Он является расширением стандартного элемента управления, который присоединяется к существующему элементу управления TextBox на веб-странице и обеспечивает новую функциональную возможность. Более подробную информацию о расширениях элементов управления и библиотеке AJAX Control Toolkit см. в главе 20.

Библиотеку AJAX Control Toolkit можно загрузить с веб-сайта `http://ajaxcontroltoolkit.codeplex.com/`. Для того чтобы добавить элементы управления из библиотеки AJAX Control Toolkit в инструментальное окно Toolbox системы Visual Studio, необходимо следовать инструкциям, приведенным в главе 20. Как только это будет сделано, добавьте элемент управления CalendarExtender в шаблон `DateCalendar_Edit.ascx`. Затем установите свойства `TargetControlID` и `Format` так, как показано в следующем фрагменте кода.

```
<cc1:CalendarExtender ID="CalendarExtender1" TargetControlID="TextBox1"
       Format="d-MMM-yyyy" runat="server">
      </cc1:CalendarExtender>
```

Сниппет кода DynamicData\FieldTemplates\DateCalendar_Edit.ascx

На последнем этапе установите связь между полями в модели данных и новыми шаблонами полей. В этом примере поля OrderDate, ShipDate и DueDate должны быть связаны с таблицей SalesOrderHeader. Модифицируйте частичный класс `SalesOrderHeader` и создайте метакласс, как было описано ранее. Для связи заданных полей с шаблоном полей используется атрибут `UIHint` (см. следующий фрагмент кода).



C#

```
namespace DynDataWebApp
{
    [ScaffoldTable(true)]
    [MetadataType(typeof(SalesOrderHeaderMetadata))]
    public partial class SalesOrderHeader
    {
    }
    public class SalesOrderHeaderMetadata
    {
        [DisplayFormat(DataFormatString = "{0:dd-MMM-yyyy}",
                       ApplyFormatInEditMode = true)]
        [UIHint("DateCalendar")]
         public object OrderDate;

        [DisplayFormat(DataFormatString = "{0:dd-MMM-yyyy}",
                       ApplyFormatInEditMode = true)]
        [UIHint("DateCalendar")]
         public object DueDate;

        [DisplayFormat(DataFormatString = "{0:dd-MMM-yyyy}",
                       ApplyFormatInEditMode = true)]
        [UIHint("DateCalendar")]
        public object ShipDate;
    }
}
```

Сниппет кода SalesOrderHeader.cs



VB

```
<ScaffoldTable(True)> _
<MetadataType(GetType(SalesOrderHeaderMetadata))> _
Partial Public Class SalesOrderHeader
End Class
Public Class SalesOrderHeaderMetadata
    <DisplayFormat(DataFormatString:="{0:dd-MMM-yyyy}",
                   ApplyFormatInEditMode:=True)> _
    <UIHint("DateCalendar")> _
    Public OrderDate As Object

    <DisplayFormat(DataFormatString:="{0:dd-MMM-yyyy}",
                   ApplyFormatInEditMode:=True)> _
    <UIHint("DateCalendar")> _
    Public DueDate As Object

    <DisplayFormat(DataFormatString:="{0:dd-MMM-yyyy}",
                   ApplyFormatInEditMode:=True)> _
    <UIHint("DateCalendar")> _
    Public ShipDate As Object
End Class
```

Сниппет кода SalesOrderHeader.vb

На рис. 23.11 показан пользовательский шаблон поля в представлении Edit для записи таблицы SalesOrderHeader.

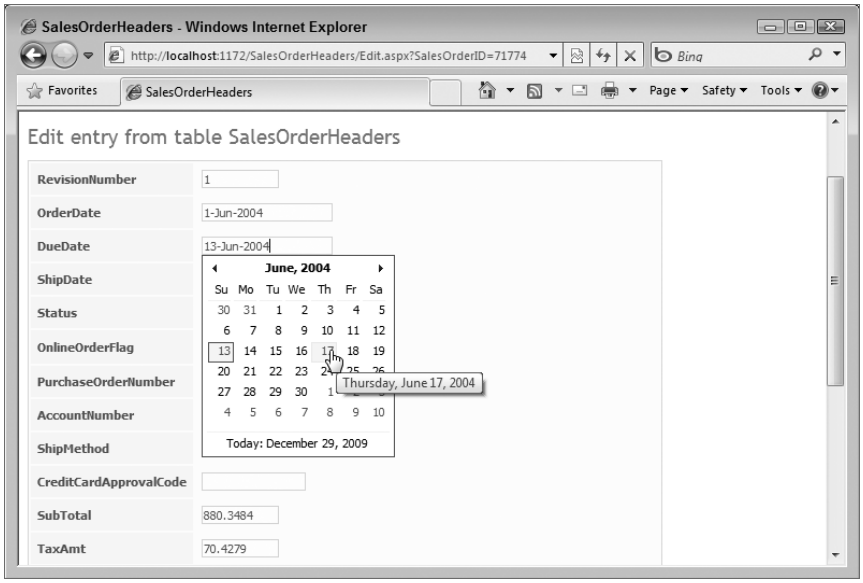


Рис. 23.11

Шаблоны сущностей

Шаблоны сущностей используются для визуализации отдельных записей таблиц в пользовательском интерфейсе. Шаблоны сущностей, заданные по умолчанию, хранятся в папке `DynamicData\EntityTemplates`. Они включают шаблоны для создания, редактирования и визуализации записи. Эти шаблоны работают с шаблонами страниц, заданными по умолчанию, и визуализируют пользовательский интерфейс в виде HTML-таблицы с двумя столбцами: в левом столбце размещается метка, в правом — поле данных.

Настройка существующих шаблонов сущностей влияет на все таблицы. Кроме того, можно создать новый пользовательский шаблон сущностей для конкретной таблицы. Это позволяет создавать совершено иную компоновку при редактировании записей в базах данных по сравнению с обычным просмотром записи.

Чтобы создать новый шаблон сущности, щелкните правой кнопкой мыши на папке `DynamicData\EntityTemplate` и выберите команду Add⇨New Item. Затем выберите новый элемент Web User Control и назовите его `SalesOrderHeaders.ascx`.

Шаблоны, заданные по умолчанию, используют элемент управления EntityTemplate, отдаленно похожий на серверный веб-элемент управления Repeater. Этот элемент управления динамически генерирует все поля таблицы по модели данных. В данном случае вместо использования элемента управления EntityTemplate можно вручную задать визуализируемые поля. В следующем фрагменте кода приведена пользовательская разметка для шаблона сущности, которая визуализирует подмножество данных.



```
<tr>
   <td class="DDLightHeader">
      <asp:Label ID="Label1" runat="server" Text="Customer" />
   </td>
   <td>
      Acct No:
```

```
        <asp:DynamicControl ID="DynamicControl1" runat="server"
             DataField="AccountNumber" />
        <br/>
    PO No:
    <asp:DynamicControl ID="DynamicControl2" runat="server"
         DataField="PurchaseOrderNumber" />
    </td>
</tr>
<tr>
    <td class="DDLightHeader">
        <asp:Label ID="Label2" runat="server" Text="Dates" />
    </td>
    <td>
        Ordered:
        <asp:DynamicControl ID="DynamicControl3" runat="server"
             DataField="OrderDate" />
    <br/>
    Due:
    <asp:DynamicControl ID="DynamicControl4" runat="server"
         DataField="DueDate" />
    <br/>
    Shipped:
    <asp:DynamicControl ID="DynamicControl5" runat="server"
         DataField="ShipDate" />
    </td>
</tr>
<tr>
    <td class="DDLightHeader">
        <asp:Label ID="Label3" runat="server" Text="Amount" />
    </td>
    <td>
    Sub Total:
    <asp:DynamicControl ID="DynamicControl6" runat="server"
         DataField="SubTotal" DataFormatString="{0:c}" />
    <br/>
    Tax:
    <asp:DynamicControl ID="DynamicControl7" runat="server"
         DataField="TaxAmt" DataFormatString="{0:c}" />
     <br/>
     Freight:
     <asp:DynamicControl ID="DynamicControl8" runat="server"
          DataField="Freight" DataFormatString="{0:c}" />
     </td>
</tr>
```

Сниппет кода DynamicData\EntityTemplates\SalesOrderHeader.ascx

И наконец, измените пользовательский веб-элемент управления, чтобы он был производным от класса `System.Web.DynamicData.EntityTemplate UserControl`, а не `System.Web.UI.UserControl`.



C#

```
public partial class SalesOrderHeaders :
        System.Web.DynamicData.EntityTemplateUserControl
```

Сниппет кода DynamicData\EntityTemplates\SalesOrderHeader.ascx.cs



VB

```
Public Class SalesOrderHeaders
    Inherits System.Web.DynamicData.EntityTemplateUserControl
```

Сниппет кода DynamicData\EntityTemplates\SalesOrderHeader.ascx.vb

Теперь можно собрать и запустить проект для тестирования нового шаблона сущности. На рис. 23.12, *слева*, показан шаблон сущности, заданный по умолчанию, а *справа* — новый настраиваемый шаблон для таблицы SalesOrderHeader. Представления Edit и Insert остались неизменными, поскольку единственным настроенным шаблоном был шаблон Details, предназначенный только для чтения.

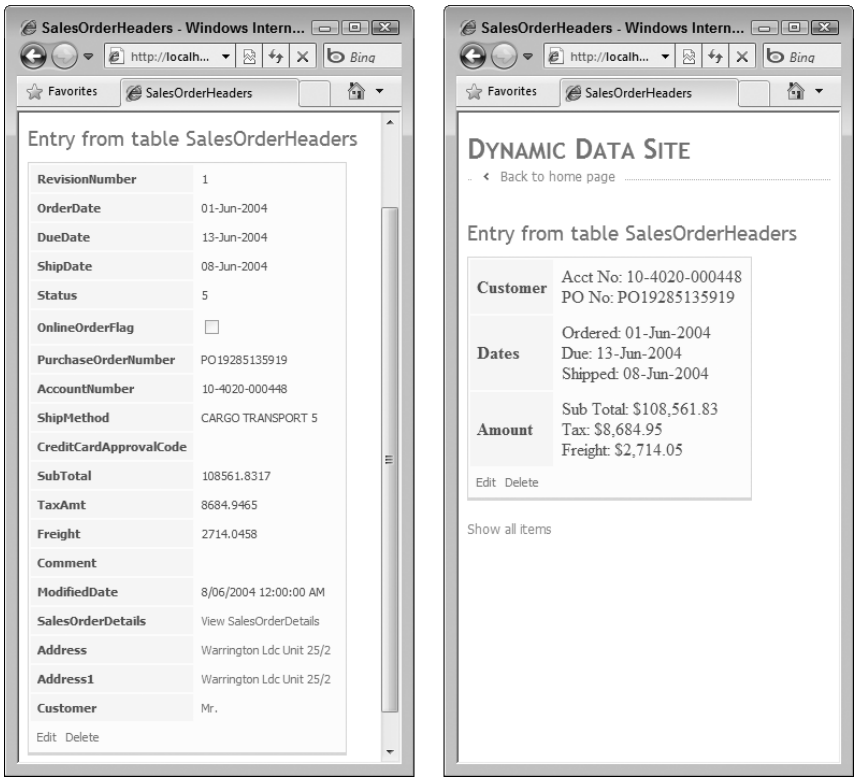


Рис. 23.12

Шаблоны фильтров

Шаблоны фильтров используются для визуализации элемента управления, который выполняет фильтрацию строк, показанных в таблице. Механизм Dynamic Data содержит три шаблона фильтров, которые хранятся в папке `DynamicData\Filters`. Эти фильтры имеют очевидные имена: фильтр `Boolean` используется для данных булевого типа, фильтр `Enumeration` используется, когда тип данных отображается в перечисление, а фильтр `ForeignKey` — для внешних ключей.

На рис. 23.13 показаны четыре шаблона фильтров, которые по умолчанию визуализируются для таблицы SalesOrderHeader. Первый фильтр, OnlineOrderFlag, является булевым фильтром и содержит только три опции — All, True и False. Остальные три фильтра генерируются по внешним ключам, и каждый из них содержит много опций.

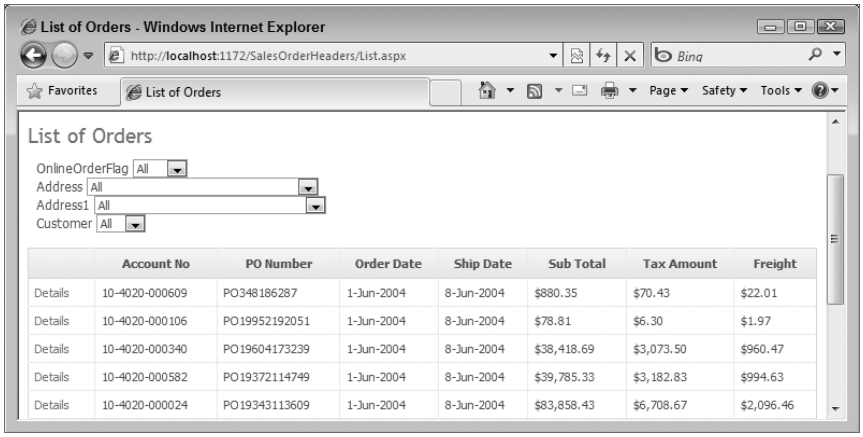


Рис. 23.13



Возможно, читатели уже заметили, что значения, отображаемые в раскрывающемся списке Customer, представляют собой обращение к пользователю (Mr, Mrs и т.д.) и не имеют никакого смысла в дальнейшей работе. Для того чтобы выбрать поле, отображаемое по внешнему ключу, механизм Dynamic Data находит первое поле в таблице, имеющее строковый тип. Его можно заменить любым другим полем таблицы, включив в модель данных класс с атрибутом `DisplayColumn`. Однако мы хотим, чтобы в таблице `Customer` визуализировалась строка, содержащая количество полей (`FirstName, LastName`). Для этого следует заменить метод `ToString` в классе модели данных для таблицы `Customer`.

К сожалению, раскрывающиеся списки полезны, только если они содержат меньше нескольких сотен пунктов. Иначе визуализация веб-страницы замедлится и по списку будет сложно перемещаться. Поскольку количество записей в таблицах достигает нескольких тысяч и более, использование раскрывающихся списков для внешних ключей `Address`, `Address1` и `Customer` сделает визуализацию страницы неприемлемой.

Если необходимо использовать именно эти фильтры, необходимо сделать нечто более сложное, например, добавить в фильтр ForeignKey, заданный по умолчанию, элемент управления для поиска, который выполнял бы обратный вызов сервера и выводил на экран корректные записи, соответствующие критерию поиска, разумеется, в рамках запроса с помощью технологии AJAX. Однако такое упражнение выходит за рамки нашей книги, поэтому вместо него покажем, как указать, какие поля следует визуализировать как фильтры.



В оставшейся части главы мы предполагаем, что для таблицы SalesOrderDetail уже создан пользовательский шаблон страницы, описанный ранее.

Откройте пользовательский шаблон `List.aspx` для таблицы SalesOrderHeader из папки `DynamicData\CustomPages\SalesOrderHeaders`. Найдите на этой странице элемент управления QueryableFilterRepeater. Этот элемент управления используется для генерации списка фильтров. Удалите его и на его место добавьте элемент управления DynamicFilter, как показано в следующем фрагменте кода. Свойству `DataField` необхо-

димо присвоить правильное поле данных, соответствующее фильтру, а свойство FilterUIHint должно ссылаться на корректный шаблон фильтра.



```
Online Order:
<asp:DynamicFilter ID="OnlineOrderFilter" runat="server"
     DataField="OnlineOrderFlag" FilterUIHint="Boolean"
     OnFilterChanged="DynamicFilter_FilterChanged">
</asp:DynamicFilter>
```

Сниппет кода DynamicData\CustomPages\SalesOrderHeader\List.aspx

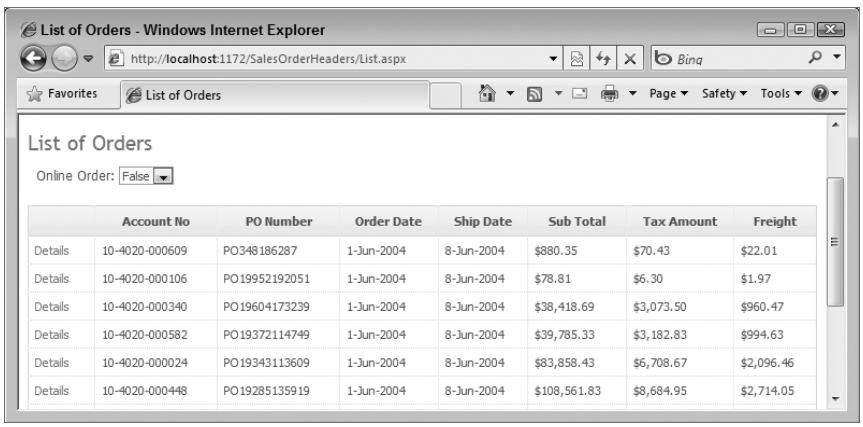
Затем найдите элемент управления QueryExtender, расположенный в нижней части страницы. Этот элемент управления используется для связывания элемента управления DynamicFilter с источником данных, чтобы при изменении фильтра использовался правильный запрос. Измените свойство ControlID так, чтобы оно соответствовало имени только что добавленного элемента управления DynamicFilter, как показано в следующем коде.



```
<asp:QueryExtender TargetControlID="GridDataSource"
ID="GridQueryExtender" runat="server">
    <asp:DynamicFilterExpression ControlID="OnlineOrderFilter" />
</asp:QueryExtender>
```

Сниппет кода DynamicData\CustomPages\SalesOrderHeader\List.aspx

Теперь следует удалить часть кода, которая используется только элементом управления QueryableFilterRepeater. Откройте исходный файл (List.aspx.cs или List.aspx.vb) и удалите метод Label_PreRender. Сохранив изменения и запустив проект на выполнение, вы увидите только один фильтр для таблицы SalesOrderHeader, показанный на рис. 23.14.



	Account No	PO Number	Order Date	Ship Date	Sub Total	Tax Amount	Freight
Details	10-4020-000609	PO348186287	1-Jun-2004	8-Jun-2004	$880.35	$70.43	$22.01
Details	10-4020-000106	PO19952192051	1-Jun-2004	8-Jun-2004	$78.81	$6.30	$1.97
Details	10-4020-000340	PO19604173239	1-Jun-2004	8-Jun-2004	$38,418.69	$3,073.50	$960.47
Details	10-4020-000582	PO19372114749	1-Jun-2004	8-Jun-2004	$39,785.33	$3,182.83	$994.63
Details	10-4020-000024	PO19343113609	1-Jun-2004	8-Jun-2004	$83,858.43	$6,708.67	$2,096.46
Details	10-4020-000448	PO19285135919	1-Jun-2004	8-Jun-2004	$108,561.83	$8,684.95	$2,714.05

Рис. 23.14

Подключение механизма Dynamic Data к существующим проектам

Механизм Dynamic Data несомненно является очень мощным средством создания абсолютно новых веб-приложений, управляемых данными. Однако версия механизма Dynamic Data, поставляемая с системой Visual Studio 2010, позволяет использовать

некоторые функциональные возможности этого инструмента в существующих проектах типа Web Application или Web Site.

Для этого был разработан метод расширения EnableDynamicData. Его можно вызвать из любого класса, который реализует интерфейс System.Web.UI.INamingContainer. Он включает в себя элементы управления Repeater, DataGrid, DataList, CheckBoxList, ChangePassword, LoginView, Menu, SiteMapNodeItem и RadioButtonList.

Добавление новых функциональных возможностей в существующий веб-элемент управления не требует, чтобы приложение использовало технологию LINQ to SQL или Entity Framework. Фактически приложение может использовать любой механизм доступа к данным, включая старую технологию ADO.NET. Это объясняется тем, что механизм Dynamic Data подключается таким образом, что не требуется использовать скаффолдинг. При этом он позволяет использовать как шаблоны полей, так и проверку корректности и визуализацию атрибутов, как описано выше.

Например, для того, чтобы подключить механизм Dynamic Data к элементу управления GridView, вызовите метод расширения EnableDynamicData, как показано в следующем фрагменте кода.

C#

```
GridView1.EnableDynamicData(typeof(Product));
```

VB

```
GridView1.EnableDynamicData(GetType(Product))
```

Теперь можно создать класс Product с открытыми свойствами, соответствующий данным, отображаемым в элементе управления GridView1. Каждое из этих свойств можно снабдить атрибутами из пространства имен System.ComponentModel.DataAnnotations, например Required, StringLength, RegularExpression или DisplayFormat. Платформа ASP.NET интерпретирует эти атрибуты на этапе выполнения программы и автоматически применит к ним соответствующую проверку корректности и форматирование.

Это позволяет любому приложению использовать функциональные возможности механизма Dynamic Data без внесения существенных изменений.

Резюме

В этой главе показано, как с помощью механизма ASP.NET Dynamic Data создать веб-приложение, управляемое данными, с минимальным объемом кодирования или вообще без создания кода вручную. Еще более важно то, что мы продемонстрировали, насколько гибок механизм Dynamic Data, на примере настройки модели данных и веб-страниц.

Освободив разработчиков от низкоуровневого программирования, механизм Dynamic Data ускорил процесс создания приложений и позволил разработчикам предоставлять конечным пользователям более ценные функциональные возможности.

24

Коллекция программных продуктов SharePoint

В ЭТОЙ ГЛАВЕ...

- Настройка среды разработки для коллекции SharePoint
- Разработка пользовательских компонентов для коллекции SharePoint: веб-части, список и рабочие процессы
- Отладка и тестирование проектов SharePoint
- Упаковка и развертывание компонентов SharePoint

За последние несколько лет уровень интереса и количество инсталляций коллекции SharePoint достигли такого масштаба, что она стала одной из наиболее быстро развивающихся серий продукции компании Microsoft.

По существу, SharePoint — это коллекция связанных между собой продуктов и технологий, охватывающих широкий круг вопросов управления документооборотом и информационным наполнением веб-страниц, взаимодействия с помощью веб, а также веб-поиска. SharePoint — это также очень гибкая платформа для внедрения приложений, позволяющая разрабатывать и разворачивать практически все: от отдельных веб-частей до полноценных веб-приложений.

Несмотря на то что коллекцию SharePoint можно использовать для предоставления веб-сайтов анонимным внешним посетителям, она намного лучше подходит для веб-сайтов, предусматривающих регистрацию пользователей, особенно тех, кто обслуживает работников внутри организации. Коллекция SharePoint содержит намного больше низкоуровневого интеграционного кода, чем обычно требуется от таких сред, включая код для аутентификации и авторизации, интеграции с пакетом Microsoft Office, доступа к внешним данным и автоматизации документооборота.

В главе описывается применение средств разработки из коллекции SharePoint в системе Visual Studio 2010, а также демонстрируется сборка, отладка и развертывание решений, созданных с помощью коллекции SharePoint.



Кроме системы Visual Studio 2010, для создания решений с помощью коллекции SharePoint можно использовать свободно распространяемую программу SharePoint Designer 2010. Эта программа воплощает совершенно другой подход к реализации приложений, предоставляя элементы решения SharePoint на высоком уровне абстракции, скрывающем многие детали. Она также содержит некоторые инструменты WYSIWYG для просмотра и редактирования компонентов на существующих веб-сайтах, созданных с помощью коллекции SharePoint. В этом качестве программа SharePoint Designer часто рассматривается как удобный инструмент для людей, не занимающихся профессиональной разработкой приложений (специалистов по информационным технологиям и конечных пользователей). Однако она полезна и для профессионалов при решении определенных задач, связанных с разработкой и конфигурацией приложений. Например, с помощью программы SharePoint Designer намного легче создать компоновку страницы или страницу мастера. Как правило, опытные разработчики, использующие коллекцию SharePoint, применяют оба инструмента для создания своих решений.

Подготовка среды разработки

Одна из основных претензий к предыдущим версиям коллекции SharePoint заключалась в том, что разработчик должен был в качестве локальной среды разработки использовать серверную операционную систему Windows Server. Это объяснялось тем, что версии SharePoint 2007 и предшествующие ей могли запускаться только под управлением серверных операционных систем, причем для отладки и тестирования интеграции среда SharePoint должна была запускаться локально.

К счастью, этот недостаток в версии SharePoint 2010 был исправлен. Кроме операционной системы Windows Server 2008, платформу SharePoint можно инсталлировать также в операционных системах Windows 7 и Windows Vista (Service Pack 1 и более поздние версии). К сожалению, для этого компьютер, на котором будет выполняться разработка, должен обладать мощным аппаратным обеспечением. Платформа SharePoint 2010 требует 64-битовой операционной системы и 4 Гбайт оперативной памяти для редакции SharePoint Foundation, а также от 6 до 8 Гбайт оперативной памяти для редакции SharePoint Server.

РЕДАКЦИИ SHAREPOINT SERVER И SHAREPOINT FOUNDATION

Платформа SharePoint 2010 имеет две редакции: SharePoint Server и SharePoint Foundation. Платформа SharePoint Foundation, в предыдущих версиях она называлась Windows SharePoint Services (WSS), распространяется свободно и предназначена для небольших организаций или проектов. Она включает в себя поддержку веб-частей и веб-приложений, а также функциональные возможности для взаимодействия через веб, например блоки, вики, календари и дискуссии.

С другой стороны, платформа SharePoint Server нацелена на крупные предприятия и сложные сценарии разработки. Она является дорогим инструментом для создания серверных продуктов и требует от каждого пользователя наличия лицензии клиентского доступа (Client Access License — CAL). Платформа SharePoint Server включает в себя все функциональные возможности платформы SharePoint Foundation, позволяет создавать группы веб-сайтов SharePoint, реализовывать сложные сценарии навигации, индексированный поиск, доступ к данным из баз данных, персонализацию и технологию Single Sign On.

Инсталляция необходимых инструментов



Платформа SharePoint 2010 в момент написания книги была реализована в виде бета-версии, и поэтому некоторые инструкции в окончательном выпуске могут быть изменены. Если в ней появятся важные изменения, пользователи смогут найти поправки и обновления на веб-сайте `www.wrox.com`. Для этого следует выполнить поиск книги по индексу ISBN 978-0-470-54865-3 и найти главу 24 в списке ошибок.

При работе под управлением операционной системы Windows Server процесс установки платформы SharePoint довольно прост. Пакет инсталляции содержит инструмент Prerequisite Installer (`PrerequisiteInstaller.exe`), который проверяет и инсталлирует все необходимые компоненты. Однако в версии SharePoint 2010 Beta 2 этот инструмент не работал ни в системе Windows 7, ни в системе Windows Vista. Если необходимо инсталлировать платформу SharePoint 2010 в одной из этих клиентских операционных систем, то следует предварительно инсталлировать и вручную настроить большое количество компонентов.

Независимо от используемой операционной системы, сначала необходимо инсталлировать "заплатку" WCF Hotfix для системы Microsoft Windows. Ее можно загрузить со следующих веб-страниц.

- Для систем Windows Vista или Windows Server 2008:
  `http://go.microsoft.com/fwlink/?linkID=160770`
- Для систем Windows 7 или Windows Server 2008 R2:
  `http://go.microsoft.com/fwlink/?LinkID=166231`



Дальнейшие инструкции основаны на предположении, что инсталляционный пакет имеет вид самораспаковывающегося архива `SharePoint.exe` для версии SharePoint Foundation или `OfficeServer.exe` для версии SharePoint Server. Если же инсталляция производится с CD/DVD, то следующий этап можно пропустить, поскольку содержимое и структура папок диска совпадают с содержимым и структурой папок, извлекаемых из архива файлов.

Если вам не требуется вся мощь редакции SharePoint Server, воспользуйтесь преимуществами менее требовательной к ресурсам редакции SharePoint Foundation. Поскольку платформа SharePoint Server построена на основе платформы SharePoint Foundation, все, кто работал с версией SharePoint Foundation, смогут работать с версией SharePoint Server.

Начните с создания папки для инсталляционных файлов, например `c:\SharePoint`, и скопируйте самораспаковывающийся инсталляционный архив в эту папку. Затем извлеките инсталляционные файл, выполнив следующую команду (для версии SharePoint Foundation).

```
c:\SharePoint\SharePoint.exe /extract:c:\SharePoint
```

Для версии SharePoint Server имя модуля `SharePoint.exe` необходимо заменить на `OfficeServer.exe`.

При инсталляции платформы SharePoint в системе Windows Server необходимо запустить инструмент Prerequisite Installer (`PrerequisiteInstaller.exe`), а затем перейти к следующему разделу ("Инсталляция платформы SharePoint 2010"). В противном случае, если целевой операционной системой является Windows Vista или Windows 7, необходимо вручную инсталлировать вспомогательные компоненты, как описано ниже.

Для системы Windows Vista необходимы следующие компоненты.

- **.NET Framework 3.5 SP1.** Если система Visual Studio 2010 уже установлена, то эта платформа уже инсталлирована. Ее можно загрузить с веб-страницы `professionalvisualstudio.com/link/1024A`.
- **Windows PowerShell 2.0.** Эта оболочка доступна на веб-странице `professionalvisualstudio.com/link/1024B`.
- **Windows Installer 4.5 Redistributable.** Эта программа доступна на веб-странице `professionalvisualstudio.com/link/1024C`.

Для система Windows 7 и Windows Vista необходимы следующие компоненты.

- **Microsoft FilterPack 2.0.** Этот пакет поставляется в инсталляционном наборе файлов для платформы SharePoint. Следует запустить инсталляционный пакет, расположенный по адресу `c:\SharePoint\PrerequisiteInstallerFiles\FilterPack\FilterPack.msi`.
- **Microsoft Sync Framework.** Если система Visual Studio 2010 уже установлена, то эта платформа уже инсталлирована. Ее можно загрузить с веб-страницы `professionalvisualstudio.com/link/1024D`.
- **SQL Server 2008 Native Client.** Если система Visual Studio 2010 уже установлена, то этот интерфейс уже инсталлирован. Он доступен по адресу `professionalvisualstudio.com/link/1024E`.
- **Windows Identity Foundation Runtime.** Платформа, ранее известная под кодовым названием "Geneva". Доступна по адресу `professionalvisualstudio.com/link/1024F`.
- **ADO.NET Data Services**. Эта платформа доступна по адресу `professionalvisualstudio.com/link/1024G`.
- **Chart Controls.** Эта программа для платформы SharePoint Foundation не нужна. Доступна по адресу `professionalvisualstudio.com/link/1024H`.
- **Microsoft ADOMD.NET.** Этот компонент для платформы SharePoint Foundation не нужен. Доступен по адресу `professionalvisualstudio.com/link/1024I`.

На последнем этапе необходимо подключить все требуемые свойства Windows Features. Эти свойства перечислены на рис. 24.1 и могут быть подключены с помощью раздела Programs and Features панели управления Control Panel. Кроме того, с веб-

страницы professionalvisualstudio.com/link/1024J можно загрузить сценарий, который автоматически включает все эти свойства.

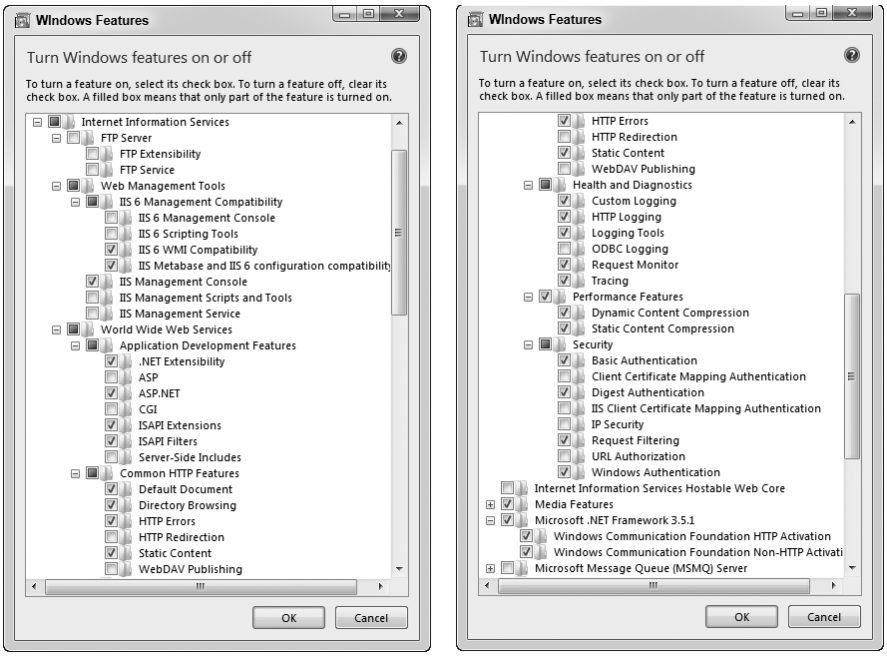


Рис. 24.1

Инсталляция платформы SharePoint 2010

Теперь, когда все необходимые компоненты заранее установлены, можно приступать к инсталляции платформы SharePoint Foundation или SharePoint Server. При инсталляции этих платформ под управлением операционной системы Windows Server нужно запустить на выполнение файл инсталляции `setup.exe`. Однако, если инсталляция производится в системе Windows 7 или Windows Vista, при попытке запустить модуль `setup.exe` на экране появится ошибка (рис. 24.2).

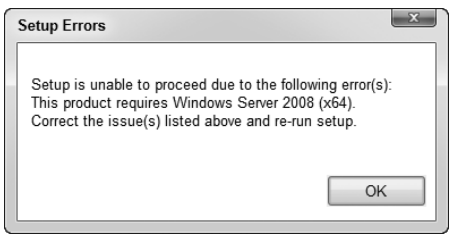


Рис. 24.2

Для того чтобы снять это ограничение, необходимо редактировать файл `config.xml`, расположенный в папке `c:\SharePoint\files\Setup`. Добавьте в дескриптор `<configuration>` следующую строку.

```
<Setting Id="AllowWindowsClientInstall" Value="True"/>
```

Сохраните файл конфигурации и выполните модуль `setup.exe`. Затем, следуя инструкциям программы инсталляции, выберите команду Standalone installation (Install single server standalone using default settings). После того как выполнение программы инсталляции будет завершено, можно запустить на выполнение мастер SharePoint

Products Configuration Wizard. Когда он закончит работу, сайт SharePoint, заданный по умолчанию, откроется в новом окне браузера (рис. 24.3).

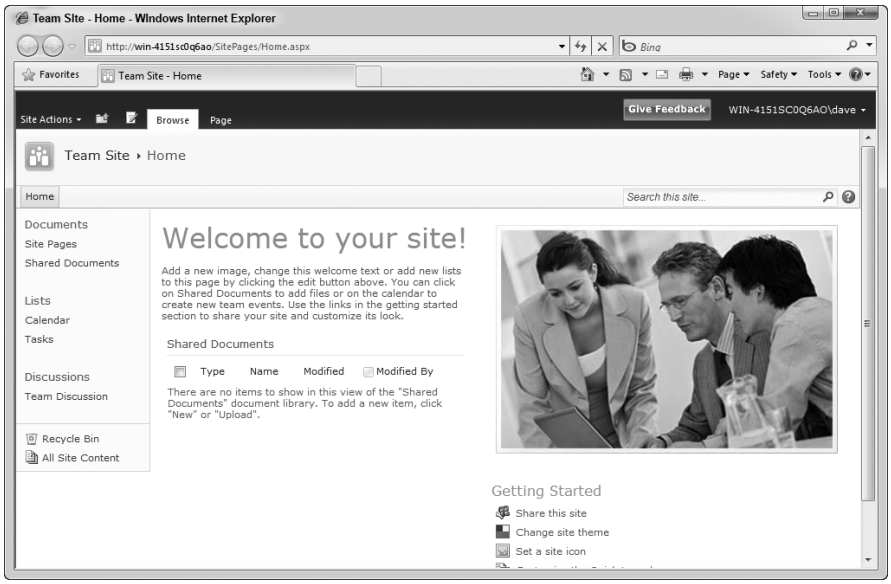


Рис. 24.3

Изучение платформы SharePoint 2010

Первое впечатление от платформы SharePoint может оказаться ошеломительным. Одна из причин заключается в том, что бо́льшая часть терминологии, используемой в платформе SharePoint, веб-разработчикам незнакома, даже если они хорошо изучили платформу ASP.NET. Прежде чем приступать в разработке решения с помощью платформы SharePoint, полезно изучить ее компоненты, например типы контента, коллекцию Features, получателей событий, списки, рабочие процессы и веб-части.

Окно Server Explorer в системе Visual Studio 2010 позволяет рассматривать сайт SharePoint и перебирать его компоненты. Для того чтобы соединиться с сайтом SharePoint или разработать и отладить решение SharePoint, необходимо запустить систему Visual Studio с правами администратора. Для этого следует щелкнуть правой кнопкой мыши на пиктограмме системы Visual Studio 2010 и выбрать команду Run as Administrator.



Для того чтобы всегда запускать систему Visual Studio 2010 с правами администратора, щелкните правой кнопкой мыши на пиктограмме системы и выберите команду Properties, а затем выберите корешок закладки Compatibility и установите флажок Run This Program as an Administrator.

Откройте окно Server Explorer, выбрав команду View⇨Server Explorer. Соединиться с сайтом SharePoint можно, только если на локальном компьютере установлена платформа SharePoint. По умолчанию соединение с локальной платформой SharePoint автоматически указывается в узле SharePoint Connections. Кроме того, можно добавить соединение с удаленным сервером, щелкнув правой кнопкой мыши на узле SharePoint Connections и выбрав команду Add Connection.

Когда пользователь щелкает мышью на компоненте SharePoint в окне Server Explorer, в окне Properties выводятся свойства этого компонента. Окно Server Explorer открывает к компоненту SharePoint доступ только для чтения. Окно Server Explorer и свойства сайта SharePoint показаны на рис. 24.4.

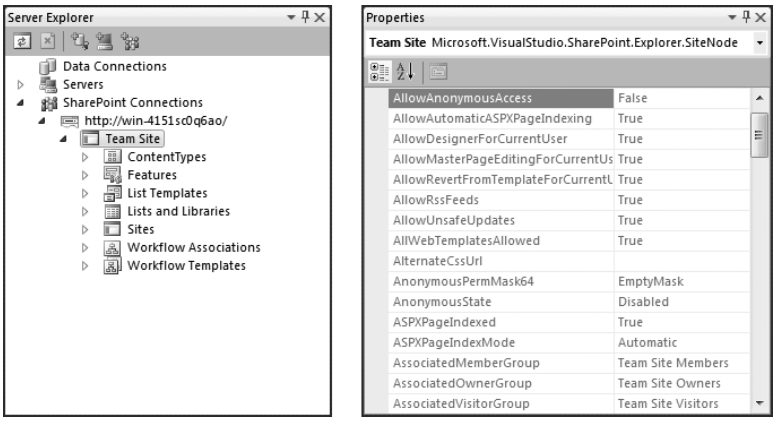


Рис. 24.4

Разобравшись, как соединиться с сайтом SharePoint и просматривать его, попробуем понять основные концепции, используемые в технологии SharePoint.

Типы контента (content types) определяют способы определения разных видов информационного наполнения на платформе SharePoint, например документ или уведомление. С типом контента связана совокупность полей, определяющих метаданные контента. Например, тип контента `Document`, показанный на рис. 24.5, имеет такие поля, как заголовок и дата последнего изменения. Тип контента имеет свойства, определяющие настройки, например шаблон для вывода на экран, редактирования или создания нового экземпляра.

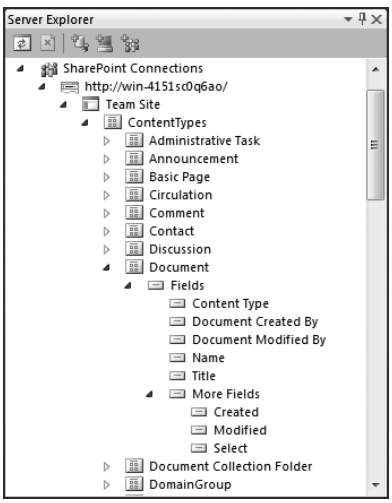


Рис. 24.5

Коллекция Features содержит ресурсы, описывающие логический набор функциональных возможностей. Например, платформа SharePoint поставляется с коллекцией Features, содержащей *листы рассылки* (discussion lists), *библиотеки документов* (document libraries) и *списки обследуемого аппаратного и программного обеспечения* (survey lists).

Коллекция Features содержит шаблоны, страницы, списки определений, получателей событий и рабочие процессы. Кроме того, она содержит такие ресурсы, как изображения, файлы на языке JavaScript и файлы CSS. В нее также входят *получатели событий* (*event receivers*), представляющие собой обработчики событий, которые вызываются, когда активизируется, деактивизируется, инсталлируется, деинсталлируется или обновляется та или иная функциональная возможность из коллекции Features. Обработчики событий могут также создаваться для других компонентов платформы SharePoint, например для списков и сайтов.

Списки (lists) являются основой платформы SharePoint и используются практически везде. Такие функциональные возможности, как обследования, выпуски и библиотеки документов, созданы на основе списков. *Определение списка* (*list definition*) задает поля, формы, представления (страницы `.aspx`) и типы контента, связанного со списком. Конкретная реализация определения списка называется *экземпляром списка* (*list instance*).

Рабочие процессы (workflows) на платформе SharePoint используются для автоматизации бизнес-процессов. Рабочие процессы SharePoint на самом деле построены на основе одного и того же механизма (Windows Workflow Foundation), который поставляется вместе с платформой .NET v3.5. Рабочие процессы могут быть связаны с конкретным сайтом SharePoint, списком или типом контента.

В заключение отметим, что *веб-части* (Web Parts) — это серверные элементы веб-управления, внедренные в страницу Web Part на платформе SharePoint. Пользователи могут персонализировать страницу Web Part и выбрать для визуализации одну или несколько таких страниц. Веб-части могут визуализировать все, от простых статических меток, содержащих информацию на веб-сайте, до сложных форм, предназначенных для ввода деловых данных.

Создание проекта Sharepoint

Ознакомившись с основными концепциями технологии SharePoint, перейдем к созданию вашего первого решения на платформе SharePoint. Выберите в среде Visual Studio 2010 команду File⇨New⇨Project. Отфильтруйте типы проектов, выбрав команду Visual C# или Visual Basic, а затем команды SharePoint и 2010. На экране появятся доступные типы проектов на платформе SharePoint (рис. 24.6).

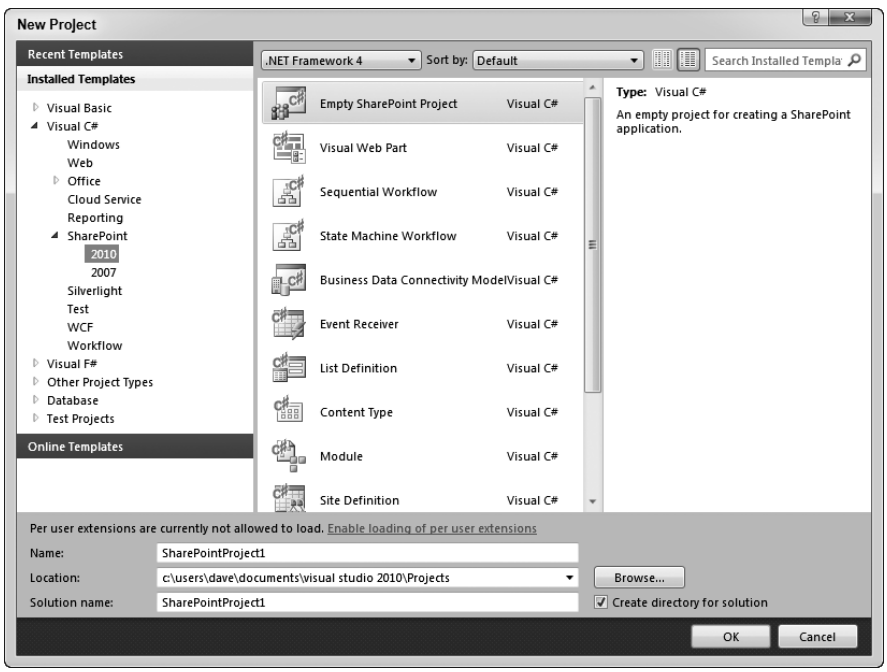


Рис. 24.6

Вместе с системой Visual Studio 2010 поставляется большое количество проектных шаблонов для платформы SharePoint. Большинство из компонентов SharePoint, которые можно создать на основе этих шаблонов, можно создать и как отдельные компоненты в существующем решении SharePoint. Поэтому выберите новый проект Empty SharePoint Project.

Щелкните на кнопке OK, и система Visual Studio запустит мастер SharePoint Customization Wizard (рис. 24.7). После этого вы получите приглашение задать вид сайта и уровень безопасности для отладки. Поскольку невозможно отладить сайты SharePoint на удаленных компьютерах, можно выбрать только локальный сайт SharePoint. Вы должны также выбрать уровень доверия для решения SharePoint, которое будет развернуто для отладки. Выберите команду Deploy as a Farm Solution и щелкните на кнопке Finish.

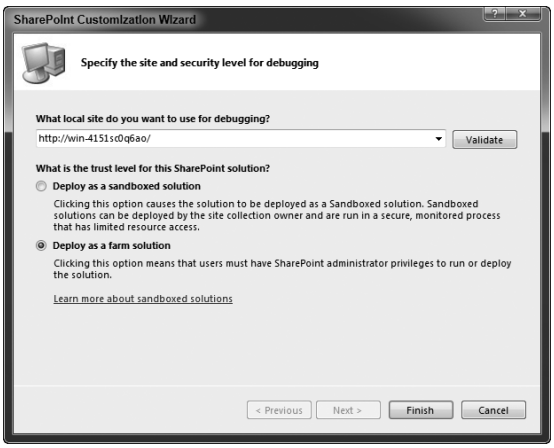


Рис. 24.7



Решения, запускаемые в ограниченной среде, имеют ограниченный уровень доверия и могут использовать только часть функциональных возможностей. Ограниченная среда отслеживает показатели производительности приложения, включая время загрузки центрального процессора, уровень загрузки памяти и время выполнения запросов к базе данных. Кроме того, решения, выполняемые в ограниченной среде, невозможно активизировать, не выполнив процесс проверки корректности. Это позволяет администратору приложения SharePoint быть уверенным в том, что злонамеренный компонент не повиляет на остальную часть окружения SharePoint.

Когда проект SharePoint будет создан, разработчик заметит в окне Solution Explorer два уникальных узла. Эти узлы есть в каждом проекте SharePoint, и их невозможно удалить, переместить или переименовать.

Узел Features может содержать одну или несколько функциональных возможностей платформы SharePoint. Как указывалось в предыдущем разделе, коллекция Feature содержит ресурсы, описывающие логический набор функциональных возможностей. Каждый раз при добавлении нового компонента, например визуальной вебчасти, он добавляется в коллекцию Feature, т.е. в узел Features. В зависимости от *области видимости* компонента, он будет добавлен либо в существующую коллекцию Feature, либо будет создана новая. Функциональные возможности обсуждаются в этой главе в разделе “Работа с компонентами коллекции Feature”.

Узел Package содержит один файл, служащий механизмом развертывания для проекта SharePoint. Пакет имеет расширение `.wsp` и является логическим эквивалентом файла инсталляции. Пакет содержит набор функциональных свойств, определения сайтов и дополнительные сборки, которые разворачиваются на сайте SharePoint. Пакеты обсуждаются ниже, в разделе “Упаковка и развертывание”.

Для того чтобы добавить компонент SharePoint в решение, щелкните правой кнопкой мыши на проекте в окне Solution Explorer и выберите команду Add⇨New Item. Как показано на рис. 24.8, система Visual Studio поставляется с шаблонами для многих компонентов платформы SharePoint. Выберите элемент Application Page, введите имя `MyPage.aspx` и щелкните на кнопке OK.

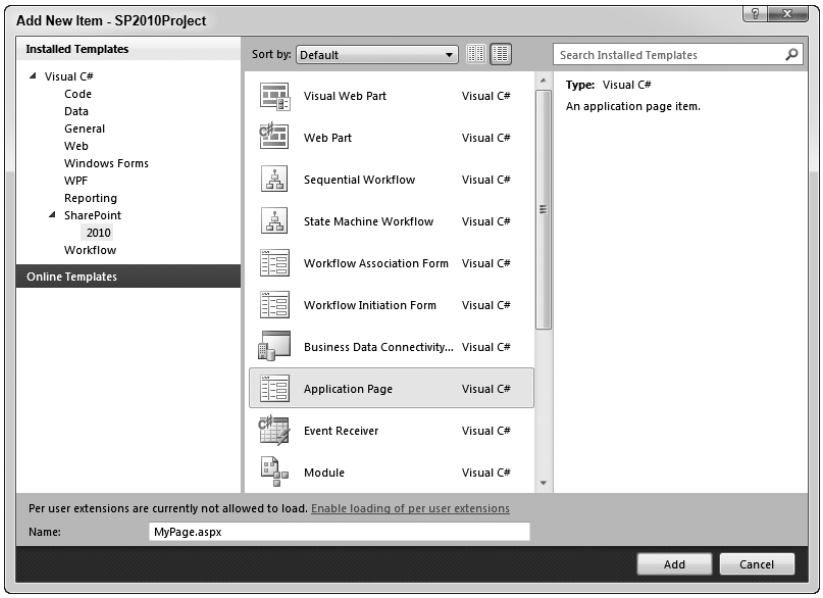


Рис. 24.8

Application page — это один из двух типов веб-страниц ASP.NET, существующих на сайтах SharePoint. Большинство страниц, с которыми взаимодействуют пользователи сайтов на платформе SharePoint, по существу, являются *страницами контента* (content pages). Система Visual Studio не содержит шаблон для страниц контента. Вместо этого страницы контента создаются и редактируются такими инструментами, как SharePoint Designer, или с помощью объектной модели SharePoint Foundation. Страницы контента можно добавлять в библиотеку страниц платформы SharePoint. Кроме того, в них могут внедряться динамические веб-части.



Объектная модель SharePoint Foundation 2010 состоит из более чем 70 пространств имен и предоставляет интерфейс прикладного программирования, позволяющий программно выполнять большинство административных и пользовательских задач. Большинство классов содержится в сборках `Microsoft.SharePoint.dll` и `Microsoft.SharePoint.Client.dll`. Эти классы можно использовать только для работы с локальной платформой SharePoint Foundation или средой SharePoint Server.

Несмотря на то что страницы приложения не могут делать так много, как страницы контента, он гораздо лучше поддерживают работу с пользовательским кодом. По этой причине страницы приложения часто используются для реализации функций, не связанных с администрированием.

Когда страница приложения добавляется в проект, она не добавляется в узел проекта, а размещается в папке `Layouts` папки нижнего уровня с тем же именем, что и проект.

Папка `Layouts` не может изменяться, а папку нижнего уровня можно переименовать в любой момент. Папка `Layouts` представляет собой пример *сопоставленной* папки (SharePoint Mapped Folder). Такая папка, по существу, является сокращенным вариантом стандартной папки на платформе SharePoint и позволяет не указывать полный путь к папке, находящейся в решении SharePoint. Для того чтобы добавить сопоставленные папки в свой проект, следует щелкнуть правой кнопкой мыши на проекте и выбрать команду Add⇨SharePoint Mapped Folder. В открывшемся диалоговом окне отобразятся все папки, доступные на платформе SharePoint (рис. 24.9).

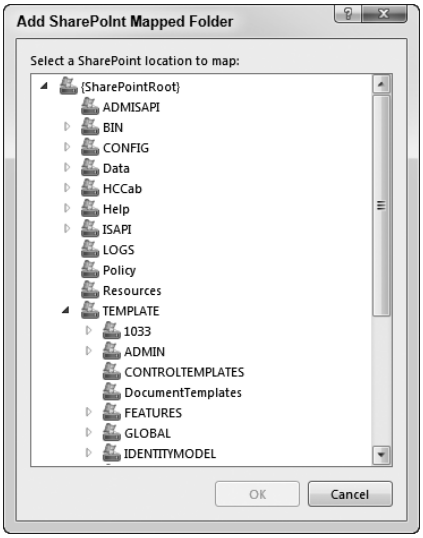


Рис. 24.9

Страницы приложения визуализируются с помощью страницы мастера SharePoint на этапе выполнения и поэтому содержат несколько элементов управления ASP.NET Content в качестве заполнителей для разных областей страницы мастера. Кроме редактирования кода, лежащего в основе страницы, разработчик может добавить на страницу приложения статический контент, стандартные элементы управления HTML и веб-элементы управления ASP.NET.

Как и для любого другого типа проекта, чтобы выполнить сборку и выполнение проекта в режиме Debug, необходимо нажать клавишу <F5>. Система Visual Studio автоматически упаковывает и развертывает страницу приложения на локальной платформе SharePoint, а затем открывает браузер на домашней странице сайта SharePoint. Для того чтобы увидеть страницу приложения, разработчик должен самостоятельно перейти на нее по адресу `http://ServerName/_layouts/ProjectName/MyPage.aspx` (рис. 24.10). Страницу приложения можно отлаживать точно так же, как любую другую форму на платформе ASP.NET.

Создание пользовательских компонентов SharePoint

В этом разделе мы рассмотрим процесс разработки некоторых из наиболее распространенных компонентов на платформе SharePoint.

Разработка веб-частей

В системе Visual Studio 2010 можно создать веб-части двух типов: ASP.NET Web Parts (известные также как Visual Web Parts) и SharePoint-based Web Parts.

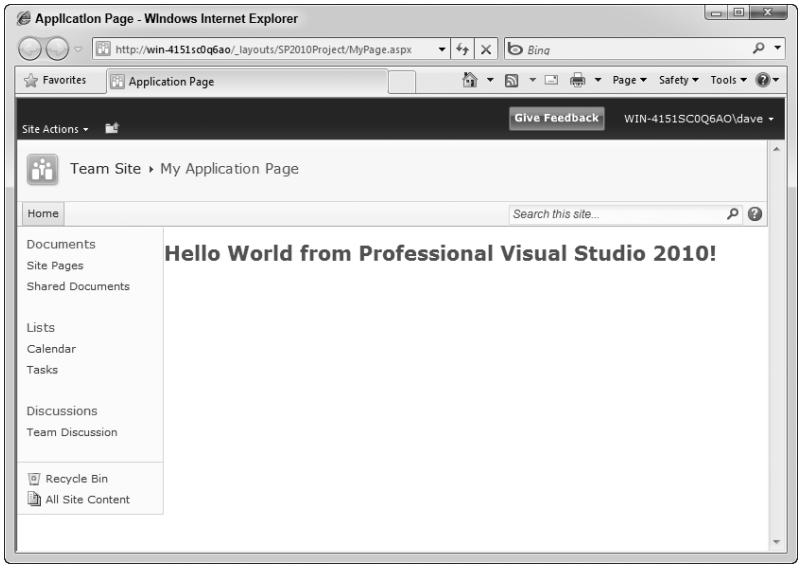


Рис. 24.10

Компоненты ASP.NET Web Parts представляют собой новинку для платформы SharePoint 2010 и являются производными от класса `System.Web.UI.WebControls.WebParts.WebPart`. Их можно использовать за пределами платформы SharePoint в любом веб-приложении на платформе ASP.NET, реализующем функциональные возможности компонентов ASP.NET Web Parts. Однако компоненты ASP.NET Web Parts нельзя использовать в ограниченном решении. Система Visual Studio 2010 включает в себя визуальный конструктор для веб-частей ASP.NET Web Parts.

Компоненты SharePoint-based Web Parts являются производными от класса `Microsoft.SharePoint.WebPartPages.WebPart`. Компоненты SharePoint-based Web Parts могут использоваться только на сайтах платформы SharePoint. В системе Visual Studio 2010 для них не предусмотрен визуальный конструктор. Вместо этого разработчик должен проектировать веб-части с помощью кода, замещая методы `CreateChildControls()` или `Render()`.

Компоненты ASP.NET Web Parts рекомендуются для разработки новых веб-частей. Для того чтобы создать новый компонент ASP.NET Web Part, необходимо щелкнуть правой кнопкой мыши в окне Solution Explorer и выбрать команду Add⇨New Item. Затем необходимо выбрать шаблон Visual Web Part, ввести имя `MyWebPart` и щелкнуть на кнопке Add.

Когда будет создана новая веб-часть, в проекте появится несколько файлов. Файл `MyWebPart.cs` (или `MyWebPart.vb`, если используется язык VB) представляет собой точку входа для веб-части. Когда на этапе выполнения загружается данная веб-часть, создается объект класса, содержащегося в этом файле. Файлы `Elements.xml` и `MyWebPart.webpart` — это манифестные XML-файлы, содержащие метаданные о веб-части для платформы SharePoint. Файл `MyWebPartUserControl.ascx` представляет собой пользовательский элемент управления для платформы .NET, обеспечивающий пользовательский интерфейс для веб-части. Именно здесь разработчик может настраивать компоновку и добавлять по необходимости веб-элементы управления и код.

Спроектировав веб-часть и добавив необходимую логику, следует собрать и выполнить проект. Система Visual Studio автоматически упаковывает и развертывает

веб-части для локального сайта SharePoint. Выбрав команду Site Actions⇨Edit Page, можно добавить веб-часть в существующую страницу на платформе SharePoint. Для этого необходимо щелкнуть на корешке закладки Insert на панели Ribbon и выбрать пункт Web Part, чтобы увидеть список доступных компонентов Web Parts. Разработанный пользователем компонент Web Part по умолчанию будет расположен в категории Custom (рис. 24.11).

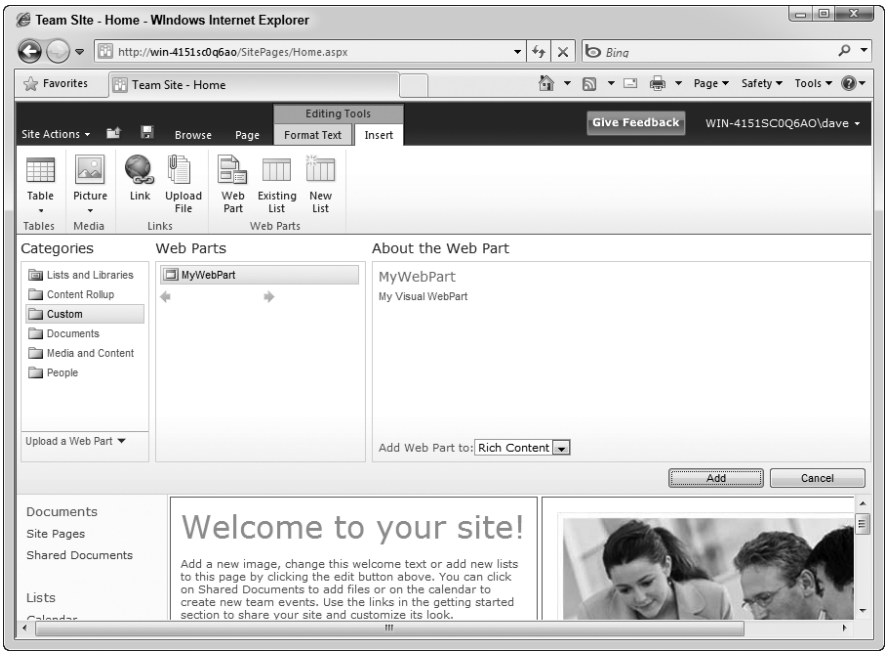


Рис. 24.11



Категорию, в которой появляется компонент Web Part, можно изменить, отредактировав файл `Elements.xml`.

Создание типов контента и списков

Типы контента и списки являются двумя основными строительными конструкциями платформы SharePoint. Они используются для реализации многих встроенных функциональных свойств.

Создайте новый пользовательский тип контента, щелкнув правой кнопкой мыши на проекте в окне Solution Explorer и выбрав команду Add⇨New Item. Выберите шаблон Content Type, введите имя `MyContentType` и щелкните на кнопке Add. В мастере SharePoint Customization Wizard выберите тип Task в качестве базового типа контента, от которого будет выполняться наследование, и щелкните на кнопке Finish. Система Visual Studio создаст пользовательский тип контента, который представляет собой простое XML-определение типа контента в файле `Elements.xml`.

Затем создайте пользовательское поле, которое может использоваться новым типом контента. В диалоговом окне Add New Item выберите новый компонент Empty

Element, введите имя Owner и щелкните на кнопке Add. Добавьте в узел <Elements> в файле Elements.xml следующую строку.

```
<Field ID="{3BA8B2E2-4BEA-4305-ACD2-9511C5E45738}"
       Type="User"
       Name="Owner"
       DisplayName="Task Owner">
</Field>
```

Теперь вернитесь назад к файлу Elements.xml для типа MyContentType. Добавьте в узел <FieldRefs> следующую строку на языке XML, чтобы пользовательское поле Owner стало доступным для нового типа контента.

```
<FieldRef ID="{3BA8B2E2-4BEA-4305-ACD2-9511C5E45738}" Name="Owner"/>
```

Создайте новый список определения SharePoint для этого типа контента. В окне Add New Item выберите новый компонент List Definition From Content Type, укажите имя MyCustomTasksList и щелкните на кнопке Add. Система Visual Studio вызовет мастер SharePoint Customization Wizard (рис. 24.12).

Введите имя компонента и убедитесь, что пользовательский тип контента, созданный ранее, есть в раскрывающемся списке. Также проверьте, что установлен флажок для добавления экземпляра списка.

В заключение нам необходимо настроить наследование списка так, чтобы на экране появлялся информативный заголовок. По умолчанию заголовком экземпляра списка является имя ProjectName-ListInstanceName. Откройте файл Elements.xml в экземпляре списка ListInstance1 и отредактируйте атрибут Title. Сохраните файл и нажмите клавишу <F5>, чтобы собрать и выполнить проект.

Когда откроется сайт SharePoint, в левой части страницы Home вы увидите новый список. Щелкните на нем, а затем на корешке закладки Items на панели Ribbon. Щелкните на кнопке New Item, чтобы открыть диалоговое окно New Item (рис. 24.13). Обратите внимание на то, что новое пользовательское поле находится в нижней части диалогового окна.



Рис. 24.12

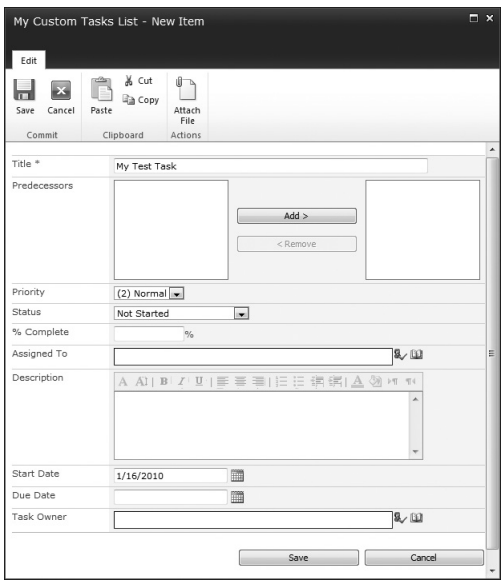


Рис. 24.13



Каждое пользовательское поле, созданное разработчиком, должно иметь уникальный идентификатор. Для того чтобы сгенерировать новый идентификатор GUID в системе Visual Studio, необходимо выбрать команду Tools⇨Create GUID.

Добавление получателей событий

Получателей событий можно добавлять во многие типы платформы SharePoint, включая списки, элементы списков, рабочие процессы, элементы коллекции Features и административные задания сайта SharePoint. В этом разделе мы добавим новый получатель событий в пользовательский список, созданный в предыдущем разделе.

Начните с выбора нового компонента Event Receiver в диалоговом окне Add New Item. Затем щелкните на кнопке Add. После этого на экране появится мастер SharePoint Customization Wizard (рис. 24.14). Выберите тип List Item Events Task в качестве типа получателя событий, а пользовательское задание — как источник события. Установите флажок An item was added и щелкните на кнопке Finish.

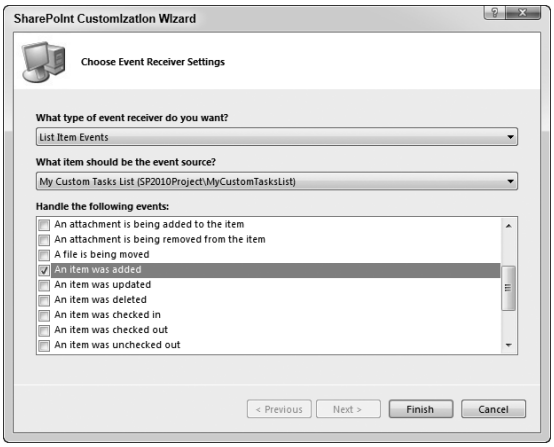


Рис. 24.14

Система Visual Studio создаст новый получатель события в виде класса, производного от базового класса `Microsoft.SharePoint.SPItemEventReceiver`. Метод `ItemAdded` можно заменить. Модифицируйте его, добавив следующий код, устанавливающий значение свойства Due Date нового задания равным пяти дням, считая от даты, определяемой значением свойства StartDate.



C#

```
public override void ItemAdded(SPItemEventProperties properties)
{
    var startDate = DateTime.Parse(properties.ListItem["Start
Date"].ToString());
    properties.ListItem["Due Date"] = startDate.AddDays(5);
    properties.ListItem.Update();
    base.ItemAdded(properties);
}
```

Сниппет кода MyEventReceiver.cs



VB

```
Public Overrides Sub ItemAdded(ByVal properties As SPItemEventProperties)
    Dim startDate = DateTime.Parse(properties.ListItem("Start
Date").ToString())
    properties.ListItem("Due Date") = startDate.AddDays(5)
    properties.ListItem.Update()
    MyBase.ItemAdded(properties)
End Sub
```

Сниппет кода MyEventReceiver.vb



Модифицируя файл определения списка `Schema.xml`, разработчик может настраивать многие аспекты списка, включая поля, которые должны визуализироваться по умолчанию.

При попытке собрать и выполнить проект может появиться сообщение о конфликте развертывания (рис. 24.15). Установите флажок, чтобы это сообщение больше не появлялось, и щелкните на кнопке Resolve Automatically.

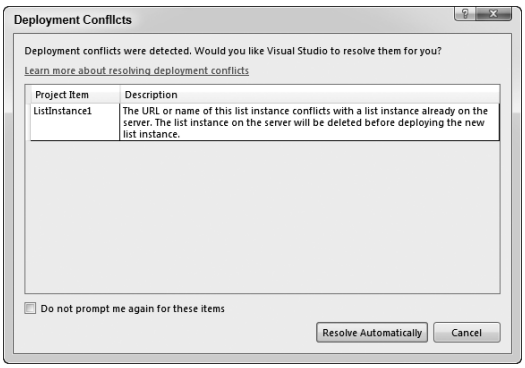


Рис. 24.15

Теперь, после того как вы добавили новое задание в пользовательский список заданий, при сохранении данного компонента свойство Due Date будет устанавливаться автоматически.

Создание рабочих процессов на платформе SharePoint

Система Visual Studio 2010 содержит рабочие процессы на платформе SharePoint двух типов: *последовательный рабочий процесс* (sequential workflow) и *рабочий процесс конечного автомата* (state machine workflow).

Последовательный рабочий процесс состоит из этапов, выполняемых по порядку. Например, предположим, что поступает документ, генерирующий электронное сообщение для верификатора. Верификатор открывает этот документ на платформе SharePoint и либо одобряет, либо отвергает его. Если документ одобрен, то он публикуется. Если отклонен, то отправителю документа отсылается электронное сообщение о причинах отказа.

Рабочий процесс конечного автомата состоит из набора состояний, переходов и действий. Разработчик определяет исходное состояние рабочего процесса, и он переходит в новое состояние в зависимости от произошедшего события. Например, рабочий процесс может иметь состояния `Document Created` и `Document Published`, а событиями, управляющими переходом в эти состояния, являются события `Document Submitted` и `Document Approved`.

Для того чтобы создать новый рабочий процесс на платформе SharePoint, необходимо щелкнуть правой кнопкой мыши на проекте в окне Solution Explorer и выбрать команду Add⇨New Item. Затем следует выбрать шаблон Sequential Workflow, ввести имя `MyWorkflow` и щелкнуть на кнопке Add.

Система Visual Studio запустит мастер SharePoint Customization Wizard. В первом окне мастера необходимо ввести осмысленное имя рабочего процесса и убедиться, что он имеет тип List Workflow (рис. 24.16).

В следующем окне необходимо указать автоматическую ассоциацию рабочего процесса, которая должна быть создана при запуске сеанса отладки. По умолчанию (рис. 24.17) рабочий поток ассоциируется с библиотекой документов `Shared Documents`. Оставьте эту настройку без изменения и щелкните на кнопке Next.

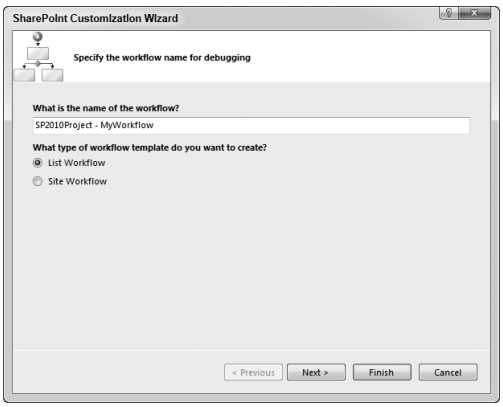


Рис. 24.16



Рис. 24.17

На последнем этапе работы мастера SharePoint CustomizationWizard необходимо указать, как будет начинаться рабочий процесс. Оставьте настройки, заданные по умолчанию (запускать вручную при создании элемента), и щелкните на кнопке Finish. Система Visual Studio создаст рабочий процесс и откроет его в визуальном конструкторе Workflow Designer (рис. 24.18).

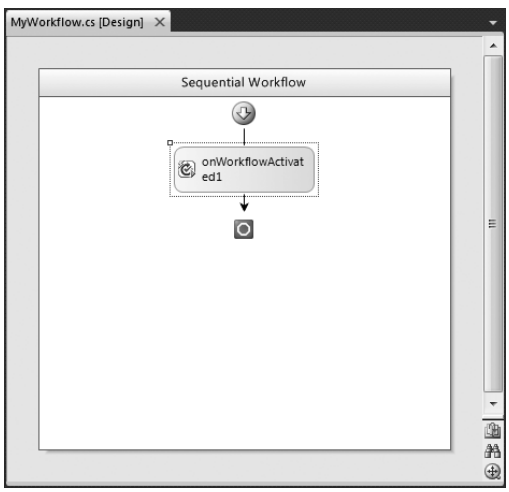


Рис. 24.18

Поскольку рабочие потоки на платформе SharePoint создаются на основе механизма Windows Workflow, мы не будет тратить время на объяснение способов настройки рабочих потоков. Вместо этого сошлемся на главу 32, в которой подробно описывается механизм Windows Workflow. Отметим лишь, что рабочие процессы на платформе SharePoint 2010 запускаются только при наличии версии 3.5 механизма Windows Workflow.

Для того чтобы протестировать рабочий процесс, его можно запустить на локальной платформе SharePoint. В ходе выполнения решения система Visual Studio автоматически упаковывает и развертывает рабочий процесс с заданными ассоциациями. В данном случае рабочий процесс будет вызван при добавлении в библиотеку `Shared Documents` нового документа. Рабочий процесс можно отладить, расставив точки прерывания в коде и проследив выполнение точно так же, как и любой проект в системе Visual Studio.

Работа с компонентами коллекции Feature

Компоненты Feature изначально были предназначены для администраторов серверов SharePoint и предоставляли им возможность управлять указанными элементами. Каждый раз при создании нового элемента в проект SharePoint добавляется компонент Feature.



Рис. 24.19

Компоненты Feature хранятся в узле Features пользовательского проекта SharePoint. Система Visual Studio содержит визуальный конструктор Feature Designer (рис. 24.19), который открывается, если дважды щелкнуть на компоненте Feature.

Визуальный конструктор Feature Designer позволяет задавать заголовок и описание компонента Feature, отображаемого в проекте SharePoint. Кроме того, пользователь может задать область видимости компонента Feature, ограничив ее всем сервером, всеми веб-сайтами в коллекции сайтов, конкретным веб-сайтом или всеми веб-сайтами в веб-приложении.

С помощью визуального конструктора Feature Designer можно включать элементы в компонент Feature и исключать их из него. Например, на рис. 24.19 показано, что все элементы платформы SharePoint в проекте, кроме элемента `MyWorkflow`, были включены в компонент Feature. Если в проекте есть несколько компонентов Feature, то между ними можно установить зависимости, чтобы гарантировать, что один компонент Feature не может быть активизирован, если не активизирован другой.

На платформе SharePoint администраторы могут активизировать и деактивизировать компоненты Feature, используя экраны администраторов Manage Site Features или Site Collection Features, которые открываются после выбора команды Site Actions⇨ Site Settings (рис. 24.20).

Упаковка и развертывание

Платформа SharePoint поддерживает формат пользовательской упаковки Windows SharePoint Package (WSP). WSP-файлы могут содержать компоненты Feature, определения сайтов, страницы шаблонов и приложений, а также дополнительные сборки. WSP-файлы создаются в папке `bin/debug` или `bin/release` при создании решения SharePoint в системе Visual Studio. Затем администратор может инсталлировать WSP-файл на удаленном сервере SharePoint.

При создании проекта SharePoint одновременно создается файл определения пакета, который находится в узле Packages. Определение пакета описывает, что должно входить в файл WSP. Система Visual Studio содержит визуальный конструктор Package Designer и инструментальное окно Packaging Explorer, которые облегчают

создание пакетов. Если дважды щелкнуть на файле пакета, то откроется файл и эти средства проектирования. На рис. 24.21 показан файл пакета, включающий в себя страницу приложения и один компонент Feature.

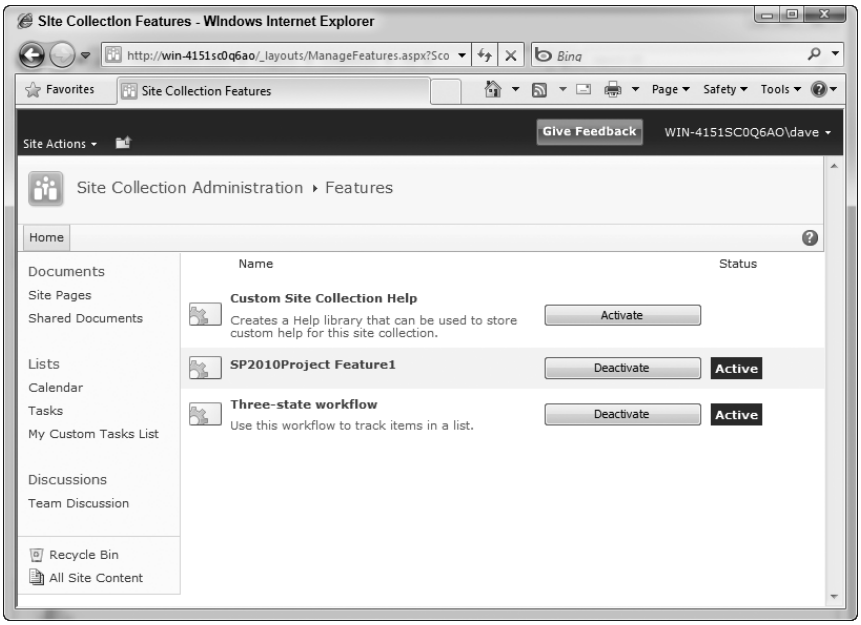


Рис. 24.20

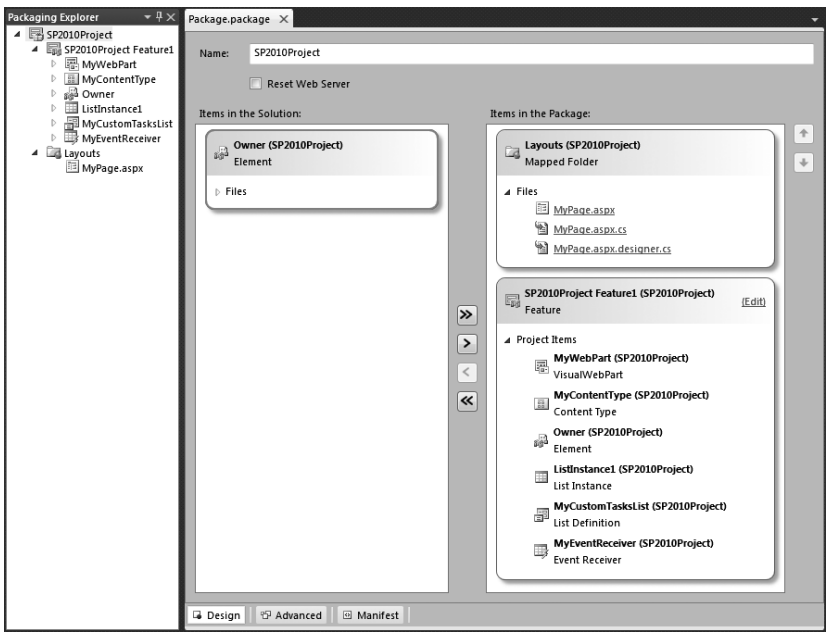


Рис. 24.21

Когда пользователь нажимает клавишу <F5>, открыв проект SharePoint, система Visual Studio экономит массу времени, автоматически разворачивая все элементы, существующие в пользовательском проекте, на локальной платформе SharePoint. Этапы развертывания регламентируются страницей свойств проекта SharePoint (рис. 24.22). Для того чтобы открыть эту страницу свойств, необходимо щелкнуть на проекте в окне Solution Explorer и выбрать команду Properties.

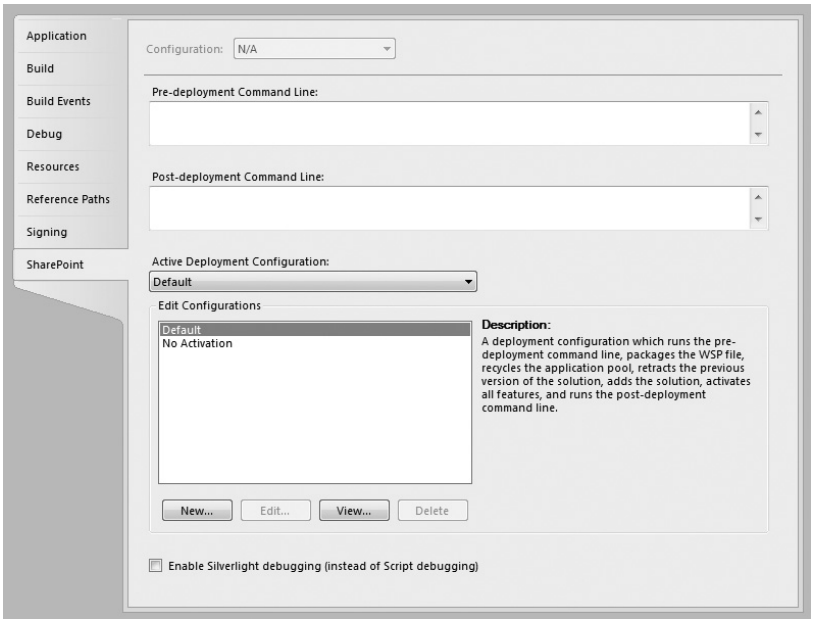


Рис. 24.22

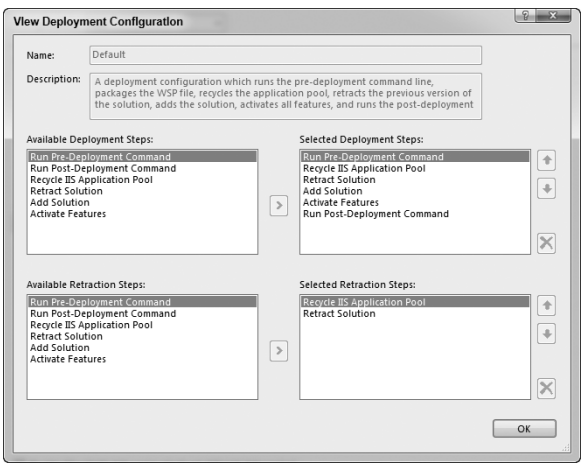


Рис. 24.23

Пользователь может задать программу, запускаемую из командной строки, или сценарий, которые должны выполняться до и после того, как система Visual Studio развернет решение на локальной платформе SharePoint. Сам процесс развертывания регламентируется конфигурацией развертывания. Для того чтобы увидеть этапы развертывания, необходимо дважды щелкнуть на конфигурации в списке Edit Configurations. Конфигурация развертывания, заданная по умолчанию, показана на рис. 24.23.

В заключение отметим, что, щелкнув правой кнопкой мыши на проекте в окне Solution Explorer и выбрав команду Retract, можно удалить компоненты платформы SharePoint из ее локальной инсталляции.

Резюме

В этой главе показано, как создать решения на платформе Microsoft SharePoint 2010. Инструменты для разработки приложений в системе Visual Studio 2010 позволяют легко разрабатывать веб-части, рабочие процессы, пользовательские списки и полные веб-приложения, которые выполняются в полнофункциональной среде SharePoint.

Эта глава содержит лишь поверхностное изложение того, что можно сделать с помощью платформы SharePoint 2010. Для более глубокого изучения этого вопроса рекомендуем посетить SharePoint Developer Center по адресу `http://msdn.microsoft.com/sharepoint`, веб-сайт SharePoint Dev Wiki по адресу `http://www.sharepointdevwiki.com` или прочитать книгу Tom Rizzo, Reza Alirezaei, Jeff Fried, and Paul Swider *Professional SharePoint 2010 Development.*

25

Платформа Windows Azure

В ЭТОЙ ГЛАВЕ...

- Принципы платформы Windows Azure
- Сборка, тестирование и развертывание приложений с помощью платформы Windows Azure
- Хранение данных в таблицах, массивах двоичных данных и очередях платформы Windows Azure
- Использование технологии SQL Azure
- Принципы технологии AppFabric

В последние годы компании Google, Amazon и множество других провайдеров, вышедших на рынок, приняли на вооружение технологию облачных вычислений. Компания Microsoft имеет свою собственную точку зрения на настольные, мобильные и серверные вычисления, позволяющую и независимым производителям, и самой Microsoft создавать превосходное программное обеспечение. Не вдаваясь в формальное определение *"облачных" вычислений* (cloud computing), отметим, что выбор такого метода для выполнения приложения объясняется множеством причин. К ним относится высокая степень доступности, способность масштабировать приложения с учетом требований заказчика и, конечно, уменьшение затрат.

Эта глава состоит из трех разделов, посвященных технологиям Windows Azure, SQL Azure и AppFabric. Платформа Windows Azure играет роль ведущего узла по отношению к приложению пользователя, позволяя динамически изменять количество параллельно выполняемых экземпляров. Она также предоставляет услуги по хранению данных в виде таблиц, массивов *двоичных данных* (blobs) и очередей. Технология SQL Azure обеспечивает полноценное обслуживание баз данных, хранящихся в "облаке". И наконец, технология AppFabric позволяет аутентифицировать пользователей, управлять доступом к приложению и службам, а также упрощать процесс предоставления службы из организации.

Платформа Windows Azure

Как и при работе с большинством технологий компании Microsoft, легче всего изучить платформы Windows Azure, создавая, собирая и выполняя новое приложение. При этом в диалоговом окне New Project появится новый узел, Cloud Service, содержащий отдельный проектный шаблон, также называемый Cloud Service (рис. 25.1).

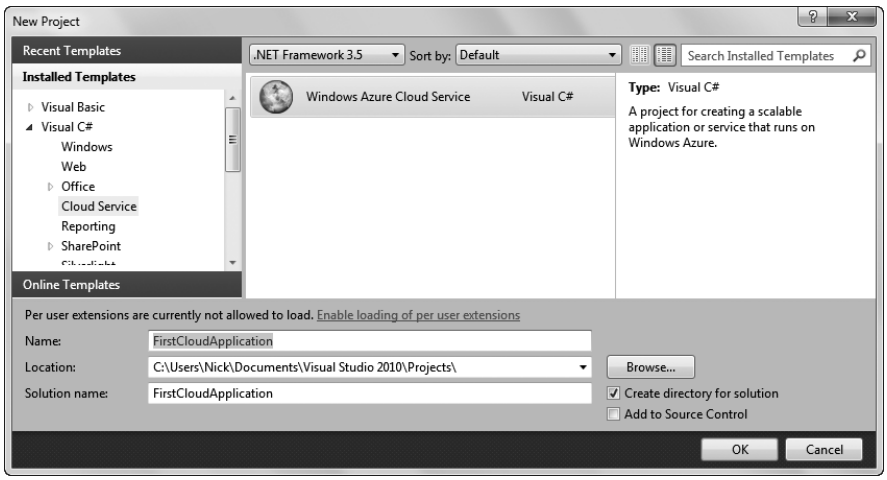


Рис. 25.1

После выбора проектного шаблона Cloud Service пользователь получает приглашение добавить в свое приложение одну или несколько ролей. Проект по технологии Azure может быть разделен на несколько разных ролей в зависимости от вида работы, которые они выполняют, и от того, принимают ли они какие-нибудь данные. Проще говоря, роли типа Web Role могут принимать данные, введенные пользователем через входящее соединение (например, http port 80), а роли типа Worker Role – нет. Типичный сценарий состоит из роли Web Role, используемой для получения данных. Эту роль может исполнять веб-сайт или веб-служба. Роль Web Role может передавать данные, например, с помощью очереди, роли Worker Role, которая их обрабатывает. Это разделение означает, что существуют два яруса, которые можно масштабировать независимо друг от друга, повышая гибкость приложения.

На рис. 25.2 показаны роли ASP.NET Web Role и Worker Role, добавленные в решение, выполняющее "облачные" вычисления. Для этого пользователь выбрал эти роли и щелкнул на кнопке со стрелкой, направленной вправо. Щелкнув на символе редактирования выбранной роли, можно переименовать ее, а затем щелкнуть на кнопке OK и тем самым завершить создание приложения.

Как показано на рис. 25.3, созданное приложение состоит из проектов, соответствующих каждой из выбранных ролей (`CloudFront` и `CloudWorker` соответственно), и дополнительного проекта `FirstCloudApplication`, определяющего список ролей и другую информацию для приложения Azure.

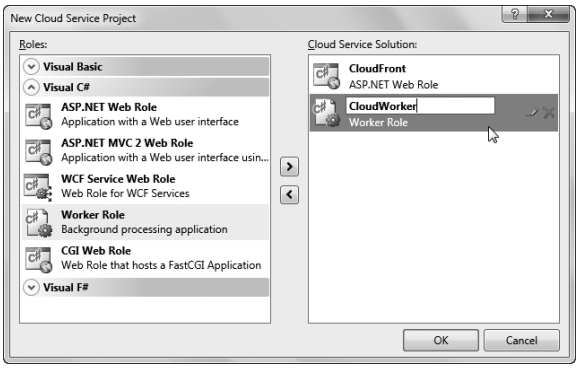


Рис. 25.2

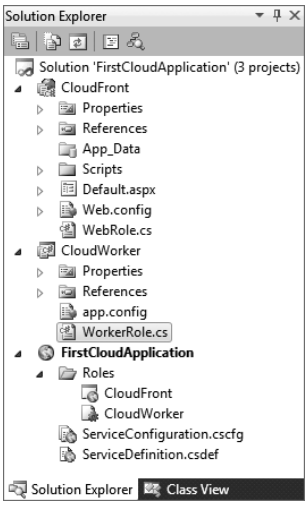


Рис. 25.3

Проект `CloudFront`, по существу, имеет тип ASP.NET Web Application. Если щелкнуть правой копкой мыши на этом проекте и выбрать команду Set as Startup Project, то можно запустить этот проект как любой обычный проект ASP.NET. С другой стороны, проект `CloudWorker` — это просто библиотека классов, содержащая единственный класс `WorkerRole` с точкой входа для рабочей роли.

Для того чтобы выполнить приложение по технологии Azure, следует убедиться, что проект `FirstCloudApplication` задан как `Startup Project`, а затем нажать клавишу <F5>, чтобы начать отладку. Если приложение по технологии Azure выполняется впервые, то откроется диалоговое окно, инициализирующее службу Development Storage. Этот процесс может занять минуту или две; после его завершения на панели задач Windows появятся две новые пиктограммы. Первая из них позволяет управлять функциями службы Development Storage, задавать три локальных службы, служащих отражением хранилищ платформы Azure для таблиц, массивов двоичных данных и очередей. Вторая позволяет следить за работой службы Development Fabric и представляет собой локальную копию среды для хостинга платформы Azure, в которой можно запускать, отлаживать и тестировать приложение.

После инициализации службы Development Storage в браузере открывается `Default.aspx` проекта `CloudFront`. Несмотря на то что пользователь видит единственный экземпляр браузера, на самом деле в среде симуляции Development Fabric одновременно будут выполняться несколько экземпляров веб-роли.

Среда симуляции Development Fabric

Проект `FirstCloudApplication` содержит два файла, определяющих атрибуты приложения для платформы Azure. Первый файл, `ServiceDefinition.csdef`, определяет структуру и атрибуты ролей, образующих приложение. Например, если одна из ролей должна что-то записывать в файловую систему, то можно задать свойство `LocalStorage`, предоставив роли ограниченный доступ к небольшой части диска, в которую она может записывать и откуда может считывать временные файлы. В этом

файле также определяются все настройки, требующиеся роли на этапе выполнения. Определение настроек позволяет повысить адаптивность ролей на этапе выполнения без их повторной сборки и публикации.

Второй файл, `ServiceConfiguration.cscfg`, определяет конфигурацию ролей на этапе выполнения. Он содержит множество экземпляров каждой роли, которые должны быть выполнены, а также все настройки, определенные в файле `ServiceDefinition`. Если пользователь модифицирует значения в этом файле конфигурации, например, изменив атрибут счетчика элементов `Instances` на четыре для обеих ролей, а затем повторно запустит свое приложение, то оно будет выполнено в новой конфигурации в локальной среде симуляции Development Fabric.

Если щелкнуть правой кнопкой мыши на пиктограмме Development Fabric панели задач Windows и выбрать команду Show Development Fabric UI, то можно увидеть иерархическое представление выполняемых приложений в среде Development Fabric (рис. 25.4). Погружаясь в структуру элементов развертывания, можно увидеть приложение `FirstCloudApplication` и две роли — `CloudFront` и `CloudWorker`.

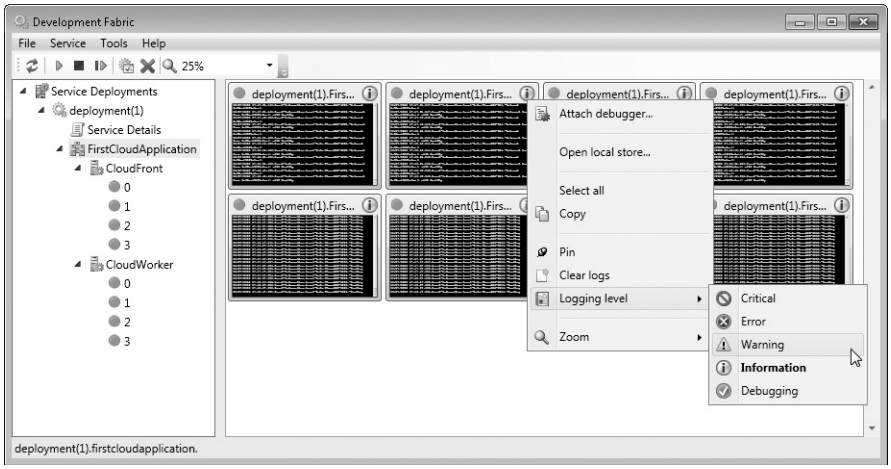


Рис. 25.4

Каждая из этих ролей состоит из множества экземпляров (зеленые точки). На рис. 25.4 их четыре. На правой панели можно увидеть *записи журнала* (log output) для каждого из выполняемых экземпляров. Щелкнув на строке заголовка любого из этих экземпляров, можно развернуть окно для вывода записей журнала на весь экран. Пиктограмма в правом верхнем углу окна для каждого экземпляров означает уровень сообщений, которые могут быть занесены в журнал (logging level). Его можно изменить, щелкнув правой кнопкой мыши на заголовке и выбрав желательный уровень из меню Logging Level.

Хранилище таблиц, массивов двоичных данных и очередей

До сих пор мы рассматривали веб-роль без содержимого и рабочую роль, которая ничего не делала. Для того чтобы добавить содержимое для веб-роли, достаточно добавить на страницу `Default.aspx` элементы управления, точно так же, как это делается в обычных веб-приложениях. Начнем с добавления поля ввода `JobDetailsText`

и кнопки SubmitJob. Дважды щелкните на кнопке, чтобы открыть соответствующий файл кода.

Данные между веб-ролями и рабочими ролями можно передавать, записывая их в таблицу (структурированные данные), в массивы двоичных данных (отдельные бинарные объекты) или очереди (сообщения). Для работы с этим хранилищем на платформе Azure предназначен интерфейс REST. Однако для платформы .NET этот интерфейс не слишком удобен. К счастью, разработчики платформы Azure создали оболочку для этих функциональных возможностей, которая значительно облегчает использование хранилища Windows Azure. Если посмотреть на ссылки в проектах Web Role и Worker Role, то можно увидеть ссылку на библиотеку Microsoft.WindowsAzure.Storage.dll, содержащую классы оболочки и методы, которые можно использовать в своем приложении.

В исходный файл для страницы Default.aspx необходимо внести следующий код, получающий ссылку на очередь, а затем добавляющий простое сообщение. Обратите внимание на то, что в некоторых местах файла может понадобиться вставить инструкции using.



C#

```
protected void SubmitJob_Click(object sender, EventArgs e){
    // считываем установки конфигурации, соответствующие учетной записи
    var storageAccount = CloudStorageAccount.
                         FromConfigurationSetting("DataConnectionString");
    // создаем очередь для взаимодействия с рабочей ролью
    var queueStorage = storageAccount.CreateCloudQueueClient();
    var queue = queueStorage.GetQueueReference("sample");
    queue.CreateIfNotExist();
    queue.AddMessage(new CloudQueueMessage(this.JobDetailsText.Text));
}
```

Сниппет кода Default.aspx.cs



VB

```
Protected Sub SubmitJob_Click(ByVal sender As Object,
                              ByVal e As EventArgs) Handles SubmitJob.Click
    ' считываем установки конфигурации, соответствующие учетной записи
    Dim storageAccount = CloudStorageAccount.
                         FromConfigurationSetting("DataConnectionString")
    ' создаем очередь для взаимодействия с рабочей ролью
    Dim queueStorage = storageAccount.CreateCloudQueueClient()
    Dim queue = queueStorage.GetQueueReference("sample")
    queue.CreateIfNotExist()
    queue.AddMessage(New CloudQueueMessage(Me.JobDetailsText.Text))
End Sub
```

Сниппет кода Default.aspx.vb

Этот код получает значение, введенное в поле ввода JobDetailsText, и добавляет его в очередь в виде сообщения.

Кроме того, необходимо изменить рабочую роль, чтобы выталкивать сообщение из очереди и выполнять соответствующие действия. Следующий код находит следующее сообщение в очереди и записывает ответ в журнал, прежде чем удалить сообщение из очереди. Если сообщение не удаляется из очереди, оно заталкивается обратно в очередь после некоторой задержки, которой можно управлять, чтобы гарантировать, что все сообщения были обработаны хотя бы один раз, даже если рабочая роль прервала обработку.



C#

```
private CloudQueue queue;
public override bool OnStart(){

    DiagnosticMonitor.Start("DiagnosticsConnectionString");

    Microsoft.WindowsAzure.CloudStorageAccount.
             SetConfigurationSettingPublisher((configName, configSetter) => {

configSetter(Microsoft.WindowsAzure.ServiceRuntime.RoleEnvironment.
                GetConfigurationSettingValue(configName));
                });

    Trace.TraceInformation("Worker entry point called");

    // считываем установки конфигурации, соответствующие учетной записи
    var storageAccount = CloudStorageAccount.
           FromConfigurationSetting("DataConnectionString");
           // создаем очередь для взаимодействия с веб-ролью
           var queueStorage = storageAccount.CreateCloudQueueClient();
           var queue = queueStorage.GetQueueReference("sample");
           queue.CreateIfNotExist();
           return base.OnStart();
}

public override void Run(){
    Trace.TraceInformation("CloudWorker entry point called");
    while (true){
         try{
              // Выталкиваем следующее сообщение из очереди
              CloudQueueMessage msg = queue.GetMessage();
              if (msg != null){
                  // Анализируем содержание сообщения как описание задания
                  string jd = msg.AsString;
                  Trace.TraceInformation("Processed {0}", jd);
                  // Удаляем сообщение из очереди
                  queue.DeleteMessage(msg);
             }
             else{
                  Thread.Sleep(10000);
         }
         Trace.TraceInformation("Working");
    }
    catch (Exception ex){
    Trace.TraceError(ex.Message);
    }
  }
}
```

Сниппет кода WorkerRole.cs



VB

```
Private queue As CloudQueue
Public Overrides Function OnStart() As Boolean
    DiagnosticMonitor.Start("DiagnosticsConnectionString")

    CloudStorageAccount.SetConfigurationSettingPublisher(
               Function(configName, configSetter)
                     configSetter(RoleEnvironment.
                         GetConfigurationSettingValue(configName)))
    Trace.TraceInformation("Worker entry point called")
```

```
        ' считываем установки конфигурации, соответствующие учетной записи
        Dim storageAccount = CloudStorageAccount.
                          FromConfigurationSetting("DataConnectionString")

        ' создаем очередь для взаимодействия с веб-ролью
        Dim queueStorage = storageAccount.CreateCloudQueueClient()
        queue = queueStorage.GetQueueReference("sample")
        queue.CreateIfNotExist()
        Return MyBase.OnStart()
    End Function

    Public Overrides Sub Run()
        Trace.TraceInformation("CloudWorker entry point called.")
        Do While (True)
            Try
                ' Выталкиваем следующее сообщение из очереди
                Dim msg As CloudQueueMessage = queue.GetMessage()
                If (msg IsNot Nothing) Then
                    ' Анализируем содержание сообщения как описание задания
                    Dim jd As String = msg.AsString
                    Trace.TraceInformation("Processed {0}", jd)
                    ' Удаляем сообщение из очереди
                    queue.DeleteMessage(msg)
                Else
                    Thread.Sleep(10000)
                End If
                Trace.TraceInformation("Working")
            Catch ex As StorageClientException
                Trace.TraceError(ex.Message)
            End Try
        Loop
    End Function
```

Сниппет кода WorkerRole.vb

В этом коде замещаются два метода: `OnStart` и `Run`. Первый метод используется для загрузки конфигурации и задания локальных переменных, необходимых для работы с хранилищем Windows Azure, в то время как метод `Run` содержит бесконечный цикл, продолжающий выталкивать сообщения из очереди. Перед тем как запустить модифицированные роли, необходимо задать местоположение хранилища для очереди, которое будет использоваться. Несмотря на то что в конечном счете для этого будет использоваться учетная запись хранилища, разработчик должен указать информацию о хранилище Development Storage. Для этого следует внести изменения в файл `ServiceConfiguration`.

```
<?xml version="1.0"?>
<ServiceConfiguration serviceName="FirstCloudApplication"
xmlsn="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration">
<Role name="CloudFront">
<Instances count="2" />
<ConfigurationSettings>
<Setting name="DataConnectionString" value="UseDevelopmentStorage=true" />
<Setting name="DiagnosticsConnectionString" value="UseDevelopmentStorage=true" />
<!-- <Setting name="DeploymentConnectionString" value="DefaultEndpointsProtocol=
https;AccountName=[YOUR_ACCOUNT_NAME];AccountKey=[YOUR_ACCOUNT_KEY]" /> -->
</ConfigurationSettings>
</Role>
<Role name="CloudWorker">
```

```
<Instances count="2" />
<ConfigurationSettings>
<Setting name="DataConnectionString" value="UseDevelopmentStorage=true" />
<Setting name="DiagnosticsConnectionString" value="UseDevelopmentStorage=true" />
</ConfigurationSettings>
</Role>
</ServiceConfiguration>
```

Обратите внимание на то, что в этом коде для ролей `CloudWorker` и `CloudFront` определены настройки `DataConnectionString` и `DiagnosticsConnectionString`. В данном случае установлено значение, соответствующее использованию учетной записи среды Development Storage. При развертывании приложения на платформе Windows Azure необходимо заменить эту настройку строкой, включающей в себя имя учетной записи и ключ в формате, проиллюстрированном строкой `DeploymentConnectionString`. Для того чтобы эти значения стали доступными для ролей, необходимо также изменить файл `ServiceDefinition`, чтобы указать, какие настройки определены для каждой роли.

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceDefinition name="FirstCloudApplication"
xmlsn="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
<WebRole name="CloudFront" enableNativeCodeExecution="false">
<InputEndpoints>
<!-- Must use port 80 for http and port 443 for https when running in the cloud -->
<InputEndpoint name="HttpIn" protocol="http" port="80" />
</InputEndpoints>
<ConfigurationSettings>
<Setting name="DataConnectionString" />
<Setting name="DiagnosticsConnectionString" />
</ConfigurationSettings>
</WebRole>
<WorkerRole name="CloudWorker" enableNativeCodeExecution="false">
<ConfigurationSettings>
<Setting name="DataConnectionString" />
<Setting name="DiagnosticsConnectionString" />
</ConfigurationSettings>
</WorkerRole>
</ServiceDefinition>
```

Внеся эти изменения, попробуем запустить наше приложение на платформе Azure. Обратите внимание на то, что после щелчка на кнопке Submit на экране появится сообщение “Processed” в одном из выполняющихся экземпляров рабочей роли в среде Development Fabric UI.

Развертывание приложения

Приложение Azure, созданное с помощью среды симуляции Development Fabric и хранилища Development Storage, можно развернуть на платформе Windows Azure. Однако перед этим необходимо создать учетные записи для служб хостинга и хранения. Зайдите на веб-сайт `http://www.azure.com` и зарегистрируйтесь, используя свои учетные данные на платформе Windows Azure. После регистрации щелкните на гиперссылке Go to the Windows Azure Developer portal. Откроется портал Windows Azure, который выглядит примерно так, как показано на рис. 25.5.

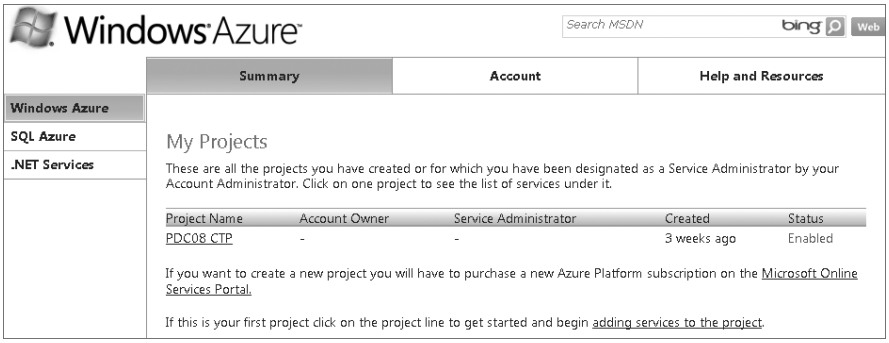


Рис. 25.5

Щелкните на имени проекта, затем на кнопке New Service и выберите тип службы, который хотите добавить. Для приложения `FirstCloudApplication` необходим и хостинг, и хранилище, поэтому добавьте по одному экземпляру того и другого. Добавив службу `Storage Account`, вы увидите экран конфигурации (рис. 25.6).

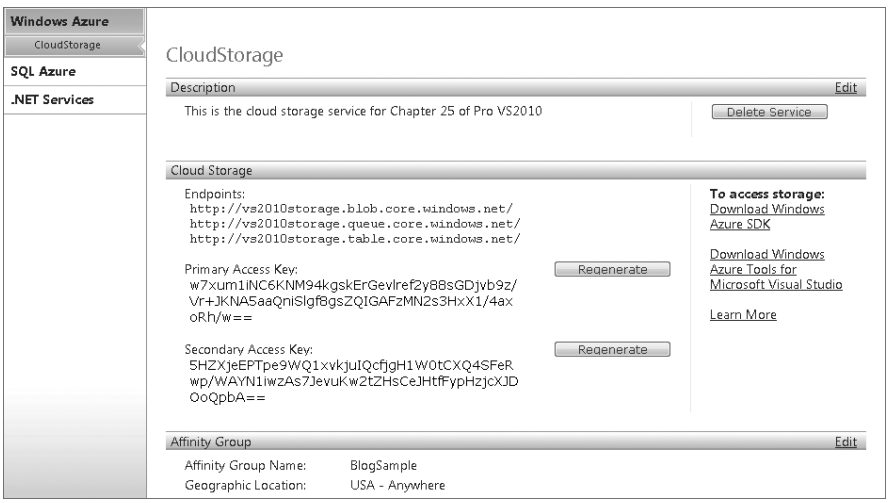


Рис. 25.6

Теперь необходимо скопировать учетную информацию и конечную точку хранилища в файл `ServiceConfiguration`. После этого следует снова запустить приложение на платформе Azure. На этот раз она будет выполняться в локальной среде симуляции Development Fabric, но используя хранилище Azure, а не Development Storage. Это хороший тест для проверки того, что приложение правильно развернуто в “облаке”.

Задав учетную запись для службы хостинга, вы увидите экран, похожий на рис. 25.7.

На рис. 25.7 показано, что на самом деле в нашем распоряжении есть две среды для развертывания приложения: `Production` и `Staging`. В соответствии с хорошо продуманной стратегией развертывания платформа Azure поддерживает развертывание в среде `Staging`, а если пользователь чувствует себя удобно, переходит в среду `Production`.

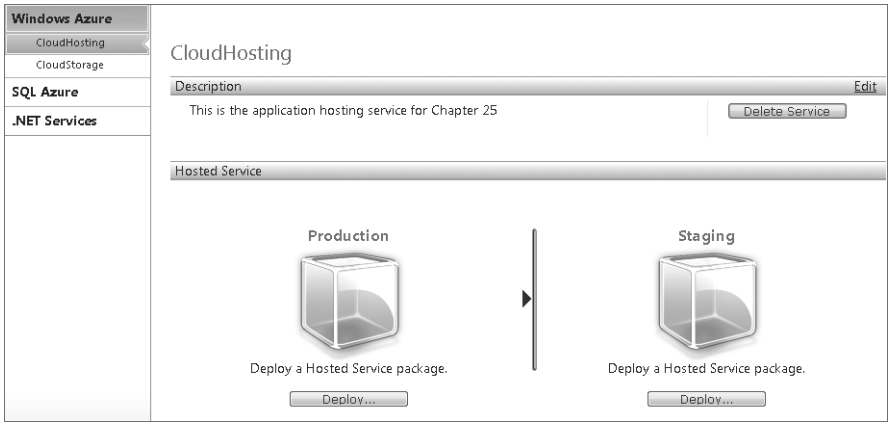


Рис. 25.7

Вернитесь к системе Visual Studio 2010, щелкните правой кнопкой мыши на проекте `FirstCloudApplication` и выберите команду `Publish`. Она выполняет сборку приложения, а также генерирует пакет развертывания и файл конфигурации. После завершения процесса они отображаются в диалоговом окне Windows Explorer. Вернитесь к платформе Windows Azure и щелкните на кнопке Deploy, расположенной под узлом Staging (см. рис. 25.7). Система предложит выбрать пакет развертывания и файл конфигурации. Закончив загрузку, вернитесь на страницу службы хостинга, где вы увидите обновленную среду Staging (рис. 25.8).

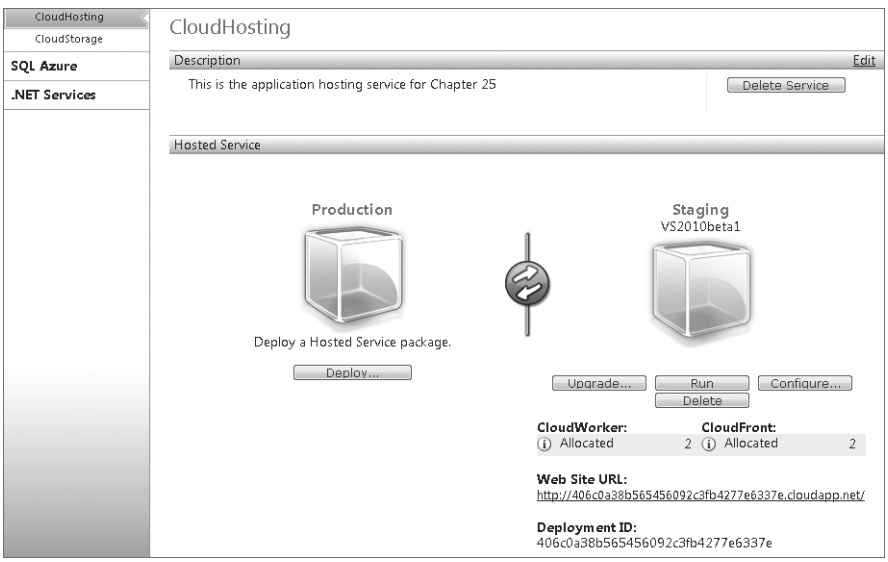


Рис. 25.8

В отличие от веб-приложений на платформе ASP.NET, которые автоматически выполняются сразу после полного развертывания, приложения на платформе Azure необходимо запускать. Для этого следует щелкнуть на кнопке Run. Экран обновится и будет содержать

сообщение об инициализации всех ролей. В конце концов, появится сообщение `Started` о том, что все роли готовы принимать данные и выполнять свои задания.

На последнем этапе выполняется продвижение из среды `Staging` в среду `Production`. Слово "продвижение" имеет важный смысл, так как этот переход выполняется интеллектуальным маршрутизатором. Поскольку переход из одной среды в другую происходит достаточно быстро (в зависимости от того, как быстро маршрутизатор внесет изменения), почти мгновенно, у пользователя нет возможности перехватить ошибку. Для того чтобы выполнить переход из среды `Staging` в среду `Production`, необходимо щелкнуть на круглой кнопке с вращающимися стрелками, расположенной между областями Staging и Production на портале Azure.

Настройка приложения

Со временем требования, предъявляемые к приложению, могут измениться, и возникнет необходимость внести изменения в настройки приложения, записанные в файле `ServiceConfiguration`. Это можно выполнить в динамическом режиме, щелкнув на кнопке Configure (в области Staging на рис. 25.8). На рис. 25.9 показан экран конфигурации, на котором можно изменить XML-конфигурацию или загрузить альтернативный файл конфигурации. Используя этот метод, следует изменять лишь настройки развертывания в среде `Staging`, поскольку они не должны повлиять на настройки в среде `Production`. Рекомендуется начинать с идентичного развертывания в средах `Production` и `Staging`, изменить настройки в среде `Staging`, позволить их инициализацию и запуск приложения, а затем перейти в среду `Production`. После этого можно модифицировать второе развертывание в соответствии с первым.

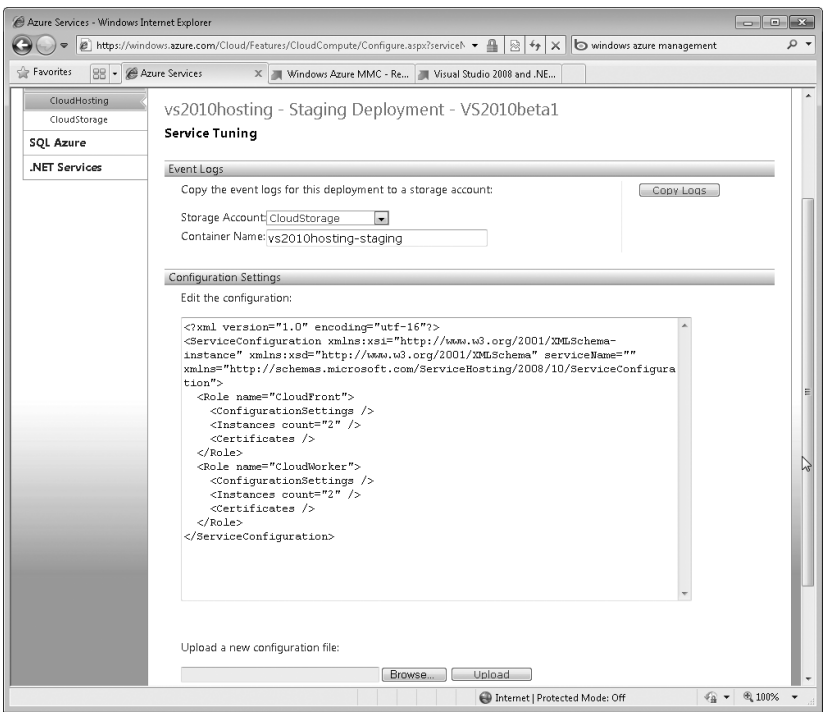


Рис. 25.9

Этот экран позволяет также экспортировать в учетную запись хранилища журналы, сгенерированные ролями в приложении. После копирования этих журналов разработчик должен извлечь их из соответствующей учетной записи хранилища. Для этого можно использовать пример `CloudDrive` в пакете Azure SDK, позволяющий отображать учетную запись хранилища в виде локального дисковода, к которому можно обращаться в программе Powershell.

Технология SQL Azure

Кроме хранилища таблиц, массивов двоичных данных, платформа Windows Azure реализует полноценный хостинг реляционных данных в виде технологии SQL Azure. Каждую базу данных SQL Azure можно рассматривать как внедренный экземпляр базы данных SQL Server 2008, который функционирует в режиме широкого доступа. Это означает, что в любой момент времени существуют три синхронизированных экземпляра пользовательской базы данных. Если один из этих экземпляров даст сбой, то немедленно будет подключен новый экземпляр и данные будут синхронизированы, чтобы гарантировать их доступность.

Для создания базы данных SQL Azure необходимо зарегистрироваться на портале Windows Azure и перейти на корешок закладки SQL Azure. Здесь пользователь может управлять своей учетной записью SQL Azure, а также создавать и удалять базы данных. После создания базы данных пользователь может извлечь строку соединения, необходимую для связи с базой данных, выбрав базу данных и щелкнув на кнопке Connection String (рис. 25.10).

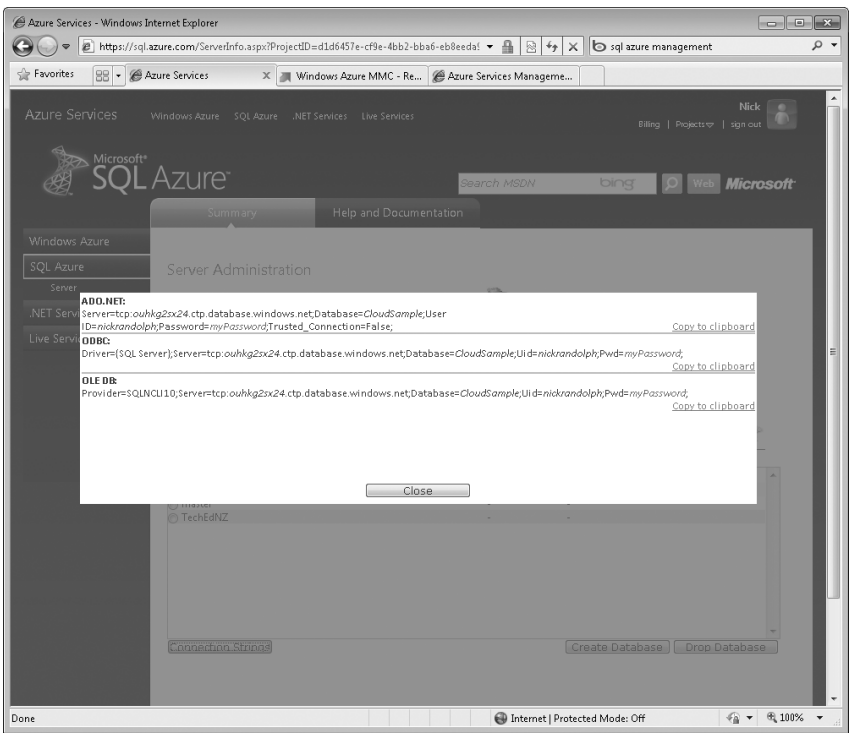


Рис. 25.10

На момент написания данной книги существовало множество способов взаимодействия с базами данных SQL Azure. Несмотря на то что технология SQL Azure основана на технологии SQL Server 2008, существует большое количество ограничений на работу большинства графических средств, таких как SQL Server Management Studio (Object Browser) и Visual Studio 2010. Тем не менее можно соединить окно запросов утилиты SQL Server Management Studio с базой данных SQL Azure и применить инструкции T-SQL к своей базе данных. Для работы с платформой SQL Azure можно также использовать некоторые инструменты, разработанные сторонними поставщиками, а также надстройки системы Visual Studio 2010, такие как SQL Azure Migration Wizard, SQL Azure Manager и SQL Azure Explorer.

Из своего приложения пользователь может соединиться с платформой SQL Azure, используя строку соединения, извлеченную из страницы портала Windows Azure. При этом для создания, обновления, чтения и удаления данных из пользовательской базы данных SQL Azure можно использовать разнообразные технологии, созданные на основе технологии ADO.NET, например LINQ to SQL, Entity Framework, или саму технологию ADO.NET.

Технология AppFabric

Третьим компонентом платформы Azure Platform является технология AppFabric, которая в свою очередь состоит из средств Service Bus и Access Control Service. В среде, где организации все больше стремятся внедрить некоторые или все свои приложения в "облако", основными вопросами становятся обеспечение соединения и безопасности. Технология AppFabric предлагает решение для соединения приложений и унификации их систем безопасности.

Средство Service Bus

Несмотря на то что большинство организаций имеют выход в Интернет, связь между офисами или между отдельными лицами, находящимися в пути, часто вызывает нарекания. Все чаще компании функционируют под защитой одного или нескольких брандмауэров, которые не только ограничивают трафик, но и выполняют перевод сетевых адресов. Это значит, что к компьютерам, защищенным этими брандмауэрами, нелегко обратиться из-за пределов сети компании. Кроме того, по мере уменьшения количества открытых адресов IPv4 все больше соединений имеют динамические IP-адреса. Это делает внедрение приложений в сети компании с открытым доступом практически невозможной.

Средство Service Bus позволяет регистрировать службу на специальном публично адресуемом указателе URL с помощью регистра служб. Для этого запрос, посланный этому указателю URL, направляется службе через существующее исходящее соединение, созданное самой службой. Работать с компонентом Service Bus также просто, как изменять свои существующие сопряжения WCF на новые релейные сопряжения. Будучи частью выполняемой службы, он регистрируется в регистре служб и инициирует исходящее соединение, требуемое для всех дальнейших коммуникаций.

Средство Access Control Service

Если организация хочет интегрировать многочисленные "облачные" приложения и/или местные резервные приложения, возникает необходимость как-то следить за тем, кто имеет доступ (аутентификация) к конкретным ресурсам (авторизация). Эту функцию выполняет средство Access Control Service (ACS). Несмотря на свою относительную незрелость, компонент ACS способен верифицировать пользователей путем проверки правильности их входной информации, ее перевода и отправки выходной информации конкретным приложениям. Например, пользователь может зарегистрироваться в приложении, указав свой электронный адрес и пароль. Эта входная информация может быть использована для его аутентификации, а также для того, чтобы обнаружить, что он принадлежит группе любителей приложения ABC, к которому он пытается получить доступ. Выходная информация может состоять из электронного адреса пользователя и названия группы любителей приложения. Обратите внимание на то, что, поскольку между приложением ABC и компонентом ACS заранее установлены доверительные отношения (проверенные с помощью подписи выходной информации), приложение ABC может доверять выходной информации.

Резюме

В этой главе мы рассмотрели платформу Azure Platform и показали, как она обеспечивает вход компании Microsoft в пространство "облачных" вычислений. Используя систему Visual Studio 2010, можно адаптировать существующее или создать новое приложение или службу для внедрения в "облако". Локальное хранилище Development Storage и среда Fabric представляют собой отличное решение для тестирования. Это означает, что при публикации своего приложения на платформе Windows Azure пользователь может быть уверенным в том, что оно будет работать без больших проблем.

Даже если пользователь не желает внедрять свое приложение в "облако", он может использовать технологию SQL Azure и AppFabric для хранения своих данных, решения проблем со связью и унификации систем безопасности.

ЧАСТЬ VI

Данные

- **ГЛАВА 26.** Инструменты Visual Database Tools
- **ГЛАВА 27.** Объекты Dataset и DataBinding
- **ГЛАВА 28.** Технология Language Integrated Queries (LINQ)
- **ГЛАВА 29.** Технология ADO.NET Entity Framework
- **ГЛАВА 30.** Создание отчетов

26

Инструменты Visual Database Tools

В ЭТОЙ ГЛАВЕ...

- Принципы работы информационно-ориентированных инструментальных окон в системе Visual Studio 2010
- Создание и проектирование баз данных
- Навигация по источникам данных
- Ввод и просмотр данных с помощью системы Visual Studio 2010

Связь с базами данных является существенным фактором практически во всех приложениях, независимо от того, является ли оно программой для системы Windows-based, веб-сайтом или сервисом. Первая версия системы Visual Studio .NET предоставила разработчикам огромное множество возможностей для просмотра файлов баз данных в своих файловых системах и локальных серверах, окно Server Explorer, элементы управления для работы с данными и компоненты для связи с данными. Технология .NET Framework, лежавшая в основе этой системы, содержала переработанную модель доступа к базам данным ADO.NET (database engine), полнее соответствующую современным методам создания приложения.

Эти функциональные возможности были включены и в систему Visual Studio 2010. При этом были устранены дефекты, добавлены инструменты и новые возможности интегрированной среды разработки, обеспечивающие более прямой доступ к данным из пользовательского приложения. В главе рассказывается, как создать и обрабатывать данные с помощью разнообразных инструментальных окон в системе Visual Studio 2010. В совокупности эти инструменты называются Visual Database Tools.

Окна для работы с базами данных в системе Visual Studio 2010

Большое количество окон в системе Visual Studio 2010 предназначено специально для работы с базами данных и их компонентами. Пользователь может найти большинство необходимых инструментов, не выходя из интегрированной среды разработки: от окна Data Sources, показывающего файлы данных, связанных с проектом, и узла Data Connections в окне Server Explorer до редактора Database Diagram Editor и визуального проектировщика схем баз данных. Трудно себе представить такой инструмент для работы с базами данных, которого не было бы в системе Visual Studio.

На рис. 26.1 показан сеанс редактирования базы данных с помощью интегрированной среды разработки Visual Studio 2010. Обратите внимание на то, как изменяются окна, инструментальные панели и меню, чтобы обеспечить согласованность с конкретным контекстом редактирования таблицы базы данных. В главной области расположен список столбцов, принадлежащих таблице, его сопровождает меню Table Designer, а ниже расположена область редактирования Column Properties. Обычное инструментальное окно Properties содержит свойства текущей таблицы. Ниже мы рассмотрим каждое из этих окон и опишем их назначение, чтобы вы могли работать с ними как можно эффективнее.

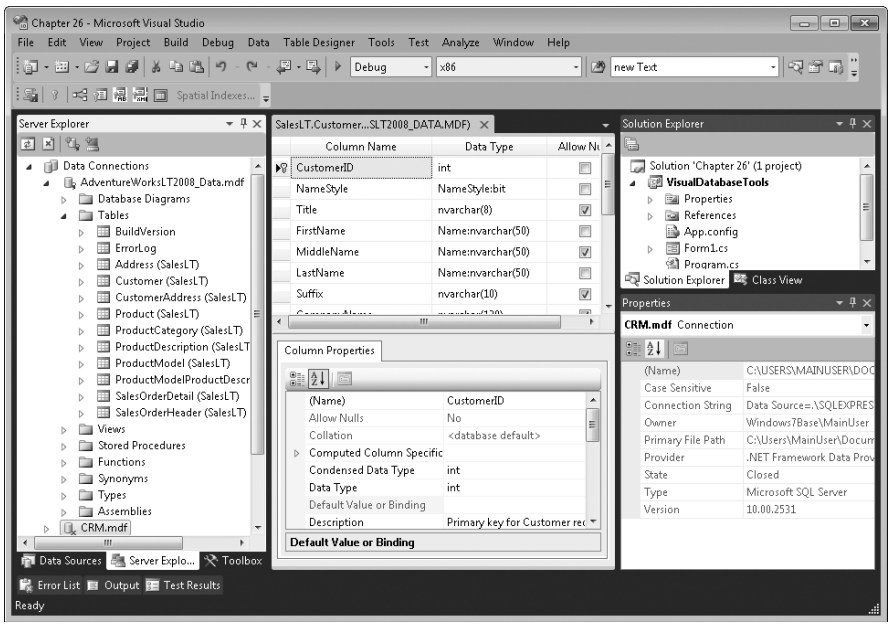


Рис. 26.1

Окно Server Explorer

В главе 12 рассказывалось о том, как можно использовать окно Server Explorer для навигации по компонентам, образующим систему (и, вообще, по компонентам, с которыми существует связь). Один из компонентов — Data Connections — был в той главе

оставлен вне поля зрения. С помощью этого узла система Visual Studio 2010 обеспечивает довольно большое количество функциональных возможностей, характерных для других программ, например SQL Server Management Studio, для создания и модифицирования баз данных.

На рис. 26.1 показано окно Server Explorer, в котором показаны узел, соответствующий активному соединению с базой данных (`AdventureWorksLT2009_Data.mdf`), и узел, соответствующий другой базе данных, которая еще не соединена с системой Visual Studio (`CRM.mdf`). Пиктограмма базы данных показывает, является ли соединение с ней активным, и содержит большое количество дочерних узлов, относящихся к типичным компонентам современных баз данных, например Tables, Views и Stored Procedures. Развернув эти узлы, можно увидеть списки конкретных компонентов баз данных вместе с информацией о них. Например, узел Tables содержит узел для таблицы `Customer`, которая в свою очередь содержит узел для каждого из столбцов, например `CustomerID`, `FirstName` и `LastName`. Щелкнув на этих узлах, можно быстро просмотреть свойства компонентов в инструментальном окне Properties. Это представление базы данных принято по умолчанию; его можно переключить на представление Object Type или Schema, выбрав команду Change View, а затем указав новое представление в контекстном меню, связанном с узлом базы данных и открывающимся после щелчка правой кнопкой мыши. Каждое из этих представлений просто группирует информацию о базе данных в иерархии разного вида.

Для того чтобы добавить новое соединение с базой данных в окно Server Explorer, необходимо щелкнуть на кнопке Connect to Database в верхней части окна Server Explorer (третья пиктограмма слева) или щелкнуть правой кнопкой мыши на корневом узле Data Connections и выбрать команду Add Connection в контекстном меню. Если соединение добавляется в первый раз, система Visual Studio спросит, с каким типом источника данных следует установить соединение. Система Visual Studio 2010 поставляется вместе с большим количеством средств для соединения Data Source, включая Access, SQL Server и Oracle, а также с универсальным драйвером ODBC. Она также содержит средство для соединения с источником данных для баз данных Microsoft SQL Server Database File и Microsoft SQL Server Compact.

Команда Database File впервые появилась в системе SQL Server 2005, а затем была позаимствована оттуда родственными системами Microsoft Access и MSDE. С помощью команды SQL Server Database File можно создать простой файл для отдельной базы данных. Это означает, что пользователь не обязан соединять его с экземпляром системы SQL Server. Благодаря этому такой файл обладает высокой мобильностью — пользователь может просто поставлять файл с расширением `.mdf`, содержащим базу данных, вместе со своим приложением. В качестве альтернативы можно использовать базу данных SQL Server Compact (SSC), которая значительно снижает требования системы к приложению. Вместо требования предварительной инсталляции экземпляра системы SQL Server одновременно с приложением пользователя разворачивается система SSC.

После того как пользователь выберет тип источника данных, откроется диалоговое окно Add Connection. На рис. 26.2 показано окно Add Connection, содержащее соединение SQL Server Database File и настройки для соответствующего типа источника данных. Если соединение с данными в системе Visual Studio ранее уже было установлено, то это окно появится сразу.



Рис. 26.2

Кнопка Change открывает окно Data Sources, позволяя пользователю добавлять соединения с базами данных разных типов в свой сеанс работы с системой Visual Studio. Обратите внимание на то, как легко создать соединение SQL Server Database File. Достаточно набрать имя требуемого файла или найти его в окне просмотра и указать имя новой базы данных. Если необходимо установить соединение с существующей базой данных, следует воспользоваться кнопкой Browse, позволяющей просматривать файловую систему.

В общем, пользователь должен лишь указать, использует ли его конфигурация SQL Server средства аутентификации Windows или SQL Server Authentication. По умолчанию система Visual Studio 2010 содержит инсталляционный пакет системы SQL Server 2005 Express, использующей в качестве основной модели аутентификации режим Windows Authentication.



Кнопка Test Connection выполняет проверку и выводит сообщение об ошибке, если пользователь попытается соединиться с новой базой данных. Это объясняется тем, что она не существует, пока пользователь не щелкнет на кнопке OK, и, следовательно, соединяться пока не с чем!

Когда пользователь щелкает на кнопке OK, система Visual Studio пытается установить соединение с базой данных. В случае успеха она добавляет его в узел Data Connections вместе с дочерними узлами для основных типов данных в этой базе (см. выше). Если же базы данных не существует, система Visual Studio спрашивает у пользователя, следует ли создать ее. Пользователь может также создать новую базу данных, выбрав команду Create New SQL Server Database в контекстном меню узла Data Connections в окне Server Explorer.

Table Editing

Проще всего отредактировать таблицу в базе данных, щелкнув на ней в окне Server Explorer. После этого в основной рабочей области открывается окно редактирования, состоящее из двух компонентов. В верхней части вводится имя каждого поля, тип данных и основная информация, например длина текстовых полей, или является ли поле обнуляемым.

Щелкнув правой кнопкой мыши на поле, пользователь получает доступ к набору операций, которые можно к нему применить (рис. 26.3). Это контекстное меню содержит те же самые команды, что и меню Table Designer, которое отображается на экране, когда пользователь редактирует таблицу, но обычно проще использовать контекстное меню, поскольку в этом случае ясно видно, какое поле модифицируется.

В нижней части области редактирования таблицы находится окно Column Properties с текущим выбранным столбцом. В отличие от сетки, в которой просто перечислены атрибуты Column Name, Data Type и указано, является ли окно обнуляемым, об-

ласть свойств столбца позволяет задавать все доступные свойства для источника данных конкретного типа.

На рис. 26.4 показан пример окна Column Properties для поля CustomerID, которое было определено вместе с идентификатором, который автоматически увеличивается на единицу при добавлении в таблицу новой записи.

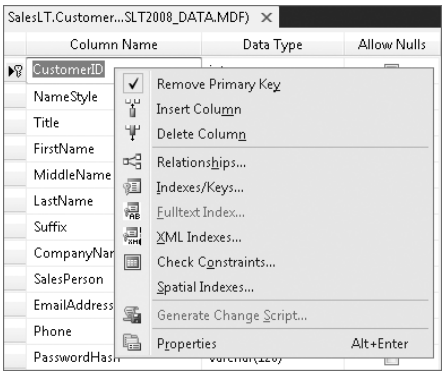


Рис. 26.3

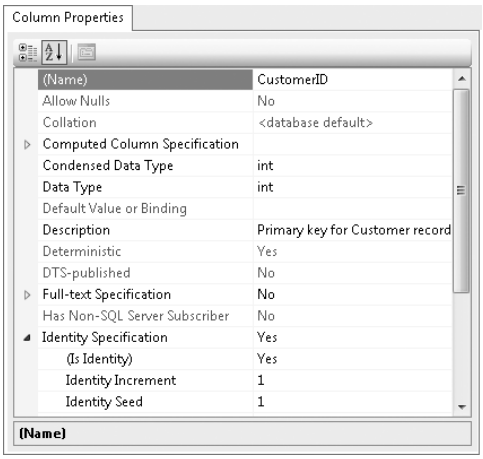


Рис. 26.4

Редактирование отношений

Большинство баз данных, используемых в приложениях, созданных по технологии .NET, по своей природе являются реляционными. Это значит, что соединение с таблицей устанавливается одновременно с определением отношений. Для того чтобы создать отношение, откройте одну из таблиц, которая станет частью отношений, и щелкните на кнопке Relationships инструментальной панели или выберите команду Table Designer⇨Relationships. В открывшемся диалоговом окне Foreign Key Relationships (рис. 26.5) доступны все существующие отношения, связанные с выбранной таблицей.

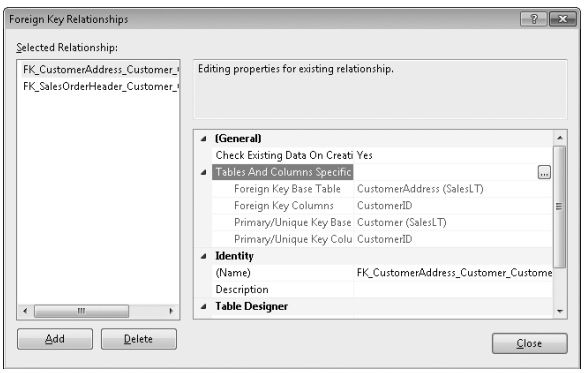


Рис. 26.5

Для того чтобы создать новое отношение, необходимо щелкнуть на кнопке Add. Если требуется отредактировать существующее отношение, его можно выбрать в списке. Найдите пункт Tables and Columns Specification в сетке свойств и щелкните на связанном с ним эллипсисе, чтобы задать таблицы и столбцы, которые необходимо связать друг с другом. В диалоговом окне Tables and Columns, показанном на рис. 26.6, сначала выберите, какая из таблиц содержит первичный ключ, с которым должна соединяться выбранная таблица. Обратите внимание на то, что для новых отношений поле ввода Foreign key table уже заполнено именем текущей таблицы и не может изменяться.

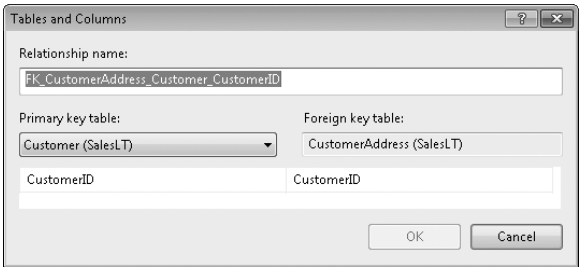


Рис. 26.6

Установив параметр Primary key table, можно соединять поля в каждой таблице, которые должны быть связаны друг с другом. В отношение можно добавить сразу несколько полей, щелкнув на пустой строке, добавленной при вставке предыдущего поля. Если установка отношений завершена, щелкните на кнопке OK, чтобы сохранить настройки и вернуться к диалоговому окну Foreign Key Relationships.

Представления

Представления (views) — это заранее определенные запросы, которые появляются в приложении как таблицы и могут состоять из нескольких таблиц. Для того чтобы добавить новое представление, необходимо выбрать команду Data⇨Add New⇨View или щелкнуть правой кнопкой мыши на узле Views в окне Server Explorer и выбрать в контекстном меню команду Add New View.

Сначала необходимо выбрать, какие таблицы, другие представления и синонимы будут включены в текущее представление. Когда пользователь выберет добавляемый компонент, на экране появится окно редактора View (рис. 26.7). Этот редактор знаком каждому, кто работал с визуальным проектировщиком баз данных, например с программой Access. Таблицы и другие компоненты показаны в верхней части окна, где можно выбрать поля, которые необходимо включить в приложение. В верхней части окна показаны также соединения между функциями и таблицами. В окне View на рис. 26.7 показаны соединения между тремя таблицами, которые выражаются в виде связей всех строк таблицы `Customer` с таблицами `CustomerAddress` и `Address`. На рис. 26.7 показано также, что, щелкнув правой кнопкой мыши на соединении между таблицами, можно изменить тип соединения. Если необходимо добавить дополнительные таблицы, нужно щелкнуть правой кнопкой мыши и выбрать команду Add Table.

В средней части окна показана текущая таблица. В ней также можно добавлять столбцы для сортировки и фильтрации. В области, непосредственно расположенной под таблицей, показаны инструкции SQL, с помощью которых было получено данное представление. Содержимое всех этих трех панелей можно изменять, при этом остальные панели будут модифицироваться автоматически.

Нижнюю часть области редактора представлений можно использовать для выполнения инструкций SQL и просмотра результатов. Для того чтобы выполнить эти инструкции, необходимо выбрать команду Execute SQL из контекстного меню любой из панелей или щелкнуть на одноименной кнопке инструментальной панели View Designer.

Хранимые процедуры и функции

Для того чтобы создавать и модифицировать хранимые процедуры и функции, система Visual Studio 2010 использует текстовый редактор, показанный на рис. 26.8. Поскольку система Visual Studio поддерживает технологию автоматического дополнения IntelliSense при создании определений процедур и функций, при обнаружении ошибки она не позволит сохранить код.

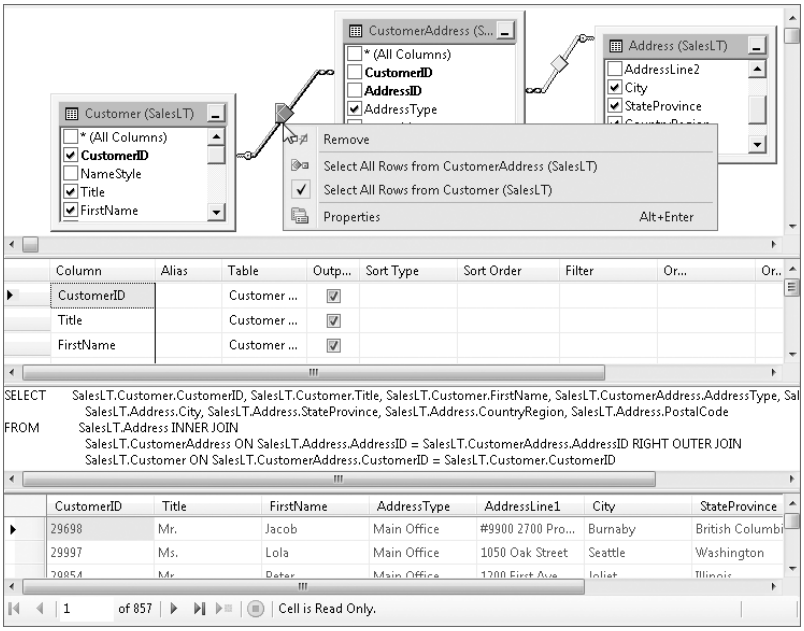


Рис. 26.7

Для упрощения процесса создания и отладки хранимых процедур и функций существуют сокращенные варианты выбора команд Insert SQL, Run Selection и Execute из контекстного меню текстового редактора. Вставляемые инструкции демонстрируются в модальном диалоговом окне Query Builder, показанном на рис. 26.7. Команда Run Selection пытается выполнить все выбранные инструкции SQL и выводит результаты в окне Output. В заключение сокращенная команда Execute запускает на выполнение всю хранимую процедуру или функцию. Если эти процедуры или функции принимают входные параметры, то открывается диалоговое окно, подобное показанному на рис. 26.9, в котором можно задать соответствующие тестовые значения. Результаты также отображаются в окне Output.

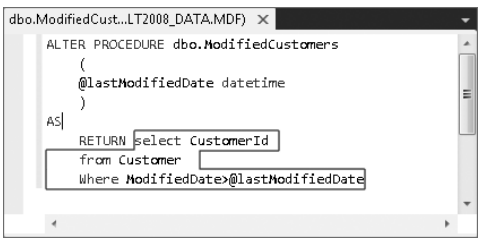


Рис. 26.8



Рис. 26.9

Диаграммы баз данных

Пользователь может также создать визуальное представление таблиц баз данных с помощью *диаграмм баз данных* (database diagrams). Для того чтобы создать диаграмму, необходимо выполнить команду Data⇨Add New⇨Diagram из контекстного меню или щелкнуть правой кнопкой мыши на узле Database Diagrams в окне Server Explorer и выбрать в контекстном меню команду Add New Diagram.

При создании первой диаграммы в базе данных система Visual Studio может предложить пользователю автоматически добавлять необходимые системные таблицы и данные в базу данных. Если запретить это действие, то пользователь вообще не сможет создавать диаграммы; такие образом, по существу, это просто уведомление, а не предложение.

В процессе создания диаграммы пользователь может выбирать таблицы, которые хочет видеть на диаграмме. Правда, добавлять их можно будет только после того, как в интегрированную среду разработки будет включено меню Database Diagram. С помощью этого меню можно изменять внешний вид диаграммы в редакторе, увеличивать и уменьшать масштаб изображения, разбивать диаграммы на страницы, а также включать и отключать имена отношений.

Поскольку диаграммы баз данных могут быть довольно большими, интегрированная среда разработки предусматривает простой способ их просмотра. В левом нижнем углу редактора Database Diagram находится пиктограмма с четырехнаправленными стрелками. После щелчка на этой пиктограмме появится свернутое представление диаграммы (рис. 26.10).

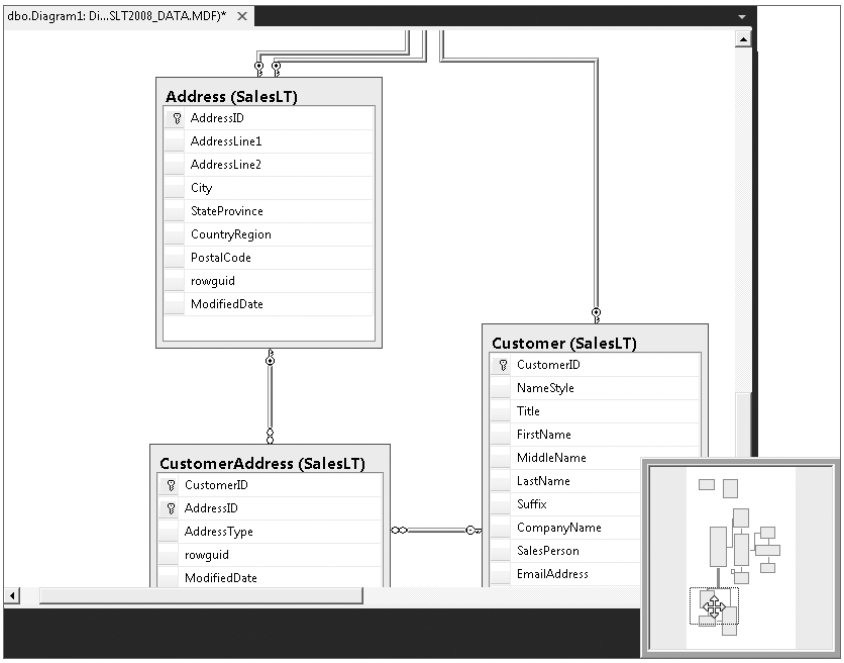


Рис. 26.10

Для того чтобы увидеть требуемый компонент и работать с ним в области просмотра интегрированной среды разработки, нужно нажать кнопку мыши и перетаскивать ее курсор по диаграмме, пока не обнаружится искомый компонент.

Окно Data Sources

Окно Data Sources, которое обычно появляется в той же инструментальной области экрана, что и окно Solution Explorer, содержит все активные источники данных,

известные проекту, например наборы данных (в противоположность компонентам Data Connections в окне Server Explorer, которые известны системе Visual Studio). Для того чтобы открыть инструментальное окно Data Sources, необходимо выбрать команду Data⇨Show Data Sources.

Окно Data Sources имеет два основных представления, зависящих от активного документа в рабочей области интегрированной среды разработки. Если пользователь редактирует код, то в окне Data Sources отображаются таблицы и поля с пиктограммами, представляющими их тип. Это помогает пользователю писать код, поскольку он может легко ссылаться на тип без просмотра определения таблицы. Это представление показано рис. 26.11, *справа*

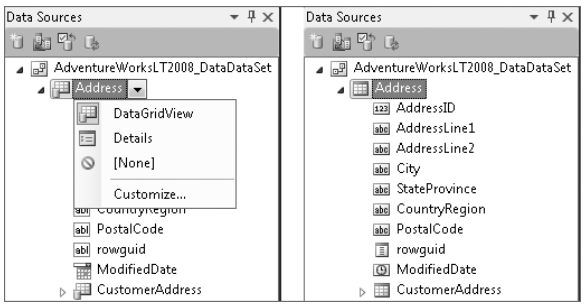


Рис. 26.11

Однако, если пользователь редактирует форму в представлении Design, представление Data Sources изменяется и отображает таблицы и поля вместе с пиктограммами, представляющими типы их текущих элементов управления, заданных по умолчанию (изначально они задаются на странице Data UI Customization окна Options). На рис. 26.11, *слева*, продемонстрированы текстовые поля, использующие элементы управления TextBox, в то время как поле ModifiedDate использует элемент управления DateTimePicker. Пиктограммы для таблиц свидетельствуют о том, что все таблицы были вставлены по умолчанию как компоненты DataGridView (см. раскрывающийся список).

В следующей главе мы покажем, как добавлять и модифицировать источники данных, а также как использовать окно Data Sources для связывания данных с элементом управления на форме. Для того чтобы создать пользовательский интерфейс, классы данных или полей можно просто перетаскивать из окна Data Sources на форму.

Редактирование данных

Система Visual Studio 2010 позволяет также просматривать и редактировать данные, содержащиеся в пользовательских базах данных. Для того чтобы отредактировать информацию, выделите таблицу, которую пользователь хочет просматривать в окне Server Explorer, и выберите команду Data⇨Show Table Data. Данные представляются в виде таблицы, как показано на рис. 26.12. Это позволяет редактировать ее содержимое или тестировать данные, которые необходимо в нее включить. Используя кнопки в нижней части таблицы, пользователь может перемещаться по записям и даже создавать новые строки. В ходе редактирования информации редактор таблиц выводит на экран рядом с измененными полями соответствующие индикаторы.

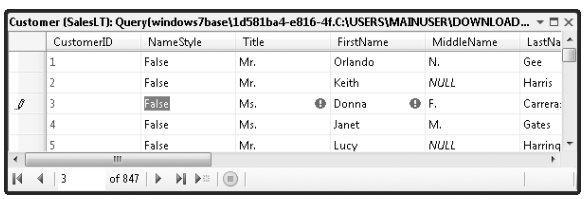


Рис. 26.12

Кроме того, щелкнув правой кнопкой мыши в любом месте таблицы и выбрав соответствующую команду из подменю Pane, можно вывести на экран диаграмму, критерии и панели SQL, связанные с редактируемой таблицей. Эта возможность может оказаться полезной для уточнения инструкций SQL, используемых для извлечения данных, например для фильтрации таблицы по заданному шаблону или просто для получения первых пятидесяти строк.

Предварительный просмотр данных

Пользователь может предварительно просматривать данные из разных источников, чтобы убедиться, что соответствующий запрос вернет требуемую информацию. Для этого в проектировщике схем баз данных необходимо щелкнуть правой кнопкой мыши на тестируемом запросе и выбрать команду Preview Data из контекстного меню. В качестве альтернативы можно выбрать в инструментальном окне Data Sources команду Preview Data из контекстного меню, связанного с любым источником данных.

Диалоговое окно Preview Data содержит список объектов, по умолчанию связанных с запросом, который хочет тестировать пользователь. Для того чтобы просмотреть пример данных, показанных на рис. 26.13, необходимо щелкнуть на кнопке Preview. Небольшая строка состояния содержит информацию об общем количестве строк, возвращенных по запросу, а также количество включенных столбцов.

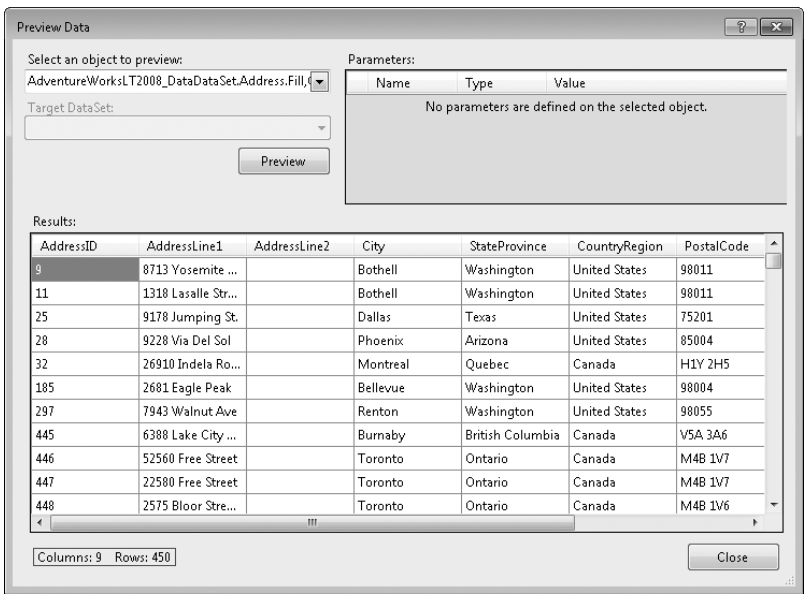


Рис. 26.13

Если пользователь хочет выполнить другой запрос, он может выбрать объект из раскрывающегося списка с помощью команды Select. Этот список содержит другие запросы в том же источнике данных, в других источниках данных и вообще в решении пользователя. Если просматриваемый запрос имеет параметры, их значения можно задать в списке Parameters в правой верхней панели этого диалогового окна. Щелкнув на кнопке Preview, пользователь посылает запрос соответствующему источнику данных и выводит результаты на панели Results в диалоговом окне Preview Data.

Резюме

Имея в своем распоряжении разнообразные инструменты и окна системы Visual Studio 2010, пользователь может легко создавать и эксплуатировать базы данных, не покидая интегрированную среду разработки. Он может манипулировать данными, а также визуально задавать схемы баз данных, используя инструментальное окно Properties в сочетании с инструментом Schema Designer.

Если пользователь хочет работать с данными, система Visual Studio предоставляет ему набор компонентов, которые можно перетаскивать и отпускать, а также связывать с источниками данных. Этот процесс может быть таким же простым, как установки и сброс флажков и переключателей, и настолько же функциональным, как компонент DataGridView, позволяющий полностью просматривать таблицы. В следующей главе мы покажем, что возможность перетаскивать на форму целые таблицы или отдельные поля из окна Data Sources и предоставлять системе Visual Studio автоматически создавать соответствующие элементы управления является главным преимуществом при быстрой разработке приложений.

27

Компоненты DataSet и DataBinding

В ЭТОЙ ГЛАВЕ...

- Создание компонентов DataSet
- Соединение визуальных элементов управления с компонентами DataSet с помощью компонента DataBinding
- Как связать друг с другом элементы управления BindingSource и BindingNavigator
- Создание цепочек элементов управления BindingSource и использование компонента DataGridView
- Использование источников данных Service и Object

Большая часть приложений использует определенную форму хранения данных. Данные могут храниться в виде сериализованных объектов или XML-данных, но для долгосрочного хранения, обеспечивающего одновременный доступ для многих пользователей, чаще всего используются базы данных. Технология .NET Framework обеспечивает полноценную поддержку работы с базами данных и другими источниками данных. В этой главе показано, как использовать компоненты DataSet для создания приложений, работающих с базами данных.

Во второй части главы мы расскажем, как использовать компоненты DataBinding для соединения визуальных элементов управления с данными, которые они отображают на экране. Будет показано, как они взаимодействуют и как с помощью визуальных проектировщиков можно управлять внешним представлением данных.

Примеры, использованные в главе, основаны на эталонной базе данных `AdventureWorksLT`, которую можно загрузить с веб-страницы `http://professionalvisualstudio.com/link/1029A`.

Обзор компонентов DataSet

Компонент DataSet технологии .NET Framework — это сложный объект, приблизительно соответствующий представлению базы данных в памяти компьютера. Он содержит компоненты DataTable, соответствующие таблицам базы данных. Они, в свою очередь, содержат ряд компонентов DataColumn, которые определяют состав компонентов DataRow. Компонент DataRow соответствует строке в таблице базы данных. Можно также установить отношение между компонентами DataTable в компоненте DataSet, точно так же, как устанавливаются отношения между таблицами в базе данных.

Одной из нерешенных проблем, связанных с парадигмой объектно-ориентированного программирования, является то, что она не совсем точно соответствует реляционной модели базы данных. Объект DataSet прошел длинный путь эволюции, чтобы восполнить этот недостаток, поскольку его можно использовать для представления и работы с реляционными данными в объектно-ориентированном виде. Тем не менее основным недостатком объекта DataSet является его слабая типизированность. Несмотря на то что тип каждого столбца можно запросить еще до обращения к элементам базы данных, это связано с дополнительными затратами ресурсов и может сделать код почти непонятным. Сильно типизированные компоненты DataSet сочетают в себе преимущества простого компонента DataSet и строгий контроль типов (иначе говоря, создание строго типизированных свойств для всех полей базы данных, чтобы обеспечить корректный доступ к данным на этапе проектирования). Для этого используется специализированный инструмент MSDataSetGenerator, который преобразовывает схему XML в строго типизированный компонент DataSet, заменяя, по существу, многие проверки типов на этапе выполнения проверкой с помощью кода, генерируемого на этапе проектирования. В следующем сниппете кода продемонстрирована разница между простым компонентом DataSet (в первой половине сниппета) и строго типизированным компонентом DataSet (во второй половине).



VB

```
'Простой компонент DataSet
Dim nontypedAwds As DataSet = RetrieveData()
Dim nontypedcustomers As DataTable = nontypedAwds.Tables("Customer")
Dim nontypedfirstcustomer As DataRow = nontypedcustomers.Rows(0)
MessageBox.Show(nontypedfirstcustomer.Item("FirstName"))

'Строго типизированный компонент DataSet
Dim awds As AdventureWorksLTDataSet = RetrieveData()
Dim customers As AdventureWorksLTDataSet.CustomerDataTable =
awds.Customer
Dim firstcustomer As AdventureWorksLTDataSet.CustomerRow =
customers.Rows(0)
MessageBox.Show(firstcustomer.FirstName)
```

Сниппет кода CustomersForm.vb



C#

```
// Простой компонент DataSet
DataSet nontypedAwds = RetrieveData();
DataTable nontypedcustomers = nontypedAwds.Tables["Customer"];
DataRow nontypedfirstcustomer = nontypedcustomers.Rows[0];
MessageBox.Show(nontypedfirstcustomer["FirstName"].ToString());

// Строго типизированный компонент DataSet
AdventureWorksLTDataSet awds = RetrieveData();
```

```
AdventureWorksLTDataSet.CustomerDataTable customers = awds.Customer;
AdventureWorksLTDataSet.CustomerRow firstcustomer =
                    customers.Rows[0] as
                    AdventureWorksLTDataSet.CustomerRow;
MessageBox.Show(firstcustomer.FirstName);
```

Сниппет кода CustomersForm.cs

Используя простой компонент DataSet, можно выполнять поиск таблицы и имени столбца, используя строковые литералы. Строковые литералы могут быть источником сложностей и должны использоваться только в сгенерированном коде, а по возможности не использоваться вообще.

Добавление источника данных

Строго типизированный объект DataSet можно создать вручную, создав схему XSD с помощью редактора схем XML. Для того чтобы создать объект DataSet, необходимо задать в качестве *специализированного инструмента* (custom tool) для XSD-файла компонент MSDataSetGenerator. В результате возникнет исходный код проектировщика, необходимый для строго типизированного доступа к объекту DataSet.

Создать схему XSD вручную трудно, и без крайней необходимости делать это не следует; к счастью, в большинстве случаев источником данных являются базы данных, для которых в системе Visual Studio 2010 есть мастер, с помощью которого можно генерировать необходимые схемы, основываясь на структуре пользовательской базы данных. Ниже мы покажем, как создать источник данных и связать его с пользовательским интерфейсом. Для начала создадим проект `CustomerObjects`, используя проектный шаблон Windows Forms Application.



Несмотря на то что такой функциональной возможности для проектов типа ASP.NET не существует, в качестве временного решения можно реализовать доступ ко всем данным с помощью библиотеки классов.

Для того чтобы создать строго типизированный объект DataSet на основе существующей базы данных, выберите команду Add New Data Source из меню Data и выполните следующие действия.

1. На первом этапе работы мастера Data Source Configuration Wizard выбирается тип источника данных, предназначенного для работы, – Database, Service, Object или SharePoint. В данном случае мы хотим работать с данными из базы данных, поэтому выберите пиктограмму Database и щелкните на кнопке Next.
2. С появлением технологии ADO.NET Entity Framework возникли две разные модели данных, которые можно выбирать для представления отображения между данными из базы данных и сущностей платформы .NET: объект DataSet и модель Entity Data Model. Технология Entity Framework описана в главе 29. Для продолжения дважды щелкните на пиктограмме DataSet.
3. Следующее окно позволяет выбрать соединение с базой данных. Чтобы создать новое соединение, щелкните на кнопке New Connection, которая открывает диалоговое окно Add Connection. Атрибуты, содержащиеся в этом окне, зависят о типа базы данных, с которой устанавливается соединение. По умолчанию выбирается провайдер SQL Server. Для этого необходимо задать имя сервера, механизм аутентификации (Windows или SQL Server), а также имя ба-

зы данных. Существует также команда Test Connection, с помощью которой можно проверить корректность заданных свойств.

4. Как только пользователь задаст соединение, оно сохраняется как настройка приложения в файле конфигурации приложения.

   Когда впоследствии приложение будет развернуто, строку соединения можно будет изменить, чтобы указать рабочую базу данных. Этот процесс может оказаться более долгим, чем ожидалось, потому что при этом устанавливаются различные уровни безопасности. Поскольку строка соединения хранится в файле конфигурации как обычная строка без какой-либо схемы, довольно легко совершить ошибку при попытке изменить ее. В главе 36 мы расскажем больше о строках соединения, а также покажем, как настраивать их на разные источники данных.

5. Установив соединение, мы переходим на следующий этап и указываем данные, которые будут извлекаться из базы данных. На этом этапе нам предоставляется список таблиц, представлений, хранимых процедур и функций, которые можно выбрать и включить в объект DataSet. На рис. 27.1 показан заключительный этап работы мастера Data Source Configuration Wizard, на котором выбирается столбец из таблицы `Customer` в базе данных `AdventureWorksLT`. Установив флажок `Enable Local Database Caching`, можно перейти в режим offline и работать с данными без соединения. Для этого используется механизм Synchronization Services платформы ADO.NET, который синхронизирует данные, хранящиеся в приложении и в базе данных. Этот механизм описан в главе 34.

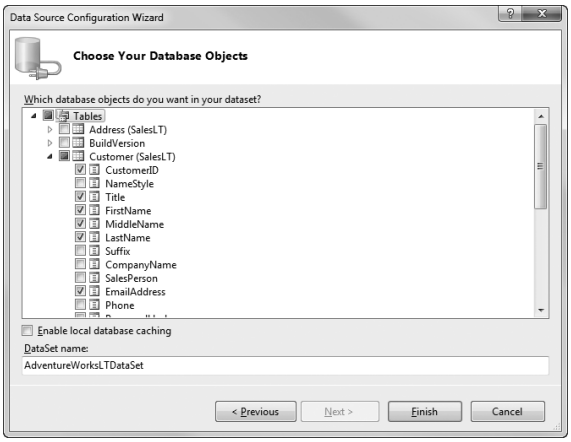


Рис. 27.1



В системе Windows есть одна малоизвестная утилита, предназначенная для создания строк соединения, даже если система Visual Studio не инсталлирована. Она реализована в виде диалогового окна Data Link Properties, и с ее помощью можно редактировать файлы Universal Data Link, имеющие расширение `.udl`. Если необходимо создать или протестировать строку соединения, можно создать новый текстовый документ, переименовать его в `something.udl` и дважды щелкнуть на нем мышью. После этого откроется диалоговое окно Data Link Properties, позволяющее создавать и тестировать строки соединения для многих провайдеров. Как только пользователь выберет соответствующее соединение, эта информация будет записана в файл UDL в виде строки соединения, которую можно извлечь, открыв тот же самый файл программой Notepad. Это особенно полезно, если необходимо протестировать уровни безопасности и решить некоторые проблемы, связанные с соединением.

6. Щелкните на кнопке Finish, чтобы добавить новый объект DataSet в окно Data Sources, показанное на рис. 27.2, где можно найти всю информацию, которая должна быть извлечена из объекта DataSet. Каждый столбец изображается в виде пиктограмм, которые изменяются в зависимости от типа данных в столбце. На левой панели (которая появляется, когда используется рабочая область проектирования) пиктограммы изображают визуальные элементы управления, которые по умолчанию используются для представления столбцов; на правой панели (которая появляется на экране, когда фокус находится в окне редактирования кода) пиктограммы означают типы данных.

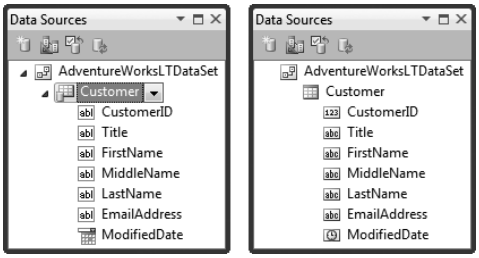
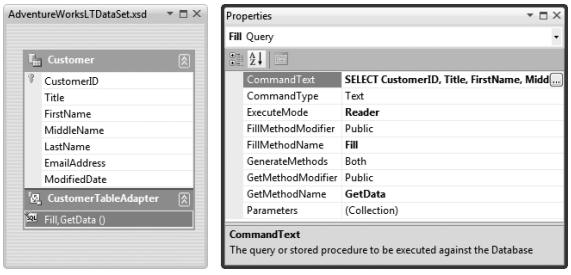


Рис. 27.2



Проектировщик объектов DataSet

Мастер Data Source Configuration Wizard использует схему базы данных, для того чтобы угадать соответствующий тип платформы .NET для строк объекта DataTable. Когда мастер ошибается, может оказаться полезным редактировать объект DataSet без его помощи. Для этого нужно щелкнуть правой кнопкой мыши на объекте DataSet в окне Data Sources и выбрать команду Edit DataSet with Designer из контекстного меню. В качестве альтернативы можно открыть окно Data Sources, дважды щелкнув на файле XSD в окне Solution Explorer. В результате в главном окне откроется редактор объектов DataSet, как показано на рис. 27.3.

Рис. 27.3

Здесь начинает проявляться мощь строго типизированных объектов DataSet. В объект DataSet добавлена не только строго типизированная таблица (`Customer`), но и адаптер таблиц `CustomerTableAdapter`. Этот компонент TableAdapter используется для выбора и модификации базы данных из объекта DataTable, к которому он относится. Если в объект DataSet добавлено несколько таблиц, каждой из них будет соответствовать отдельный компонент TableAdapter. Несмотря на то что отдельный компонент TableAdapter можно легко настроить так, чтобы он возвращал информацию, извлеченную из нескольких таблиц, содержащихся в базе данных, с его помощью трудно модифицировать, вставлять и удалять записи.

Как показано на рис. 27.3, компонент CustomerTableAdapter был создан с методами Fill и GetData, которые предназначены для извлечения данных из базы данных. Следующий код демонстрирует, как использовать метод Fill для заполнения существующего строго типизированного объекта DataTable, возможно, находящегося в объекте DataSet. В качестве альтернативы можно использовать метод GetData, создающий новый экземпляр строго типизированного компонента DataTable.

VB

```
Dim ta As New AdventureWorksLTDataSetTableAdapters.CustomerTableAdapter

'Пункт 1- Создать новый объект класса CustomerDataTable и вызвать метод Fill
Dim customers1 As New AdventureWorksLTDataSet.CustomerDataTable
ta.Fill(customers1)

'Пункт 2- Вызвать метод GetData, который создаст объект класса
CustomerDataTable
Dim customers2 As AdventureWorksLTDataSet.CustomerDataTable = ta.GetData
```

На рис. 27.3 показано, что методы Fill и GetData используются парой, потому что они применяются к одному и тому же запросу. Для конфигурирования запроса можно использовать окно Properties. Запрос может возвращать данные тремя способами: с помощью текстовой команды (как показано в примере), хранимой процедуры или компонента TableDirect (откуда извлекается содержимое имени таблицы, заданной в строке CommandText). Она задается в поле CommandType. Несмотря на то что строку CommandText можно редактировать непосредственно в окне Properties, в нем трудно просмотреть весь запрос и легко сделать ошибки. Щелкнув на кнопке с эллипсисом (см. в правом верхнем углу на рис. 27.3), можно открыть окно Query Builder, показанное на рис. 27.4.

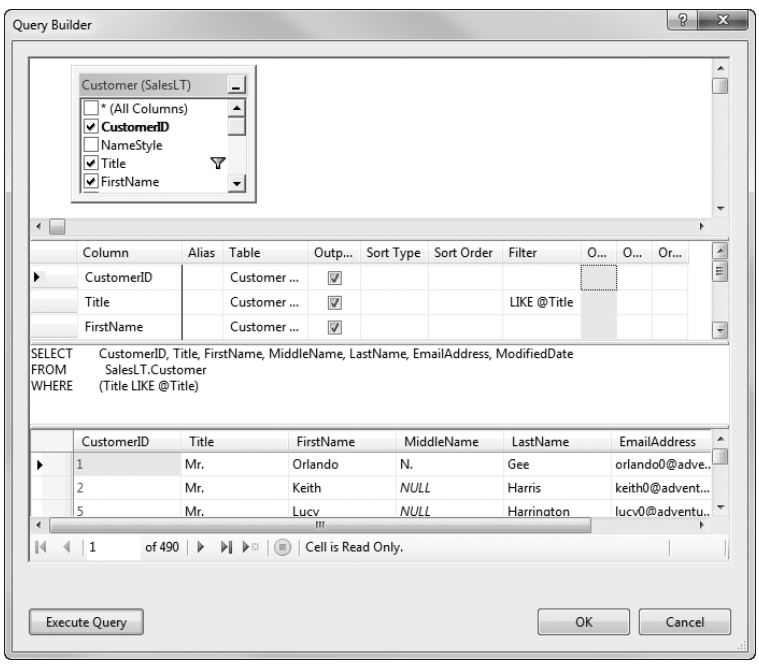


Рис. 27.4

Диалоговое окно Query Builder разделено на четыре панели. На верхней панели размещена диаграмма таблиц, упомянутых в запросе, и выбранные столбцы. Вторая панель демонстрирует список столбцов, связанных с запросом. Эти столбцы представляют собой либо столбцы результатов, таких как `FirstName` и `LastName`, либо условия, как, например, поле `Title`, либо и то и другое. Третья панель, разумеется, содержит команду SQL, которая должна быть выполнена. Последняя панель содержит пример данных, которые можно извлечь, щелкнув на кнопке Execute Query. Если инструкция SQL имеет параметры (в данном случае `@Title`), то откроется диалоговое окно, позволяющее ввести значения, которые будут использоваться при выполнении инструкции.

Для того чтобы изменить запрос, можно модифицировать первые три панели. При перемещении с одной панели на другую изменения, внесенные на одной панели, будут отображаться на других панелях. Каждую из этих панелей можно скрыть, сняв соответствующий флажок с помощью команды Panes из контекстного меню. Условия можно добавить с помощью столбца Filter. Он может содержать параметры (такие как `@Title`), которые должны начинаться с символа `@`.

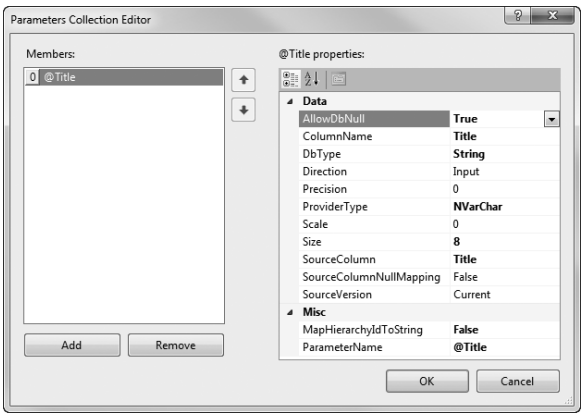


Рис. 27.5

Вернувшись к проектировщику DataSet и окну Properties, которое связано с методом `Fill`, щелкните на кнопке с многоточием, чтобы проверить список параметров. В результате откроется окно Parameters Collection Editor (рис. 27.5).

Иногда компонент Query Builder не может правильно определить тип параметра. В этом случае тип необходимо модифицировать с помощью диалогового окна. Кроме того, в окне Properties, соответствующем запросу, можно указать, следует ли создать методы `Fill` и/или `GetData`, используя свойство `GenerateMethods`, имеющее значение `Fill`, `Get` или `Both`. Кроме того, можно задать имена и уровень доступа к генерируемым методам.

Привязка данных

Наиболее распространенными являются приложения, которые извлекают данные из базы, отображают их на экране, позволяют вносить в них изменения, а затем сохраняют эти изменения в исходной базе данных. Посредники, которые соединяют данные, хранящиеся в памяти компьютера, с визуальными элементами управления, называются DataBinding. Компоненты DataBinding часто становятся проклятием разработчика, потому что их очень трудно заставить правильно работать. Большинство разработчиков рано или поздно пишут свои собственные оболочки, чтобы обеспечить правильное связывание данных с элементами управления, расположенными на экране. Система Visual Studio 2010 значительно улучшила работу компонентов DataBinding. Примеры, использованные в следующих разделах, основаны на эталон-

ной базе данных AdventureWorksLT. Для простоты мы будем работать с одним Windows-приложением, но концепции, рассмотренные здесь, можно расширить и на случай нескольких ярусов.

Мы создадим приложение, которое будет управлять клиентской базой компании AdventureWorks. Для начала нам необходимо убедиться, что база данных AdventureWorksLTDataSet содержит таблицы Customer и Address. (Базу данных AdventureWorksDataSet можно позаимствовать из предыдущего примера, щелкнув на пиктограмме Configure DataSet with Wizard в окне Data Source и отредактировав таблицы, включенные в объект DataSet.) Открыв проектировщик форм (подойдет любая пустая форма из проекта) и окно Data Sources, установите для таблицы Customer режим Details, используя раскрывающийся список. Прежде чем создавать элементы управления для редактирования, следует наладить список столбцов для таблицы Customer. Нас не интересуют поля CustomerID, NameStyle, PasswordHash, PasswordSalt или поля глобальных идентификаторов, поэтому установите для них режим None (снова воспользуйтесь раскрывающимся списком этих узлов в окне Data Sources). Поле ModifiedDate должно автоматически учитывать изменения, поэтому оно должно появляться в виде метки, чтобы предотвратить редактирование.

Теперь вы готовы перетащить узел Customer на рабочую поверхность проектировщика формы. При этом на форму будут автоматически добавлены элементы управления для каждого из заданных столбцов. Кроме того, на форму будут добавлены элементы BindingSource, BindingNavigator, AdventureWorksDataSet, CustomerTableAdapter, TableAdapterManager и ToolStrip (рис. 27.6).

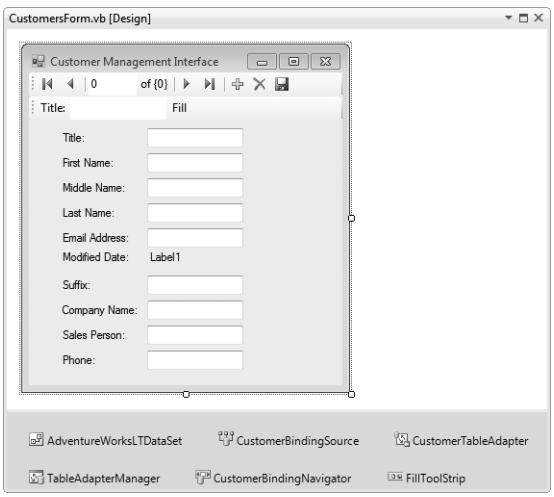


Рис. 27.6

В этот момент можно собрать и выполнить приложение и пройтись по записям с помощью навигационных элементов управления. Кроме того, все компоненты можно пока рассматривать изолированно друг от друга, не интересуясь, как они взаимодействуют между собой. Начнем с компонентов AdventureWorksDataSet и CustomerTableAdapter, поскольку они создают основу для поиска информации и хранения изменений в базе данных. Компонент AdventureWorksDataSet, добавленный на форму, по существу, представляет собой класс AdventureWorksDataSet, созданный мастером Data Source Configuration Wizard. Этот экземпляр можно использовать для хранения информации для всех таблиц, принадлежащих форме. Для того чтобы заполнить объект DataSet, следует вызвать метод Fill. Если вы откроете исходный файл, лежащий в основе формы, то увидите, что метод Fill вызывается из обработчика события Click кнопки Fill, находящейся на инструментальной панели.



VB

```
Private Sub FillToolStripButton_Click(ByVal sender As Object,
                                      ByVal e As EventArgs) _
```

```
                                                    Handles FillToolStripButton.Click
    Try
         Me.CustomerTableAdapter.Fill(Me.AdventureWorksLTDataSet.Customer,
                                      TitleToolStripTextBox.Text)
    Catch ex As System.Exception
         System.Windows.Forms.MessageBox.Show(ex.Message)
    End Try
End Sub
```

Сниппет кода CustomersForm.vb



C#

```
private void fillToolStripButton_Click(object sender, EventArgs e){
   try{
        this.customerTableAdapter.Fill(
            this.adventureWorksLTDataSet.Customer,
                 titleToolStripTextBox.Text);
   }
   catch (System.Exception ex){
        System.Windows.Forms.MessageBox.Show(ex.Message);
   }
}
```

Сниппет кода CustomersForm.cs

Для того чтобы расширить эту форму, добавим на нее компонент `TableAdapter` для каждой таблицы, принадлежащей компоненту `AdventureWorksDataSet`.

Компонент BindingSource

Следующий компонент, который нас интересует, называется `CustomerBindingSource`. Он автоматически добавляется в область невидимых компонентов проектировщика формы. Этот элемент управления используется для обеспечения связи между всеми элементами управления, находящимися в рабочей области проектировщика, и соответствующими элементами для работы с данными. По существу, этот элемент управления представляет собой оболочку для компонента `CurrencyManager`. Однако компонент BindingSource значительно сокращает количество обработчиков событий и объем кода, который необходимо написать. В отличие от компонентов `AdventureWorksDataSet` и `CustomerTableAdapter`, представляющих собой экземпляры строго типизированных классов с теми же самыми именами, компонент `CustomerBindingSource` является экземпляром обычного класса `BindingSource`, поставляемого вместе с платформой .NET Framework.

Посмотрим на свойства компонента `CustomerBindingSource`, чтобы понять, что он собой представляет. На рис. 27.7 показано окно Properties для компонента `CustomerBindingSource`. Особый интерес представляют свойства DataSource и DataMember. В раскрывающемся списке атрибутов свойства DataSource показаны доступные источники данных. Экземпляр компонента `AdventureWorksDataSet`, добавленный на форму, перечислен в разделе `CustomerForm List Instances`. Выбрав тип `AdventureWorksDataSet` в узле Project Data Sources, можно создать другой экземпляр на форме, вместо повторного использования существующего объекта DataSet. В поле DataMember необходимо указать таблицу для использования в компоненте DataBinding. Позднее мы увидим, как поле DataMember можно использовать для задания отношения, в котором определен внешний ключ, чтобы увидеть связанные с ним данные.

Итак, мы указали, что компонент CustomerBindingSource будет связан с данными из таблицы Customer в компоненте AdventureWorksDataSet. Осталось связать отдельные элементы управления, находящиеся на форме, с компонентом BindingSource и соответствующим столбцом в таблице Customer. Для этого необходимо задать компонент DataBinding для каждого элемента управления. На рис. 27.8 показана сетка свойств Properties для поля FirstNameTextBox. Узел DataBindings раскрыт, чтобы показать связь со свойством Text.

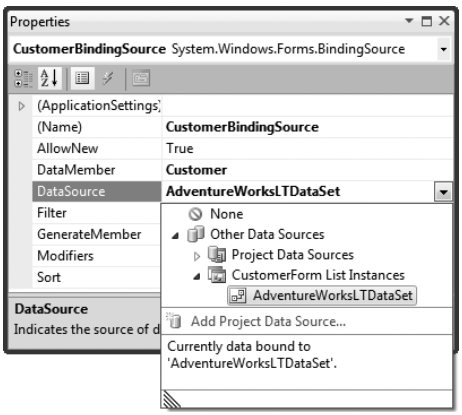


Рис. 27.7

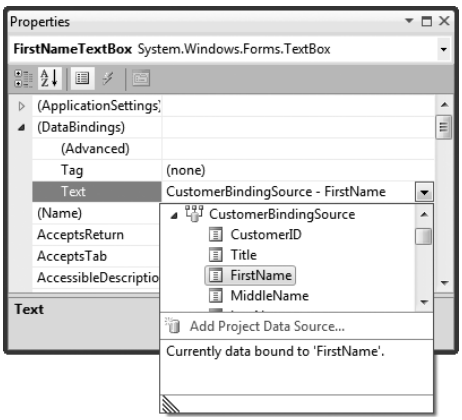


Рис. 27.8

В раскрывающемся списке видно, что свойство Text связано с полем FirstName из компонента CustomerBindingSource. Поскольку компонент CustomerBindingSource связан с таблицей Customer, свойство Text фактически связано со столбцом FirstName в этой таблице. Если заглянуть в файл проектировщика для этой формы, то можно увидеть, что эта связь установлена с помощью нового компонента Binding, как показано в следующем фрагменте кода.

```
Me.FirstNameTextBox.DataBindings.Add(
                    New System.Windows.Forms.Binding("Text",
                                                     Me.CustomerBindingSource,
                                                     "FirstName", True))
```

Компонент Binding используется для того, чтобы гарантировать, что между свойством Text поля FirstNameTextBox и полем FirstName из компонента CustomerBindingSource установлена двусторонняя связь. Все элементы управления для других элементов управления имеют аналогичные связи между их свойством Text и соответствующими полями компонента CustomerBindingSource.

Когда текущее приложение будет запущено на выполнение, можно заметить, что значение Modified Date отображается в формате представления даты, принятом по умолчанию, например “13/10/2004”. С учетом природы приложения может оказаться удобнее использовать другой формат даты, например “Friday, 13 October 2004”. Для этого необходимо задать дополнительные свойства компонента DataBinding. Выберите компонент ModifiedDateLabel1, раскройте узел DataBindings в инструментальном окне Properties и выберите элемент Advanced, чтобы открыть диалоговое окно Formatting and Advanced Binding (рис. 27.9).

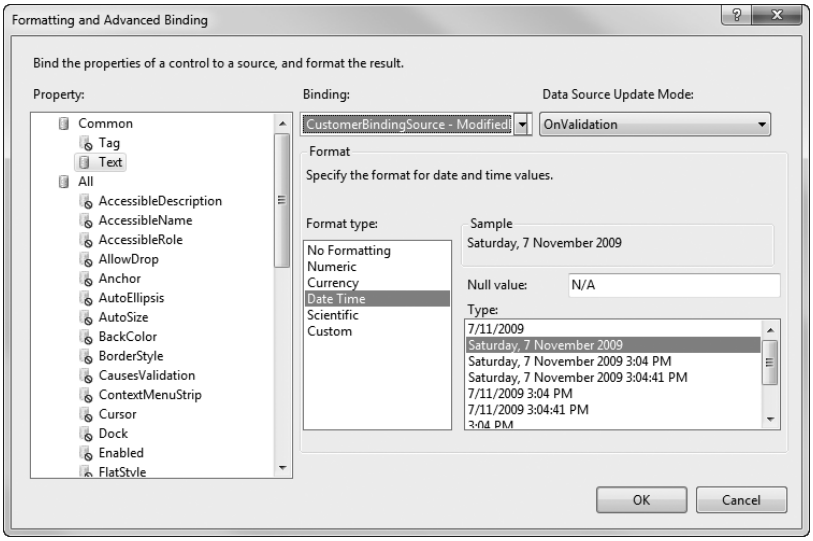


Рис. 27.9

В нижней части рис. 27.9 можно увидеть, что мы выбрали один из встроенных типов форматирования — Date Time. Ему соответствует список вариантов форматирования, в котором в качестве примера форматирования значений выбран пункт "Saturday, 7 November 2009". В этом диалоговом окне также задано значение "N/A", которое должно выводиться на экран, если в соответствующей строке нет значения `Modified Date`. В следующем фрагменте кода видно, что для создания компонента DataBinding для значения `Modified Date` мы добавили три дополнительных параметра.

VB

```
Me.ModifiedDateLabel1.DataBindings.Add(
            New System.Windows.Forms.Binding("Text",
                        Me.CustomerBindingSource,
                        "ModifiedDate", True,
                        System.Windows.Forms.DataSourceUpdateMode.OnValidation,
                        "N/A", "D"))
```

Значение `OnValidation` означает, что источник данных будет обновлен, когда визуальный элемент управления пройдет проверку корректности. Это значение задается по умолчанию и только в этом месте, чтобы можно было задать следующие два параметра. `N/A` — это значение, которое используется, когда нет соответствующего значения `Modified Date`, а `D` — это сокращенная строка форматирования для выбранного формата данных.

Компонент BindingNavigator

Несмотря на то что компонент `CustomerBindingNavigator`, являющийся экземпляром класса `BindingNavigator`, появляется на рабочей поверхности проектировщика в области невидимых компонентов, он имеет визуальное представление в виде инструментальной панели, прикрепленной к верхнему краю формы. Как и обычные инструментальные панели, этот элемент управления можно прикрепить к любому краю формы. По существу, во многом компонент `BindingNavigator` ведет

себя точно так же, как инструментальная панель, кнопки и другие элементы управления которой можно добавлять в список Items. Когда компонент BindingNavigator впервые добавляется на форму, он содержит ряд кнопок со стандартными функциональными возможностями для работы с данными, например для перехода к первому или последнему элементу, а также для добавления, удаления и сохранения элементов.

Компонент BindingNavigator не только создает эти стандартные элементы управления, но и связывает их друг с другом. На рис. 27.10 показано окно Properties для компонента BindingNavigator, в котором раскрыты разделы Data и Items. В разделе Data можно увидеть, что ассоциированным компонентом BindingSource является элемент CustomerBindingSource, который используется для выполнения всех действий, соответствующих разным вариантам щелчков мыши. Раздел Items играет важную роль, потому что каждое свойство определяет действие, например AddNewItem. Значение свойства определяет элемент ToolStripItem, которому оно соответствует, в данном случае — кнопку BindingNavigatorAddNewItem.

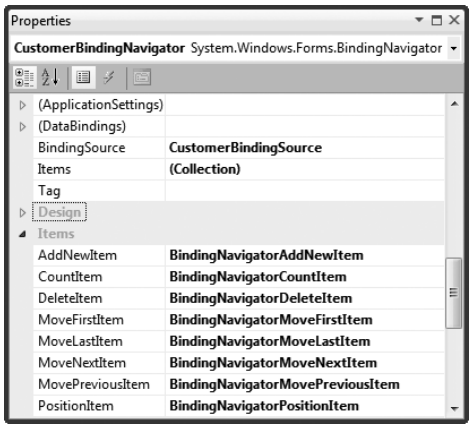


Рис. 27.10

Когда приложение уже выполняется и кнопка присвоена свойству AddNewItem, метод OnAddNew неявно связывается с событием Click для этой кнопки. Это демонстрирует следующий фрагмент кода, извлеченный из класса BindingNavigator с помощью надстройки Reflector. Свойство AddNewItem вызывает метод WireUpButton, передаваясь в делегате методу OnAddNew.

VB

```
Public Property AddNewItem As ToolStripItem
    Get
          If ((Not Me.addNewItem Is Nothing) AndAlso Me.addNewItem.IsDisposed)
               Then                Me.addNewItem = Nothing
          End If
          Return Me.addNewItem
    End Get
    Set(ByVal value As ToolStripItem)
        Me.WireUpButton(Me.addNewItem, value, _
                                    New EventHandler(AddressOf Me.OnAddNew))
    End Set
End Property

Private Sub OnAddNew(ByVal sender As Object, ByVal e As EventArgs)
    If (Me.Validate AndAlso (Not Me.bindingSource Is Nothing)) Then
          Me.bindingSource.AddNew
          Me.RefreshItemsInternal
    End If
End Sub

Private Sub WireUpButton(ByRef oldButton As ToolStripItem, _
                         ByVal newButton As ToolStripItem, _
                         ByVal clickHandler As EventHandler)
    If (Not oldButton Is newButton) Then
        If (Not oldButton Is Nothing) Then
```

```
            RemoveHandler oldButton.Click, clickHandler
        End If
        If (Not newButton Is Nothing) Then
            AddHandler newButton.Click, clickHandler
        End If
        oldButton = newButton
        Me.RefreshItemsInternal
    End If
End Sub
```

Метод `OnAddNew` выполняет ряд важных действий. Во-первых, он инициирует проверку корректности активного поля, которая будет описана позднее в этой главе. Во-вторых, и это наиболее важный аспект метода `OnAddNew`, он вызывает метод `AddNew` из класса `BindingSource`. Другие свойства объекта `BindingNavigator` также отображаются в соответствующие методы класса `BindingSource`, и важно помнить, что именно объект класса `BindingSource`, а не класса `BindingNavigator` выполняет работу, когда он получает доступ к источнику данных.

Выбор источников данных

После того как мы разобрались, как работает компонент `BindingSource`, настало время улучшить пользовательский интерфейс. В настоящий момент поле `Sales Person` отображается на экране в виде поля ввода, но оно должно быть ограничено продавцами компании AdventureWorks. Итак, вместо текстового поля было бы намного лучше, чтобы на экран выводился раскрывающийся список сотрудников, в котором пользователь может выбрать нужную фамилию.

Начните с удаления элемента `SalesPersonTextBox` из формы. Затем добавьте на форму элемент управления ComboBox из инструментальной панели. Выбрав новый элемент ComboBox, обратите внимание на то, что к нему прикреплен интеллектуальный дескриптор. Развернув этот дескриптор и установив флажок Use Data Bound Items, откройте окно, содержащее опции Data Binding Mode (рис. 27.11).

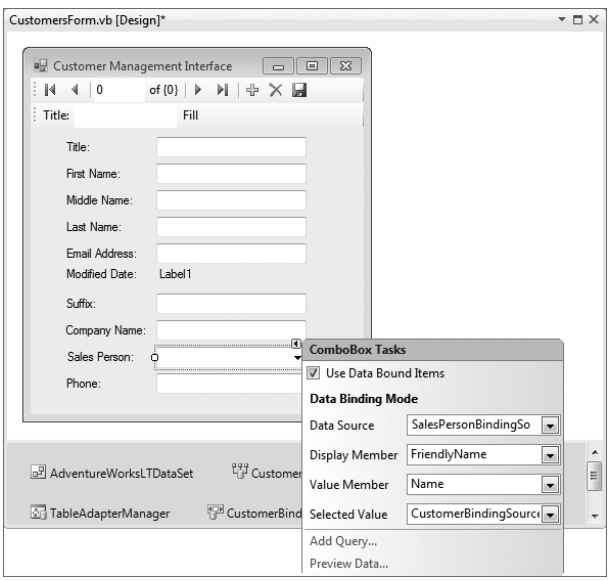


Рис. 27.11

Для того чтобы компонент DataBinding работал правильно, необходимо определить четыре аспекта. Во-первых, следует обеспечить возможность выбора источника данных для списка сотрудников, из которого пользователь будет делать выбор. К сожалению, список сотрудников не содержится в базе данных (впрочем, он может принадлежать базе данных, если он поступает из отдельной системы, скажем, из службы каталогов Active Directory). В данном примере список сотрудников определен как фиксированный массив объектов класса SalesPerson.



VB

```
Public Class SalesPerson
    Public ReadOnly Property FriendlyName
        Get
            Return Name.Replace("adventure-works\", String.Empty)
        End Get
End Property

Public Property Name As String

Public Shared Function Staff() As SalesPerson()
    Return {
       New SalesPerson() With {.Name = "adventure-works\pamela0"},
       New SalesPerson() With {.Name = "adventure-works\david8"},
       New SalesPerson() With {.Name = "adventure-works\jillian0"},
       New SalesPerson() With {.Name = "adventure-works\garrett1"},
       New SalesPerson() With {.Name = "adventure-works\jae0"},
       New SalesPerson() With {.Name = "adventure-works\linda3"},
       New SalesPerson() With {.Name = "adventure-works\jos й 1"},
       New SalesPerson() With {.Name = "adventure-works\michael9"},
       New SalesPerson() With {.Name = "adventure-works\shu0"}
    }
    End Function
End Class
```

Сниппет кода SalesPerson.vb



C#

```
public class SalesPerson{
    public string FriendlyName{
        get{
            return Name.Replace(@"adventure-works\", String.Empty);
        }
    }

public string Name { get; set; }

public static SalesPerson[] Staff(){
    return new SalesPerson[]{
                    new SalesPerson() {Name= @"adventure-works\pamela0"},
                    new SalesPerson() {Name= @"adventure-works\david8"},
                    new SalesPerson() {Name= @"adventure-works\jillian0"},
                    new SalesPerson() {Name= @"adventure-works\garrett1"},
                    new SalesPerson() {Name= @"adventure-works\jae0"},
                    new SalesPerson() {Name= @"adventure-works\linda3"},
                    new SalesPerson() {Name= @"adventure-works\jos й 1"},
                    new SalesPerson() {Name= @"adventure-works\michael9"},
                    new SalesPerson() {Name= @"adventure-works\shu0"}
            };
    }
}
```

Сниппет кода SalesPerson.cs

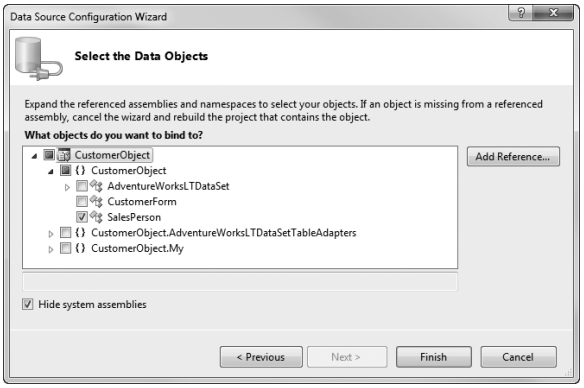


Рис. 27.12

Раскрыв список Data Source, можете выбрать любой существующий источник данных. Несмотря на то что список сотрудников, возвращенный методом `Staff` класса `SalesPerson`, содержится в проекте, его пока нельзя использовать как источник данных. Во-первых, необходимо добавить в проект новый источник данных `Object`. Это можно сделать, непосредственно выбрав ссылку Add Project DataSource в раскрывающемся списке Data Source. Эта ссылка вызывает мастер Data Source Configuration Wizard, описанный выше. Однако на этот раз в качестве типа источника данных выберите компонент `Object`. Затем выберите объект или объекты, которые следует включить в источник данных (рис. 27.12).

После того как выберете объект класса `SalesPerson` и щелкнете на кнопке Finish, будет создан источник данных, который будет автоматически присвоен свойству Data Source раскрывающегося списка `Sales Person`. Свойства Display Member и Value Member соответствуют свойствам объекта класса `SalesPerson`, которые должны выводиться на экран и использоваться для выбора элемента. В данном случае в классе `SalesPerson` определено свойство "только для чтения" с именем `FriendlyName`, которое просто удаляет префикс "adventure-works" и выводится на экране раскрывающегося списка. Однако свойство `Value` должно быть установлено равным свойству `Name`, чтобы соответствовать значению, введенному в поле `SalesPerson` в таблице `Customer`. И наконец, свойство `Selected Value` должно быть установленным равным свойству `SalesPerson` объекта `CustomerBindingSource`. Это свойство определяет объект класса `Sales Person` для объекта `Customer`, который выведен на экран.

Несмотря на то что вы собрали раскрывающийся список `Sales Person`, если вы запустите приложение в текущем виде, этот список окажется пустым, поскольку вы не заполнили объект `SalesPersonBindingSource`. Объект `BindingSource` имеет свойство `DataSource`, которое необходимо установить, чтобы заполнять объекты `BindingSource`. Это можно сделать на форме в событии `Load`.



VB

```
Private Sub CustomerForm_Load(ByVal sender As Object,
                              ByVal e As EventArgs) Handles MyBase.Load
        Me.SalesPersonBindingSource.DataSource = SalesPerson.Staff
End SubPrivate
```

Сниппет кода CustomersForm.vb



C#

```
private void CustomerForm_Load(object sender, EventArgs e){
    this.salesPersonBindingSource.DataSource = SalesPerson.Staff();
}
```

Сниппет кода CustomersForm.cs

Теперь, если выполнить приложение, вместо поля ввода с числовым значением вы получите удобный раскрывающийся список, в котором можно выбирать объект класса `SalesPerson`.

Сохранение изменений

Итак, вы получили удобный интерфейс и теперь должны обеспечить поддержку для внесения изменений и добавления новых записей. Если дважды щелкнуть на пиктограмме Save, которая находится на инструментальной панели CustomerBindingNavigator, откроется окно для редактирования кода, содержащее заглушку, обеспечивающую сохранение изменений в таблице Customer. Как показано в следующем фрагменте кода, есть три основных этапа: проверяется правильность формы, каждый объект BindingSource получает команду завершить текущее редактирование, а затем вызывается метод UpdateAll из класса TableAdapterManager.

VB

```
Private Sub CustomerBindingNavigatorSaveItem_Click(ByVal sender As Object,
                                                   ByVal e As System.EventArgs) _
                                Handles CustomerBindingNavigatorSaveItem.Click
    Me.Validate()
    Me.CustomerBindingSource.EndEdit()
    Me.TableAdapterManager.UpdateAll(Me.AdventureWorksLTDataSet)
End Sub
```

C#

```
private void customerBindingNavigatorSaveItem_Click(object sender, EventArgs e){
    this.Validate();
    this.customerBindingSource.EndEdit();
    this.tableAdapterManager.UpdateAll(this.adventureWorksLTDataSet);
}
```

Этот код будет выполнен без модификаций, но он не обновляет поле ModifiedDate, чтобы отметить, что информация в объекте Customer была изменена. Необходимо исправить метод Update, используемый компонентом CustomerTableAdapter, чтобы автоматически обновить поле ModifiedDate. Используя проектировщик DataSet, выберите компонент CustomerTableAdapter, откройте окно Properties, разверните узел UpdateCommand и щелкните на кнопке с эллипсисом, расположенным рядом с полем CommandText. В результате откроется диалоговое окно Query Builder, описанное ранее. Сбросьте флажок, расположенный на пересечении столбца Set и строки универсального идентификатора (поскольку он никогда не изменяется). В столбце New Value измените значение @ModifiedDate на getdate(), чтобы автоматически устанавливать модифицированную дату равной дате выполнения запроса. В результате должен сгенерироваться запрос, аналогичный показанному на рис. 27.13.

После внесения этих изменений при сохранении записи в поле ModifiedDate будет автоматически заноситься текущая дата.

Вставка новых компонентов

Итак, вы получили простое приложение, позволяющее просматривать и вносить изменения в существующий набор индивидуальных клиентов. Осталось только обеспечить возможность создавать новых клиентов. Как показано выше, по умолчанию кнопка Add на компоненте BindingNavigator автоматически связана с методом AddNew из класса BindingSource. В данном случае нам на самом деле нужны некие значения, установленные по умолчанию для записи, созданной в таблице Customer. Для этого необходимо описать логику работы кнопки Add.

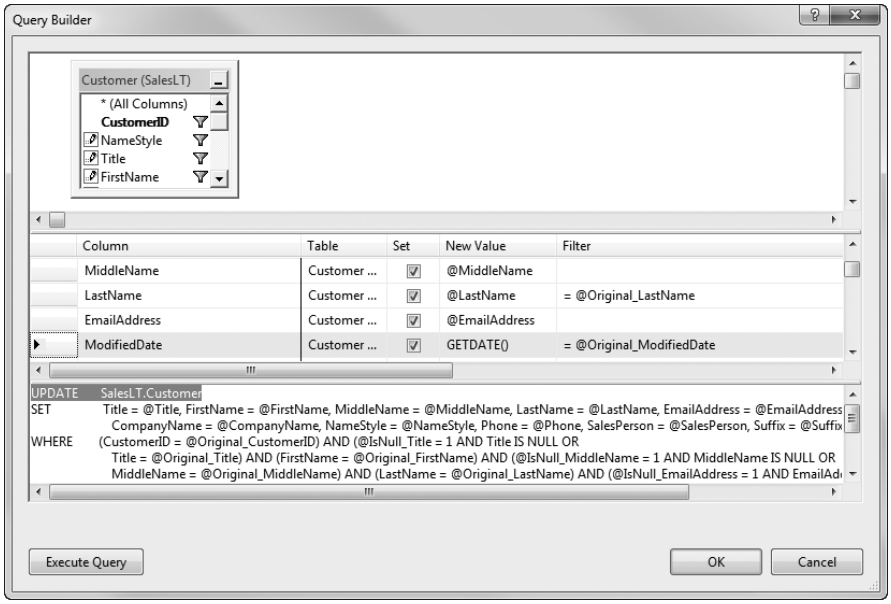


Рис. 27.13

На первом этапе удалите автоматическую связь, установив свойство `AddNewItem` объекта `CustomerBindingNavigator` равным `(None)`, в противном случае после каждого щелчка на кнопке Add будут создаваться две записи. Затем дважды щелкните на кнопке Add, чтобы создать для нее обработчик события. После этого можете изменить обработчик события, заданный по умолчанию, как показано ниже, чтобы задавать начальные значения для нового клиента, а также создавать записи в двух других таблицах.

VB

```
Private Sub BindingNavigatorAddNewItem_Click(ByVal sender As System.Object,
                                             ByVal e As System.EventArgs) _
                                    Handles BindingNavigatorAddNewItem.Click
    Dim drv As DataRowView

    'Создает запись в таблице Customer
    drv = TryCast(Me.CustomerBindingSource.AddNew, DataRowView)
    Dim customer = TryCast(drv.Row, AdventureWorksLTDataSet.CustomerRow)
    customer.rowguid = Guid.NewGuid
    customer.PasswordHash = String.Empty
    customer.PasswordSalt = String.Empty
    customer.ModifiedDate = Now
    customer.FirstName = " <first name> "
    customer.LastName = " <last name> "
    customer.NameStyle = False
    Me.CustomerBindingSource.EndEdit()
End Sub
```

C#

```
private void bindingNavigatorAddNewItem_Click(object sender,
                                              EventArgs e){
    DataRowView drv;

    //Создает запись в таблице Customer
```

```
        drv = this.customerBindingSource.AddNew() as DataRowView;
        var customer = drv.Row as AdventureWorksLTDataSet.CustomerRow;
        customer.rowguid = Guid.NewGuid();
        customer.PasswordHash = String.Empty;
        customer.PasswordSalt = String.Empty;
        customer.ModifiedDate = DateTime.Now;
        customer.FirstName = " <first name> ";
        customer.LastName = " <last name> ";
        customer.NameStyle = false;
        this.customerBindingSource.EndEdit();
    }
```

Может показаться, что вы напрасно установили несколько свойств, — например, `PasswordSalt` и `PasswordHash` заданы равными пустой строке. Необходимо убедиться, что новая строка соответствует ограничениям, установленным для базы данных. Поскольку эти поля не могут устанавливаться пользователем, необходимо гарантировать, что они инициализированы значениями, которые могут быть приняты базой данных. Очевидно, с точки зрения безопасности поля `PasswordSalt` и `PasswordHash` должны быть установлены с соответствующими значениями.

Такое выполнение приложения вместо автоматического связывания обработчика событий позволяет создавать новую запись о клиенте с помощью кнопки Add. Если значения введены в каждое из этих полей, то эти изменения можно сохранить.

Проверка корректности

В предыдущем разделе мы добавили функциональную возможность создавать новую запись о клиенте. Если при создании новой записи ввести неправильные данные, например, забыть ввести фамилию, то запись будет отвергнута, когда пользователь щелкнет на кнопке Save. Схема базы данных `AdventureWorksDataSet` содержит много ограничений, например, поле `FirstName` не может быть нулевым. Это условие проверяется при выполнении определенных действий, например, при сохранении или при переходе от одной записи к другой. Если проверка заканчивается неудачей, то генерируется исключение. Существуют два варианта. Во-первых, можно перехватывать эти исключения, что в принципе является плохой практикой программирования, поскольку исключения нельзя использовать для управления процессом выполнения приложения. Во-вторых, исключения можно предотвратить, проверяя данные до проверки самой схемы. Ранее в главе при изучении того, как компонент `BindingNavigator` автоматически связывает метод `AddNew` с компонентом `BindingSource`, мы видели, что метод `OnAddNew` содержит вызов метода `Validate`. Этот метод в свою очередь вызывает метод `Validate` из активного элемента управления, который возвращает булево значение, определяющее, будет ли выполняться данное действие. Этот образ действий используется всеми автоматически связываемыми событиями и должен использоваться в обработчиках событий для кнопок навигации.

Метод `Validate` из активного элемента управления порождает два события — `Validating` и `Validated`, — которые возникают до и после процесса проверки корректности соответственно. Поскольку вы хотите управлять процессом проверки правильности, добавьте обработчик события `Validating`. Например, вы могли бы добавить обработчик события `Validating` в элемент управления FirstNameTextBox.

VB

```
Private Sub FirstNameTextBox_Validating(ByVal sender As System.Object, _
                            ByVal e As System.ComponentModel.CancelEventArgs) _
```

```
                                                Handles FirstNameTextBox.Validating
    Dim firstNameTxt As TextBox = TryCast(sender, TextBox)
    If firstNameTxt Is Nothing Then Return
    e.Cancel = (firstNameTxt.Text = String.Empty)
End Sub
```

C#

```
    private void firstNameTextBox_Validating(object sender, CancelEventArgs e){
    var firstNameTxt = sender as TextBox;
    if (firstNameTxt == null) return;
    e.Cancel = (firstNameTxt.Text == String.Empty);
}
```



Рис. 27.14

Несмотря на то что этот метод не дает пользователю выйти из поля ввода до его завершения, он ничего не сообщает ему о причинах своих действий. К счастью, в платформе .NET Framework есть элемент управления ErrorProvider, который можно перетащить на форму из инструментального окна Toolbox. Этот элемент управления ведет себя аналогично подсказке. Для каждого элемента управления, находящегося на форме, пользователь может задать строку Error. В результате рядом с соответствующим элементом управления будут появляться пиктограмма и подсказка, содержащая строку Error. Эта ситуация проиллюстрирована на рис. 27.14, где строка Error задана равной значению поля `FirstNameTextBox`.

Очевидно, что для поля `FirstNameTextBox` нужно задать строку Error только для ситуации, когда в поле нет текста. Основываясь на примере, в котором мы добавляли обработчик события `Validating`, можно модифицировать этот код, задав строку Error.

VB

```
Private Sub FirstNameTextBox_Validating(ByVal sender As System.Object, _
                            ByVal e As System.ComponentModel.CancelEventArgs) _
                                                Handles FirstNameTextBox.Validating
    Dim firstNameTxt As TextBox = TryCast(sender, TextBox)
    If firstNameTxt Is Nothing Then Return
    e.Cancel = (firstNameTxt.Text = String.Empty)

    If String.IsNullOrWhiteSpace(firstNameTxt.Text) Then
       Me.ErrorProvider1.SetError(firstNameTxt, "First Name must be specified")
    Else
       Me.ErrorProvider1.SetError(firstNameTxt, Nothing)
    End If
End Sub
```

C#

```
private void firstNameTextBox_Validating(object sender, CancelEventArgs e){
    var firstNameTxt = sender as TextBox;
```

```
if (firstNameTxt == null) return;
    e.Cancel = (firstNameTxt.Text == String.Empty);
    if (String.IsNullOrEmpty(firstNameTxt.Text)){
        this.errorProvider1.SetError(firstNameTxt,
                                     "First Name must be specified");
   }
   else{
        this.errorProvider1.SetError(firstNameTxt, null);
   }
}
```

Можно представить себе ситуацию, в которой описание обработчиков событий, проверяющих корректность данных для каждого элемента управления и задающих соответствующие сообщения об ошибках, представляет собой очень долгий и трудоемкий процесс. Вместо проверки корректности каждого элемента управления по отдельности данный процесс можно оптимизировать, введя единственный обработчик событий, который делегирует проверку корректности соответствующему классу контроллера. Это позволяет избежать смешения бизнес-логики и пользовательского интерфейса.

Настраиваемые объекты DataSet

Итак, мы создали форму, содержащую некую информацию о клиенте. Однако у нас нет адреса клиента, а именно, нет полей `Main Office Address` и/или `Shipping Address`. Если посмотреть на структуру базы данных `AdventureWorksLT`, то можно заметить, что между таблицами `Customer` и `Address` есть отношение "многие ко многим", реализуемое посредством таблицы ссылок `CustomerAddress`. Таблица `CustomerAddress` содержит столбец `AddressType`, содержащий информацию о типе адреса. И хотя данная структура допускает, что несколько клиентов могут иметь одинаковые адреса, пользовательский интерфейс, созданный нами, интересует только адрес конкретного клиента. Если просто добавить все три эти таблицы в объект DataSet, то будет нелегко связать их данные с пользовательским интерфейсом. По этой причине целесообразно настроить генерируемый объект DataSet так, чтобы он объединял таблицы `CustomerAddress` и `Address` в одно целое.

Откройте проектировщик объектов DataSet, дважды щелкнув на файле `AdventureWorksLTDataSet.xsd` в окне Solution Explorer. Выберите узел AddressTableAdapter, который вам уже встречался ранее, раскройте его свойство SelectCommand в инструментальном окне Properties и щелкните на эллипсисе, расположенном справа от свойства CommandText. В результате вновь откроется надстройка Query Builder. Теперь вам нужна таблица `Address` на панели диаграмм. Щелкните правой кнопкой мыши на этой панели и выберите команду Add Table, а затем таблицу `CustomerAddress`. Проверьте все поля в таблице `CustomerAddress`, за исключением поля `AddressID`, затем перейдите на панель Criteria и измените параметр `Alias` для столбца, содержащего универсальный идентификатор, и столбца `ModifiedDate` из таблицы `CustomerAddress`. Результат должен выглядеть примерно так, как показано на рис. 27.15.

Когда вы щелкнете на кнопке OK, вы получите приглашение повторно сгенерировать инструкции `Update` и `Insert`. Генератор кода не поддерживает обновление нескольких таблиц, поэтому даст сбой, независимо от того, какой из вариантов вы выберете. Это значит, что вы должны вручную определить инструкции обновления, вставки и удаления. Это можно сделать с помощью определения хранимых процедур для базы данных `AdventureWorksLT` и изменения свойств `CommandType` и `CommandText` для соответствующих команд в компоненте `AddressTableAdapter`, как показано на рис. 27.16.

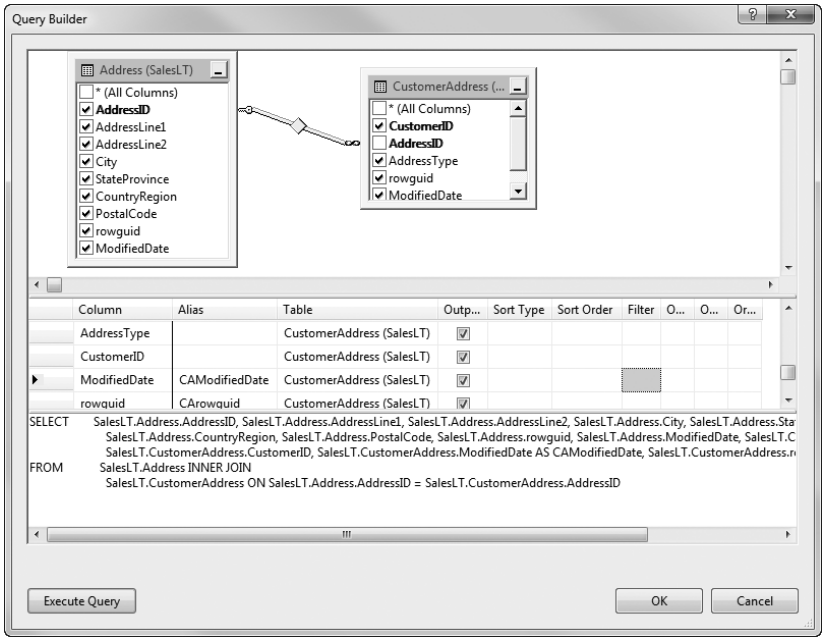


Рис. 27.15

Теперь объект DataSet содержит объекты `Customer` и `Address` типа DataTable, и единственное, чего вам не достает, — это отношения, связывающего их друг с другом. Поскольку вы самостоятельно настроили объект `Address` DataTable, проектировщик не может автоматически создать это отношение. Для того чтобы создать отношение, щелкните правой кнопкой мыши на рабочей области проектирования объектов DataSet и выберите команду Add⇨Relation. В результате откроется диалоговое окно Relation (рис. 27.17).

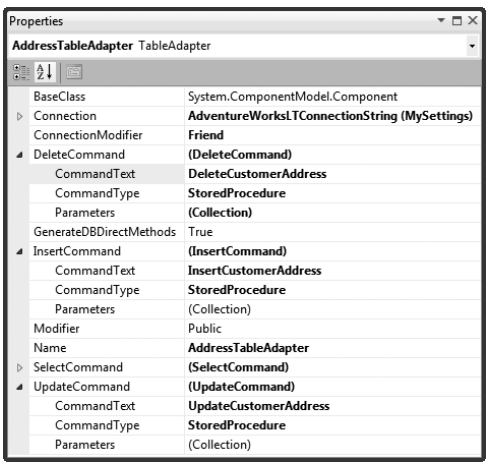


Рис. 27.16



Рис. 27.17

Точно так же как был создан объект Address DataTable, объединяющий таблицы CustomerAddress и Address, создайте базовый класс Customer DataTable и производный класс Address. В диалоговом окне вы увидите линию отношения, соединяющую два объекта DataTable в рабочей области проектирования объектов DataSet.

Цепочки объектов BindingSource и DataGridView

Завершив настройку объекта DataSet с помощью объектов Customer и Address типа DataTable, вы готовы связать данные из базы Address с пользовательским интерфейсом. До сих пор вы работали с простыми элементами управления для ввода данных, такими как поля ввода, раскрывающиеся списки и метки. При этом вы видели, что компонент BindingNavigator позволяет прокручивать список элементов. Иногда более удобно представлять данные в виде сетки. В этом вам поможет элемент DataGridView, сочетающий мощь компонента BindingSource и простоту сетки.

В данном примере мы дополним интерфейс Customer Management, добавив информацию об адресе с помощью компонента DataGridView. Вернувшись к окну Data Sources, выберите узел Address в узле Customer. В раскрывающемся списке выберите компонент DataGridView и перетащите этот узел на пустое пространство формы. При этом на форму будут добавлены соответствующие компоненты BindingSource и TableAdapter, а также компонент DataGridView, демонстрирующий каждый столбец в таблице Address (рис. 27.18).

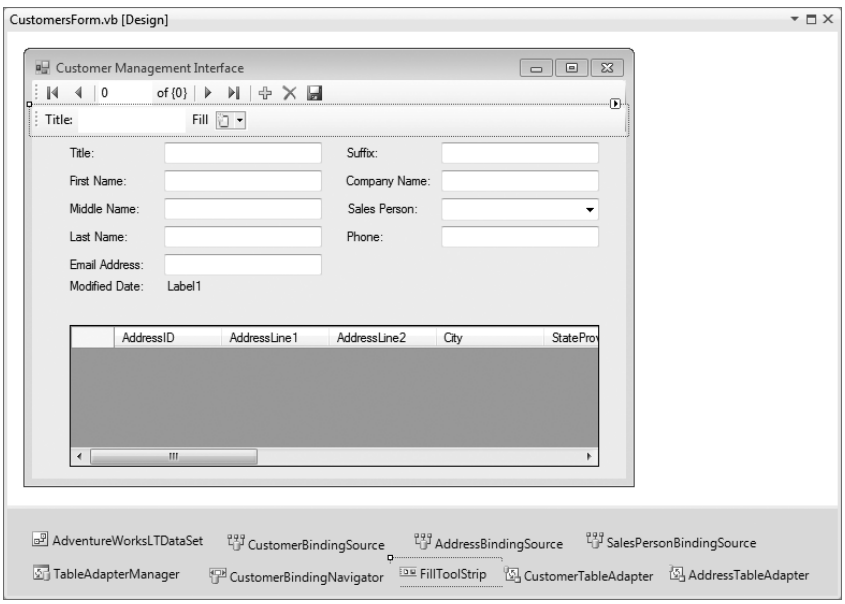


Рис. 27.18

Напомним, что объект CustomerBindingSource содержит компонент AdventureWorksLTDataSet типа DataSource, а таблицу Customer задает как объект типа DataMember. Это означает, что элементы управления, представляющие собой связь между данными с помощью объекта CustomerBindingSource, связаны с полем в таблице Customer. Если вы посмотрите на объект AddressBindingSource, то увидите, что его

источником данных на самом деле является объект CustomerBindingSource, в котором свойство DataMember установлено равным объекту Customer_Address, представляющему собой отношение, созданное между двумя объектами типа DataTable. Можно предположить, что любые элементы управления, используемые для связи между данными с помощью объекта AddressBindingSource, связаны с полем в таблице Address. Однако разница заключается в том, что в отличие от объекта CustomerBindingSource, возвращающего все объекты типа Customer, объект AddressBindingSource содержит только объекты типа Address, связанные с текущим выбранным объектом типа Customer.

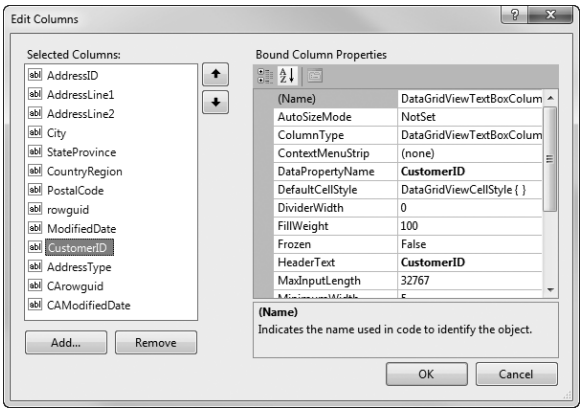


Рис. 27.19

В отличие от компоновки Details, когда вы перетаскиваете компонент DataGridView на форму, игнорируются все установки, заданные для отдельных столбцов. Вместо этого каждый столбец добавляется на сетку как простое поле ввода. Для того чтобы модифицировать список отображаемых столбцов, можно использовать интеллектуальный дескриптор для вновь добавленного компонента DataGridView или выбрать команду Edit Columns из контекстного меню. В открывшемся диалоговом окне Edit Columns (рис. 27.19) можно добавлять, удалять или переупорядочивать столбцы.

Указав соответствующие столбцы, запустите приложение на выполнение, и каждый пользователь базы данных увидит список заказов.

Работа с источниками данных

В этой главе мы работали со строго типизированными компонентами DataSet, содержащими многочисленные строки из таблицы Customer, основываясь на параметре Title. До сих пор наш пример имел только один ярус — само приложение типа Windows Forms. В этом разделе будет показано, как можно использовать систему Visual Studio 2010 для создания многоярусного приложения.

Начнем с создания двух новых проектов, CustomerBrowser (Windows Forms Application) и CustomerService (ASP.NET Web Service Application). Измените тип Application Type первоначального проекта на тип Class Library, дважды щелкнув на узле Properties в окне Solution Explorer, а затем изменив поле Application type на закладке Application.



Поскольку данный раздел посвящен работе с приложениями для платформы ASP.NET, если вы работаете под управлением операционной системы Windows Vista, рекомендуем запустить систему Visual Studio 2010 в режиме Administrator. Это позволит присоединить отладчик к соответствующему процессу.

В проекте типа Web Service добавьте ссылку на проект типа Class Library. Необходимо также модифицировать класс `Service`, чтобы он содержал два метода вместо веб-метода `HelloWorld`, заданного по умолчанию.



VB

```
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.ComponentModel
Imports CustomerObject

<System.Web.Services.WebService(Namespace:="http://tempuri.org/")> _
<System.Web.Services.WebServiceBinding(ConformsTo:=_
WsiProfiles.BasicProfile1_1)> _
<ToolboxItem(False)> _
Public Class CustomerService
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function RetrieveCustomers(ByVal Title As String) _
                      As AdventureWorksLTDataSet.CustomerDataTable
        Dim ta As New
            AdventureWorksLTDataSetTableAdapters.CustomerTableAdapter
        Return ta.GetData(Title)
    End Function

    <WebMethod()> _
    Public Sub SaveCustomers(ByVal changes As Data.DataSet)
        Dim changesTable As Data.DataTable = changes.Tables(0)
        Dim ta As New
            AdventureWorksLTDataSetTableAdapters.CustomerTableAdapter
        ta.Update(changesTable.Select)
    End Sub
End
```

Сниппет кода CustomerService.asmx.vb



C#

```
namespace CustomerService{
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    public class CustomerService : System.Web.Services.WebService{
        [WebMethod]
        public AdventureWorksLTDataSet.CustomerDataTable RetrieveCustomers
                                                          (string title){
            var ta = new CustomerObject.AdventureWorksLTDataSetTableAdapters.
                                                    CustomerTableAdapter();
            return ta.GetData(title);
        }

        [WebMethod()]
        public void SaveCustomers(DataSet changes){
            var changesTable = changes.Tables[0] as DataTable;
            var ta = new CustomerObject.AdventureWorksLTDataSetTableAdapters.
                                                    CustomerTableAdapter();
            ta.Update(changesTable.Select());
        }
    }
}
```

Сниппет кода CustomerService.asmx.cs

Первый веб-метод, как следует из его имени, находит список клиентов по параметру `Title`, который ему передается. В этом методе можно создать новый экземпляр строго типизированного класса `TableAdapter` и вернуть объект класса `DataTable`, обнаруженный методом `GetData`. Второй веб-метод используется для сохранения изменений, внесенных в объекте типа `DataTable`, причем снова используется строго типизированный класс `TableAdapter`. Легко заметить, что объект класса `DataSet`, который передается этому методу как параметр, не является строго типизированным. К сожалению, генерируемый строго типизированный класс `DataSet` не содержит строго типизированного метода `GetChanges`, который будет использован позднее для генерации объекта класса `DataSet`, содержащего только измененные данные. Этот новый объект класса `DataSet` передается методу `SaveCustomers`, чтобы посылать веб-службе только измененные данные.

Компоненты Web Service Data Source

Эти изменения, внесенные в веб-службу, завершают разработку серверной стороны процесса, но наше приложение по-прежнему не имеет доступа к данным. Для того чтобы получить доступ к данным из своего приложения, необходимо добавить в приложение источник данных. Вновь воспользуйтесь мастером Add New Data Source Wizard, но на этот раз выберите в окне Data Source Type пункт Service. Для того чтобы добавить компонент Web Service Data Source, щелкните на кнопке Advanced, а затем на ссылке Add Web Reference. Щелкнув на ссылке "Web services in this solution", можно увидеть список веб-служб, доступных в пользовательском решении. Веб-служба, с которой вы только что работали, должна появиться в этом списке. После щелчка на гиперссылке для этой веб-службы вы увидите кнопку Add Reference (рис. 27.20).

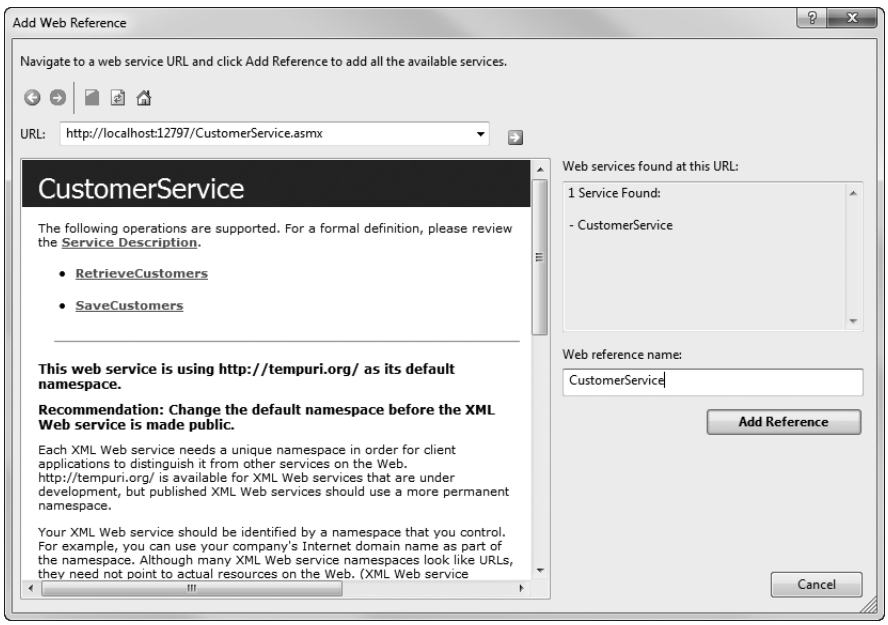


Рис. 27.20



Если после щелчка на гиперссылке появляется сообщение об ошибке, возможно, придется создать и выполнить проект типа ASP.NET Web Service Application. При этом будет запущен служба, позволяющая корректно извлечь информационную схему из диалогового окна Add Web Reference.

Щелчок на кнопке Add Reference добавляет компонент `AdventureWorksDataSet` в окно Data Sources в узле CustomerService. Развернув этот узел, вы увидите, что его источник данных очень похож на источник данных в библиотеке классов.

Просмотр данных

Для того чтобы действительно увидеть данные, возвращенные веб-службой, необходимо добавить на форму несколько элементов управления. Откройте форму так, чтобы в главном окне открылся проектировщик. Находясь в окне Data Sources, щелкните на узле Customer и выберите пункт Details из раскрывающегося списка. Это означает, что при перетаскивании узла Customer на форму система Visual Studio 2010 создаст элементы управления для визуализации деталей таблицы `Customer` (например, содержимое строк), а не компонента DataGridView, предусмотренного по умолчанию. Затем выберите атрибуты, которые хотите визуализировать, щелкая на них, и используемый тип элементов управления. После перетаскивания узла Customer на форму компоновка должна выглядеть примерно так, как показано на рис. 27.21.

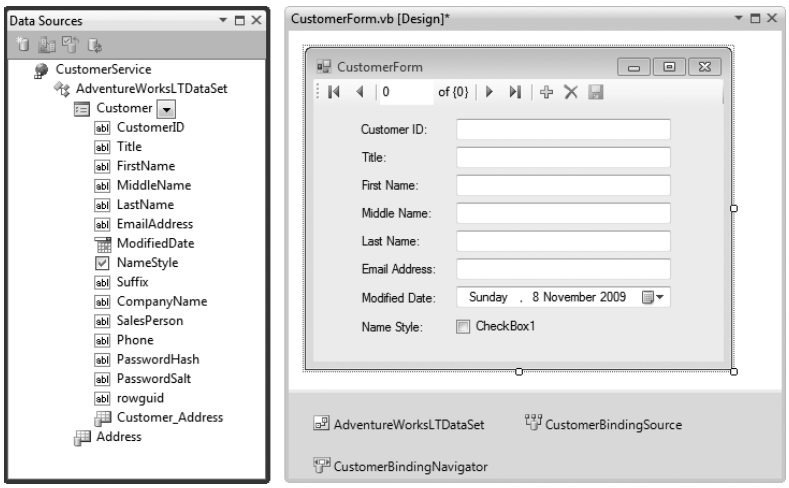


Рис. 27.21

Кроме элементов управления для визуализации и редактирования информации, в верхнюю часть формы был добавлен элемент управления Navigator, а в область невидимых компонентов — объекты `AdventureWorksDataSet` и `CustomerBindingSource`.

На последнем этапе необходимо связать событие Load с формой для извлечения данных из веб-службы и добавить кнопку Save на компонент навигации, чтобы сохранить изменения. Щелкните правой кнопкой мыши на пиктограмме сохранения и выберите команду Enabled, чтобы кнопка Save появилась на элементе управления для навигации, а затем дважды щелкните на пиктограмме сохранения, чтобы сгене-

рировать заглушку для обработчика событий. Добавьте следующий код для загрузки данных и сохраните изменения с помощью ранее созданной веб-службы.



VB

```
Public Class CustomerForm
    Private Sub CustomerForm_Load(ByVal sender As System.Object,
                                  ByVal e As System.EventArgs) Handles Me.Load
       Me.CustomerBindingSource.DataSource = _
       My.WebServices.CustomerService.RetrieveCustomers("%mr%")
    End Sub

    Private Sub CustomerBindingNavigatorSaveItem_Click _
                               (ByVal sender As System.Object,
                                ByVal e As System.EventArgs)
                                Handles
CustomerBindingNavigatorSaveItem.Click
       Me.CustomerBindingSource.EndEdit()
       Dim ds = CType(Me.CustomerBindingSource.DataSource, _
                 CustomerService.AdventureWorksLTDataSet.CustomerDataTable)
       Dim changesTable As DataTable = ds.GetChanges()
       Dim changes As New DataSet
       changes.Tables.Add(changesTable)
       My.WebServices.CustomerService.SaveCustomers(changes)
    End Sub
End Class
```

Сниппет кода CustomersForm.vb

C#

```
private void CustomersForm_Load(object sender, EventArgs e){
    var service = new CustomerService.CustomerService();
    this.CustomerBindingSource.DataSource =
service.RetrieveCustomers("%mr%"); ;
}

private void CustomerBindingNavigatorSaveItem_Click(object sender,
                                                    EventArgs e){
    this.CustomerBindingSource.EndEdit();
    var ds = this.CustomerBindingSource.DataSource
                as CustomerService.AdventureWorksLTDataSet.CustomerDataTable;
    var changesTable = ds.GetChanges();
    var changes = new DataSet();
    changes.Tables.Add(changesTable);
    var service = new CustomerService.CustomerService();
    service.SaveCustomers(changes);
}
```

Сниппет кода CustomersForm.cs

Для того чтобы извлечь список клиентов из веб-службы, необходимо лишь вызвать соответствующий веб-метод — в данном случае `RetrieveCustomers`. Передайте ему параметр `%mr%`, означающий, что возвращать необходимо только записи о клиентах, в которых поле `Title` содержит буквы "mr". Метод `Save` немного сложнее, поскольку вы должны завершить текущее редактирование (чтобы убедиться, что все изменения сохранены), извлечь компонент DataTable, а затем извлечь изменения из нового компонента DataTable. Хотя было бы проще передавать веб-службе `SaveCustomers` компонент DataTable, в качестве параметров и возвращаемых значений веб-службы можно указывать только компоненты DataSet. При этом можно создать новый компонент DataSet и добавить измененный компонент DataTable в список таблиц. Затем новый компонент DataSet передается в метод `SaveCustomers`. Как указывалось ранее, ме-

тод `GetChanges` возвращает простой объект `DataTable`, что достойно сожаления, потому что это ограничивает сценарии обработки строго типизированных данных.

На этом мы завершаем описание сценария работы со строго типизированными компонентами DataSet и двухъярусного решения для доступа и редактирования баз данных с помощью интерфейса веб-службы.

Резюме

Эта глава представляет собой введение в строго типизированные компоненты DataSet. Поддержка системы Visual Studio 2010 для создания и работы со строго типизированными компонентами DataSet упрощает быструю разработку приложений. Очевидно, что это лишь первый шаг в процессе построения моста над пропастью между объектно-ориентированным программированием и реляционными базами данных.

Надеемся, эта глава дала достаточно полное представление об элементах управления BindingSource, BindingNavigator и других элементах управления, позволяющих быстро разрабатывать приложения для работы с данными. Поскольку новые элементы управления поддерживают работу как с компонентами DataSet, так и со специальными пользовательскими объектами, они могут значительно сократить затраты времени на разработку приложений.

28

Технология Language Integrated Queries (LINQ)

В ЭТОЙ ГЛАВЕ...

- Создание запросов на объекты с помощью технологии LINQ
- Создание запросов и поиск в XML-документах с помощью методов XLINQ
- Создание запросов и обновление данных с помощью технологии LINQ для SQL

Язык интегрированных запросов LINQ (Language Integrated Queries) был разработан для того, чтобы создать общую модель программирования для запросов. В данной главе будет показано, как очень многословный императивный код можно превратить в несколько декларативных строк. Это позволяет сделать код более наглядным и менее традиционным; иначе говоря, теперь можно описывать, что должно случиться, а не детализировать, как это следует сделать.

Несмотря на то что технология LINQ позволяет легко фильтровать и сортировать данные и создавать проекты на основе объектных графов, хранящихся в памяти, гораздо чаще источником данных является база данных или файл, например XML-файл. В главе излагаются основы технологии LINQ to XML, существенно упрощающей работу с XML-данными по сравнению с традиционными методами, такими как объектная модель документов, XSLT или XPath. Кроме того, будет показано, как использовать технологию LINQ to SQL для работы с традиционными базами данных, такими как SQL Server, позволяя пользователю писать инструкции LINQ для запроса к базе данных, извлекать необходимые данные и наполнять ими объекты по технологии .NET. В главе 29 будет описана платформа ADO.NET Entity Framework, для которой также существует LINQ-провайдер. Это означает, что, управляя объектом данных на протяжении его жизненного цикла, пользователь может сочетать мощь декларативных запросов с точностью платформы Entity Framework.

Провайдеры LINQ

Одним из основных принципов технологии LINQ является способность абстрагировать синтаксис запроса от соответствующего способа хранения данных. Технология LINQ внедрена в разные языки платформы .NET, например C# и VB, и сочетает в себе разные языковые свойства, например *методы расширения* (extension methods), *вывод типов* (type inferences), *анонимные типы* (anonymous types) и *лямбда-выражения* (Lambda-expressions), обеспечивающие единообразный синтаксис для запросов.

Вместе с системой Visual Studio 2010 и платформой .NET Framework 4.0 поставляется большое количество источников данных, допускающих применение технологии LINQ: Objects, DataSets, SQL, Entities и XML; каждый из них имеет свой собственный провайдер LINQ, способный обращаться с запросами к соответствующему источнику данных. Технология LINQ не ограничена только этими источниками данных. Ее провайдеры могут обращаться с запросами к любым другим источникам данных. Например, существует провайдер LINQ для запросов к коллекции SharePoint. Документация, поставляемая вместе с системой Visual Studio 2010, содержит подробные инструкции, позволяющие пользователю создать свой собственный провайдер LINQ.

В настоящей главе рассматриваются стандартные операции языка LINQ, относящиеся к стандартным объектам платформы .NET. Кроме того, в ней показано, как одни и те же запросы можно применять как к XML-, так и к SQL-источникам данных. Легко убедиться, что при этом синтаксис запросов остается неизменным, изменяются только источники данных.

Традиционные запросы

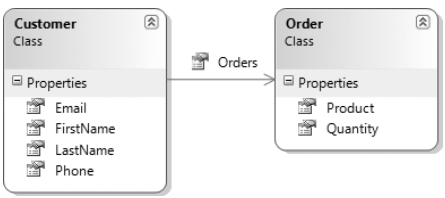


Рис. 28.1

Вместо детального разбора технологии LINQ начнем с примера, демонстрирующего некоторые преимущества новых запросов. В этом сценарии исследователь желает выяснить, существует ли корреляция между длиной имени клиента и средним объемом его заказа, проанализировав коллекцию объектов, описывающих клиентов. Как показано на рис. 28.1, отношения между клиентом и заказом представляют собой обычное отношение “один ко многим” (one-to-many relationship).

В рассматриваемом нами запросе исследователь ищет среднюю сумму заказа `Milk`, в котором имя клиента содержит пять или больше букв, причем записи упорядочены по имени клиента.



C#

```
private void OldStyleQuery(){
    Customer[] customers = BuildCustomers();
    List <SearchResult> results = new List <SearchResult> ();
    SearchForProduct matcher = new SearchForProduct() { Product = "Milk" };
    foreach (Customer c in customers){
        if (c.FirstName.Length> = 5){
            Order[] orders = Array.FindAll(c.Orders, matcher.ProductMatch);
            if (orders.Length> 0){
                SearchResult cr = new SearchResult();
                cr.Customer = c.FirstName + " " + c.LastName;
```

```
                foreach (Order o in orders){
                    cr.Quantity += o.Quantity;
                    cr.Count++;
                }
                results.Add(cr);
            }
        }
    }
    results.Sort(CompareSearchResults);
    ObjectDumper.Write(results, Writer);
}
```

Сниппет кода MainForm.cs



VB

```
Private Sub OldStyleQuery()
    Dim customers As Customer() = BuildCustomers()

    Dim results As New List(Of SearchResult)
    Dim matcher As New SearchForProduct() With {.Product = "Milk"}

    For Each c As Customer In customers
        If c.FirstName.Length> = 5 Then
            Dim orders As Order() = Array.FindAll(c.Orders, _
                                        AddressOf matcher.ProductMatch)
            If orders.Length> 0 Then
                Dim cr As New SearchResult
                cr.Customer = c.FirstName & " " & c.LastName
                For Each o As Order In orders
                    cr.Quantity += o.Quantity
                    cr.Count += 1
                Next
                results.Add(cr)
            End If
        End If
   Next
   results.Sort(AddressOf CompareSearchResults)

   ObjectDumper.Write(results, Writer)
End Sub
```

Сниппет кода MainForm.vb

Прежде чем мы погрузимся в анализ этого кода и покажем, как язык LINQ может улучшить этот сниппет, посмотрим, как он работает. В первых строках вызывается метод, который генерирует объекты типа `Customer`. Данный метод будет использоваться во всех сниппетах в этой главе. Основной цикл в этом методе просматривает массив объектов в поисках клиентов, имя которых состоит из пяти и более букв. Когда такой клиент будет найден, вызывается метод `Array.FindAll`, извлекающий все заказы, для которых этот предикат является истинным. Поскольку мы еще не знакомы с анонимными методами, мы не можем задать короткую предикатную функцию в этом методе. В результате приходится применять обычный способ и создавать простой класс, способный хранить переменную запроса (в данном случае строку `Milk`), являющуюся целью поиска, и метод, получающий типа объекта, который мы ищем, в данном случае тип `Order`. Ознакомившись с лямбда-выражениями, мы сможем переписать этот код в следующем виде.

C#

```
var orders = Array.FindAll(c.Orders, order=> order.Product =="Milk");
```

VB

```
Dim orders = Array.FindAll(c.Orders,
                           Function(o As Order) o.Product = "Milk")
```

Здесь мы видим преимущества вывода типов, позволяющие определить тип переменных, описывающих заказ, который, разумеется, остается массивов заказов.

Вернемся к сниппету. Как только мы найдем заказы, нам потребуется просмотреть их, просуммировать заказанные количества и сохранить их вместе с именами клиентов и количествами заказов. Именно эта информация является результатом поиска, и, как легко убедиться, для ее хранения мы используем объект типа `SearchResult`. Для удобства объект типа `SearchResult` имеет свойство `Average`, предназначенное только для чтения, которое просто делит общий объем заказанного товара на количество заказов. Поскольку нам необходимо упорядочить список клиентов, мы используем метод `Sort` класса `List`, передавая ему адрес метода сравнения. И снова использование лямбда-выражений позволяет переписать этот код в виде лаконичной инструкции.

C#

```
results.Sort((r1, r2) => string.Compare(r1.Customer, r2.Customer));
```

VB

```
results.Sort( Function(r1 as SearchResult, r2 as SearchResult) _
                                    String.Compare(r1.Customer, r2.Customer))
```

Последняя часть сниппета выводит результаты поиска на печать. Для этого мы использовали готовый объект `ObjectDumper`, который поставляется вместе с системой Visual Studio 2010. Это простой класс, который проходит по коллекции объектов, выводя на печать значения открытых свойств. В данном случае вывод выглядит так, как показано ниже (рис. 28.2).

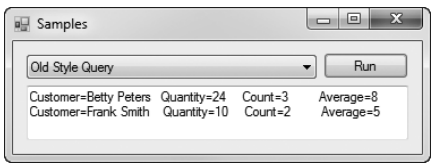


Рис. 28.2

Этот относительно простой запрос свидетельствует о том, что традиционный код слишком подробный и требует дополнительных классов для реализации логики запроса и получения результатов. Используя мощь технологии LINQ, пользователь может создать простое выражение, которое ясно описывает, каким должен быть результат поиска.

Фрагменты запроса

В этом разделе рассматриваются многочисленные операции запроса, образующие основу языка LINQ. Люди, привыкшие писать инструкции SQL, увидят несомненное сходство между ними, хотя порядок и синтаксис инструкций может немного отличаться. Существует множество операций запроса и огромное количество веб-сайтов, содержащих инструкции по их использованию. В данный момент нас интересуют операции, необходимые для улучшения запроса на поиск, представленного в начале главы.

Инструкция `From`

В отличие от языка SQL, в котором первой является инструкция `Select`, в языке LINQ первой обычно является инструкция `From`. Одним из основных факторов, повлиявших на создание языка LINQ, было желание обеспечить поддержку технологии IntelliSense в системе Visual Studio 2010. Возможно, читатели уже задавались вопро-

сом: почему технология IntelliSense не поддерживается в среде SQL Management Studio for SQL Server 2005 при создании запросов? Оказывается, для того чтобы определить, что следует выбрать, пользователь системы SQL Management Studio for SQL Server 2005 должен знать, откуда поступили данные. Изменив порядок выполнения инструкций, создатели языка LINQ наделили его способностью генерировать средства технологии IntelliSense одновременно с началом набора текста.

Подсказка, приведенная на рис. 28.3, свидетельствует о том, что инструкция `From` состоит из двух частей: `<element>` и `<collection>`. Второй из них является коллекцией, из которой извлекаются данные, а первый, по существу, представляет собой переменную итерации, которую можно использовать для ссылки на запрашиваемый элемент. Затем эту пару можно повторно применить к каждой коллекции, являющейся источником данных.

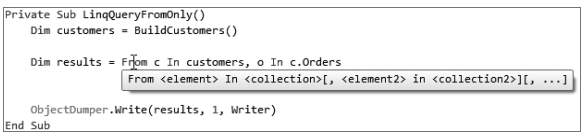


Рис. 28.3

В данном случае мы видим, что пользователь обращается к коллекции клиентов, используя итерационную переменную `c`, и коллекции заказов `c.Orders`, используя итерационную переменную `o`. Между двумя этими коллекциями существует неявное объединение благодаря отношению между клиентом и его заказами. Как можно себе представить, результатом этого запроса будет векторное произведение элементов каждой из этих коллекций. Это приведет к образованию пар (клиент, заказ) для каждого заказа, сделанного клиентом.

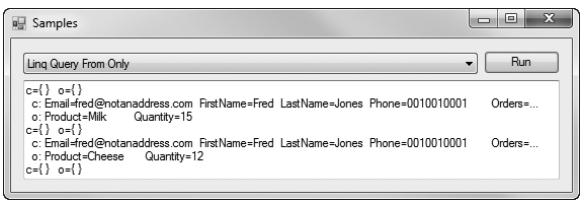


Рис. 28.4

Обратите внимание на то, что мы не использовали инструкцию `Select`, поскольку собираемся просто вернуть все элементы, но как будет выглядеть запись результатов? Если посмотреть на подсказку к результатам, то можно увидеть, что существует перечисление `IEnumerable` анонимного типа. Анонимные типы интенсивно используются в языке LINQ, поэтому для каждого результата не требуется создавать отдельный класс. Если вспомнить, как выглядел первоначальный код, то мы увидим класс `SearchResult`, предназначенный для хранения каждого результата. Анонимные типы означают, что для хранения результатов больше не нужно создавать класс. На этапе компиляции типы, содержащие соответствующие свойства, создаются динамически, предоставляя пользователю строго типизированное множество результатов с поддержкой технологии IntelliSense. Хотя подсказка к результатам может сообщить лишь о том, что результаты представлены в виде перечисления `IEnumerable` анонимного типа, когда пользователь начнет использовать результаты коллекции, он увидит, что этот тип имеет два свойства, `c` и `o`, имеющие типы `Customer` и `Order` соответственно. Результаты работы этого кода, представляющие собой пары (клиент, заказ), показаны на рис. 28.4.



На самом деле синтаксис языка C# требует, чтобы каждое выражение запроса LINQ содержало инструкцию `Select`, даже если пользователь возвращает все объекты в предложении `From`.

Инструкция `Select`

В предыдущем сниппете кода множеством результатов была коллекция пар (клиент, заказ), в котором должны содержаться имя клиента и информация о заказе. Для этого используется инструкция `Select`, аналогичная инструкции языка SQL.



C#

```
private void LinqQueryWithSelect(){
    var customers = BuildCustomers();
    var results = from c in customers
                  from o in c.Orders
                  select new{c.FirstName,
                             c.LastName,o.Product,o.Quantity};
    ObjectDumper.Write(results, Writer);
}
```

Сниппет кода MainForm.cs



VB

```
Private Sub LinqQueryWithSelect()
    Dim customers = BuildCustomers()

    Dim results = From c In customers, o In c.Orders
                  Select c.FirstName, c.LastName, o.Product, o.Quantity

    ObjectDumper.Write(results, Writer)
End Sub
```

Сниппет кода MainForm.vb

Теперь выполнение этого кода приведет к созданию множества результатов, представляющего собой коллекцию объектов, имеющих свойства `FirstName`, `LastName`, `Product` и `Quantity`. Этот факт иллюстрируется выводом, продемонстрированным на рис. 28.5.

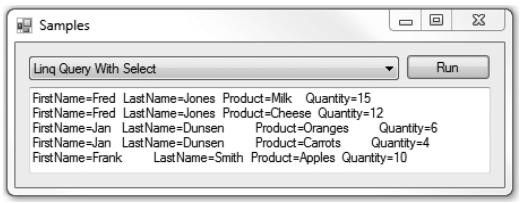


Рис. 28.5

Инструкция `Where`

До сих пор мы видели, как легко можно преобразовать иерархию (клиент, заказ) в множество результатов, имеющих соответствующие свойства. Однако мы еще не использовали фильтр, чтобы найти клиентов, имена которых состоят их пяти и более букв и которые заказали товар `Milk`. Следующий сниппет знакомит нас с инструкцией `Where`, которая ограничивает коллекцию по указанным осям.



C#

```
private void LinqQueryWithWhere(){
    var customers = BuildCustomers();

    var results = from c in customers
        from o in c.Orders
        where c.FirstName.Length> = 5 & &
              o.Product == "Milk"
        select new { c.FirstName, c.LastName, o.Product, o.Quantity };
```

```
    ObjectDumper.Write(results, Writer);
}
```

Сниппет кода MainForm.cs



VB

```
Private Sub LinqQueryWithWhere()
    Dim customers = BuildCustomers()

    Dim results = From c In customers, o In c.Orders
                  Where c.FirstName.Length> = 5 And
                        o.Product = "Milk"
                  Select c.FirstName, c.LastName, o.Product, o.Quantity

    ObjectDumper.Write(results, Writer)
End Sub
```

Сниппет кода MainForm.vb

Результат работы этого запроса похож на предыдущий тем, что он тоже представляет собой множество результатов анонимного типа с четырьмя свойствами: `FirstName`, `LastName`, `Product` и `Quantity`.

Инструкция `Group By`

Первоначальный запрос возвращал список всех заказов товара `Milk` для всех клиентов. Для клиента, который разместил два заказа на товар `Milk`, мы получим две записи в этом множестве результатов. На самом деле мы хотели бы сгруппировать эти заказы по клиентам и вычислить средний объем заказа. Не удивительно, что эту работу выполняет инструкция `Group By`, продемонстрированная в следующем сниппете кода.



C#

```
private void LinqQueryWithGroupingAndWhere(){
    var customers = BuildCustomers();

    var results = from c in customers
                  from o in c.Orders
                  where c.FirstName.Length> = 5 & &
                        o.Product == "Milk"
                  group o by c into avg
                  select new { avg.Key.FirstName, avg.Key.LastName,
                               avg = avg.Average(o => o.Quantity) };
    ObjectDumper.Write(results, Writer);
}
```

Сниппет кода MainForm.cs



VB

```
Private Sub LinqQueryWithGroupingAndWhere()
     Dim customers = BuildCustomers()

     Dim results = From c In customers, o In c.Orders _
                   Where c.FirstName.Length> = 5 And _
                         o.Product = "Milk" _
                   Group By c Into avg = Average(o.Quantity) _
                   Select c.FirstName, c.LastName, avg

    ObjectDumper.Write(results)
End Sub
```

Сниппет кода MainForm.vb

Инструкция `Group By` имеет немного странный синтаксис. По существу, она означает "сгруппировать данные по оси X и поместить результат в другую переменную, которая может использоваться где-то в другом месте". В данном случае "другой переменной" является переменная `avg`, которая будет содержать интересующее нас среднее значение. Поскольку мы выполняем группировку с помощью итерационной переменной `c`, ее можно по-прежнему использовать в инструкции `Select` наряду с альтернативной переменной инструкции `Group By`. Обратите внимание на то, что этот пример на языке C# немного отличается тем, что группировка выполняется с помощью переменной `c`, но теперь у пользователя есть доступ к ее свойству `Key` через альтернативную переменную. Выполнив этот код, получим результаты, представленные на рис. 28.6, которые намного лучше соответствуют нашему первоначальному запросу.

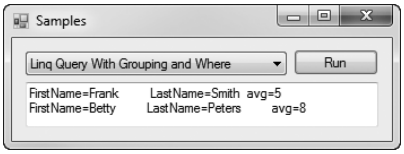


Рис. 28.6

Проекции клиентов

Нам все еще необходимо улучшить результаты вывода, чтобы вывести на печать правильно отформатированное имя клиента и соответствующим образом названное свойство среднего значения, а не имена `FirstName`, `LastName` и `avg`. Для этого можно уточнить свойства, содержащиеся в анонимном типе, созданном как часть инструкции проектирования `Select`. На рис. 28.7 показано, как можно создать анонимные типы с именованными свойствами.

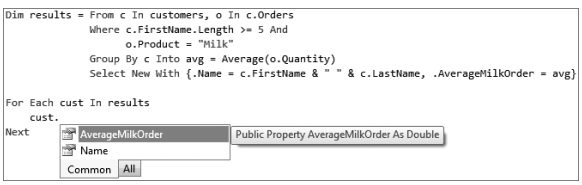


Рис. 28.7

Этот рисунок также иллюстрирует то, что свойство `AverageMilkOrder` должно иметь тип `Double`, поскольку именно это ожидается от функции `Average`. Данное поведение является строго типизированным и может помочь пользователю при создании и использовании полноценных инструкций LINQ.

Инструкция `Order By`

Последнее, что мы должны сделать с помощью инструкции LINQ, — упорядочить результаты. Это можно сделать, упорядочив клиентов по их свойству `FirstName`, как показано в следующем сниппете.



C#

```
private void LinqQueryWithGroupingAndWhere(){
    var customers = BuildCustomers();

    var results = from c in customers
                  from o in c.Orders
                  orderby c.FirstName
                  where c.FirstName.Length> = 5 & &
                        o.Product == "Milk"
                  group o by c into avg
                  select new { Name = avg.Key.FirstName + " " +
                                      avg.Key.LastName,
```

```
                        AverageMilkOrder = avg.Average(o => o.Quantity) };
    ObjectDumper.Write(results, Writer);
}
```

Сниппет кода MainForm.cs



VB

```
Private Sub FinalLinqQuery()
    Dim customers = BuildCustomers()

    Dim results = From c In customers, o In c.Orders
                  Order By c.FirstName
                  Where c.FirstName.Length> = 5 And
                        o.Product = "Milk
                  Group By c Into avg = Average(o.Quantity)
                  Select New With {.Name = c.FirstName & " " & c.LastName,
                                   .AverageMilkOrder = avg}
    ObjectDumper.Write(results)
End Sub
```

Сниппет кода MainForm.vb

Порядок вывода результатов запроса можно легко изменить на противоположный. В данном случае для этого достаточно использовать ключевое слово `Descending` (по умолчанию используется ключевое слово `Ascending`) в конце инструкции `Order By` или применить преобразование `Reverse` ко всему множеству результатов:

```
Order By c.FirstName Descending
```

или

```
ObjectDumper.Write(results.Reverse)
```

Как видим, окончательный запрос является намного более наглядным, чем первоначальный. Легко видеть также, что имена клиентов и средние объемы их заказов выводятся в упорядоченном виде. Кроме того, мы применили фильтрацию, основываясь на длине имени клиента и объемах заказа товара `Milk`, и результаты упорядочены по именам клиентов. Помимо прочего, пользователю не обязательно создавать новые дополнительные классы для выполнения этого запроса.

Отладка и выполнение

Следует знать, что в языке LINQ запросы не выполняются, пока они не используются. По существу, каждый раз, когда используется запрос LINQ, он будет выполнен повторно. Это может создать некоторые проблемы при отладке и неожиданно снизить производительность работы программы, если запросы выполняются многократно. В рассмотренном нами коде мы объявили инструкцию LINQ, а затем передали объект результатов объекту `ObjectDumper`, который в свою очередь выполняет проход по результатам запроса. Если повторить вызов объекта `ObjectDumper`, то он снова выполнит проход по всем результатам.

К сожалению, это замедление программы усложняет отладку инструкций LINQ. Если выбрать инструкцию LINQ и установить на нее точку прерывания, то приложение остановится в этой точке прерывания. Если переместиться на следующую инструкцию, то объект результатов просто сообщит, что он находится в состоянии "In-Memory Query". В языке C# отладка немного проще, потому что на самом деле точка прерывания устанавливается в инструкции LINQ. Как следует из рис. 28.5, точка прерывания была установлена на условной инструкции. По стеку вызовов видно, что те-

кущая точка выполнения уже не находится в методе `FinalQuery`; фактически она находится в методе `ObjectDumper.Write`.

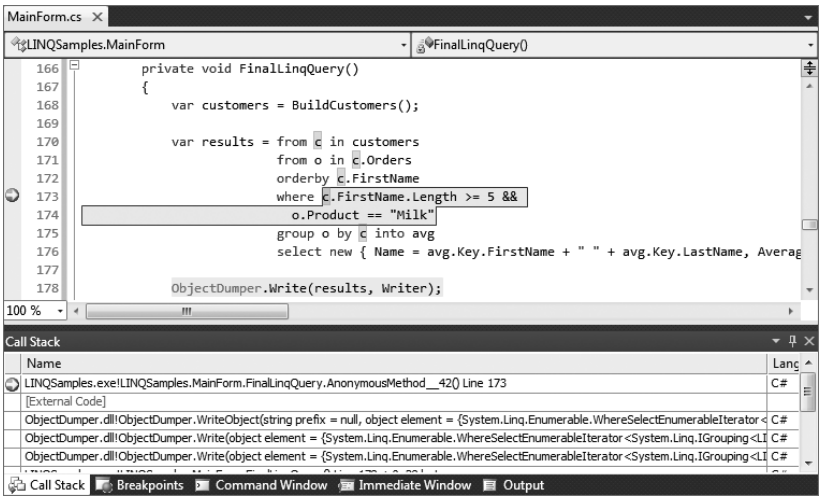


Рис. 28.8

Если необходимо принудительно выполнить инструкцию LINQ, пользователь может применить метод `ToArray` или `ToList` к объекту результатов. В этом случае запрос будет выполнен принудительно и вернет массив или список соответствующего типа.

Этот массив можно использовать в других запросах, уменьшив необходимость многократно выполнять инструкцию LINQ.



Устанавливая точку прерывания в инструкции LINQ в программе на языке C#, необходимо поместить курсор в точку, где должна находиться точка прерывания, и нажать клавишу <F9> (или использовать контекстное меню, которое открывается после щелчка правой кнопкой мыши), а не на поле. Щелчок на поле установит точку прерывания на всю инструкцию LINQ, а это совсем не то, чего мы хотели.

Технология LINQ to XML

Если читатели уже работали с языком XML на платформе .NET, то они помнят, что с объектной моделью довольно неудобно работать. Например, для того чтобы создать даже отдельный XML-элемент, необходимо иметь объект `XmlDocument` .

```
Dim x as New XmlDocument
x.AppendChild(x.CreateElement("Customer"))
```

Как будет показано далее, при использовании языка LINQ для запросов и создания XML-документов эта объектная модель не позволяет создавать элементы на ходу. По этой причине была создана новая объектная модель XML, находящаяся в сборке `System.Xml.Linq`, представленной на рис. 28.9.

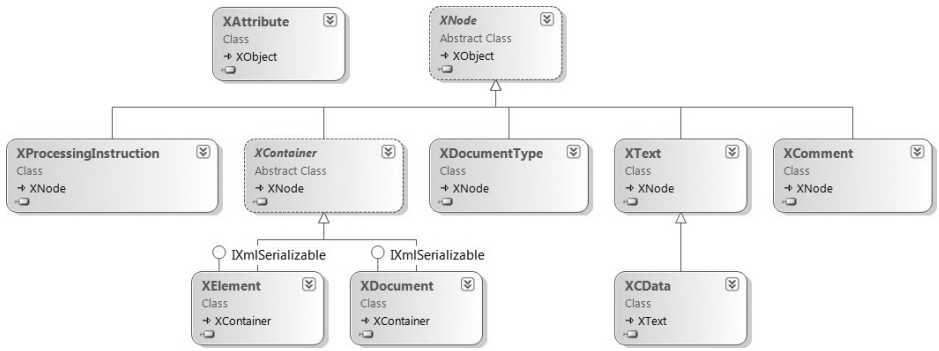


Рис. 28.9

Как следует из рис. 28.9, существуют классы, соответствующие определенным частям XML-документа: `XComment`, `XAttribute` и `XElements`. Крупнейшее усовершенствование заключается в том, что объекты большинства из этих классов могут создаваться с помощью конструкторов, получающих параметры `Name` и `Content`. В следующем коде на языке C# был создан элемент `Customers`, содержащий единственный элемент класса `Customer`. Этот элемент принимает атрибут `Name` и набор элементов класса `Order`.

C#

```
XElement x = new XElement("Customers",
                          new XElement("Customer",
                                       new XAttribute("Name","Bob Jones"),
                          new XElement("Order",
                                       new XAttribute("Product", "Milk"),
                          new XAttribute("Quantity", 2)),
                                       new XElement("Order",
                          new XAttribute("Product", "Bread"),
                                       new XAttribute("Quantity", 10)),
                          new XElement("Order",
                                       new XAttribute("Product", "Apples"),
                          new XAttribute("Quantity", 5))
                          )
);
```

Несмотря на то что этот сниппет кода является довольно многословным и в нем трудно отличить реальные XML-данные от окружающего кода платформы .NET, все же он намного лучше, чем старая объектная модель XML, требовавшая, чтобы элементы создавались индивидуально и лишь затем добавлялись в родительский узел.



Несмотря на то что можно написать точно такой же код на языке VB, используя конструкторы классов `XElement` и `XAttribute`, поддержка XML-литералов (которая будет рассматриваться в следующем разделе) делает это излишним.

Литералы языка VB XML

Одной из наиболее крупных новаций в языке VB является поддержка XML-литералов. Как и строки, и целые числа, XML-литерал считается одним из важнейших элементов кода. Следующий сниппет иллюстрирует точно такой же XML-документ, как и созданный

предыдущим сниппетом кода на языке C#, но на этот раз созданный с помощью XML-литерала и языка VB.

VB

```
Dim cust = <Customers>
               <Customer Name="Bob Jones">
                    <Order Product="Milk" Quantity="2"/>
                    <Order Product="Bread" Quantity="10"/>
                    <Order Product="Apples" Quantity="5"/>
               </Customer>
           </Customers>
```

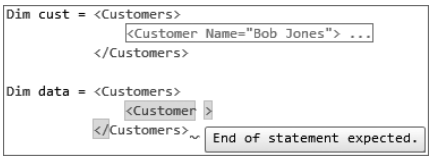


Рис. 28.10

Программист может не только присваивать XML-литерал в коде, но и получить поддержку проектировщика при создании и работе с XML-документом. Например, когда программист вводит символ > в новый элемент, он автоматически создает закрывающий XML-дескриптор. На рис. 28.10 показано, что XML literal `Customers` может быть сжат, как и другие блоки кода в среде Visual Studio 2010.

На рис. 28.8 видно также, что в XML-литерале, который присваивается переменной, есть ошибка. В данном случае пропущен закрывающий дескриптор элемента `Customer`. Поддержка проектировщика является незаменимой при проверке XML-литералов и предотвращает ошибки на этапе выполнения программ, когда элементы XML разбираются в объектах `XElement`.

Вставка элемента XML как объекта класса XElement

К сожалению, в языке C# нет естественной поддержки XML-литералов, что превращает генерацию документов XML в неприятный процесс, даже с учетом новой объектной модели. К счастью, существует полезная надстройка, которая вставляет сниппет XML из буфера в окно редактирования кода в виде ряда объектов класса `XElement`. Это особенно заметно, когда документ XML создается с нуля. Надстройка PasteXmlAsLinq находится в папке `LinqSamples` и доступна в примерах кода на языке C#, поставляемых вместе с системой Visual Studio 2010. Достаточно просто открыть среду Visual Studio 2010, построить решение, перейти к папке вывода и скопировать файлы вывода (а именно `PasteXmlAsLinq.Addin` и `PasteXmlAsLinq.dll`) в папку с надстройками для системы Visual Studio 2010. После повторного запуска Visual Studio 2010 пользователь увидит новую команду Paste XML as XElement в меню Edit окна редактирования кода (рис. 28.11).

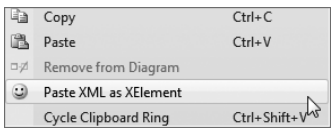


Рис. 28.11



Система Visual Studio 2010 ищет надстройки в разных местах, определенных в диалоговом окне Options (меню Tools). Как правило, она ищет их в папке надстроек, расположенной под корневым каталогом документов Visual Studio, например `C:\users\username\Documents\Visual Studio 2010\Addins`. Если папки `Addins` не существует, ее необходимо создать.

Для того чтобы работать с этой надстройкой, необходимо лишь создать XML-сниппет в своем любимом редакторе документов XML. На рис. 28.12 использован редактор XML

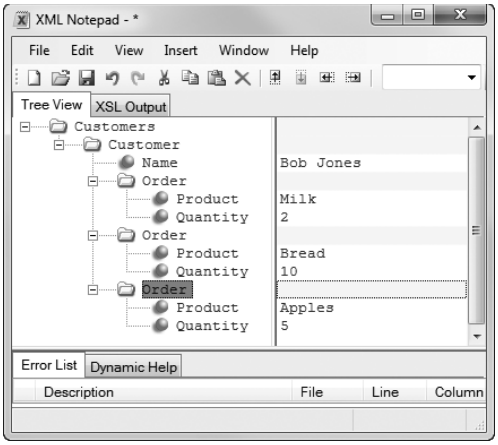


Рис. 28.12

Notepad, свободно распространяемый на веб-сайте `www.microsoft.com`, но пользователь может выбрать встроенный редактор документов XML в среде Visual Studio 2010.

Как только XML-сниппет создан, его следует скопировать в буфер (например, нажав комбинацию клавиш <Ctrl+C>). Затем необходимо установить курсор в точку, в которую требуется вставить сниппет в среде Visual Studio 2010, и выбрать команду Paste XML as XElement из меню Edit. (Разумеется, если эта процедура выполняется часто, то следует присвоить ей какую-то сокращенную комбинацию клавиш, а не использовать меню.) Код, сгенерированный надстройкой, приведен ниже.



C#

```
XElement xml = new XElement("Customers",
                    new XElement("Customer",
                        new XAttribute("Name", "Bob Jones"),
                            new XElement("Order",
                                new XAttribute("Product", "Milk"),
                                new XAttribute("Quantity", "2")
                    ),
                    new XElement("Order",
                        new XAttribute("Product", "Bread"),
                        new XAttribute("Quantity", "10")
                    ),
                    new XElement("Order",
                        new XAttribute("Product", "Apples"),
                        new XAttribute("Quantity", "5")
                    )));
```

Сниппет кода MainForm.cs

Создание документа XML с помощью языка LINQ

Несмотря на то что создание документа XML с помощью новой объектной модели намного проще, чем раньше, реальная мощь новой объектной модели проявляется, когда пользователь сочетает ее с языком LINQ в виде технологии LINQ to XML (XLINQ). Соединяя полноценные возможности запроса с возможностью создавать сложный XML-код в отдельном документе, пользователь может генерировать целые XML-документы с помощью единственной строки. Рассмотрим тот же самый пример о клиентах и заказах. В этом случае у нас есть массив клиентов, каждый из которых мог сделать любое количество заказов. Мы хотим создать код XML, который перечислял бы клиентов и связанные с ними заказы. Начнем с создания списка клиентов, а затем введем заказы.

Для начала создадим XML-литерал, определяющий структуру, которую мы хотим создать.

C#

```
XElement customerXml = new XElement("Customers",
                           new XElement("Customer",
                               new XAttribute("Name", "Bob Jones")));
```

VB

```
Dim customerXml = <Customers>
                        <Customer Name="Bob Jones">
                        </Customer>
                  </Customers>
```

Хотя мы можем упростить этот код, сжав элемент `Customer` в дескриптор `<Customer Name = "Bob Jones"/>`, мы собираемся добавлять заказы как дочерние элементы, поэтому будем использовать отдельный закрывающий XML-элемент.

Пустое выражение

Если у нас есть много клиентов, то элементы класса `Customer` будут повторяться один за другим, причем имя `Bob Jones` будет заменяться другими именами клиентов. Прежде чем приступить к замене имени, сначала необходимо получить повторяющийся элемент класса `Customer`. Для этого создается так называемое *пустое выражение* (expression hole), синтаксис которого знаком каждому, кто работал с технологией ASP.

C#

```
XElement customerXml = new XElement("Customers",
                           from c in customers
                           select new XElement("Customer",
                                      new XAttribute("Name",
                                                     "Bob Jones")));
```

VB

```
Dim customerXml = <Customers>
                           <%= From c In customers _
                               Select <Customer Name="Bob Jones">
                                      </Customer> %>
                           </Customers>
```

Как видим, в коде на языке VB для определения пустого выражения использована конструкция `<%= %>`, в которую будет добавлена инструкция LINQ. Синтаксис языка C# этого не требует, поскольку инструкция LINQ просто становится аргументом конструктора класса `XElement`. Инструкция `Select` создает проекцию на элемент XML для каждого клиента из массива `Customers`, основываясь на статическом значении `"Bob Jones"`. Для того чтобы возвращать имя каждого клиента, нам снова необходимо использовать пустое выражение. На рис. 28.13 показано, как система Visual Studio 2010 обеспечивает полноценную поддержку технологии IntelliSense в этих пустых выражениях.



Рис. 28.13

В следующем сниппете используется переменная цикла `Name`, позволяющая перечислять клиентов по их полным именам. Эта переменная цикла затем используется для установки атрибута `Name` в узле клиента.



C#

```
XElement customerXml = new XElement("Customers",
                           from c in customers
```

```
                                let name = c.FirstName + " " + c.LastName
                                orderby name
                                select new XElement("Customer",
                                       new XAttribute("Name", name),
                                           from o in c.Orders
                                           select new XElement("Order",
                                                  new XAttribute("Product",
                                                                 o.Product),
                                                  new XAttribute("Quantity",
                                                                 o.Quantity))));
```

Сниппет кода MainForm.cs



VB

```
Dim customerXml = <Customers>
                      <%= From c In customers _
                          Let Name = c.FirstName & " " & c.LastName _
                          Order By Name _
                          Select <Customer Name=<%= Name %>>
                                     <%= From o In c.Orders _
                                         Select
                                         <Order
                                                Product=<%= o.Product %>
                                                Quantity=<%= o.Quantity %>
                                                       /> %>
                                 </Customer> %>
                  </Customers>
```

Сниппет кода MainForm.vb

Этот сниппет имеет еще одну особенность: мы включили создание элементов класса `Order` для каждого клиента. Несмотря на то что вторая, вложенная инструкция LINQ может оказаться независимой от первой, между ними существует неявное объединение, обеспечиваемое переменной цикла `c`. Следовательно, вторая инструкция LINQ представляет собой итеративный перебор заказов конкретного клиента, в ходе которого создаются элементы класса `Order` с атрибутами `Product` и `Quantity`.

Как видим, эквивалентный код на языке C# немного сложнее читать, но нисколько не сложнее выполнять. В нем нет необходимости в пустых выражениях, поскольку язык C# не поддерживает литералы XML; вместо этого инструкция LINQ просто вкладывается в конструкцию XML. При создании сложного XML-документа это быстро становится утомительным, поэтому язык VB в настоящее время содержит XML-литералы как одну из основных функциональных возможностей языка.

Запросы XML

Кроме возможности легко создавать документы XML, язык LINQ позволяет создавать запросы XML. В этом разделе для иллюстрации возможностей, связанных с запросами XML, используется XML-элемент `Customers`.

```
<Customers>
    <Customer Name="Bob Jones">
        <Order Product="Milk" Quantity="2"/>
        <Order Product="Bread" Quantity="10"/>
        <Order Product="Apples" Quantity="5"/>
    </Customer>
</Customers>
```

Два следующих сниппета кода демонстрируют один и тот же запрос с помощью языков VB и C# соответственно. В обоих случаях переменная `customerXml` (объект `XElement`) запрашивает все объекты класса `Customer`, из которых извлекается атрибут `Name`. Затем атрибут `Name` разделяется пробелами между именами, и создается новый объект класса `Customer`.



C#

```
var results = from cust in customerXml.Elements("Customer")
              let nameBits = cust.Attribute("Name").Value.Split(' ')
              select new Customer() {FirstName = nameBits[0],
                                     LastName=nameBits[1] };
```

Сниппет кода MainForm.cs



VB

```
Dim results = From cust In customerXml. <Customer>
              Let nameBits = cust.@Name.Split(" "c)
              Select New Customer() With {.FirstName = nameBits(0),
                                          .LastName = nameBits(1)}
```

Сниппет кода MainForm.vb

Легко видеть, что поддержка XML в языке VB расширяет возможности создания запросов к элементам, используя метод `.<elementName>`, и к атрибутам, используя метод `.@attributeName`. На рис. 28.14 показана поддержка технологии IntelliSense для переменной `customerXml` на примере трех запросов XM.



Рис. 28.14

В предыдущем запросе мы уже демонстрировали вторую и третью возможности, связанные с извлечением атрибута и информации об элементе. Третья возможность позволяет извлекать все подэлементы заданного элемента. Например, следующий код извлекает все заказы из документа XML, независимо от того, какому элементу `customer` они принадлежат.

```
Dim allOrders = From cust In customerXml. <Order>
                Select New Order With {.Product = cust.@Product,
                                       .Quantity = CInt(cust.@Quantity)}
```

Поддержка схем

Несмотря на то что язык VB поддерживает создание запросов XML, используя элементы и атрибуты, он не позволяет проверять правильность их имен, которые вводит пользователь. Для того чтобы снизить вероятность ввода неправильного имени, программист может импортировать XML-схему, расширяющую поддержку технологии IntelliSense возможностями включать имена элементов и атрибутов. Схема XML импортируется как любое другое пространство имен платформы .NET. Сначала необходимо добавить ссылку на схему XML в свой проект, а затем добавить инструкцию `Imports` в начало файла с исходным кодом.



В отличие от других инструкций импортирования импорт схемы XML невозможно добавить в проектировщик Project Properties Designer, поэтому программист должен самостоятельно добавить эту инструкцию в начало каждого файла с исходным кодом, в котором он хочет обеспечить поддержку технологии IntelliSense.

Если при работе с существующим XML-файлом в распоряжении программиста нет удобной готовой схемы, то ее создание вручную было бы большим излишеством. К счастью, коллектив разработчиков языка VB включил в систему Visual Studio 2010 мастер XML to Schema Inference Wizard. После инсталляции этот мастер позволяет создавать новые XML-схемы на основе XML-сниппетов, исходного файла XML или указателя URL на источник XML-документов. В данном примере мы начнем с XML-сниппета, который приведен ниже.



```
<c:Customers
xmlns:c="http://www.professionalvisualstudio.com/chapter28/customers">
     <c:Customer Name="Bob Jones">
          <c:Order Product="Milk" Quantity="2" />
          <c:Order Product="Cereal" Quantity="10" />
     </c:Customer>
     <c:Customer Name="Alastair Kelly">
          <c:Order Product="Milk" Quantity="9" />
          <c:Order Product="Bread" Quantity="7" />
     </c:Customer>
</c:Customers>
```

Сниппет кода customers.xml

Обратите внимание на то, что в отличие от предыдущих XML-сниппетов он включает пространство имен, — это необходимо, поскольку XML-схема основана на импортировании пространства имен (а не конкретного XSD-файла). Для генерации XML-схемы на основе этого сниппета щелкните правой кнопкой на своем проекте в окне Solution Explorer и выберите команду Add New Item. Если мастер XML to Schema Inference Wizard инсталлирован, то существует дополнительный шаблонный элемент XML To Schema, как показано на рис. 28.15.

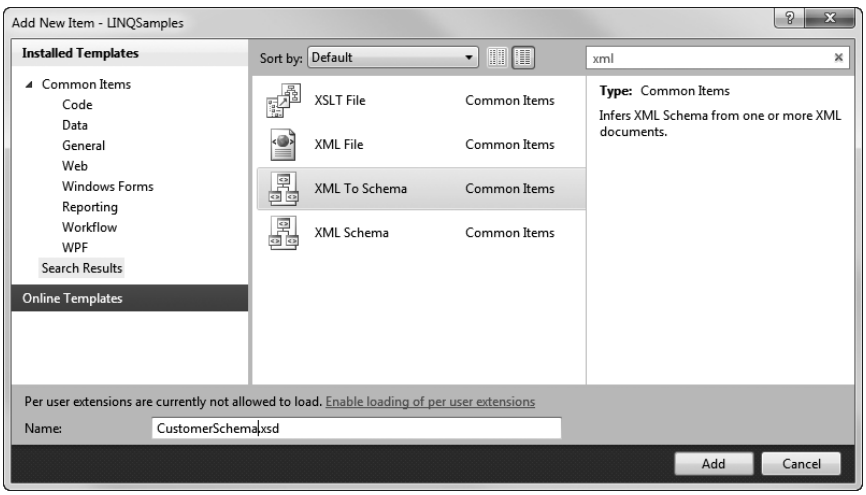


Рис. 28.15

Выбрав этот элемент и щелкнув на кнопке OK, вы получите приглашение выбрать местоположение документа XML, на основе которого будет генерироваться схема. Щелкните на кнопке Type or Paste XML и вставьте XML-сниппет в предложенную область текста. После щелчка на кнопке OK произойдет генерация файла `CustomersSchema.xsd`, содержащего схему, созданную на основе указанных XML-ресурсов. На следующем этапе эта схема импортируется в пользовательский файл исходного кода путем добавления инструкции `Imports` в пространство имен XML, как показано на рис. 28.16.



Рис. 28.16

На рис. 28.16 показано альтернативное имя `c` для пространства имен XML, которое будет использоваться в коде для ссылки на элементы и атрибуты из этого пространства имен. Если вы щелкнете на кнопке `<` или `@`, то увидите, что список IntelliSense теперь содержит соответствующие элементы и атрибуты из импортированной XML-схемы. На рис. 28.17 продемонстрированы новые дополнения, к которым можно обращаться с помощью переменной `customerXml`. Во вложенной инструкции XLINQ (например, в запросе на заказы конкретного клиента) можно увидеть только подмножество элементов схемы (иначе говоря, только элемент `c:Order`).



Рис. 28.17

Технология LINQ to SQL

Читателям может показаться, что мы собираемся описывать еще одну технологию доступа к данным. На самом деле мы рассмотрим расширение существующей модели ADO.NET. Технология LINQ to SQL представляет собой нечто большее, чем простую возможность писать инструкции LINQ для запроса информации из базы данных. Она обеспечивает объект *слоем объектно-реляционного отображения* (relational mapping layer), позволяющим отслеживать изменения в существующих объектах, а также добавлять или удалять объекты, подобно строкам базы данных.

Начнем с обзора некоторых функциональных возможностей технологии LINQ to SQL и связанных с ними проектировщиков. В этом разделе мы будем использовать эталонную базу данных AdventureWorksLT, которую можно загрузить с веб-страницы `http://professionalvisualstudio.com/link/1029A`. Мы планируем завершить разработку запроса, которая была начата в предыдущих разделах, и найти клиентов, имена которых состоят из пяти и более букв, а также вычислить их средний объем заказа. Ранее товаром был `Milk`, но теперь мы рассмотрим компанию по производству велосипедов и будем использовать продукт “HL Touring Seat/Saddle”.



Следует помнить, что при импорте XML-схемы используемые элементы и атрибуты не проверяются. Импорт XML-схемы всего лишь повышает возможности поддержки технологии IntelliSense при создании запроса XLINQ.

Создание объектной модели

Для достижения целей, поставленных в этой главе, будем использовать обычное приложение Visual Basic Windows Forms из диалогового окна New Project. Сначала необходимо создать соединение Data Connection с базой данных AdventureWorksLT (см. Главу 27). На следующем этапе необходимо добавить новый элемент LINQ to SQL Classes с именем `AdventureLite.dbml` в диалоговом окне Add New Item. В результате будут созданы три файла, которые следует добавить в свой проект: `AdventureLite.dbml`, представляющий собой файл отображения; `AdventureLite.dbml.layout`, который подобно проектировщику классов используется для макетирования информации об отображении, чтобы упростить ее обработку, и `AdventureLite.designer.vb`, содержащий классы, в которых данные загружаются как часть технологии LINQ to SQL.



Если не установлен флажок Show All Files, эти элементы могут быть представлены только одним элементом `AdventureLite.dbml`. Выберите проект и щелкните на соответствующей кнопке в инструментальном окне Solution Explorer.

К сожалению, в отличие от других визуальных проектировщиков в системе Visual Studio 2010, имеющих полезных мастеров, облегчающих начало работы, проектировщик LINQ to SQL сразу появляется в виде пустой рабочей области, которая показана на рис. 28.18, *посередине*.

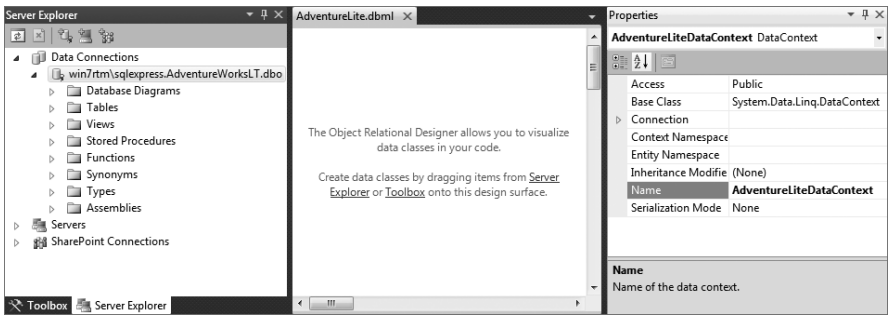


Рис. 28.18

На рис. 28.18, *справа*, показаны свойства, связанные с основной областью проектирования и представляющие собой класс `DataContext`. Если сравнить технологии LINQ и ADO.NET, то инструкция LINQ примерно соответствует команде, в то время как класс `DataContext` — соединению. Это соответствие является лишь приблизительным, поскольку класс `DataContext` фактически является оболочкой соединения с базой данных, что позволяет обслуживать объект на протяжении его жизненного цикла. Например, если выполняется инструкция языка LINQ to SQL, именно методы класса `DataContext` в итоге выполнят запрос к базе данных, создание объектов на основе возвращенных данных и отслеживание изменения и удаления этих объектов.

Если читатели уже работали с проектировщиками классов, то проектировщик LINQ to SQL покажется им очень знакомым. Как свидетельствует инструкция, показанная на рис. 28.18, *посередине*, пользователь может начать создавать отображения данных, перетаскивая элементы из окна Server Explorer (или создавать их вручную,

перетаскивая их из окна Toolbox). В данном случае мы хотим развернуть узел Tables, выбрать таблицы Customer, SalesOrderHeader, SalesOrderDetail и Product и перетащить их на область проектирования. На рис. 28.19 видно, что многие классы и свойства были переименованы, чтобы объектную модель было легче читать при создании инструкций LINQ. Это хороший пример преимущества, которое возникает благодаря отделению объектной модели (например, классов Order или OrderItem) от соответствующих ей данных (в данном случае таблицы SalesOrderHeader и SalesOrderDetail). Поскольку нам не нужны все свойства, которые были созданы автоматически, рекомендуется выбрать их в проектировщике и удалить. Окончательный результат показан на рис. 28.19.

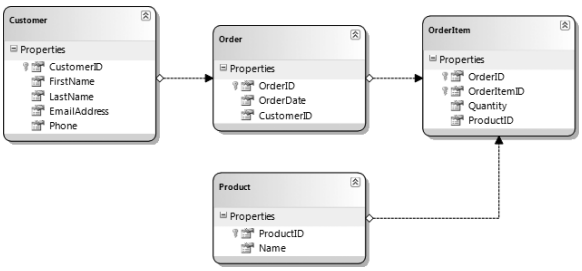


Рис. 28.19

Стоит отметить, что пользователь может модифицировать детали ассоциации между объектами. На рис. 28.20 показано инструментальное окно Properties для ассоциации между класами Product и OrderItem. Здесь мы установили значение свойства Child Property равным False, потому что не собираемся отслеживать связи в обратном направлении от объектов класса Product ко всем объектам класса OrderItems. Кроме того, мы переименовали свойство Parent Property в Product, чтобы сделать ассоциацию более понятной (хотя следует отметить, что в раскрывающемся списке в верхней части окна Properties используются оригинальные имена таблиц сервера SQL Server).

Легко видеть, что мы можем проверить, были ли созданы свойства, которые можно использовать для переходов между экземплярами заданных классов. Несмотря на то что это может выглядеть несколько тривиальным, но если пользователь подумает о том, что может произойти при переходе от объекта класса Order к ассоциированному с ним объекту класса OrderItems, то он сразу поймет, что, если полная иерархия объектов не поместится в памяти, могут возникнуть проблемы. Например, если объекты класса OrderItems еще не загружены в память, технология LINQ to SQL прервет навигацию, обратится к базе данных и извлечет соответствующие данные в упорядоченном виде, чтобы заполнить объекты класса OrderItems.

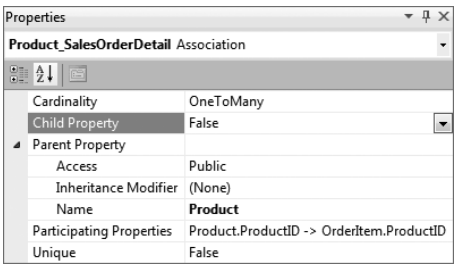


Рис. 28.20

Другое интересное свойство, продемонстрированное на рис. 28.20, называется Participating Properties. В процессе редактирования этого свойства открывается окно Association Editor, в котором можно задать свойства отношения между двумя классами технологии LINQ to SQL. Для того чтобы открыть это диалоговое окно, можно также щелкнуть правой кнопкой мыши на ассоциации в области проектирования и выбрать команду Edit Association. При перетаскивании элементов из окна Server Explorer в область проектирования Association Editor вряд ли понадобится. Однако он особенно полезен, если пользователь создает отображение LINQ to SQL вручную, поскольку при этом он может управлять процессом выравнивания ассоциаций объектов в соответствии с отношениями между данными.

Создание запросов с помощью технологии LINQ to SQL

В предыдущих разделах вы видели достаточно много инструкций языка LINQ, чтобы понять, как с их помощью создать инструкции, которые фильтруют, сортируют, объединяют и проектируют релевантные данные. С учетом этого проверим следующий сниппет кода на языке LINQ to SQL.



C#

```
public void SampleLinqToSql(){
    using (var aw = new AdventureLiteDataContext()){

        var custs = from c in aw.Customers
                    from o in c.Orders
                    from oi in o.OrderItems
                    where c.FirstName.Length> =5 & &
                          oi.Product.Name == "HL Touring Seat/Saddle"
                    group oi by c into avg
                    let name = avg.Key.FirstName + " " + avg.Key.LastName
                    orderby name
                    select new { Name = name,
                                 AverageOrder = avg.Average(oi =>
                                                oi.Quantity) };
        foreach (var c in custs){
            MessageBox.Show(c.Name + " = " + c.AverageOrder);
       }
    }
}
```

Сниппет кода MainForm.cs



VB

```
Using aw As New AdventureLiteDataContext
    Dim custs = From c In aw.Customers, o In c.Orders, oi In o.OrderItems
                Where c.FirstName.Length> = 5 And
                      oi.Product.Name = "HL Touring Seat/Saddle"
                Group By c Into avg = Average(oi.Quantity)
                Let Name = c.FirstName & " " & c.LastName
                Order By Name
                Select New With {Name,.AverageOrder = avg}

    For Each c In custs
        MessageBox.Show(c.Name & " = " & c.AverageOrder)
    Next
End Using
```

Сниппет кода MainForm.vb

Самой важной отличительной чертой этих фрагментов кода является то, что вместо объектов классов `Customer` и `Order`, существующих в памяти до создания и выполнения инструкции LINQ, все данные теперь загружаются в момент выполнения инструкции LINQ. Класс `AdventureLiteDataContext` является средством для открытия соединения с базой данных, формирования и выполнения релевантной инструкции SQL по отношению к базе данных, а также загрузки возвращаемых данных в соответствующие объекты.

Для того чтобы выполнить инструкцию LINQ, необходимо просмотреть таблицы `Customers`, `Orders`, `OrderItems` и `Product`. Очевидно, если бы эту работу выполнял ряд инструкций языка SQL, то процесс стал бы ужасно медленным. К счастью, перевод инструкции LINQ в команды SQL выполняется как отдельная операция.



Это правило имеет одно исключение; например, если в середине инструкции LINQ вызвать метод `ToList`, то она может быть разделена на несколько инструкций SQL. Несмотря на то что язык LINQ to SQL позволяет пользователю абстрагироваться от явных команд SQL, необходимо знать, как будет переводиться запрос и как это может повлиять на производительность приложения.

Для того чтобы увидеть реальный сгенерированный код SQL, можно использовать пример `QueryVisualizer`, поставляемый вместе с системой Visual Studio 2010 (он расположен в папке `LinqSamples` файла `CSharpSamples.zip`, который находится в каталоге `C:\Program Files\Microsoft Visual Studio 10.0\Samples\1033`). Откройте и скомпилируйте этот пример, а затем поместите сгенерированный файл `LinqToSqlQueryVisualizer.dll` в папку визуализатора (как правило, `c:\Users\<username>\Documents\VisualStudio2010\Visualizers`). После повторного запуска системы Visual Studio 2010 этот визуализатор можно будет использовать для того, чтобы просматривать реальные коды SQL, сгенерированные технологией LINQ to SQL для конкретной инструкции LINQ. На рис. 28.21 показана подсказка для инструкции LINQ to SQL в языке C# (в языке VB все точно так же, за исключением того, что пользователь не может увидеть сгенерированный код SQL в первой строке подсказки).

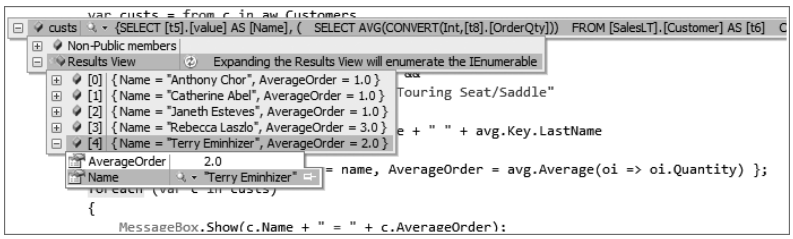


Рис. 28.21

Добавив визуализатор, пользователь увидит в первой строке подсказки пиктограмму с изображением увеличительного стекла. После щелчка на ней открывается окно LINQ to SQL Debug Visualizer, в котором отражается способ преобразования инструкции LINQ to SQL в инструкцию SQL. На рис. 28.22 показан визуализатор, в котором в верхней части экрана изображен процесс анализа запроса компилятором, а в нижней — сгенерированная инструкция SQL. Щелчок на кнопке Execute открывает окно QueryResult (врезка на рис. 28.22), в которое выводятся результаты работы инструкции SQL. Обратите внимание на то, что инструкцию SQL можно модифицировать, пока не будет получен правильный результат. Это позволяет быстро исправлять ошибки в инструкции LINQ.

Вставки, обновления и удаления

В предыдущем сниппете кода класс `DataContext` действовал как инструмент, посредством которого выполнялись запросы LINQ to SQL. Для того чтобы лучше понять, как работает класс `DataContext`, попробуем вставить новую категорию товаров в базу данных `AdventureWorksLT`. Сначала необходимо добавить таблицу `ProductCategory` в область проектирования LINQ to SQL. В данном случае не обязательно модифицировать ее свойства, поэтому перетащите таблицу `ProductCategory` на область проектирования. Затем добавьте в свою базу данных новую категорию. Для этого необходим следующий код.

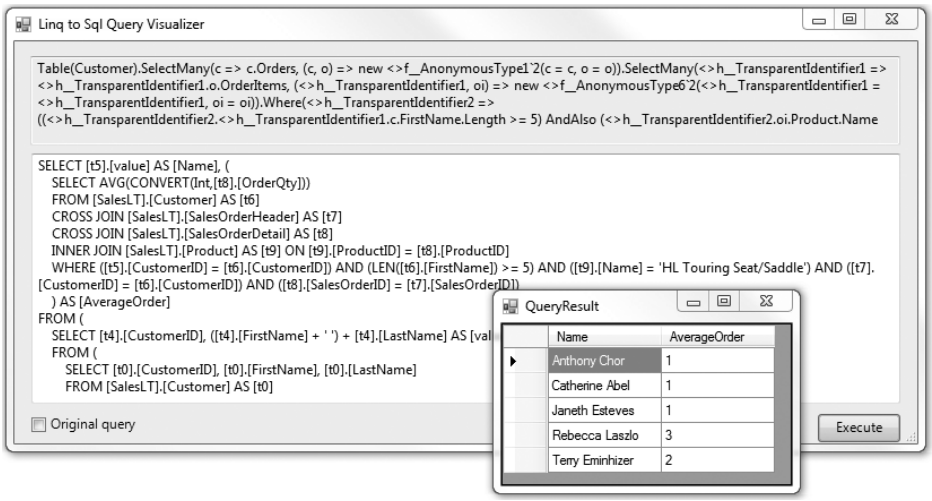


Рис. 28.22

C#

```
using(var aw = new AdventureLiteDataContext()){
    var cat = new ProductCategory();
    cat.Name = "Extreme Bike";
    aw.ProductCategories.InsertOnSubmit(cat);
    aw.SubmitChanges();
}
```

Сниппет кода MainForm.cs

VB

```
Using aw As New AdventureLiteDataContext
    Dim cat As New ProductCategory
    cat.Name = "Extreme Bike"
    aw.ProductCategories.InsertOnSubmit(cat)
    aw.SubmitChanges()
End Using
```

Сниппет кода MainForm.vb

Этот код вставляет новую категорию в коллекцию товаров, которая размещена в памяти классом `DataContext`. Когда вы вызовете метод `SubmitChanges` класса `DataContext`, он обнаружит, что вы уже добавили новую категорию товаров, и вставит соответствующие записи. Аналогичный процесс используется при внесении изменений в существующие записи. В следующем примере мы извлекаем категорию товара, только вставленную в таблицу с помощью синтаксиса `Contains`. Поскольку в таблице есть только одно совпадение, для работы с одной категорией можно использовать метод `FirstOrDefault`.

C#

```
using (var aw = new AdventureLiteDataContext()){
    var cat = (from pc in aw.ProductCategories
               where pc.Name.Contains("Extreme")
               select pc).FirstOrDefault();
    cat.Name = "Extreme Offroad Bike";
    aw.SubmitChanges();
}
```

Сниппет кода MainForm.cs



VB

```
Using aw As New AdventureLiteDataContext
    Dim cat = (From pc In aw.ProductCategories
               Where pc.Name.Contains("Extreme")).FirstOrDefault
        cat.Name = "Extreme Offroad Bike"
        aw.SubmitChanges()
End Using
```

Сниппет кода MainForm.cs

Для того чтобы обновить базу данных после изменения названия категории, необходимо лишь вызвать метод `SubmitChanges` класса `DataContext`. Не вдаваясь в излишние подробности, скажем, что класс `DataContext` изменяет каждое свойство объекта LINQ to SQL и поэтому знает, какой объект следует обновить при вызове метода `SubmitChanges`. Если необходимо удалить объект, достаточно получить экземпляр объекта LINQ to SQL точно так же, как это делалось при обновлении базы, а затем применить метод `DeleteOnSubmit` к соответствующей коллекции. Например, для того чтобы удалить категорию товаров, необходимо вызвать методы `aw.ProductCategories.DeleteOnSubmit(categoryToDelete)` и `aw.SubmitChanges`.

Хранимые процедуры

Когда речь заходит о языке LINQ to SQL, часто спрашивают, можно использовать свои собственные хранимые процедуры вместо инструкций SQL, генерируемых во время выполнения приложения. К счастью, для вставок, обновлений и удалений можно легко определить хранимую процедуру, которую следует использовать. Хранимые процедуры можно использовать для создания экземпляров классов LINQ to SQL. Начнем с добавления простой хранимой процедуры в базу `AdventureWorksLT`. Щелкните правой кнопкой мыши на узле Stored Procedures под соединением базы данных в инструментальном окне Server Explorer и выберите команду Add New Stored Procedure, чтобы откроть окно редактирования кода с шаблоном новой хранимой процедуры. В следующем коде мы выбрали пять полей, относящихся к объекту класса `Customer`.

```
CREATE PROCEDURE dbo.GetCustomers
AS
BEGIN
    SET NOCOUNT ON
    SELECT c.CustomerID, c.FirstName, c.LastName, c.EmailAddress, c.Phone
    FROM SalesLT.Customer AS c
END;
```

После того как сохраните эту хранимую процедуру, она появится под узлом Stored Procedures. Если теперь открыть проектировщик `AdventureLite LINQ to SQL`, то можно будет перетащить эту хранимую процедуру на правую панель области проектирования. На рис. 28.23 видно, что типом значения, возвращаемого методом `GetCustomers`, является `Auto-generated Type`. Это значит, что мы можем лишь запрашивать информацию у возвращаемого объекта. В идеале мы хотели бы иметь возможность вносить изменения в эти объекты и использовать класс `DataContext`, чтобы сохранить эти изменения в базе данных.

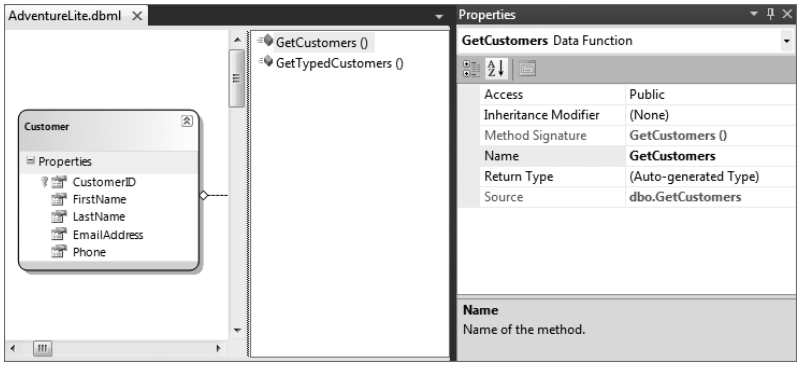


Рис. 28.23



Мы не обязаны согласовывать свойства со столбцами хранимых процедур, поскольку это отображение автоматически обрабатывается классом `DataContext`. Это обоюдоострое решение: очевидно, оно работает, когда имена столбцов отображаются в имена источников в классе языка LINQ to SQL, но может вызвать ошибку во время выполнения приложения, если обнаружатся пропущенные или несоответствующие столбцы.

Второй метод, `GetTypedCustomers`, возвращает объект класса `Customer`. Для того чтобы создать этот метод, можно либо перетащить на правую панель хранимую процедуру `GetCustomers`, а затем установить атрибут `Return Type` равным `Customer`, либо перетащить хранимую процедуру `GetCustomers` в класс `Customer` на левой панели области проектирования. Второй способ также приведет к созданию метода на правой панели, но тип возвращаемого значения `Customer` будет задан автоматически.

Определив эти хранимые процедуры как методы в области проектирования, следует присвоить им имена, совпадающие с именами соответствующих методов класса `DataContext`.



C#

```
using (var aw = new AdventureLiteDataContext()){
    var customers = aw.GetCustomers();
    foreach (var c in customers){
        MessageBox.Show(c.FirstName);
    }
}
```

Сниппет кода MainForm.cs



VB

```
Using aw As New AdventureLiteDataContext
    Dim customers = aw.GetCustomers

    For Each c In customers
        MsgBox(c.FirstName)
    Next
End Using
```

Сниппет кода MainForm.vb

Покажем, как можно использовать хранимые процедуры для создания экземпляров классов LINQ to SQL. Если вместо этого пользователь желает обновить, вставить или удалить объекты с помощью хранимых процедур, достаточно повторить почти такой же процесс, за исключением того, что потребуется определить соответствующее поведение класса LINQ to SQL. Начнем с создания хранимой процедуры для вставки новой категории товаров.

```
CREATE PROCEDURE dbo.InsertProductCategory
    (
    @categoryName nvarchar(50),
    @categoryId int OUTPUT
    )
AS
BEGIN
    INSERT INTO SalesLT.ProductCategory (Name) VALUES (@categoryName)
    SELECT @categoryId=@@identity
END;
```

Повторяя процесс, описанный выше, вы должны перетащить вновь созданную хранимую процедуру из окна Server Explorer на правую панель области проектирования LINQ to SQL. Затем в инструментальном окне Properties следует модифицировать свойство `Insert` класса `ProductCategory`. В открывшемся диалоговом окне, показанном на рис. 28.24, можно выбрать, использовать ли код, генерируемый на этапе выполнения приложения, или настроить используемый метод. На рис. 28.24 был выбран метод `InsertProductCategory`. Сначала свойства `Class Properties` остаются неопределенными, поскольку система Visual Studio 2010 не способна угадывать, какие свойства должны отображаться в аргументы метода. Достаточно просто согласовать их со свойствами `id` и `name`. Теперь, когда класс `DataContext` приступит к вставке категории `ProductCategory`, он будет использовать хранимую процедуру, а не инструкцию SQL, генерируемую на этапе выполнения.

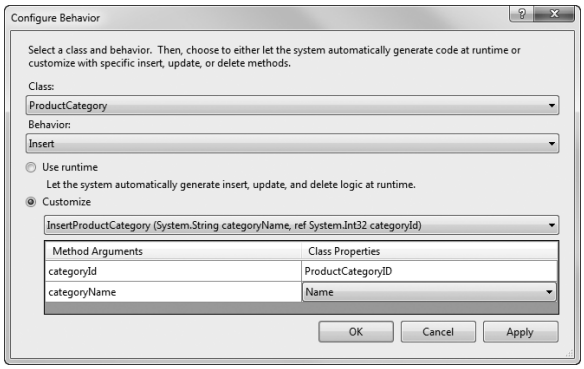


Рис. 28.24

Связывание объектов LINQ to SQL

При использовании привязок вместе с объектами LINQ to SQL важно помнить, что они, по существу, являются обычными объектами платформы .NET. Это значит, что новый объект источника данных можно создать с помощью инструментального окна Data Sources. В рассмотренных выше примерах мы вынуждены были использовать мастер Add New Data Source Wizard, выбирая объект класса `Customer`. Поскольку объекты `Order` и `OrderItem` доступны через навигационные свойства классов `Orders` и `OrderItems`, теперь их не обязательно явно добавлять в окно Data Source.

Создав объект источника данных (рис. 28.25, *слева*), можно перетащить узлы на форму, чтобы создать соответствующие компоненты данных. Перейдите к узлу Customer, откройте список, чтобы указать, что вы хотите создать компонент DataGrid-

View, и перетащите его на свою форму. Затем укажите, что вы хотите создать компонент Orders (дочерний узел узла Customer), и также перетащите его на форму. Заметим, что в вашем распоряжении нет навигатора привязок для данного источника привязок, поэтому перетащите на форму компонент BindingNavigator из окна Toolbox и установите его свойство `BindingSource` равным `OrdersBindingSource`. Это свойство будет создано, когда вы будете перетаскивать компонент над узлом Orders. И наконец, поскольку мы хотим отобразить на экране все компоненты OrderItems в сетке DataGridView, задайте их в раскрывающемся списке и перетащите этот узел на форму. После выполнения этих операций вы должны получить нечто подобное показанному на рис. 28.25. Обратите внимание на то, что мы также включили кнопку, которая будет использоваться для загрузки данных, и удобно разместили информацию о компоненте Order на панели.

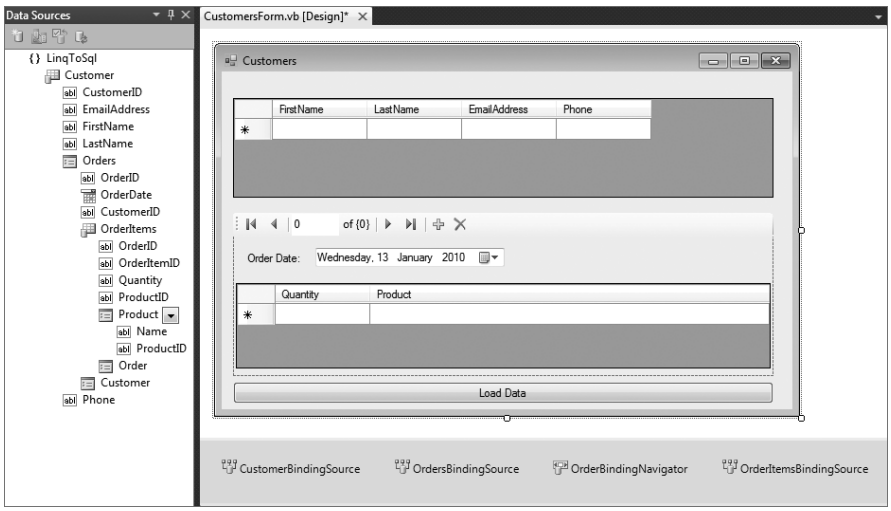


Рис. 28.25

Читатели уже заметили, что столбцы сетки данных OrderItems не соответствуют столбцам, изображенным на рис. 28.25. По умолчанию должны появиться столбцы `Quantity`, `Order` и `Product`. Очевидно, что два последних столбца не представляют для нас интереса, но у нас нет простого способа отобразить на экране название товара в заказе с помощью текущих объектов LINQ to SQL. К счастью, существует простой способ эффективно скрыть переход от компонента OrderItem к столбцу `Product`, чтобы название товара появилось как свойство класса `OrderItem`.

Для этого необходимо добавить в класс `OrderItem` свое собственное свойство. Каждый класс языка LINQ to SQL генерируется как частичный класс, т.е. расширение этого класса осуществляется простым щелчком правой кнопкой мыши на проектировщике LINQ to SQL и выбором команды View Code. В результате генерируется файл исходного кода, в данном случае `AdventureLite.vb` (или `AdventureLite.cs`), и включается определение частичного класса. Затем пользователь может добавить свой собственный код. В следующем сниппете мы добавили свойство `Product`, упрощающее доступ к названию заказанного товара .



C#

```
partial class OrderItem{
    public string ProductName{
        get{
             return this.Product.Name;
           }
    }
}
```

Сниппет кода AdventureLite.cs



VB

```
Partial Class OrderItem
    Public ReadOnly Property ProductName() As String
        Get
            Return Me.Product.Name
        End Get
    End Property
End Class
```

Сниппет кода AdventureLite.vb

По некоторым причинам, возможно, потому, что оно добавлено во второй файл исходного кода, это свойство не распознается инструментальным окном Data Sources. Однако мы можем по-прежнему привязать столбец `Product` к этому свойству, задав вручную поле `DataPropertyName` в диалоговом окне Edit Columns для сетки данных.

В заключение необходимо загрузить реальные данные, когда пользователь щелкнет на кнопке. Эту задачу решает следующий код.

C#

```
private void btnLoadData_Click(object sender, EventArgs e){
    using (var aw = new AdventureLiteDataContext()){
        var cust = aw.Customers;
        this.customerBindingSource.DataSource = cust;
    }
}
```

VB

```
Private Sub btnLoad_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) Handles btnLoad.Click
    Using aw As New AdventureLiteDataContext
         Dim custs = From c In aw.Customers
         Me.CustomerBindingSource.DataSource = custs
    End Using
End Sub
```

Если приложение будет запущено, то, когда пользователь щелкнет на кнопке, информация о клиентах заполнит верхнюю сетку данных. Однако, какие бы клиенты ни были выбраны, эта информация не появится в области информации о заказе. Причина заключается в том, что язык LINQ to SQL использует “ленивую” загрузку, т.е. загружает информацию только в тот момент, когда она запрашивается. Используя средство визуализации данных, описанное ранее, можно убедиться, что они содержат только информацию о клиенте.

```
SELECT [t0].[CustomerID], [t0].[FirstName], [t0].[LastName],
[t0].[EmailAddress],
[t0].[Phone]
FROM [SalesLT].[Customer] AS [t0]
```

Существуют два способа устранить эту проблему. Во-первых, можно заставить язык LINQ to SQL вернуть все данные из компонентов `Order`, `OrderItem` и `Product`, задав это требование как часть первоначального запроса. Для этого необходимо модифицировать код, обрабатывающий щелчок на кнопке, следующим образом.

C#

```
private void btnLoadData_Click(object sender, EventArgs e){
    using (var aw = new AdventureLiteDataContext()){
        var loadOptions =new System.Data.Linq.DataLoadOptions();
        loadOptions.LoadWith <Customer> (c=> c.Orders);
        loadOptions.LoadWith <Order> (o=> o.OrderItems);
        loadOptions.LoadWith <OrderItem> (o=> o.Product);
        aw.LoadOptions = loadOptions;

        var cust = aw.Customers;
        this.customerBindingSource.DataSource = cust;
    }
}
```

VB

```
Private Sub btnLoad_Click(ByVal sender As System.Object,
                         ByVal e As System.EventArgs) Handles btnLoad.Click
    Using aw As New AdventureLiteDataContext
        Dim loadOptions As New System.Data.Linq.DataLoadOptions
         loadOptions.LoadWith(Of Customer)(Function(c As Customer) c.Orders)
         loadOptions.LoadWith(Of Order)(Function(o As Order) o.OrderItems)
         loadOptions.LoadWith(Of OrderItem)(Function(oi As OrderItem) _
                                  oi.Product)
         aw.LoadOptions = loadOptions

         Dim custs = From c In aw.Customers
         Me.CustomerBindingSource.DataSource = aw.Customers
    End Using
End Sub
```

По существу, этот код сообщает классу `DataContext`, чтобы он извлек объекты класса `Customer`, а затем обязательно перешел к свойству `Orders`. Аналогично объекты класса `Order` должны обратиться к свойству `OrderItems` и т.д. Следует помнить, что это решение может оказаться неэффективным, если клиентов слишком много. По мере увеличения количества клиентов и заказов производительность приложения будет падать все ниже, так что это не совсем удачное решение; тем не менее оно иллюстрирует, как можно использовать свойство `LoadOptions` класса `DataContext`.

Другая альтернатива предлагает не использовать класс `DataContext`. Необходимо вспомнить, что происходит за "кулисами" класса `DataBinding`. Когда мы выбираем клиента в сетке данных, объект `OrderBindingSource` обновляется. Он пытается обратиться к свойству `Orders` соответствующего клиента. Если бы мы использовали класс `DataContext`, то не смогли бы заполнить это свойство `Orders`. Поэтому намного лучше изменить код следующим образом.



C#

```
private AdventureLiteDataContext aw = new AdventureLiteDataContext();
private void btnLoadData_Click(object sender, EventArgs e){
    var cust = aw.Customers;
    this.customerBindingSource.DataSource = cust;
}
```

Сниппет кода CustomersForm.cs



VB

```
Private aw As New AdventureLiteDataContext()
Private Sub btnLoad_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) Handles btnLoad.Click
    Dim custs = From c In aw.Customers
    Me.CustomerBindingSource.DataSource = custs
End Sub
```

Сниппет кода CustomersForm.vb

Поскольку класс `DataContext` по-прежнему существует, то когда источник привязки перебирает разные свойства, инструкции LINQ to SQL заполняют эти свойства данными. Это решение намного лучше масштабируется и позволяет заполнять целые иерархии клиентов одним щелчком мышью.

Программа LINQPad

Хотя основным мотивом создания технологии LINQ было желание сделать код более наглядным, во многих случаях создание и отладка запросов стали гораздо сложнее. Тот факт, что выражения языка LINQ выполняются только тогда, когда происходит перебор результатов, может привести к путанице и неожиданностям. Один из наиболее полезных инструментов для создания выражений LINQ был создан Джозефом Альбахари (Joseph Albahari) и называется LINQPad (`www.linqpad.net`). На рис. 28.26 показано, как с помощью этого редактора писать выражения.

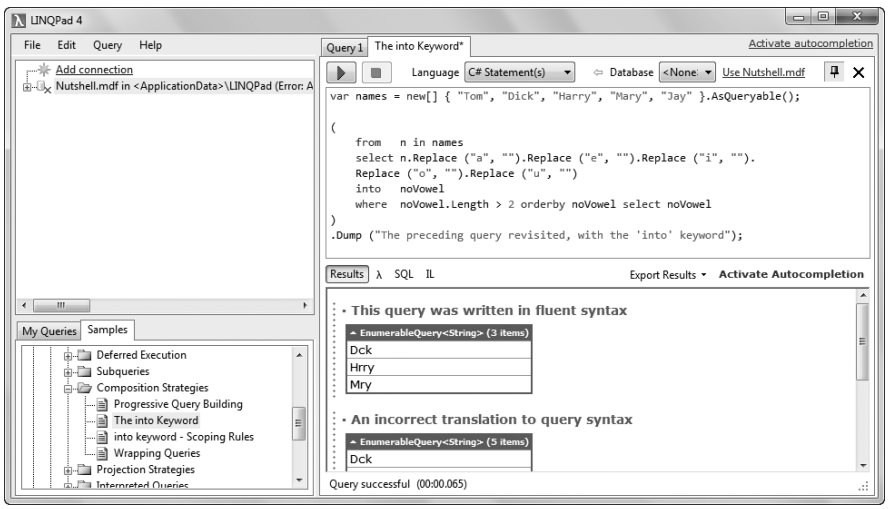


Рис. 28.26

На нижней панели видны результаты выполнения этого выражения. С помощью наладки механизма выражений LINQ можно получить правильный результат, не компилируя и не выполняя все приложение.

Резюме

В этой главе изложено введение в язык интегрированных запросов (Language Integrated Queries — LINQ), представляющий значительный шаг в направлении общей модели программирования для доступа к данным. Было показано, что инструкции языка LINQ помогают повысить наглядность кода, поскольку теперь программист не обязан кодировать все детали данных, которые следует просмотреть, условные инструкции для выбора объектов или создание множества результатов.

Мы также описали новую объектную модель XML, интеграцию языка XML и языка VB, использование языка LINQ для запроса XML-документов, а также поддержку технологии IntelliSense в системе Visual Studio 2010 для работы с языком XML в программах на языке VB.

В заключение были изложены некоторые сведения о технологии LINQ to SQL и ее использовании для объектно-реляционного отображения. Хотя мы ограничились только отображением объекта в отдельную таблицу, это значительно упростило работу с базой данных.

В следующей главе речь пойдет о том, насколько мощной является технология LINQ в сочетании с платформой ADO.NET Entity Framework при управлении жизненным циклом объектов. Обладая намного более сложными средствами отображения, эта технология в будущем резко изменит способ работы с данными.

29

Технология ADO.NET Entity Framework

В ЭТОЙ ГЛАВЕ...

- Что такое Entity Framework
- Создание модели Entity Framework
- Запросы в модели Entity Framework

Одним из основных требований в бизнес-приложениях (и многих других типах приложений) является способность хранить данные в базах данных и извлекать их оттуда. Однако это легче сказать, чем сделать, потому что реляционная схема базы данных не сочетается с иерархией объектов, с которой мы предпочитаем работать в коде. Для создания и заполнения таких иерархий объектов требуется писать большой объем кода для передачи данных от источника приемнику – дружественные объектные модели, которые потом, как правило, достаточно трудно поддерживать. Фактически это был такой мощный источник постоянного разочарования, что многие разработчики обратились к написанию генераторов кода и прочих инструментов, которые автоматически создают код для доступа к базе данных на основе информации о ее структуре. Однако генераторы кода обычно создают взаимно однозначное отображение между структурой базы данных и объектной моделью, что приводит к проблеме объектно-реляционного несоответствия, которая состоит в том, что то, как данные хранятся в базе данных, не обязательно непосредственно связано с тем, как разработчики хотели бы смоделировать данные в виде объектов. Это привело к концепции объектно-реляционного отображения (Object Relational Mapping), где может быть спроектирована идеальная объектная модель для работы с данными в коде, которая затем может быть отображена на схему базы данных. По завершении отображения механизм объектно-реляционного отображения (Object Relational Mapper –

ORM) должен взять на себя бремя по переводу между объектной моделью и базой данных, оставляя разработчикам решения бизнес-проблем (а не технологические вопросы работы с данными).

Для многих разработчиков технология ORM представляется Святым Граалем для работы с данными из базы данных в виде объектов. Нет недостачи в постоянных жарких дебатах о сильных и слабых сторонах различных вариантов технологии ORM и о том, как должен выглядеть идеальный механизм ORM. Мы не будем разбираться в этих аргументах в данной главе, а просто ознакомимся с технологией ADO.NET Entity Framework — ORM от Microsoft.

Если обратиться к истории, то с момента возникновения платформы .NET Framework появился ряд средств для доступа к данным в базах данных, и все — под знаменем технологии ADO.NET. Сначала это был низкоуровневый доступ через механизм SqlConnection (и подключения к другим типам баз данных) с использованием таких средств, как читатели данных. Затем появились более высокоуровневые средства доступа с использованием типизированных наборов данных (Typed DataSets). В версии NET Framework 3.5 появилась технология LINQ to SQL, впервые предоставляя встроенное средство для работы с данными как с объектами.

Однако в течение длительного времени компания Microsoft не включала механизм ORM в платформу .NET Framework (несмотря на ряд ранних неудачных попыток сделать это с архитектурой ObjectSpaces). К тому времени уже имелся ряд механизмов ORM, доступных для использования с платформой .NET Framework, в числе самых популярных — NHibernate и LLBLGen Pro. В конечном итоге компании Microsoft удалось выпустить свой собственный механизм ORM, который она назвала ADO.NET Entity Framework и который вошел в версию .NET Framework 3.5 SP1.

Выпуск технологии Entity Framework, несмотря на долгое ожидание, не был безоблачным, — более того, многими разработчиками (в том числе и сотрудниками Microsoft со статусом MVP (Most Valuable Professional — особо ценный специалист)) была даже подписана петиция протеста. Однако в версии .NET Framework 4 в технологию Entity Framework было добавлено много улучшений, которые существенно уменьшили количество недостатков (и недовольных продуктом).

Эта глава проведет вас через процесс создания модели базы данных в соответствии с технологией Entity Framework и ознакомит с тем, как использовать эту технологию для запросов и обновлений базы данных. Технология Entity Framework — это огромная тема, и имеются целые книги, посвященные использованию данного продукта. Осветить все его особенности невозможно, так что эта глава посвящена обсуждению только лишь некоторых из его основных функций и тому, как начать работу и создать базовую модель.

Модели Entity Framework, создаваемые в этой главе, будут использованы в ряде последующих глав, где в примерах потребуется доступ к базе данных.

Что такое Entity Framework

По существу, технология Entity Framework реализует объектно-реляционное отображение (ORM). Объектно-реляционное отображение позволяет создавать концептуальную модель объекта, отображать его на базу данных, а механизм ORM будет заботиться о переводе ваших запросов из объектной модели в запросы к базе данных, возвращая данные в виде объектов, которые вы определили в своей модели.

Сравнение с технологией LINQ to SQL

Стандартный вопрос от разработчиков — об отношениях между технологиями Entity Framework и LINQ to SQL, и какую технологию следует применять при создании приложений, ориентированных на работу с данными. Рассмотрим преимущества каждой из упомянутых технологий.

Преимущество технологии LINQ to SQL над технологией Entity Framework

- Легкость обучения и создания запросов.

Преимущества технологии Entity Framework над технологией LINQ to SQL

- Позволяет создавать концептуальную модель базы данных, а не просто взаимно однозначное соответствие базы данных объектам (например, когда один объект отображается на несколько таблиц базы данных, поддержка наследования, определение сложных свойств).
- Возможность генерации базы данных на основе модели.
- Поддержка баз данных, отличных от SQL Server.
- Поддержка отношений "многие ко многим".
- Поддержка отложенной загрузки.
- Синхронизация, обеспечивающая обновление баз данных без потерь настроек вашей модели.
- Продолжает развиваться в отличие от LINQ to SQL.

Концепции Entity Framework

Ниже описаны некоторые из важных концепций Entity Framework, а также термины, использующиеся в этой главе.

- **Сущностная модель (Entity Model).** Модель, которую вы создаете с использованием технологии Entity Framework, состоит из трех частей.
  - **Концептуальная модель.** Представляет объектную модель, включая сущности, их свойства и связи между ними.
  - **Модель хранения.** Представляет структуру базы данных, включая таблицы/представления/хранимые процедуры, столбцы, внешние ключи и т.д.
  - **Отображение.** Представляет "клей", соединяющий модель хранения и концептуальную модель (т.е. слой между базой данных и объектной моделью), путем отображения одной на другую.

    Каждая из этих частей поддерживается технологией Entity Framework на основе языка XML с использованием *предметно-ориентированного языка* (domain-specific language — DSL).
  - **Сущность (Entity).** По сути, сущности представляют собой объекты (со свойствами), на которые отображается модель базы данных.
  - **Набор сущностей (Entity Set).** Представляет собой коррекцию данных сущностей. Сущность можно представлять как строку в базе данных, а набор сущностей — как таблицу.
  - **Связь (Association).** Связи определяют взаимоотношения сущностей в вашей сущностной модели и концептуально представляют собой то же, что и взаимоотношения в базе данных.

- **Отображение (Mapping).** Представляет собой базовую концепцию механизма ORM. По сути, это слой перевода из реляционной схемы базы данных в объекты кода.

С чего начать

Для демонстрации некоторых различных возможностей технологии Entity Framework пример в этом разделе использует учебную базу данных `AdventureWorksLT`, разработанную компанией Microsoft как один из образцов баз данных для SQL Server. База `AdventureWorksLT` представляет собой упрощенный вариант полной базы данных `AdventureWorks`, что упрощает демонстрацию концепций технологии Entity Framework без тех дополнительных сложностей, которые создает использование полной базы данных.

База данных `AdventureWorksLT` доступна для загрузки с веб-сайта CodePlex в виде сценария, расположенного по адресу

`http://professionalvisualstudio.com/link/1029A`

Adventure Works Cycles представляет собой вымышленную компанию по продаже цепей для велосипедов, а база данных `AdventureWorksLT` используется для хранения и доступа к данным о продажах.

Следуйте подробным инструкциям с веб-сайта CodePlex о том, как установить базу данных из загруженного сценария в экземпляр SQL Server (версии SQL Server Express Edition вполне достаточно), который находится на вашем компьютере или доступен для него.

Итак, перейдем к созданию проекта, который содержит модель Entity Framework этой базы данных. Начните с открытия диалогового окна New Project и создания нового проекта. Учебный проект, который будет создан в этой главе, использует шаблон проекта WPF. Вы будете выводить данные в управляющем элементе WPF DataGrid, определенном в файле `MainWindow.xaml` под именем `dgEntityFrameworkData`.

Теперь, когда у вас есть проект, который будет размещать и выполнять запросы к модели Entity Framework, пришло время для создания этой модели.

Создание сущностной модели

Есть два способа создания модели. Как правило, модель создается на основе структуры существующей базы данных, однако с помощью технологии Entity Framework можно начать и с пустой модели, а затем заставить механизм Entity Framework генерировать структуру базы данных на ее основе.

Учебный проект использует для создания модели первый метод, основанный на структуре базы данных `AdventureWorksLT`.

Мастер Entity Data Model Wizard

Откройте диалоговое окно Add New Item вашего проекта, перейдите к категории данных и выберите ADO.NET Entity Data Model в качестве шаблона (рис. 29.1). Дайте модели имя `AdventureWorksLTModel.edmx`.

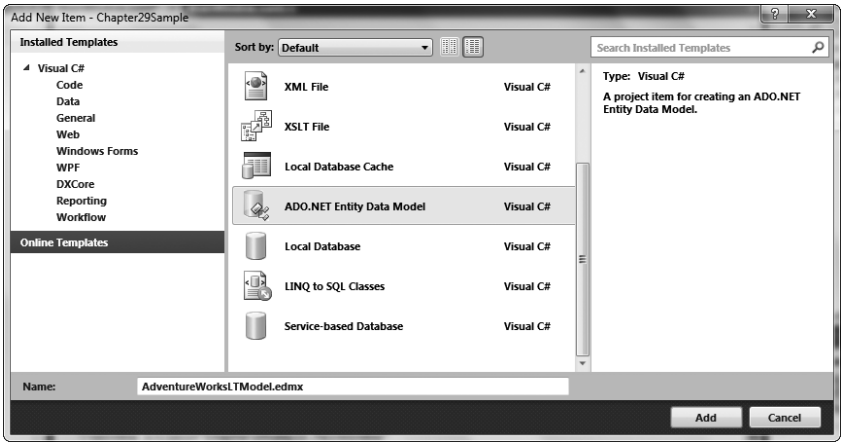


Рис. 29.1

При этом будет запущен мастер Entity Data Model Wizard, который поможет начать построение модели Entity Framework.

Вы увидите диалоговое окно, показанное на рис. 29.2, которое позволяет выбрать, хотите ли вы автоматически создать модель на основе базы данных (Generate from Database) или начать с пустой модели (Empty Model).

Пустая модель полезна, если вы хотите создать модель с нуля и либо отобразить ее на данную базу данных вручную, либо позволить Entity Framework создать базу данных на основе вашей модели.

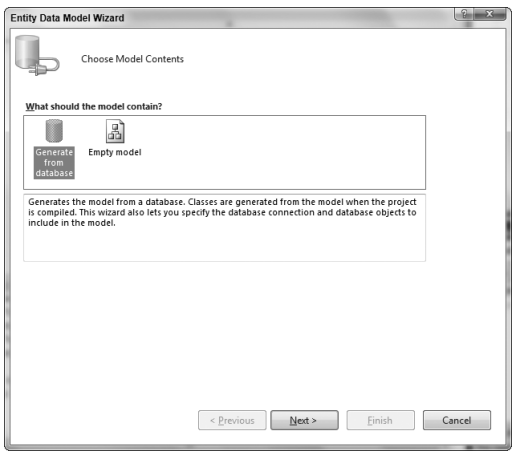


Рис. 29.2

Однако, как указывалось ранее, здесь будет создана модель на основе базы данных `AdventureWorksLT`, так что вам надо выбрать вариант Generate from Database и попросить мастера помочь создать модель на основе существующей базы данных.

На следующем шаге нужно создать соединение с базой данных (рис. 29.3). Можете выбрать из выпадающего списка последнее подключение к базе данных, но если его там нет (например, если это первое подключение к базе данных), создайте новое подключение. Для этого щелкните на кнопке New Connection и пройдите стандартную процедуру выбора экземпляра SQL Server, учетных данных для аутентификации и, наконец, самой базы данных.

Если в качестве данных для аутентификации вы используете имя пользователя и пароль, можно не включать их в строку подключения (которая содержит детали, необходимые для подключения к базе данных) при ее сохранении, потому что эта строка хранится в виде обычного текста, так что любому, кто увидит ее, обеспечен доступ к базе данных. В этом случае, чтобы создать подключение к базе данных, следует передавать указанные данные модели перед выполнением запросов. Если вы не установите флаг сохранения настроек соединения в файле `App.config`, то вам по-

требуется передавать модели не только упомянутые имя и пароль, но и другую информацию о подключении к базе данных.

На следующем шаге мастер использует созданное на предыдущем шаге подключение для соединения с базой данных и извлечения ее структуры (т.е. ее таблиц, представлений и хранимых процедур), которые отображаются в дереве выбора элементов, которые должны быть включены в вашу модель (рис. 29.4).

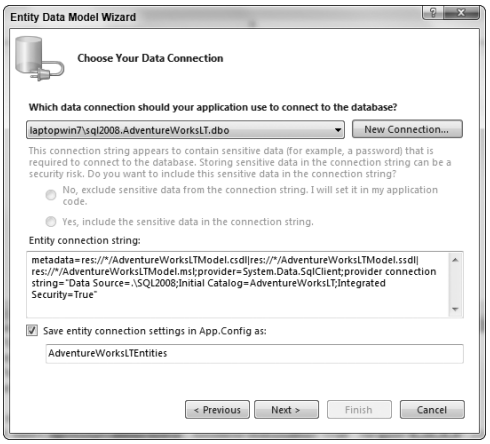


Рис. 29.3

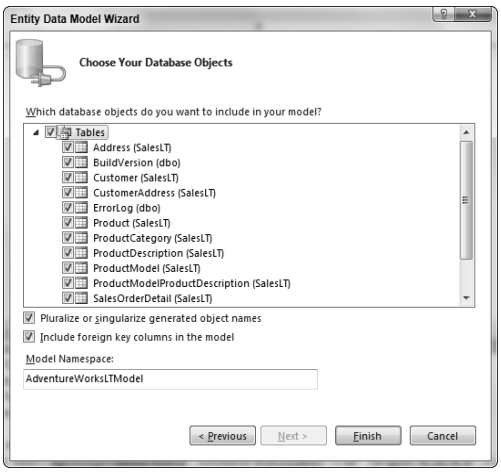


Рис. 29.4

Другие настройки, которые могут быть указаны в этом окне, включают следующее.

- **Pluralize or Singularize Generated Object Names.** Будучи установлена, эта настройка принимает имя таблицы/представления/хранимой процедуры и использует множественное или единственное число в названии, в зависимости от того, как имя используется в модели (коллекции будут использовать форму множественного числа, объекты — единственное число и т.д.).
- **Include Foreign Key Columns in the Model.** Предыдущая версия Entity Framework для столбцов внешних ключей в объектах не создавала свойства, делая это только для создания отношений. Однако в ряде сценариев это не подходит, делая невозможным решение ряда простых задач в связи с отсутствием указанных свойств. В новой же версии можно потребовать их включения в объекты с помощью указанной настройки.
- **Model Namespace.** Эта настройка позволяет указать пространство имен, в котором будут создаваться все классы, связанные с моделью. По умолчанию модель существует в своем собственном пространстве имен (которое по умолчанию совпадает с именем модели, введенном в диалоговом окне Add New Item), а не в пространстве имен проекта по умолчанию, чтобы избежать конфликта с существующими в проекте классами с теми же именами.

Выберите в базе данных все таблицы, которые должны быть включены в модель. После щелчка на кнопке Finish будет создана модель Entity Framework, которая отображается на базу данных. Здесь вы можете просматривать модель в Entity Framework и настраивать ее согласно вашим требованиям (а также вашему вкусу или стандартам), чтобы сделать ее идеальной для запросов в вашем коде.

Проектировщик Entity Framework

После создания модели Entity Framework она открывается в проектировщике Entity Framework, показанном на рис. 29.5.

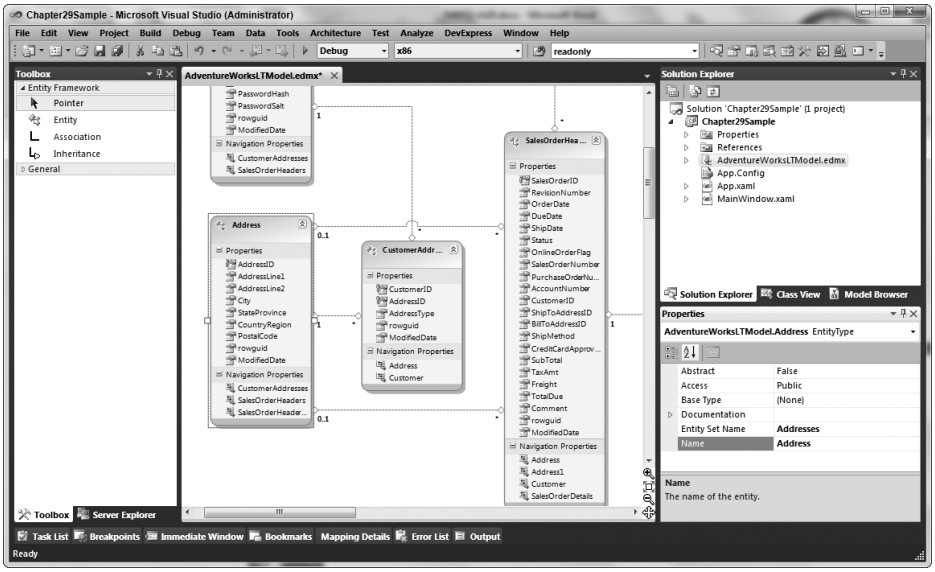


Рис. 29.5

Вы увидите, что проектировщик автоматически расположил все созданные мастером объекты, показывая созданные между ними связи.

Можете перемещать объекты по поверхности проектировщика, и он автоматически будет перемещать линии связей и пытаться аккуратно расположить их на экране. Объекты автоматически выравниваются по сетке, которую можно увидеть, если выбрать команду Grid⇨Show Grid контекстного меню, вызываемого щелчком правой кнопки мыши. Отключить привязку к сетке можно щелчком правой кнопкой мыши на поверхности проектировщика и снятия флажка Grid⇨Show Grid. При этом вы получаете возможность более точного управления диаграммой макета, но при использовании привязки схема все же получается более аккуратной.

При перемещении объектов диаграммы (или добавлении новых) можно запутать диаграмму и получить массу пересекающихся линий связей, проходящих во всех направлениях. Это не беда — всегда можно щелкнуть правой кнопкой мыши на поверхности проектировщика и выбрать в контекстном меню команду Diagram⇨ Layout Diagram, что заставит проектировщик аккуратно расположить объекты в соответствии со своим собственным алгоритмом.

Модели Entity Framework могут быстро стать большими и трудными для навигации в проектировщике Entity Framework. К счастью, в нем имеется несколько инструментов, делающих навигацию немного более легкой. Проектировщик позволяет увеличивать и уменьшать масштаб с помощью кнопок масштабирования в нижнем правом углу (ниже вертикальной полосы прокрутки, как показано на рис. 29.6). Кнопка между кнопками увеличения/уменьшения масштабирует изображение до величины 100%.



Рис. 29.6

Для увеличения до заранее заданной величины щелкните правой кнопкой мыши на поверхности проектировщика и выберите один из вариантов в меню Zoom. В этом меню вы также найдете команду Zoom to Fit (Увеличить до размеров экрана), а также команду Custom, которая выводит диалоговое окно, позволяющее ввести конкретное значение масштаба.

Кроме того, выбор объекта в окне инструмента Properties (из выпадающего списка) автоматически выбирает нужный объект в проектировщике и выводит его в поле зрения; щелчок правой кнопкой мыши на объекте в окне инструментов Model Browser (оно будет описано чуть позже) и выбор команды контекстного меню Show in Designer делает то же самое. Так очень легко перейти к конкретным объектам в проектировщике, и по необходимости вы сможете вносить любые изменения в объекты.

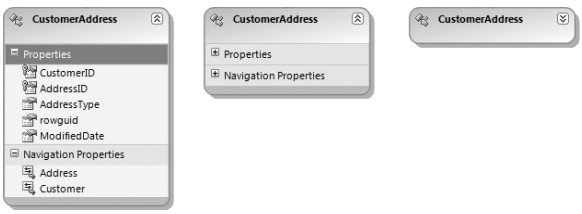


Рис. 29.7

Чтобы минимизировать пространство, занятое объектом, щелкните на пиктограмме в правом верхнем углу объекта. Кроме того, можете сворачивать и разворачивать группы Properties и Navigation Properties, щелкая на пиктограммах +/- слева от них. На рис. 29.7 показан объект в нормальном развернутом состоянии, со свернутыми свойствами и в полностью свернутом состоянии.

Вы можете развернуть все свернутые объекты одновременно, щелкнув правой кнопкой мыши на поверхности проектировщика и выбрав в контекстном меню команду Diagram⇨Expand All. Точно так же можно свернуть все объекты, воспользовавшись командой Diagram⇨Collapse All контекстного меню, вызываемого щелчком правой кнопки мыши.

Визуальное представление модели может оказаться полезным в проектной документации вашего приложения. Проектировщик предоставляет средства для сохранения схемы модели в файле изображения. Щелкните правой кнопкой мыши в любом месте проектировщика и выберите команду Diagram⇨Export as Image из контекстного меню. Открывшееся диалоговое окно Save As позволит выбрать место для сохранения изображений. Обратите внимание на то, что по умолчанию выполняется сохранение в формате BMP. Но в выпадающем списке Save As Type можно выбрать и другие форматы для сохранения, такие как JPEG, GIF, PNG и TIFF. Пожалуй, наилучший компромисс между качеством изображения и размером файла предлагает формат PNG.

Часто бывает полезно отобразить типы каждого свойства объекта (особенно при сохранении диаграммы для использования в документации). Эту возможность можно включить с помощью команды Scalar Property Format⇨Display Name and Type контекстного меню, вызываемого щелчком правой кнопки мыши. Вернуться к показу только имени свойства можно с помощью команды Scalar Property Format⇨Display Name контекстного меню, вызываемого щелчком правой кнопки мыши.

Как и у большинства проектировщиков в Visual Studio, инструментальные окна Toolbox и Properties являются неотъемлемой частью проектировщика. Окно Toolbox (рис. 29.8) содержит три элемента управления: Entity, Association и Inheritance. Как

использовать эти элементы управления, вы узнаете в ближайшее время. В окне Properties отображаются свойства выбранного в проектировщике элемента (или элементов), что позволяет по мере необходимости изменять их значения.

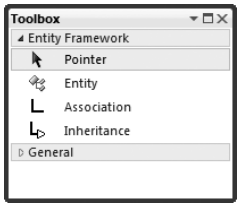


Рис. 29.8

В дополнение к окнам Toolbox и Properties проектировщик Entity Framework включает также два других инструментальных окна — Model Browser и Mapping Details, — характерных только для него и предназначенных для работы с данными.

Инструментальное окно Model Browser (рис. 29.9) позволяет просматривать иерархию как концептуальной модели базы данных, так и ее модели хранения. Щелчок на элементе в иерархии модели хранения показывает его свойства в окне Properties; однако они не могут быть изменены (потому что это инструмент моделирования объекта, а не базы данных). Единственные изменения, которые можно внести в модель хранения, — это удаление таблиц, представлений и хранимых процедур (которые не будут модифицировать лежащую в основе базу данных). Щелчки на элементах в иерархии концептуальной модели также отображают их свойства (которые могут быть изменены) в окне Properties, а их отображения выводятся в окне Mapping Details. Щелчок правой кнопкой мыши на объекте в иерархии и выбор пункта Show in Designer контекстного меню, вызываемого щелчком правой кнопки мыши, выводит выбранный объект или связь в проектировщике.

Второе изображение на рис. 29.9 демонстрирует поиск функциональных возможностей, доступных в окне Model Browser. Как уже говорилось ранее, поскольку ваша модель может быть достаточно большой, может быть трудно найти в ней искомое. А потому хорошая функция поиска оказывается совсем не лишней. Введите искомый термин в текстовое поле поиска в верхней части окна и нажмите клавишу <Enter>. В данном примере ищется слово `Address`, так что в результате оказываются подсвечены все содержащие этот текст имена иерархии (в том числе объекты, связи, свойства и т.д.). Обратите внимание, как на вертикальной полосе прокрутки отмечаются те места в иерархии, где найдены соответствия условию поиска. Чуть ниже текстового поля поиска имеются стрелки, направленные вверх и вниз, с помощью которых можно перемещаться по результатам поиска. По завершении работы можно щелкнуть на пиктограмме в виде креста рядом со стрелками и вернуть обычный режим работы окна.

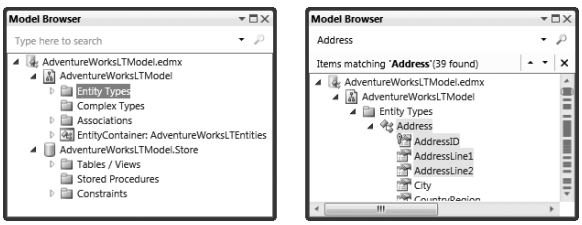


Рис. 29.9

Инструментальное окно Mapping Details (рис. 29.10) позволяет изменять отображение между концептуальной моделью и моделью хранения для объекта. Выбор объекта в проектировщике, окне Mapping Details или Properties показывает в инструментальном окне отображения между свойствами объекта и столбцами в базе данных. У вас есть два способа отображения свойств объекта на базу данных: через таблицы и представления либо через функции (т.е. хранимые процедуры). В левой части инструментального окна имеются две пиктограммы, которые позволяют менять вид между отображением на таблицы и представления и между отображением на функции. Однако здесь мы остановимся только на отображении объектов на таблицы и представления.

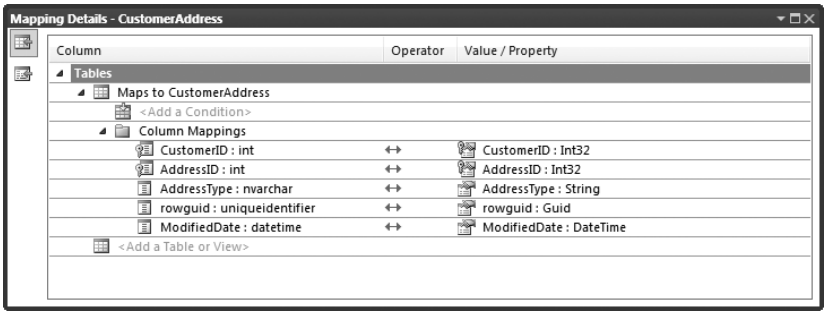


Рис. 29.10

Отображение таблиц/представлений имеет иерархию (в столбце Column), указывающую таблицу (таблицы), отображаемую на объект, со столбцами под ним. Для этих столбцов можете указать свойства вашего объекта (в столбце Value/Property), щелкнув в ячейке, открыв выпадающий список и выбрав в нем соответствующего свойства.

Один объект может отображаться на несколько таблиц или отображений базы данных (в одном объекте, как говорилось ранее, могут объединяться несколько таблиц или отображений). Чтобы добавить в иерархию другую таблицу или представление для отображения на ваш объект, щелкните на кнопке в нижней строке, где имеется надпись <Add a Table or View>, и выберите из выпадающего списка таблицу или представление. При добавлении таблицы в окно Mapping Details для отображения на объект автоматически выполняется сопоставление столбцов с тем же именем на свойства объектов, и создается отображение между ними. Удалить таблицу из иерархии можно путем выбора ее строки и нажатия клавиши <Delete>.

Условия представляют собой мощную возможность технологии Entity Framework, которая позволяет выборочно выбирать таблицу, которую требуется отобразить на объект во время выполнения на основе одного или нескольких указанных вами условий. Например, у вас есть единый объект в вашей модели, `Product`, который отображается на таблицу, которая в базе данных носит название `Products`. Однако у вас есть дополнительные расширенные свойства вашего объекта, которые отображаются на одну из двух таблиц на основе значения свойства `ProductType` объекта, — если это товар одного типа, он будет отображаться на столбец в одной таблице, если другого — в другой. Вы можете добиться этого, добавив условие к отображению таблицы. В окне Mapping Details щелкните в строке непосредственно под таблицей, где имеется надпись `<Add a Condition>`. Откройте выпадающий список, в котором будут содержаться все свойства объекта. Выберите свойство, на котором будет базироваться ваше условие (в данном примере это свойство `ProductType`), выберите оператор и введите значение для сравнения со свойством. Обратите внимание на то, что имеются только два оператора: равно (`=`) и `Is`. Можете ввести дополнительные условия, необходимые для определения того, должна ли таблица использоваться в качестве источника данных для этого свойства.



Отметим, что ряд усовершенствованных возможностей Entity Framework в проектировщике Entity Framework не доступен (такие как работа со схемой хранения, аннотации, ссылки на другие модели и т.д.). Однако эти действия могут быть выполнены путем непосредственного изменения файлов схемы, которые представляют собой XML-файлы.

Создание и изменение объектов

Мастер Entity Data Model Wizard дает хорошую отправную точку, построив сущностную модель. В некоторых случаях этого может быть достаточно, и вы можете начать писать код для запросов, но можете и пройтись по созданной модели и изменить ее дизайн согласно вашим требованиям.

Технология Entity Framework предоставляет концептуальную модель для проектирования и работы, и вы больше не ограничены взаимно однозначными связями между схемой базы данных и объектной моделью, — так что изменения, вносимые в сущностную модель, никак не влияют на базы данных. Таким образом, можете удалить из нее свойства объектов, изменить их имена и тому подобное, и это никак не повлияет на базу данных. Кроме того, поскольку любые вносимые вами изменения выполняются в концептуальной модели, а обновление модели из базы данных не влияет на концептуальную модель, а только на модель хранения, — все внесенные вами изменения не могут быть потеряны.

Изменение имен свойств

Часто приходится иметь дело с базами данных, в которых таблицы и столбцы содержат префиксы и суффиксы, используют верхний или нижний регистр букв в именах и даже имена, которые не соответствуют их реальной функции. Именно здесь использование механизма ORM наподобие технологии Entity Framework может продемонстрировать всю свою мощь, потому что вы можете изменить все упомянутое на концептуальном уровне сущностной модели, чтобы упростить работу с кодом (с более понятными и стандартизированными именами объектов и связей), не меняя схему базы данных. К счастью, таблицы и столбцы базы данных `AdventureWorksLT` имеют достаточно понятные имена, но если бы вы захотели сделать это, то вам надо было бы просто дважды щелкнуть на свойстве в проектировщике (или выбрать интересующее вас свойство и нажать клавишу <F2>) и внести изменения в появившееся текстовое поле. Кроме того, можете выбрать свойство в проектировщике, в окне Model Browser или Properties и изменить имя свойства.

Добавление свойств к объекту

Теперь рассмотрим процесс добавления свойства к объекту. Имеются три типа свойств.

- **Скалярные свойства.** Свойства примитивного типа, такие как строка, целое число, логическое значение и т.д.
- **Сложные свойства.** Группировка скалярных свойств аналогична структуре в коде. Такая группировка свойств может сделать вашу модель намного более понятной и управляемой.
- **Свойства навигации.** Используются для навигации по связям. Например, объект `SalesOrderHeader` содержит свойство навигации `SalesOrderDetails`, которое позволяет перейти к коллекции объектов `SalesOrderDetail`, связанных с текущим объектом `SalesOrderHeader`. Создание связи между двумя объектами автоматически создает необходимые свойства навигации.

Самый простой способ испытать эту возможность — удалить свойство из существующего объекта и вновь добавить его вручную. Удалите свойство объекта, выбрав его

в проектировщике и нажав клавишу <Delete>. Теперь, чтобы вернуть свойство обратно, выберите команду Add⇨Scalar Property контекстного меню, вызываемого щелчком правой кнопки мыши. При создании большого количества свойств проще выбрать свойство или заголовок свойств и нажать клавишу <Insert>. Новое свойство будет добавлено к объекту с именем, которое отображается в текстовом поле, чтобы вы могли по необходимости изменить его.

Следующим шагом является установка типа свойства, для чего вам придется перейти к окну Properties. Тип по умолчанию — строка, но вы можете изменить его на требующийся вам тип, устанавливая свойство Type.

Свойства, которые вы хотите использовать в качестве ключевых (т.е. свойства, которые используются для идентификации объекта), нуждаются в установке их свойства Entity Key равным `True`. В проектировщике у такого свойства появится маленькая пиктограмма в виде ключа, что позволяет легко определить, какие свойства используются для идентификации объекта.

Можете установить множество других свойств свойства, включая значение по умолчанию, максимальную длину (для строк), и допустимо ли отсутствие значения для данного свойства. Можете также указать доступность методов получения и установки значения свойства — открытые, закрытые и так далее; это полезно, например, для свойства, которое будет отображено на столбец с вычисляемым значением, когда установка значения не имеет смысла: достаточно объявить метод установки закрытым, и приложение не сможет им воспользоваться.

Последняя задача заключается в отображении свойства на модель хранения. Это можно сделать, как описано выше в главе, с помощью инструментального окна Mapping Details.

Создание сложных типов

Создание сложного типа состоит в группировке различных связанных между собой свойств в одно составное свойство. Хотя можно создать сложный тип с нуля, более простой способ заключается в реорганизации, когда для включения в сложный тип выбираются скалярные свойства объекта, а затем проектировщик создает из них требуемый сложный тип. Следуйте приведенным далее инструкциям, чтобы собрать свойства объекта `Customer`, связанные с именем клиента, в один сложный тип.

- Выберите свойства объекта `Customer`, связанные с именем клиента (`FirstName`, `LastName`, `MiddleName`, `NameStyle`, `Suffix`, `Title`), выделив первое свойство, и удерживая нажатой клавишу <Ctrl>, выделите другие требуемые свойства (так они окажутся выбранными одновременно).
- Щелкните правой кнопкой мыши на одном из выбранных свойств и выберите команду контекстного меню Refactor into New Complex Type.
- В окне Model Browser появится вновь созданный сложный тип с именем в текстовом поле, чтобы вы могли изменить его на более подходящее. В данном примере назовем его `CustomerName`.
- Проектировщик Entity Framework создаст сложный тип, добавит в него выбранные свойства, удалит отдельные выбранные свойства объекта и добавит созданный сложный тип как новое свойство объекта на их место. Однако это свойство будет иметь имя `ComplexProperty`, так что вы скорее всего захотите переименовать его в нечто более подходящее. Выберите свойство в проектировщике, нажмите клавишу <F2> и введите в текстовое поле новое имя.

Вы должны убедиться, что работать со сгруппированными свойствами проще — как в проектировщике, так и при написании кода.

Создание объекта

До сих пор вы изменяли существующие объекты, созданные мастером Entity Data Model Wizard. Теперь рассмотрим процесс создания объекта с нуля, а затем — процесс его отображения на таблицу/представление/хранимую процедуру вашей модели хранения. Большинство этих аспектов уже рассматривалось, но мы пройдем все необходимые шаги, чтобы получить настроенные объекты с нуля.

Имеются два способа создания объектов вручную. Первый начинается с выбора команды Add⇨Entity контекстного меню, вызываемого щелчком правой кнопки мыши. Это действие вызывает всплывающее диалоговое окно, показанное на рис. 29.11, которое поможет вам настроить начальную конфигурацию объекта. Когда вы введете имя объекта в поле Entity Name, то заметите, что поле Entity Set автоматически обновляется формой множественного числа имени объекта (хотя, если требуется, можете изменить имя набора объектов на другое). Выпадающий список Base Type позволяет выбрать существующий объект модели, который будет наследоваться (этот вопрос будет рассматриваться в ближайшее время). Имеется также раздел для указания имени и типа свойства, которое будет автоматически создаваться и устанавливаться в качестве ключа объекта.

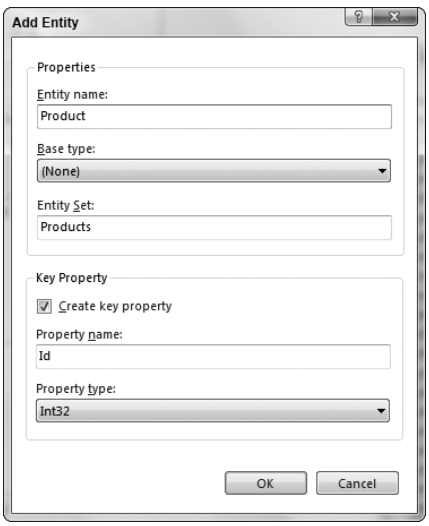


Рис. 29.11

Другой способ создания объекта — перетащить компонент Entity из панели элементов на поверхность проектировщика. Однако это не приводит к появлению диалогового окна из предыдущего метода, а сразу же создает объект с именем, именем набора объектов и ключом по умолчанию. Затем вы можете воспользоваться проектировщиком для изменения конфигурации в соответствии с вашими потребностями.

Ниже описаны действия, которые необходимо выполнить для завершения настройки объекта.

- Создайте связь наследования, указывая, что данный объект должен наследовать базовый объект.
- Создайте необходимые свойства объекта, установив как минимум одно из них в качестве ключа объекта.
- Отобразите эти свойства на схему хранения, воспользовавшись инструментальным окном Mapping Details.
- Создайте необходимые связи с другими объектами модели.
- Проверьте модель, чтобы убедиться, что объект отображен корректно.



Все объекты должны иметь ключ, использующийся для уникальной идентификации объекта. Ключи объектов концептуально представляют собой то же, что и первичные ключи базы данных.

Как уже говорилось ранее, вы не ограничены отображением одного объекта на одну таблицу или представление базы данных. Это одно из преимуществ концептуальной модели базы данных — связанные данные могут располагаться в нескольких таблицах базы данных, но благодаря наличию уровня концептуальной модели в технологии Entity Framework все эти источники можно собрать вместе, в единое целое, чтобы упростить работу с данными в коде.



Убедитесь, что вы не слишком сильно привязаны к структуре базы данных при создании вашей сущностной модели, — преимущество концептуальной модели заключается в том, что она позволяет проектировать модель на основе планируемого использования ее в коде. Поэтому сосредоточьтесь сначала на проектировании сущностной модели, а затем смотрите, как она будет отображаться на базу данных.

Создание и изменение связей объектов

Имеются два способа создания связей между двумя объектами. Первый заключается в использовании команды Add⇨Association контекстного меню, вызываемого щелчком правой кнопки мыши. При этом откроется диалоговое окно, показанное на рис. 29.12.

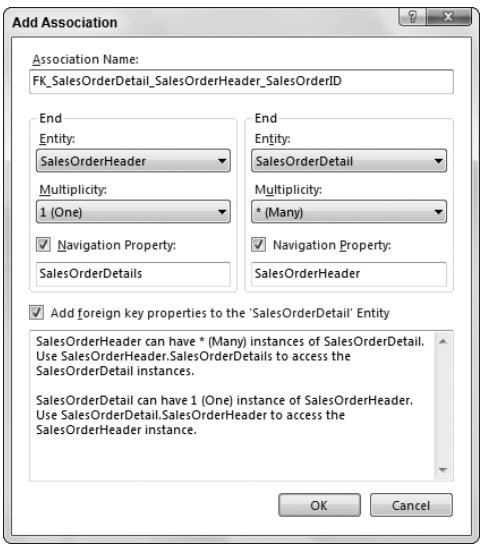


Рис. 29.12

В этом диалоговом окне доступны следующие управляющие элементы.

- **Association Name.** Здесь указывается имя связи, которое станет именем внешнего ключа в базе данных при обновлении базы данных из модели.
- **Endpoints.** Здесь указываются объекты, являющиеся "концами" связи, тип связи ("один к одному", "один ко многим" и т.д.) и имя свойств навигации, которые будут созданы у обоих объектов для перехода от одного объекта к другому по этой связи.
- **Add foreign key properties to the entity.** Этот переключатель позволяет создать свойство у "внешнего" объекта, которое будет действовать в качестве внешнего ключа и отображаться по связи на ключ объекта. Если вы уже добавили свойство, которое формирует внешний ключ для связанного объекта, то должны снять этот переключатель.

Другой способ создания связи — щелкните на компоненте Association на панели инструментов, выберите один объект, который представляет собой один конец связи, затем второй объект на другом конце связи (если это связь типа "один ко многим", первым выберите объект со стороны "один" связи). Использование этого метода дает связи имя по умолчанию, создает свойства навигации у обоих объектов и предполагает наличие связи типа "один ко многим". Свойство внешнего ключа создаваться не будет. По мере необходимости можете изменить эту связь с помощью окна Properties.



Вы не сможете использовать компонент связи при перетаскивании из панели инструментов.

Несмотря на созданную связь, работа еще не закончена (если только вы не использовали первый метод и не создали свойство внешнего ключа). Теперь нужно отобразить свойство, действующее как внешний ключ одного объекта, на ключевое свойство другого. Объект, первичный ключ которого является одной конечной точкой связи, известен, но надо явно указать Entity Framework, какое свойство следует использовать в качестве свойства внешнего ключа. Можете сделать это, выбрав связь в проектировщике и используя окно Mapping Details для отображения свойств.

Как только это будет сделано, вы можете определить ссылочное ограничение для связи, которое можно назначить с помощью щелчка в проектировщике на связи и выбора свойства Referential Constraint в окне Properties.

Наследование объектов

Таким же образом, как классы могут наследовать другие классы (наследование — фундаментальная концепция объектно-ориентированного программирования), объекты могут наследовать другие объекты. Имеется несколько способов указать, что один объект должен наследовать другой, но самый простой способ заключается в выборе объекта в проектировщике, поиске свойства Base Type в окне Properties и выборе из выпадающего списка объекта, который должен быть наследован текущим.

Проверка сущностной модели

Иногда ваша сущностная модель может оказаться некорректной (например, когда свойство объекта не отображается на модель хранения, или его тип не может быть преобразован в/из типа данных столбца в базе данных, на который он отображается); однако, несмотря на некорректность модели, проект будет по-прежнему компилируем.

Можно выполнить проверку, чтобы убедиться в корректности модели. Это делается при помощи пункта Validate контекстного меню, вызываемого щелчком правой кнопки мыши на поверхности проектировщика. При этом выполняется проверка на наличие ошибок в вашей модели, и если они есть, они отображаются в окне Error List.

Можно также установить свойство Validate On Build концептуальной модели равным `True` (для этого щелкните на пустом месте проектировщика, а затем найдите это свойство в окне Properties), что приведет к автоматической проверке модели каждый раз при компиляции проекта. Однако некорректность модели не предотвратит успешной компиляции проекта.

Обновление сущностной модели с изменением базы данных

Как правило, структура базы данных часто обновляется на всех этапах разработки проектов, поэтому необходим способ обновить модель на основании изменений в базе данных. Для этого щелкните правой кнопкой мыши в области проектировщика и выберите команду меню Database⇨Update Model. При этом откроется мастер обновлений Update Wizard, показанный на рис. 29.13. Этот мастер получает схему базы данных, сравнивает ее с текущей моделью хранения и определяет различия. Эти различия отображаются на вкладках мастера: вкладка Add содержит объекты базы дан-

ных, которых нет в вашей модели хранения, вкладка Refresh содержит объекты базы данных, которые отличаются от соответствующих объектов модели хранения, а вкладка Delete содержит объекты базы данных, которые имеются в модели хранения, но не в самой базе данных.

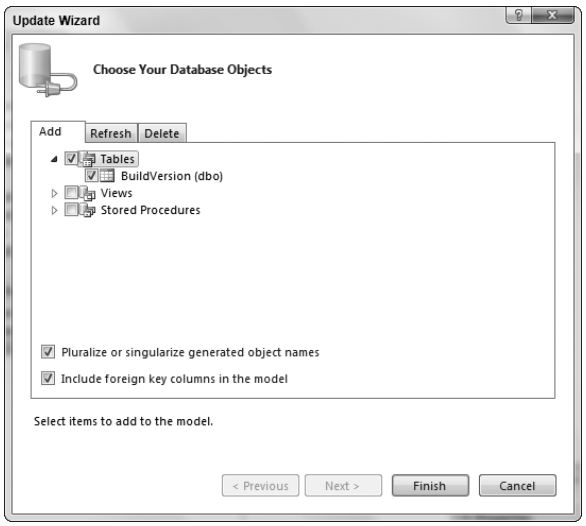


Рис. 29.13

Выберите те элементы на указанных вкладках, которые хотите добавить, обновить или удалить, и щелкните на кнопке Finish, чтобы согласовать модель с базой данных.

Запросы к сущностной модели

Теперь, когда вы создали вашу сущностную модель, вы, несомненно, захотите ее протестировать, выполнив запросы, поработав с полученными данными, внеся в них изменения и сохранив их в базе данных. Entity Framework предоставляет несколько способов выполнения запросов к модели, в том числе LINQ to Entities, Entity SQL и методы построителя запросов. Однако в этой главе основное внимание уделено запросам с помощью LINQ to Entities.

Обзор технологии LINQ to Entities

Технология LINQ рассматривалась в предыдущей главе, где особое внимание было уделено использованию LINQ to Objects, LINQ to SQL и LINQ to XML; однако технология Entity Framework имеет расширение LINQ с собственной реализацией — LINQ to Entities. Технология LINQ to Entities позволяет создавать строго типизированные LINQ-запросы к сущностной модели и возвращает данные в виде объектов. Технология LINQ to Entities отображает LINQ-запросы к концептуальной модели в SQL-запросы к схеме базы данных. Это чрезвычайно мощная возможность Entity Framework, позволяющая абстрагироваться от необходимости писать SQL-запросы для работы с данными из базы данных.

Получение контекста объекта

Для подключения к сущностной модели необходимо создать экземпляр контекста объекта в вашей модели. Так как, когда вы завершаете с работу с контекстом, его надо удалить, имеет смысл использовать блок `using` для указания времени жизни переменной.

VB

```
Using context As New AdventureWorksLTEntities()
     'Здесь выполняются запросы
End Using
```

C#

```
using (AdventureWorksLTEntities context =
        new AdventureWorksLTEntities())
{
    //Здесь выполняются запросы
}
```



Обратите внимание на то, что любые запросы в блоке `using` контекста выполняются во время нахождения контекста объекта в области видимости. Как указывалось в главе 28, выполнение запросов LINQ откладывается до итерирования результатов (т.е. запрос к базе данных не выполняется до тех пор, пока код не использует его результаты). Это означает, что если переменная, содержащая контекст, вышла из области видимости прежде, чем вы на самом деле использовали результаты запроса, то запрос не будет выполнен. Поэтому убедитесь, что вы обратились к результатам запроса до того, как переменная контекста выйдет из области видимости.

Если вам необходимо определить подключение к базе данных (например, если необходимо передать учетные данные пользователя или использовать пользовательскую строку подключения, а не данные из файла `App.config`), вы можете сделать это, передав строку соединения конструктору контекста (в данном случае `AdventureWorksLTEntities`).

Операции CRUD

Трудно спорить с тем, что наиболее важными запросами к базе данных являются запросы CRUD (Create/Read/Update/Delete — Создания/Чтения/Обновления/Удаления). Операции чтения возвращают информацию из базы данных, в то время как остальные операции вносят изменения в последнюю. Создадим несколько запросов LINQ to Entities, чтобы продемонстрировать извлечения некоторых данных из базы (в качестве объектов), изменения этих объектов с последующим сохранением изменений в базе данных.



Хотя написание запросов LINQ to Entities достаточно быстрое дело, можно воспользоваться таким полезным инструментом, как LINQPad, который позволяет создавать запросы к сущностной модели и немедленно их выполнять — так что вы можете сразу же проверить работу написанного вами запроса. LINQPad можно найти по адресу `http://www.linqpad.net`.

Выборка данных

Так же как и в случае SQL, запросы LINQ to Entity состоят из предложений выбора, фильтрации, упорядочения и группировки. Примеры таких запросов вы встретите в этой главе. Результаты запросов могут быть присвоены свойству `ItemsSource` элемента управления `DataGrid`, созданного ранее в файле `MainWindow.xaml`, что позволяет визуализировать эти результаты.

VB

```
dgEntityFrameworkData.ItemsSource = qry
```

C#

```
dgEntityFrameworkData.ItemsSource = qry;
```

Имеется несколько способов запроса сущностной модели в технологии LINQ to Entity, но здесь мы будем работать только с одним из них. Будем также считать, что запрос выполняется в приведенном ранее блоке `using`, с переменной, содержащей экземпляр контекста объекта с именем `context`.

Для возврата всего множества клиентов из базы данных можно написать запрос на выборку следующим образом.

VB

```
Dim qry = From c In context.Customers
          Select c
```

C#

```
var qry = from c in context.Customers
          select c;
```

Отфильтровать результаты можно с помощью предложения `where`, которое может включать даже функции и свойства, такие как `StartsWith`, `Length` и т.д. Приведенный далее пример возвращает всех клиентов, фамилии которых начинаются на букву `A`.

VB

```
Dim qry = From c In context.Customers
    Where c.Name.LastName.StartsWith("A")
    Select c
```

C#

```
var qry = from c in context.Customers
    where c.Name.LastName.StartsWith("A")
    select c;
```

Можно упорядочить результаты с помощью предложения `order by` — в приведенном далее примере результаты упорядочены по фамилиям клиентов.

VB

```
Dim qry = From c In context.Customers
    Order By c.Name.LastName Ascending
    Select c
```

C#

```
var qry = from c in context.Customers
    orderby c.Name.LastName ascending
    select c;
```

Вы можете группировать и агрегировать результаты с помощью предложения `group by` — так, в данном примере результаты группируются по продавцам, возвра-

щая количество продаж у каждого продавца. Заметим, что вместо возвращения объекта Customer вы запрашиваете у механизма LINQ to Entities неявно типизированную переменную, содержащую продавца и количество его продаж.

VB

```
Dim qry = From c In context.Customers
    Group c By salesperson = c.SalesPerson
    Into grouping = Group
    Select New With
    {
        .SalesPerson = salesperson,
        .SalesCount = grouping.Count()
    }
```

C#

```
var qry = from c in context.Customers
    group c by c.SalesPerson into grouping
    select new
    {
        SalesPerson = grouping.Key,
        SalesCount = grouping.Count()
    };
```



Мониторинг SQL запросов, сгенерированных и выполненных технологией Entity Framework, может оказаться полезен для того, чтобы убедиться в корректности взаимодействия сущностной модели и базы данных. Например, вы можете обнаружить, что поскольку связь осуществляется с отложенной загрузкой, при обходе иерархии объектов с использованием этой связи в цикле выполняется многократное излишнее обращение к базе данных. Поэтому, если у вас установлена версия SQL Server Standard (или более мощная версия), можете воспользоваться инструментом SQL Profiler для мониторинга запросов к базе данных и изменения запросов LINQ, если это окажется необходимо. При использовании версии SQL Server Express можно бесплатно загрузить профайлер с открытым исходным кодом SQL Express Profiler с адреса `http://code.google.com/p/sqlexpressprofiler/downloads/list`.

Сохранение данных

Технология Entity Framework использует отслеживание изменений — при внесении изменений в данные в модели оно будет отслеживать изменившиеся данные, и когда вы запрашиваете сохранение изменений в базе данных, они будут внесены в базу в пакетном режиме. Такая фиксация изменений осуществляется с помощью метода `SaveChanges()` контекста объекта.

VB

```
context.SaveChanges()
```

C#

```
context.SaveChanges();
```

Имеется ряд способов обновления существующих данных (для разных сценариев), но для простоты в данном примере мы используем самый простой из них.

Операции обновления

Предположим, вы хотите изменить имя клиента (с идентификатором 1), которого вы выбрали из базы данных следующим образом.

VB

```
Dim qry = From c In context.Customers
    Where c.CustomerID = 1
    Select c

Dim customer As Customer = qry.FirstOrDefault()
```

C#

```
var qry = from c in context.Customers
    where c.CustomerID == 1
    select c;

Customer customer = qry.FirstOrDefault();
```

Все, что вам надо сделать, — это изменить свойство имени у полученного клиента (далее Entity Framework автоматически будет отслеживать это изменение), и вызвать метод `SaveChanges()` контекста объекта.

VB

```
customer.Name.FirstName = "Chris"
customer.Name.LastName = "Anderson"

context.SaveChanges()
```

C#

```
customer.Name.FirstName = "Chris";
customer.Name.LastName = "Anderson";

context.SaveChanges();
```

Операции создания

Чтобы добавить новый объект в множество, создайте экземпляр объекта, присвойте значения его свойствам и сохраните изменения.

VB

```
Customer customer = new Customer()
customer.Name.FirstName = "Chris"
customer.Name.LastName = "Anderson"
customer.Name.Title = "Mr."
customer.PasswordHash = "*****"
customer.PasswordSalt = "*****"
customer.ModifiedDate = DateTime.Now
context.Customers.AddObject(customer)

context.SaveChanges()
```

C#

```
Customer customer = new Customer();
customer.Name.FirstName = "Chris";
customer.Name.LastName = "Anderson";
customer.Name.Title = "Mr.";
customer.PasswordHash = "*****";
customer.PasswordSalt = "*****";
customer.ModifiedDate = DateTime.Now;
context.Customers.AddObject(customer);

context.SaveChanges();
```

После сохранения изменений в базе данных ваш объект будет иметь первичный ключ, автоматически сгенерированный для данной строки базой данных и присвоенный свойству `CustomerID`.

Операции удаления

Для удаления объекта воспользуйтесь методом `DeleteObject()` множества, содержащего этот объект.

VB

```
context.Customers.DeleteObject(customer)
```

C#

```
context.Customers.DeleteObject(customer);
```

Навигация по связям

Конечно, работа с данными редко предполагает использование одной таблицы/объекта, так что свойства навигации свойства, используемые связями, оказываются весьма кстати. Клиент может иметь один или несколько адресов, которые моделируются в сущностной модели объектом `Customer`, имеющим связь с объектом `CustomerAddress` ("один ко многим"), который в свою очередь связан с объектом `Address` ("многие к одному"). Свойства навигации этих связей позволяют очень легко получить адреса клиентов.

Начнем с запроса из более раннего примера, который возвращает объект клиента.

VB

```
Dim qry = From c In context.Customers
    Where c.CustomerID = 1
    Select c

Dim customer As Customer = qry.FirstOrDefault()
```

C#

```
var qry = from c in context.Customers
    where c.CustomerID == 1
    select c;

Customer customer = qry.FirstOrDefault();
```

Перечислить адреса и работать с ними с помощью свойств навигации можно следующим образом.

VB

```
For Each customerAddress As CustomerAddress
    In customer.CustomerAddresses
    Dim address As Address = customerAddress.Address
    ' Некоторые действия с объектом адреса
Next customerAddress
```

C#

```
foreach (CustomerAddress customerAddress
         in customer.CustomerAddresses)
{
    Address address = customerAddress.Address;
    // Некоторые действия с объектом адреса
}
```

Обратите внимание на навигацию через объект CustomerAddress для получения объекта Address клиента. Благодаря наличию связей в механизме Entity Framework нет никакой необходимости в использовании соединений.

Однако при этом возникает одна проблема. Когда вы выполняете переход через объект CustomerAddress к объекту Address, выполняется еще один запрос к базе данных, чтобы получить список объектов CustomerAddress для данного клиента, а затем в цикле for выполняется еще один запрос к базе данных для каждого объекта CustomerAddress, чтобы получить соответствующий объект Address! Все это известно под названием отложенной загрузки — когда сущностная модель запрашивает данные у базы данных только тогда, когда реально в них нуждается. В определенных ситуациях это дает некоторые преимущества, однако в данном случае это приводит к множеству обращений к базе данных, увеличению нагрузки на сервер базы данных, снижению производительности приложения и уменьшению масштабируемости вашего приложения. Таким образом, это определенно не наилучший сценарий.

Вместо этого можно потребовать от сущностной модели, чтобы при запросе клиента выполнялась загрузка связанных объектов CustomerAddress и объектов Address. Так, одним запросом будут загружены все данные и будут сняты все вышеупомянутые вопросы, поскольку при навигации с использованием связей все объекты будут находиться в памяти и дополнительные обращения к базе данных будут не нужны. Выполнить запрос так, как описано выше, можно с применением метода Include, указывая путь (в виде строки) из свойств навигации (в записи с точкой) к связанным объектам, которые вы хотите извлечь из базы данных одновременно с запрашиваемым объектом.

VB

```
Dim qry = From c In context.Customers
          .Include("CustomerAddresses")
          .Include("CustomerAddresses.Address")
      Where c.CustomerID = 1
      Select c

Dim customer As Customer = qry.FirstOrDefault()
```

C#

```
var qry = from c in context.Customers
          .Include("CustomerAddresses")
          .Include("CustomerAddresses.Address")
      where c.CustomerID == 1
      select c;

Customer customer = qry.FirstOrDefault();
```

Расширенная функциональность

В технологии Entity Framework имеется слишком много функциональных возможностей, чтобы подробно рассматривать их в данной главе, поэтому ограничимся кратким обзором некоторых из наиболее важных возможностей, которые в дальнейшем при желании вы можете изучить самостоятельно.

Обновление базы данных из сущностной модели

Как упоминалось ранее, в технологии Entity Framework возможно создание модели с нуля, с последующим созданием базы данных в соответствии с этой моделью. Кроме того, можно начать с существующей базы данных, а затем использовать технологию

Entity Framework для обновления структуры базы данных на основе новых объектов, свойств и связей, добавленных к сущностной модели. Для обновления структуры базы данных на основе дополнений к вашей модели можно использовать мастер Generate Database Wizard, вызываемый командой Generate Database from Model контекстного меню, вызываемого щелчком правой кнопки мыши на проектировщике.

Добавление бизнес-логики в объекты

Хотя мы в основном строим с помощью технологии Entity Framework модели данных, а не бизнес-объекты, тем не менее в объекты можно добавлять и бизнес-логику. Генерируемые механизмом Entity Framework объекты представляют собой частичные классы, что позволят расширять их и добавлять в них свой собственный код. Этот код может реагировать на различные события объекта; можно добавлять к объекту методы, которые клиентское приложение будет использовать для решения различных задач или выполнения своих действий.

Например, вы можете захотеть иметь объект `Product` в модели AdventureWorksLT, которому автоматически присваивается значение свойству `SellEndDate`, если установлено свойство `SellStartDate` (и только если свойство `SellEndDate` не имеет значения). Кроме того, вы, возможно, захотите добавить некоторую логику для проверки, или бизнес-логику, которая должна выполняться при сохранении объекта.

Каждое свойство объекта имеет два частичных метода, которые можно расширить: метод `Changing` (перед изменением свойств) и метод `Changed` (после изменения). Можете расширить эти частичные методы в вашем частичном классе так, чтобы они соответствующим образом реагировали на изменение значения свойства.

Классы POCO (Plain Old CLR Objects)

Одной из основных жалоб на первую версию технологии Entity Framework была та, что ваши объекты должны были быть унаследованы от класса `EntityObject` (или реализовывать набор данных интерфейсов), а такая зависимость от технологии Entity Framework делала их неподходящими для использования в проектах с применением разработки на основе тестов (Test Driven Development –TDD) и разработки на основе предметной области (Domain Driven Design – DDD). Кроме того, многие разработчики хотели, чтобы логика их классов не зависела от того, как они сохраняются (такие классы называются persistence ignorant).

По умолчанию объекты, сгенерированные мастером Entity Model Data Wizard в версии Entity Framework v4, все еще наследуются от класса `EntityObject`, но теперь у вас есть возможность использовать свои собственные классы, которые не должны наследоваться от класса `EntityObject` или реализовывать какие-либо интерфейсы технологии Entity Framework, и чей дизайн находится полностью под вашим контролем. Такие типы классов часто называют Plain Old CLR Objects, или для краткости просто POCO.

Резюме

В этой главе вы узнали, что технология Entity Framework представляет собой механизм объектно-реляционного отображения (ORM), что позволяет создавать концептуальную модель для того, чтобы более продуктивно взаимодействовать с базами данных. Затем вы узнали, как создавать сущностные модели и как писать запросы в своем коде.

30

Создание отчетов

В ЭТОЙ ГЛАВЕ...

- Проектирование отчетов
- Генерация отчетов
- Передача отчетов

Одним из ключевых компонентов почти каждого бизнес-приложения являются отчеты. Имеется множество инструментов, предназначенных для получения отчетов, и зачастую трудно выбрать, какой из них лучше подходит для вашего приложения или системы (все они, как правило, работают по-разному и имеют собственные плюсы и минусы).

Система Visual Studio 2010 содержит встроенный проектировщик отчетов, который сохраняет их в файлы с использованием спецификации RDL; отчеты, созданные с его помощью, могут генерироваться как с использованием локального генератора отчетов, так и путем подготовки на удаленном сервере с работающим SQL Server Reporting Services.

Профессиональные версии Visual Studio 2010 также поставляются с хорошо известным инструментом создания отчетов Crystal Reports. Однако в этой главе мы рассмотрим проектировщик отчетов Visual Studio и узнаем, как использовать его для проектирования и генерации отчетов с использованием локального механизма генерации отчетов.

С чего начать

Когда вы начинаете проектировать отчет, вы либо хотите добавить его в существующий проект, либо начать совершенно новый проект приложения с отчетами. В последнем случае самый простой способ начать работу заключается в создании нового проекта с применением шаблона проекта Reports Application. При этом создаются уже настроенный проект Windows Forms со всеми необходимыми ссылками на

сборки, форма с элементом управления Report Viewer и пустой отчет. Рассмотрим первый сценарий и как начать работу вручную (что требует не так уж много дополнительной работы).

Отчеты могут просматриваться в любом приложении Windows Forms или ASP.NET с помощью элемента управления Report Viewer. Имеются два элемента управления Report Viewer: один для использования в веб-проектах, другой — для использования в проектах Windows Forms, и оба они практически идентичны как по внешнему виду, так и по их использованию для вывода отчетов.



Для вывода отчетов в WPF-приложении можно использовать возможность совместимости Windows Forms, описанную в главе 18, и элемент управления Windows Forms (поскольку в WPF элемент управления Report Viewer отсутствует). Вывод отчетов в приложениях Silverlight немного сложнее, поскольку Silverlight не имеет ни элемента управления Report Viewer, ни поддержки для вывода на печать. В этом случае, пожалуй, лучше всего готовить отчеты в формате PDF, доводить их до клиента с помощью обработчика HTTP и отображать в отдельном окне браузера.

Теперь необходимо добавить в проект ссылки на сборки, которые требуются для использования элемента управления Report Viewer и механизма генерации отчетов. Если вы работаете с проектом ASP.NET, добавьте ссылку на `Microsoft.Reporting.WebForms.dll`, а если работаете с проектом Windows Forms — ссылку на `Microsoft.Reporting.WinForms.dll`. Элемент управления Report Viewer должен быть в панели инструментов для обоих типов проектов, так что, перетащив его в свой отчет, вы автоматически добавите в проект все необходимые ссылки на сборки.

Добавьте файл определения отчета к вашему проекту. Добавьте новый элемент в проект и выберите подраздел Reporting, как показано на рис. 30.1.

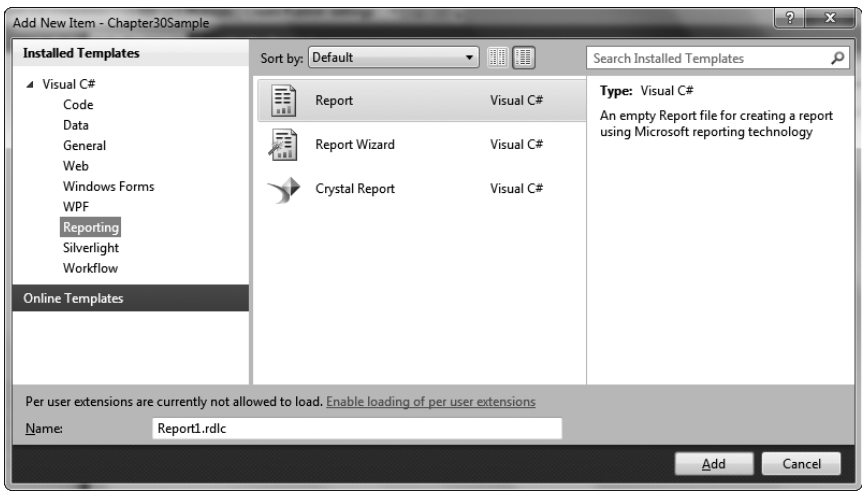


Рис. 30.1

Здесь представляют интерес две команды: Report и Report Wizard. Выбор команды Report создает пустой файл определения отчета — по существу, чистый лист, с которого можете начать работу. Выбор команды Report Wizard создает файл определения

отчета и автоматически запускает мастер отчетов (подробнее о котором рассказывается далее в главе), который и будет разрабатывать макет отчета, основываясь на ваших указаниях. Обычно отчет создается с помощью мастера, а затем модифицируется в соответствии с вашими требованиями.

Перед тем как перейти к проектированию отчета, важно уточнить различные части системы отчетности, условия их использования и взаимодействия (все это вначале кажется очень запутанным). Есть шесть основных частей.

- Проектировщик отчетов (Report Designer).
- Файл определения отчета (Report Definition File).
- Источники данных (Data Sources).
- Механизм генерации отчетов (Reporting Engine).
- Отчет (Report).
- Средство просмотра отчета (Report Viewer).

Вы используете *проектировщик отчетов* для разработки *файла определения отчета*, создания его структуры и определения различных правил размещения отчета. Во время выполнения вы передаете *файл определения отчета* и один или несколько *источников данных механизму генерации отчетов. Механизм генерации отчетов* использует их для генерации *отчета*, который затем выводится *средством просмотра отчета* (или в некотором альтернативном виде, например в формате PDF).



Вас может сбить с толку то, что Report Viewer *является* локальным механизмом генерации отчетов. Таким образом, вы передаете файл определения отчета и источники данных механизму Report Viewer, после чего последний генерирует и выводит отчет. С концептуальной точки зрения, пожалуй, имеет смысл думать о генерации и выводе отчета как об отдельных компонентах.

Проектирование отчетов

Обратимся теперь к проектированию отчета. Сначала рассмотрим ручной процесс проектирования отчета, а затем вы ознакомитесь с тем, как Report Wizard автоматизирует процесс проектирования. Пока что будем иметь дело с пустым отчетом, который был создан путем добавления нового элемента в проект с использованием шаблона отчета. При создании этого элемента он немедленно открывается в проектировщике отчетов, как показано на рис. 30.2.

В области документа имеется поверхность проектирования, на которой вы и располагаете отчет. Внизу слева находится окно Report Data, где будут содержаться поля данных, которые можно перетаскивать в отчет. Если вы случайно закроете это окно, его можно открыть снова, используя команду меню View⇨Report Data. Над ним в окне Toolbox содержатся элементы управления, которые могут быть добавлены на поверхность отчета. Работая с проектировщиком поверхности отчета, вы заметите, что в полосу меню добавляется подменю Report.

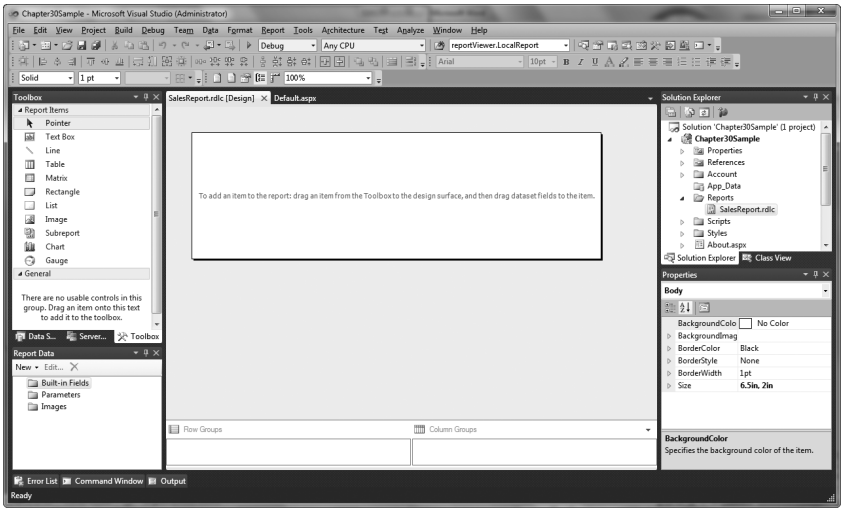


Рис. 30.2



К сожалению, в силу природы локального механизма генерации отчетов, который не может самостоятельно запрашивать источники данных, нет возможности предварительного просмотра отчета в проектировщике. Это означает, что, для того чтобы просмотреть вывод отчета, вы должны создать форму с элементом управления Report Viewer и написать код, который заполняет структуры данных и инициирует процесс вывода. Это может сделать процесс разработки отчета несколько затрудненным, и, возможно, стоит создать временный проект, который позволяет легко просмотреть ваш отчет. Код для решения этой задачи вы найдете ниже в главе.

Определение источников данных

Прежде чем вы сможете проектировать отчет, вы должны разобраться с источником данных, потому что именно источник данных будет определять большую часть дизайна отчета. Во время разработки источник данных не содержит никаких данных, но отчет нуждается в информации об их структуре.

Важно понимать, что при работе с *локальным* механизмом генерации отчетов вы должны передать ему данные для генерации отчета, — самостоятельно такой механизм данные не запрашивает. Положительной стороной этого является то, что данные могут поступать из различных источников; все, что вам нужно сделать, — это запросить данные, после чего вы сможете работать с ними и передать их механизму генерации отчетов в структуре, которую он понимает. Основные структуры, которые можно использовать для заполнения отчета и которые понимает механизм генерации отчетов, включают наборы данных, объекты и объекты Entity Framework.

Можете использовать модель Entity Framework в качестве источника данных для отчета, однако ограничение локального механизма генерации отчетов заключается в том, что вы не сможете объединять в отчете данные из различных источников данных (в данном случае объектов) — что довольно часто требуется в реальных приложениях, если, конечно, вы не импортировали из базы данных в модель Entity

Framework представления, отвечающие требованиям вашего отчета. Таким образом, вам необходимо создать либо типизированный набор данных, либо класс для заполнения соединенными данными, который затем можно будет передать механизму генерации отчетов.



Серверный механизм генерации отчетов (SQL Server Reporting Services) может запросить базу данных SQL Server самостоятельно (как и некоторые другие различные источники данных через OLEDB и ODBC), при этом запрос на получение данных для использования в отчете хранится в файле определения отчета. Вы можете определить эти файлы, которые предназначены для использования SQL Server Reporting Services, довольно легко, потому что они будут иметь расширение `.rdl`, в то время как файлы для использования в локальных механизмах генерации отчетов имеют расширение `.rdlc` (`c` означает обработку на стороне клиента). Достаточно легко конвертировать отчеты, использующие локальные механизмы генерации отчетов, в отчеты, использующие SQL Server Reporting Services, так как основные форматы файлов основаны на одном и том же языке определения отчетов (Report Definition Language — RDL). Причина, по которой вы можете захотеть использовать SQL Server Reporting Services вместо локального механизма, — снизить нагрузку на сервер (например, веб-сервер). Создание отчетов может быть достаточно ресурсоемким и требовать большой загрузки процессора, так что вы можете сделать вашу систему более масштабируемой, передав эту задачу другому серверу. SQL Server Reporting Services требует полной лицензии SQL Server, но если вы используете SQL Server Express Edition, то воспользуйтесь ограниченной версией, установив бесплатный SQL Server Express Edition с Advanced Services.

Для примера воспользуемся моделью Entity Framework AdventureWorksLT, созданной в главе 29, в качестве источника данных для этого отчета. Первый шаг заключается в добавлении объекта этой модели как источника данных для отчета. Для этого откройте меню New в окно инструментов Report Data и выберите команду меню Dataset. Откроется окно Dataset Properties, показанное на рис. 30.3.

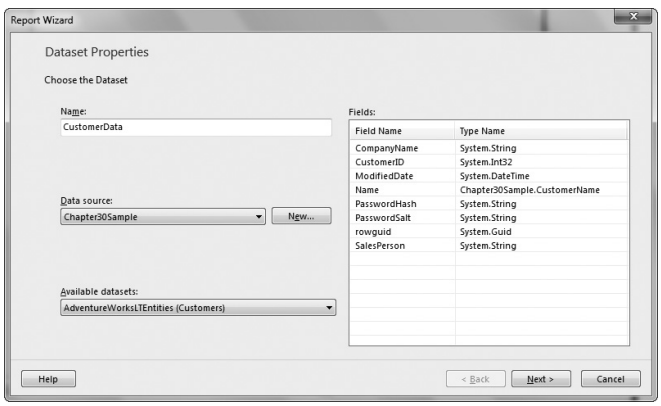


Рис. 30.3

Дайте источнику данных значимое имя, потому что вы будете ссылаться на него в коде при передаче данных локальному механизму генерации отчетов. Введите это имя в текстовое поле Name. Теперь вам нужно выбрать расположение источника данных из раскрывающегося списка Data Source. Обычно источник данных находится в вашем проекте, так что можете просто выбрать его из списка.

Щелкните на кнопке New, чтобы добавить источник данных в проект (например, для создания новой сущностной модели, если таковая еще не существует). Откроется мастер Data Source Configuration Wizard, о котором рассказывалось в главе 27. Будем считать, что в проекте уже имеется модель Entity Framework базы данных AdventureWorksLT, созданная в главе 29, поэтому можете пропустить этот шаг и выбрать тип объектов, которые будут переданы в отчет (в данном примере это объекты `Product`) из раскрывающегося списка Available Datasets. Поиск выбираемого элемента при работе с объектами Entity Framework может сначала показаться довольно запутанным, но первая часть имени элемента представляет собой родительский объект, а имя фактического объекта, который вы хотите использовать в отчете, следует за ним в скобках. Так, для выбора объекта `Product` в модели `AdventureWorksLTEntities` выбирается элемент `AdventureWorksLTEntities(Product)`. При выборе элемента в списке Fields отображаются все его поля. Этот источник данных будет отображаться в окне Report Data, а под ним будут перечислены все поля, которые можно использовать в отчете.

Если этот источник данных изменится (например, если к нему будет добавлено новое поле), щелкните на нем правой кнопкой мыши и выберите из контекстного меню команду Refresh, чтобы обновить структуру источника.

Элементы управления для отчетов

Если вы взглянете на окно Toolbox, то увидите, что в нем имеются типы элементов управления, которые можете использовать в своем отчете, как показано на рис. 30.4.

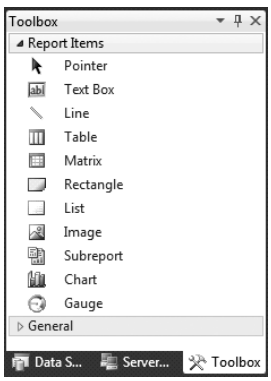


Рис. 30.4

Чтобы использовать элемент управления, перетащите его на свой отчет в нужное местоположение, после чего задайте его свойства с помощью окна Properties. Можно также выбрать элемент управления на панели инструментов и начертить его на поверхности проектировщика отчетов. Еще один метод состоит в щелчке правой кнопкой мыши в любом месте отчета, выборе подменю Insert, а затем элемента управления, который нужно вставить в отчет.

Теперь внимательно рассмотрим каждый из этих элементов.

Элемент управления Text Box

Имя Text Box немного запутывает, так как вы, вероятно, сразу же подумали об элементе управления, в который пользователь может вводить текст (и который не имеет смысла в отчете) — как в элемент управления Text Box в приложениях Windows Forms и других платформ. Эту мысль поддерживает и пиктограмма (текстовое поле с курсором), но на самом деле этот элемент предназначен исключительно для отображения текста, а не для его ввода. Элемент управления Text Box используется не только для отображения статического текста, но может содержать и выражения

(которые вычисляются при генерации отчета, например, значения полей данных, статистические функции и формулы). Выражения могут быть введены непосредственно в текстовое поле или быть созданы с использованием конструктора выражений (о котором рассказывается ниже в этой главе) с помощью щелчка правой кнопкой мыши на текстовом поле и выбора команды меню Expression.

При перетаскивании поля данных в отчет в этом месте создается текстовое поле, содержащее заполнитель. Заполнитель содержит выражение, которое будет получать и отображать значение этого поля. Заполнитель по сути представляет собой способ скрыть выражения в текстовых полях для упрощения конструирования отчетов. Вы можете рассматривать его как функцию без параметров, которая имеет имя (метку поля) и содержит код (выражение). В проектировщике отчетов текстовое поле выводится как метка, а не как (потенциально длинное и сложное) выражение.



Если вы перетащили поле данных на отчет и в нем выводится `<<Expr>>`, это означает, что для данного поля было создано сложное выражение (например, такое, как получение значения поля из первой строки набора данных), которое и скрывается за заполнителем `<<Expr>>`. Вы можете заменить выводимый текст. Сначала щелкните на заполнителе `<<Expr>>`, затем выберите в контекстном меню, которое открывается щелчком правой кнопки мыши, команду Placeholder Properties и дайте содержательное имя, введя его в текстовое поле Label.

Можно также перетащить поле данных в существующее текстовое поле. При этом будет создан заполнитель с выражением для вывода значения перетащенного поля. Вы можете это сделать, если, например, хотите выводить значение этого поля вместе с некоторым статическим текстом или скомбинировать значения нескольких полей в одном текстовом поле.



Можно быстро создать выражение для отображения значения поля данных, вводя имя поля в квадратных скобках (например, `[EmailAddress]`). Этот текст автоматически превратится в заполнитель с выражением для отображения значения соответствующего поля.

Для того чтобы создать заполнитель вручную, переведите поле в режим редактирования (когда в нем выводится курсор для ввода текста), затем щелкните правой кнопкой мыши и выберите в контекстном меню команду Create Placeholder. Создание заполнителей и выражений подробно рассматривается ниже в этой главе.

Формат текста в текстовом поле (в целом) может быть установлен несколькими способами. Свойства форматирования текстового поля могут быть найдены в окне Properties, а кроме того, имеется вкладка Font в окне Text Box Properties для текстового поля (команда Text Box Properties контекстного меню, вызываемого щелчком правой кнопки мыши). Другой способ заключается в использовании настроек форматирования, которые имеются на панели инструментов Report Formatting. Это самый простой способ, но у него есть и другое преимущество. При выборе текстового поля в проектировщике и настроек форматирования из этой панели они будут применены ко всему тексту. Однако текст в поле не обязан быть весь в одном и том же формате. Выделив текст *в текстовом поле* и выбрав настройки форматирования в панели, вы примените их только к выделенному тексту. Конечно, вы можете также использовать

стандартные комбинации клавиш форматирования, такие как <Ctrl+B> для полужирного текста, и иные.

Когда выводится поле данных с числом или датой, довольно часто необходимо отформатировать его для отображения в отчете. Если текстовое поле содержит только выражение, выберите это текстовое поле, щелкните на нем правой кнопкой мыши, выберите команду меню Text Box Properties, а затем вкладку Number (показана на рис. 30.5). Если же текстовое поле содержит текст или другие значения полей, можно отформатировать только значение заполнителя, выделив его в текстовом поле, и щелкнув правой кнопкой мыши с последующим выбором команды меню Placeholder Properties и вкладки Number. Затем из предложенных вариантов форматирования выберите, как именно следует форматировать поле. Если стандартный формат недоступен, выберите в списке Category пункт Custom и введите собственную строку формата или запишите выражение для форматирования значения, щелкнув на кнопке *fx*.

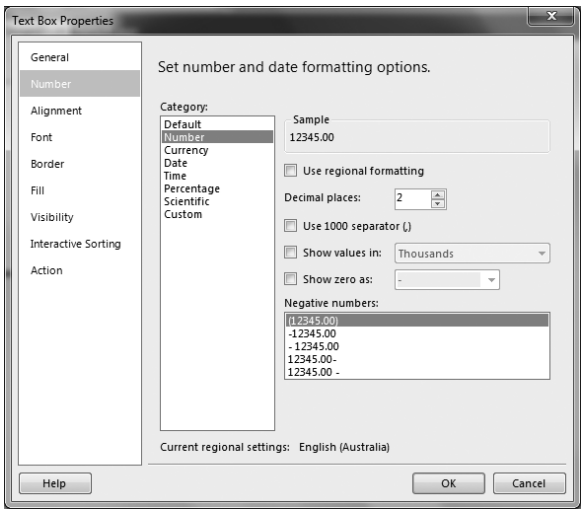


Рис. 30.5

Элементы управления Line и Rectangle

Для черчения в отчете можно использовать элементы управления Line и Rectangle. Элемент управления Line часто используется в качестве разделителя между различными частями отчета. Элемент управления Rectangle, как правило, используется для отделения некоторой области отчета. Элемент управления Rectangle является контейнером, т.е. в нем могут быть расположены другие элементы управления, и при перемещении прямоугольника они будут перемещаться вместе с ним.

Элемент управления Table

Элемент управления Table используется для отображения данных в табличном виде, с фиксированными столбцами и переменным количеством строк (в зависимости от данных, используемых для заполнения отчета). В дополнение к данным таблица может также отображать заголовки столбцов, заголовки групп строк и итоговые строки.

По умолчанию каждая ячейка в таблице представляет собой элемент управления Text Box (а следовательно, каждая ячейка имеет те же возможности, что и описанные для элемента Text Box). Однако ячейка может также содержать любой элемент управления с панели инструментов (например, Image, Chart, Gauge и т.д.) — для этого надо просто перетащить соответствующий элемент управления из панели элементов в ячейку.

При первом перетаскивании элемента управления Table на ваш отчет вы увидите, что он содержит строку заголовка и строку данных, как показано на рис. 30.6.

Для того чтобы отобразить данные в таблице, перетащите поле из соответствующего источника данных в окне Report Data и поместите его в столбец таблицы. Вы увидите заполнитель с выражением для отображения значения в строке данных; кроме того, будет автоматически заполнена названием строка заголовка для этого столбца. Это несколько видоизмененное имя поля — в имя будут вставлены пробелы перед буквами в верхнем регистре (так что поле ProductNumber в заголовке окажется Product Number). Если это имя вас не устраивает, его можно заменить, вводя подходящее имя на месте автоматически созданного.

Другой способ настройки полей, которые должны отображаться в столбцах, состоит в размещении указателя мыши над ячейкой в строке данных и щелчке на пиктограмме в ее правом верхнем углу (рис. 30.7). При этом появится меню, из которого можно выбрать поле для отображения в данном столбце.



Рис. 30.6

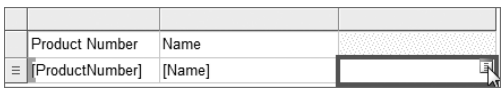


Рис. 30.7

Если в вашем отчете несколько наборов и вы не указали, какой из них является источником данных для таблицы, то после первого щелчка на указанной пиктограмме от вас потребуют указать набор данных. Затем этот набор данных будет установлен в качестве источника данных для таблицы, и в следующий раз после щелчка на пиктограмме будут отображаться только поля из этого набора данных.

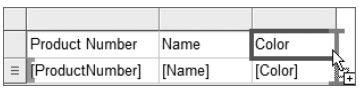


Рис. 30.8

При перетаскивании в отчет таблица будет состоять из трех столбцов, но вы можете добавить дополнительные столбцы, перетаскивая поле из окна Report Data на таблицу так, чтобы точка вставки оказалась на ее правой границе (рис. 30.8).

Точно так же можете вставить столбец в таблицу, позиционируя точку вставки в то место, где он должен быть вставлен. Можно также добавить или вставить новый столбец щелчком правой кнопкой мыши на серой метке-манипуляторе в столбце, после выбрать подменю Insert Column и местоположение (Left или Right) по отношению к выбранному столбцу.

Для того чтобы удалить нежелательные столбцы, щелкните правой кнопкой мыши на серой метке-манипуляторе в столбце и выберите из меню команду Delete Columns.

Таблицы могут содержать данные только из одного набора данных, так что вы не сможете объединять данные из нескольких источников в одной таблице (например, включить данные из источников данных Orders и Customers, чтобы вывести в таблице каждый заказ с именем клиента, который сделал этот заказ). Поэтому такое соединение должно делаться на уровне данных, которые затем будут переданы в отчет.

Чтобы указать, какой набор данных будет служить источником данных для таблицы, выберите ее и свойство DataSetName в окне Properties. Можно изменить используемый источник данных, выбрав из раскрывающегося списка другой.

Зачастую требуется отобразить агрегированные значения в нижней части таблицы, например строку "итого". Есть два способа сделать это. Если у вас есть числовое поле, которое вы хотите просуммировать по всем столбцам, щелкните правой кнопкой мыши на ячейке (не на заполнителе, а на всей ячейке!) и выберите команду Add Total в нижней части меню (она доступна только для числовых полей). При этом будет добавлена новая строка для отображения итоговых значений ниже строки данных, и в нее будет вставлено статистическое выражение SUM, как показано на рис. 30.9.

	Product Number	Name	Color	List Price
≡	[ProductNumber]	[Name]	[Color]	[ListPrice]
				[Sum(ListPrice)]

Рис. 30.9.

Поскольку команда меню Add Total доступна только для числовых полей, вам придется создавать итоговые строки вручную (например, если вы хотите подсчитать количество элементов). Щелкните правой кнопкой мыши на серой метке-манипуляторе строки данных и выберите команду меню Insert Row⇨Outside Group⇨Below. Затем можете написать требуемое статистическое выражение во вновь вставленной строке.

Если хотите изменить тип итоговой статистической функции, вам необходимо изменить выражение. Вместо того чтобы вручную вносить изменения, можно быстро сделать это, выбрав заполнитель (а не ячейку!), щелкнув правой кнопкой мыши, выбрав подменю Summarize By, а в нем — искомую статистическую функцию.

Таблица может фильтровать и сортировать данные из источника перед отображением. Обе эти возможности могут настраиваться в окне Tablix Properties (команда Tablix Properties контекстного меню, вызываемого щелчком правой кнопки мыши на серой метке-манипуляторе области таблицы). Вкладка Filter позволяет определить фильтры, каждый из которых состоит из выражения, оператора и значения. Вкладка Sorting позволяет указать одно или несколько полей для сортировки данных и порядок сортировки для каждого из них.

Вы также можете группировать строки в таблице, показывая между группами заголовок каждой группы. Например, можно группировать заказы по клиентам и выводить имя клиента в строке заголовка группы (который соответственно не должен отображаться в виде столбцов). Вы можете иметь несколько уровней группировки, создавая сложные вложенные иерархии. И вновь имеется несколько способов решить эту задачу. Один из них заключается в том, чтобы выбрать таблицу и перетащить поле из окна Report Data на панель Row Groups в нижней части проектировщика *над* уже имеющейся здесь записью (Details). Другой способ, который дает дополнительные возможности для настройки группировки, заключается в щелчке правой кнопкой мыши на серой метке-манипуляторе строки данных и выборе команды меню Add Group⇨Parent Group. При этом на экран будет выведено окно Tablix Group (рис. 30.10).

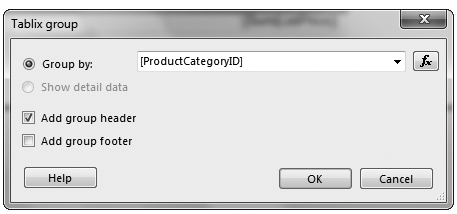


Рис. 30.10

В нем вы можете выбрать поле или выражение для группировки, добавить заголовок группы и/или нижний колонтитул. Например, эти дополнительные возможности могут быть полезны, если вы хотите вывести поле группировки в заголовке над данными, а итоговые значения — под ними.

По умолчанию (даже если вы выберете создание строки заголовка группы или если имеется столбец, выводящий значение поля группировки) новый столбец будет

вставлен слева от данных и настроен для отображения значения поля группы. Вы сможете безопасно удалить этот столбец, не влияя на группировку, если это не то поведение, которое вам требуется.



При добавлении группы со строкой заголовка имеется возможность внесения ряда улучшений в макет отчета. Сначала удалите добавленный столбец, а затем настройте первую ячейку в строке заголовка группы, чтобы отобразить поле группировки. Затем выберите все ячейки в строке заголовка группы, щелкните правой кнопкой мыши и выберите в меню команду Merge Cells, чтобы превратить их в одну ячейку (растягивая поле группировки на все столбцы). Можете также добавить границу или цвет фона строки заголовка группы, чтобы выделить ее.

По умолчанию к таблице не применяется форматирование, кроме сплошной светло-серой рамки вокруг ячеек. Однако зачастую желательно иметь границу вокруг таблицы, между столбцами и даже между отдельными ячейками. Возможно, вы захотите отделить линией заголовок таблицы (и/или нижний колонтитул) от данных. Во всех этих случаях самый простой способ установить границы — это выбрать ячейки, к которым будут применены границы, и использовать панель инструментов Report Borders (рис. 30.11) для их настройки.

Часто также требуется установить цвет фона для строки заголовка (а также соответствующий цвет переднего плана). Самый простой способ сделать это — выбрать ячейки и воспользоваться кнопками Background Color и Foreground Color панели инструментов Report Formatting, чтобы выбрать используемый цвет (рис. 30.12).

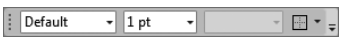


Рис. 30.11

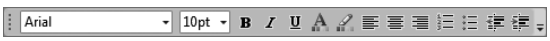


Рис. 30.12

Элемент управления Matrix

Элемент управления Matrix используется для межтабличных отчетов (по аналогии со сводными таблицами в Excel). По сути, элемент управления Matrix группирует данные в двух измерениях (строки и столбцы) и используется при наличии двух переменных и агрегированных данных для каждой их комбинации. Так, например, если вы хотите увидеть общий объем продаж по категориям товаров в каждой стране, элемент управления Matrix будет идеальным решением (рис. 30.13). В этом случае категория товара и страна представляют собой переменные, общий доход — агрегированное значение (продажи продуктов данной категории в данной стране). Матрицы являются одним из самых важных и мощных элементов управления в отчетности, поскольку действительно помогают получить полезную информацию из сырых данных.

	United Kingdom	United States
Bike Racks	$1584.00	$720.00
Bottles and Cages	$86.35	$71.86
Bottom Brackets	$777.52	$542.65
Brakes	$383.40	$447.30
Caps	$91.70	$185.67
Chains	$48.58	$48.58
Cleaners	$137.40	$114.48

Рис. 30.13

Что касается применения элемента управления Matrix в сравнении с таблицами, то это то, что во время разработки вы не знаете, сколько столбцов в ней будет. И количество строк, и количество столбцов матрицы (как и их заголовки) определяются конкретными данными.



Матрицы тесно связаны с таблицами; оба эти элемента управления (а также элемент управления List, который будет вскоре рассмотрен) основаны на базовом элементе управления — Tablix. Тем не менее они рассматриваются как отдельные элементы управления, различающиеся по способу их применения.

При перетаскивании элемента управления Matrix в отчет вы увидите, что он содержит как заголовок столбца, так и заголовок строки, которые пересекаются в ячейке данных (рис. 30.14), и в нижней части проектировщика записи группировки содержат как панель Row Groups, так и Column Groups (в то время как в случае элемента управления Table имелась запись только для группировки строк).

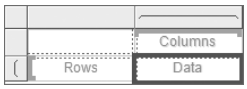


Рис. 30.14

В данном случае вы используете пример вывода общего объема продаж в категории товаров в каждой стране, как описано ранее. Ваш источник данных (коллекция специально созданных и заполненных пользовательских объектов в качестве источника данных для отчета) содержит четыре поля: `ProductCategory`, `Country`, `Revenue` и `OrderQuantity`. Вам нужно перетащить поле `ProductCategory` из окна Report Data на заголовок строки (помеченный как Rows), а поле `Country` — на заголовок столбца (помеченный как Columns). Затем перетащите поле `Revenue` (или `OrderQuantity`) в ячейку данных (помеченную как Data), и дело сделано. В случае числового поля `Revenue` к нему будет автоматически применена агрегирующая функция `SUM`.



Проектировщик автоматически вставляет метку заголовка в левую верхнюю ячейку, но обычно ее удаляют из отчета.

Матрица в проектировщике отчетов будет выглядеть так, как показано на рис. 30.15, а после добавления определенного форматирования вы получите результат, похожий на показанный ранее на рис. 30.13, когда вы генерировали отчет.

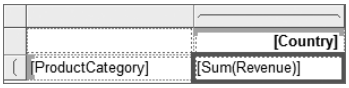


Рис. 30.15

Как и в случае элемента управления Table, можно вывести итоговые данные, но элемент управления Matrix позволяет вывести итоговые данные как для столбцов, так и для строк. После щелчка правой кнопкой мыши на ячейке данных вы заметите, что команда меню Add Total на самом деле представляет собой подменю (в отличие от случая элемента управления Table), из которого можете выбрать итоговое значение строк и столбцов.

Элемент управления Matrix не ограничивает вас одним агрегатным значением на “пересечение”. Например, вы можете показать как общий доход, так и количество для каждой страны и категории товара. Просто поместите другое поле данных для агрегирования (например, поле `OrderQuantity`) рядом с полем `Revenue` в матрице, и оно также будет отображаться для каждой страны (рис. 30.16).

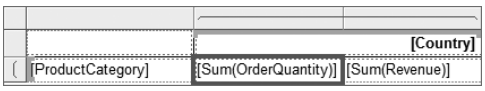


Рис. 30.16

Можно расширить матрицы для отображения дополнительных “измерений” с помощью групп строк или столбцов. И опять-таки — просто перетащите дополнительные поля для группировки в соответствующую позицию в области заголовка группировки строк или столбцов.

Элемент управления List

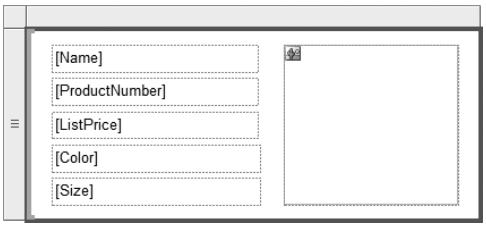


Рис. 30.17

Списки представляют собой более свободное средство отображения данных по сравнению с таблицами или матрицами и тем самым обеспечивают бо́льшую гибкость при выводе данных. Если вы перетащите поле непосредственно в отчет, то увидите, что из набора данных выводятся данные этого поля в первой строке, но элемент управления List позволяет вам определить шаблон (рис. 30.17) и задать источник данных, заполнение и вывод этого шаблона для каждой строки или группы.

Будучи еще одной формой базового элемента управления, используемого элементами управления Table и Matrix, элемент List обладает многими их возможностями.

Элемент управления Image

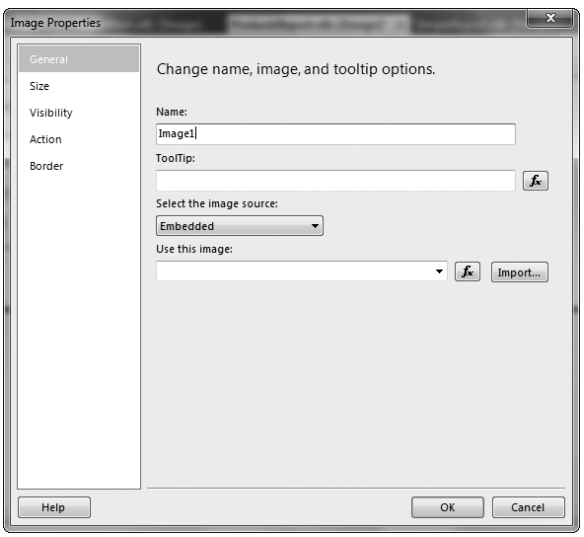


Рис. 30.18

Элемент управления Image используется для вывода изображений в вашем отчете. Источник изображения может находиться в вашем проекте (например, встроенные ресурсы изображений в проекте), быть внешним изображением (определяемым в файловой системе или URL) или быть получен из поля базы данных (типа BLOB). При перетаскивании этого элемента управления в отчет выводится окно, которое позволит установить указанные параметры (и другие, такие как его размер, границы и т.д.), как показано на рис. 30.18.

Варианты, предлагаемые в окне, зависят от источника, выбранного из раскрывающегося списка Select the Image Source.

Если хотите показать внешние изображения (например, заданные путем к файлу), то надо учесть два обстоятельства. Вы должны добавить префикс протокола (например, `file://`, `http://` и т.д.), а кроме того, должны установить свойство `EnableExternalImages` объекта `LocalReport` равным `true`, поскольку по умолчанию возможность вывода внешних изображений отключена.

```
reportViewer.LocalReport.EnableExternalImages = true;
```

Элемент управления Subreport

Элемент управления Subreport используется в качестве заполнителя, куда может быть вставлено содержимое другого отчета (тем самым создаются сложные иерархические отчеты). Этот вопрос рассматривается ниже в главе.

Элемент управления Chart

Диаграммы обеспечивают более наглядное представление данных, позволяя легко отличить нормальные данные и аномалии.

Когда вы перетаскиваете элемент управления Chart в отчет, тут же открывается окно Select Chart Type (рис. 30.19), которое позволяет выбрать один из широкого спектра доступных типов диаграмм.

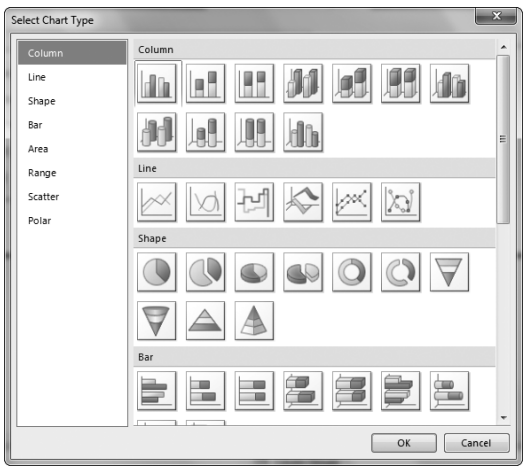


Рис. 30.19

Вы всегда сможете изменить тип диаграммы на более позднем этапе с помощью команды Change Chart Type контекстного меню, вызываемого щелчком правой кнопки мыши.

После двойного щелчка на диаграмме (как и на других элементах управления) последняя переводится в режим редактирования (рис. 30.20), и вы видите, что она состоит из нескольких подэлементов. В зависимости от типа диаграммы можно выбирать различные элементы управления, располагающиеся на ее поверхности. Однако диаграммы любого типа имеют дополнительно название и легенду. Можете изменить эти компоненты или удалить их по своему усмотрению.

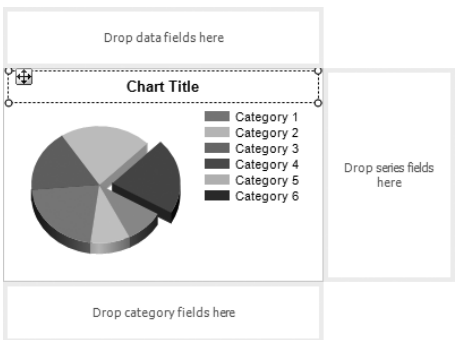


Рис. 30.20

Диаграммы состоят из категорий, рядов и данных, по существу представляющих оси. Категории используются для группировки данных, данные указывают источник значений для отображения, а ряды добавляют дополнительные "измерения", которые будут определены при генерации отчета (это та же концепция, на которой построена работа элемента управления Matrix). Для простых диаграмм настраивают оси категорий и данных, а для более сложных используют еще и оси рядов.

Когда диаграмма находится в режиме редактирования, на ней выводятся рабочие зоны (по одной на каждой оси), куда можно перетащить поля, которые они должны использовать. Для более сложных диаграмм в каждую зону можно перетащить несколько полей для вывода нескольких значений.

Используя тот же источник данных, что и при генерации матрицы, начнем с генерации простой гистограммы (общий объем продаж по категориям). Перетащите на отчет элемент управления Chart, установите его тип 3-D Clustered Bar chart и дважды щелкните на диаграмме, чтобы перевести ее в режим редактирования. Перетащите поле `ProductCategory` в зону Category, а поле `Revenue` — в зону Data. Измените названия диаграммы и осей, как сочтете нужным, и выполните другие необходимые действия, чтобы вывести метки для каждой категории. Щелкните правой кнопкой мыши на вертикальной оси и выберите в меню команду Axis Labels, а также измените значение параметра Interval с Auto на 1. Теперь при генерации отчета вы получите результат, похожий на рис. 30.21.

Заметим, что в настоящее время легенда не имеет никакого реального значения, так как в гистограмме она предназначена для рядов сгруппированных значений, которые в этой диаграмме не используются.

Сгенерируем диаграмму, которая работает так же, как элемент управления Matrix, установив группировку рядов для добавления дополнительного измерения к предыдущей диаграмме (так что она теперь показывает общее количество продаж по каждой категории в стране). Перетащите поле `Country` в зону Series серии и запустите отчет снова. Вы получите общие объемы продаж для каждой категории товара по странам, как показано на рис. 30.22.

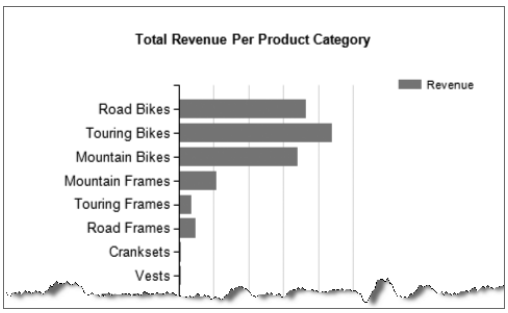


Рис. 30.21

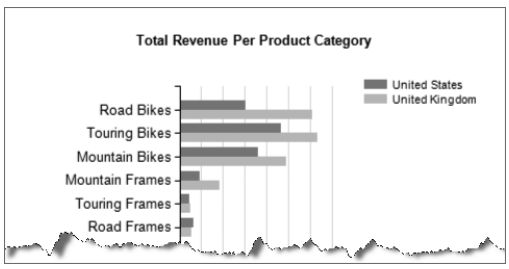


Рис. 30.22

Обратите внимание на то, что теперь легенда указывает, какой цвет соответствует каждой стране, поскольку теперь использована ось рядов.

Элемент управления Gauge

Элемент управления Gauge является еще одним средством визуального представления данных. Датчики, как правило, предназначены для отображения одного значения (хотя некоторые могут выводить фиксированное количество отдельных значений). Это может быть весьма полезно при отображении, например, ключевых показателей эффективности (Key Performance Indicators — KPI).

Когда вы перетаскиваете элемент управления Gauge в отчет, открывается окно Select Gauge Type, показанное на рис. 30.23, которое позволяет выбрать один из различных линейных и радиальных типов датчиков.



В отличие от элемента управления Chart вы не сможете изменить тип элемента управления Gauge после его создания.

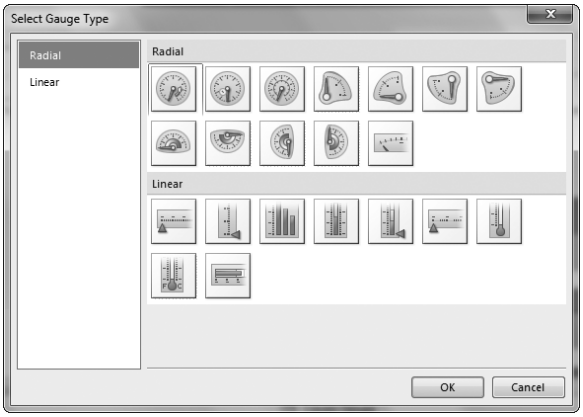


Рис. 30.23

В данном примере используем датчик Radial with Mini Gauge. При переводе датчика в режим редактирования (двойным щелчком мыши) над ним отобразится рабочая зона (рис. 30.24), в которой располагается один или несколько заполнителей полей (в зависимости от того, сколько значений может выводить датчик). Так как мы выбрали датчик для отображения двух значений (основной датчик и мини-датчик), он содержит два заполнителя. При перетаскивании поля из окна Report Data в заполнитель вы увидите автоматическое применение агрегации, поскольку датчик отображает только одно значение. В случае числовых полей будет автоматически применяться функция SUM, а для полей других типов — функция COUNT.



Рис. 30.24

Датчики имеют фиксированный масштаб, и вы должны указать минимальное и максимальное отображаемые значения. Природа элемента управления такова, что он не определяет автоматически эти значения. Чтобы изменить упомянутые значения, щелкните правой кнопкой мыши на элементе управления и выберите команду меню Scale Properties. Откроется диалоговое окно, показанное на рис. 30.25.

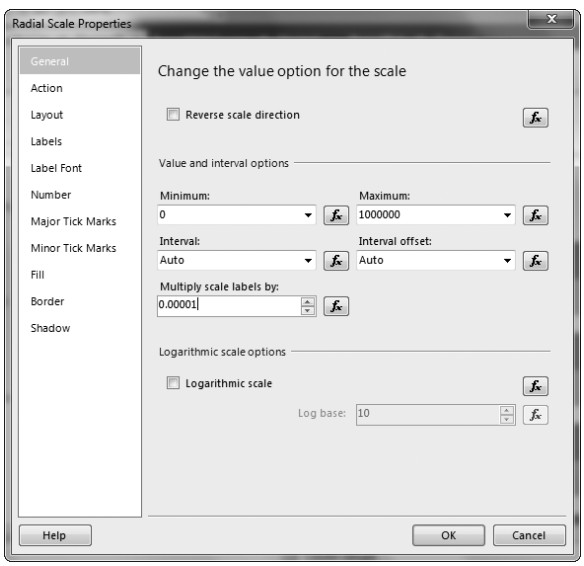


Рис. 30.25

Поскольку в данном примере ожидаются значения до 1 млн., установите это значение в качестве максимального. Опции выбора интервала оставьте автоматически определяемыми, хотя можете изменить их, если метки при выводе выглядят не так, как вы хотите. При работе с очень малыми или очень большими значениями (как в данном примере) может быть полезно установить значение настройки Multiply Scale Labels By. Вместо

вывода больших чисел можно указать, что значения должны умножаться на 0,00001, т.е. вместо 100 000 будет выводиться 1, вместо 200 000 — 2 и так далее (что существенно меньше загромождает элемент управления). В таком случае стоит также добавить метку с пояснением, что на самом деле представляют собой указанные величины. Для этого воспользуйтесь командой Add Label контекстного меню, вызываемого щелчком правой кнопки мыши.

Можете также добавить один или несколько диапазонов для элемента управления. Например, можете указать, что диапазон приемлемых значений будет выделен зеленым цветом, а красным будут выделены значения, которые должны вызывать повышенное внимание. Щелкните правой кнопкой мыши на элементе управления и выберите команду меню Add Range. В элемент управления будет автоматически вставлен диапазон, свойства которого можно изменить с помощью команды Range Properties контекстного меню, вызываемого щелчком правой кнопки мыши. В соответствующем диалоговом окне можно установить значения начала и конца диапазона, а также его начальную и конечную ширину (как правило, это одно и то же значение). На вкладке Fill можете изменить цвет интервала (как правило, зеленый — это "хорошо", красный — "плохо").



Рис. 30.26

Окончательный вид элемента управления показан на рис. 30.26.

Выражения, заполнители и агрегаты

Выражения обеспечивают гибкость и мощь отчетов и используются везде — от получения значений из набора данных, агрегирования и преобразования данных и выполнения расчетов и до процессов принятия решений с использованием условных операторов (`IIF` и т.п.). Все, что динамически вставляется в отчет при его генерации, обрабатывается выражениями. Можете рассматривать выражения как формулы, которые возвращают значения. С помощью выражений в отчете можно управлять почти всем, в том числе большинством свойств элементов управления. К этому времени вы уже видели, что выражения генерируются при перетаскивании полей в отчет и как сложность выражений "скрывается" за заполнителями. Все выражения начинаются со знака равенства (=) и возвращают единственное значение.

Выражения можно разделить на простые и сложные. Простые выражения используют только одно поле, к которому может быть применена статистическая функция. При отображении в проектировщике отчетов простые выражения в качестве метки используют упрощенную версию базового выражения. Ниже приведен пример простого выражения.

```
= Fields!Revenue.Value
```

В проектировщике отчетов оно будет отображаться как `[Revenue]`.

Сложные выражения либо работают с несколькими полями, либо включают операторы и выводятся в проектировщике отчетов как `<<Expr>>` в качестве метки заполнителя по умолчанию (хотя эта строка может быть заменена более значимой). В сложных выражениях используется синтаксис VB, хотя они по-прежнему должны состоять только из одной строки кода, который возвращает значение. Однако они могут вызывать более сложные многострочные функции, о чем будет рассказано в следующем разделе. Пример сложного выражения приведен ниже.

```
= Fields!ProductCategory.Value + " sold to " + Fields!Country.Value
```

Теперь рассмотрим процесс создания выражения. Как говорилось ранее, когда вы перетаскиваете поле в отчет, создается выражение, которое возвращает значение данного поля из набора данных. Чтобы увидеть это в действии, перетащите в отчет таблицу, а затем — поле из окна Report Data в одну из ее клеток. Как уже говорилось выше, то, что отображается в ячейке, — это метка заполнителя. При выборе команды Expression контекстного меню, вызываемого щелчком правой кнопки мыши, можно просмотреть и отредактировать лежащее в ее основе выражение. При этом открывается диалоговое окно Expression Builder, показанное на рис. 30.27.

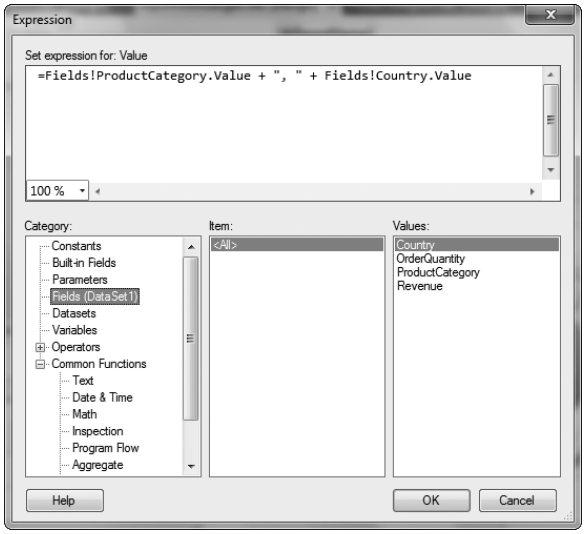


Рис. 30.27

Как можно предположить из названия, окно Expression Builder помогает в создании выражений. В верхней части располагается область кода, где можно ввести выражение, а внизу — дерево категорий, список элементов категорий и список значений (который появляется только при доступных значениях). Область кода поддерживает технологию IntelliSense, всплывающие подсказки (отображающие параметры функций) и проверку синтаксиса (волнистые красные линии подчеркивают найденные ошибки), хотя, к сожалению, подсветка синтаксиса не поддерживается. Нижние области конструктора помогают в создании выражения, что особенно полезно, если вы плохо знаете синтаксис или какая функциональная возможность доступна. Дерево Category позволяет выбрать категорию (например, набор данных, тип оператора, тип функции и т.д.). Список Item отображает доступные в этой категории элементы, а список Values — значения этого элемента, если таковые доступны. Для функций и операторов вместо списка Values отображается некоторая полезная информация о выбранном элементе (что они делают и примеры их использования).

При создании отчета вы заметите, что в диалоговых окнах рядом со многими свойствами имеется кнопка *fx* или элемент Expression (в раскрывающемся списке в окне Properties). Это означает, что данные свойства могут иметь связанные с ними выражения, и щелчок на кнопке или выбор элемента из раскрывающегося списка приводит к открытию диалогового окна Expression Builder, в котором можно создать выражение для управления значением этого свойства. Это особенно полезно в сценариях условного форматирования, таких как включение-выключение видимости или цвет элемента управления, зависящий от выводимых данных.



В сценариях условного форматирования часто используется функция `IIF` (`Inline If`), позволяющая выбирать между двумя значениями на основе результата некоторого выражения (результат применяется в качестве значения свойства). Другими часто используемыми функциями являются `Choose` и `Switch`.

Иногда требуется использовать расчетное значение в нескольких местах отчета и, вместо того, чтобы пересчитывать значение несколько раз, хотелось бы вычислить его только один раз, после чего повторно использовать это вычисленное значение (что ускорит процесс генерации отчета). Здесь могут быть полезными переменные. Исходя из названия "переменные", вы можете подумать, что сможете изменить их значения (например, использовать их в сценарии, вычисляющем итоговые значения), но, к сожалению, это не так. Значение переменных может быть установлено только один раз, а затем оно просто используется без необходимости пересчета.



Вычисление итоговых значений в отчетах в действительности реализуется с помощью функции `RunningValue` (встроенной в механизм генерации отчетов) в выражениях.

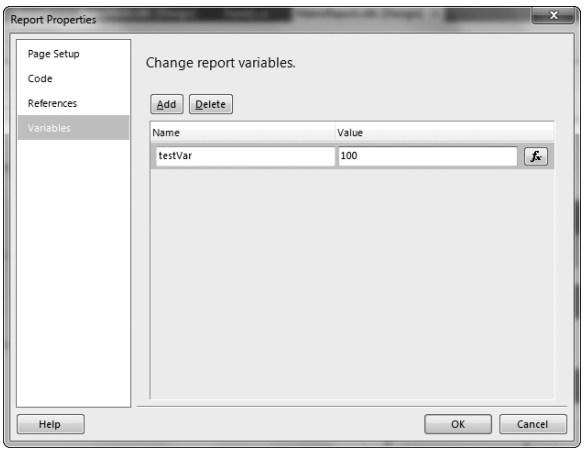


Рис. 30.28

Имеются два типа переменных: переменные отчета и переменные группы, с именами, соответствующими их области видимости. Значение переменной отчета устанавливается в диалоговом окне Report⇨ Report Properties на вкладке Variables, показанной на рис. 30.28.

Определенные здесь переменные будут доступны в любом месте отчета. Новая переменная создается щелчком на кнопке Add, после чего вы можете дать новой переменной имя и значение. Если это константное значение, можете указать его; можете также щелкнуть на кнопке *fx*, чтобы создать выражение для вычисления значения. Такое вычисление будет выполняться только один раз, после чего вычисленное значение будет использовано при всех ссылках на данную переменную.



Переменные, доступные для применения в выражениях, можно найти в Expression Builder в категории Variables.

Так, если вы создали переменную `testVar`, можете использовать ее в выражении примерно следующим образом.

```
= Variables!testVar.Value
```

Еще одно применение переменных отчета — для определения константных значений для последующего использования в отчете, что позволяет централизованно переопределять значения, которые используются в нескольких местах отчета, без жесткого кодирования их в этих местах.

Другой тип переменной — переменные группы. Они работают так же, как и переменные отчета, за исключением того, что область видимости ограничивается текущей группировкой в элементе управления Table/Matrix/List (а также дочерними группировками). Соответствующее значение вычисляется каждый раз при изменении груп-

пировки, поэтому если у вас имеются вычисления для каждой группы, которые затем используются в этой группе повторно, используйте для них переменные группы. Чтобы создать переменные группы, откройте окно Group Properties, перейдите на вкладку Variables, а затем создайте и используйте переменные так же, как и переменные отчета. Можете протестировать как повторное использование вычисленного значения, так и повторный пересчет при изменении группы, создав следующее выражение и отслеживая изменение вывода.

```
= Round(Rnd() * 100)
```

Пользовательский код

Иногда встроенных функций недостаточно для решения поставленных задач. Если необходима сложная многострочная функция для выполнения расчетов или принятия решения, она должна быть написана вне конструктора выражений (так как выражения могут быть только однострочными). У вас есть два пути достижения этой цели: внедрение кода в сам отчет или ссылка на внешнюю сборку .NET, которая содержит пользовательские функции. Оба варианта доступны на уровне отчета с помощью команды меню Report⇨Report Properties.

Когда вы выберете вкладку Code, вы увидите окно, показанное на рис. 30.29, в котором уже введен исходный текст пользовательской функции.

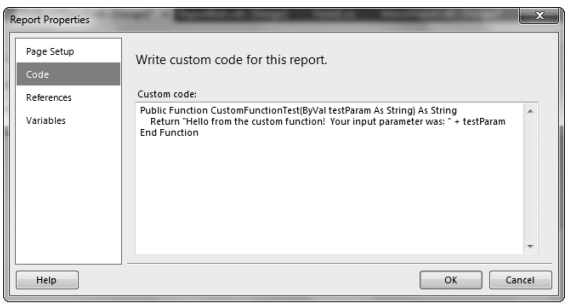


Рис. 30.29

Как видите, это очень простой редактор кода. В нем нет подсветки синтаксиса, проверки ошибок или поддержки IntelliSense, так что использовать его не очень приятно. Если в коде есть ошибки, они будут обнаружены, когда компиляция проекта будет неуспешна (с указанием причины ошибки в окне `Error List`). После того как вы напишете исходный текст функции (с использованием VB в качестве языка программирования), можете добавить в ваш отчет текстовое поле, открытый конструктор выражений и вызвать функцию следующим образом.

```
= Code.CustomFunctionTest("Test Input")
```



Обратите внимание на то, что IntelliSense в конструкторе выражений не отображает имена доступных функций при вводе в редакторе текста `Code.` и какие параметры принимает функция. Кроме того, автоматически добавляются только ссылки на сборки `System.Convert`, `System.Math` и `Microsoft.VisualBasic`. Если вам нужно использовать иные сборки, кроме этих, добавьте ссылки на них на вкладке References, которую мы рассмотрим в ближайшее время.

Вызов функции, показанной на рис. 30.29, выведет в текстовом поле следующее.

```
Hello from the custom function! Your input parameter was: Test Input
```

Если хотите повторно использовать пользовательские функции в нескольких отчетах, лучше внесите код в. NET-сборку и ссылайтесь на нее в каждом отчете, в котором требуются эти функции. Можете создать проект библиотеки классов, написать код (на языке программирования VB или C#), а затем ссылаться на него в отчете. К сожалению, вы столкнетесь с трудностями, связанными с доступностью сборки для отчета и настройками безопасности, чтобы отчет мог воспользоваться вашими функциями, — это далеко не простой процесс. Но, чтобы пользовательские функции заработали, вам придется пройти через него.

Создайте новый проект с использованием шаблона Class Library под названием `CustomReportingFunctions`. Создайте класс `MyFunctions` и добавьте в него следующую функцию.

VB

```
Public Shared Function _
    CustomFunctionTest(ByVal testParam As String) As String
    Return "Your input parameter was: " + testParam
End Function
```

C#

```
public static string CustomFunctionTest(string testParam)
{
    return "Your input parameter was: " + testParam;
}
```

Вы должны также добавить к сборке следующий атрибут, для того чтобы обеспечить возможность ее вызова механизмом создания отчетов. Он добавляется к `AssemblyInfo.vb` в случае разработки на VB (в папке `My Project`, чтобы найти файл, следует включить опцию Show All Files) и к `AssemblyInfo.cs` — в случае разработки на C# (в папке `Properties`).

VB

```
<Assembly: System.Security.AllowPartiallyTrustedCallers>
```

C#

```
[assembly: System.Security.AllowPartiallyTrustedCallers]
```

Чтобы отчет мог обнаружить сборку, она должна быть инсталлирована в глобальный кэш сборок (Global Assembly Cache — GAC). Это означает, что вы должны снабдить сборку *криптографической подписью* (strong name), открыв окно Properties сборки, выбрав вкладку Signing, установив переключатель Sign the Assembly и выбирая/создавая ключевой файл криптографической подписи. Затем скомпилируйте проект и инсталлируйте сборку в GAC, открыв командную строку Visual Studio и введя команду

```
gacutil -i <путь_к_сборке>
```

где `<путь_к_сборке>` следует заменить фактическим путем к скомпилированной сборке.



Всякий раз при обновлении сборки не забывайте инсталлировать ее в глобальном кэше GAC.

Теперь можете ссылаться на сборку в отчете. Откройте окно Report Properties и перейдите на вкладку References (рис. 30.30). Щелкните на кнопке Add, а затем на кнопке с многоточием у появившегося пустого поля ввода. Найдите нужную сборку и щелкните на кнопке OK.

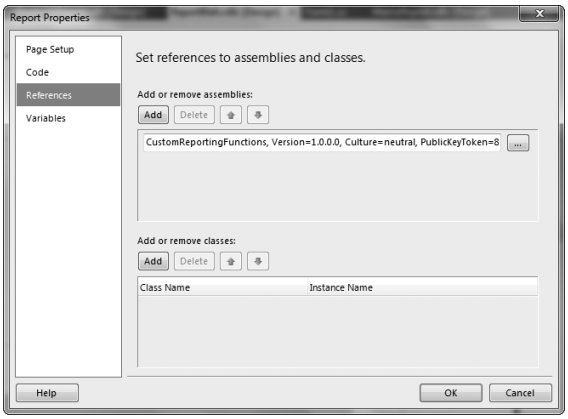


Рис. 30.30

Обратите внимание на область Add or Remove Classes ниже области Add or Remove Assemblies. Она используется для автоматического создания экземпляров классов в сборках. Вы сделали свою функцию доступной для совместного использования (статической, как это называется в C#), так что вам не нужен экземпляр класса `MyFunctions`. Однако, если функция не является совместно используемой и вам необходим экземпляр класса, сконфигурируйте эти экземпляры здесь (потому что в выражении невозможно создать экземпляр класса). Для этого укажите имя класса (включая пространство имен) и дайте ему имя экземпляра (т.е. имя переменной, которую будете использовать в выражениях для ссылки на экземпляр класса). Механизм генерации отчетов осуществит создание экземпляров класса и присвоит ссылку переменной с указанным именем, так что ее можно будет использовать в выражениях.

Итак, вы готовы к обращению к вашей функции в выражении, хотя и несколько иначе, чем вы использовали функцию, встроенную в отчет. Вам необходимо обратиться к функции по ее полному имени, которое включает пространство имен, класс и имя функции. Соответствующий пример приведен ниже.

```
=CustomReportingFunctions.MyFunctions.CustomFunctionTest(" Test Input")
```

Работа почти сделана, но еще не совсем. Последний кусочек головоломки — указать, что сборка должна запускаться с полным доверием в области механизма генерации отчетов Это делается в начале процесса подготовки отчета (он будет рассмотрен немного позже) и требует строгое имя сборки.

VB

```
Dim customAssemblyName As String = "CustomReportingFunctions,
                                    Version=1.0.0.0, " & "Culture=neutral,
                                              PublicKeyToken=b9c8e588f9750854"

Dim customAssembly As Assembly = Assembly.Load(customAssemblyName)
Dim assemblyStrongName As StrongName = CreateStrongName(customAssembly)
reportEngine.AddFullTrustModuleInSandboxAppDomain(assemblyStrongName)
```

C#

```
string customAssemblyName = "CustomReportingFunctions Version=1.0.0.0, " +
                            "Culture=neutral, PublicKeyToken=b9c8e588f9750854";
Assembly customAssembly = Assembly.Load(customAssemblyName);
StrongName assemblyStrongName = CreateStrongName(customAssembly);
reportEngine. AddFullTrustModuleInSandboxAppDomain(assemblyStrongName);
```

Следует сделать пару замечаний об этом коде. Первое заключается в том, что пользовательская сборка загружается из кэша GAC по ее имени, включая версию, культуру и лексему открытого ключа. Эту строку можно получить копированием из того места, где вы добавляли к отчету ссылку на сборку в его диалоговом окне Report Properties.

Второе — это применение функции `GetStrongName` для получения объекта `StrongName`, код которой приводится ниже.



VB

```
Private Shared Function CreateStrongName(ByVal assembly As Assembly)_
                                                          As StrongName
    Dim assemblyName As AssemblyName = assembly.GetName()

    If assemblyName Is Nothing Then
        Throw New InvalidOperationException("Could not get assembly name")
    End If

    ' Получение открытого ключа
    Dim publicKey As Byte() = assemblyName.GetPublicKey()

    If publicKey Is Nothing OrElse publicKey.Length = 0 Then
        Throw New InvalidOperationException("Assembly is not strongly named")
    End If

    Dim keyBlob As New StrongNamePublicKeyBlob(publicKey)

    ' Создание StrongName
    Return New StrongName(keyBlob, assemblyName.Name, assemblyName.Version)
End Function
```

Сниппет кода CreateStrongName.vb



C#

```
private static StrongName CreateStrongName(Assembly assembly)
{
    AssemblyName assemblyName = assembly.GetName();
    if (assemblyName == null)
        throw new InvalidOperationException("Could not get assembly name");

    // Получение открытого ключа
    byte[] publicKey = assemblyName.GetPublicKey();

    if (publicKey == null || publicKey.Length == 0)
        throw new InvalidOperationException("Assembly is not strongly
                                            named");

    StrongNamePublicKeyBlob keyBlob = new StrongNamePublicKeyBlob(publicKey);

    // Создание StrongName
    return new StrongName(keyBlob, assemblyName.Name, assemblyName.Version);
}
```

Сниппет кода CreateStrongName.cs

Теперь при запуске отчета будет получен тот же вывод, что и при использовании внедренного в отчет кода, но в более удобной для работы форме.

Макет отчета

Обычно отчеты предназначаются для печати, так что вы должны рассмотреть, как ваш отчет будет выглядеть в печатном виде. Первое, в чем следует убедиться, — что размеры вашего отчета соответствуют размеру бумаги, на котором он будет напечатан. Откройте окно Report Properties с помощью команды меню Report⇨Report Properties. Выберите вкладку Page Setup, а на ней — размер бумаги, поля и ориентацию страницы (книжная или альбомная).

Многие отчеты стремятся выйти за рамки одной страницы, так что может быть полезно выводить что-то в верхней и нижней части каждой страницы, чтобы показать, какой компании принадлежит данный отчет, какой номер текущей страницы в отчете и т.д. До сих пор вы работали только с телом отчета, а теперь можно добавить в него верхние и нижние колонтитулы страницы и использовать их для указан-

ных целей. Заголовки страницы, как правило, используются для отображения логотипа компании, ее названия и информации о ней. Нижние колонтитулы страницы, как правило, используются для отображения номеров страниц, названия отчета и, возможно, некоторых итоговых данных, отображаемых на этой странице.

Добавить заголовок страницы в отчет можно с помощью команды меню Report⇨Add Page Header. При этом в проектировщике отчетов в отчет выше текста добавляется область заголовка страницы (рис. 30.31), размер которой можете изменить в соответствии с вашими потребностями и в которую можете поместить различные элементы управления, такие как текстовые поля и изображения. Можете даже разместить такие элементы управления, как Table или Gauge, хотя это делается весьма редко. Если вы перетащите поле из окна Report Data непосредственно на заголовок страницы, то заметите, что он создает сложные выражения (как и в теле отчета), так что, если хотите выводить некоторые итоговые данные, начните с добавления таблицы.

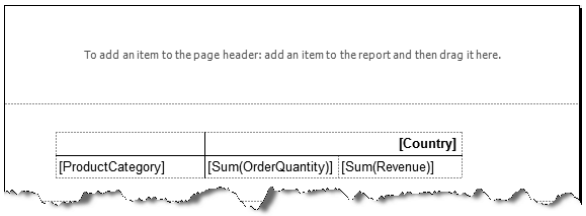


Рис. 30.31

Добавление колонтитула практически ничем не отличается: для добавления нижнего колонтитула выберите команду меню Report⇨Add Page Footer в проектировщике отчетов (рис. 30.32).

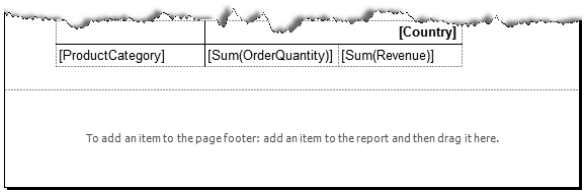


Рис. 30.32

Можете использовать встроенные поля отчета (для отображения такой информации, как номер страницы, количество страниц, название отчета, время составления отчета и т.д.), которые могут быть использованы в любом месте вашего отчета. Их можно найти в категории Built-in Fields в окне Report Data.



Значение поля Report Name получается из имени файла отчета с удаленным расширением.

Обычно номера страниц выводятся в форме “Стр. 1 из 6”. Однако номера страниц и количество страниц в отчете — разные поля, так что для такого вывода надо перетащить в колонтитул текстовое поле, а в него — соответствующие поля в виде

```
Стр. [&PageNumber] из [&TotalPages]
```

Значения в квадратных скобках автоматически преобразуются в заполнители с соответствующими выражениями (символ & указывает, что это ссылки на глобальные переменные), которые получают значения из встроенных полей. Можно также перетащить эти поля из окна Report Data в текстовое поле и добавить между ними статический текст.



Не следует удалять заголовок и колонтитул страницы после их создания (с помощью команд Remove Page Header или Remove Page Footer меню Report), поскольку при этом будет удалено и их содержимое и возврат заголовка или колонтитула назад его уже не вернет. Никакие предупреждения при удалении не выводятся, так что, если вы случайно сделали это, воспользуйтесь функцией Undo для восстановления предыдущего состояния.

Вы можете спросить: как создать верхние и нижние колонтитулы отчета, которые появляются только на его первой или последней странице, а не на каждой. Пример такого заголовка отчета — отображение его названия и другой информации в верхней части отчета (только на первой странице); в конце отчета, на последней странице, может быть размещено примечание, которое будет содержать некоторые итоговые данные.

Проектировщик отчетов не поддерживает заголовки в качестве специальных частей отчета, потому что вы можете просто включить их в основной текст отчета. Поместив содержимое заголовка отчета в его начало, вы добьетесь того, что оно будет отображаться только один раз (при этом оно может охватывать несколько страниц), а в конце отчета можете расположить примечания к отчету. Единственный вопрос — как убрать обычный заголовок на первой странице отчета (поскольку вы хотите иметь только заголовок отчета), и то же относится к колонтитулу на последней странице. Для решения этого вопроса щелкните правой кнопкой мыши на заголовке отчета и выберите из меню команду Header Properties. На вкладке General сбросьте флажок Print on First Page. Повторите этот же процесс и для колонтитула: щелкните на нем правой кнопкой мыши, выберите команду меню Footer Properties, а затем сбросьте флажок Print on Last Page.

Наконец, рассмотрите разрывы страниц в макете отчета. Например, вы можете захотеть, чтобы там, где это возможно, таблица располагалась только на одной странице, а не половина на одной странице, а половина на другой. Возможно, у вас есть сгруппированные данные и вы хотите, чтобы каждая группа начиналась с новой страницы. Чтобы добиться этого, установите разбиение на страницы для тех элементов управления, которые их поддерживают (Table, Matrix, List, Rectangle, Gauge и Chart). Каждый из этих элементов обладает свойством PageBreak (выберите элемент управления в проектировщике отчетов и найдите это свойство в окне Properties). Это дает вам возможность начать новую страницу перед данным элементом управления (или вообще разместить такой элемент управления на отдельной странице). Можете установить значение свойства `KeepTogether` равным `true`, и если вывод элемента управления разбивается на две страницы, то механизм генерации отчетов будет пытаться, начав вывод с новой страницы, разместить этот элемент управления на одной странице. При группировке данных в таблице, матрице или списке можно также установить параметры разбиения страницы для групп. При просмотре свойств группы (щелкните правой кнопкой мыши на группе в панели Row Groups в нижней части проектировщика и выберите из меню Group Properties)

обратите внимание на вкладку Page Breaks. Здесь можно указать, должны ли быть разрывы страницы между группами, а также следует ли делать дополнительные разрывы страницы до и/или после каждой группы.

Подотчеты

Вложенные отчеты представляют собой возможность, которая позволяет вставлять содержимое одного отчета в другой. Можете вставить содержимое какого-либо отчета (без колонтитулов) в другой, добавив элемент управления Subreport в основной отчет и определив для него свойство `ReportPath`, как путь к другому отчету для отображения в указанной области. Подотчеты обладают преимуществом повторного использования, так что изменения, внесенные в одном месте, будут автоматически внесены в другие отчеты (например, стандартный заголовок с информацией о компании, используемый всеми отчетами).

Начнем со сценария, в котором содержание подотчета не связано с главным отчетом. Создайте новый отчет и поместите в него текстовое поле с некоторым текстом. Добавьте элемент управления Subreport в основной доклад и установите свойство `ReportName` равным имени файла другого отчета (но без расширения).



К сожалению, отчет, используемый в качестве подотчета, должен располагаться в той же папке, что и основной отчет.

При запуске проекта и просмотре отчета вы увидите, как содержимое вложенного отчета сливается с основным отчетом.

Перейдем теперь к немного более сложной ситуации — подключим источник данных к подотчету и выведем в нем некоторые данные (автономно от основного отчета). Основной вопрос сейчас заключается в том, как передать данные в отчет, поскольку источники данных не используются совместно основным отчетом и подотчетом. Это делается с помощью обработки события `SubreportProcessing` объекта `LocalReport` в коде, который настраивает элемент управления Report Viewer (более подробно он будет рассмотрен позже). Вам необходимо добавить обработчик этого события так, как показано ниже,

VB

```
AddHandler reportViewer.LocalReport.SubreportProcessing, AddressOf
ProcessSubreport
```

C#

```
reportViewer.LocalReport.SubreportProcessing += ProcessSubreport;
```

и добавить функцию обработки, которая добавляет данные в объект `SubreportProcessingEventArgs`, переданный в качестве параметра (включая имя набора данных).

VB

```
Private Sub ProcessSubreport(ByVal sender As System.Object,
                             ByVal e As SubreportProcessingEventArgs)
    e.DataSources.Add(New ReportDataSource("DataSetName", data))
End Sub
```

C#

```
private void ProcessSubreport(object sender, SubreportProcessingEventArgs e)
{
    e.DataSources.Add(new ReportDataSource("DataSetName", data));
}
```

Теперь при запуске проекта подотчет будет заполнен данными.

Рассмотрим несколько более сложный сценарий, в котором то, что отображается в подотчете, зависит от данных в основном отчете. Например, пусть основной доклад отображает детальную информацию о каждом клиенте, но вы хотите, чтобы под информацией о каждом клиенте были показаны его заказы в прошлом месяце в виде подотчета. Так как для получения сведений о заказах подотчет должен знать, о каком клиенте идет речь, необходимо использовать Report Parameters.



Заметим, что при таком сценарии предполагается масса накладных расходов. Будет множество обращений к базе данных — по одному для каждого клиента для возврата информации о его заказах, что будет только зря загружать сервер баз данных. Лучший и более эффективный способ заключается в возврате соединения информации о клиенте и наборе данных с информацией о заказах и использовании элемента управления Table для группирования данных по клиентам и вывода информации об их заказах. Однако данный сценарий используется сугубо в учебных целях, как пример передачи информации от основного отчета подотчетам.

Создайте отчет, который будет основным отчетом, для отображения детальной информации о каждом клиенте (в списке) и еще один отчет (подотчет), показывающий сделанные клиентом заказы. Под полями детальной информации о клиенте, находящимися в списке, добавьте элемент управления Subreport, который указывает на созданный подотчет, и подключите к нему код, как было описано выше. Вы заметите, что при обработке события `SubreportProcessing` для возврата данных о заказах в подотчет вы должны знать, для какого клиента должен выполняться возврат данных (подотчет будет выводиться для каждого клиента, так что обработчик события также будет вызываться для возврата информации о заказах для каждого клиента). Здесь необходимо создать Report Parameter для подотчета, который основной отчет будет использовать для передачи ему идентификатора текущего клиента.

Чтобы добавить новый параметр в подотчет, перейдите к окну Report Data, щелкните правой кнопкой мыши на папке Parameters и выберите из меню команду Add Parameter. Создайте параметр `CustomerID` и установите его тип данных `Integer`.

Вернитесь к основному отчету, выберите в проектировщике элемент управления Subreport, затем команду Subreport Properties контекстного меню, вызываемого щелчком правой кнопки мыши, и перейдите к вкладке Parameters. Щелкните на кнопке Add, укажите `CustomerID` в качестве имени параметра и введите `[CustomerID]` в качестве его значения. Теперь всякий раз при выводе подотчета ему будет передаваться текущее значение поля идентификатора клиента.

Последнее, что надо сделать, — это получить значение параметра в обработчике события `ProcessSubreport` и соответствующим образом отфильтровать полученные результаты.

VB

```
Private Sub ProcessSubreport(ByVal sender As System.Object,
                             ByVal e As SubreportProcessingEventArgs)
    Dim customerID As Integer =
```

```
        Convert.ToInt32(e.Parameters("CustomerID").Values(0))
    Dim fromDate As DateTime = DateTime.Today.AddMonths(-1)

    Dim qry = From co In context.SalesOrderHeaders
              Where co.CustomerID = customerID AndAlso co.OrderDate > fromDate
              Select co

    e.DataSources.Add(New ReportDataSource("OrderData", qry))
End Sub
```

C#

```
public void ProcessSubreport(object sender, SubreportProcessingEventArgs e)
{
    int customerID = Convert.ToInt32(e.Parameters["CustomerID"].Values[0]);
    DateTime fromDate = DateTime.Today.AddMonths(-1);

    var qry = from co in context.SalesOrderHeaders
              where co.CustomerID == customerID && co.OrderDate > fromDate
              select co;

    e.DataSources.Add(new ReportDataSource("OrderData", qry));
}
```

Мастер отчетов

Простейший способ начать работу по созданию отчета — воспользоваться мастером Report Wizard. Этот мастер проведет вас через все основные шаги генерации отчета и, основываясь на введенной вами информации, сгенерирует отчет, который вы сможете настроить так, чтобы он отвечал вашим требованиям.

Мастер Report Wizard проведет вас через следующие этапы.

- **Выбор/создание источника данных.** Позволяет выбрать существующий источник данных или создать новый источник данных для отчета. Это тот же шаг, что и описанный выше, в разделе "Определение источников данных".
- **Организация полей.** Перетащите поля в список Values, чтобы создать простую таблицу. Добавьте поля в список Row Groups для группировки строк в таблице по этим полям, а также поля в список Column Groups для группировки столбцов по этим полям (что превратит таблицу в матрицу).
- **Выбор макета.** Позволяет добавить промежуточные итоги и итоговые значения для строк и столбцов.
- **Выбор стиля.** Позволяет выбрать различные цвета и стили для вывода. Если хотите создать свою собственную цветовую схему, сделайте это, модифицируя файл `StyleTemplates.xml` в папке `C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\PrivateAssemblies\1033` на вашем компьютере (этот путь может быть иным на вашей машине, — он зависит от места установки Visual Studio).

Для того чтобы запустить мастер отчетов, необходимо создать новый файл отчета (вы не сможете использовать мастер отчетов для существующего файла или после того, как он запущен). Добавьте новый элемент в проект и из подраздела Reporting выберите новый элемент Report Wizard.

Мастер отчетов проведет вас через ряд шагов для создания простого отчета. После завершения этих шагов будет сгенерирован отчет, который откроется в проектировщике отчетов, чтобы вы могли внести в него изменения в случае необходимости.



Это отличный момент для начала обучения созданию отчетов; когда вы ознакомитесь с разработкой отчетов поближе, вы будете создавать их самостоятельно, используя мастер все реже и реже.

Вывод отчетов

Теперь, когда вы разработали отчет, настало время сгенерировать его, заполнив фактическими данными. Здесь используется элемент управления Report Viewer, так как он содержит локальный механизм генерации отчетов из файлов определения отчета и источников данных.

Элементы управления Report Viewer

Имеются две версии элемента управления Report Viewer: одна для использования в веб-приложениях, другая для использования в приложениях Windows. Однако по способу их применения для генерации и отображения отчетов они практически идентичны.

Windows-версия элемента управления показана на рис. 30.33.

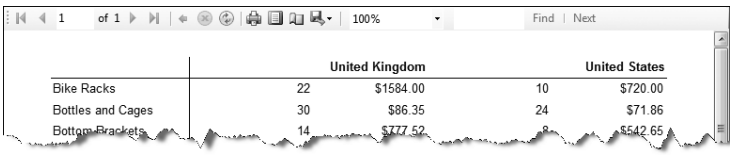


Рис. 30.33

Report Viewer содержит панель инструментов для выполнения различных функций (таких как Refresh, Export, Print и т.д.) и окно для просмотра доклада страница за страницей. Отдельные функции на панели можно отключить с помощью свойств элемента управления Report Viewer; каждая кнопка панели генерирует после щелчка на ней свое событие (хотя соответствующее поведение осуществляется элементом управления Report Viewer автоматически, если только оно не отменено в обработчике событий).

Для того чтобы использовать элемент управления Report Viewer в проекте Windows Forms, перетащите его на форму из панели инструментов.

Веб-версия выглядит очень похоже (рис. 30.34), но отчет выводится в браузере.

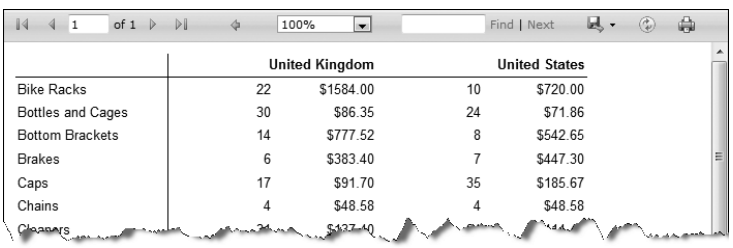


Рис. 30.34

Чтобы использовать веб-версию элемента управления Report Viewer, его можно перетащить на страницу из панели инструментов (на вкладке Reporting). При этом к сборке/пространству имен `Microsoft.ReportViewer.WebForms` добавляется

префикс пространства имен rsweb, а также следующий дескриптор для использования элемента управления Report Viewer.

```
<rsweb:ReportViewer ID="reportViewer" runat="server" />
```

Веб-версия элемента управления Report Viewer требует также наличия на странице Script Manager. Если его еще нет на странице, перетащите его из панели инструментов (вкладка AJAX Extensions).

При выводе отчета в веб-версии элемента управления Report Viewer вы обнаружите, что кнопка Print на панели инструментов отображается только в Internet Explorer (IE), но не в других браузерах, таких как Firefox. Это связано с тем, что для печати отчета из браузера элементу управления Report Viewer нужен элемент управления ActiveX, а элементы управления ActiveX работают только в IE. Поскольку печать из других браузеров невозможна, кнопка Print в них не отображается. После первого щелчка на кнопке Print в IE у вас запросят разрешение на установку элемента управления ActiveX.

Генерация отчета

Процесс формирования доклада сводится к пояснению механизма генерации отчетов, какой файл определения отчета использовать и как передать ему данные (объекты, таблицы данных и т.д.) для заполнения отчета.

По умолчанию файл определения отчета встраивается в сборку, хотя часто лучше иметь его в качестве отдельного файла, чтобы по необходимости его можно было легко обновить без перекомпиляции сборки. Вместе с тем встраивание в сборку означает, что потребуется распространять меньшее количество файлов, что в некоторых случаях предпочтительнее, чем возможность легкого изменения отчета. Установите Build Action в файле определения отчета равным Embedded Resource, чтобы он встраивался в сборку (значение по умолчанию); в противном случае установите его равным Content.

Приведенный далее код требуется для генерации отчета на основе файла определения отчета и заполнения его некоторыми данными (переменная data содержит коллекцию объектов из модели Entity Framework, которая используется для заполнения источника данных CustomerData в отчете).

VB

```
Dim reportEngine As LocalReport = reportViewer.LocalReport
reportEngine.ReportPath = "CustomerReport.rdlc"
reportEngine.DataSources.Add(New ReportDataSource("CustomerData", data))
reportViewer.RefreshReport() ' Только для Windows-версии Report Viewer
```

C#

```
LocalReport reportEngine = reportViewer.LocalReport;
reportEngine.ReportPath = "CustomerReport.rdlc";
reportEngine.DataSources.Add(new ReportDataSource("CustomerData", data));
reportViewer.RefreshReport(); // Только для Windows-версии Report Viewer
```

Здесь вы получаете существующий объект LocalReport от элемента управления Report Viewer, устанавливаете значения для его свойств, а затем используете функцию RefreshReport элемента управления Report Viewer для запуска механизма генерации отчетов.

Если вы выбрали встраивание доклада в сборку, то вместо установки свойства ReportPath объекта LocalReport установите свойство ReportEmbeddedResource. Это должен быть квалифицированный путь к ресурсу (чувствительный к регистру), включающий пространство имен и расширение.

VB

```
reportEngine.ReportEmbeddedResource = "Chapter30Sample.CustomerReport.rdlc"
```

C#

```
reportEngine.ReportEmbeddedResource = "Chapter30Sample.CustomerReport.rdlc";
```

Если в вашем отчете имеется один или несколько подотчетов, то вы должны также обрабатывать событие `SubreportProcessing` объекта `LocalReport`, как это было продемонстрировано при обсуждении элемента управления Subreport. Если вы используете пользовательские сборки, то вам нужно включить код для указания, что пользовательская сборка пользуется доверием. Кроме того, вам может потребоваться установить свойства объекта `LocalReport`, чтобы позволить отчету использовать внешние изображения, гиперссылки и т.д. Представленный здесь код — базовый вариант, необходимый для генерации отчетов и отображения их в элементе управления Report Viewer.

Вывод отчетов в различных форматах

Отчет не обязательно должен выводиться в элементе управления Report Viewer. В некоторых случаях вам может потребоваться создать отчет и отправить его по электронной почте в формате PDF без какого-либо взаимодействия с пользователем или выдать отчет в формате PDF в ответ на вызов веб-службы. Элемент управления Report Viewer позволяет экспортировать отчет в различные форматы (Excel, PDF, Word и т.д.) как с помощью кнопок панели инструментов, так и с помощью кода. Это делается путем создания объекта `LocalReport`, установки его требуемых свойств. Затем с помощью функции `Render` объекта `LocalReport` отчет выводится в указанном формате (в виде потока или массива байтов).

Функция `Render` имеет ряд перегруженных вариантов, самой простой из которых является функция, которой надо передать желаемый формат (в данном случае PDF), и она вернет массив, содержащий отчет.

VB

```
Dim reportOutput As Byte() = reportEngine.Render("PDF")
```

C#

```
byte[] reportOutput = reportEngine.Render("PDF");
```

Механизм генерации отчетов может генерировать отчет в разных форматах. Корректные значения включают следующее.

- **PDF.** Вывод в файл Adobe Acrobat.
- **Word.** Вывод в документ Microsoft Word.
- **Excel.** Вывод в электронную таблицу Microsoft Excel.
- **Image.** Вывод в файл изображения в формате TIFF.

Для вывода в поток (такой как поток ответа HTTP или файловый поток) можно преобразовать массив байтов в поток.

VB

```
Dim stream As MemoryStream = New MemoryStream(reportOutput)
stream.Seek(0, SeekOrigin.Begin)
```

C#

```
MemoryStream stream = new MemoryStream(reportOutput);
stream.Seek(0, SeekOrigin.Begin);
```

В случае больших отчетов (когда может потребоваться слишком много памяти) можно выполнить запись непосредственно в поток с помощью одной из перегрузок функции Render, передавая ей функцию обратного вызова, которая создает и возвращает поток для записи как значение параметра createStream.

VB

```
Private Function CreateReportFileStream(ByVal fileName As String,
                                        ByVal extension As String,
                                        ByVal encoding As Encoding,
                                        ByVal mimeType As String,
                                        ByVal willSeek As Boolean) As Stream
    Return New FileStream(fileName & "." & extension, FileMode.Create)
End Function
```

C#

```
private Stream CreateReportFileStream(string fileName, string extension,
                              Encoding encoding, string mimeType, bool willSeek)
{
    return new FileStream(fileName + "." + extension, FileMode.Create);
}
```

После этого можно вызвать функцию Render следующим образом.

VB

```
Dim warnings As Warning() = Nothing
reportEngine.Render("PDF", Nothing, AddressOf CreateReportFileStream, warnings)
```

C#

```
Warning[] warnings;
reportEngine.Render("PDF", null, CreateReportFileStream, out warnings);
```

Распространение отчетов

Теперь, когда вами разработан отчет, его можно передать пользователям как часть вашего приложения. Однако элемент управления Report Viewer не является частью платформы .NET Framework и, таким образом, должен быть установлен отдельно. Поисковый веб-запрос “Report Viewer Redistributable” должен помочь вам найти инсталлятор сборок Report Viewer.

Альтернативой является распространение сборок Report Viewer, на которые ссылается ваше приложение. Однако следует отметить, что сюда не включен .cab-инсталлятор элемента управления ActiveX, который обеспечивает печать отчета при его просмотре в браузере IE. Если эта функциональная возможность требуется вашему приложению, то лучше всего использовать инсталлятор Report Viewer.

Резюме

В этой главе вы узнали, как использовать проектировщик отчетов Visual Studio для разработки отчета, заполнения его данными и вывода пользователю. К сожалению, отчетность — невероятно сложная тема, и ее невозможно охватить полностью в одной главе. Надеемся, эта глава была неплохим введением в данную тему, которое поможет вам двигаться дальше в правильном направлении.

ЧАСТЬ VII

Службы приложений

- **ГЛАВА 31.** Платформа Windows Communication Foundation (WCF)
- **ГЛАВА 32.** Платформа Windows Workflow Foundation (WF)
- **ГЛАВА 33.** Службы клиентских приложений
- **ГЛАВА 34.** Службы синхронизации
- **ГЛАВА 35.** Службы WCF RIA

31

Платформа Windows Communication Foundation (WCF)

В ЭТОЙ ГЛАВЕ...

- Понятие службы WCF
- Создание службы WCF
- Настройка конечных точек службы WCF
- Хостинг службы WCF
- Потребление службы WCF

Большинство систем требует специальных средств для связи между их отдельными компонентами — как правило, между сервером и клиентом. Этот вид связи обеспечивает множество разных технологий, а платформа Windows Communication Foundation (WCF) предоставляет объединенную архитектуру для их реализации. Эта глава знакомит с архитектурой служб WCF, способами их создания, хостинга, а также потребления служб WCF в вашей системе.

Что такое WCF

В пределах инфраструктуры .NET Framework существует множество способов общения между приложениями, включая дистанционный доступ, веб-службы, несметное количество сетевых протоколов и многие другие способы. Это нередко сбивало с толку разработчиков приложений, которым приходилось не только выбирать под-

ходящую технологию, но и писать весь низкоуровневый код, который должен позволить их приложениям использовать различные технологии в зависимости от того, где или как они были бы развернуты. Например, когда пользователи подключены непосредственно к интранету, для выигрыша в скорости, вероятно, лучше использовать дистанционный доступ или прямое подключение TCP/IP. Но это отнюдь не идеальное решение для связи, если приложение расположено за пределами корпоративного брандмауэра, тогда была бы предпочтительна защищенная веб-служба.

Платформа WCF предназначена для решения проблемы этого вида и предоставления средств сборки технологически агностических приложений обмена сообщениями, которые могут затем быть настроены (в текстовых файлах конфигурации) так, чтобы каждая служба поддерживала определенные технологии и способы ее использования. Таким образом, вам нужно только написать одну службу, и она сможет поддерживать все разнообразие технологий связи, поддерживаемых платформой WCF. По существу, платформа WCF — это унифицированный уровень связи для приложений .NET.

Первые шаги

Служба WCF может быть добавлена к существующему проекту (такому как веб-приложение) или может быть создана как автономный проект. В этом примере вы будете создавать автономную службу, так вы сможете легко увидеть, как одиночная служба может быть настроена и использована во многих сценариях связи.

Открыв диалоговое окно New Project (Новый проект) и щелкнув на разделе WCF (для языка VB или C#), вы увидите несколько разных типов проектов WCF, как показано на рис. 31.1.

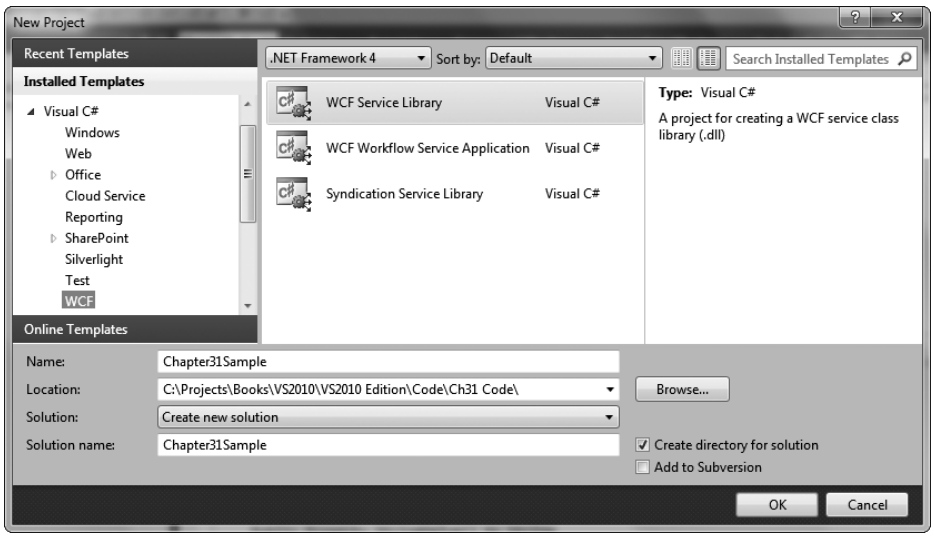


Рис. 31.1

Шаблон проекта WCF Workflow Service Application (Приложение службы WCF Workflow) обеспечивает простейший способ предоставления доступа к Windows Workflow (WF) (это обсуждается в главе 32). Шаблон проекта Syndication Service Library

(Библиотека службы синдикации) используется для предоставления данных как ленты RSS. Однако в примере этой главы вы будете использовать шаблон проекта WCF Service Library (Библиотека службы WCF).



Если просмотреть в диалоговом окне New Project раздел Web, то можно увидеть, что там тоже есть шаблон проекта WCF Service Application, который *не находится* в разделе WCF. Этот шаблон проекта создает службу WCF, которая уже настроена для хостинга в пределах веб-приложения ASP.NET.

По умолчанию новый проект библиотеки службы WCF будет включать файлы `IService1.vb` и `Service1.vb` (или `.cs`, если вы используете язык C#), которые определяют контракт и реализацию базовой службы соответственно. Когда вы откроете эти файлы, вы увидите, что они уже предоставляют некоторые операции и данные, как пример для собственных операций и данных. Все это можно убрать, пока у вас есть лишь интерфейс, не содержащий ничего определенного (оставив, тем не менее, атрибут `ServiceContract`), и класс, который реализует этот интерфейс. Либо можете удалить оба файла и начать заново.

Когда вы захотите добавить дополнительные службы в свой проект, вы найдете в диалоговом окне Add New Item шаблон элемента WCF Service, который добавит в ваш проект и интерфейс, и класс для использования контракта и реализации службы.

Определение контрактов

Этот пример проекта предоставит некоторые данные из объектной модели Entity Framework, которую вы создали в главе 29 для базы данных `AdventureWorksLT`, а также некоторые операции, которые могут быть выполнены с этими данными. Для этого создаются контракты, определяющие операции и структуру данных, которые будут предоставлены во вне. Существуют три основных типа контрактов: контракты службы, контракты данных и контракты сообщений.

- Контракт службы — это группа операций, по существу, детализирующих возможности службы.
- Контракт данных детализирует структуру данных, передаваемых между службой и клиентом.
- Контракт сообщений детализирует структуру сообщений, передаваемых между службой и клиентом. Это важно, когда служба должна соответствовать определенному формату сообщений. Данная тема дополнительная и не обязательна для простых служб, поэтому мы не будем рассматривать данный тип контракта в текущей главе.

Эти контракты определяются с помощью специальных атрибутов, применяемых к классам и интерфейсам службы.

В настоящей главе мы рассмотрим пример создания службы WCF, предоставляющей клиентским приложениям данные из базы данных `AdventureWorksLT`. Для того чтобы сделать это, вы предоставите операции для работы с клиентскими данными, которые, в свою очередь, предоставят данные о клиенте, находящиеся непосредственно в базе данных.

Начнем пример с нового проекта. Удалите файлы `IService1` (`.vb` или `.cs`) и `Service1` (`.vb` или `.cs`). С помощью шаблона элемента WCF Service добавьте в проект новый элемент `CustomerService`. В результате к вашему проекту будут добавлены два новых файла: `CustomerService` (`.vb` или `.cs`) и `ICustomerService` (`.vb` или `.cs`).



При разработке служб есть два главных аспекта. Можете избрать подход с предварительной реализацией (где вы сначала пишете код, а затем применяете к нему атрибуты для создания контракта) или сначала создать контракт (где вы сначала разрабатываете схему или документ WSDL и на их основе создаете код). Всестороннее обсуждение этих подходов выходит за рамки данной главы; однако платформа WCF способна обеспечить оба подхода. Пример в этой главе следует подходу предварительного контракта.

Создание контракта службы

Сначала займемся определением контракта службы. Ниже приведены операции, которые вы хотите предоставить для внешнего окружения.

- `AddCustomer`
- `GetCustomer`
- `UpdateCustomer`
- `DeleteCustomer`
- `GetCustomerList`

Как можно заметить, первые четыре операции — это стандартные операции CRUD (Create, Read, Update и Delete — создание, чтение, модификация и удаление) для работы с данными. Последняя операция возвратит список всех клиентов в базе данных.

Теперь, когда вы знаете, какие операции нужны, можете определить свой контракт службы.



Возможно, вы обратили внимание на то, что в типовой реализации шаблона проекта WCF все атрибуты службы были определены в интерфейсе. Но создание интерфейса для установки атрибутов контракта не является обязательным — фактически вы не обязаны создавать интерфейс вообще и можете вместо этого установить атрибуты непосредственно в классе. Однако стандартная практика (и рекомендуемая) диктует, что контракт должен быть определен как интерфейс и внутри интерфейса. Поэтому будем следовать в примере общепринятой практике.

Определите свои операции в интерфейсе `ICustomerService`. Но эти операции предоставят данные с помощью класса данных, который вы пока еще не определили, поэтому временно создайте заглушку класса данных, — вскоре мы сможем его конкретизировать. Добавьте в проект новый класс `CustomerData` и оставьте его как есть, поскольку он должен действовать как заглушка. Все операции должны быть помечены атрибутом `OperationContract`.

VB

```
<ServiceContract([Namespace]:="http://www.professionalvisualstudio.com")>
Public Interface ICustomerService
    <OperationContract()>
    Function AddCustomer(ByVal customer As CustomerData) As Integer

    <OperationContract()>
    Function GetCustomer(ByVal customerID As Integer) As CustomerData

    <OperationContract()>
    Sub UpdateCustomer(ByVal customer As CustomerData)

    <OperationContract()>
    Sub DeleteCustomer(ByVal customerID As Integer)

    <OperationContract()>
    Function GetCustomerList() As List(Of CustomerData)
End Interface
```

C#

```
[ServiceContract(Namespace="http://www.professionalvisualstudio.com")]
public interface ICustomerService
{
    [OperationContract]
    int AddCustomer(CustomerData customer);

    [OperationContract]
    CustomerData GetCustomer(int customerID);

    [OperationContract]
    void UpdateCustomer(CustomerData customer);

    [OperationContract]
    void DeleteCustomer(int customerID);

    [OperationContract]
    List<CustomerData> GetCustomerList();
}
```

У атрибутов `ServiceContract` и `OperationContract` есть множество свойств, к которым вы можете применять значения и изменять их стандартное поведение. Например, у обоих атрибутов есть такое свойство, как имя (позволяющее определить название службы или операции, как указано внешне). Обратите, в частности, внимание на свойство `Namespace` атрибута `ServiceContract`, которое вы всегда должны определять явно (как было сделано в приведенном выше коде). Если пространство имен не было указано явно, схема и документ WSDL, созданный для службы, будут использовать его как пространство имен `http://tempuri.org`. Но чтобы снизить шанс конфликтов с другими службами, лучше использовать нечто уникальное, такое как URL вашей компании.

Теперь, когда вы определили контракт, необходимо фактически реализовать эти операции. Откройте класс `CustomerService`, который реализует интерфейс `ICustomerService`. Язык VB реализует методы автоматически (вам, возможно, придется нажать клавишу <Enter> , чтобы фактически реализовать интерфейс), а в языке C# для автоматической реализации методов можете использовать смарт-тег (<Ctrl+.>). Контракт службы теперь закончен и готов для реализации операцией (т.е. написания кода, который выполняет каждую операцию). Но, прежде чем вы сделаете это, необходимо определить свойства класса данных, а также контракт данных.

Создание контракта данных

Из некоторых предоставляемых службой операций вы возвращаете объекты, содержащие данные, и передаете им объекты как параметры. Поэтому вы должны определить структуру этих передаваемых объектов данных и снабдить их классы атрибутами контракта данных.



Начиная с версии .NET Framework 3.5 SP1 больше не обязательно явно определять контракт для классов данных, если классы являются открытыми и у каждого есть стандартный конструктор (это называется наличием *ожидаемого контракта данных* (inferred data contract), в отличие от *формального контракта данных* (formal data contract)). Однако имеет смысл (и рекомендуется), так или иначе, создавать формальный контракт, особенно если необходимо соответствие определенному формату сообщений, или при обращении к вашей службе клиентов, отличных от клиентов .NET, или при необходимости явно определить свойства в классе данных, включаемые в сообщение. Поскольку явное определение контракта данных рекомендуется в большинстве случаев, этот подход мы и рассмотрим в примере.

Данный пример потребует только одного класса данных, `CustomerData`, который вы уже создали (хотя никакие его свойства пока еще не определены) и теперь снабдите атрибутами контракта данных. Поскольку атрибуты контракта службы находятся в пространстве имен `System.ServiceModel`, а атрибуты контракта данных — в пространстве имен `System.Runtime.Serialization`, разработчики на языке C# должны будут начать класс с добавления оператора `using` для этого пространства имен.

```
using System.Runtime.Serialization;
```

Каждый класс данных должен быть сначала помечен атрибутом `DataContract`, а затем можете пометить каждое свойство, подлежащее сериализации, атрибутом `DataMember`.

VB

```
<DataContract([Namespace]:="http://www.professionalvisualstudio.com")>
Public Class CustomerData
    <DataMember()> Public Property CustomerID As Integer
    <DataMember()> Public Property Title As String
    <DataMember()> Public Property FirstName As String
    <DataMember()> Public Property MiddleName As String
    <DataMember()> Public Property LastName As String
    <DataMember()> Public Property Suffix As String
    <DataMember()> Public Property CompanyName As String
    <DataMember()> Public Property EmailAddress As String
    <DataMember()> Public Property Phone As String
End Class
```

C#

```
[DataContract(Namespace="http://www.professionalvisualstudio.com")]
public class CustomerData
{
    [DataMember] public int CustomerID { get; set; }
    [DataMember] public string Title { get; set; }
    [DataMember] public string FirstName { get; set; }
    [DataMember] public string MiddleName { get; set; }
    [DataMember] public string LastName { get; set; }
    [DataMember] public string Suffix { get; set; }
```

```
    [DataMember] public string CompanyName { get; set; }
    [DataMember] public string EmailAddress { get; set; }
    [DataMember] public string Phone { get; set; }
}
```

Если сериализация свойства не нужна, просто не применяйте к нему атрибут `DataMember`. Подобно атрибутам контракта службы, вы можете также установить значение каждого из свойств, которыми обладает каждый атрибут. Например, атрибут `DataContract` позволяет установить такие свойства, как пространство имен для контракта данных класса (свойство `Namespace`) и альтернативное имя для контракта данных класса (свойство `Name`). У атрибута `DataMember` также есть много свойств, которые вы можете установить, включая название элемента (свойство `Name`) и обязан ли элемент иметь заданное значение (свойство `IsRequired`).



У вас может возникнуть вопрос: почему при определении контракта данных вы помечаете сами классы данных, а не определяете контракт в интерфейсе, как контракт службы (что считалось хорошей практикой). Дело в том, что сериализироваться могут только конкретные типы, а интерфейсы не могут (следовательно, они не могут быть определены как параметры или типы возвращаемого значения в вызовах WCF). Когда тип объекта определяет только интерфейс, при десериализации не будет известно, объект какого конкретного типа следует создать. Способ обойти эту проблему существует, но он не рассматривается в данной главе. Обратите внимание на то, что если вы попытаетесь создавать интерфейс и отметить его атрибутом `DataContract`, то это приведет к ошибке во время компиляции.

Вам следует знать о некоторых проблемах при разработке контрактов данных. Если ваш класс данных наследуется от другого класса, который не отмечен атрибутом `DataContract`, то вы получите ошибку при попытке запуска службы. Поэтому вы должны либо отметить наследуемый класс атрибутами контракта данных, либо удалить атрибуты контракта данных из класса данных (хотя это и не рекомендуется), таким образом, вместо этого ожидается контракт данных.

Если вы решите использовать ожидаемый контракт данных и не отмечать классы данных атрибутами контракта данных, то все открытые свойства будут сериализироваться. Но по необходимости можете исключить свойства из состава сериализируемых, — достаточно отметить их атрибутом `IgnoreDataMember`. Недостатком ожидаемых контрактов данных является то, что классы данных должны иметь стандартный конструктор (т.е. конструктор без параметров) или не иметь никаких конструкторов вообще (тогда для него компилятор сам создаст стандартный конструктор). Если у вас в классе данных с ожидаемым контрактом не будет стандартного конструктора, то вы получите ошибку, когда попытаетесь запустить службу. Обратите внимание на то, что когда объект этого типа передается как параметр операции, при создании объекта будет вызван стандартный конструктор и выполнен любой находящийся в нем код.



Хотя это и не строго обязательно, лучше хранить классы контракта данных отдельно от других классов приложения и использовать их только для передачи данных в службы и из них (как *объекты передачи данных* (Data Transfer Objects — DTO)). Таким образом, вы минимизируете зависимости между своим приложением и службами, которые оно предоставляет или вызывает.

Настройка конечных точек службы WCF

У службы WCF есть три основных компонента: адрес, привязка и контракт (легко запомнить: Address, Binding и Contract — ABC).

- Адрес определяет расположение службы в форме URL.
- Привязка определяет протокол и кодировку символов, используемых для связи (и как).
- Контракт детализирует возможности и средства службы (и какой).

Конфигурации каждого из этих компонентов объединяются, чтобы сформировать конечную точку. Каждая комбинация этих компонентов формирует отдельную конечную точку, хотя может быть проще рассмотреть это как наличие нескольких конечных точек у каждой службы (т.е. комбинаций адресов и привязок). Мощь платформы WCF как раз и обусловлена тем, что она абстрагирует эти компоненты от реализации службы, позволяя настраивать их согласно технологиям, которые поддержит служба.

Но эта мощь добавляет сложности настройки конечных точек и может сделать ее довольно тяжелой. В частности, из-за множества различных типов поддерживаемых привязок с огромным количеством параметров каждой. На платформе WCF 4.0 конфигурация проще, чем у предыдущих версий, за счет поддержки стандартных конечных точек, стандартных сопоставлений протоколов, стандартных конфигураций привязок и стандартных конфигураций режимов, — все они снижают объем необходимых настроек. Поскольку конфигурация конечной точки может стать очень сложной, эта глава сосредоточена только на наиболее распространенных требованиях.

Конечные точки для службы определяются в файле `App.config`. Хотя вы можете открыть файл `App.config` и отредактировать его вручную, в состав среды разработки Visual Studio входит инструмент редактирования файлов конфигурации, существенно упрощающий процесс настройки. Щелкните правой кнопкой мыши на файле `App.config` в окне Solution Explorer и в появившемся контекстном меню выберите команду Edit WCF Configuration, чтобы открыть окно Microsoft Service Configuration Editor, показанное на рис. 31.2.

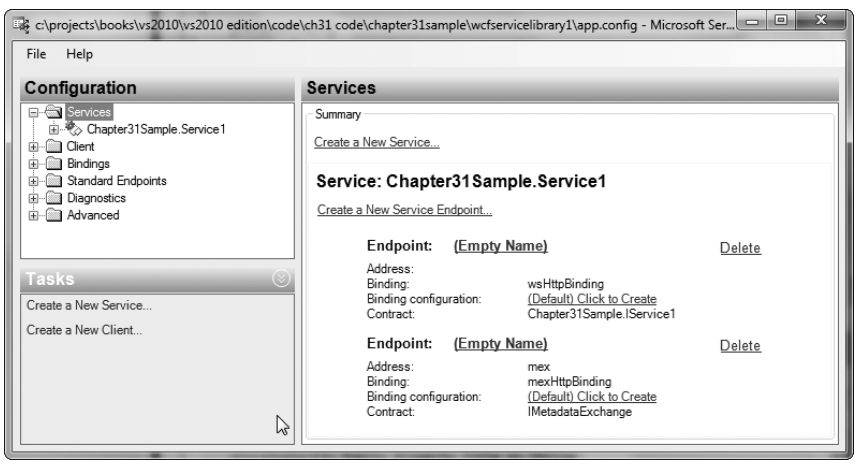


Рис. 31.2

Наибольший интерес представляет узел Services, после выбора которого в панели Services отображается отчет обо всех настроенных службах и их соответствующих конечных точках. Вы найдете, что служба уже здесь перечислена, хотя это конфигурация для стандартной службы, созданной шаблоном проекта (`Service1`), который больше не существует. Поэтому можете удалить эту службу из конфигурации и запустить новую (щелкните на службе и нажмите клавишу <Delete>).



Если вы попробуете запустить службу (подробности — в следующем разделе), без правильной настройки конечной точки для нее (или с неправильным именем службы в конфигурации), то получите ошибку, сообщающую, что WCF Service Host не может найти метаданные службы. Если вы получаете эту ошибку, проверьте имя службы (включая ее пространство имен) в конфигурации, — оно должно соответствовать своему имени в фактической реализации службы.

Первый шаг — определить вашу службу в конфигурации. На панели задач щелкните на гиперссылке Create a New Service, чтобы запустить мастер нового элемента службы New Service Element Wizard. В поле типа службы можете непосредственно ввести полностью квалифицированное имя вашей службы (т.е. включая ее пространство имен) или щелкнуть на кнопке Browse (Просмотр), чтобы найти доступные службы (лучше использовать функцию Browse, поскольку это автоматически заполняет поля на следующем этапе). Если вы используете эту возможность, то должны сначала откомпилировать свой проект, а затем перейти к папке `bin\Debug`, чтобы найти сборку и отобразить находящиеся в ней службы (рис. 31.3). Итак, вы определили реализацию службы, теперь необходимо определить контракт, привязку и адрес для конечной точки.

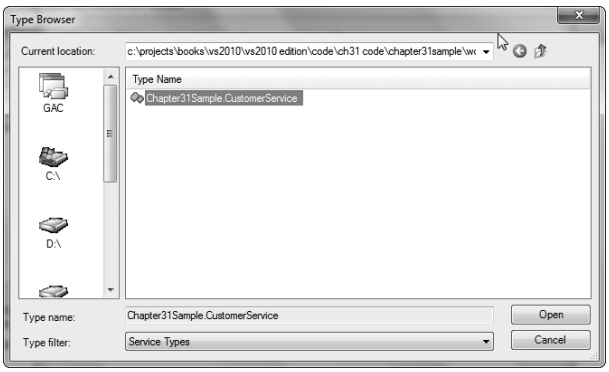


Рис. 31.3

Если на предыдущем этапе вы использовали кнопку Browse (как рекомендуется), на следующем этапе (определение контракта службы) все уже будет заполнено автоматически (рис. 31.4). В противном случае преступайте к заполнению.

Следующий этап позволяет создать новую конфигурацию привязки или использовать существующую, как показано на рис. 31.5. Однако первая возможность, вероятно, немного вводит в заблуждение, поскольку она не создает новую конфигурацию привязки, а помогает выбирать (с использованием мастера), какую из стандартных конфигураций привязки вы хотите использовать для конечной точки. У каждой привязки есть стандартная конфигурация, но для нее могут быть созданы дополнительные конфигурации (в узле Bindings дерева Configuration), что позволяет подробно настроить режим привязки. При несметном количестве доступных параметров специальная конфигурация привязок может стать довольно сложной. Однако в большинстве случаев оказывается, что вы нуждаетесь только в стандартной привязке (если только у

вас не будет фактической потребности в изменении ее режима). В этой главе подразумевается, что для наших потребностей вполне достаточно стандартной привязки.



Рис. 31.4

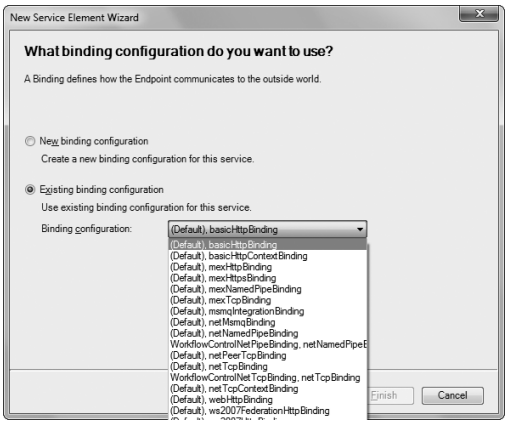


Рис. 31.5

Фактически имеющиеся на этом экране возможности (несмотря на отображаемый текст) позволяют либо запустить другую часть мастера, который поможет решить, какая из стандартных конфигураций привязки необходима (первая возможность), или выбрать из списка существующих конфигураций привязки (т.е. стандартная или специальная конфигурация привязки), если вы знаете, какая необходима (вторая возможность).

На самом деле выбор используемой привязки зависит от вашего сценария применения службы. Мастер поможет вам выбрать привязку, с подробным описанием каждой возможности и ее цели. Но вы должны помнить, что не все клиенты способны поддерживать выбранную привязку, поэтому должны также учитывать, какие клиенты будут использовать вашу службу и выбирать привязку соответственно. Конечно, вы сможете добавить дополнительные конечные точки с различными привязками, чтобы поддержать клиентов каждого типа. Наиболее распространенные привязки — это `basicHttpBinding` и `wsHttpBinding`, обе со связью по протоколу HTTP. Привязка `basicHttpBinding` используется для связи тем же способом, что и у веб-служб ASMX (это соответствует спецификации WS-I Basic Profile 1.1). Привязка `wsHttpBinding` реализует множество дополнительных спецификаций, в отличие от привязки `basicHttpBinding` (включая спецификации надежности и безопасности), а также такие дополнительные возможности, как поддержка транзакций. Однако устаревшие клиенты платформы .NET (до версии .NET Framework 3.0), клиенты, отличные от клиентов платформы .NET, мобильные клиенты и клиенты платформы Silverlight не смогут получить доступ к службе, используя эту привязку. Для данного примера выберем привязку `wsHttpBinding`.

Заключительный этап подразумевает определение адреса конечной точки. Вводя только название, можете определить полный адрес, который будет использоваться с протоколом (такой как `http://`), или определить относительный адрес (относительно базового адреса), как обсуждается далее. В данном случае удалите стандартный элемент и оставьте пустое место, — эта конечная точка будет просто использовать ба-

зовый адрес, который вы собираетесь установить. По завершении этого этапа отобразится предупреждение, но его можно игнорировать.

В заключение на экран будет выведен отчет о конфигурации конечной точки, и вы можете закрыть мастер. Этот мастер позволил вам создать одну конечную точку для службы, но, вероятно, вам понадобится реализовать несколько конечных точек. Это можно легко сделать с помощью мастера нового элемента конечной точки службы New Service Endpoint Element Wizard и создать дополнительные конечные точки. Под созданным узлом службы будет узел конечных точек. Выберите его, а затем щелкните в панели задач на гиперссылке Create a New Service Endpoint, чтобы запустить мастер, который поможет создать новую конечную точку.

Как уже упоминалось, теперь вы должны настроить для конечной точки базовый адрес. Поскольку вы выбрали привязку `wsHttpBinding`, необходимо использовать стандартный URL протокола HTTP, который сделает службу доступной. Под недавно созданным узлом службы находится узел Host. Выберите его и в появившейся панели Host щелкните на кнопке New, чтобы добавить новый базовый адрес в список (который в настоящее время пуст). Открывшееся диалоговое окно, запрашивающее базовый адрес, содержит стандартную запись. Адрес, который вы вводите здесь, будет в значительной степени зависеть от привязки, которая была выбрана ранее. Поскольку вы выбрали одну из привязок протокола HTTP, используйте для данного примера базовый адрес `http://localhost:8733/Chapter31Sample` (порт 8733 был выбран наугад).

Теперь ваша служба настроена с конечными точками, которые она поддержит. Есть и другая тема, связанная с конфигурацией служб, которую стоит затронуть, — это режим. В основном режимы WCF изменяют работу службы или конечной точки. Режим службы, содержащий два элемента, уже был настроен для службы шаблоном проекта. Если вы развернете узел Advanced и выберете в нем узел Service Behaviors, то обнаружите, что были определены режимы расширения элемента `serviceMetadata` и `serviceDebug`. Режим расширения элемента `serviceMetadata` позволяет публиковать метаданные службы. Чтобы вашу службу можно было обнаружить и добавить ссылку на нее в проект клиента (т.е. создать прокси-сервер), служба должна публиковать свои метаданные. С привязкой `mexHttpBinding` вы могли бы настроить это как отдельную конечную точку, но этот режим объединит привязку со службой без необходимости явной настройки самой службы. Это позволит обнаружить все ваши службы. Щелчок на узле serviceMetadata в дереве представит все его свойства; удостоверьтесь, что для свойств `HttpGetEnabled` и `HttpsGetEnabled` установлено значение `True`. Другой элемент режима, `serviceDebug`, — это расширение режима. При отладке вашей службы может быть полезно отображать в браузере справочную страницу, когда вы переходите к ней (по существу, публикация его документа WSDL по URL протокола HTTP). Для этого следует установить для свойств `HttpHelpPageEnabled` и `HttpsHelpPageEnabled` значение `True`. Во время отладки имеет смысл установить в состояние `True` еще одно свойство, `IncludeExceptionDetailsInFaults`, чтобы иметь возможность просмотреть стек трассировки, если произошло исключение. Хотя при отладке этот режим очень полезен, в целях безопасности перед развертыванием службы его рекомендуется отменить.

Хостинг служб WCF

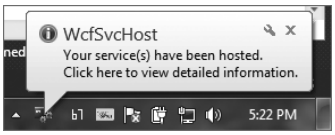


Рис. 31.6

Теперь, когда изменения внесены, можно откомпилировать и запустить проект WCF Service Library. В отличие от библиотеки стандартного класса, библиотека WCF Service Library может быть "запущена", поскольку среда разработки Visual Studio 2010 связана с утилитой WCF Service Host. Это приложение обеспечивает хостинг служб WCF при их отладке. На рис. 31.6 показано, что эта утилита доступна в панели задач.

Как свидетельствует всплывающая подсказка на рис. 31.6, щелчок на подсказке или пиктограмме панели задач приведет к открытию диалогового окна, отображающего подробную информацию о выполняющейся службе. Если служба запущена неправильно, это диалоговое окно может помочь выяснить, что идет не так.



Если запуск выполняется без соответствующих прав, можете получить от хоста службы WCF сообщение об ошибке регистрации URL, который определили в файле конфигурации. Проблема — в политике безопасности компьютера, которая препятствует регистрации URL хоста службы WCF, который вы определили. Если вы получаете это сообщение об ошибке, то можете решить проблему, выполнив в приглашении командной строки приведенные ниже команды (т.е. осуществить запуск с правами администратор), заменив параметры согласно адресу службы и вашему имени пользователя Windows.

```
netsh http add urlacl url=http://+:8733/Chapter31Sample
user=<username>
```

Эта команда позволит конкретному пользователю зарегистрировать URL, которому соответствует префикс `url`. Теперь, когда вы снова попытаетесь запускать свою библиотеку WCF Service Library, она должна заработать успешно.

Как можно заметить на рис. 31.7, кроме хоста вашей службы WCF, среда разработки Visual Studio 2010 запускает также утилиту WCF Test Client. Эта утилита автоматически обнаруживает выполняющиеся службы и отображает простое древовидное представление служб и их операций.

Если вы дважды щелкнете на операции службы, вкладка на правой стороне диалогового окна изменится на представление значений запросов и ответов. В отличие от простой справочной страницы веб-службы ASP.NET, клиент WCF Test Client способен помочь вам моделировать обращения к службе WCF, содержащие сложные типы. Как можно заметить на рис. 31.7, в разделе Request отображается каждый параметр, а параметр операции `AddCustomer` из объекта `customer` был раскрыт вместе с полями данных для каждого из его свойств (которые были отмечены атрибутом `DataMember`). После установки значений для каждого из этих свойств можете вызвать операцию, щелкнув на кнопке Invoke. Любое возвращаемое значение (рис. 31.8) также отображается со всеми подробностями в разделе Response.

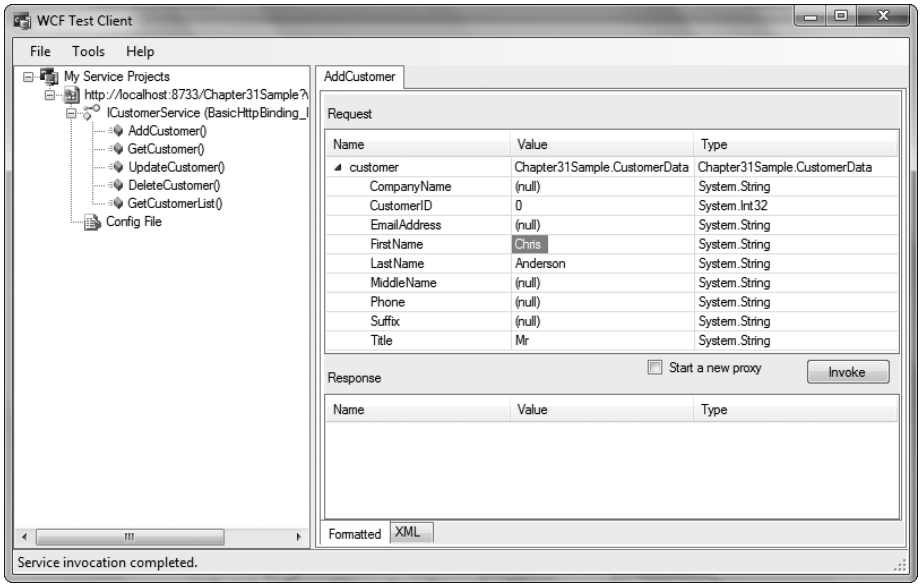


Рис. 31.7

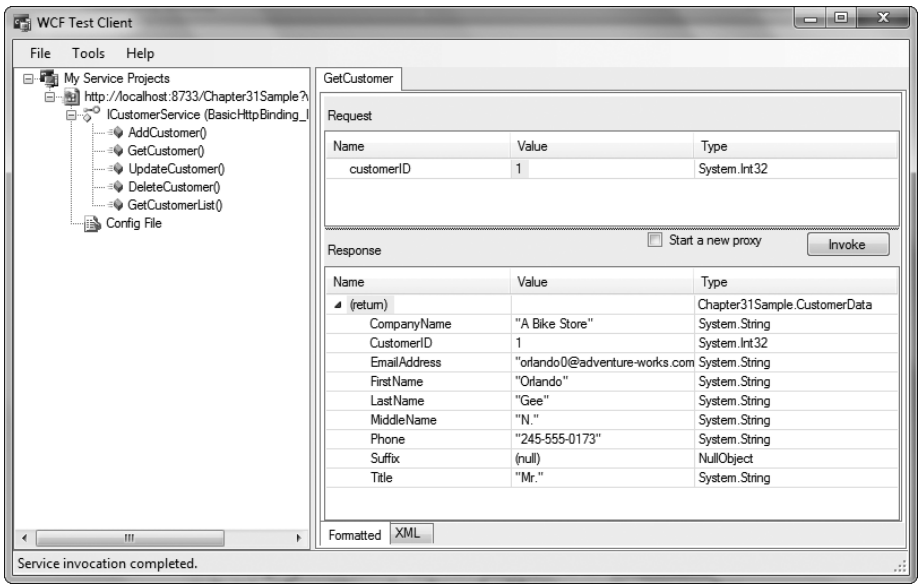


Рис. 31.8

Если вы пытаетесь изолировать проблему, то будет очень полезно просматривать, какая именно информация поступает в каждом запросе к службе. Для этого можете использовать инструменты стороннего производителя, такие как прокси Fiddler, но для простого представления передаваемой и принимаемой информации в формате XML достаточно перейти на вкладку XML. На рис. 31.9 представлено тело запроса и ответа XML. Обратите внимание на наличие дополнительного кода XML в каждом запросе и ответе, — это оболочка SOAP.

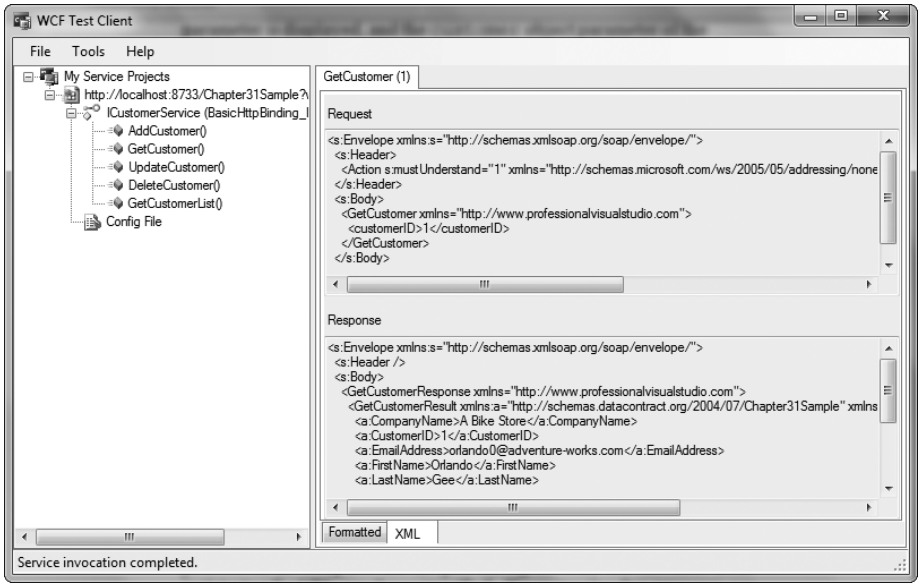


Рис. 31.9

Это прекрасно при отладке службы, но для эксплуатации необходим настоящий хостинг службы. Для хостинга службы есть много возможностей, и ваш выбор в действительности зависит от вашей ситуации. Если служба действует как сервер (к которому обращаются клиенты) и общение идет по протоколу HTTP, то наилучшим выбором, вероятно, будет сервер *информационных служб Интернета* (Internet Information Services — IIS). Если служба используется для связи между двумя приложениями, то хостом службы имеет смысл сделать само приложение. Можно рассмотреть и другие возможности, например хостинг на службе Windows Service или (если машина хоста имеет операционную систему Windows Vista/7 или Windows Server 2008) *службу процесса активизации Windows* (Windows Process Activation Services — WAS). Мы рассмотрим два наиболее распространенных случая: хостинг службы на сервере IIS и хостинг на приложении .NET (которое будет консольным приложением).

В первом примере рассмотрим хостинг службы WCF на сервере IIS. Сначала необходимо установить папку и обязательные файлы. Создайте новую папку (в папке `wwwroot` вашего сервера IIS или где-нибудь по вашему выбору) и дайте ей имя по своему выбору, а также создайте другую папку в ней по имени `bin`. Скопируйте в эту папку откомпилированную сборку службы, т.е. файл `.dll`. Скопируйте также файл `App.config` в папку на один уровень выше, т.е. в первую созданную папку, и переименуйте его в `web.config`.

Теперь необходимо создать простой текстовый файл (в интегрированной среде разработки Visual Studio, в приложении Note или любом текстовом редакторе по вашему выбору) и назвать его `CustomerService.svc` (это может быть любое название, но расширение `.svc` действительно обязательно). Поместите в файл строку

```
<%@ServiceHost Service="Chapter31SampleCS.CustomerService"%>
```

По существу, это указывает, что сервер IIS должен содержать службу `Chapter31SampleCS.CustomerService` (которую он ожидает найти в одной из сборок, находящихся в папке `bin`).

Таким образом, в папке у вас должны быть файлы `CustomerService.svc` и `web.config`, а во внутренней папке `bin` — сборка службы (`.dll`). Удостоверьтесь (в правах папки), что у процесса IIS есть доступ к этой папке на чтение.

А теперь необходимо настроить службу на сервере IIS. Откройте сервер IIS, в стандартном веб-сайте добавьте новое приложение. Дайте ему имя (например, `CustomerService`) и определите физический путь как созданную ранее папку. Удостоверьтесь также в том, что выбрано использование пула приложений ASP.NET v4.0 (таким образом, будет использована платформа .NET Framework версии v4.0), и это все!

Затем можете перейти в браузере по URL службы и посмотреть, работает ли она, а также использовать клиент WCF Test Client для фактической проверки операций.



Если вы создадите проект, используя шаблон проекта WCF Service Application, то правильная структура и необходимые файлы будут автоматически созданы и готовы для хостинга на сервере IIS.

В другом примере рассмотрим хостинг службы WCF в приложении .NET (известный также как локальный хостинг службы). Вы можете поместить в этот проект либо непосредственно код службы (созданный ранее), либо ссылку на проект службы, который создали ранее. Для этого примера создайте простое консольное приложение, действующее как хост, и ссылку на существующий проект службы. Создайте в среде разработки Visual Studio новый проект консольного приложения, `CustomerServiceHost`, и добавьте ссылку на проект службы. Придется также добавить ссылку на сборку `System.ServiceModel`. Скопируйте в этот проект файл `App.config` из проекта службы (так вы сможете использовать установленную ранее конфигурацию службы).

Используйте для хостинга службы следующий код.

VB

```
Imports System.ServiceModel
Imports Chapter31SampleVB

Module CustomerServiceHost
    Sub Main()
        Using svcHost As New ServiceHost(GetType(CustomerService))
            Try
                'Открыть службу и закрыть ее снова, когда пользователь
                ' нажимает клавишу
                svcHost.Open()

                Console.WriteLine("The service is running...")
                Console.ReadLine()

                'Закрыть ServiceHost.
                svcHost.Close()

            Catch ex As Exception
                Console.WriteLine(ex.Message)
                Console.ReadLine()
            End Try
        End Using
    End Sub
End Module
```

C#

```
using System;
using System.ServiceModel;
using Chapter31SampleCS;
```

```
namespace CustomerServiceHost
{
    class Program
    {
        static void Main(string[] args)
        {
            using (ServiceHost serviceHost =
                    new ServiceHost(typeof(CustomerService)))
            {
                try
                {
                    // Открыть службу и закрыть ее снова, когда
                    // пользователь нажимает клавишу
                    serviceHost.Open();

                    Console.WriteLine("The service is running...");
                    Console.ReadLine();

                    serviceHost.Close();
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.Message);
                    Console.ReadLine();
                }
            }
        }
    }
}
```

В итоге конфигурация службы читается из файла `.config` (хотя она может быть также определена программно). Таким образом, необходимо лишь создать объект хоста службы (передав ему тип службы, которая будет содержаться) и открыть хост. Когда все будет сделано, остается лишь закрыть и очистить хост!

Запустите проект и обратитесь к службе, используя URL, определенный в файле `.config`. Как можно заметить, для хостинга службы WCF нужно совсем немного кода.

Использование службы WCF

Вы успешно создали свою службу WCF, и настало время получить к ней доступ из приложения. Для этого добавьте в свое решение проект Windows Forms `CustomerServiceClient`.

Затем добавьте к приложению Windows Forms ссылку на службу WCF. Щелкните правой кнопкой мыши на узле проекта в проводнике решений и в появившемся контекстном меню выберите команду Add Service Reference. В открывшемся диалоговом окне, представленном на рис. 31.10, определите службу WCF, ссылку на которую необходимо добавить. Как можно заметить, здесь есть очень удобная кнопка Discover, которую можно использовать для быстрого поиска службы в пределах текущего решения.

Выберите в древовидном представлении служб узел `ICustomerService`, смените пространство имен на `CustomerServices` и щелкните на кнопке OK, чтобы закончить процесс. Следующий этап подразумевает создание формы, которая позволит отобразить и отредактировать данные службы. Поместите код общения со службой в код этой формы. Начните с добавления в начало кода оператора `using` (или `Imports`) для пространства имен службы.

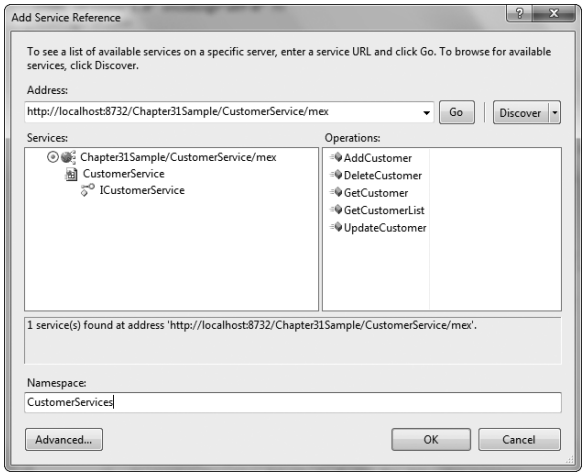


Рис. 31.10

VB

```
Imports CustomerServiceClient.CustomerServices
```

C#

```
using CustomerServiceClient.CustomerServices;
```

Предположим, вы имеете в форме элемент управления `BindingSource` по имени `customerDataBindingSource`, свойство `DataSource` которого должно содержать список клиентов, получаемый от службы. Все, что необходимо, — это создать экземпляр прокси-сервера службы и вызвать операцию, в результате данные будут возвращены.

VB

```
Dim service As New CustomerService
customerDataBindingSource.DataSource = service.GetCustomerList()
```

C#

```
CustomerService service = new CustomerService();
customerDataBindingSource.DataSource = service.GetCustomerList();
```

Можете запустить это приложение, и оно будет общаться со службой WCF. Данный пример продемонстрировал синхронную связь со службой WCF (т.е. поток UI приостанавливается, пока не будет получен ответ от сервера), но у этого подхода есть недостаток: приложение не отвечает пользователю, пока не будет получен ответ от службы. Хотя код синхронного вызова службы проще, он не предусматривает удобства для пользователя. К счастью, вы можете также вызывать службы WCF асинхронно. Это позволяет клиенту отправлять запрос службе и продолжать выполняться, не ожидая ответа. Когда ответ будет получен, произойдет событие, которое приложение может обработать и отреагировать на ответ.



Клиенты Silverlight поддерживают только асинхронные вызовы служб.

Для того чтобы позволить создавать на прокси-сервере службы асинхронные методы, необходимо специально запросить это, установив флажок Generate Asynchronous Operations в диалоговом окне Configure Service Reference (подробности — далее). Для

асинхронного вызова службы WCF создайте экземпляр службы, обработайте событие Completed для ассоциированной операции, а затем вызовите метод операции с суффиксом Async.

VB

```
Dim service As New CustomerService
AddHandler service.GetCustomerListCompleted, _
        AddressOf service_GetCustomerListCompleted
service.GetCustomerListAsync()
```

C#

```
CustomerService service = new CustomerService();
service.GetCustomerListCompleted += service_GetCustomerListCompleted;
service.GetCustomerListAsync();
```

Когда операция завершится, ответ на ее вызов возвратится немедленно и будет вызван соответствующий обработчик события. Данные, которые были возвращены службой, будут переданы в обработчик событий с помощью e.Result.

VB

```
Private Sub service_GetCustomerListCompleted(ByVal sender As Object, _
            ByVal e As GetCustomerListCompletedEventArgs)
    customerDataBindingSource.DataSource = e.Result
End Sub
```

C#

```
private void service_GetCustomerListCompleted(object sender,
        GetCustomerListCompletedEventArgs e)
{
    customerDataBindingSource.DataSource = e.Result;
}
```

Добавив ссылку на службу WCF в свое улучшенное клиентское приложение, вы увидите, что файл App.config был добавлен в проект (если он еще не существовал). В любом случае, если вы обращаете внимание на этот файл, то увидите, что теперь он содержит элемент system.serviceModel, который содержит привязки и клиентские элементы. В пределах элемента привязок, как можно заметить, есть элемент wsHttpBinding (это стандартная привязка WCF), определяющий способ общения со службой WCF. Здесь можно заметить элементы, переопределяющие некоторые из значений по умолчанию. Элемент Client содержит элемент конечной точки. В нем определены такие значения, как Address (которым в данном случае является URL), Binding (который ссылается на настроенную привязку wsHttpBinding, определенную в элементе привязок) и Contract (которым является интерфейс CustomerServices.ICustomerService вызываемой службы WCF). Поскольку вся эта информация определяется в файле конфигурации, при изменении любого из этих элементов (например, URL конечной точки) вы сможете модифицировать только файл конфигурации и избежать необходимости перекомпилировать все приложение.

После внесения изменений в службу понадобится модифицировать прокси-сервер службы, который был создан средой разработки Visual Studio при добавлении ссылки на службу в проект (в противном случае он останется устаревшим и не представит новые операции, добавленные в проект, и т.д.). Для этого щелкните правой кнопкой мыши на ссылке на службу (в узле ссылок на службу в вашем проекте) и выберите в появившемся контекстном меню команду Update Service Reference.

Если вы щелкнете правой кнопкой мыши на ссылке на службу (в узле ссылок на службу в вашем проекте), то обнаружите в появившемся контекстном меню команду Configure Service Reference. После его выбора откроется диалоговое окно, представленное на рис. 31.11 (оно открывается также в диалоговом окне Add Service Reference после щелчка на кнопке Advanced).

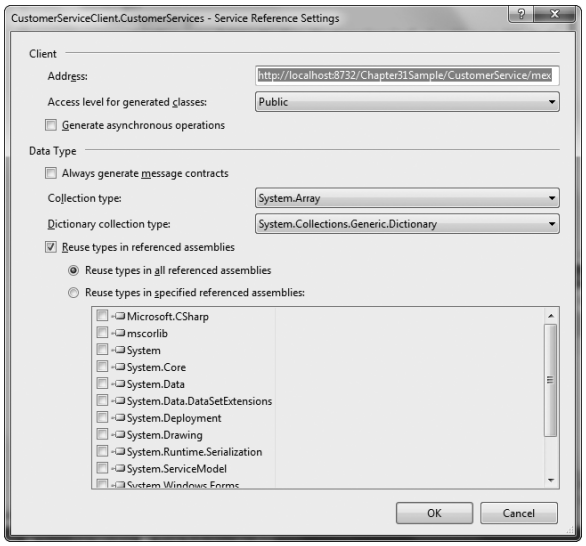


Рис. 31.11

Это диалоговое окно позволяет настроить созданный прокси-сервер службы с помощью множества доступных параметров. Особенно интересен флажок Reuse types in referenced assemblies. Если этот флажок установлен и генератор ссылки на службу выяснит, что тип (т.е. класс/объект), используемый или возвращаемый службой, определяется в сборке, на которую ссылается клиент, то создаваемый код прокси-сервера будет возвращать или получать объекты именно этого типа, а не будет создавать прокси-класс. Главное преимущество здесь в том, что вы управляете обеими частями системы (и сервером, и клиентом) и хотите передать между ними объекты, которые связывают бизнес-логику (такую как логика проверки правильности, бизнес-правила и т.д.). Потребительский процесс (на стороне клиента) копирует значения свойств из прокси-объекта в бизнес-объект (когда запрашивает данные), а затем значения свойств из бизнес-объекта в прокси-объект (чтобы возвратить данные на сервер). Но эта возможность означает, что у вас может быть как сервер и ссылка на сборку клиента, которая содержит типы, передаваемые между ними (с соответствующим кодом, реализующим бизнес-логику в обоих частях), так и просто передача объектов между сервером и клиентом, без прокси-класса в качестве посредника (на клиентской стороне). Это избавляет от необходимости писать большое количество кода для сопоставления множества свойств, что затрудняет обслуживание и повышает вероятность ошибок.

Резюме

В этой главе вы узнали, как создать службу WCF, обеспечить ее хостинг и настройку в различных случаях использования. Однако платформа WCF — это не конец истории об уровнях связи; фактически на основе платформы WCF построено множество технологий, расширяющих ее возможности. К ним относятся службы WCF Data Services и WCF RIA Services, которые подробно описаны в главе 35.

32

Платформа Windows Workflow Foundation (WF)

В ЭТОЙ ГЛАВЕ...

- Понятие Windows Workflow Foundation
- Создание простого рабочего потока
- Хостинг и выполнение рабочего потока
- Хостинг проектировщика рабочего потока в вашем приложении

Windows Workflow Foundation (WF) — это мощнейшая платформа для разработки и выполнения рабочих потоков, основа множества бизнес-приложений. Платформа WF впервые появилась в инфраструктуре .NET Framework 3.0 и была полностью перепроектирована и переделана для инфраструктуры .NET Framework версии 4.0, чтобы преодолеть некоторые из проблем, которые имелись в ее предыдущих воплощениях. К сожалению, это сделало ее несовместимой с рабочими потоками, созданными для предыдущих версий, но в результате получилась намного более надежная технология. В этой главе описано использование проектировщика WF, а также процесс создания и запуска рабочих потоков с использованием платформы WF.

Что такое Windows Workflow Foundation

Прежде чем переходить к обсуждению платформы Windows Workflow Foundation, сначала выясним, что такое рабочий поток. *Рабочий поток* (workflow) — это, по существу, модель этапов, которые формируют бизнес-процесс. Например, он может включать утверждение одобрения документа, отслеживание состояния работ и т.д.

Хорошо разработанный рабочий поток требует четкого разделения между этапами бизнес-процесса (работа, которую следует сделать) и бизнес-правил или логики, которая связывает их (поток).

Платформа Windows Workflow — это полное решение для рабочего потока, включая проект и компоненты времени выполнения, необходимые для разработки, создания и запуска рабочих потоков. Затем эти рабочие потоки могут располагаться в приложении или предоставляться вовне как служба.



Одно из главных преимуществ платформы WF в том, что вы можете содержать в своем приложении и среду выполнения WF, и проектировщик WF, позволяя конечным пользователям самим перенастроить рабочие потоки с помощью проектировщика WF, содержащегося в вашем приложении.

Использование платформы WF требует разделения бизнес-процесса на отдельные задачи (известные как действия), которые затем могут быть декларативно объединены и контролироваться в рабочем потоке с конфигурацией, перестраиваемой с использованием проектировщика WF.

Вы можете использовать платформу WF в собственных продуктах, но вы также найдете ее встроенной в различные продукты корпорации Microsoft, включая коллекцию программных продуктов и компонентов SharePoint и набор интегрированных технологий Windows Server AppFabric.

Зачем использовать рабочий поток Windows

Вот обычный вопрос, задаваемый теми, кто изучает платформу WF: зачем использовать ее, вместо того, чтобы встраивать бизнес-логику непосредственно в код? Это очень правильный вопрос, и ответ на него действительно зависит от бизнес-проблемы, которую вам предстоит решить, и бизнес-процесса, который необходимо моделировать. В этой главе рассматриваются некоторые из случаев, когда имеет смысл использовать платформу WF, но сначала рассмотрим преимущества, которые она может предоставить.

Один из основных случаев, когда вы извлекаете максимум преимуществ от использования платформы WF, — это бизнес-процесс, который изменяется часто (или часто изменяются правила в бизнес-процессе). В качестве альтернативы у вас может быть приложение, которое развертывается на разных клиентах, у каждого из которых есть различные бизнес-процессы. Бизнес-логика и правила, которые формируют рабочие потоки, на платформе WF определяются декларативно, а не встраиваются в код, что позволяет перенастраивать рабочий поток без необходимости перекомпилировать приложение. В сочетании со способностью содержать проектировщик WF в собственном приложении, это позволяет создавать приложения с очень быстро перестраиваемой конфигурацией.

Другой случай, когда использование платформы WF предоставляет много преимуществ, — это моделирование процессов, выполняющихся долго. Некоторые рабочие потоки могут выполниться от считанных секунд и минут до часов, дней и даже лет. Платформа WF служит основой для обслуживания этих долго выполняющихся процессов, обеспечивая возможность сохранения (персистентность) рабочего потока при ожидании события (чтобы не оставаться резидентным в памяти) и способность возобновления после перезапуска компьютера.

Преимущество разработки и визуализации рабочих потоков в проектировщике WF заключается в том, что диаграмма рабочего потока может использоваться как форма документации бизнес-процесса и логики. Эта диаграмма может быть экспортирована из проектировщика WF как изображение и использована в документации или представлениях. Это помогает обеспечить высокую наглядность моделируемых бизнес-процессов.

И наконец, нет смысла использовать платформу WF во всех приложениях, которые включают бизнес-процесс, требующий моделирования. Если любые из перечисленных выше преимуществ являются основными требованиями для вашего приложения, следует серьезно рассмотреть возможность разработки ваших рабочих потоков и действий с использованием платформы WF. Но если ни в одном из перечисленных преимуществ нет необходимости (ни сейчас, ни в будущем), то решение необходимо принимать на основании предпочтений вашей группы разработки и практических соображений: возможно применение такой инфраструктуры все же предоставит больше преимуществ, чем создаст потенциальных проблем.

Концепции рабочего потока

Прежде чем перейти к практическим аспектам разработки и выполнения рабочих потоков, рассмотрим некоторые из важнейших концепций рабочих потоков и относящуюся к ним терминологию.

Действия

Действие (activity) — это отдельный, дискретный модуль работ, т.е. решения задачи. Действие не обязано решать только одну задачу, фактически оно может содержать другие действия (называется *составным действием* (composite activity)), каждое из которых может содержать другие действия и т.д. *Рабочий поток* (workflow) сам является действием и как действие контролирует поток действий (эту тему мы обсудим чуть позже). Можете считать действие фундаментным строительным блоком рабочих потоков.

Действие может иметь входящие и выходящие аргументы, обеспечивающие ввод и вывод данных, а также возвращать значение. У действия могут также быть переменные, которые (как и в коде) хранят значение. Любые действия, которые содержит данное действие, также могут обращаться к значению этой переменной. Действие, в котором определяется переменная, определяет ее *область видимости* (scope).



Действия можно рассматривать как аналог метода в обычном коде.

В состав платформы WF входит базовая библиотека предопределенных действий, составляющих широкое разнообразие задач, которые вы можете использовать в своем рабочем потоке. Сюда относятся следующие действия.

- Контроль выполнения потока (`If`, `DoWhile`, `ForEach`, `Switch` и т.д.).
- Обеспечение функций обмена сообщениями (для общения со службами).
- Сохранение текущего экземпляра рабочего потока в базе данных.
- Поддержка транзакций.

- Управление коллекциями (добавление и удаление элементов коллекции, очистка коллекции и выяснение наличия элемента).
- Обработка ошибок (`try`, `catch`, `throw`, `rethrow`).
- Несколько основных функций (задержка, присвоение переменных, вывод на консоль и т.д.).

Конечно, несмотря на этот широкий диапазон предопределенных действий, вы, без сомнения, будете создавать собственные действия для решения своих задач, особенно если у вас будет сложная логика. Их реализуют в коде, они присутствуют в панели инструментов проектировщика WF, и вы можете перетаскивать их в свой рабочий поток.

При создании собственных специальных действий у вас есть выбор из множества специальных типов действий: `Activity`, `CodeActivity`, `NativeActivity` и `DynamicActivity` (специальное действие наследуется от одного из этих базовых классов).

Действия на базе класса `Activity` состоят из других действий и разрабатываются визуально в проектировщике WF. Как уже упоминалось, рабочие потоки — это тоже действия, таким образом, ваш рабочий поток фактически происходит от класса `Activity`. Действия, составленные этим способом, применяются также в других действиях.

Действие на базе класса `CodeActivity`, как и предполагает его название, является действием, у которого действия и логика определяются в коде. Это код, фактически класс, наследующий класс `CodeActivity` и переопределяющий метод `Execute`, в который должен быть помещен код, подлежащий выполнению.

Действия не обязаны выполняться синхронно, возможна блокировка продолжительного рабочего потока при выполнении долго выполняющейся задачи или при ожидании завершения операции, получения ответа или ввода пользователя, или события, которое должно произойти. Вы можете создавать асинхронные действия, наследуя класс `AsyncCodeActivity`. Он очень похож на класс `CodeActivity`, но вместо одного метода `Execute`, который должен быть переопределен, здесь есть методы `BeginExecute` и `EndExecute`. При запуске асинхронного действия его выполнение происходит в потоке, отдельном от планировщика, и завершается немедленно. Затем он может продолжить выполнение, не блокируя выполнение основного рабочего потока. Планировщик, который вызывал его, будет уведомлен о завершении выполнения.



Рабочий поток не может быть сохранен или загружен, пока выполняется асинхронное действие.

Действие на базе класса `NativeActivity` очень похоже на действие, унаследованное от класса `CodeActivity`, но в отличие от класса `CodeActivity`, ограниченного взаимодействием с аргументами и переменными, у класса `NativeActivity` есть полный доступ ко всем функциям, предоставленным средой выполнения рабочего потока (который передает ему объект `NativeActivityContext`). Это подразумевает способность планирования выполнения действия и его отмены, прерывая выполнения действия, доступа к закладкам действия, а также планирования и отслеживания функций.

Действия управления потоком

Действия управления потоком используются, по существу, для обеспечения связи между действиями, организации их в рабочий поток, а также формирования логики и правил моделируемого процесса. Действия управления потоком — это обычные стан-

дартные действия, но разработанные для того, чтобы контролировать выполнение потока действий, которые оно содержит (за счет планирования выполнения этих действий).

Есть два основных типа действий управления потоком (по существу, это типы рабочего потока): классы `Sequence` и `Flowchart`. Класс `Sequence` (Последовательность) выполняет действия, которые он содержит (как и предлагает его название), последовательно. Здесь невозможно вернуться к предыдущему шагу последовательности; выполнение может происходить только по порядку. Класс `Flowchart` (Блок-схема), напротив, позволяет вернуться к выполнению предыдущего шага, — это больше подходит для принятия решений процессами, чем последовательности.

Если вы работали с предыдущими версиями платформы WF, то, вероятно, заметите, что в ней больше нет действия управления потоком конечного автомата. Тем не менее в ближайшем будущем ожидается внеплановый выпуск действия управления потоком конечного автомата.

Вы не ограничены использованием одного действия управления потоком в рабочем потоке. Поскольку это тоже действия, можете использовать их совместно и подбирать их так, как нужно в конкретном рабочем потоке.

Выражения

Выражения (expression) — код на языке VB (и только VB), который возвращает значение и используется в проектировщике для контроля значений переменных и аргументов. Вы можете считать их подобием формул, скажем, приложения Excel. Выражения, как правило, связаны с входными аргументам действий, обычно устанавливают значение переменных или определяют условия в действиях (такие как действие `If`).

Среда выполнения рабочего потока или планировщик

Среда выполнения рабочего потока (run time), известная также как *планировщик* (scheduler), является процессором, который получает файл определения рабочего потока и выполняет его в контексте приложения хоста. *Приложение хоста* (host application) запускает данный рабочий поток в среде выполнения рабочего потока, используя класс `WorkflowInvoker`, `WorkflowApplication` или `WorkflowServiceHost`.

Класс `WorkflowInvoker` используется в "автоматическом" подходе к выполнению рабочего потока, предоставляя среде выполнения рабочего потока самой отработать все выполнение рабочего потока. Класс `WorkflowApplication` используется при необходимости "ручного" подхода к выполнению рабочего потока (такого как возобновление сохраненного экземпляра), позволяя контролировать выполнение хосту. Класс `WorkflowServiceHost` используется при хостинге рабочего потока как службы, которая будет использоваться клиентскими приложениями.

Закладки

Закладка (bookmark) отмечает место в рабочем потоке, с которого его выполнение может быть продолжено в более поздний момент времени. Закладки позволяют "приостановить" экземпляр рабочего потока, пока он ожидает получения ввода, определяя точку, с которой он будет возобновлен, когда этот ввод будет получен. Закладка предоставляет имя и определяет *функцию обратного вызова* (callback function),

точно определяя действие, которое выполняется в настоящее время, а также метод в действии, который должен быть вызван при возобновлении рабочего потока.

Создание закладки останавливает выполнение рабочего потока и освобождает пул рабочего потока (хотя рабочий поток не завершен, а просто приостановлен), позволяя сохранить и выгрузить рабочий поток. Затем хост получает ввод, который ожидает рабочий поток, и возобновляет выполнение экземпляра рабочего потока с позиции закладки (передавая все данные, полученные от ожидаемого ввода, методу обратного вызова).

Закладки особенно полезны в продолжительных процессах, где рабочий поток ожидает получения ввода, который потенциально может быть получен в течение достаточно длительного времени. Тем временем высвобождаются ресурсы, которые использует рабочий поток (они пригодятся для использования другими рабочими потоками), а его состояние может быть сохранено на диске (по необходимости).

Персистентность

Персистентность (persistence) позволяет сериализировать текущее состояние экземпляра рабочего потока и его метаданные (включая значения переменных в области видимости, аргументы, данные закладок и т.д.), а затем, с помощью провайдера персистентности, сохранить их в хранилище данных (известном как хранилище экземпляра), чтобы впоследствии возвратить их и возобновить рабочий поток. Чтобы сохранить экземпляр рабочего потока, его выполнение следует приостановить (как при ожидании ввода) и определить закладку, чтобы отметить текущий пункт выполнения рабочего потока.

Персистентность особенно важна, если у вас есть долго выполняющиеся рабочие потоки, если необходимо быть в состоянии выгружать ожидающие рабочие потоки или если компьютер сервера может перезапускаться во время ожидания рабочих потоков, или если выполнение способно переноситься на другой сервер (как в ферме серверов).



Обратите внимание на то, что сохраняется не сам экземпляр рабочего потока, а только его состояние. Необходимо знать о последствиях изменения рабочего потока, в то время как его экземпляры все еще существуют и сохраняются, и поступать соответственно.

Платформа WF поставляется со стандартным провайдером персистентности, `SqlWorkflowInstanceStore`, который обеспечивает сохранение экземпляра рабочего потока в базе данных SQL Server. Вы можете также создать собственный провайдер персистентности, наследуя от класса `InstanceStore`.

Имеются два способа сохранения экземпляра рабочего потока. Нужно использовать в рабочем потоке предопределенное действие `Persist` из панели инструментов, которое сохранит экземпляр рабочего потока во время выполнения. Вторая возможность — зарегистрировать на хосте обработчик события `PersistableIdle`, которое происходит при приостановке выполнения экземпляра рабочего потока (но еще не завершено). Затем хост может решить, следует ли сохранить экземпляр рабочего потока, возвращая значение из перечисления `PersistIdleAction`, которое укажет среде выполнения, что нужно сделать.

Отслеживание

Платформа WF позволяет реализовать отслеживание рабочих потоков и регистрировать различные аспекты выполнения рабочего потока для последующего анализа. Отслеживание обеспечивает просмотр рабочего потока, позволяя увидеть то, что было сделано в его прошлом и текущем состоянии, создавая записи отслеживания.

Настроив профиль отслеживания, который может быть определен либо в файле `App.config`, либо в коде, можете определить степень подробности записей отслеживания. Это позволит определить, какие именно записи отслеживания необходимо создавать во время выполнения рабочего потока. Можно фиксировать записи отслеживания о жизненном цикле рабочего потока (например, когда рабочий поток запускается или завершается), о жизненном цикле действия (например, когда действие планируется или завершается либо когда происходит ошибка), записи возобновления закладки и специальные записи отслеживания (которые вы можете создавать в своих специальных действиях). Эти записи отслеживания могут включать связанные данные, такие как текущие значения переменных и аргументов.

Место для записей отслеживания определяется в разделе отслеживания. По умолчанию среда выполнения WF сохраняет записи отслеживания в файле регистрации событий Windows. Можете создать собственные разделы отслеживания, если, например, хотите заносить записи отслеживания в другое хранилище, такое как база данных.

Можно также отслеживать выполнение рабочего потока в целях диагностики, для чего используются стандартные слушатели отслеживания платформы .NET. Отслеживание может быть настроено в файле `App.config`.

Первые шаги

Сначала откройте диалоговое окно New Project и перейдите к разделу Workflow предпочитаемого вами языка (рис. 32.1).

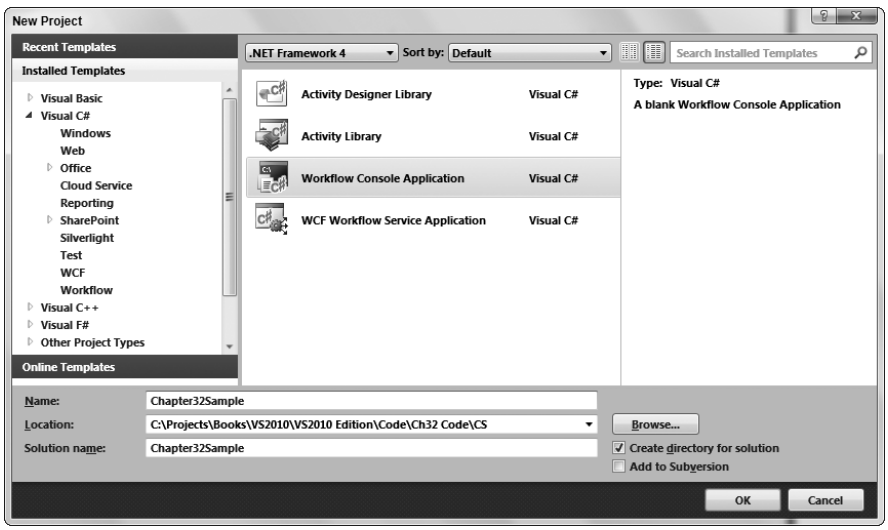


Рис. 32.1

Как можно заметить, здесь есть на выбор четыре типа проектов.

- Activity Designer Library. Создает библиотеку для многократного использования проектировщиков действий, а также настраивает вид и поведение действий в проектировщике WF.
- Activity Library. Создает проект библиотеки для многократного использования действий (состоящих из других действий), которую впоследствии можно использовать в собственных рабочих потоках. Считайте это библиотекой классов, но для рабочих потоков.
- Workflow Console Application. Создает пустой рабочий поток, расположенный в консольном приложении.
- WCF Workflow Service Application. Создает рабочий поток, хранящийся на компьютере и публикуемый как служба WCF.



Вы не ограничены хостингом рабочих потоков в консольном приложении или службе WCF, можете также располагать их на других платформах, таких как приложения Windows Forms, WPF или ASP.NET. Добавьте рабочий поток в существующий проект, используя диалоговое окно Add New Item и выбрав элемент Activity в разделе Workflow (здесь нет элемента Workflow, поскольку рабочий поток, по существу, это тоже действие, содержащее другие действия).

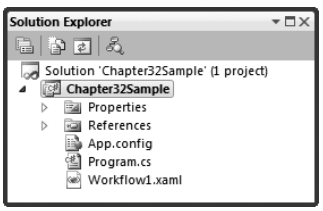


Рис. 32.2

Для примера проекта используем самую простую возможность получения и запуска, применив шаблон проекта Workflow Console Application. Как можно заметить на рис. 32.2, созданный проект очень прост, он содержит файлы `Program.cs/Module1.vb` и `Workflow1.xaml`. Класс `Program` (для языка C#) или модуль `Module1` (для языка VB), как в любом консольном приложении, содержит точку входа приложения (т.е. статический, совместно используемый метод `Main`), которая автоматически настраивается на создание экземпляра и запуск рабочего потока. Собственно рабочий поток определяется в файле `Workflow1.xaml`.



Файл рабочего потока имеет формат XAML, который вы можете распознать, поскольку он используется для определения пользовательских интерфейсов на платформах WPF и Silverlight. Но в данном случае он используется для декларативного определения рабочего потока. Можете просмотреть и отредактировать файл XAML для рабочего потока, щелкнув правой кнопкой мыши на файле и выбрав в появившемся контекстном меню команду View Code.

Прежде чем вы сделаете что-нибудь еще, откомпилируйте и запустите приложение как есть, чтобы увидеть результат. Вы должны заметить, что окно консоли появляется на миг и приложение автоматически завершается (поскольку в настоящее время оно ничего не делает).

Название `Workflow1.xaml` не является осмысленным, поэтому вы, без сомнения, захотите изменить его на нечто более подходящее. К сожалению, среда разработки

Visual Studio не особенно поможет в этом отношении (в отличие от форм и классов), поскольку изменение имени файла не изменит автоматически класс, созданный для рабочего потока, при этом он не изменит ссылок на класс, когда вы измените его имя в проектировщике. Например, чтобы переименовать рабочий поток и его соответствующий класс в `SimpleProcessWorkflow`, понадобится следующее.

- Изменить имя файла (в проводнике решений).
- Изменить имя соответствующего класса (щелкнув на рабочей области и присвоив имя в свойстве `Name` в окне Properties).
- Изменить все существующие ссылки на класс рабочего потока. В данном случае, пока вы еще ничего не сделали со своим проектом, единственная ссылка будет в классе `Program` (для языка C#) или в модуле `Module1` (для языка VB), ее нужно изменить соответственно.

Обратите внимание на то, что имя класса не распознается средствами технологии IntelliSense и при вводе выдаст ошибку, если вы не откомпилировали проект после изменения имени класса (поскольку только после этого компилятор восстановит класс).

Проектировщик Workflow Foundation

Проектировщик WF позволяет перетаскивать действия управления потоком и стандартные действия (из панели инструментов) на поверхность проекта рабочего потока и соединять их так, чтобы получился рабочий поток. Это очень похоже на создание формы, когда вы перетаскиваете элементы управления из панели инструментов на рабочую область проектировщика, но в данном случае вы создаете рабочий поток. Когда вы только создаете проект, пустой рабочий поток выглядит в проектировщике так, как показано на рис. 32.3.

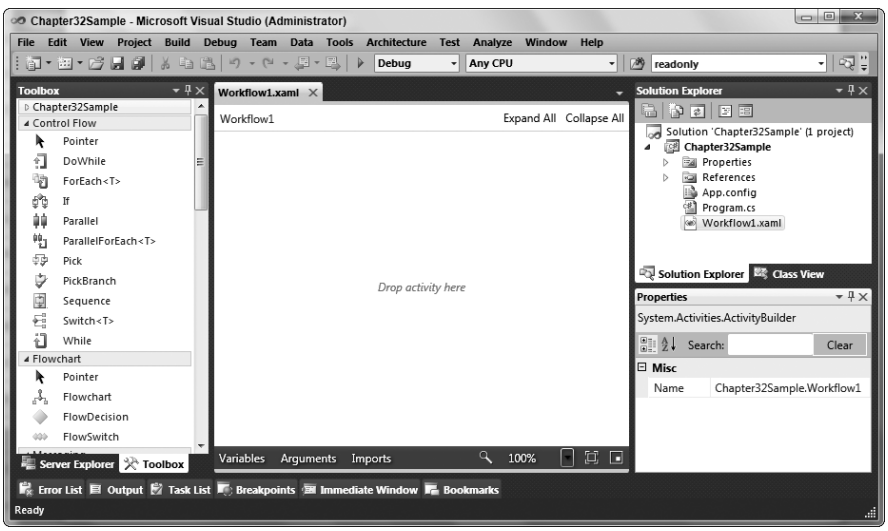


Рис. 32.3

В нижней части окна проектировщика находятся три кнопки: Variables, Arguments и Imports. Щелчки на этих кнопках приводят к появлению в нижней части окна проектировщика панелей, позволяющих изменять соответствующие конфигурации.

Переменные, предназначенные для использования действиями, могут быть определены в пределах некой области видимости, которая определяется родительским действием. Добавьте переменную, развернув панель Variables (как показано на рис. 32.4), щелкнув в области Create Variable и введя ее имя. Можете задать тип переменной, щелкнув в столбце Variable Type и выбрав тип из списка. Если необходимый тип в списке отсутствует, щелкните на элементе Browse for Types, чтобы открыть диалоговое окно, позволяющее ввести полностью квалифицированное имя типа, или просмотрите дерево сборок, на которые имеются ссылки, чтобы найти его. Щелчок в столбце Scope отобразит раскрывающийся список, который позволит изменить область видимости переменной (выбрав действие, которому она принадлежит). У этого действия и его дочерних действий будет доступ к данной переменной. Щелчок в столбце Default позволит ввести выражение (в коде на языке VB), которое установит значение по умолчанию для переменной.

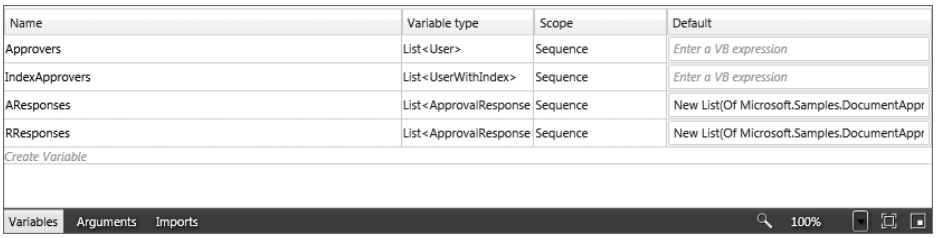


Рис. 32.4



Столбец значений по умолчанию принимает выражения, а не значения. Если захотите присвоить переменной значение, а не выражение, введите литеральное значение, а не само значение. Литеральные значения для числовых значений идентичны, но если переменная будет строкой, то необходимо заключить ее в двойные кавычки. Это относится также к установке значения по умолчанию для аргументов.

Панель Arguments (рис. 32.5) позволяет определять для действия аргументы ввода и вывода, обеспечивающие обмен данными с действием. Существуют четыре типа аргументов, т.е. направлений передачи аргументов: *входящие аргументы* (Input), *исходящие аргументы* (Output), *входящие и исходящие аргументы* (In/Out) и *аргументы свойств* (Property).

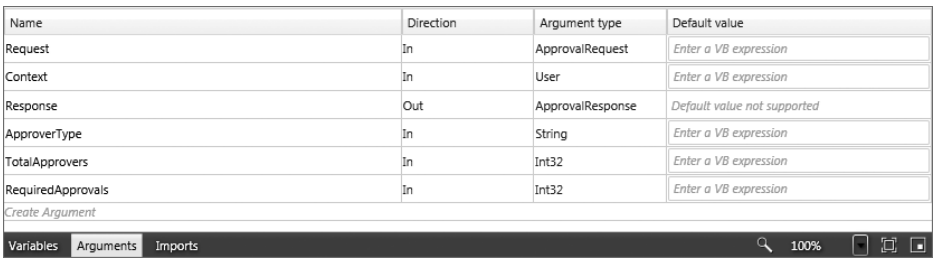


Рис. 32.5

- Входящие аргументы (Input) можно концептуально считать аналогом параметров, передаваемых методам по значению в обычном коде.
- Исходящие аргументы (Output) можно концептуально считать аналогом исходящих параметров методов в обычном коде, значения которого устанавливаются в методе и возвращаются вызывающей стороне.
- Входящие и исходящие аргументы (In/Out) можно концептуально считать аналогом передачи методам параметров по ссылке в обычном коде.
- Аргументы свойств (Property) можно концептуально считать аналогом присвоения значений свойствам объекта в обычном коде.

Добавьте аргумент, отобразив панель Arguments, щелкнув в области с надписью Create Argument и введя его имя. Определите вид аргумента, щелкнув в столбце Direction и выбрав тип из раскрывающегося списка. Можете установить тип аргумента, щелкнув в столбце Argument Type и выбрав тип из раскрывающегося списка. Подобно переменным, можете также назначать аргументу выражение в качестве значения по умолчанию (только для аргументов In и Property).



У действий может быть также возвращаемое значение.

Панель Imports (рис. 32.6) позволяет импортировать пространства имен для использования в выражениях (то же самое, что операторы `using` в языке C#). Вверху панели есть раскрывающийся список, позволяющий указать пространство имен для импорта и добавления в список или выбрать пространство имен, уже находящееся в списке.

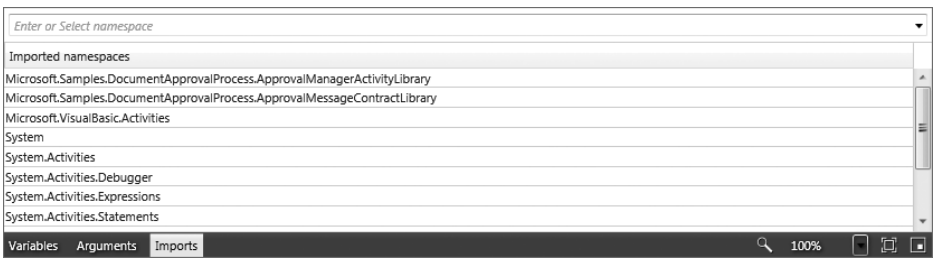


Рис. 32.6

По мере увеличения сложности рабочие потоки могут стать весьма большими и потенциально громоздкими, но, к счастью, проектировщик WF содержит несколько инструментов, способных помочь вам справиться с моделью и обеспечить навигацию по ней. Вы обнаружите, что у некоторых действий есть пиктограмма в их верхнем правом углу, позволяющая прокрутить их так, чтобы отображался только заголовок (т.е. свернуть их), или развернуть, если они свернуты. Поскольку вы можете многократно вкладывать в действия другие действия, потенциально создавая довольно глубокую и сложную иерархию, может быть очень полезно скрывать часть этих сложностей, свертывая те действия, которые вы не редактируете в данный момент. Свертывание действий поможет сэкономить пространство, которое они занимают в диаграмме рабочего потока, и может также использоваться для сокрытия сложной иерархии вложенных действий, содержавшихся в них. Впоследствии вы сможете развернуть действия снова, щелкнув на той же пиктограмме (стрелка которой изменит направление согласно состоянию действия).

В правом верхнем углу проектировщика имеются кнопки Expand All и Collapse All (после щелчка обе меняются на Restore). Нередко может быть полезно "свернуть" весь рабочий поток (используя кнопку Collapse All) до его действий верхнего уровня, от которых вы можете затем переходить к конкретным действиям, развертывая их как нужно, чтобы следовать определенным логическим путем. Кроме того, можете также использовать кнопку Expand All, чтобы развернуть все действия, формирующие рабочий поток, и получить общее представление о нем.

Вы можете изменять масштаб представления рабочего потока, используя раскрывающийся список в правой нижней части окна проектировщика (где перечислены проценты масштаба), или щелкнуть на пиктограмме с изображением лупы, чтобы вернуть 100%-ный масштаб представления. Пиктограмма справа от раскрывающегося списка автоматически выберет уровень масштабирования, который позволит отобразить весь рабочий поток в пределах видимой области окна проектировщика, избавляя от необходимости прокрутки окна.

Если у вас есть рабочий поток с действиями, которые вы не хотите сворачивать, и слишком большой, чтобы полностью поместиться в видимой области окна проектировщика, используйте окно Overview, которое открывается после щелчка на самой правой пиктограмме в правой нижней части проектировщика. Это отобразит в окне проектировщика специальное окно (как показано на рис. 32.7), которое позволит панорамировать рабочий поток при перетаскивании оранжевого прямоугольника (представляющего видимую часть рабочего потока), чтобы отобразить ту часть рабочего потока, которую вы хотите просмотреть в настоящее время.

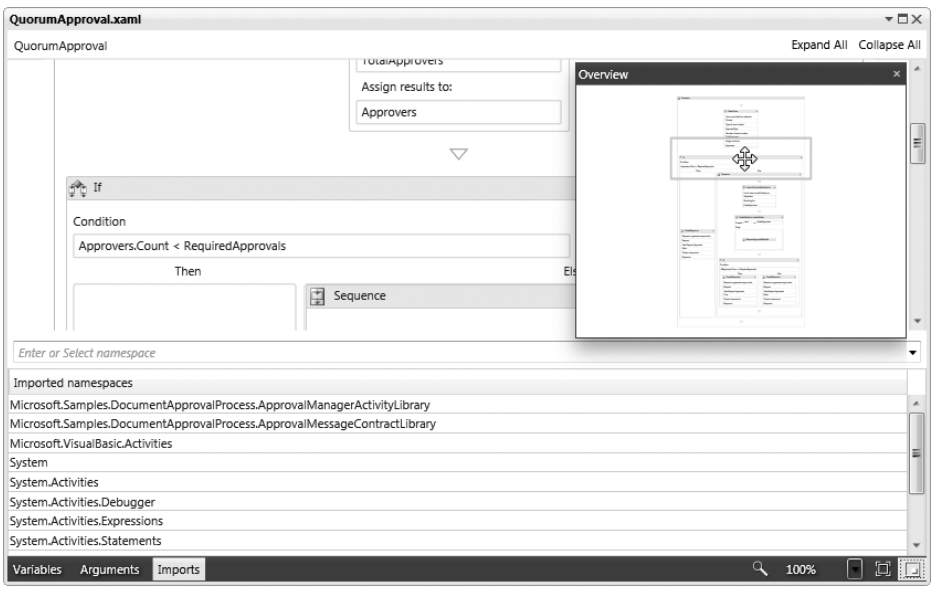


Рис. 32.7

Как уже упоминалось, одним из преимуществ использования платформы WF является то, что диаграмма рабочего потока применяется как форма документации для вашего бизнес-процесса, логики и правил. Зачастую бывает очень полезно поместить эту диаграмму в документацию или представления, тем более что сделать это очень

просто. Щелкните правой кнопкой мыши где-нибудь на рабочей области проектировщика. В контекстном меню есть две команды, которые можно использовать с этой целью: Save as Image и Copy as Image. При выборе команды меню Copy as Image изображение всей диаграммы копируется в буфер обмена, а при выборе команды меню Save as Image отображается диалоговое окно, позволяющее сохранить диаграмму в документе формата JPEG, PNG, GIF или XPS (по вашему выбору). Затем вы можете вставить диаграмму в свой документ или представление, если скопировали его в буфер обмена, или импортировать его, если сохраняли на диске.

Создание рабочего потока

В данном разделе рассматривается процесс создания очень простого рабочего потока, который демонстрирует множество возможностей платформы WF. Для этого примера будет написано простое консольное приложение, получающее ввод от пользователя, но делающее это в рабочем потоке, а не в обычном коде.

Разработка рабочего потока

В первую очередь необходимо поместить в проектировщик действие управления потоком, которое планирует выполнение содержащихся в нем действий. Для этого примера мы используем действие `Sequence`. Найдите его в разделе Control Flow панели инструментов и перетащите в свой рабочий поток `SimpleProcessWorkflow`, как показано на рис. 32.8.

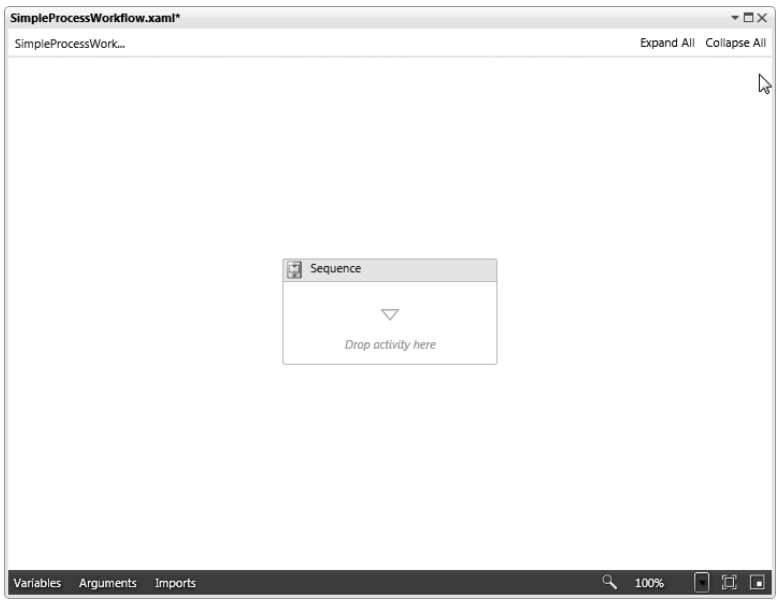


Рис. 32.8

Теперь имеет смысл дать осмысленное имя. Щелкните на имени в заголовке и измените его на **SimpleProcessSequence**. Или выберите действие и установите значение его свойства DisplayName в окне инструментов Properties.

Для первого примера заставим рабочий поток выполнять цикл do/while, который будет выводить на консоль сообщение пять раз. Для этого необходимо перетащить из раздела Control Flow панели инструментов действие DoWhile в действие Sequence. Сделав это, вы увидите, что новое действие и действие Sequence теперь отображаются как недопустимые (об этом свидетельствует красная пиктограмма с восклицательным знаком с правой стороны заголовков обоих действий). Дело в том, что для действия DoWhile должно быть установлено условное выражение, прежде чем оно может считаться допустимым.



Если вы попытаетесь откомпилировать приложение, у которого есть недопустимое действие, то оно будет откомпилировано все равно, но когда вы попробуете запустить его, то получите ошибку во время выполнения программы. Список всех ошибок проверки правильности рабочего потока вы увидите в окне Error List.

Поскольку вы хотите поместить в действие DoWhile несколько действий, добавьте действие Sequence как его дочернее. Назовите эту последовательность WriteHelloWorldSequence.

Теперь найдите в панели инструментов действие WriteLine (раздел Primitives) и перетащите его в действие WriteHelloWorldSequence. Чтобы обеспечить вывод в командную строку фразы Hello World, установите в его аргумент Text значение "Hello_World" (если аргументу присваивается выражение, имеющее строковое значение, то ему необходимо присвоить литеральное значение, заключенное в кавычки).

Для того чтобы вывод был заметней, перетащите действие Delay (из раздела Primitives панели инструментов) в действие WriteHelloWorldSequence, а затем сделайте то же самое с действием WriteLine. Аргумент Duration действия Delay имеет тип TimeSpan. Используйте выражение, чтобы определить его значение как 200 миллисекунд, поскольку оно читается лучше, чем литеральное значение.

```
TimeSpan.FromMilliseconds(200)
```

Для контроля времени выполнения этого цикла добавьте в действие SimpleProcessSequence переменную Counter (которая будет доступна для всех действий последовательности). Выберите действие SimpleProcessSequence и отобразите панель Variables. Щелкните на надписи Create Variable, введите имя **Counter**, тип Int32 и значение по умолчанию 0.

Вернитесь к действию DoWhile и укажите следующее выражение как его условие.

```
Counter < 5
```

Заключительный этап подразумевает фактически инкремент переменной Counter. Добавьте к последовательности (после действия Delay) действие Assign (из раздела Primitives панели инструментов) и установите его аргумент To в Counter, а аргумент Value в Counter + 1.

Ваш пример рабочего потока теперь закончен и должен выглядеть так, как рис. 32.9.

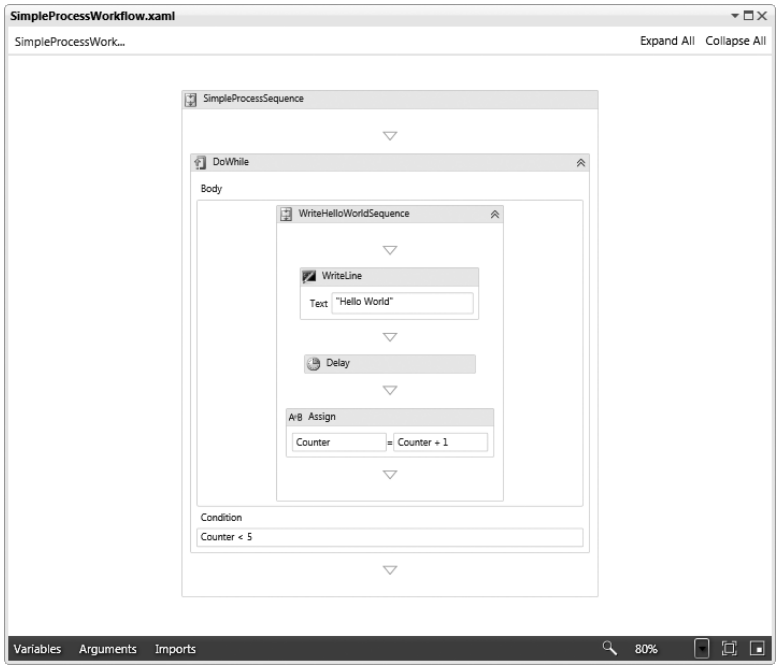


Рис. 32.9

Запустите приложение, результат выполнения рабочего потока которого приведен на рис. 32.10.

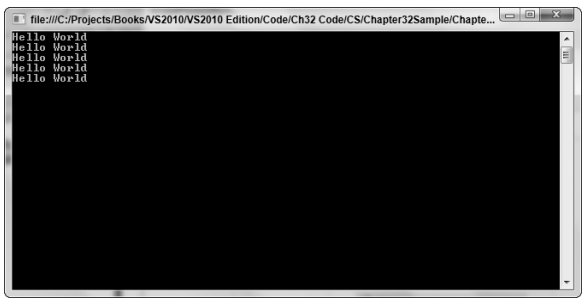


Рис. 32.10

Создание кода действий

Теперь создайте специальное действие, работа которого по получению ввода от пользователя определяется в коде. Добавьте в свой проект новый элемент, выбрав шаблон элемента Code Activity в разделе Workflow диалогового окна Add New Item (рис. 32.11), и назовите его **`UserInput`**.

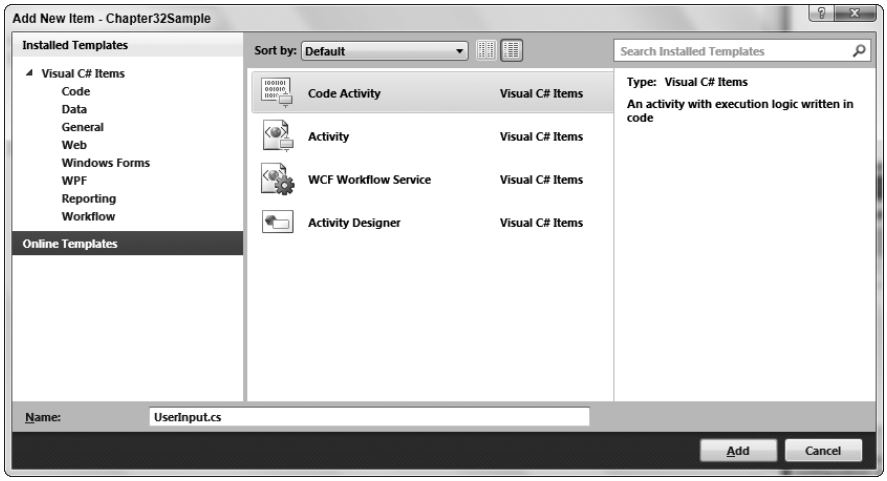


Рис. 32.11

В результате будет создан класс, производный от класса `System.Activities.CodeActivity`, и переопределяющий метод `Execute`, в который вы можете вписать код, выполняющий это действие. Он также содержит простой входящий аргумент `Text`, определенный как свойство класса, который вы можете удалить, поскольку это действие не будет требовать ввода данных (удалите также строку кода в методе `Execute`, который получает его значение).

Это действие получит ввод от пользователя, который смогут использовать другие действия в рабочем потоке. Вы можете возвратить значение либо как аргумент вывода, либо как возвращаемое значение. Подходит и то и другое, поэтому для данного примера выберем возвращаемое значение.

Для того чтобы возвратить значение вместо наследования от класса `CodeActivity`, необходимо наследовать от его общей версии (в которую вы передаете тип, возвращающий действие). Измените класс так, чтобы он происходил от общего класса `CodeActivity`, передав ему тип возвращаемого значения. Измените метод `Execute`, чтобы он возвращал тип, отличный от `void` (на языке C#), или функцию, которая возвращает тип (на языке VB). Тем самым мы просто вернем значение, возвращаемое функцией `Console.ReadLine()`, в методе `Execute`.

VB

```
Public NotInheritable Class UserInput
    Inherits CodeActivity(Of String)

    Protected Overrides Function Execute(ByVal context As CodeActivityContext) _
                                         As String

        Return Console.ReadLine()
    End Function
End Class
```

C#

```
public sealed class UserInput : CodeActivity<string>
{
```

```
protected override string Execute(CodeActivityContext context)
    {
        return Console.ReadLine();
    }
}
```

Если теперь вы переключитесь назад на рабочий поток, находясь в проектировщике, то увидите, что в панели инструментов этого действия нет. Но как только вы откомпилируете свой проект, оно появится в разделе с тем же именем, что и у вашего проекта (рис. 32.12).

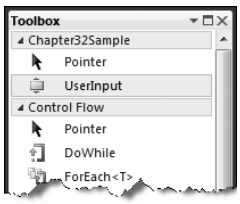


Рис. 32.12

Перетащите действие из панели инструментов в рабочий поток, в основную последовательность `SimpleProcessSequence` и поместите его после действия `DoWhile`. Как можно заметить, у проектировщика нет хорошего пользовательского интерфейса для действий (только простой блок), но вы можете создать его. Однако данная тема не рассматривается в этой главе.

В окне Properties доступно свойство `Result`, в котором может быть определено выражение для работы со значением, возвращаемым методом `Execute`. Осталось только присвоить возвращаемое значение переменной, которую впоследствии могут использовать действия, принадлежащие последовательности. Создайте в панели Variables новую переменную `UserInputValue` типа `String`. Теперь в окне Properties можно установить переменную `UserInputValue` как выражение для свойства `Result`, которое присвоит возвращаемое значение действия переменной `UserInputValue`. Можете проверить, как это работает, добавив после действия `UserInput` действие `WriteLine`, которое выводит значение этой переменной на консоль.

Выполнение рабочего потока

Если просмотреть метод `Main` (точка входа приложения) в файле `Program.cs` (для языка C#) или файл `Module1.vb` (для языка VB), то можно найти код, используемый для выполнения рабочего потока.

VB

```
WorkflowInvoker.Invoke(New SimpleProcessWorkflow())
```

C#

```
WorkflowInvoker.Invoke(new SimpleProcessWorkflow());
```

Для вызова рабочего потока, который, как описано выше, не имеет никакого контроля над фактическим выполнением рабочего потока, кроме простой инициализации его выполнения, используется класс `WorkflowInvoker`.

Но если необходимо больше контроля над выполнением рабочего потока (для возобновления выполнения от закладки или сохранения/загрузки рабочего потока), придется обратиться к классу `WorkflowApplication`, чтобы вызвать рабочий поток. Простейшее использование класса `WorkflowApplication` для вызова рабочего потока — это обработка его события `Complete`.

VB

```
Dim syncEvent As New AutoResetEvent(False)

Dim app As New WorkflowApplication(New SimpleProcessWorkflow())
```

```
app.Completed = Function(args)
        Console.WriteLine("Workflow instance has completed!")
        Thread.Sleep(1000)
        syncEvent.Set()
        Return Nothing
    End Function

app.Run()
syncEvent.WaitOne()
```

C#

```
AutoResetEvent syncEvent = new AutoResetEvent(false);

WorkflowApplication app = new WorkflowApplication(new SimpleProcessWorkflow());

app.Completed = (e) =>
{
    Console.WriteLine("Workflow instance has completed!");
    Thread.Sleep(1000);
    syncEvent.Set();
};

app.Run();
syncEvent.WaitOne();
```



Для работы с фрагментами кода, приведенными выше, в начало файла необходимо добавить оператор `Imports/using` для пространства имен `System.Threading`.

Этот код назначает делегата, который будет запущен, когда рабочий поток завершит выполнение. Поскольку метод `Run` завершается немедленно, вы будете ожидать завершения выполнения рабочего потока, прежде чем продолжить (и выйти из приложения) использование метода `WaitOne` на базе события `AutoResetEvent`, уведомляющего обработчик `Completed` о том, что он может разрешить продолжить выполнение потока.



Хотя мы упомянули здесь слово “события”, вы, возможно, заметили из фрагментов кода, что это вовсе не события. Это свойства, которым вы можете назначить делегатов. Но в целях упрощения их описания мы продолжим называть их событиями.

Выполнение рабочего потока через класс `WorkflowApplication` фактически вызывает его как фоновый поток с методом `Run`, завершающимся немедленно. Хост может назначить обработчики различным событиям, передаваемым классом `WorkflowApplication` (возникающим, например, когда экземпляр рабочего потока завершен, ожидает, передает необработанное исключение и т.д.), а также получает способность прервать/отменять/прекращать экземпляр рабочего потока, загружать его из хранилища экземпляра, сохранять, выгружать и возобновлять его от закладки.

Вы можете передавать входящие аргументы в рабочий поток и получать из него значения исходящих аргументов. Входящие аргументы предоставляются как свойства класса рабочего потока, поэтому присвойте им значения прежде, чем вызвать рабочий поток. Исходящие аргументы возвращаются в словарь (представляющий собой возвращаемое значение метода `Workflowlnvoker.Invoke`), где каждый элемент имеет строковый ключ с названием аргумента и соответствующее объектное значение, которое можно привести к соответствующему типу.

Как уже упоминалось, рабочие потоки и действия — это файлы XAML. По умолчанию файл XAML компилируется в приложение (как ресурс), но что если вы хотите использовать в своих интересах тот факт, что можно перенастроить рабочий поток без перекомпиляции приложения? В этом случае файл XAML должен быть включен в проект как файл содержимого, и его следует динамически загружать в приложение из файла. Вот где полезен класс `ActivityXamlServices`. Загрузите файл XAML как действие, используя класс `ActivityXamlServices`, а затем вызовите (т.е. выполните) действие, которое он возвращает, с помощью класса `WorkflowInvoker` или `WorkflowApplication`.

VB

```
Dim activity As Activity =
ActivityXamlServices.Load("SimpleProcessWorkflow.xaml")
WorkflowInvoker.Invoke(activity)
```

C#

```
Activity activity = ActivityXamlServices.Load("SimpleProcessWorkflow.xaml");

WorkflowInvoker.Invoke(activity);
```



Загрузка и выполнение рабочего потока из файла становятся немного сложней, когда он использует специальные действия (такие как `UserInput`), поскольку среда выполнения будет нуждаться в ссылке на сборку, содержащую эти специальные действия, чтобы она могла использовать их. Однако эта тема не обсуждается в данной главе.

Отладка рабочих потоков

Кроме улучшенной поддержки проектировщика для создания рабочих потоков, платформа WF предоставляет также возможность отладки. Чтобы определить контрольную точку в рабочем потоке, выберите действие и нажмите клавишу <F9> или выберите из контекстного меню, появляющегося после щелчка правой кнопкой мыши, команду Breakpoint⇨Insert Breakpoint. Как выглядит действие, когда на нем установлена точка прерывания, показано на рис. 32.13, *слева*, а как выделяется действие при пошаговом проходе рабочего потока, когда он является текущим элементом выполнения, — на рис. 32.13, *справа*.



Рис. 32.13. Точка прерывания установлена

Как и в обычном сеансе отладки, в рабочем потоке вы шаг за шагом прокручиваете код, используя комбинации клавиш. Нажатие клавиши <F10> шаг за шагом прокручивает рабочий поток, а нажатие клавиши <F11> позволяет перейти к текущему действию. В окне Locals можете просмотреть текущие значения переменных данной области видимости.

Конечно, код ваших специальных действий может быть отлажен как обычно, с помощью установки точек прерывания в редакторе кода и пошагового прохода кода.

Проверка рабочих потоков

Наличие четкой инфраструктуры проверки чрезвычайно важно в бизнес-приложениях, причем особенно важно, чтобы базовая бизнес-логика приложения достаточно полно охватывалась проверками. Поэтому очень важно проверять рабочие потоки, лежащие в основе вашей бизнес-логики. К счастью, это действительно возможно, и вы можете использовать предпочитаемую вами инфраструктуру проверки, даже практику *разработки через тестирование* (Test Driven Development — TDD), если хотите. Как упоминалось в разделе "Выполнение рабочего потока", используя метод `WorkflowInvoker.Invoke` для выполнения рабочего потока, вы можете передать значения входящих аргументов в рабочий поток и получать значения выходных аргументов (в словаре). Поэтому проверка вашего рабочего потока столь же проста, как передача значений входящих аргументов и проверка соответствия значений выходных аргументов ожидаемым значениям.

Хостинг проектировщика рабочего потока

Одним из преимуществ наличия декларативного рабочего потока с перестраиваемой конфигурацией является возможность перенастройки приложения в соответствии с бизнес-требованиями, без необходимости перекомпилировать его. Это означает (в теории), что конечный пользователь, выбрав подходящий инструмент (т.е. проектировщик WF), должен быть в состоянии модифицировать рабочий поток без привлечения его разработчика (однако создание специальных действий — это уже другая история). Конечно, использование проектировщика WF для модификации рабочего потока — это, вероятно, слишком сложная задача для необученного конечного пользователя. Действительно, этот инструмент предназначен для использования разработчиками. Как уже говорилось, при небольшом обучении и здравом смысле пользователь уровня IT-специалиста (такой как бизнес-аналитик или другой) сможет вполне успешно решить эту задачу.

Если дело обстоит так, то не составит труда расположить проектировщик WF в собственном приложении и предоставить его конечному пользователю для модификации. Проектировщик WF — это компонент WPF, который вы можете располагать в своих приложениях WPF, предоставляя его, по необходимости, пользователям для модификации рабочего потока. Можете также содержать проектировщик WF в приложении Windows Forms, используя совместимость WPF, описанную в главе 18. Но данная глава посвящена его хостингу именно в приложении WPF.



Изучение этой темы подразумевает, что у вас есть некоторый опыт работы с технологией WPF и языком XAML. Более подробная информация по этой теме приведена в главе 18.

Создайте новый проект WPF `WFDesignerHost` и добавьте в него ссылки на следующие сборки.

- System.Activities.dll
- System.Activities.Core.Presentation.dll
- System.Activities. Presentation.dll

Вам также придется добавить ссылки на любые сборки, содержащие специальные действия, которые хотите использовать в рабочих потоках своего приложения.

У проектировщика есть три основных (отдельных) компонента: панель инструментов, окно Properties и рабочая область проекта. Давайте создадим пользовательский интерфейс, который создаст экземпляры и отобразит их все.

Откройте файл MainWindow.xaml и определите элемент управления Grid по имени WFLayoutGrid. Добавьте в этот элемент три столбца (их размеры можно будет определить позже). В первом столбце расположите панель инструментов Toolbox, во втором — поверхность проекта и в третьем — окно Properties. Панель инструментов может быть создана либо декларативно, с помощью языка XAML, либо в коде, а рабочая область проекта и окно Properties могут быть созданы только в коде. Для этого примера мы создадим все три этих элемента управления в коде.

Откройте код файла MainWindow.xaml. Импортируйте пространства имен, приведенные ниже.

VB

```
Imports System.Activities
Imports System.Activities.Core.Presentation
Imports System.Activities.Presentation
Imports System.Activities.Presentation.Toolbox
Imports System.Activities.Statements
Imports System.Linq
Imports System.Reflection
Imports System.Windows
Imports System.Windows.Controls
```

C#

```
using System;
using System.Activities;
using System.Activities.Core.Presentation;
using System.Activities.Presentation;
using System.Activities.Presentation.Toolbox;
using System.Activities.Statements;
using System.Linq;
using System.Reflection;
using System.Windows;
using System.Windows.Controls;
```

Сначала зарегистрируйте метаданные проектировщика.

VB

```
Private Sub RegisterMetadata()
    Dim metaData As New DesignerMetadata()
    metaData.Register()
End Sub
```

C#

```
private void RegisterMetadata()
{
    DesignerMetadata metaData = new DesignerMetadata();
    metaData.Register();
}
```

Затем добавьте к странице панель инструментов. Вы увидите, что панель инструментов не заполняется действиями автоматически, поэтому необходимо самостоятельно заполнить ее действиями, которые должны быть доступны пользователю. Следующий код отрабатывает эти задачи при создании экземпляра панели инструментов и добавляет все действия в той же сборке, где находится действие `Sequence`.

VB

```
Private Sub AddToolboxControl(ByVal parent As Grid, ByVal row As Integer,
                              ByVal column As Integer)
    Dim toolbox As New ToolboxControl()

    Dim category As New ToolboxCategory("Activities")
    toolbox.Categories.Add(category)

    Dim query =
                From type In Assembly.GetAssembly(GetType(Sequence)).GetTypes()
                Where type.IsPublic AndAlso
                Not type.IsNested AndAlso
                Not type.IsAbstract AndAlso
                Not type.ContainsGenericParameters AndAlso
                (GetType(Activity).IsAssignableFrom(type) OrElse
                    GetType(IActivityTemplateFactory).IsAssignableFrom(type))
                Order By type.Name
                Select New ToolboxItemWrapper(type)

    query.ToList().ForEach(Function(item)
            category.Add(item)
            Return Nothing
        End Function)

    Grid.SetRow(toolbox, row)
    Grid.SetColumn(toolbox, column)
    parent.Children.Add(toolbox)
End Sub
```

C#

```
private void AddToolboxControl(Grid parent, int row, int column)
{
    ToolboxControl toolbox = new ToolboxControl();

    ToolboxCategory category = new ToolboxCategory("Activities");
    toolbox.Categories.Add(category);

    var query = from type in Assembly.GetAssembly(typeof(Sequence)).GetTypes()
                where type.IsPublic &&
                !type.IsNested &&
                !type.IsAbstract &&
                !type.ContainsGenericParameters &&
                (typeof(Activity).IsAssignableFrom(type) ||
                typeof(IActivityTemplateFactory).IsAssignableFrom(type))
                orderby type.Name
                select new ToolboxItemWrapper(type);

    query.ToList().ForEach(item => category.Add(item));

    Grid.SetRow(toolbox, row);
    Grid.SetColumn(toolbox, column);
    parent.Children.Add(toolbox);
}
```

Добавьте проектировщик и окно Properties (оба — элементы управления, возвращенные из экземпляра класса `WorkflowDesigner`).

VB

```
Private Sub AddDesigner(ByVal parent As Grid,
                        ByVal designerRow As Integer,
                        ByVal designerColumn As Integer,
                        ByVal propertiesRow As Integer,
                        ByVal propertiesColumn As Integer)
    Dim designer As New WorkflowDesigner()
    designer.Load(New Sequence())

    Grid.SetRow(designer.View, designerRow)
    Grid.SetColumn(designer.View, designerColumn)
    parent.Children.Add(designer.View)

    Grid.SetRow(designer.PropertyInspectorView, propertiesRow)
    Grid.SetColumn(designer.PropertyInspectorView, propertiesColumn)
    parent.Children.Add(designer.PropertyInspectorView)
End Sub
```

C#

```
private void AddDesigner(Grid parent, int designerRow,
      int designerColumn, int propertiesRow, int propertiesColumn)
{
    WorkflowDesigner designer = new WorkflowDesigner();
    designer.Load(new Sequence());

    Grid.SetRow(designer.View, designerRow);
    Grid.SetColumn(designer.View, designerColumn);
    parent.Children.Add(designer.View);

    Grid.SetRow(designer.PropertyInspectorView, propertiesRow);
    Grid.SetColumn(designer.PropertyInspectorView, propertiesColumn);
    parent.Children.Add(designer.PropertyInspectorView);
}
```

Вызовите эти три функции из метода/конструктора окна New.

C#

```
Public Sub New()
    InitializeComponent()

    RegisterMetadata()
    AddToolboxControl(WFLayoutGrid, 0, 0)
    AddDesigner(WFLayoutGrid, 0, 1, 0, 2)
End Sub

public MainWindow()
{
    InitializeComponent();

    RegisterMetadata();
    AddToolboxControl(WFLayoutGrid, 0, 0);
    AddDesigner(WFLayoutGrid, 0, 1, 0, 2);
}
```

Теперь можете запустить и проверить проект. Ваш окончательный пользовательский интерфейс должен выглядеть так, как на рис. 32.14 (он может быть, конечно, улучшен, если затратить больше времени на разработку страницы).



Рис. 32.14

Резюме

В этой главе вы узнали, что платформа Windows Workflow — это средство определения бизнес-процесса, которое особенно полезно при наличии часто изменяющихся или длинных бизнес-процессов. Вы также узнали, как создать и запустить простой рабочий поток, как организовать хостинг проектировщика рабочего потока в собственном приложении. Платформа Windows Workflow быстро становится стандартом для реализации рабочих потоков на платформе Microsoft, позволяя многократно использовать полученные здесь навыки для создания рабочих потоков в различных приложениях, которые их поддерживают.

33

Службы клиентских приложений

В ЭТОЙ ГЛАВЕ...

- Доступ к службам клиентских приложений
- Управление ролями приложения
- Персистентность пользовательских параметров
- Определение специального диалогового окна регистрации

Создание приложений, построенных вокруг служб и отделения опыта их использования от серверных хранилищ данных, обусловило появление требований для приложений без постоянного подключения. *Приложения без постоянного подключения* (occasionally connected application) — это те приложения, которые продолжают работать независимо от доступности сети. В главе 34 вы узнаете, как могут синхронизироваться данные локального хранилища, что позволяет пользователю продолжать работу, когда приложение отключено от сети. Но этот случай ведет к дискуссии (нередко весьма горячей) о безопасности. Поскольку безопасность (т.е. аутентификация пользователя и авторизация его роли) зачастую обслуживается централизованно, довольно трудно распространить ее на приложения без постоянного подключения.

В этой главе вы ознакомитесь со службами клиентских приложений, которые дополняют службы приложений ASP.NET Application Services для использования в клиентских приложениях. Службы приложений ASP.NET Application Services — это модель аутентификации пользователя, авторизации его роли и профильного управления на основе провайдера, которая в прошлом ограничивалась веб-службами и веб-сайтами. В среде разработки Visual Studio 2010 вы можете настроить свое улучшенное клиентское приложение, приложение Windows Forms или WPF так, чтобы использовать эти службы и проверять пользователей, ограничивая их функциональные возможности на основании присвоенных им ролей, и сохранять их личные настройки централизованно.

Клиентские службы

В данной главе вы ознакомитесь с различными службами приложений на примере простого приложения WPF. В данном случае это приложение `ClientServices`, которое вы можете создать, выбрав команду меню File⇨New⇨Project, а затем шаблон WPF Application (для языка C # или VB).

Для того чтобы начать использовать службы клиентских приложений, необходимо установить соответствующий флажок на вкладке Services дизайнера свойств проекта, как показано на рис. 33.1. По умолчанию должен использоваться режим аутентификации Windows. Это идеально, если вы создаете приложение для работы в пределах одной организации, и можете считать, что у вас всегда есть сертификаты домена. Выбор данной возможности гарантирует, что эти сертификаты домена используются для доступа к службам ролей и параметров. В качестве альтернативы можете использовать аутентификацию с помощью форм, тогда у вас есть полный контроль над механизмом аутентификации пользователей. Мы вернемся к этой теме позже.

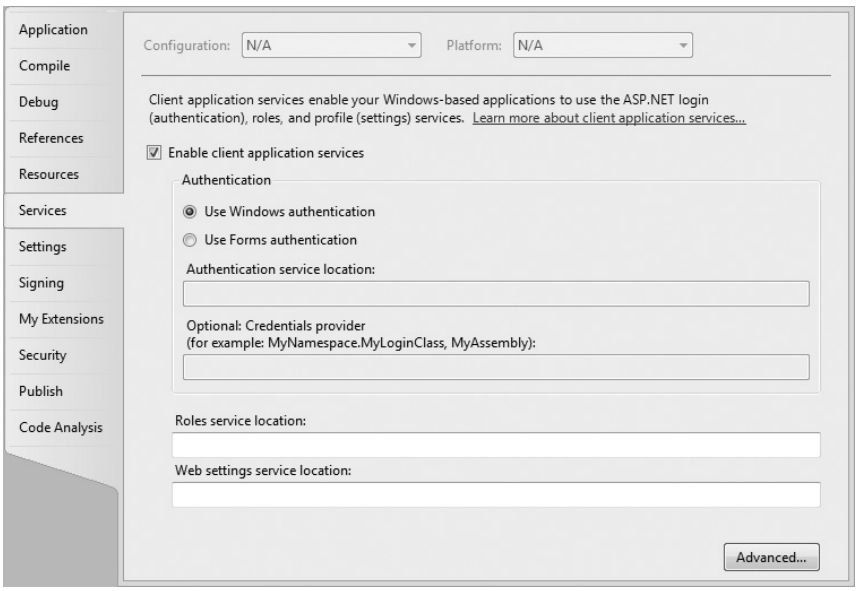


Рис. 33.1



Добавлять службы клиентских приложений к существующим приложениям в дизайнере свойств проекта среды разработки Visual Studio 2010 можно точно так же, как и в новые приложения.

Как можно заметить, когда вы подключаете службы клиентских приложений, в ваше приложение добавляется файл `app.config`, если его еще нет. Особенно интересен раздел `<system.web>`, фрагмент которого должен выглядеть так, как показано ниже.



```
<system.web>
    <membership defaultProvider="ClientAuthenticationMembershipProvider">
        <providers>
            <add name="ClientAuthenticationMembershipProvider" type=
"System.Web.ClientServices.Providers.ClientWindowsAuthentication
MembershipProvider, System.Web.Extensions, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" serviceUri="" connectionStringName="Default
Connection" credentialsProvider=""/>
        </providers>
    </membership>
    <roleManager defaultProvider="ClientRoleProvider" enabled="true">
        <providers>
            <add name="ClientRoleProvider"
type="System.Web.ClientServices.Providers.ClientRoleProvider, System.Web.Ext
ensions, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
serviceUri="" cacheTimeout="86400" connectionStringName="DefaultConnection"/>
        </providers>
    </roleManager>
</system.web>
```

Сниппет кода app.config

Как можно заметить, здесь были определены провайдеры для управления членством и ролями. Вы можете дополнить инфраструктуру служб клиентских приложений, создав собственные провайдеры, способные обращаться непосредственно к базе данных или некоторому другому дистанционному хранилищу сертификатов, такому как активный каталог. По существу, проектировщик свойств проекта лишь модифицирует файл `app.config` так, чтобы использовать провайдеры, поставляемые с инфраструктурой .NET Framework, и определяет связанные с этим свойства. Для того чтобы реализовать собственный провайдер, необходимо создать реальные классы, которые реализуют абстрактные методы, определенные в классах `System.Web.Security.RoleProvider`, `System.Web.Security.MembershipProvider` или `System.Configuration.SettingsProvider` (в зависимости от провайдера, который вы реализуете).

После того как определите провайдеры стандартной роли и принадлежности, используйте службы клиентских приложений для проверки пользователя приложения. Для этого необходимо вызвать метод `ValidateUser` класса `System.Web.Security.Membership`, как показано в следующем фрагменте кода.



C#

```
using System.Web.Security;z
public partial class MainWindow : Window{
    public MainWindow(){
        InitializeComponent();
    }
    private void Window_Loaded(object sender, RoutedEventArgs e){
        if (Membership.ValidateUser(null, null)){
            MessageBox.Show("User is valid");
        }
        else{
           MessageBox.Show("Unable to verify user, application exiting");
           this.Close();
           return;
        }
    }
}
```

Сниппет кода MainWindow.cs

VB

```
Imports System.Web.Security
Class MainWindow
    Private Sub Window_Loaded(ByVal sender As System.Object,
                yVal e As System.Windows.RoutedEventArgs) Handles Me.Loaded
        If Membership.ValidateUser(Nothing, Nothing) Then
            MessageBox.Show("User is valid")
        Else
            MessageBox.Show("Unable to verify user, application exiting")
            Me.Close()
            Return
        End If
    End Sub
End Class
```

Сниппет кода MainWindow.vb

Интересно то, что здесь нет перегрузки метода `ValidateUser`, который не получает аргументов; вместо этого при аутентификации с помощью системы Windows вы должны использовать для аргументов пароля и имени пользователя `Nothing` (на языке VB) или `null` (на языке C #). В данном случае метод `ValidateUser` делает несколько больше, чем заполнение свойства `CurrentPrincipal` приложения для использования службы клиентских приложений, чтобы определить, каким ролям принадлежит пользователь, и по умолчанию возвращает значение `true`. Впоследствии вы увидите, что использование этого метода является эквивалентом регистрации пользователя в приложении.



Приведенный выше фрагмент кода и другие фрагменты в этой главе могут потребовать импорта пространства имен `System.Web.Security` в файл класса. Возможно, вам также придется вручную добавить ссылку на библиотеку `System.Web.dll`, чтобы разрешить ссылки на типы.

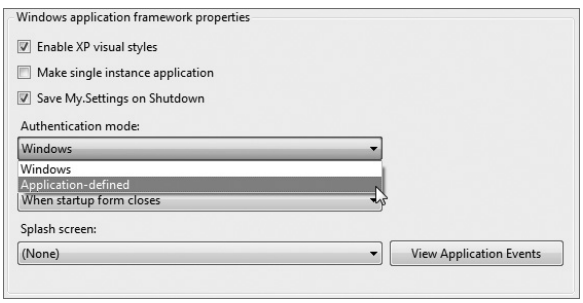


Рис. 33.2

Клиентские службы приложений включают нечто, зачастую называемое *инфраструктурой приложения* (application framework), осуществляющей защиту. У языка VB в течение долгого времени была собственная инфраструктура приложений для приложений Windows Forms Applications, использование которой разрешалось и запрещалось на вкладке Application дизайнера свойств проекта. Эта инфраструктура уже включает ограниченную поддержку аутентификации пользователя, но конфликтует с клиентскими службами приложений. На рис. 33.2 показано, как выбрать определенный приложением режим аутентификации, чтобы вы могли использовать в своем приложении и инфраструктуру приложений Windows, и службы клиентских приложений.

Обратите внимание на то, что этот параметр доступен, только если вы разрабатываете приложение Windows Forms Application на языке VB.

Ролевая авторизация

Мы показали, как подключить службы клиентского приложения, но в действительности не придавали им значения, поскольку пользователь уже был аутентифицирован операционной системой, когда вы использовали аутентификацию Windows для клиентского приложения. Но ни одна операционная система не способна определить, к какой роли принадлежит пользователь и к каким функциональным возможностям приложения он может обратиться. Хотя это может сделать само клиентское приложение, будет довольно трудно реализовать все перестановки пользователей, и система окажется непрактичной в обслуживании, поскольку при каждом добавлении пользователя или изменении его роли придется развертывать новую версию приложения. Вместо этого желательно иметь корреляцию между пользователями и ролями, обслуживаемыми на сервере, позволяя приложению работать с намного меньшим набором ролей, чтобы контролировать доступ к функциональным возможностям.

Истинная мощь служб клиентского приложения становится очевидной, когда вы объединяете клиентскую инфраструктуру приложения со службой приложений ASP.NET Application Services. Для того чтобы увидеть, как это работает, добавьте в свое решение новый проект (VB или C #), используя шаблон ASP.NET Empty Web Application (в узле Web диалогового окна New Project), и назовите его `ApplicationServices`.

Щелкните правой кнопкой мыши в окне Solution Explorer на недавно созданном проекте и в появившемся контекстном меню выберите команду Properties, чтобы отобразить окно дизайнера свойств проекта. Поскольку вы будете обращаться к этому веб-приложению из других частей решения, желательно использовать предопределенный порт и виртуальный каталог на сервере Visual Studio Development Server. На вкладке Web установите порт к `12345` и маршрут к виртуальной папке `/ApplicationServices`.

Службы приложений ASP.NET Application Services являются основанной на провайдере моделью аутентификации пользователей, управления ролями и хранения профилей. Каждый из этих компонентов может быть задействован независимо, и вы можете использовать либо встроенные провайдеры, либо создать собственные. Для того чтобы предоставить службу управления ролями для доступа через службы клиентского приложения, добавьте в файл `web.config` проекта `ApplicationServices` следующий фрагмент кода перед элементом `<system.web>`.



```
<system.web.extensions>
    <scripting>
        <webServices>
            <roleService enabled="true"/>
        </webServices>
    </scripting>
</system.web.extensions>
```

Сниппет кода web.config

Поскольку для определения, каким ролям принадлежит пользователь, необходимо выполнить некую специальную логику, придется создать специальный провайдер роли, `CustomRoles`, чтобы он занял место стандартного провайдера роли. Для этого добавьте в свой проект новый класс и реализуйте абстрактный класс `RoleProvider`. В этом провайдере роли вас интересует только возвращение значения для метода `GetRolesForUser`; все остальные методы могут быть оставлены как заглушки.



C#

```
public class CustomRoles: RoleProvider{
    public override string[] GetRolesForUser(string username){
        if (username.ToLower().Contains("nick")){
            return new string[] { "All Nicks" };
        }
        else{
            return new string[] { };
        }
    }
```

Сниппет кода CustomRoles.cs



VB

```
Public Class CustomRoles
    Inherits RoleProvider

    Public Overrides Function GetRolesForUser(ByVal username As String)
                                                                As String()
        If username.ToLower.Contains("nick") Then
            Return New String() {"All Nicks"}
        Else
            Return New String() {}
        End If
    End Function
```

Сниппет кода CustomRoles.vb

Теперь вы имеете специальный провайдер роли и обеспечили управление ролью. Отсутствует лишь связующий элемент, который сообщает службе управления ролями об использовании вашего провайдера роли. Добавьте следующий узел `roleManager` в элемент `<system.web>` файла `web.config`.



```
<roleManager enabled="true" defaultProvider="CustomRoles">
    <providers>
        <add name=" CustomRoles"
             type="AuthenticationServices.CustomRoles"/>
    </providers>
</roleManager>
```

Сниппет кода web.config

Последнее, что требуется сделать, — это использовать данную информацию роли в приложении. Для этого сначала необходимо снабдить приложение URI для загрузки информации роли. На вкладке Services свойств проекта `ClientServices` (см. рис. 33.1) введите **`http://localhost:12345/ApplicationServices`**, затем добавьте в метод `Window_Loaded` вызов метода `IsUserInRole`.



C#

```
private void Window_Loaded(object sender, RoutedEventArgs e){
    if (Membership.ValidateUser(null, null))
    { // Закомментировано для краткости.
    }
    if (Roles.IsUserInRole("All Nicks")){
        MessageBox.Show("User is a Nick, so should have Admin rights.");
    }
}
```

Сниппет кода MainWindow.cs



VB

```
Private Sub Window_Loaded(ByVal sender As System.Object,
            ByVal e As System.Windows.RoutedEventArgs) Handles Me.Loaded
    If Membership.ValidateUser(Nothing, Nothing) Then
        '. Закомментировано для краткости .
    End If
    If Roles.IsUserInRole("All Nicks") Then
        MessageBox.Show("User is a Nick, so should have Admin rights.")
    End If
End Sub
```

Сниппет кода MainWindow.vb

Для того чтобы увидеть ваш специальный провайдер роли в действии, установите точку прерывания в методе `GetRolesForUser`. Чтобы достичь этой точки прерывания, и приложение-клиент, и веб-приложение должны работать в режиме отладки. Для этого щелкните в окне Solution Explorer правой кнопкой мыши на узле Solution и в появившемся контекстном меню выберите команду Properties. В узле Startup Project выберите пункт Multiple Startup Projects и установите запуск обоих проектов. Теперь, запустив решение, вы увидите, что метод `GetRolesForUser` вызывается с сертификатами Windows текущего пользователя, как часть проверки его достоверности.

Аутентификация пользователя

В некоторых организациях можно использовать аутентификацию Windows для проверки достоверности всех пользователей. К сожалению, это возможно не всегда, поэтому разработчики приложений должны придумывать собственные решения для определения того, какие пользователи должны иметь доступ к системе. Такой процесс обычно упоминается как *аутентификация на основе форм*, поскольку он, как правило, требует ввода имени пользователя и пароля в форме регистрации некоторого вида. И службы приложений ASP.NET Application Services, и службы клиентского приложения поддерживают аутентификацию на базе форм как альтернативу аутентификации Windows.

Сначала необходимо разрешить доступ службам клиентского приложения к службе управления принадлежностью. Для этого добавьте элемент `<authenticationService>` в элемент `<system.web.extensions>` файла `web.config`. Заметьте, что мы отключили требование SSL, — это, конечно, против всех рекомендаций по безопасности, и для рабочих систем так не делают.



```
<system.web.extensions>
    <scripting>
        <webServices>
            <authenticationService enabled="true" requireSSL="false"/>
            <roleService enabled="true"/>
```

Сниппет кода web.config

Следующий этап подразумевает создание специального провайдера принадлежности, который определит, допустима ли для приложения конкретная комбинация имени пользователя и пароля. Для этого добавьте в приложение `ApplicationServices` новый класс, `CustomAuthentication`, и унаследуйте его от класса `MembershipProvider`. Как и с созданным ранее провайдером роли, вы собираетесь обеспечить лишь минимальную реализацию, которая проверяет сертификат, убеждаясь, что пароль — это представление имени

пользователя в обратном направлении, что имя пользователя находится в предопределенном списке.



C#

```
public class CustomAuthentication : MembershipProvider{
    private string[] mValidUsers = { "Nick" };

    public override bool ValidateUser(string username, string password)
    {
        var reversed = new string(password.Reverse().ToArray());
        return (from user in mValidUsers
                where string.Compare(user, username, true) == 0 &&
                        user == reversed
                select user).Count() > 0;
    }
    // Остальная часть реализации опущена для краткости
}
```

Сниппет кода CustomAuthentication.es



VB

```
Public Class CustomAuthentication
    Inherits MembershipProvider
    Private mValidUsers As String() = {"Nick"}

    Public Overrides Function ValidateUser(ByVal username As String,
                                           ByVal password As String) As Boolean
        Dim reversed As String = New String(password.Reverse.ToArray)
        Return (From user In mValidUsers
                Where String.Compare(user, username, True) = 0 And
                    user = reversed).Count > 0
    End

    ' Остальная часть реализации опущена для краткости
End Class
```

Сниппет кода CustomAuthentication.vb

Как и с созданным вами провайдером роли, вам понадобится сообщить системе управления принадлежности, что она должна использовать ваш провайдер принадлежности. Для этого добавьте следующий фрагмент в элемент `<system.web>` файла `web.config`.



```
<membership defaultProvider="CustomAuthentication">
    <providers>
        <add name="CustomAuthentication"
        type="ApplicationServices.CustomAuthentication"/>
    </providers>
</membership>
<authentication mode="Forms"/>
```

Сниппет кода web.config

Чтобы использовать изменения системы аутентификации, в клиентское приложение понадобится внести лишь минимум изменений. На вкладке Services дизайнера свойств проекта выберите переключатель Use Forms Authentication (Использовать аутентификацию с помощью форм), чтобы сделать доступными текстовые поля Authentication Service Location (Расположение службы аутентификации) и Optional: Credentials Provider (Необязательно: провайдер сертификата). Пока что определите только расположение службы аутентификации как `http://localhost:12345/ApplicationServices`.

Ранее, при использовании аутентификации Windows, в момент инициализации службы клиентского приложения вы вызывали метод `ValidateUser`, передав в качестве двух его аргументов значения `Nothing`. Это было сделано потому, что пользовательские сертификаты могли быть автоматически определены из текущего пользовательского контекста, в котором выполнялось приложение. К сожалению, это невозможно при аутентификации с помощью форм, поэтому необходимо предоставить имя пользователя и пароль.



C#

```
private void Window_Loaded(object sender, RoutedEventArgs e){
    if (Membership.ValidateUser("Nick", "kciN")){
        MessageBox.Show("User is valid");
```

Сниппет кода MainWindow.cs



VB

```
Private Sub Window_Loaded(ByVal sender As System.Object,
                          ByVal e As System.Windows.RoutedEventArgs) _
                          Handles Me.Loaded
    If Membership.ValidateUser("Nick", "kciN") Then
        MessageBox.Show("User is valid")
```

Сниппет кода MainWindow.vb

Если вы определите точку прерывания в методе `ValidateUser` проекта `ApplicationServices`, то увидите, что при запуске этого решения происходит обращение к серверу, чтобы проверить пользователя. Как вы увидите позже, затем эта информация может кэшироваться локально, чтобы облегчить проверку пользователя без подключения.

Параметры

В инфраструктуре .NET Framework v2.0 концепция параметров области видимости `User` была введена для того, чтобы информация о пользователях могла сохраняться между сеансами приложения. Например, позиция окна или информация темы вполне может храниться как параметр пользователя. К сожалению, ранее не было никакого способа централизованного управления этой информацией. В то же время у службы приложений ASP.NET Application Services была концепция информации профиля, которая, по сути, была информацией о пользователях, отслеживаемая на сервере и применяемая веб-приложениями. Естественно, с введением служб клиентского приложения появился смысл объединять эти концепции, чтобы позволить хранить параметры в веб. У этих параметров есть область видимости `User (Web)`.

Подобно службам принадлежности и роли, необходимо разрешить службам клиентского приложения доступ к службе профиля. Для этого следует добавить элемент `<profileService>` в элемент `<system.web.extensions>` файла `web.config`.



```
<system.web.extensions>
    <scripting>
        <webServices>
            <profileService enabled="true"
                readAccessProperties="Nickname"
                writeAccessProperties="Nickname" />
            <authenticationService enabled="true" requireSSL="false"/>
```

Сниппет кода web.config

Согласно приведенным выше примерам, вы создаете специальный провайдер профиля, который использует словарь, расположенный в оперативной памяти, для хранения псевдонимов пользователей. Безусловно, это не лучший способ отслеживания информации профиля, поскольку она бы терялась при каждой смене веб-сервера и не распространялась бы на несколько веб-серверов. Однако в проект `ApplicationServices` необходимо добавить новый класс, `CustomProfile`, производный от класса `ProfileProvider`.



C#

```
using System.Web.Profile;
using System.Configuration;
public class CustomProfile : ProfileProvider{
    private Dictionary<string, string> nicknames =
                                        new Dictionary<string, string>();
    public override System.Configuration.SettingsPropertyValueCollection
        GetPropertyValues(System.Configuration.SettingsContext context,
            System.Configuration.SettingsPropertyCollection collection){
        var vals = new SettingsPropertyValueCollection();
        foreach (SettingsProperty setting in collection){
            var value = new SettingsPropertyValue(setting);
            if (nicknames.ContainsKey(setting.Name)) {
                value.PropertyValue = nicknames[setting.Name];
            }
            vals.Add(value);
        }
        return vals;
    }
    public override void SetPropertyValues(SettingsContext context,
                            SettingsPropertyValueCollection collection){
        foreach (SettingsPropertyValue setting in collection){
            nicknames[setting.Name] = setting.PropertyValue.ToString();
        }
    }
// Остальная часть реализации опущена для краткости
}
```

Сниппет кода CustomProfile.cs



VB

```
Imports System.Configuration

Public Class CustomProfile
    Inherits ProfileProvider
    Private nicknames As New Dictionary(Of String, String)

    Public Overrides Function GetPropertyValues(ByVal context
                                        As SettingsContext,
                        ByVal collection As SettingsPropertyCollection) _
                                    As SettingsPropertyValueCollection
        Dim vals As New SettingsPropertyValueCollection
        For Each setting As SettingsProperty In collection
            Dim value As New SettingsPropertyValue(setting)
            If nicknames.ContainsKey(setting.Name) Then
                value.PropertyValue = nicknames.Item(setting.Name)
            End If
            vals.Add(value)
        Next
        Return vals
    End Function

    Public Overrides Sub SetPropertyValues(ByVal context
                                    As SettingsContext,
                    ByVal collection As SettingsPropertyValueCollection)
```

```
        For Each setting As SettingsPropertyValue In collection
            nicknames.Item(setting.Name)  = setting.PropertyValue.ToString
        Next
        End Sub

        ' Остальная часть реализации опущена для краткости
    End Class
```

Сниппет кода CustomProfile.vb

Отличие от службы профиля в том, что, когда вы определяете используемый провайдер в элементе `<system.web>` файла `web.config`, вам следует также объявить, какие свойства могут быть сохранены службой профиля (см. следующий фрагмент кода). Для того чтобы эти свойства стали доступными через службы клиентского приложения, у них должна быть соответствующая запись в атрибутах `readAccessProperties` и `writeAccessProperties` элемента `<profileService>`, представленного ранее.



```
<profile enabled="true" defaultProvider="CustomProfile">
    <providers>
        <add name="CustomProfile"
type="ApplicationServices.CustomProfile"/>
    </providers>
    <properties>
        <add name="Nickname" type="string"
            readOnly="false" defaultValue="{nickname}"
            serializeAs="String" allowAnonymous="false" />
    </properties>
</profile>
```

Сниппет кода web.config

Кроме того, самый простой способ создания полной службы профиля заключается в использовании утилиты `aspnet_regsql.exe` (как правило, `c:\Windows\Microsoft.NET\Framework\v4-0.21006\aspnet_regsql.exe`) для заполнения существующей базы данных SQL Server с соответствующей структурой таблицы. Затем можете использовать встроенный провайдер `SqlProfileProvider` (`SqlMembershipProvider` и `SqlRoleProvider` для провайдеров принадлежности и роли соответственно), чтобы сохранять и возвращать информацию профиля. Для того чтобы использовать этот провайдер, замените элемент профиля, который вы добавили ранее, следующим.

```
<profile enabled="true" defaultProvider="CustomProfile">
    <providers>
        <add name="SqlProvider"
            type="System.Web.Profile.SqlProfileProvider"
            connectionStringName="SqlServices"
            applicationName="SampleApplication"
            description="SqlProfileProvider for SampleApplication" />
```

Обратите внимание на то, что атрибут `connectionStringName` должен соответствовать названию строки подключения сервера SQL Server, расположенной в разделе `connectionStrings` файла `web.config`.

Для того чтобы использовать в клиентском приложении созданный вами специальный провайдер профиля, необходимо определить расположение службы веб-параметров на вкладке Services дизайнера свойств проекта. Расположение должно быть таким же, как у служб роли и аутентификации: `http://localhost:12345/ApplicationServices`.



Рис. 33.3

Именно здесь особенно полезна поддержка среды разработки 2010 Visual Studio для параметров приложения. Если вы теперь перейдете на вкладку параметров дизайнера свойств проекта и щелкнете на кнопке Load Web Settings, вы сначала запросите аутентификационную информацию, поскольку для доступа к службе профиля необходимо в первую очередь проверить пользователя. На рис. 33.3 показано диалоговое окно с соответствующей аутентификационной информацией.

После ввода допустимого набора сертификатов запрашивается служба профиля и на поверхность проекта параметров добавляется новый ряд, как показано на рис. 33.4. Как можно заметить, область видимости этого параметра действительно `User` (`Web`) и значение по умолчанию, определенное в файле `web.config`, было получено.

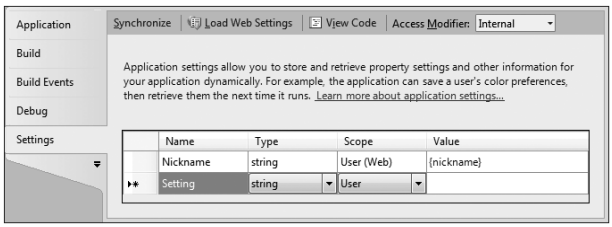


Рис. 33.4

Если вы обратите внимание на файл `app.config` приложения-клиента, то обнаружите, что к элементу `configSections` был добавлен новый элемент `sectionGroup`. Это просто объявляет класс, который будет использован для обработки специального раздела, который был добавлен для поддержки новых пользовательских параметров.

```
<configSections>
    <sectionGroup name="userSettings"
             type="System.Configuration.UserSettingsGroup, System,
             Version=4.0.0.0, Culture=neutral,
             PublicKeyToken=b77a5c5 61934e089" >
    <section name="ClientServices.Properties.Settings"
             type="System.Configuration.ClientSettingsSection, System,
             Version=4.0.0.0, Culture=neutral,
             PublicKeyToken=b77a5c561934e089" allowExeDefinition=
             "MachineToLocalUser" requirePermission="false" />
    </sectionGroup>
</configSections>
```

В конце файла `app.config` находится специальный раздел, который был создан. Как можно было ожидать, этот параметр носит название `Nickname`, и его значение соответствует значению по умолчанию, определенному в файле `web.config` проекта `ApplicationServices`.

```
<userSettings>
    <ClientAppServicesVB.MySettings>
        <setting name="Nickname" serializeAs="String">
            <value>{nickname}</value>
        </setting>
    </ClientAppServicesVB.MySettings>
</userSettings>
```

Чтобы использовать это в коде, можете применить тот же синтаксис, что и у любого другого параметра. Здесь вы просто возвращаете текущее значение, запрашиваете новое, а затем сохраняете его.



C#

```
private void Window_Loaded(object sender, RoutedEventArgs e){
    // Закомментировано для краткости
    MessageBox.Show(My.Settings.Nickname)
    Properties.Settings.Default.Nickname = "Not the default Name";
    My.Settings.Save()
```

Сниппет кода MainWindow.cs



VB

```
Private Sub Window_Loaded(ByVal sender As System.Object,
                          ByVal e As System.Windows.RoutedEventArgs) _
                          Handles Me.Loaded
    ' Закомментировано для краткости
    MessageBox.Show(My.Settings.Nickname)
    My.Settings.Nickname = InputBox("Please specify a nickname:",
"Nickname")
    My.Settings.Save()
```

Сниппет кода MainWindow.vb

Если запустить это приложение снова, то будет возвращен псевдоним, который вы предоставили вначале.

Форма регистрации

Ранее, при рассмотрении аутентификации с помощью форм, вы использовали при проверке пользователя жестко заданное имя пользователя и пароль. Хотя для приложения вполне возможно запрашивать у пользователя его аутентификационную информацию прежде, чем вызвать метод `ValidateUser` с предоставленными значениями, есть лучший способ использования инфраструктуры служб клиентского приложения. Вместо того чтобы вызывать метод `ValidateUser` с комбинацией имени пользователя и пароля, вернемся к поставке значения `Nothing` как аргумента и определим провайдер сертификатов; впоследствии службы клиентского приложения вызовут провайдер, чтобы определить набор аутентификационной информации для использования.



C#

```
private void Window_Loaded(object sender, RoutedEventArgs e){
    if (Membership.ValidateUser(null, null)){
        MessageBox.Show("User is valid");
```

Сниппет кода MainWindow.cs



VB

```
Private Sub Window_Loaded(ByVal sender As System.Object,
                          ByVal e As System.Windows.RoutedEventArgs) _
```

```
                                Handles Me.Loaded
    If Membership.ValidateUser(Nothing, Nothing) Then
        MessageBox.Show("User is valid")
```

Сниппет кода MainWindow.vb

Это может показаться сложней, так как создать провайдера сертификата относительно просто. Начнем с добавления к клиентскому приложению формы регистрации. Для этого выберите шаблон Login Form в диалоговом окне Add New Item и назовите его LoginForm. К сожалению, этот шаблон доступен только для разработчиков VB как форма Windows Forms. Если захотите создать версию WPF или будете работать в C #, то вам придется добавить новое окно в проект ClientServices и добавить элементы управления TextBox (по имени UsernameTextBox), PasswordBox (по имени PasswordTextBox) и две кнопки Button (OK и Cancel). Откройте дизайнер, щелкните на кнопке OK и измените значение свойства DialogResult на OK.

Для того чтобы использовать эту форму регистрации как провайдер сертификатов, измените ее так, чтобы реализовать интерфейс IClientFormsAuthentication-CredentialsProvider. Альтернативная стратегия подразумевала бы наличие отдельного класса, который реализует этот интерфейс, и последующее отображение формы регистрации при вызове метода GetCredentials. Следующий фрагмент содержит код для класса LoginForm, демонстрирующий реализацию интерфейса IClientForms-AuthenticationCredentialsProvider.



C#

```
using System.Web.ClientServices.Providers;
public partial class LoginForm : Window,
          IClientFormsAuthenticationCredentialsProvider {
    public LoginForm(){
        InitializeComponent();
    }

    private void OK_Click(object sender, RoutedEventArgs e){
        this.DialogResult = true;
        this.Close();
    }

    private void Cancel_Click(object sender, RoutedEventArgs e){
        this.DialogResult = false;
        this.Close();
    }

    public ClientFormsAuthenticationCredentials GetCredentials(){
        if (this.ShowDialog() ?? false) {
            return new ClientFormsAuthenticationCredentials(
                        UsernameTextBox.Text,
                        PasswordTextBox.
                        Password,
                        false);
        }
        else{
            return null;
        }
    }
}
```

Сниппет кода LoginForm.xaml.cs



VB

```
Imports System.Web.ClientServices.Providers

Public Class LoginForm
    Implements IClientFormsAuthenticationCredentialsProvider

    Public Function GetCredentials() As
ClientFormsAuthenticationCredentials _
           Implements
IClientFormsAuthenticationCredentialsProvider.GetCredentials
        If Me.ShowDialog() = Forms.DialogResult.OK Then
            Return New
ClientFormsAuthenticationCredentials(UsernameTextBox.Text,
                                     PasswordTextBox.Text,
                                                  False)
        Else
            Return Nothing
        End If
    End Function
End Class
```

Сниппет кода LoginForm.vb



Вы обратили внимание на то, что фрагменты кода C# и VB существенно отличаются? Это связано с тем, что язык C# использует новое окно WPF, в то время как фрагмент кода VB использует шаблон Windows Form Login Form.

Как можно заметить из этого фрагмента, после щелчка на кнопке Cancel метод `GetCredentials` возвращает объект `ClientFormsAuthenticationCredentials`, если поставляется регистрационная информация, или значение `Nothing` (на языке VB), или `null` (на языке C#) в противном случае. Безусловно, существует только один способ сбора аутентификационной информации, и нет никаких требований запрашивать эту информацию у пользователя. (Обычные альтернативы — использование аппаратных ключей или служебных удостоверений личности.)

Создание провайдера сертификатов — это только вопрос информирования служб клиентского приложения, что они должны использовать его. Для этого используется поле Optional: Credentials Provider вкладки Services дизайнера свойств проекта, как показано на рис. 33.5.

Теперь, когда вы запустите приложение, оно запросит ввести имя пользователя и пароль для доступа к нему. Затем эта информация передается провайдеру принадлежности на сервере, чтобы проверить пользователя.

Поддержка автономной работы

На предыдущих этапах, если у вас была точка прерывания в коде провайдера роли на сервере, вы, возможно, обратили внимание, что она достигалась только вначале, при запуске приложения. Дело в кэшировании информации роли в автономном режиме (offline). Если на вкладке Services дизайнера свойств проекта вы щелкнете на кнопке Advanced, то увидите множество свойств, которые могут быть откорректированы для контроля режима без подключения к сети (рис. 33.6).

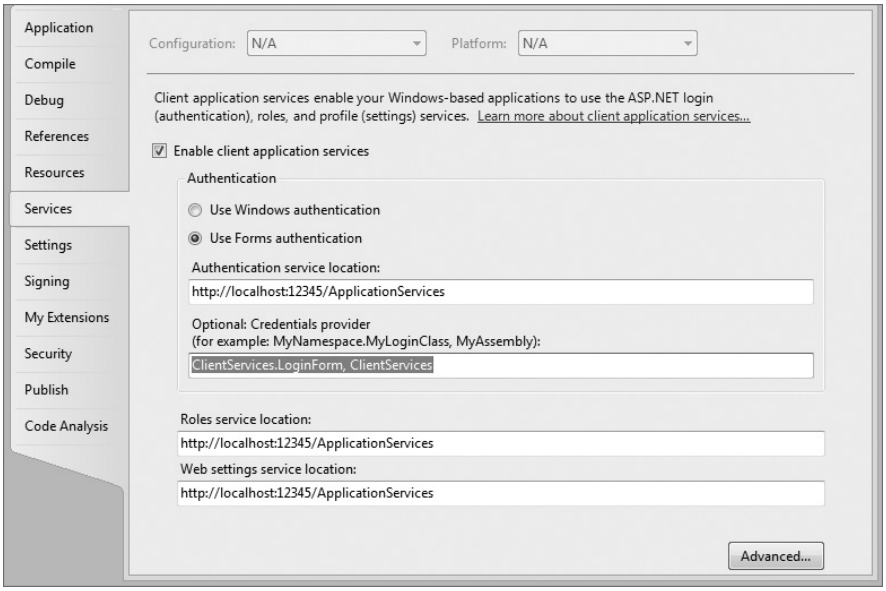


Рис. 33.5

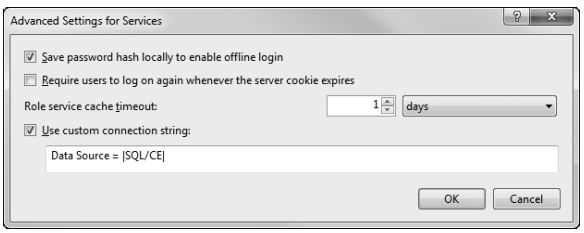


Рис. 33.6

Поле Role service cache timeout определяет, как часто сервер запрашивает информацию о роли. Поскольку это значение определяет максимальный период, который необходим для изменения роли, которые будут распространяться между подключенными клиентами, важно установить это свойство в соответствии с тем, как часто вы ожидаете изменений информации о роли. Безусловно, если приложение выполнится без подключения, то изменения будут получены при следующем подключении приложения (подразумевавший, что период кэширования будет превышен, пока приложение оставалось в автономном режиме).

Установка флажка Save Password Hash означает, что для регистрации пользователя приложение не должно быть подключено к сети. Хранимый хеш пароля используется только тогда, когда приложение работает в автономном режиме, в отличие от информации роли, для которой запрашивается кэш, если период не был превышен.

Подключено ли приложение к сети или нет, оно поддерживается службами клиентского приложения, поскольку полностью независимо от фактического подключения к сети или доступности сервера. В зависимости от вашего приложения может иметь смысл связать их друг с другом, как показано в следующем примере, где во время запуска приложения и при изменении сетевого статуса устанавливается статус от-

ключения. Щелкните во вкладке Application (VB) проектировщика свойств проекта на кнопке View Application Events или откройте файл App.xaml и добавьте обработчик для события Startup. В результате на экране появится файл кода, в который может быть вставлен код, приведенный ниже.



C#

```
using System.Net.NetworkInformation;
public partial class App : Application{
    private void Application_Startup(object sender, StartupEventArgs e){
        NetworkChange.NetworkAvailabilityChanged +=
        new NetworkAvailabilityChangedEventHandler
                            (NetworkChange_NetworkAvailabilityChanged);
        this.UpdateConnectivity();
    }

    private void NetworkChange_NetworkAvailabilityChanged(object sender,
                                    NetworkAvailabilityEventArgs e){
        this.UpdateConnectivity();
    }

    private void UpdateConnectivity(){
        System.Web.ClientServices.ConnectivityStatus.IsOffline =

!System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable();
    }
}
```

Сниппет кода Application.xaml.cs



VB

```
Class Application
    Private Sub MyApplication_Startup(ByVal sender As Object,
        ByVal e As System.Windows.StartupEventArgs) Handles Me.Startup
            AddHandler System.Net.NetworkInformation.NetworkChange.
            NetworkAvailabilityChanged, _
                        AddressOf MyApplication_NetworkAvailabilityChanged
            UpdateConnectivity()
    End Sub

    Private Sub MyApplication_NetworkAvailabilityChanged(
        ByVal sender As Object,
        ByVal e As
System.Net.NetworkInformation.NetworkAvailabilityEventArgs)
        UpdateConnectivity()
    End Sub

    Private Sub UpdateConnectivity()
        System.Web.ClientServices.ConnectivityStatus.IsOffline = Not _
        My.Computer.Network.IsAvailable()
    End Sub
End Class
```

Сниппет кода Application.xaml.vb

Вы, возможно, обратили внимание на то, что это весьма рудиментарный способ обнаружения подключения приложения к сети, и большинству приложений потребуется более сложная логика определения, фактически ли они подключены. Следует еще рассмотреть то, что когда приложение возвращается в режим подключения, может понадобиться подтвердить, что пользовательская информация все еще не устарела. Для этого используется метод RevalidateUser объекта ClientFormsIdentity (подходит только для аутентификации с помощью форм).

C#

```
(System.Threading.Thread.CurrentPrincipal.Identity as
          System.Web.ClientServices.ClientFormsIdentity).RevalidateUser()
```

VB

```
CType(System.Threading.Thread.CurrentPrincipal.Identity,
          System.Web.ClientServices.ClientFormsIdentity).RevalidateUser()
```

Последнее свойство в диалоговом окне Advanced определяет, где хранится кэшируемая информация сертификата и роли. Этот флажок установлен, поскольку ранее в примере мы решили использовать аутентификацию Windows. Если вы используете аутентификацию с помощью форм, сбросьте этот флажок. Службы клиентского приложения будут использовать файлы `.clientdata`, чтобы хранить данные каждого пользователя по адресу `Application.UserAppDataPath`, которым обычно является нечто вроде `C:\Users\Nick\AppData\Roaming\ClientServices\1.0.0.0` (в операционной системе Windows XP оно может быть другим). Использование специальной строки подключения позволяет использовать для хранения аутентификационной информации файл базы данных SQL Server Compact Edition (SSCE). Это потребует поддержки аутентификации Windows без подключения.



К сожалению, возможности проектировщика ограничены, и он не позволяет определять любые существующие подключения, которые у вас могут быть. Если вы измените файл `app.config`, то сможете настроить приложение так, чтобы использовать то же подключение.

Это может быть дополнительным преимуществом, поскольку свойство `|SQL/CE|` источника данных (являющееся значением по умолчанию) фактически позволяет службам клиентского приложения управлять созданием и установкой файла базы данных SSCE (в противном случае удостоверьтесь в наличии соответствующих таблиц).

Создаваемые файлы имеют расширение `.spf`, а не `.sdf`, — они все еще файлы базы данных SSCE, и вы можете просматривать их в среде разработки Visual Studio 2010.

Резюме

В этой главе вы узнали, как службы приложений ASP.NET Application Services могут быть расширены для использования с клиентскими приложениями. Обладая встроенной поддержкой функций отсутствия подключения, службы клиентских приложений позволяют создавать приложения, способные переходить между режимами отключения и подключения к сети. Совместно со службами синхронизации Microsoft ADO.NET Synchronization Services они обеспечивают инфраструктуру, необходимую для создания весьма сложных приложений без постоянного подключения.

34

Службы синхронизации

В ЭТОЙ ГЛАВЕ...

- Что такое приложение без постоянного подключения и зачем оно нужно
- Использование служб синхронизации для написания приложения без постоянного подключения
- Отделение служб синхронизации на нескольких уровнях
- Выполнение одино- и двунаправленной синхронизации

Приложения прошли долгий путь от автономных приложений, которые не используют данные совместно, до общедоступных веб-приложений, в которых все подключаются к тому же хранилищу данных. Впоследствии мы рассмотрим одноранговые приложения, в которых информация совместно используется узлами, но никакого централизованного хранилища данных нет. В корпоративном пространстве ключевые слова Software as a Service (SaaS) и Software and Services (S+S) определили переход от централизованных хранилищ данных, через сторонние данные и службы приложений, к гибридной модели, где данные и службы объединяются в пределах улучшенного приложения.

Одна из причин, по которым организации в прошлом склонилась к веб-приложениям, была необходимость рационализировать свои данные в единое централизованное хранилище. Хотя улучшенные клиентские приложения способны хорошо работать по сети с низким запаздыванием, используя то же хранилище данных, они быстро становятся непригодными, если каждое действие требует передачи данных между клиентом и сервером по медленной общей сети. Для того чтобы сократить это запаздывание, используется альтернативная стратегия, подразумевающая синхронизацию части хранилища данных на клиентской машине и применение локальных запросов данных. Это не только улучшает производительность, так как все запросы данных осуществляются локально, но и уменьшает нагрузку на сервер. В этой главе вы узнаете, как создавать приложения, которые лишь иногда подключаются к сети, и получать улучшенные интерактивные приложения, используя службы синхронизации Microsoft Synchronization Services for ADO.NET.

Приложения без постоянного подключения

Приложение без постоянного подключения (occasionally connected application) способно продолжать работу независимо от состояния подключения. Существует много разных способов получения доступа к данным, когда приложение не подключено к сети. Пассивные системы просто кэшируют данные, к которым они обращаются на сервере, чтобы при отсутствии подключения осталась доступна, по крайней мере, часть информации. К сожалению, эта стратегия означает, что доступным окажется очень ограниченный набор данных, и действительно подходящим этот подход окажется только в тех случаях, когда есть нестабильное или ненадежное подключение, а не в полностью автономном приложении. В последнем случае необходима активная система, которая синхронизирует данные с локальной системой. Службы синхронизации Microsoft Synchronization Services for ADO.NET (Sync Services) — это инфраструктура синхронизации, существенно упрощающая проблему синхронизации данных между любым сервером и локальной системой.

Прямое подключение к серверу



Рис. 34.1

Для ознакомления со службами синхронизации Sync Services воспользуемся простой базой данных, состоящей из одной таблицы, которая отслеживает клиентов. Ее можно создать с помощью окна Server Explorer в среде разработки Visual Studio 2010. Щелкните правой кнопкой мыши на узле Data Connections и в появившемся контекстном меню выберите команду Create New SQL Server Database. Диалоговое окно Create New SQL Server Database, в котором можно определить сервер и название новой базы данных, представлено на рис. 34.1.

Когда вы щелкнете на кнопке OK, база данных `CRM` будет добавлена к экземпляру сервера SQL Server Express, а подключение к данным будет добавлено в узел Data Connections окна Server Explorer. В узле Tables недавно созданного подключения к данным выберите команду Add New Table из контекстного меню, появляющегося после щелчка правой кнопкой мыши, и создайте столбцы `CustomerId`, `Name`, `Email` и `Phone`, чтобы таблица приобрела такой же вид, как на рис. 34.2.

Теперь, когда у вас есть простая база данных для работы, пришло время создать новое приложение Windows Forms. В данном случае приложение называется `QuickCRM`, и, как можно заметить в окне проводника решений на рис. 34.3, форма `Form1` переименована в `MainForm`, а также добавлены две дополнительные формы, `ServerForm` и `LocalForm`.

У формы `MainForm` есть две кнопки, как видно в области редактирования на рис. 34.3, и следующий код, для запуска соответствующих форм.

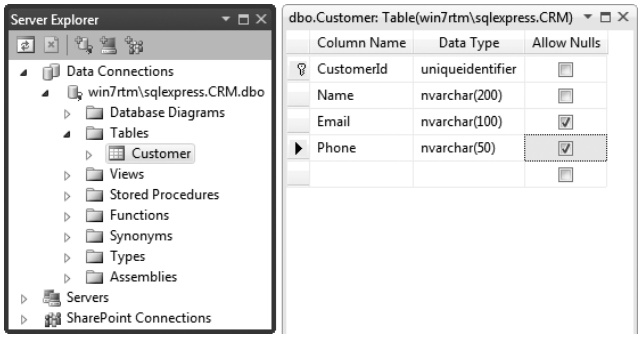


Рис. 34.2

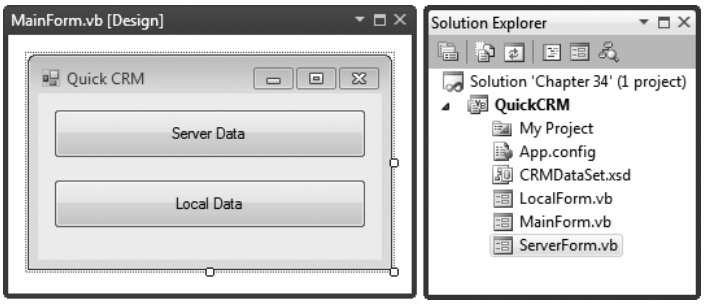


Рис. 34.3

VB

```
Public Class MainForm
    Private Sub ServerButton_Click(ByVal sender As System.Object,
                                   ByVal e As System.EventArgs) _
                                Handles ServerButton.Click
        My.Forms.ServerForm.Show()
    End Sub

    Private Sub LocalButton_Click(ByVal sender As System.Object,
                                  ByVal e As System.EventArgs) _
                                Handles LocalButton.Click
        My.Forms.LocalForm.Show()
    End Sub
End Class
```

Сниппет кода MainForm.vb

C#

```
public partial class MainForm : Form {
    public MainForm(){
        InitializeComponent();
    }

    private void ServerButton_Click(object sender, EventArgs e){
        (new ServerForm()).ShowDialog();
    }
    private void LocalButton_Click(object sender, EventArgs e){
        (new LocalForm()).ShowDialog();
    }
}
```

Сниппет кода MainForm.cs

Перед тем как рассмотреть использование служб синхронизации Sync Services для работы с локальными данными, изучим, как можно создать версию с постоянным подключением или подключенным сервером. В меню Data выберите пункт Add New Data Source и пройдите этапы мастера конфигурации источника данных Data Source Configuration Wizard, установив флажок DataSet и задействовав созданную ранее базу данных CRM, сохраняя строку подключения в файле конфигурации приложения и добавив таблицу Customer в CRMDataSet.

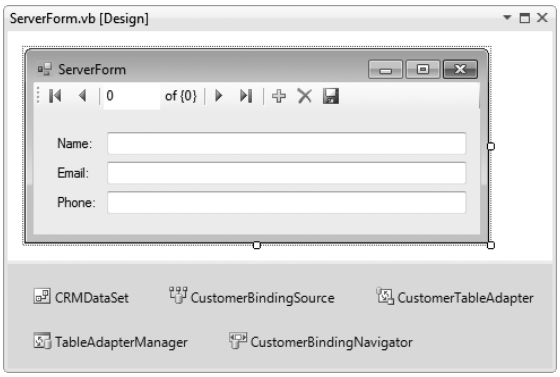


Рис. 34.4

Откройте дизайнер формы ServerForm, дважды щелкнув на ней в окне Solution Explorer. Если окно Data Sources еще не открыто, выберите команду Show Data Sources в меню Data. Используя контекстное меню узла Customer, выберите команду Details, а затем команду None из узла CustomerId. Перемещение узла Customer на рабочую область проекта ServerForm добавляет соответствующие элементы управления, чтобы вы могли находить, редактировать и сохранять записи в таблице Customer базы данных CRM, как показано на рис. 34.4.

Как заявлено в определении таблицы, столбец CustomerId не может быть пуст, поэтому необходимо гарантировать, что все новые записи будут иметь идентификатор. Для этого пригодится обработчик события CurrentChanged объекта CustomerBindingSource. Вы можете обратиться к нему либо непосредственно в коде формы ServerForm, либо выбрать объект CustomerBindingSource и найти событие CurrentChanged в окне Properties.

VB

```
Private Sub CustomerBindingSource_CurrentChanged _
                (ByVal sender As System.Object,
                 ByVal e As System.EventArgs) _
                        Handles CustomerBindingSource.CurrentChanged
    If Me.CustomerBindingSource.Current Is Nothing Then
        Return
    End If

    Dim c As CRMDataSet.CustomerRow =
CType(CType(Me.CustomerBindingSource.Current,
                        DataRowView).Row,CRMDataSet.CustomerRow)
    If c.RowState = DataRowState.Detached Then
        c.CustomerId = Guid.NewGuid
    End If
End Sub
```

Сниппет кода ServerForm.vb

C#

```
private void customerBindingSource_CurrentChanged(object sender,
EventArgs e){
    if (this.customerBindingSource.Current == null){
        return;
    }
```

```
        var c = (this.customerBindingSource.Current as DataRowView)
                    .Row as CRMDataSet.CustomerRow;
        if (c.RowState == DataRowState.Detached){
            c.CustomerId = Guid.NewGuid();
        }
    }
```

Сниппет кода ServerForm.cs

На этом заканчивается часть приложения, которая непосредственно подключается к базе данных, чтобы получить доступ к данным. Запустите приложение и удостоверьтесь в том, что можете обращаться к данным при подключении к базе данных. Если база данных отключена или нет подключения, приложение передаст исключение, когда вы попытаетесь получить данные из базы данных или сохранить внесенные изменения.

Приступая к работе со службами синхронизации

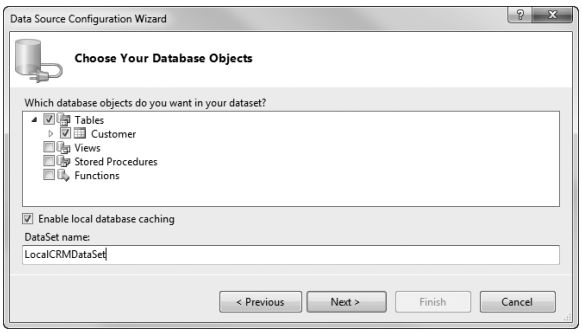


Рис. 34.5

Для начала работы со службами синхронизации Sync Services необходимо добавить в проект элемент локального кэша базы данных Local Database Cache. Прежде для этого использовалось диалоговое окно Add New Item. Однако в среде разработки Visual Studio 2010 можно использовать для этого тот же подход, что и при добавлении источника данных, который подключен к базе данных SQL Server. Запустите мастер конфигурации источника данных Data Source Configuration Wizard, выбрав команду Add New Data Source в окне Data Sources или в меню Data. Пройдите этапы мастера, определив строку подключения и выбрав таблицу `Customer`. В заключительном окне, где вы определяете название создаваемого набора данных, установите флажок Enable Local Database Caching, как показано на рис. 34.5.

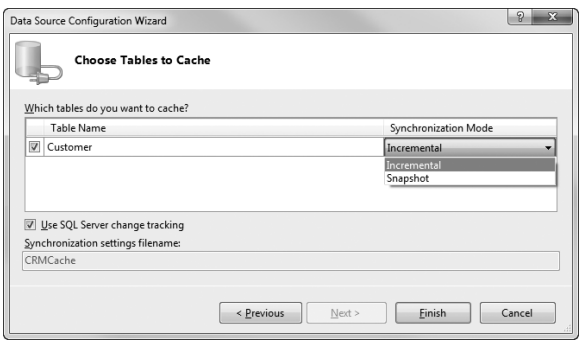


Рис. 34.6

Щелкните на кнопке Next, чтобы перейти к следующему этапу мастера, который позволит настроить способ синхронизации данных между сервером и вашим локальным кэшем базы данных. Как видно на рис. 34.6, для каждой таблицы можно выбрать режим синхронизации Incremental или Snapshot. Первый лучше для таблиц, содержащих большое количество данных, которые изменяются часто; последний — для таблиц, содержащих мало данных, которые изменяются нечасто и не требуют отслеживания изменений.

Другая возможность, представленная на рис. 34.6, разрешает отслеживание изменений SQL Server. Службы синхронизации Sync Services полагаются на способность отслеживать изменения данных, чтобы обеспечить синхронизацию между сервером и клиентом. Для этого предоставляются два механизма. Либо разрешите отслеживание изменений, когда изменения на сервере автоматически отслеживаются базой данных SQL Server, либо настройте службы синхронизации Sync Services так, чтобы отслеживать изменения в таблицах базы данных. Первый доступен только с сервером SQL Server 2008, а последний требует дополнительных полей, триггеров и таблицы, чтобы обеспечить эквивалентные возможности отслеживания изменений. Если вы собираетесь установить базу данных к SQL Server 2008, рекомендуется разрешить отслеживание изменений.

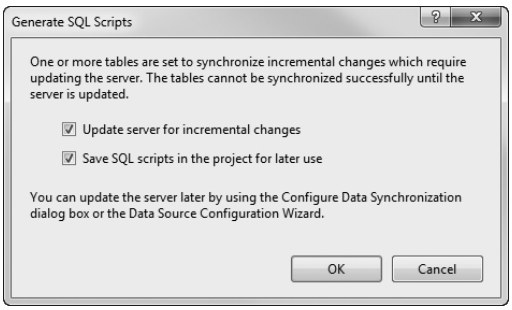


Рис. 34.7

Когда вы щелкаете на кнопке Finish, мастер запросит подтверждения, что вы хотите применить изменения сервера немедленно (рис. 34.7). Если вы работаете над совместно используемой базой данных, просмотрите созданные сценарии, прежде чем позволить им выполняться. В данном примере оставьте оба флажка установленными, чтобы создать сценарии базы данных (включая сценарии отмены) и добавить их в ваш проект, а также запустить их в базе данных сервера, чтобы либо разрешить отслеживание изменений, либо создать дополнительные столбцы отслеживания изменений, триггеры и таблицы.

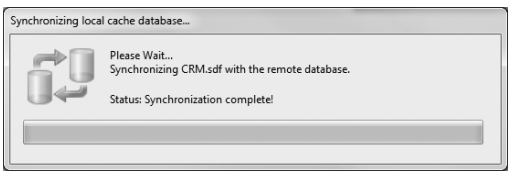


Рис. 34.8

Щелчок на кнопке OK сохранит эту конфигурацию в форме класса синхронизации и запустит синхронизацию между сервером и локальным файлом данных, как показано на рис. 34.8.

Принудительная синхронизация на настоящий момент означает, что недавно созданный файл базы данных SQL Server Compact (SSC) заполняется всеми корректными данными и схемами, доступными на сервере. В ваш проект добавляется также узел `LocalCRMDataSet`.

Если вы посмотрите теперь на окно Data Sources, то увидите, что узел `LocalCRMDataSet` содержит узел `Customer`. Установите для узла `Customer` значение `Details`, а для узла `CustomerId` — `None`. Затем перетащите узел `Customer` на поверхность проекта `LocalForm`. В результате должна получиться форма, подобная представленной на рис. 34.9.

Добавление этих компонентов привносит в проект формы те же компоненты и тот же код, что и при непосредственном подключении к серверу. Различие здесь в том, что адаптер `CustomerTableAdapter` соединяется с локальной базой данных вместо сервера. Как и прежде, необходимо добавить код определения идентификатора `CustomerId` новых записей в обработчике события `CurrentChanged` объекта `CustomerBindingSource`.

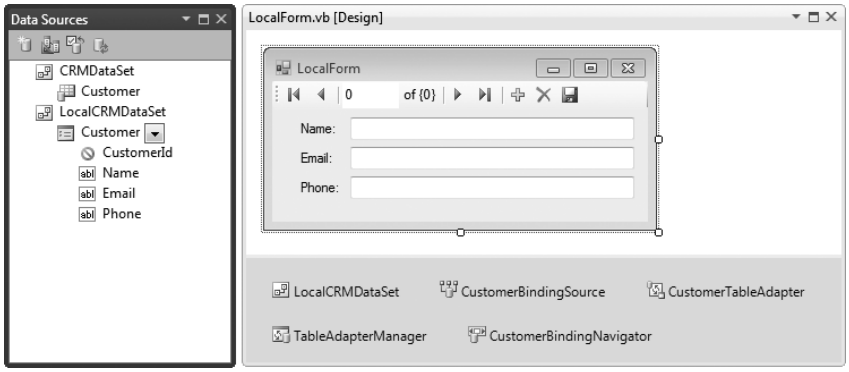


Рис. 34.9

Последнее, что необходимо сделать, — это добавить в данную часть проекта механизм запуска процесса синхронизации. Просто добавьте кнопку `SynchronizeButton` в нижнюю часть формы `LocalForm` и дважды щелкните на ней, чтобы создать обработчик события щелчка. Затем добавьте следующий код запуска синхронизации.

VB

```
Private Sub SynchronizeButton_Click(ByVal sender As System.Object,
                                    ByVal e As System.EventArgs)
                                        Handles SynchronizeButton.Click
    Dim syncAgent As New CRMCacheSyncAgent()
    Dim syncStats As Microsoft.Synchronization.Data.SyncStatistics =
                                        syncAgent.Synchronize()

    Me.CustomerTableAdapter.Fill(Me.LocalCRMDataSet.Customer)
End Sub
```

Сниппет кода LocalForm.vb

C#

```
private void SynchronizeButton_Click(object sender, EventArgs e){
    var syncAgent = new CRMCacheSyncAgent();
    var syncStats = syncAgent.Synchronize();

    this.customerTableAdapter.Fill(this.localCRMDataSet.Customer);
}
```

Сниппет кода LocalForm.cs

Обратите особое внимание на предпоследнюю строку этого фрагмента, в которой вы используете адаптер `CustomerTableAdapter` для заполнения таблицы `Customer`. Это важно, так как без такой строки пользовательский интерфейс не будет отображать изменения в базе данных SSC, которые были внесены процессом синхронизации.

Службы синхронизации на нескольких уровнях

До сих пор весь процесс синхронизации проводился в пределах клиентского приложения с прямым подключением к серверу. Одна из задач приложения без постоянного подключения заключается в способности синхронизации данных по любому подключению, будь то корпоративный интранет или общедоступный Интернет.

К сожалению, в текущем приложении необходимо предоставить доступ к серверу SQL Server, чтобы приложение могло соединиться с ним. Это явная брешь в системе безопасности, которую вы можете закрыть, применив весьма распространенный подход. Службы синхронизации Sync Services были разработаны с учетом этого, позволяя изолировать компоненты сервера в службе, которая может быть вызвана во время синхронизации.

Службы синхронизации Sync Services обеспечивают отделение процесса синхронизации, чтобы клиентское приложение общалось со службой WCF, а не непосредственно с базой данных сервера. Для этого необходимо создать службу WCF, которая реализует четыре метода, составляющих службу синхронизации, как показано в следующем интерфейсе `IServiceCRMCacheSyncContract`.

VB

```
<ServiceContractAttribute()> _
Public Interface IServiceCRMCacheSyncContract
    <OperationContract()> _
    Function ApplyChanges(ByVal groupMetadata As SyncGroupMetadata, _
                          ByVal dataSet As DataSet, _
                          ByVal syncSession As SyncSession) As SyncContext
    <OperationContract()> _
    Function GetChanges(ByVal groupMetadata As SyncGroupMetadata, _
                        ByVal syncSession As SyncSession) As SyncContext
    <OperationContract()> _
    Function GetSchema(ByVal tableNames As Collection(Of String), _
                       ByVal syncSession As SyncSession) As SyncSchema
    <OperationContract()> _
    Function GetServerInfo(ByVal syncSession As SyncSession)_
                           As SyncServerInfo
End Interface
```

По существу, служба WCF действует как дистанционный прокси-сервер для провайдера сервера, используемого службой синхронизации Sync Service. Чтобы использовать службу WCF, сначала необходимо добавить ее в проект клиента. Для этого щелкните правой кнопкой мыши на проекте и выберите в контекстном меню пункт Add Service Reference. Затем необходимо установить дистанционный провайдер Remote Provider агента синхронизации Sync Agent как новый экземпляр прокси-сервера `ServerSyncProviderProxy`. Конструктор класса `ServerSyncProviderProxy` получает один параметр, который должен быть прокси-классом, созданным для службы WCF Service с помощью пункта контекстного меню Add Service Reference. Теперь при вызове метода `Synchronize` служба синхронизации Sync Services будет использовать дистанционный провайдер Remote Provider для вызова методов службы WCF. Служба WCF будет в свою очередь общаться с базой данных сервера, выполняя логику синхронизации.

Фоновая синхронизация

Вы, возможно, обратили внимание на то, что после щелчка на кнопке синхронизации пользовательский интерфейс как бы зависает, пока синхронизация не заканчивается. Безусловно, в реальном приложении это неприемлемо, поэтому необходимо синхронизировать данные в фоновом режиме, чтобы позволить пользователю продолжить работу. Добавив компонент `BackgroundWorker` (из группы Components панели инструментов) в форму `LocalForm`, вы можете сделать это при минимальном изменении вашего прило-

жения. Следующий код иллюстрирует, как можно связать событие bgWorker компонента BackgroundWorker с реализацией службы синхронизации Sync Service. Для этого воспользуемся дополнительной кнопкой, SynchronizeInBackgroundButton, которую добавим в форму LocalForm.



VB

```
Private Sub SynchronizeInBackgroundButton_Click(ByVal sender As_
                                        System.Object,
                                        ByVal e As System.EventArgs) _
                Handles SynchronizeInBackgroundButton.Click
    Me.SynchronizeButton.Enabled = False
    Me.SynchronizeInBackgroundButton.Enabled = False
    Me.bgWorker.RunWorkerAsync(New CRMCacheSyncAgent())
End Sub

Private Sub bgWorker_DoWork(ByVal sender As System.Object,
                    ByVal e As System.ComponentModel.DoWorkEventArgs) _
                              Handles bgWorker.DoWork
    Dim syncAgent As Microsoft.Synchronization.SyncAgent =
                TryCast(e.Argument, Microsoft.Synchronization.SyncAgent)
    If syncAgent Is Nothing Then Return
    syncAgent.Synchronize()
End Sub

Private Sub bgWorker_RunWorkerCompleted(ByVal sender As System.Object, _
         ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs) _
                    Handles bgWorker.RunWorkerCompleted
    Me.CustomerTableAdapter.Fill(Me.LocalCRMDataSet.Customer)
    Me.SynchronizeInBackgroundButton.Enabled = True
    Me.SynchronizeButton.Enabled = True
End Sub
```

Сниппет кода LocalForm.vb



C#

```
private void SynchronizeInBackgroundButton_Click(object sender, EventArgs e){
    this.SynchronizeButton.Enabled =false;
    this.SynchronizeInBackgroundButton.Enabled = false;
    this.bgWorker.RunWorkerAsync(new CRMCacheSyncAgent());
}

private void bgWorker_DoWork(object sender, DoWorkEventArgs e){
    var syncAgent = e.Argument as Microsoft.Synchronization.SyncAgent;
    if (syncAgent == null) return;
    syncAgent.Synchronize();
}

private void bgWorker_RunWorkerCompleted(object sender,
                        RunWorkerCompletedEventArgs e){
    this.customerTableAdapter.Fill(this.localCRMDataSet.Customer);

    this.SynchronizeInBackgroundButton.Enabled = true;
    this.SynchronizeButton.Enabled = true;
}
```

Сниппет кода LocalForm.cs

В этом фрагменте кода вы не сообщаете о ходе процесса, но служба синхронизации действительно обеспечивает весьма передовую модель событий, которую можно использовать для индикации процесса синхронизации. Если хотите оповещать о ходе процесса с помощью компонента BackgroundWorker, необходимо задействовать

свойство WorkerReportsProgress. Следующий код иллюстрирует, как можно использовать событие ApplyChanges на клиентском компоненте служб синхронизации Sync Services для индикации процесса (в данном случае в метке SyncProgressLabel, добавленной к форме). Другие события соответствуют различным пунктам процесса синхронизации.



VB

```
Private Sub bgWorker_DoWork(ByVal sender As System.Object, _
         ByVal e As System.ComponentModel.DoWorkEventArgs) _
    Handles bgWorker.DoWork
    Dim syncAgent As Microsoft.Synchronization.SyncAgent = _
            TryCast(e.Argument, Microsoft.Synchronization.SyncAgent)
    If syncAgent Is Nothing Then Return
    Dim clientProvider As _

Microsoft.Synchronization.Data.SqlServerCe.SqlCeClientSyncProvider = _
        CType(syncAgent.LocalProvider, _

Microsoft.Synchronization.Data.SqlServerCe.SqlCeClientSyncProvider)
    AddHandler clientProvider.SyncProgress, AddressOf SyncProgress
    syncAgent.Synchronize()
End Sub
Private Sub SyncProgress(ByVal sender As Object, _
        ByVal e As Microsoft.Synchronization.Data.SyncProgressEventArgs)
    Dim progress = 0
    If (e.GroupProgress.TotalChanges > 0) Then
        progress = (e.GroupProgress.TotalChanges -
                    e.GroupProgress.TotalChangesPending) _
                    * 100 / e.GroupProgress.TotalChanges
    End If
    Me.bgWorker.ReportProgress(progress, e.SyncStage.ToString())
End Sub

Private Sub bgWorker_ProgressChanged(ByVal sender As Object, _
          ByVal e As System.ComponentModel.ProgressChangedEventArgs) _
                            Handles bgWorker.ProgressChanged
    Me.SyncProgressLabel.Text = e.UserState.ToString
End Sub
```

Сниппет кода LocalForm.vb



C#

```
private void bgWorker_DoWork(object sender, DoWorkEventArgs e){
    var syncAgent = e.Argument as Microsoft.Synchronization.SyncAgent;
    if (syncAgent == null) return;
    var clientProvider = syncAgent.LocalProvider as

Microsoft.Synchronization.Data.SqlServerCe.SqlCeClientSyncProvider;
    clientProvider.SyncProgress += SyncProgress;

    syncAgent.Synchronize();
}

private void SyncProgress(object sender,
                Microsoft.Synchronization.Data.SyncProgressEventArgs e){
    var progress = 0;
    if(e.GroupProgress.TotalChanges>0){
        progress = (e.GroupProgress.TotalChanges -
                    e.GroupProgress.TotalChangesPending)
                    *100 /e.GroupProgress.TotalChanges;
```

```
    }
    this.bgWorker.ReportProgress(progress, e.SyncStage.ToString());
}

private void bgWorker_ProgressChanged(object sender,
ProgressChangedEventArgs e){
    this.SyncProgressLabel.Text = e.UserState.ToString();
}
```

Сниппет кода LocalForm.cs

Изменения клиента

Работая с данным примером, вы, возможно, задавали вопрос: почему ни одно из изменений, которые были внесены на клиенте, не синхронизируются на сервере? Если вернуться к рис. 34.6, то можно заметить, что был выбран режим синхронизации Incremental, который подразумевает синхронизацию изменений между клиентом и сервером. Но дело обстоит не так, и это подтверждает формулировка данного элемента управления, приведенная выше. В любом случае этот элемент управления позволяет только выбирать параметры, имеющие отношение к загружаемым данным. Чтобы обеспечить передачу изменений в обоих направлениях, необходимо переопределить стандартный режим каждой таблицы, которая подлежит синхронизации. Снова щелкните правой кнопкой мыши в проводнике решений на объекте `CRMCache` и в появившемся контекстном меню выберите пункт View Code. В следующем коде установите для свойства `SyncDirection` таблицы `CustomerSyncTable` двунаправленный режим. Можете также сделать это для элемента `ServerCRMCache`, чтобы оба механизма синхронизации позволили передавать изменения между клиентом и сервером.



VB

```
Partial Public Class CRMCacheSyncAgent
    Partial Class CustomerS yncTable
        Private Sub OnInitialized()
            Me.SyncDirection = _
               Microsoft.Synchronization.Data.SyncDirection.Bidirectional
        End Sub
    End Class
End Class
```

Сниппет кода CRMCache.vb



C#

```
partial class CRMCacheSyncAgent{
    partial class CustomerSyncTable{
        private void OnInitialized(){
            this.SyncDirection =
              Microsoft.Synchronization.Data.SyncDirection.Bidirectional;
        }
    }
}
```

Сниппет кода CRMCache.cs

Если бы вы синхронизировали другие таблицы, то должны были бы установить свойства `SyncDirection` каждой в соответствии с `SyncTables`. Альтернативная реализация подразумевала бы помещение этого кода в метод `OnInitialized` самого агента `SyncAgent` и установку свойства `SyncDirection` каждой синхронизируемой

таблицы (например, `CustomerSyncTable`). Какой бы способ вы ни выбрали, вы должны применить значение `Bidirectional` ко всем таблицам, которые хотите синхронизировать в обоих направлениях.

Резюме

В этой главе вы узнали, как использовать службы синхронизации Microsoft Synchronization Services for ADO.NET для создания приложения без постоянного подключения. Хотя у вас могут быть и другие вопросы при создании такого приложения, такие, например, как обнаружение сетевого подключения, выполнение синхронизации как фоновой задачи и разделение компонентов клиента и сервера на разные уровни приложения. Обладая этими знаниями, можете начать работать с этой новой технологией создания улучшенных приложений, которые продолжат работать независимо от места их использования.

35

Службы WCF RIA

В ЭТОЙ ГЛАВЕ...

- Понятие служб WCF RIA Services
- Создание службы домена
- Предоставление данных
- Использование служб WCF RIA Services в приложениях Silverlight

В главе 31 упоминалось о том, что платформа WCF предоставляет стандартные средства коммуникации технологически агностическим способом. Службы WCF RIA Services (обычно называемые просто службами RIA Services) являются надстройкой платформы WCF, обеспечивающей шаблон и инфраструктуру для разработки управляемых данными приложений, которые используют данные с сервера. В настоящее время службы WCF RIA Services ориентированы на приложения Silverlight, но с расчетом поддержки дополнительных технологий представления. Эта глава рассматривает использование службы WCF RIA Services для создания сквозных приложений Silverlight.

Первые шаги

Службы RIA Services ориентированы в основном на технологию Silverlight для клиентской платформы, поэтому начнем с создания проекта Silverlight. Найдите в разделе Silverlight шаблон приложения Silverlight Business Application (рис. 35.1), который создаст всю структуру решения, необходимую для запуска служб RIA Services (проект Silverlight, веб-приложение ASP.NET и связь RIA Services между ними).

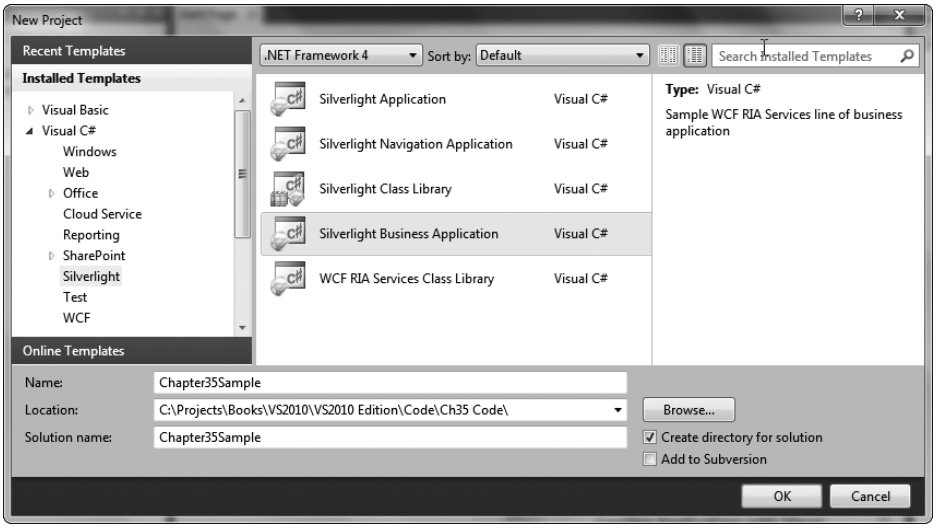


Рис. 35.1

В результате будут созданы проект Silverlight и проект ASP.NET со структурой, представленной на рис. 35.2.

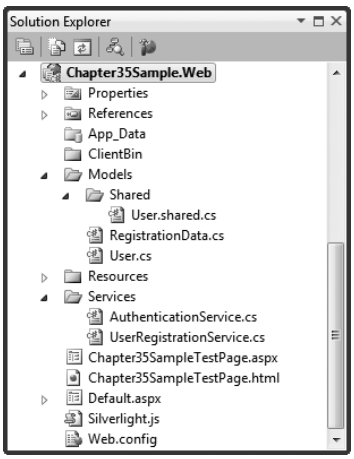


Рис. 35.2

Шаблон RIA Services обеспечивает проекту ASP.NET поддержку и реализацию нескольких базовых функциональных возможностей. Обратите внимание на рис. 35.2: в проекте есть папки `Services` и `Models`. Папка `Services` уже содержит две службы домена (`AuthenticationService` и `UserRegistrationService`), чтобы обеспечить клиенту операции аутентификации и регистрации пользователя. Папка `Models` содержит два класса данных (`User` и `RegistrationData`), передаваемых между сервером и клиентом. В папке `Models` находится также папка `Shared`, в которой расположен файл `User.shared.vb` или `User.shared.cs`, содержащий код, совместно используемый серверным и клиентским проектами.

Как упоминалось в главе 22, проекты Silverlight и ASP.NET связаны таким образом, что приложение Silverlight копируется куда-нибудь в проект ASP.NET, когда проект (решение) компилируется (настраивается в свойствах проекта ASP.NET). Однако при введении служб RIA Services картина совершенно меняется: теперь будет другая связь между проектами. Эта связь настраивается в свойствах проекта Silverlight выбором проекта ASP.NET, который будет действовать как сервер, с которым он будет общаться и на основании которого служба RIA Services создаст код (рис. 35.3).

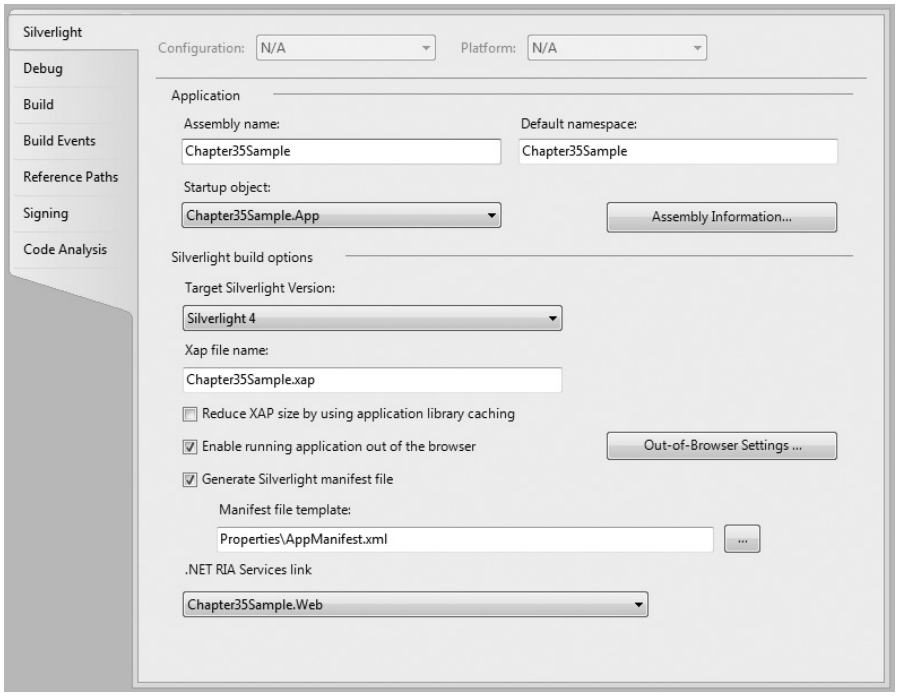


Рис. 35.3

При использовании шаблона проекта Business Application эта связь уже будет установлена; но если у вас есть существующий проект Silverlight или веб-приложение, в котором вы хотите использовать службы RIA Services, понадобится вручную соединить проекты, связав проект Silverlight с проектом ASP.NET с помощью этой возможности.

Теперь вы готовы написать некоторый код. Пример, разрабатываемый в этой главе, демонстрирует некоторые из ключевых концепций шаблона RIA Services. Ваша цель — предоставить клиентские данные и операции на сервере и сделать их доступными на клиенте.

Вы будете использовать модель Entity Framework базы данных `AdventureWorksLT`, которую создали в главе 29. В большинстве случаев нет необходимости передавать сущности вашей модели между сервером и клиентом, так как это считается плохой практикой. Сущности образуют модель вашего слоя данных, которой в принципе уровень представлений может быть недоступен. Передаете ли вы сущности или объекты POCO (Plain Old CLR Objects — старые простые объекты CLR) для своего проекта (которые в службах RIA Services называются *типами модели представления*), решение, которое вы должны будете принять, зависит от многих факторов. Использование сущностей ускорит разработку, но также снизит гибкость по сравнению с использованием типов модели представления. Службы RIA Services работают точно так же, используя типы модели представления, несмотря на большее количество работы, затрачиваемое вначале на создание служб домена Domain Services. Поэтому наилучшая практика подразумевает использование типов модели представления как механизм передачи данных; но мы сосредоточимся в этой главе на использовании сущностей, поскольку они предоставляют самые простые средства для начала разработки.

Службы домена

Теперь, когда у вас есть свои проекты сервера и клиента, соединенные через службы RIA Services, пришло время предоставить на сервере некоторые данные и операции, которые вы впоследствии будете использовать на своем клиенте.

Для начала предположим, что модель Entity Framework базы данных `AdventureWorksLT` из главы 29 была добавлена в ваш проект ASP.NET (включая добавление строки подключения, которая используется для файла `web.config`). В противном случае сделайте это сейчас.



Откомпилируйте проект ASP.NET перед переходом к следующему этапу, в противном случае мастер класса службы домена Domain Service Class Wizard не отобразит вашу модель сущности среди доступных в раскрывающемся списке DataContexts/ObjectContexts.

Для того чтобы предоставить данные клиента из вашей модели сущности, необходимо добавить службу домена в ваш проект ASP.NET. Лучшее место в проекте для добавления этой службы — папка Services. Добавьте в эту папку новый элемент и выберите в разделе Web шаблон элемента Domain Service Class (рис. 35.4). Вы будете использовать эту службу для обслуживания данных клиента, поэтому назовите его `CustomersService` (`.cs` или `.vb`).

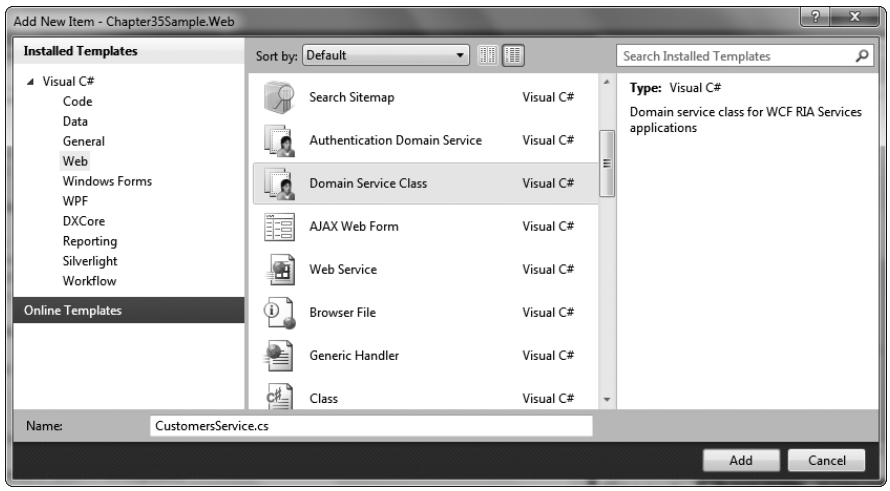


Рис. 35.4

Щелчок на кнопке OK инициализирует мастер Domain Service Class Wizard, как показано на рис. 35.5.

Если модель Entity Framework не выбрана автоматически для базы данных `AdventureWorksLT` в раскрывающемся списке Available DataContexts/ObjectContexts, выберите его сейчас. В этом списке будут отображены все сущности вашей модели сущностей. Здесь можете выбрать одну или несколько сущностей, которые хотите предоставить из вашей службы домена. После того как вы выберете сущность, мастер создаст операцию домена, возвращающую коллекцию этой сущности из службы домена. Если выбрать для сущности параметр Enable Editing, мастер создаст также для этой сущности на службе домена операции `Insert`, `Update` и `Delete`.

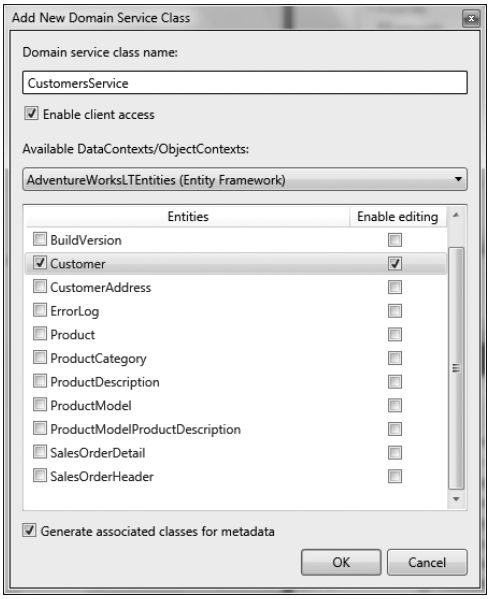


Рис. 35.5



Если вместо типов Entity Framework или LINQ-to-SQL вы используете типы модели представления или POCO, выберите из раскрывающегося списка Available DataContexts/ObjectContexts пункт <empty domain service class> и реализуйте операции домена самостоятельно.

Вам нужно удостовериться, что флажок разрешения доступа клиента установлен. Это гарантирует, что атрибут `EnableClientAccess` применяется к службе при ее создании, а значит, код будет создан на клиенте генератором приложений RIA Services, чтобы обеспечить ему доступ к службе домена.

Вы, наверное, обратили внимание на то, что в мастере есть также флажок Generate associated classes for metadata. Классы метаданных позволяют добавлять специальные атрибуты к свойствам класса передаваемых данных (так как на клиенте должны быть созданы сущность и правила проверки данных) без необходимости изменять исходный объект (это важно, когда, например, вы регенерируете код своего объекта с помощью генератора приложений или ORM). Вместо этого можете применить атрибуты к свойствам класса метаданных, которые соответствуют свойствам фактического класса. Эти атрибуты управляют созданием ассоциированной сущности на клиенте и применением других атрибутов (такие как правила проверки) к создаваемой сущности. Подробнее о классах метаданных см. в главе 23.

Создавать классы метаданных для вашей сущности не обязательно. Тем не менее это действительно повышает степень контроля над передачей данных между сервером и клиентом, поэтому мы все же рекомендуем это делать.

Выберите сущность `Customer`, установите для нее флажок Enable Editing и удостоверьтесь, что установлены оба флажка — Enable client access и Generate associated classes for metadata. Щелчок на кнопке OK автоматически создает службу домена и классы метаданных.

Операции домена

Операции домена — это операции на службе домена, которая может быть вызвана с клиента. Типы операций домена, существующие на службе домена, можно рассматривать как операции CRUD (Create, Read, Update, Delete), операции вызова или специальные операции.

Названия и сигнатуры методов этих операций основаны на соглашении, таким образом, службы RIA Services могут неявно определить, какие это типы операций, и правильно создавать соответствующую операцию в клиентском проекте. Если по некоторым причинам вы не хотите использовать данные соглашения, можете пометить операцию (т.е. метод) атрибутом, чтобы указать, какая операция представляется.



Некоторые разработчики предпочитают помечать свои операции, даже когда они следуют соглашения имени/сигнатуры, чтобы явно указать, какая операция представляется.

Теперь обратим внимание на то, какие операции домена были автоматически созданы мастером в вашей службе домена и какие другие типы операций вы можете создать.

Операции запроса

Когда вы откроете файл `CustomersService` (`.cs` или `.vb`), вы увидите, что для сущности `Customer` были реализованы основные операции CRUD. Стандартная операция `Read` (`Get` или `Query`) возвращает коллекцию сущностей со следующей сигнатурой метода.

VB

```
Public Function GetCustomers() As IQueryable(Of Customer)
```

C#

```
public IQueryable<Customer> GetCustomers()
```

Операция `GetCustomers` возвращает коллекцию `IQueryable` сущности `Customer`. Это одно из самых мощных достоинств служб RIA Services, позволяющих записать на клиенте запрос LINQ, который применяется для фильтрации и формирования сущностей, подлежащих возвращению. Этот запрос LINQ фактически сериализируется и посылается на сервер, прежде чем выполняется. Всю мощь этой возможности вы увидите тогда, когда вам потребуется реализовать на клиенте фильтрацию, разделение, группировку и сортировку. Вместо обязательной реализации на сервере сложных операций этих режимов вы нуждаетесь только в одной простой операции, которая возвращает коллекцию `IQueryable`, а для последующей фильтрации и формирования результатов на сервере перед возвращением клиент может предоставить запрос LINQ. В качестве альтернативы можете изменить операцию `Get` и добавить для нее собственные параметры, которые операция может использовать для фильтрации и формирования результатов, возвращаемых клиенту.

Операции вставки, модификации и удаления

Операции вставки (или создания), обновления и удаления вызываются автоматически, когда вы передаете на сервер набор изменений (на основании действия, предпринятого при получении результата операции запроса на клиенте) и не можете вы-

звать их явно с клиента. Эти действия рассматриваются далее в главе, а пока обратим внимание на операции, которые были автоматически реализованы мастером Domain Service Class Wizard, и их реализацию. У автоматически созданных операций есть следующие сигнатуры методов.

VB

```
Public Sub InsertCustomer(ByVal customer As Customer)
Public Sub UpdateCustomer(ByVal currentCustomer As Customer)
Public Sub DeleteCustomer(ByVal customer As Customer)
```

C#

```
public void InsertCustomer(Customer customer)
public void UpdateCustomer(Customer currentCustomer)
public void DeleteCustomer(Customer customer)
```

Каждая из них получает сущность соответствующего типа и выполняет на сервере соответствующее действие с ней. У каждой из них есть соглашение об именовании, они не должны возвращать значение, и их сигнатуры методов должны получать сущность как единственный параметр. Соглашение об именовании и альтернативный атрибут приведены ниже.

- Название метода операции вставки должно начинаться с `Insert`, `Create` или `Add`. В противном случае примените к методу атрибут `Insert`.
- Название метода операции модификации должно начинаться с `Update`, `Change` или `Modify`. В противном случае примените к методу атрибут `Update`.
- Название метода операции удаления должно начинаться с `Delete` или `Remove`. В противном случае примените к методу атрибут `Delete`.

Операции других типов

Ниже описаны другие типы операций, поддерживаемых службами RIA Services (но не полностью детализированные здесь).

- Операция вызова. По существу, то же, что и служебная операция в стандартной службе WCF Service (т.е. метод, предоставленный службой). Операции вызова создаются тогда, когда методы в контексте домена на клиенте вызываются немедленно (т.е. они не становятся в очередь, пока изменения не будут переданы серверу).
- Специальная операция. Вызывается на клиенте в любой момент, но ее выполнение откладывается на сервере до передачи изменений. Специальные операции воздействуют на сущность и фактически создаются как методы связанных с ними сущностей на клиенте, в дополнение к тому, чтобы быть созданными как методы в контексте домена, который создается для службы домена.

Использование службы домена в приложении Silverlight

Прежде чем рассмотреть фактическое использование службы домена в проекте Silverlight, обратим внимание на то, какие службы RIA Services создаются автоматически. Как уже упоминалось в этой главе, службы RIA Services автоматически создают

в проекте Silverlight код для общения с сервером. Этот код создается в папке `Generated_Code`, которая не добавляется к проекту, но может быть отображена, если вы выберете в проводнике решений проект Silverlight и щелкнете на кнопке Show All Files. Код генерируется службами RIA Services в файлах папки `Generated_Code` (как показано на рис. 35.6), причем исходный файл, содержащий сгенерированный код, называется `<имя веб-проекта>.g.vb` (или `.cs`). Например, для примера проекта исходным файлом будет `Chapter35Sample.Web.g.vb` (или `.cs`).

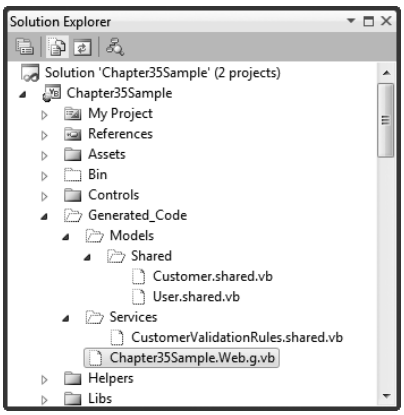


Рис. 35.6

Откройте этот файл, чтобы просмотреть его содержимое и увидеть то, что создал генератор приложений. Особенно интересно то, что сущности (или классы модели представления), представляемые службами домена в веб-проекте, создают соответствующий класс в этом файле (отмеченный атрибутами из классов метаданных или самих классов). Вы также найдете, что для каждой службы домена на клиенте будет создан соответствующий класс *контекста домена*, который отрабатывает коммуникации между службой домена и клиентом. Операции, предоставленные службой домена, будут созданы в соответствующем контексте домена, и вы будете вызывать операции из контекста домена, не пытаясь сослаться непосредственно на службу домена.



Соответствующие операции вставки, модификации и удаления на службе домена не создаются в контексте домена, поскольку эти операции обслуживаются набором изменений. Изменения, внесенные в коллекции сущности, полученной от сервера через операцию запроса, обрабатываются инфраструктурой служб RIA Services в наборе изменений, а когда метод `SubmitChanges` будет вызван в контексте домена, инфраструктура обработает вызов операции вставки, модификации или удаления на службе домена как нужно.

Если будете следовать стандартным соглашениям об именовании служб RIA Services, то служба домена `CustomersService` в веб-проекте приведет к соответствующему контексту домена в проекте Silverlight по имени `CustomersContext`.



Рис. 35.7

Теперь попытаемся заполнить таблицу данных списком клиентов, полученным от сервера. Для этого у вас есть два основных средства: либо использование декларативного подхода на основании XAML, либо подход на основании кода. Подход на основании XAML — самый простой для начала, поэтому используем именно его в данном разделе.

Самый простой способ применения подхода на основании XAML заключается в использовании окна Data Sources, подробно описанного в главе 18, и перетаскивании сущностей, предоставленных контекстом домена, из этого окна на вашу страницу. Вы обнаружите, что источник данных был уже создан в вашем проекте для каждого контекста данных, созданного генератором приложений служб RIA Services (как показано на рис. 35.7), таким образом, вы не должны волноваться о самостоятельном создании источников данных.

Для этого примера будем использовать файл CustomersService, который предоставляет сущность Customer из модели Entity Framework на сервере, поэтому перетащим на страницу сущность Customer (из источника данных CustomersContext, который выделен на рис. 35.7). Это создаст таблицу данных со столбцом для каждого свойства сущности. Теперь, если вы просмотрите код XAML, то увидите, как все связано вместе.

```
<riaControls:DomainDataSource AutoLoad="True" QueryName="GetCustomersQuery"
            Name="CustomerDomainDataSource" Height="0" Width="0">
    <riaControls:DomainDataSource.DomainContext>
        <my:CustomersContext />
    </riaControls:DomainDataSource.DomainContext>
</riaControls:DomainDataSource>

<data:DataGrid AutoGenerateColumns="False" Height="250"
        ItemsSource="{Binding ElementName=CustomerDomainDataSource, Path=Data}"
        Name="CustomerDataGrid" RowDetailsVisibilityMode="VisibleWhenSelected">
    <data:DataGrid.Columns>
            <!-- Этот код был удален для краткости-- >
    </data:DataGrid.Columns>
</data:DataGrid>
```

Используемый элемент управления DomainDataSource является частью инфраструктуры служб RIA Services и предоставляет мост для декларативного обращения к контексту домена в коде XAML. Элемент управления DomainDataSource определяет, что он должен использовать контекст CustomersContext (который соответствует контексту CustomersService на сервере) как свой контекст домена и что операцией запроса, к которой нужно обратиться в контексте домена, является операция GetCustomersQuery. Свойство AutoLoad элемента управления DomainDataSource устанавливается в состояние True, означая, что этот запрос будет вызван при загрузке страницы. И наконец, свойство ItemsSource устанавливается на таблице данных, где оно использует имя элемента привязки, чтобы связаться с элементом управления DomainDataSource и использовать его как источник данных.

Теперь можете запустить проект и убедиться, что таблица данных автоматически заполняется результатами запроса с сервера (как показано на рис. 35.8).

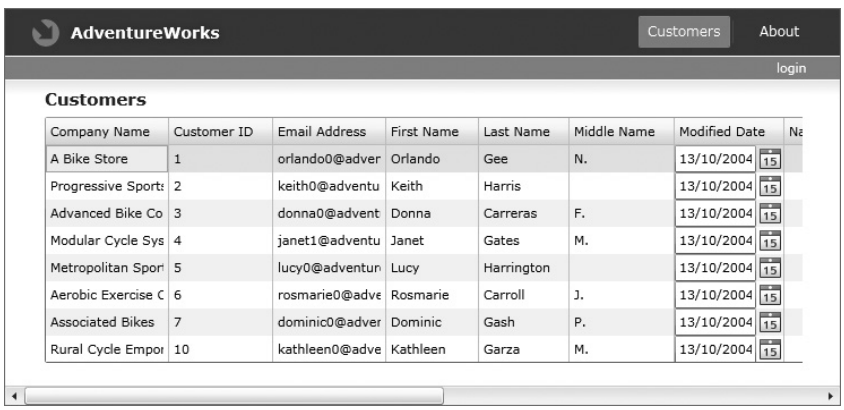


Рис. 35.8

Как уже упоминалось, преимущество возвращения коллекции IQueryable из операции службы домена состоит в том, что службы RIA Services позволяют применять фильтрацию, сортировку, группировку и возможности разбиения на страницы, выполняемые на сервере. Это также очень просто сделать в коде XAML, определив де-

скрипторы в элементе управления DomainDataSource. Рассмотрим каждый из способов по очереди.

Добавьте на страницу элемент управления textbox, который автоматически фильтрует клиентов по названию компании, и назовите его searchTextBox. Добавьте дескриптор фильтра к элементу управления DomainDataSource, который определяет имя свойства для фильтрации (PropertyPath), и оператор, определяющий соответствие (Operator). Затем добавьте класс ControlParameter к дескриптору фильтра, который связан с текстовым полем (указав его имя). Текст в элементе управления textbox используется как критерий поиска (при указании названия свойства текстового поля для получения от него значения), и фильтр запускается каждый раз, когда этот текст изменяется (при указании имени события элемента управления textbox, которое вызовет фильтрацию).

```
<riaControls:DomainDataSource AutoLoad="True" QueryName="GetCustomersQuery"
                    Name="CustomerDomainDataSource" Height="0" Width="0">
    <riaControls:DomainDataSource.DomainContext>
        <my:CustomersContext />
    </riaControls:DomainDataSource.DomainContext>

    <riaControls:DomainDataSource.FilterDescriptors>
    <riaControls:FilterDescriptor PropertyPath="CompanyName"
                Operator="Contains"
                Value="{Binding ElementName=searchTextBox, Path=Text}" />
    </riaControls:DomainDataSource.FilterDescriptors>
</riaControls:DomainDataSource>
```

Сортировка выполняется для таблицы данных автоматически (щелкните на заголовке столбца для сортировки по нему), а когда результаты будут просмотрены, они автоматически возвратятся на сервер, чтобы получить новую страницу результатов согласно текущей странице и критериям сортировки. Но вы можете определить начальную сортировку, используя дескрипторы сортировки элемента управления DomainDataSource, предоставив имена сортируемых свойств и направление сортировки.

```
<riaControls:DomainDataSource AutoLoad="True" QueryName="GetCustomersQuery"
                Name="CustomerDomainDataSource" Height="0" Width="0">
    <riaControls:DomainDataSource.DomainContext>
        <my:CustomersContext />
    </riaControls:DomainDataSource.DomainContext>

    <riaControls:DomainDataSource.SortDescriptors>
        <riaControls:SortDescriptor PropertyPath="CompanyName"
Direction="Ascending" />
    </riaControls:DomainDataSource.SortDescriptors>
</riaControls:DomainDataSource>
```

Группировка выполняется таким же образом, как и при предоставлении дескрипторов группировки и имени свойства для группировки.

```
<riaControls:DomainDataSource AutoLoad="True" QueryName="GetCustomersQuery"
            Name="CustomerDomainDataSource" Height="0" Width="0">
    <riaControls:DomainDataSource.DomainContext>
        <my:CustomersContext />
    </riaControls:DomainDataSource.DomainContext>

    <riaControls:DomainDataSource.GroupDescriptors>
        <riaControls:GroupDescriptor PropertyPath="SalesPerson" />
    </riaControls:DomainDataSource.GroupDescriptors>
</riaControls:DomainDataSource>
```

Разделение данных таблицы на страницы (чтобы отображать, например, по 20 клиентов за раз) осуществляется элементом управления `DataPager`. Добавьте этот элемент управления на свою страницу, свяжите его свойство `Source` с элементом управления `DomainDataSource` и укажите в его свойстве `PageSize` количество элементов, которые будут отображаться в таблице данных.

```
<data:DataPager PageSize="20"
          Source="{Binding Data, ElementName=CustomerDomainDataSource}"/>
```

Размер страницы определяет количество элементов, которое должно отображаться в таблице, а не количество элементов, которое должно быть получено с сервера. Если вы установите только свойство `PageSize`, то с сервера будет все еще возвращаться вся коллекция и просматриваться постранично на клиенте. Для того чтобы возвратить только одну страницу элементов за раз и возвращаться на сервер для получения следующих элементов при переходе между страницами, установите свойство `LoadSize` элемента управления `DomainDataSource`. Как правило, для обоих свойств устанавливают одинаковое значение. Теперь возвращается и отображается одна страница элементов, и на сервере будет запрашиваться и отображаться новая страница элементов каждый раз, когда вы переходите к новой странице с помощью элемента управления `DataPager`.

Любые изменения, которые вы вносите в данные таблицы данных (такие как добавление и удаление рядов, модификация значений), будут отслежены в наборе изменений инфраструктурой служб RIA Services в фоновом режиме. Передача этих изменений назад на сервер осуществляется вызовом метода `SubmitChanges()` в контексте домена. Добавьте на страницу кнопку `SubmitButton`. Добавьте в обработчик ее события `Click` строку кода, приведенную ниже.

VB

```
CustomerDomainDataSource.SubmitChanges();
```

Теперь щелчок на кнопке передаст все внесенные вами изменения назад на сервер.



Вы можете также отменить любые внесенные изменения, используя метод `RejectChanges()` элемента управления `DomainDataSource`.

Законченная страница, которая реализует загрузку, фильтрацию, сортировку, группировку, разбиение на страницы и сохранение данных, представлена на рис. 35.9.

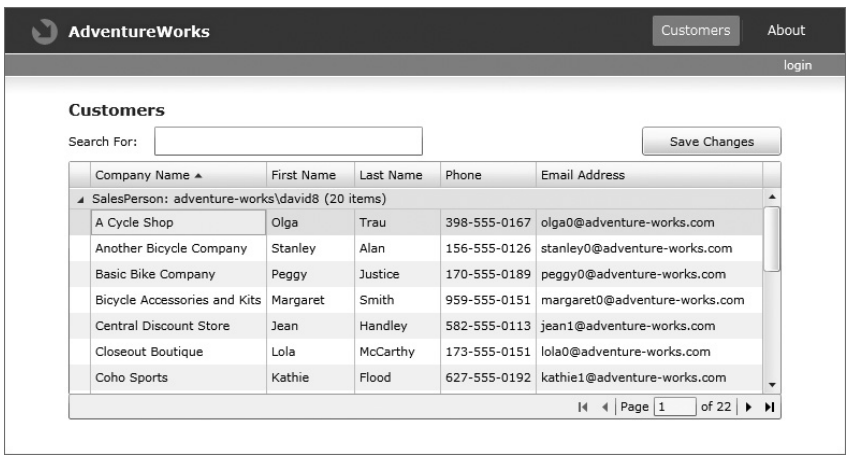


Рис. 35.9

Как можно заметить, службы RIA Services — это чрезвычайно мощная инфраструктура для управления данными, весьма упрощающая реализацию функциональных возможностей, которые когда-то были сложными, а также ускоряющая их создание в бизнес-приложениях.



Элемент управления `DomainDataSource` существенно упрощает использование данных служб RIA Services в приложении Silverlight. Но если вы захотите взаимодействовать со службой домена в коде, то это возможно при создании экземпляра соответствующего контекста домена и при использовании методов в нем. Но обратите внимание на то, что связь со службой домена осуществляется асинхронно, обязывая ваш код иметь соответствующую структуру.

Резюме

В этой главе вы узнали, как службы RIA Services платформы WCF могут значительно упростить архитектуру и разработку сквозного, управляемого данными приложения Silverlight, за счет комбинации шаблонов проектов, создания объектного кода и инфраструктуры с улучшенным набором средств. Службы RIA Services предоставляют еще больше возможностей, чем описано здесь, включая маркировку классов и их свойств атрибутами (такими как правила проверки, применяемыми службами RIA Services), использование классов метаданных (т.е. классов, связанных с сущностью, передаваемой между сервером и клиентом, к которым могут быть применены атрибуты, спроектированные таким образом, чтобы ассоциированные сущности, такие как исходные, не должны были модифицироваться), совместное использование кода сервером и клиентом, функциональные возможности встроенной аутентификации и безопасности, а также многое другое. Но эта глава должна помочь вам начать использовать службы RIA Services, чтобы обеспечить средства коммуникации между вашим приложением Silverlight и сервером.

ЧАСТЬ VIII

Конфигурация и ресурсы

- **ГЛАВА 36**. Файлы конфигурации
- **ГЛАВА 37**. Строки подключения
- **ГЛАВА 38**. Файлы ресурсов

36

Файлы конфигурации

В ЭТОЙ ГЛАВЕ...

- Конфигурационная система .NET
- Использование файлов конфигурации в вашем приложении
- Хранение пользовательских типов в файлах конфигурации

Одной из самых интересных задач при создании приложений является настройка функционирования приложения "на лету", без его перекомпоновки. История приложений, использующих файлы конфигурации для управления способом их работы, весьма длинна. Приложения .NET используют ряд XML-файлов конфигурации, в которые можно вносить изменения для определения поведения приложения. В данной главе рассматривается структура этих конфигурационных файлов и демонстрируется хранение в них пользовательской информации.

Файлы `.Config`

Система конфигурации .NET Framework состоит из нескольких файлов конфигурации (они будут рассматриваться в последующих разделах), которые могут использоваться для настройки одного или нескольких приложений в системе. Часть этой системы представляет собой модель наследования, которая обеспечивает применение конфигураций на правильном уровне. Эта модель такова, что разделы, определенные в конфигурационном файле на нижнем уровне, перекрываются такими же разделами на уровнях, находящихся выше по цепочке. Если значение (или раздел) не определено ни в одном конфигурационном файле, берутся значения по умолчанию из файлов схемы, которой должны отвечать эти конфигурационные файлы.

Файл machine.config

В корне модели наследования располагается файл machine.config (находящийся в папке *systemroot*\Microsoft.NET\Framework*versionNumber*\CONFIG\ или, в случае 64-битных систем, в *systemroot*\Microsoft.NET\Framework64*versionNumber*\CONFIG\), который определяет настройки конфигурации для всей системы. Все конфигурационные файлы наследуются из данного файла и могут переопределять его настройки.

Файл web.config

Веб-приложения настраиваются с помощью файла web.config. Этот файл может находиться в нескольких местах, в зависимости от области применения его настроек. Чтобы настройки применялись ко всем веб-приложениям на компьютере, поместите конфигурационный файл web.config в ту же папку, где находится файл machine.config. В большинстве случаев применяемые настройки должны быть более тонкими. Файл web.config может быть также размещен в любом виртуальном каталоге или подкаталоге для управления веб-приложениями на данном уровне. Если этот файл размещен в корневой папке веб-сайта, конфигурация будет применена ко всем ASP.NET-приложениям на данном сайте.

Небольшое предостережение: во время работы с виртуальными каталогами, не встроенными в структуру каталогов компьютера, возможна ситуация, когда приложение будет по-разному сконфигурировано, в зависимости от способа обращения к нему. Рассмотрим, например, страницу C:\inetpub\wwwroot\MainApplication\Contacts\Contact.aspx, настроенную для работы с виртуальными каталогами MainApplication и Contacts. Вы можете обратиться к странице так:

```
http://localhost/MainApplication/Contacts/Contact.aspx
```

Или так:

```
http://localhost/Contacts/Contact.aspx
```

В первом случае применяемые настройки конфигурации наследуются из папки MainApplication и могут быть переопределены конфигурационным файлом в папке Contacts. Однако во втором случае применяются только те настройки, которые имеются в конфигурационном файле в папке Contacts.



Внесение изменений в файл web.config приводит к перезапуску приложения ASP.NET. Это достаточно эффективный способ заставить веб-приложение сбросить кэш и вести себя так, как будто к нему обратились в первый раз, не перезапуская при этом весь сервер.

Файл app.config

Приложения Windows могут быть настроены и с помощью конфигурационного файла приложения, который также наследует machine.config. Поскольку имя выходного файла становится известным только при компиляции приложения, этот файл начинает свое существование как app.config, а затем переименовывается в *application*.exe.config в качестве части процесса сборки приложения. Например, приложение с файлом AccountingApplication.exe в качестве главного выполняемого файла будет иметь конфигурационный файл AccountingApplication.

exe.config. Этот конфигурационный файл автоматически загружается по его имени одновременно с приложением. Если app.config добавляется к динамически компонуемой библиотеке, в процессе сборки он будет переименован в *assembly*.dll.config.

Файл security.config

Наряду с конфигурационными файлами приложений имеется ряд конфигурационных файлов системы безопасности. Они также используют наследование, но в нескольких измерениях. Конфигурационные файлы системы безопасности сосредоточиваются не на приложениях; они разбиваются на конфигурационные файлы уровня предприятия (Enterprisesec.config), компьютера (Security.config) и пользователя (Security.config). Файлы уровня предприятия и компьютера хранятся там же, где и файл machine.config, в то время как файл уровня пользователя хранится в папке данных приложений конкретного пользователя.

Файл applicationHost.config

Службы IIS7 изменяют способ хранения конфигурационной информации и переходят к использованию набора конфигурационных файлов, работающих параллельно с конфигурационными файлами ASP.NET и .NET Framework. Поскольку версии IIS и .NET Framework независимы, конфигурационная информация, специфичная для конкретной технологии, хранится в файлах machine.config/web.config и applicationHost.config соответственно. Однако в связи с наличием взаимосвязи между IIS и ASP.NET, файл applicationHost.config включается в иерархию наследования конфигурационных файлов. Так как файл applicationHost.config специфичен для конкретного экземпляра IIS, он располагается в иерархии после файлов machine.config и web.config, находящихся на уровне компьютера (т.е. располагается в папке *systemroot*\Microsoft.NET\Framework*versionNumber*\CONFIG\).

Файл applicationHost.config может быть найден в папке *systemroot*\System32\InetSrv\Config, а соответствующие файлы схемы — в подкаталоге Schema. В этой папке имеются также файлы administration.config и redirection.config, ответственные за делегирование возможностей IIS и перенаправление файла конфигурации.

Схема конфигурации

Конфигурационный файл — не важно, machine.config, web.config или некий конфигурационный файл приложения — должен соответствовать одной и той же схеме конфигурации, определяющей, какие элементы должны быть в него включены. Схема располагается в файле C:\Program Files\Microsoft Visual Studio 10.0\Xml\Schemas\DotNetConfig.xsd (C:\Program Files (x86)\Microsoft Visual Studio 10.0\Xml\Schemas\DotNetConfig.xsd на 64-битных машинах) и разбита на несколько разделов.

Раздел configSections

Конфигурационные файлы могут быть настроены таким образом, чтобы содержать любые структурированные XML-данные. Для этого вам нужно определить пользовательский раздел в блоке configSections конфигурационного файла. Он определяет имя раздела конфигурации и класс, который должен быть вызван для обработки раздела.

Раздел configSections в файле machine.config определяет обработчики для каждого стандартного раздела конфигурации, рассматриваемого в данной главе. Вы можете определять собственные разделы конфигурации в вашем конфигурационном файле приложения, указывая при этом, какой класс будет использоваться для проверки и обработки этого раздела. Например, приведенный далее фрагмент определяет обработчик раздела ConfigurationApplication.My.MySettings наряду с соответствующим разделом. Схема этого раздела должна соответствовать тому, что ожидает класс System.Configuration.ClientSettingsSection, а не стандартной схеме конфигурационного файла.

```
<configuration>
<configSections>
<section name="ConfigurationApplication.My.MySettings"
               type="System.Configuration.ClientSettingsSection,
                     System, Version=2.0.0.0, Culture=neutral,
                     PublicKeyToken=b77a5c561934e089"
                     requirePermission="false" />
</configSections>
.
<ConfigurationApplication.My.MySettings>
<setting name="PrimaryServer" serializeAs="String">
<value> www.builttoroam.com </value>
</setting>
</ConfigurationApplication.My.MySettings>
</configuration>
```

Можно также включить configSections в элемент sectionGroup. Предыдущий пример может быть расширен следующим образом:

```
<configuration>
<configSections>
<sectionGroup name="applicationSettings"
              type="System.Configuration.ApplicationSettingsGroup,
                    System, Version=2.0.0.0, Culture=neutral,
                    PublicKeyToken= b77a5c561934e089">
<section name="ConfigurationApplication.My.MySettings"
         type="System.Configuration.ClientSettingsSection,
               System, Version=2.0.0.0, Culture=neutral,
               PublicKeyToken=b77a5c561934e089"
               requirePermission="false" />
<section name="ReferencedAssembly.My.MySettings"
         type="System.Configuration.ClientSettingsSection,
               System, Version=2.0.0.0, Culture=neutral,
               PublicKeyToken=b77a5c561934e089"
               requirePermission="false" />
</sectionGroup>
</configSections>
.
<applicationSettings>
<ConfigurationApplication.My.MySettings>
<setting name="PrimaryServer" serializeAs="String">
<value>www.builttoroam.com</value>
```

```
</setting>
/ConfigurationApplication.My.MySettings>
<ReferencedAssembly.My.MySettings>
<setting name="SecondaryServer" serializeAs="String">
<value>www.peaksite.com</value>
</setting>
</ReferencedAssembly.My.MySettings>
</applicationSettings>
</configuration>
```

При использовании элемент `configSections` должен быть первым потомком элемента `configuration`.

Раздел `startup`

Раздел конфигурации `startup` определяет версию платформы .NET Framework — требуемую (`requiredRuntime`) или поддерживаемую (`supportedRuntime`). По умолчанию приложение .NET будет пытаться работать с применением той версии, с которой оно было собрано. Приложение, собранное с поддержкой нескольких версий, должно указывать это в элементе `supportedRuntime`; при этом первой указывается предпочтительная версия:

```
<configuration>
<startup>
<supportedRuntime version="v4.0.20409"/>
<supportedRuntime version="v2.0.50727"/>
<supportedRuntime version="v1.1.4322"/>
</startup>
</configuration>
```

Этот раздел конфигурации используется приложением, которое тестировалось для версий .NET Framework 4.0, 2.0 и 1.1. При тестировании с версией .NET Framework 1.0 обнаружились аномалии, так что она в списке `supportedRuntime` отсутствует. Номер версии должен точно соответствовать инсталляционному каталогу (например, версия .NET Framework 4.0 обычно инсталлируется в каталог `C:\WINDOWS\Microsoft.NET\Framework\v4.0.20409\`).

Раздел `runtime`

Сборка мусора представляет собой функциональную возможность платформы .NET Framework, отличающую ее от неуправляемых сред. Процесс сбора и уничтожения объектов, на которые нет ссылок, обычно выполняется параллельно с основным приложением, в отдельном потоке. Это означает, что пользователь не должен испытывать никаких затруднений с производительностью приложения из-за сборки мусора. Однако могут возникнуть ситуации, когда такой процесс должен быть запущен основным приложением. Раздел `runtime` конфигурационного файла предоставляет ограниченный контроль над работой платформы .NET во время выполнения приложения. Среди прочего вы можете указать, должна ли сборка мусора выполняться параллельно с основным приложением.

Этот раздел может также использоваться для указания местоположения для поиска сборок, которые могут потребоваться приложению. Этот атрибут может быть полезен, если приложение обращается к сборкам, находящимся в нестандартных местах. Приведенный далее код иллюстрирует применение атрибута `codeBase` для нахожде-

ния ImportantAssembly.dll, а также для указания, что сборка мусора должна выполняться явно основным приложением:

```
<configuration>
<runtime>
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
<dependentAssembly>
<assemblyIdentity name="ImportantAssembly"
                        publicKeyToken="32ab4ba45e0a69a1"
                        culture="neutral" />
<codeBase version="2.0.0.0" href="./ImportantAssembly.dll"/>
</dependentAssembly>
</assemblyBinding>
<gcConcurrent enabled="false"/>
</runtime>
</configuration>
```

Раздел system.runtime.remoting

Раздел remoting конфигурационного файла может использоваться для указания информации об удаленных объектах и каналах, требующейся приложению. Например, канал HTTP по умолчанию может прослушивать порт 8080, что достигается следующим фрагментом:

```
<configuration>
<system.runtime.remoting>
<application>
<channels>
<channel port="8080" ref="http"/>
</channels>
</application>
</system.runtime.remoting>
</configuration>
```

Раздел system.net

В связи с возросшими требованиями более безопасных сред для работы, организации часто используют прокси-серверы для мониторинга и защиты своего сетевого трафика. В результате приложения часто оказываются неработоспособными, пока не будут корректно настроены для использования соответствующего прокси-сервера. Сетевой раздел конфигурационных файлов может применяться для указания прокси-сервера, который приложение должно использовать при осуществлении HTTP-запросов.

Платформа .NET Framework поставляется с классом SmtpClient, который можно применять для отправки электронной почты из приложения. Очевидно, что это действие требует доступа к такой информации, как информация о сервере и соответствующей учетной записи. Хотя эти сведения могут быть жестко закодированы в приложении, более гибким является подход с их указанием в конфигурационном файле, который может быть соответствующим образом изменен при установке приложения. Приведенный далее фрагмент кода иллюстрирует использование прокси-сервера по умолчанию (хотя он может и не применяться для локальных адресов) и настроек SMTP по умолчанию для работы почтового клиента:

```
<configuration>
<system.net>
<defaultProxy>
<proxy usesystemdefaults="true"
```

```
                    proxyaddress="http://192.168.200.222:3030"
                    bypassonlocal="true" />
<bypasslist>
<add address="[a-z]+\.developernews\.com" />
</bypasslist>
</defaultProxy>
<mailSettings>
<smtp deliveryMethod="network">
<network host="smtp.developernews.com"
         port="25" defaultCredentials="true" />
</smtp>
</mailSettings>
</system.net>
</configuration>
```

Раздел `cryptographySettings`

Хотя платформа .NET Framework содержит базовые реализации ряда криптографических алгоритмов, таких как хеширующие функции, иногда требуется их перекрытие. При необходимости в конфигурационный файл можно включить раздел `cryptographySettings` для того, чтобы переназначить существующие имена алгоритмов или отобразить новые имена на другие классы реализаций.

Раздел `system.diagnostics`

Самой трудной частью создания приложения всегда является отладка. Она становится еще сложнее, если приложение уже выпущено, и ошибку не удается повторить в среде отладчика. Один из особенно важных методов отладки ошибок такого типа заключается в применении инструкций трассировки:

```
Trace.WriteLine("Приложение выполнило эту строку"
                " до аварийного останова.")
```

Инструкции трассировки и отладки работают почти так же, как события и обработчики событий. Чтобы указанная инструкция `WriteLine` работала, объект должен перехватывать этот вызов `WriteLine`. Обычно это делает класс `TraceListener`. Платформа .NET Framework поддерживает ряд таких перехватчиков по умолчанию, которые могут быть подключены к приложению с помощью раздела `diagnostics` конфигурационного файла, как показано в следующем фрагменте, где к приложению подключается перехватчик трассировки `EventLog`:

```
<configuration>
<system.diagnostics>
<trace autoflush="true" indentsize="0">
<listeners>
<add name="MyEventListener"
type="System.Diagnostics.EventLogTraceListener,
system, version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
initializeData="DeveloperApplicationEventLog"/>
</listeners>
</trace>
</system.diagnostics>
</configuration>
```

Атрибут `initializeData` определяет текстовую строку, передаваемую конструктору перехватчика трассировки. В случае записи событий этот текст соответствует имени журнала регистрации событий, в который будут заноситься соответствующие записи.

В раздел diagnostics конфигурационного файла могут быть добавлены и другие элементы, например для указания уровня трассировки, определяющего детализацию сообщений, или для управления выводом диалогового окна отладки.

Раздел system.web

Раздел system.web конфигурационного файла используется для управления поведением веб-приложений. Это раздел, который может обладать весьма глубокой иерархией, так как настройки конфигурации могут быть указаны для машины, веб-сервера, веб-сайта, веб-приложения или даже для подпапок. Поскольку этот раздел управляет требованиями безопасности веб-приложения, он зачастую используется для ограничения доступа к определенным областям веб-приложения.

webServices

Хотя приложения веб-служб используют несколько конфигурационных настроек, таких как разделы authentication и impersonation, раздел system.web конфигурационного файла содержит некоторые настройки, относящиеся к работе веб-служб. Например, приведенный далее фрагмент кода позволяет использовать протоколы SOAP и Documentation, но запрещает применение приложением протоколов POST и GET:

```
<configuration>
<system.web>
<webServices>
<protocols>
<add name="HttpSoap"/>
<remove name="HttpPost"/>
<remove name="HttpGet"/>
<add name="Documentation"/>
</protocols>
</webServices>
</system.web>
</configuration>
```

По умолчанию для веб-служб разрешены только протоколы SOAP и Documentation. Достаточно часто для отладочных целей оказывается удобным разрешить протокол POST, чтобы веб-служба могла быть протестирована с помощью веб-браузера. Это следует делать на уровне приложений, включая соответствующий раздел в конфигурационный файл в папке приложения.

Раздел compiler

Раздел compiler конфигурационного файла используется для перечисления компиляторов, установленных на компьютере. В приведенном далее фрагменте кода показана ссылка на компилятор VB.NET в конфигурационном файле machine.config (приложение получает доступ к этой информации с помощью класса CodeDomProvider):

```
<configuration>
<system.codedom>
<compilers>
<compiler language="vb;vbs;visualbasic;vbscript"
extension=".vb"
type="Microsoft.VisualBasic.VBCodeProvider,
System, Version=2.0.0.0,
```

```
Culture=neutral,
PublicKeyToken=b77a5c561934e089" />
</compilers>
</system.codedom>
</configuration>
```

Атрибуты конфигурации

Все элементы конфигурации могут определять configSource, который представляет собой просто перенаправление к отдельному файлу. Это может оказаться полезным, если конфигурационный файл становится слишком большим. Приведенный далее фрагмент кода иллюстрирует, как можно выделить раздел конфигурационного файла и ссылаться на него с помощью указанного атрибута:

```
<!— Исходный конфигурационный файл —>
<configuration>
.
<WindowsApplication1.My.MySettings>
<setting name="Button1_Text" serializeAs="String">
<value> Press Me! </value>
</setting>
</WindowsApplication1.My.MySettings>
</configuration>

<!— Сокращенный конфигурационный файл
    с использованием configSource —>
<configuration>
.
<WindowsApplication1.My.MySettings configSource="MySettings.Config" />
</configuration>

<! — Код файла MySettings.Config —>
<WindowsApplication1.My.MySettings>
<setting name="Button1_Text" serializeAs="String">
<value> Press Me! </value>
</setting>
</WindowsApplication1.My.MySettings>
```

Обратите внимание на ряд ограничений при использовании атрибута configSource.

- Слияния разделов конфигурации в исходном файле и в файле, на который в нем имеется ссылка, не происходит. Если вы включаете раздел в оба файла, при попытке выполнения приложения генерируется ошибка.
- Атрибут не может быть применен к группам разделов конфигурации. Это может оказаться существенным ограничением, поскольку назначение групп разделов заключается в группировании элементов, связывающем сходные разделы конфигурации. Логическое разделение могло бы выносить все элементы из определенной группы разделов в отдельный файл конфигурации.
- Если атрибут применяется в конфигурационном файле web.config, изменение файла, на который имеется ссылка, не приводит к перезапуску приложения ASP.NET. Для того чтобы внесенные изменения вступили в силу, требуется либо вручную перезапустить приложение ASP.NET, либо модифицировать сам файл web.config.

Каждый элемент конфигурационного файла наследует ряд атрибутов, которые могут управлять возможностью перекрытия элемента. Для предотвращения перекрытия элемента, или даже целого раздела, его можно блокировать. Пять атрибутов блокировки приведены в табл. 36.1. Их можно использовать для указания любого количества атрибутов и элементов конфигурации, которые должны быть блокированы.

Таблица 36.1. Атрибуты блокировки

Элемент конфигурации	Описание
`LockItem`	Блокирует элемент, к которому применен данный атрибут, включая все прочие атрибуты, предоставляемые данным элементом и элементами, дочерними по отношению к нему
`LockAttributes`	Блокирует атрибуты из предоставленного списка, разделенного запятыми
`LockAllAttributesExcept`	Блокирует все атрибуты, за исключением атрибутов из предоставленного списка, разделенного запятыми
`LockElements`	Блокирует дочерние элементы из предоставленного списка, разделенного запятыми
`LockAllElementsExcept`	Блокирует все дочерние элементы, за исключением дочерних элементов из предоставленного списка, разделенного запятыми

Возможность блокировки элементов конфигурации особенно нужна при работе с веб-приложениями, которые могут содержать глубокую иерархию наследования конфигурации. Приложения Windows наследуют только конфигурационный файл `machine.config`, так что вряд ли вам потребуется блокировать какие-либо элементы в этом случае.

Настройки приложений

Приложения часто имеют настройки, которые не вписываются в схему конфигурации по умолчанию. В следующих разделах будут рассмотрены четыре механизма хранения такой информации.

Использование раздела `appSettings`

Первый метод заключается в использовании предопределенного раздела `appSettings` конфигурационного файла. Этот раздел может применяться для хранения простых пар "имя/значение", например, как показано в приведенном далее фрагменте, для хранения имени сервера:

```
<configuration>
<appSettings>
<add key="Server" value="http://www.builttoroam.com"/>
</appSettings>
</configuration>
```

Это значение легко получить из кода приложения с помощью свойства `AppSettings` класса `ConfigurationManager` (который требует ссылки на сборку `System.Configuration`):

VB

```
Dim server As String =
    ConfigurationManager.AppSettings("Server")
```

C#

```
var server = ConfigurationManager.AppSettings["Server"];
```

Один из недостатков такого подхода заключается в том, что имя настройки представляет собой строку, а не строго типизированное свойство. Кроме того, значение также является строкой, что зачастую неприемлемо.



В случае веб-приложений вы должны использовать класс `WebConfigurationManager` вместо класса `ConfigurationManager`, поскольку он обеспечивает доступ к дополнительной конфигурационной информации, специфичной для приложений ASP.NET.

Настройки проекта

Используя вкладку Settings свойств проекта, можно определить настройки приложения различных типов. На рис. 36.1 показано, как выглядит в этом диалоговом окне настройка `PrimaryServer`.

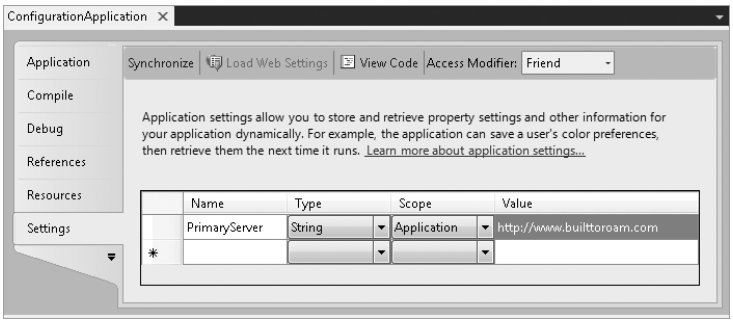


Рис. 36.1

После добавления настроек приложения таким образом раздел `appSettings` не используется, как можно было бы ожидать. Вместо этого в конфигурационном файле определяется новый раздел, как описывалось в разделе, посвященном элементу `configSections`. Вот как выглядит соответствующий фрагмент кода:

```
<configuration>
...
<ConfigurationApplication.My.MySettings>
<setting name="PrimaryServer" serializeAs="String">
<value> www.builttoroam.com </value>
</setting>
</ConfigurationApplication.My.MySettings>
</configuration>
```

Для обращения к этой настройке в коде можно использовать строго типизированное свойство.

VB

```
Dim primaryServer as String = My.Settings.PrimaryServer
```

C#

```
string primaryServer =
    Properties.Settings.Default.PrimaryServer;
```

Динамические свойства

Третий механизм хранения информации, связанной с приложением, — применение динамических свойств. Обычно они используются для динамически устанавливаемых свойств проектировщика. Например, можно установить текст для кнопки Button1 с помощью следующего блока конфигурации:

```
<configuration>
...
<applicationSettings>
<ConfigurationApplication.My.MySettings>
<setting name="Button1_Text" serializeAs="String">
<value> Press Me Now! </value>
</setting>
</ConfigurationApplication.My.MySettings>
</applicationSettings>
</configuration>
```

Вы можете заметить, что в приведенном коде используется тот же синтаксис, что и в настройках приложения, определенных проектировщиком свойств проекта. Фактически это одно и то же. Единственное отличие заключается в том, что в методе `InitializeComponent` формы имеется строка с кодом, устанавливающим текст кнопки:

VB

```
Me.Button1.Size =
    Global.ConfigurationApplication.My
          .MySettings.Default.Button1_Size
```

C#

```
this.button1.Size =
    global::ConfigurationApplication.Properties.Settings.
            Default.Button1_Size;
```

При развертывании приложения текст кнопки Button1 динамически загружается из конфигурационного файла. В приведенной далее инструкции размер управляющего элемента Button1 устанавливается равным значению, динамически загруженному из конфигурационного файла.

1. Выберите Button1 в проектировщике и нажмите <F4> для вывода окна Properties. Найдите элемент `ApplicationSettings` в категории Data или в алфавитном списке, как показано на рис. 36.2.
2. Щелкните на кнопке с многоточием за строкой PropertyBinding. После этого откроется диалоговое окно, в котором будут перечислены доступные свойства для кнопки Button1, а также установленные настройки приложения, как показано на рис. 36.3.

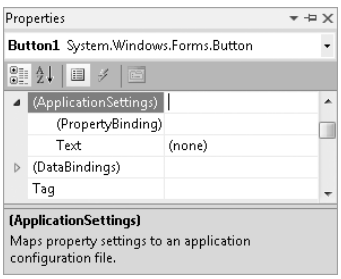


Рис. 36.2

3. Выберите раскрывающийся список рядом со свойством Size и выберите New. После этого откроется диалоговое окно, в котором можно указать значение по умолчанию, имя настройки приложения и область видимости настройки.

4. Укажите имя настройки приложения, например `Button1_Size`, а область видимости установите равной Application. Можете модифицировать значение по умолчанию или просто принять значение из текущих свойств Button1, как показано на рис. 36.4.

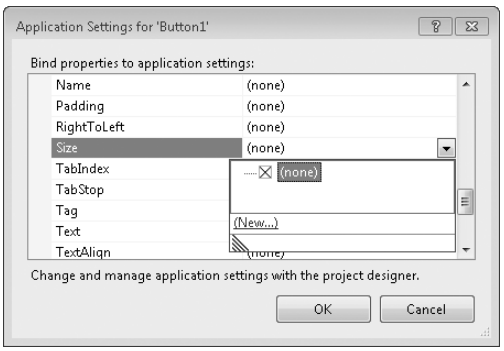


Рис. 36.3

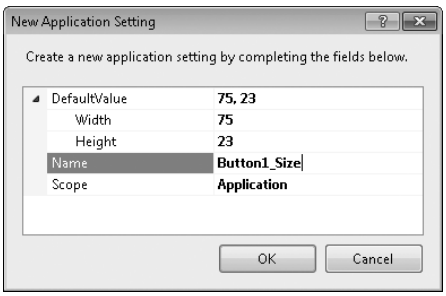


Рис. 36.4

5. Щелкните на кнопках OK в обоих диалоговых окнах. Если вы откроете конфигурационный файл `app.config`, который будет доступен в окне Solution Explorer, то увидите раздел с настройкой `Button1_Size`.

Разделы пользовательской конфигурации

Разработчики часто хотят включать в конфигурационные файлы более структурированную информацию, чем та, которая может быть включена в раздел `appSettings`. Для решения этой задачи и устранения при необходимости дополнительных конфигурационных файлов можно создать пользовательский раздел конфигурации. Этот новый раздел должен быть определен в начале конфигурационного файла с помощью элемента `configSection`, со ссылкой на класс, который должен использоваться для обработки данной части конфигурационного файла.

В прошлом этот процесс был весьма сложен, поскольку класс должен был реализовывать интерфейс `IConfigurationSectionHandler`. У него имеется простой метод, `Create`, который вызывается, когда в коде выполняется первое обращение к разделу. Поддержка обработки раздела со стороны системы крайне невелика, так что класс, реализующий данный интерфейс, часто прибегает к полному анализу XML-блока для получения настроек.

Visual Studio 2010 предоставляет гораздо лучшую поддержку создания пользовательских разделов конфигурации посредством классов `ConfigurationSection` и `ConfigurationElement`. Они представляют основу для создания классов, которые отображаются на структуру данных, хранящихся в конфигурационных файлах. Теперь вместо класса, который обрабатывает раздел конфигурации, вы можете создавать гораздо более простой класс, который отображается на раздел. Когда в коде выполняется обращение к разделу, возвращается экземпляр этого класса с соответствующим множеством элементов данных. Вся обработка XML, которая была необходима ранее, теперь выполняется .NET Framework.

Хотя процесс написания пользовательского раздела конфигурации значительно упростился, иногда возникает необходимость управлять чтением раздела. Для того чтобы получить такой контроль, можно воспользоваться одним из двух вариантов.

- Первый заключается в возврате к применению обработчика раздела конфигурации и обработке XML-файла вручную. Это может быть полезно, если требуется исходное XML-представление. Однако при этом необходима обработка XML-файла.
- Второй способ состоит в создании соответствующего промежуточного класса. Вместо непосредственного обращения к классу может быть сгенерирован другой класс, который предоставляет конфигурационную информацию требуемым способом.

Если вы решили воспользоваться одним из этих вариантов, возможно, имеет смысл вернуться на шаг назад и подумать, действительно ли структура раздела конфигурации имеет формат, подходящий для сохраняемых данных.

В приведенном далее примере вашему приложению требуется список зарегистрированных объектов, с которыми оно будет работать. Один тип объекта — компания, и вы должны предоставить приложению название компании и дату ее регистрации. XML-фрагмент конфигурационного файла может при этом иметь следующий вид:

```
<RegisteredEntities>
  <Companies>
    <add CompanyName="Random Inc"
         RegisteredDate="31/1/2005" />
    <add CompanyName="Developer Experience Inc"
         RegisteredDate="1/8/2004" />
  </Companies>
</RegisteredEntities>
```

Будучи сгенерированными, соответствующие классы могут иметь приведенный далее вид (здесь вновь требуется ссылка на сборку `System.Configuration`):



VB

```
Public Class RegisteredEntities
    Inherits ConfigurationSection

< ConfigurationProperty("Companies") > _
    Public ReadOnly Property Companies() As Companies
        Get
            Return CType(MyBase.Item("Companies"),Companies)
        End Get
    End Property
End Class

< ConfigurationCollectionAttribute(GetType(Company)) > _
Public Class Companies
    Inherits ConfigurationElementCollection

    Protected Overrides Function CreateNewElement()
        As ConfigurationElement
        Return New Company
    End Function

    Protected Overrides Function GetElementKey _
        (ByVal element As ConfigurationElement) As Object
        Return CType(element, Company).CompanyName
```

```
    End Function

    Public Sub Add(ByVal element As Company)
        Me.BaseAdd(element)
    End Sub
End Class

Public Class Company
    Inherits ConfigurationElement

< ConfigurationProperty("CompanyName",
                        DefaultValue:="Random Inc",
                        IsKey:=true, IsRequired:=true) > _
    Public Property CompanyName() As String
        Get
            Return CType(MyBase.Item("CompanyName"),String)
        End Get
        Set
            MyBase.Item("CompanyName") = value
        End Set
    End Property

< ConfigurationProperty("RegisteredDate",
                        DefaultValue:="31/1/2005",
                        IsKey:=false, IsRequired:=false) > _
    Public Property RegisteredDate() As String
        Get
            Return CType(MyBase.Item("RegisteredDate"),String)
        End Get
        Set
            MyBase.Item("RegisteredDate") = value
        End Set
    End Property
End Class
```

Сниппет кода RegisteredEntities.vb



C#

```
class RegisteredEntities : ConfigurationSection{
    [ConfigurationProperty("Companies")]
        public Companies Companies{
            get{
                return base["Companies"] as Companies;
            }
        }
}

[ConfigurationCollection(typeof(Company))]
class Companies : ConfigurationElementCollection{
    protected override ConfigurationElement
        CreateNewElement(){
        return new Company();
    }
    protected override object GetElementKey
        (ConfigurationElement element){
            return (element as Company).CompanyName;
        }
    public void Add(Company element){
        BaseAdd(element);
    }
}
```

```
class Company : ConfigurationElement{
    [ConfigurationProperty("CompanyName",
                           DefaultValue = "Random Inc",
                           IsKey = true, IsRequired = true)]
        public string CompanyName{
            get{
                return base["CompanyName"] as string;
            }
            set{
                base["CompanyName"] = value;
            }
        }
    [ConfigurationProperty("RegisteredDate",
                           DefaultValue = "31/1/2005",
                           IsKey = false, IsRequired = true)]
        public string RegisteredDate{
            get{
                return base["RegisteredDate"] as string;
            }
            set{
                base["RegisteredDate"] = value;
            }
        }
}
```

Сниппет кода RegisteredEntities.cs

Код содержит три класса, необходимых для корректного отображения функциональности этого раздела. Раздел зарегистрированных объектов соответствует классу `RegisteredEntities`, который содержит единственное свойство, возвращающее коллекцию компаний. Наличие коллекции обусловлено необходимостью поддержки нескольких компаний. Данная функциональность может быть расширена сбросом и удалением компаний, что может оказаться полезным, если имеется веб-приложение, требующее возможности управления доступом разных компаний к различным частям приложения. Далее, имеется класс `Company`, который соответствует информации об отдельной компании.

Для обращения к этому разделу из кода можно просто воспользоваться классом `configurationManager`:

VB

```
Dim registered as RegisteredEntities= _
  ctype(configurationmanager.GetSection("RegisteredEntities"),
        RegisteredEntities)
```

C#

```
var registered =
    ConfigurationManager.GetSection("RegisteredEntities")
        as RegisteredEntities;
```



Для того чтобы система конфигурации .NET корректно загружала ваш конфигурационный файл с разделом `RegisteredEntitie`, его следует зарегистрировать в элементе `configSections`. Это можно сделать, добавив строку `<section name= "RegisteredEntities" type= "Configuration-Application.RegisteredEntities, ConfigurationApplication, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />` в элемент `configSections` непосредственно перед дескриптором `</configSections>`.

Автоматизация с использованием SCDL

Вы только что узнали, как можно создавать пользовательские разделы конфигурации и отображать их на классы. Хотя это и является большим шагом вперед по сравнению с написанием обработчиков разделов, это все равно сложный процесс, чреватый появлением ошибок. Кроме того, отладка разделов конфигурации практически невозможна, так как очень сложно отследить, что именно идет не так.

Работая над другим проектом по поддержке программистов ASP.NET, руководитель группы разработчиков ASP.NET в Microsoft обнаружил, что процесс создания таких классов достаточно прост и легко может быть автоматизирован. С этой целью он создал небольшое приложение под названием SCDL (`http://blogs.msdn.com/dmitryr/archive/2005/12/07/501365.aspx`), которое может получить фрагмент конфигурационных данных, таких как рассматривавшийся раздел `RegisteredEntities`, и выдать как отображающий класс, так и файл схемы, которая представляет переданный раздел. Сгенерированный код может быть добавлен в приложение. Кроме того, если фрагмент конфигурационных данных включить в решение как некомпилируемый файл, то можно автоматизировать генерацию отображающих классов с помощью предшествующей сборки команды `batch`. Если в структуру раздела необходимо внести изменения, это можно сделать в фрагменте кода. При этом при очередной сборке классы будут обновлены автоматически.

Технология IntelliSense

Даже несмотря на корректное отображение пользовательских разделов конфигурации, Visual Studio 2010 не обеспечивает поддержку добавления пользовательских разделов в конфигурационный файл. В отличие от остальной части конфигурационного файла, которая поддерживается IntelliSense и для которой выполняется проверка, ваш пользовательский раздел проверке не подлежит.

Чтобы добиться поддержки своего пользовательского раздела со стороны IntelliSense, а также его проверки, необходимо указать Visual Studio 2010 структуру этого раздела. Это можно сделать, разместив соответствующую схему (генерируемую инструментарием SCDL) в папке схем XML, которая обычно располагается в `C:\Program Files\Microsoft Visual Studio 10.0\Xml\Schemas\`. К сожалению, этим дело не ограничивается; требуется также указать, что эта схема должна быть включена в каталог, используемый для синтаксического анализа конфигурационных файлов. Для регистрации своей схемы выполните следующие действия.

1. Сгенерируйте файл схемы из фрагмента конфигурации:
   ```
   Scdl.exe snippet.scdl snippet.vb snippet.xsd
   ```
3. Скопируйте файл схемы (в данном случае — `snippet.xsd`) в папку схем.
4. Создайте текстовый файл `Config.xsd` и включите в него приведенные далее строки. Заметим, что если ваша схема называется иначе, то вы должны внести в приведенные строки соответствующие изменения. Вы можете также добавить дополнительные строки для включения нескольких схем. Не удаляйте строку `DotNetConfig.xsd`, поскольку при этом будет отменена проверка стандартных разделов конфигурации.
   ```
   <?xml version="1.0" encoding="utf-8" ?>
   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:include schemaLocation="DotNetConfig.xsd"/>
   <xs:include schemaLocation="snippet.xsd"/>
   </xs:schema>
   ```

4. Откройте файл `Catalog.xml` в текстовом редакторе и замените строку `"DotNetConfig.xsd"` строкой `"Config.xsd"` Это заставит систему IntelliSense при работе с вашим конфигурационным файлом использовать файл `Config.xsd` вместо файла `DotNetConfig.xsd`. Однако, поскольку в файле указаны как ваша схема, так и схема `DotNetConfig.xsd`, проверка будет выполняться как для вашего раздела конфигурации, так и для стандартных разделов.

Пользовательские настройки

Поскольку конфигурационные файлы обычно используются для хранения настроек, управляющих работой приложения, зачастую приходится иметь возможность изменять их динамически для настройки индивидуальных запусков приложения. Вместо построения полностью отличной схемы для доступа и сохранения настроек можно просто изменить область видимости своих настроек. На рис. 36.5 показано, что на вкладке Settings страницы свойств проекта вы можете указать, должны ли ваши настройки иметь в качестве области видимости приложение (Application) или пользователя (User).

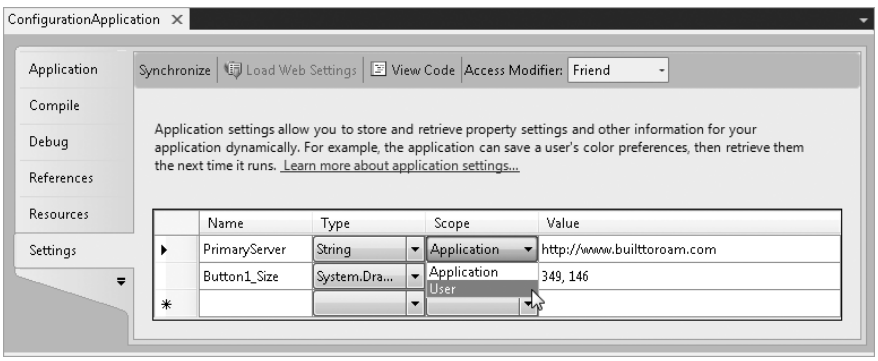


Рис. 36.5

По сути, изменяя область видимости настроек, вы выбираете, должны ли ваши настройки быть доступны только для чтения (другими словами, применимы ли они к приложению независимо от того, кто его запускает) или и для чтения, и для записи. При обращении к настройкам проекта из кода следует учитывать, что если вы попытаетесь присвоить значения настройкам приложения, то получите ошибку компиляции, в то время как пользовательским настройкам вполне можно назначать новые значения. Присваивание нового значения пользовательской настройке изменяет ее значение только на время данного сеанса работы с приложением. Если вы хотите сделать это значение постоянным, сохраняющимся от сеанса к сеансу, вызовите метод `Save` сгенерированного проектировщиком объекта `Settings`, как показано в следующем фрагменте:

VB

```
Properties.Settings.Default.BackgroundColor = Color.Blue;
Properties.Settings.Default.Save();
```

В табл. 36.2 перечислены другие методы объекта `Settings`, которые могут пригодиться при работе с пользовательскими настройками.

Таблица 36.2. Методы объекта `Settings`

Метод	Функциональность
`Save`	Сохраняет текущее значение настройки
`Reload`	Восстанавливает сохраненное значение настройки
`Reset`	Сбрасывает сохраненное и хранящееся в памяти значения настройки в значение по умолчанию (значение, которое вы определили во время разработки на вкладке Settings страницы свойств проекта). Вызывать метод `Save` после вызова `Reset` не требуется
`Upgrade`	При управлении версиями приложения можно вызвать метод `Upgrade` для обновления значений пользовательских настроек значениями, связанными с приложением. Будьте внимательны при вызове этого метода, так как при этом можно нечаянно сбросить пользовательские настройки
`(event) SettingChanging`	Событие, генерируемое перед изменением настройки
`(event) PropertyChanged`	Событие, генерируемое после изменения настройки
`(event) SettingsLoaded`	Событие, генерируемое при загрузке сохраненного значения настройки
`(event) SettingsSaving`	Событие, генерируемое перед сохранением значения настройки

При построении приложения, работающего с пользовательскими настройками, важно протестировать приложение в режиме первого запуска. При первом запуске у вашего приложения нет настроек с пользовательской областью видимости, а это означает, что приложение будет использовать либо значения из конфигурационного файла приложения, либо значения по умолчанию, закодированные в сгенерированном проектировщиком файле. При тестировании приложения кнопка Synchronize на вкладке Settings страницы Project Properties (показана в верхнем левом углу на рис. 36.5) удаляет все пользовательские настройки, которые могли быть сохранены в процессе более ранних выполнений вашего приложения.

Проекты с настройками

По мере развития приложения появляется необходимость разбивать логику на сборки, к которым обращается основное приложение. В прошлом, когда такие сборки пытались использовать настройки приложения, возникал ряд сложных проблем. Теперь, с появлением Visual Studio 2010, стало возможным совместное использование настроек приложения различными сборками с применением проектировщика свойств проекта. На рис. 36.6 показана вкладка Settings проектировщика свойств проекта для такой сборки, к которой будет выполняться обращение. В этом случае в раскрывающемся списке Access Modifier выбирается пункт Public, позволяющий обращение к настройкам со стороны главного приложения.

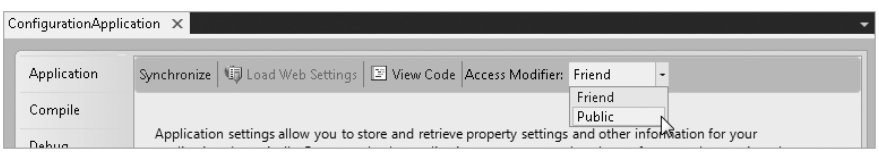


Рис. 36.6

Для того чтобы обратиться к настройкам из главного приложения, можно опять использовать сгенерированное строго типизированное свойство:

VB

```
ReferencedAssembly.My.MySettings.Default.SecondaryServer
```

C#

```
ReferencedAssembly.Properties.Settings.Default.SecondaryServer
```

Следует сказать несколько слов об использовании проектировщика свойств проекта и обращении к настройкам приложения. Если вы рассмотрите исходный текст для проектировщика настроек, то заметите, что для каждой из настроек, как уже упоминалось, для доступа определено строго типизированное свойство. Важно отметить наличие `DefaultSettingValueAttribute`, поскольку оно определяет значение, которое вернет свойство, если в конфигурационном файле не будет указано никакого значения. В приведенном далее фрагменте в случае, если в конфигурационном файле не будет определен элемент `SecondaryServer`, будет возвращено значение по умолчанию `www.peaksite.com`.

VB

```
Namespace My
   Partial Friend NotInheritable Class MySettings
    Inherits Global.System.Configuration.ApplicationSettingsBase
.
<Global.System.Configuration.ApplicationScopedSettingAttribute(), _
Global.System.Diagnostics.DebuggerNonUserCodeAttribute(), _
Global.System.Configuration.DefaultSettingValueAttribute("www.peaksite.com") > _
        Public ReadOnly Property SecondaryServer() As String
        Get
            Return CType(Me("SecondaryServer "),String)
        End Get
    End Property
  End Class
End Namespace
```

Вы можете спросить “Почему так важна работа с настройками приложения, к которому выполняется обращение?” Это связано с тем, что хотя проектировщик свойств проекта и позволяет указать, что вы хотите разрешить обращение к настройкам из другой сборки, он не позволяет указать, что приложение выполняет такое обращение из другой сборки. В результате при компиляции приложения используется только файл `app.config` из папки проекта приложения, а не объединение элементов файлов `app.config` в папке сборки, к которой выполняется обращение.

К сожалению, из-за значений атрибутов по умолчанию вы вряд ли заметите это раньше, чем приложение будет инсталлировано, после чего вы обнаружите, что некоторые из настроек в файле `app.config` отсутствуют. Поэтому вы должны вручную объединить эти файлы. Результат при этом будет следующим:

```
<configuration>
.
<applicationSettings>
<ConfigurationApplication.My.MySettings>
<setting name="PrimaryServer" serializeAs="String">
<value> www.softteq.com </value>
</setting>
</ConfigurationApplication.My.MySettings>
<ReferencedAssembly.My.MySettings>
<setting name="SecondaryServer" serializeAs="String">
<value>www.peaksite.com</value>
</setting>
</ReferencedAssembly.My.MySettings>
</applicationSettings>
</configuration>
```

Резюме

В этой главе вы изучили применение конфигурационных файлов не только для управления работой приложения, но и для хранения настроек, которые, возможно, придется изменить во время его работы. Теперь вы можете хранить в конфигурационном файле как простую информацию наподобие пар “имя/значение”, так и более структурированные данные.

37

Строки подключения

В ЭТОЙ ГЛАВЕ...

- Создание строк подключения для применения в своем приложении
- Работа с диалоговыми окнами Visual Studio 2010 Connection для настройки подключения к источнику данных
- Доступ к строкам подключения из кода

Для многих приложений требуется возможность работы с постоянными, хранящимися между сеансами работы с приложением, данными. И очевидным средством в случае программного обеспечения уровня предприятия являются реляционные базы данных. Платформа .NET Framework предоставляет поддержку работы с базами данных SQL Server, SQL Server Compact Edition, Oracle, ODBC и OLE DB. Многие другие базы данных поддерживаются с помощью сторонних провайдеров. Для подключения к любой из этих баз данных необходимо указать строку подключения, которая определяет местоположение, базу данных, информацию аутентификации и другие параметры подключения. В этой главе объясняется, как создать и сохранить строки подключения. Кроме того, вы узнаете о шифровании и работе со строками подключения в вашем коде.

Мастер строк подключения

Строки подключения похожи на XML в том, что, хотя их и можно прочесть, работать с ними непосредственно не только не доставляет ни малейшего удовольствия, но и просто не рекомендуется. Поскольку строки подключения являются настоящими строками, в них легко внести ошибки, неверно набранные слова или, например, пропустить необходимый параметр. В отличие от XML, который легко проверить на соответствие схеме, проверить строку подключения куда сложнее. Мастер строк подключения, встроенный в Visual Studio 2010, позволяет определить подключение к базе данных без редактирования самой строки вручную.

Вызвать мастер строк подключений можно по-разному, в чем вы убедитесь, применив любой управляющий элемент для работы с данными в проектировщике Windows Form или Web Form. Для иллюстрации применения мастера следуйте приведенным далее инструкциям, которые добавляют новый источник данных в существующее приложение Windows Forms, после подключения к учебной базе данных AdventureWorksLT, которую следует загрузить с веб-сайта Codeplex (зайдите на сайт по адресу `www.codeplex.com` и выполните поиск AdventureWorksLT).

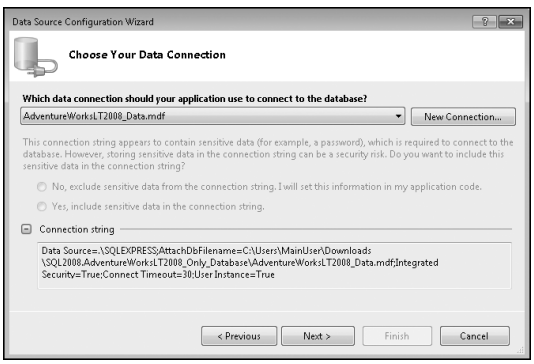


Рис. 37.1



Рис. 37.2

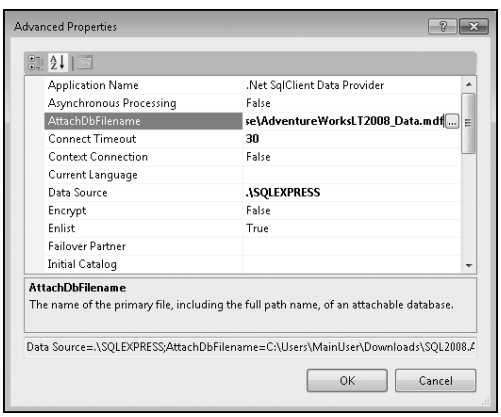


Рис. 37.3

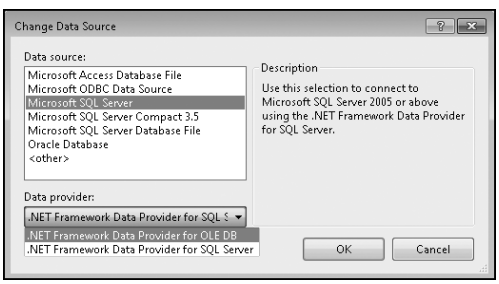


Рис. 37.4

1. Из меню Data в Visual Studio 2010 выберите команду Add New Data Source, что приведет к открытию Data Source Configuration Wizard.
2. Выбрав Database, следуйте одной из моделей – DataSet или Entity Data Model, после чего вам будет предложено указать используемое подключение к базе данных. Если подключение уже имеется, его можно выбрать из раскрывающе-

гося списка, и в нижней части экрана появится соответствующая строка подключения, как показано на рис. 37.1.

Эта строка подключения соединяется с базой данных AdventureWorksLT с использованием возможности SQL Server Express по подключению к файлу базы данных. Позже в этой главе свойства строки подключения SQL Server будут рассмотрены более подробно.

3. Для создания соединения щелкните на кнопке New Connection, что приведет к открытию диалогового окна Add Connection, в котором указываются свойства строки подключения. На рис. 37.2 показано диалоговое окно для соединения с SQL Server Database File. Вид данного диалогового окна зависит от настраиваемого источника данных.

   Обратите внимание, что на рис. 37.2 представлены только основные свойства подключения (такие, как имя файла базы данных и информация аутентификации).

4. Щелкните на кнопке Advanced, чтобы открыть показанное на рис. 37.3 окно Advanced Properties, в котором можно выполнить настройку всех свойств подключения SQL Server. В нижней части окна показана конструируемая строка подключения. Значения по умолчанию в строке подключения опущены. После того как значение установлено, оно появляется в строке подключения и в окне Properties. Список доступных свойств базируется на используемом источнике данных.

5. Щелкните на кнопке OK, чтобы вернуться в окно Add Connection, в котором можно изменить тип источника данных, щелкнув на кнопке Change. При этом откроется диалоговое окно Change Data Source, показанное на рис. 37.4.

   Список в левой части окна содержит все источники данных, зарегистрированные к настоящему моменту в файле `machine.config`. Для указанного источника данных, такого как Microsoft SQL Server, может быть несколько провайдеров данных — в нашем случае это провайдеры SQL Server и OLE DB.



Выбор другой комбинации "источник/провайдер данных" приводит к появлению другого диалогового окна Add Connection, в котором выводятся параметры, относящиеся к выбранному подключению к базе данных. В большинстве случаев для настройки подключения необходимо открыть окно Advanced Properties.

6. После указания источника данных и настроек подключения с помощью диалогового окна Add Connection вернитесь к мастеру Data Source Configuration Wizard. Если вы создаете подключение, можете сохранить строку подключения в конфигурационном файле приложения, как показано на рис. 37.5. Если вы можете гарантировать, что на более поздних этапах не произойдет изменение местоположения базы данных, режима аутентификации или других свойств подключения, то имеет смысл сохранить получившуюся строку подключения в конфигурационном файле. Такое сохранение имеет дополнительное преимущество, заключающееся в том, что одна и та же строка может повторно использоваться в ходе всего процесса выполнения приложения.

Если не сохранять строку подключения в конфигурационном файле, она должна быть явно присвоена создаваемому объекту подключения, что усложняет ее повторное использование. И напротив, сохранение строки подключения в конфигурационном файле означает, что другой объект подключения может обратиться к той же самой строке. Если в последующем подключение к базе данных изменится, вы сможете легко обновить его в одном-единственном месте.

7. Мастер Data Source Configuration Wizard продолжает проводить вас через процедуру выбора объектов баз данных, которые вы хотите добавить к своему источнику данных. Более детально данный процесс описан в главе 27.

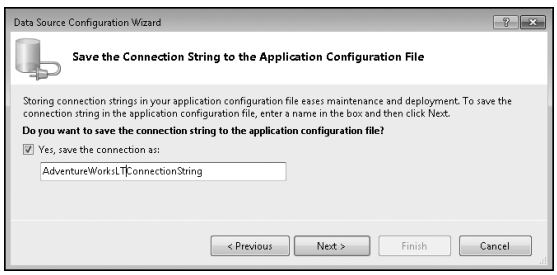


Рис. 37.5

Когда вы сохраняете строку приложения в конфигурационном файле приложения, она добавляется в раздел `connectionStrings` конфигурации, как показано в следующем фрагменте файла `app.config` (тот же раздел может иметься в файле `web.config` для веб-приложений):

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<appSettings />
<connectionStrings>
<add
      name="Connection_Strings.Properties.Settings.
      AdventureWorksLTConnectionString"
      connectionString="Data  Source=.\SQLEXPRESS;AttachDbFilename=C:\Users\
MainUser\Downloads\SQL2008.AdventureWorksLT2008_Only_Database\AdventureWorks
LT2008_Data.mdf;Integrated Security=True;Connect Timeout=30;User Instance=True"
providerName="System.Data.SqlClient" />
</connectionStrings>
</configuration>
```

Раздел `connectionStrings` конфигурационного файла использует стандартный элемент коллекции, позволяющий определить несколько строк подключения и обращаться к ним из кода приложения. Например, показанная выше строка подключения может быть использована кодом следующим образом (предполагается, что ваш проект имеет ссылку на сборку `System.Configuration`):

C#

```
private void OpenConnectionClick(object sender, EventArgs e)
{
    var sqlCon = new System.Data.SqlClient.SqlConnection();
    sqlCon.ConnectionString = ConfigurationManager.
    ConnectionStrings["AdventureWorksLTConnectionString"].ConnectionString;
    sqlCon.Open();
}
```

VB

```
Private Sub OpenConnectionClick(ByVal sender As System.Object,
                                ByVal e As System.EventArgs) _
                                Handles BtnOpenConnection.Click
    Dim sqlCon As New SqlClient.SqlConnection
    sqlCon.ConnectionString = ConfigurationManager.ConnectionStrings _
                ("AdventureWorksLTConnectionString").ConnectionString
    sqlCon.Open()
End Sub
```

Приятным результатом работы с мастером строк подключения является то, что он добавляет строго типизированную поддержку обращения к строке из вашего кода. Это означает, что вы можете обратиться к строке подключения, используя следующие строго типизированные методы, а не вызывая их с помощью строковой константы:

C#

```
Properties.Settings.Default.AdventureWorksLTConnectionString;
```

VB

```
My.Settings.AdventureWorksLTConnectionString
```

Еще одно преимущество сохранения строк подключения в конфигурационном файле состоит в том, что, когда вы редактируете настройки проекта, список строк подключения появляется в окне вместе с другими настройками проекта, как показано на рис. 37.6. Вы можете не только непосредственно отредактировать строки подключения, но и вызвать мастер строк подключения, щелкнув на кнопке с троеточием справа от значения строки подключения (это позволит обновить свойства подключения без угрозы испортить строку). Обратите внимание, что кнопка с троеточием не видна до тех пор, пока вы не щелкнете на ячейке со значением строки подключения.

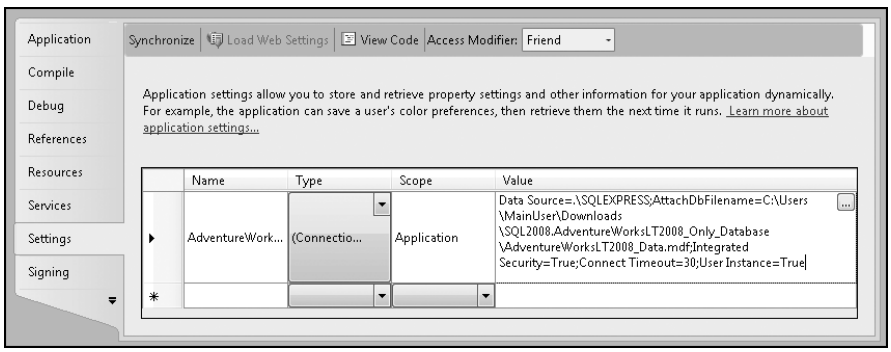


Рис. 37.6

На рис. 37.6 можно увидеть, что у имени строки подключения отсутствует длинный префикс `Connection_Strings.Properties.Settings`, имеющийся в конфигурационном файле приложения. Этот префикс используется для определения того, какие строки подключений должны быть включены и в проектировщик свойств проекта, и для предоставления строго типизированной поддержки.



Поскольку при редактировании строк подключения вручную имеется опасность неверно указать свойства источника данных, крайне рекомендуется использовать либо мастер Add Data Source Wizard, либо проектировщик свойств проекта и по возможности избегать ручного вмешательства в содержимое этих строк.

Формат строки подключения к SQL Server

Вероятно, наиболее популярным провайдером данных является SQL Server, поэтому в табл. 37.1 приведены некоторые часто используемые свойства подключения, которые вам может потребоваться указать для подключения к своему серверу баз данных.

Таблица 37.1. Некоторые распространенные свойства подключения

Свойство подключения	Описание
`Asynchronous Processing`	Определяет, должно ли подключение поддерживать асинхронные вызовы базы данных. Большинство приложений пытается уменьшить время отклика на действия пользователя, так что очень важно, чтобы при извлечении данных приложение не "замирало", перестав реагировать на действия пользователя. В прошлом это могло быть достигнуто только переносом обработки данных в отдельный поток приложения. Методы обращения к данным, такие как `ExecuteNonQuery`, в настоящее время поддерживают вызовы с использованием асинхронного шаблона `Begin` и `End`. Например, возврат из вызова `BeginExecuteNonQuery` происходит немедленно, так что пользовательский интерфейс не блокируется на время обращения к данным
`AttachDBFilename`	Введенное в SQL Server 2005, это свойство означает, что вы можете работать с базами данных, которые не постоянно подключены к экземпляру SQL Server. Это свойство представляет собой путь к основному файлу базы данных, который содержит базу данных. Указание `AttachDBFilenameeff` позволяет при необходимости эффективно подключать и отключать базу данных
`Connect Timeout`	Определяет максимальную продолжительность времени блокировки метода `Open` при попытке подключиться к базе данных. Не путайте это свойство со свойством `Timeout` класса `SQLCommand`, которое определяет тайм-аут при выполнении заданной команды
`Data Source`	Имя или IP-адрес экземпляра SQL Server, к которому обращается подключение. В случае наличия нескольких экземпляров на одной машине, или когда SQL Server назначает имя экземпляра, отличное от имени по умолчанию, этот параметр указывается как часть поля `Data Source`, например `192.168.205.223\InstanceName`
`Initial Catalog`	Указывает имя базы данных, к которой выполняется подключение
`Integrated Security`	Если используется параметр `IntegratedSecurity`, для подключения к серверу базы данных будет использован мандат Windows текущего пользователя. Для того чтобы использовать идентификатор пользователя и пароль, этот параметр должен быть установлен равным `false` . Учтите также, что при работе с ASP.NET с применением аутентификации Windows без заимствования прав, если свойство `IntegratedSecurity` равно `true`, для обращения к серверу базы данных будет использован мандат веб-пользователя
`MultipleActiveResultSets`	Позволяет вернуть по данному подключению несколько результирующих наборов. Например, одна команда может содержать две инструкции `SELECT`. Если свойство `MultipleActiveResultSets` установлено, будут возвращены и смогут использоваться для наполнения объекта `DataSet` результаты обоих инструкций `SELECT`. Это свойство совместимо только с SQL Server 2005 и более поздними версиями
`Password`	Применяется вместе с пользовательской учетной записью для доступа к серверу базы данных SQL Server
`User ID`	Определяет учетную запись пользователя, применяемую для подключения к серверу базы данных SQL Server. Свойство `IntegratedSecurity` должно быть установлено равным `false`

Каждое свойство строки подключения должно быть указано так, как показано в таблице, но их порядок в строке не имеет значения. Для отделения свойств одно от другого используются точки с запятой. Вот пример возможной строки подключения:

```
Data Source=.;Initial Catalog=AdventureWorksLT;
Integrated Security=True;MultipleActiveResultSets=True
```

Создание строки в коде

Хотя мастер строк подключения в Visual Studio 2010 и является очень удобным инструментом для их создания, очень часто такие строки необходимо создавать динамически, что легко сделать с помощью класса `SqlConnectionStringBuilder`. Имеются классы построения строк подключения для Oracle, ODBC и OLE DB, и все они — наследники обобщенного класса `DBConnectionStringBuilder`, обладающего свойством `ConnectionString`.

В приведенном далее примере демонстрируются создание объекта для построения строки подключения, основанного на существующей строке, и изменение режима аутентификации с применением идентификатора и пароля пользователя, которые передаются пользователем перед тем, как присвоить новую строку объекту подключения. Кроме того, пример демонстрирует применение свойства `MultipleActiveResultSets` для выборки из нескольких таблиц базы данных с использованием одного объекта:

C#

```
private void LoadDataClick(object sender, EventArgs e){
    // Обновление строки подключения на основе настроек пользователя
    var sqlbuilder =
        new System.Data.SqlClient.SqlConnectionStringBuilder
        (Properties.Settings.Default.
         AdventureWorksLTConnectionString);
    if (!string.IsNullOrEmpty(this.TxtUserId.Text)){
        sqlbuilder.IntegratedSecurity = false;
        sqlbuilder.UserID = this.TxtUserId.Text;
        sqlbuilder.Password = this.TxtPassword.Text;
    }
    sqlbuilder.MultipleActiveResultSets = true;
    // Создание подключения на основе обновленной строки подключения
    var sqlCon = new System.Data.SqlClient.SqlConnection();
    sqlCon.ConnectionString = sqlbuilder.ConnectionString;
    // Настройка объекта команды и создание объекта для
    // загрузки набора данных
    var sqlcmd = new System.Data.SqlClient.SqlCommand(
            "SELECT * FROM Person.Contact;" +
            "SELECT * FROM Person.ContactType", sqlCon);
    var ds = new DataSet();
    var rds =
        new System.Data.SqlClient.SqlDataAdapter(sqlcmd);
    // Открываем соединение, получаем данные и закрываем соединение
    sqlCon.Open();
    rds.Fill(ds);
    sqlCon.Close();
}
```

VB

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

```
    ' Обновление строки подключения на основе настроек пользователя
    Dim sqlbuilder As New SqlClient.SqlConnectionStringBuilder _
            (My.Settings.AdventureWorksLTConnectionString)
    If Not Me.TxtUserId.Text = "" Then
        sqlbuilder.IntegratedSecurity = False
        sqlbuilder.UserID = Me.TxtUserId.Text
        sqlbuilder.Password = Me.TxtPassword.Text
    End If
    sqlbuilder.MultipleActiveResultSets = True

    ' Создание подключения на основе обновленной строки подключения
    Dim sqlCon As New SqlClient.SqlConnection
    sqlCon.ConnectionString = sqlbuilder.ConnectionString

    ' Настройка объекта команды и создание объекта для
    ' загрузки набора данных
    Dim sqlcmd As New SqlClient.SqlCommand _
                ("SELECT * FROM Person.Contact;" & _
                 "SELECT * FROM Person.ContactType", _
         sqlCon)
    Dim ds As New DataSet
    Dim rds As New SqlClient.SqlDataAdapter(sqlcmd)

    ' Открываем соединение, получаем данные и закрываем соединение
    sqlCon.Open()
    rds.Fill(ds)
    sqlCon.Close()
End Sub
```

Важно заметить, что свойству `MultipleActiveResultSets` присваивается значение `true`, а это означает, что в одном объекте `SqlCommand` можно указать несколько инструкций `SELECT`. Затем объект `SqlCommand` используется объектом `SqlDataAdapter` для заполнения объекта `DataSet`, который будет содержать две таблицы данных, каждая из которых заполняется одной из инструкций `SELECT`.

Шифрование строк подключения

Хотя лучше всего по возможности пользоваться системами аутентификации и безопасности Windows, это получается не всегда; иногда требуется указание в строке подключения идентификатора и пароля пользователя. Рекомендуется не "прошивать" эту информацию в приложении, поскольку в этом случае ее очень легко извлечь. Поэтому либо данная информация должна вводиться пользователем всякий раз, когда он применяет эту систему, либо ее следует добавить в конфигурационный файл. В последнем случае вам требуется механизм для шифрования разделов конфигурационного файла. Здесь вы бегло ознакомитесь с тем, как выполнять шифрование раздела конфигурационного файла для веб-приложения StagingWebsite, файл `web.config` которого имеет следующий вид:

```
<?xml version="1.0"?>
<configuration>
<connectionStrings>
<add name="AdventureWorksLTConnectionString"
connectionString="Data Source=.\SQLEXPRESS;
AttachDbFilename=C:\Users\MainUser\Downloads\SQL2008
.AdventureWorksLT2008_Only_Database\AdventureWorksLT
2008_Data.mdf;Integrated Security=True;Connect Timeo
ut=30;User Instance=True"
```

```
        providerName="System.Data.SqlClient" />
  </connectionStrings>
<!—
.
—>
</configuration>
```

В командной строке выполните последовательно приведенные далее команды, заменяя UserName именем учетной записи, от имени которой запущено веб-приложение (например, учетная запись AspNet):

1. `cd \WINDOWS\Microsoft.NET\Framework\v2.0.50739`
2. `aspnet_regiis -pa "NetFrameworkConfigurationKey" "UserName"`
3. `aspnet_regiis -pe "connectionStrings" -app "/StagingWebsite"`

Выполнение этих команд модифицирует файл web.config следующим образом (если вы получите сообщение об ошибке, гласящее, что контейнер ключей RSA не найден, вам может потребоваться выполнить команду `aspnet_regiis -pc "NetFrameworkConfigurationKey" –exp` для создания этого контейнера):

```
<?xml version="1.0"?>
<configuration>
<connectionStrings
configProtectionProvider="RsaProtectedConfigurationProvider">
<EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
   xmlns="http://www.w3.org/2001/04/xmlenc#">
<EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
<EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<KeyName>Rsa Key</KeyName>
</KeyInfo>
<CipherData>
<CipherValue>Y4Be/ND8fXTKl3r0CASBK0oaOSvbyijYCVUudf1AuQl
pU2HRsTyEpR2sVpxrOukiBhvcGyWlv4EM0AB9p3Ms8FgIA3Ou6mGORhxfO9eIU
GD+M5tJSe6wn/9op8mFV4W7YQZ4WIqLaAAu7MKVI6KKK/ANIKpV8l2NdMBT3uP
OPi8=</CipherValue>
</CipherData>
</EncryptedKey>
</KeyInfo>
<CipherData>
<CipherValue>BeKnN/kQIMw9rFbck6IwX9NZA6WyOCSQlziWzCLA8Ff/J
dA0W/dWIidnjae1vgpS8ghouYn7BQocjvc0uGsGgXlPfvsLq18//1ArZDgiHVL
AXjW6b+eKbE5vaf5ss6psJdCRRB0ab5xaoNAPHH/Db9UKMycWVqP0badN+qCQz
YyU2cQFvK1S7Rum8VwgZ85Qt+FGExYpG06YqVR9tfWwqZmYwtW8izr7fijvspm
/oRK4Yd+DGBRKuXxD6EN4kFgJUil7ktzOJAwWly4bVpmwzwJT9N6yig54lobhO
ahZDP05gtkLor/HwD9IKmRvO1jv</CipherValue>
</CipherData>
</EncryptedData>
</connectionStrings>
<!—
.
—>
</configuration>
```

Как видно из этого примера, строка подключения в конфигурационном файле не видна. Показанные выше команды выполняют два действия. Первую команду можно игнорировать (она просто изменяет текущий каталог, чтобы вы могли обратиться

к выполняемому файлу asp_regiis). Вторая команда разрешает доступ к контейнеру ключей NetFrameworkConfigurationKey пользователю UserName. Этот контейнер является контейнером по умолчанию для RSAProtectedConfigurationProvider, указанного в файле machine.config. Чтобы ваше приложение могло дешифровать данные из конфигурационного файла, пользователь, от имени которого выполняется приложение, должен иметь права на доступ к указанному контейнеру. Для идентификации этого пользователя используется команда

```
System.Security.Principal.WindowsIdentity.GetCurrent().Name
```

Третья команда шифрует раздел connectionStrings конфигурационного файла для веб-приложения StagingWebsite. Другие разделы конфигурационного файла также могут быть зашифрованы той же командой. Если на некотором более позднем этапе вам потребуется дешифровать раздел конфигурационного файла, выполните ту же команду, но с ключом -pd вместо -pe, например:

```
aspnet_regiis -pd "connectionStrings" -app "/StagingWebsite"
```

Резюме

В этой главе показано, как использовать Visual Studio 2010, чтобы настроить свое приложение для подключения к базе данных с помощью строки подключения. Встроенная поддержка классов данных в .NET Framework позволяет динамически создавать и изменять строки подключения, так что вам никогда не придется заниматься ими вручную.

38

Файлы ресурсов

В ЭТОЙ ГЛАВЕ...

- Что такое ресурс приложения
- Определение и использование ресурсов в вашем приложении
- Определение ресурсов для разных культур
- Расширения типов ресурсов по умолчанию

Разработчики часто не придают значения скромным файлам ресурсов в XML-формате, так как они почти всегда скрываются Visual Studio 2010, чтобы не загромождать решение. Поскольку в основном они применяются в качестве файлов поддержки для форм или веб-страниц, вы можете создавать большие приложения и при этом непосредственно с файлами ресурсов не сталкиваться. Однако файлы ресурсов — важный инструмент, которым необходимо уметь пользоваться для того, чтобы создавать приложения, которые можно легко сопровождать и переводить на другие языки.

В первой части данной главы поясняется, почему файлы ресурсов так важны, и описываются возможности, которые они предоставляют разработчику. В остальной части главы поясняется, как применять файлы ресурсов для локализации приложения для различных языков и культур.

Что такое ресурсы

Ресурс — это данные, необходимые приложению, такие как строка, пиктограмма, изображение или даже аудиозапись. Ресурсы не являются выполняемыми и поддерживают работу приложения путем предоставления таких данных, как, например, положение, размер или иные физические свойства управляющих элементов. Хотя в основном ресурсы являются строками, изображениями, аудиозаписями и пиктограммами, они могут быть и более сложными объектами, поддерживающими сериализацию.

С приложением могут быть скомпилированы файлы ресурсов трех типов: текстовые файлы, resx-файлы (файлы ресурсов в формате XML) и бинарные файлы ресурсов. При необходимости в качестве ресурсов приложений могут встраиваться целые файлы. Большинство разработчиков, пользующихся Visual Studio 2010, применяют resx-файлы и встраиваемые файлы.

Текстовые файлы ресурсов

Текстовые файлы представляют собой самый простой тип ресурсов, поскольку они ограничены предоставлением только строковых значений. В приложениях, для работы которых требуется большое количество строк, применение простого текстового файла может быть простейшим способом обеспечения приложения ресурсами.

Формат строк, определенных в текстовом файле ресурсов, представляет собой пары "имя/значение", где имя используется для обращения к ресурсу в коде, как показано в следующем примере:

```
Error_Unable_To_Connect = Unable to connect to specified server
```

Поскольку каждая пара ограничена символом новой строки, этот символ не может быть добавлен в строку непосредственно. Однако можно прибегнуть к управляющим символам в стиле языка программирования C для вставки в текст символов новой строки (`\n`) или табуляции (`\t`).

В файл ресурсов можно вставлять комментарии, для чего следует предварить строку с комментарием точкой с запятой:

```
; Сообщение, которое отображается, если невозможно подключиться к серверу
Error_Unable_To_Connect = Unable to connect to specified server
```

Текстовые файлы ресурсов должны иметь расширение `.txt` или `.restext`. Последнее расширение может пригодиться, если вы хотите отличать текстовые файлы ресурсов от обычных текстовых файлов.

Хотя текстовые файлы ресурсов легко редактировать и обновлять, их труднее интегрировать в приложение. Текстовые файлы не могут быть непосредственно компилированы в приложение; они должны быть предварительно конвертированы в resx-файлы или бинарные файлы ресурсов. Это делается с помощью утилиты Resource Generator, `resgen.exe`, находящейся в папке `\bin` Windows SDK (т.е. ее полное имя — `C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin`):

```
resgen StringResources.txt StringResources.resources
```

Включив выходной файл (в данном случае — файл `StringResources.resources`) в приложение, можно обращаться к ресурсам из этого приложения.

Прежде чем файл ресурсов будет встроен непосредственно в приложение, текстовый файл ресурсов автоматически преобразуется в бинарный файл ресурсов. Этим обеспечивается доступ приложения к последней версии ресурсов. Чтобы обеспечить такое автоматическое конвертирование, включите текстовый файл ресурсов в приложение и установите свойство Build Action равным `None`. Перейдите в окно Project Properties проекта, в котором содержится текстовый файл ресурсов, и на вкладке Compile выберите команду Build Events. В поле prebuild events введите команду компиляции своего текстового файла ресурсов:

```
"C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin\resgen.exe"
       "$(ProjectDir)StringResources.txt"
"$(ProjectDir)StringResources.resources"
```

В ходе сборки приложения будет выполнена генерация файла ресурсов, который должен быть включен в приложение и свойство Build Action которого установлено равным EmbeddedResource. На рис. 38.1 показано включение текстового файла и файла ресурсов в приложение с соответствующими свойствами Build Action.

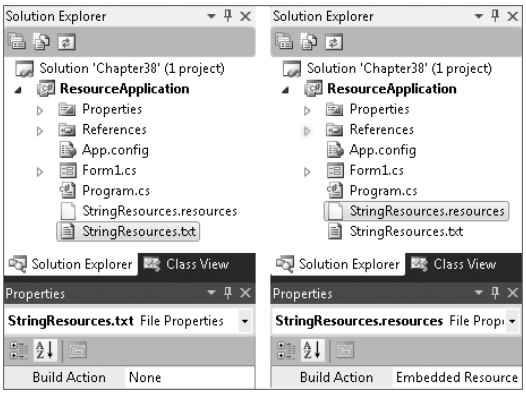


Рис. 38.1

Resx-файлы ресурсов

Более дружественным форматом для ресурсов являются XML-файлы ресурсов, которые обычно называются resx-файлами. Это простые файлы данных XML, которые содержат пары "имя/значение" узлов XML. Преимущество этого формата в том, что значения не ограничиваются только строками; они могут быть любого типа, который сериализуем или может быть представлен в виде строки.

Приведенный далее фрагмент XML демонстрирует ресурс HelloWorld и связанные с ним значение и комментарий. Как видно из кода, о типе данных нет никакой информации, поскольку это просто строковый ресурс:

```
<data name="HelloWorld">
<value> Say Hello </value>
<comment> This is how we say hello </comment>
</data>
```

В следующем фрагменте показано, как в файле ресурсов в виде строк могут храниться более сложные типы данных. Здесь также показано применение псевдонима сборки для обращения к внешней сборке с информацией о типе. При обращении к этому ресурсу информация о типе будет использована для преобразования строкового значения в объект данного типа:

```
<assembly alias="System.Drawing" name="System.Drawing, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
<data name="Button1.Location" type="System.Drawing.Point, System.Drawing">
<value> 71, 43 </value>
</data>
```

Хотя resx-файлы могут быть включены в приложение без использования генератора файлов ресурсов (Resgen), они все равно конвертируются перед компиляцией в приложение. В процессе построения для каждого resx-файла приложения генерируются соответствующие файлы ресурсов, которые впоследствии компонуются с приложением.

Бинарные ресурсы

Третьим форматом ресурсов является бинарный файл ресурса; для него используется расширение `.resources`. "За сценой" Visual Studio 2010 в качестве промежуточного шага при компиляции конвертирует все resx-файлы в `.resources`-файлы (эти файлы вы можете увидеть в папке `\obj\debug` вашего проекта). Текстовые файлы ресурсов, как вы читали ранее в этой главе, необходимо конвертировать в `.resources`-файлы вручную, с помощью утилиты Resgen. Вы можете интегрировать другие бинарные ресурсы в свой проект, просто включив в него `.resources`-файл и установив свойство Build Action равным `EmbeddedResource`.

Добавление ресурсов

Visual Studio 2010 предоставляет богатый пользовательский интерфейс для добавления и модификации файлов ресурсов. Вы можете просмотреть содержимое файла ресурсов в интегрированной среде. Однако, если ресурс не является строкой или не имеет строкового представления, вы не можете модифицировать его значение в файле ресурсов. Редактор ресурсов обеспечивает поддержку таких типов ресурсов, как строки, изображения, пиктограммы, аудиозаписи и др.

Дважды щелкнув на узле My Project (VB) или Properties (C#) проекта в Solution Explorer, можно открыть редактор свойств проекта и выбрать в нем вкладку Resources для того, чтобы открыть файл ресурсов по умолчанию (файл ресурсов проекта). В случае проекта C# затем требуется щелкнуть на представленной ссылке для создания файла ресурсов (проекты VB изначально имеют файл ресурсов по умолчанию). Когда файл ресурсов по умолчанию открывается, в верхнем левом углу редактора ресурсов появляется раскрывающийся список ресурсов разных типов, как показано на рис. 38.2. Двойной щелчок на любом resx-файле в Solution Explorer также приводит к запуску редактора ресурсов.

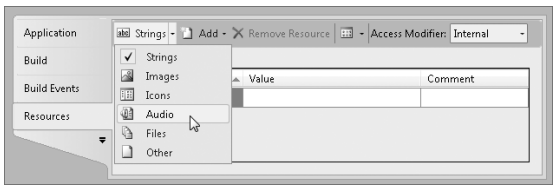


Рис. 38.2

Редактор выводит ресурс в формате, соответствующем его типу. Например, строки выводятся в редактируемых текстовых полях, а изображения — в виде миниатюр, которые можно открыть и редактировать. Добавление ресурсов также выполняется очень просто — путем выбора в раскрывающемся списке Add Resource соответствующего типа ресурса и добавления необходимой информации. После того как вы добавите новый ресурс, он появляется в редакторе ресурсов (рис. 38.3).

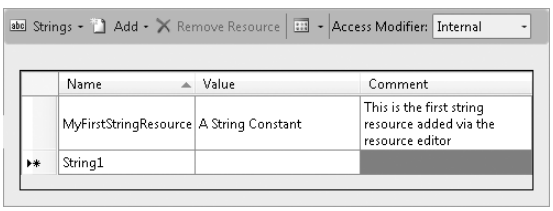


Рис. 38.3

На этом же рисунке виден дополнительный столбец, который позволяет добавить к ресурсу комментарий. К сожалению, редактор ресурсов — это единственное место в Visual Studio 2010, где можно увидеть эти комментарии.

Встраиваемые файлы как ресурсы

Зачастую необходимо встроить в приложение целый файл. Это можно сделать, включив файл в приложение и модифицировав его свойство Build Action. В зависимости от типа файла при включении такого элемента в приложение свойство Build Action (щелкните на файле и откройте окно Properties) обычно установлено равным либо `Compile`, либо `None`. Если изменить его на `EmbeddedResource`, то весь файл будет добавлен в приложение как встраиваемый ресурс.

Для добавления файла в качестве ресурса можно также воспользоваться редактором ресурсов, показанным на рис. 38.2. С помощью редактора ресурсов изображения, пиктограммы и другие файлы добавляются в существующий файл ресурсов, как элемент `resxfileref`. Файл появляется в каталоге ресурсов, но его свойство Build Action равно `None`. При построении приложения эти файлы перед компоновкой в приложение компилируются в файлы ресурсов. В прошлом данные из этих файлов извлекались и добавлялись в resx-файл в виде бинарного блока. Это означает, что, будучи добавленными, данные не могут быть легко модифицированы. В случае же ссылки на файл данные остаются в этом файле и могут быть легко обновлены.

Именование ресурсов

Ресурсы именуются с использованием имени файла ресурсов, которому они принадлежат, и корневого пространства имен. Например, если у вас имеется файл ресурсов `Sample.resources` в проекте `MyProject`, то полным именем ресурса будет `MyProject.Sample`.

Особенно важно это помнить, когда вы делаете файл встраиваемым ресурсом путем изменения свойства Build Action. Вы можете обращаться к любому файлу с использованием в качестве префикса имени проекта. В отличие от файлов ресурсов, имена прочих файлов сохраняют расширения. Так, если у вас есть файл `ASimpleDataDocument.doc` в проекте `MyProject`, то к нему следует обращаться как к `MyProject.ASimpleDataDocument.doc`.



При именовании встраиваемых ресурсов любая структура каталогов игнорируется.

Обращение к ресурсам

Метод, используемый для обращения к ресурсам, зависит от того, как они встроены в приложение. Вы уже знакомы с двумя способами встраивания ресурсов: первый — добавление файла в проект и установка свойства Build Action равным `EmbeddedResource`, второй — применение редактора ресурсов. Для обращения к ресурсам, добавленным путем изменения свойства Build Action, необходимо использовать методы `GetManifestResourceNames` и `GetManifestResourceStream`. Приведенный далее код получает имена всех ресурсов в сборке, запрашивая манифест, а затем создает поток для обращения к файлу ресурсов. Как говорилось в предыдущем разделе, имя встроенного файла ресурсов, возвращаемого методом `GetManifestResourceNames` и получаемого методом `GetManifestResourceStream`, имеет вид *Корневое_пространство_имен.Имя_файла.Расширение_файла* (например, `MyProject.AsimpleDataDocument.doc`).

VB

```
Dim names = Reflection.Assembly.GetExecutingAssembly. _
            GetManifestResourceNames
Dim resources = From n In names
    Select Assembly.GetExecutingAssembly. _
            GetManifestResourceStream(n)
For Each r In resources
    Using strm As New IO.StreamReader(r)
        MsgBox(strm.ReadToEnd)
    End Using
Next
```

C#

```
var names = Assembly.GetExecutingAssembly()
                    .GetManifestResourceNames();
var resources = from n in names
    select Assembly.GetExecutingAssembly()
                   .GetManifestResourceStream(n);
foreach (var r in resources){
    using (var strm = new StreamReader(r)){
        MessageBox.Show(strm.ReadToEnd());
    }
}
```

К ресурсам, добавленным с помощью редактора ресурсов, в коде можно обратиться с помощью менеджера ресурсов, который легко создается из имени файла ресурсов, к которому он относится, и ссылки на сборку, из которой должен быть извлечен ресурс.

VB

```
Dim res As New ResourceManager("WorkingWithResources.Resources",
                               Assembly.GetExecutingAssembly)
```

C#

```
var res = new ResourceManager("WorkingWithResources.Properties.Resources",
                              Assembly.GetExecutingAssembly());
```

Созданный ресурс можно извлечь с помощью функции `GetObject` или `GetString`:

```
res.GetObject("StringResource")
```

В случае более сложных ресурсов, таких как файлы, можно также воспользоваться функцией `GetStream`. Все три функции получают в качестве единственного параметра имя ресурса.

Файлы проектировщика

Утилита генерации ресурсов, Resgen, имеет ряд усовершенствований, которые позволяют создавать строгие типизированные классы-оболочки для ваших файлов ресурсов. Когда вы добавляете resx-файл к вашему приложению, Visual Studio 2010 автоматически создает файл проектировщика, который обеспечивает процесс создания менеджера ресурсов и обращения к ресурсам по имени. Соответствующие свойства строго типизированы и генерируются проектировщиком для снижения вероятности появления случайных некорректных обращений и преобразований типов. Например, если у вас есть строковый ресурс `StringResource`, содержащийся в файле ресурсов `MyResources`, то для обращения к строке можно использовать следующий код:

VB

```
My.Resources.MyResources.MyStringResource
```

C#

```
MyResources.StringResource
```

Обратите внимание на различие сгенерированного кода для VB и C#. Это связано с тем, что C# использует для интеграции файла ресурсов в пространство имен `My` обобщенный инструмент ResXFileCodeGenerator, в то время как в случае VB это инструмент VbMyResourcesResXFileCodeGenerator.

К сожалению, Visual Studio 2010 автоматически не генерирует файл проектировщика для текстовых файлов ресурсов, так как эти файлы не могут быть непосредственно добавлены в приложение. Процесс генерации файла ресурсов из текстового файла можно расширить, включив генерацию файла проектировщика.

Для этого при вызове Resgen добавляется новый аргумент:

```
resgen sample.txt sample.resources /str:vb
```

Чтобы можно было обращаться к ресурсам, к приложению должны быть добавлены оба выходных файла. Чтобы гарантировать корректность обращения к ресурсам, следует обеспечить соответствие имен, используемых в файле проектировщика, именам скомпилированных ресурсов. Для управления пространством имен, именем класса и выходным классом можно воспользоваться дополнительными параметрами:

```
resgen sample.txt defaultnamespace.sample.resources
        /str:vb,defaultnamespace,sample,sample.vb
```

В этом случае полностью квалифицированным выходным классом будет `defaultnamespace.sample`, а применение этого файла позволит обращаться к ресурсам без генерации исключений. Когда корректная команда вызова определена, можно соответствующим образом настроить соответствующее событие, генерируемое перед сборкой, чтобы в нем создавался файл проектировщика. При этом всякий раз при обновлении и сохранении файла и компиляции приложения файл проектировщика будет создаваться заново.

Приложение и ресурсы

Написание приложения часто требует для создания привлекательного внешнего вида различных данных, таких как изображения, пиктограммы или звуки (в целом именуемые *ресурсами*). Более того, практический опыт показывает, что следует избегать применения константных строк в приложениях. В качестве одного из решений можно разместить эти ресурсы в файлах, которые должны поставляться вместе с приложением.

Другой способ заключается во включении их в файл ресурсов, который затем можно скомпилировать с вашим приложением. Так вы не только получаете ресурсы в требующемся для работы формате, но и они автоматически становятся доступны вашему приложению.

В Visual Studio 2010 формы изначально представлены двумя файлами: сгенерированным файлом проектировщика (например, `Form1.Designer.vb`) и файлом с кодом (например, `Form1.vb`). Когда в форму впервые добавляется новый управляющий элемент, такой как кнопка, для формы автоматически создается файл ресурсов (например, `Form1.resx`). По умолчанию этот файл ресурсов содержит очень мало

данных, так как основные свойства жестко кодируются в файле проектировщика. Этот файл становится очень важным при установке возможности локализации формы. Когда это делается с помощью указания соответствующего свойства (рис. 38.4), свойства проектировщика данного управляющего элемента сохраняются в файле ресурсов.

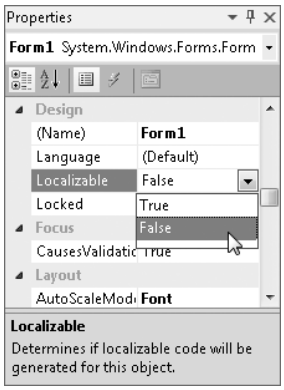


Рис. 38.4

Приведенный далее фрагмент кода показывает сгенерированный проектировщиком метод `InitializeComponent`, который создает и устанавливает свойства кнопки Button1. Вот как выглядит код, когда свойство формы `Localizable` установлено равным `False`:

```
Private Sub InitializeComponent()
    Me.Button1 = New Button
    '
    'Button1
    '
    Me.Button1.Location = New Point(71, 43)
    Me.Button1.Size = New Size(185, 166)
    Me.Button1.Text = "Button1"
    Me.Button1.TabIndex = 0
    Me.Button1.Name = "Button1"
    Me.Button1.UseVisualStyleBackColor = True
    '
    'Form1
    '
    Me.Controls.Add(Me.Button1)
End Sub
```

Если свойство формы `Localizable` установлено равным `True`, форма использует новый класс `ComponentResourceManager` для загрузки и применения свойств, найденных в прилагаемом файле ресурсов (этот класс будет подробнее рассматриваться далее в этой главе):

```
Private Sub InitializeComponent()
    Dim resources As _
            New ComponentResourceManager(GetType(Form1))
    Me.Button1 = New Button
    '
    'Button1
    '
    resources.ApplyResources(Me.Button1, "Button1")
    Me.Button1.Name = "Button1"
    Me.Button1.UseVisualStyleBackColor = True
    '
    'Form1
    '
    Me.Controls.Add(Me.Button1)
End Sub
```

Хотя генерируемые проектировщиком форм файлы ресурсов можно редактировать вручную, делать это не рекомендуется, поскольку изменения могут быть перезаписаны при очередной регенерации файла проектировщиком.

При корректном применении файлы ресурсов обеспечивают массу преимуществ, поскольку являются удобным местом для хранения строк, пиктограмм, изображений и прочих данных, к которым может обращаться приложение. Использование файлов ресурсов — для хранения как свойств форм, так и данных приложения — обязательно для приложений, которые планируется переводить для работы в иной культурной

среде (мы используем термин “культура”, поскольку различия между странами и группами населения не сводятся к различным языкам). Файлы ресурсов позволяют разработчикам предоставлять разные данные для разных культур. При запуске приложения .NET Framework использует информацию о текущей культуре для определения, какие именно данные должны быть загружены. Типичными примерами информации, которая может варьироваться в зависимости от культурных особенностей, являются заголовки, приглашения, сообщения об ошибках и изображения кнопок.

Изображения управляющих элементов



Рис. 38.5

Ряд управляющих элементов Windows Forms в качестве свойств имеют изображения. Например, управляющий элемент PictureBox имеет свойства `Image`, `ErrorImage` и `InitialImage`. Щелкните на троеточии в столбце значений окна Properties для любого из указанных свойств, и увидите диалоговое окно, показанное на рис. 38.5, которое позволяет выбрать изображение в качестве значения данного свойства.

Перед выбором изображения необходимо решить, где оно будет храниться: в файле ресурсов, связанном с текущей формой (т.е. с локальным ресурсом — выбор Local resource на рис. 38.5), или в файле ресурсов уровня проекта. Выбор первого варианта сохраняет изображение в формате блока в кодировке Base64 в самом файле ресурсов, в то время как второй добавляет изображение к проекту, а к выбранному файлу ресурсов — соответствующую ссылку. Ясно, что последний выбор обычно предпочтительнее, так как изображение можно будет изменить без обязательного последующего импорта.

Сопутствующие ресурсы

Одним из основных преимуществ размещения данных в файле ресурсов является возможность их перевода для иных культур. Вместо включения всех языков в один файл ресурсов данные для каждой культуры хранятся в файле ресурсов с суффиксом, определяемым целевой культурой.

Культуры

Культуры определяются комбинацией из двух строчных символов, представляющих язык, и двух прописных, представляющих страну или область культуры. Эти два пары символов разделены дефисом. Например, американский английский и австралийский английский представлены соответственно как `en-US` и `en-AU`. Соответствующими файлами ресурсов для указанных культур будут `MyResource.en-US.resx` и `MyResource.en-AU.resx`. Полный список идентификаторов культур можно найти по адресу `http://msdn2.microsoft.com/en-us/library/system.globalization.cultureinfo.`

aspx. Любопытства ради можете пересмотреть все доступные культуры — этот список возвращается методом CultureInfo.GetCultures(CultureTypes.AllCultures). Всего имеется около 220 культур, которые можно классифицировать следующим образом.

- **Инвариантная культура.** Идентификатор языка и страны отсутствует (например, Form1.resx). Данные от культуры не зависят, например это может быть логотип компании, который не изменяется и никак не зависит от информации о текущем культурном окружении.
- **Нейтральная культура.** Имеется только идентификатор языка (например, Form1.en.resx). Данные зависят только от языка, например простое предупреждающее сообщение, которое нужно просто перевести на другой язык.
- **Конкретная культура.** В наличии имеются идентификаторы как языка, так и страны (например, Form1.en-US.resx). Данные зависят как от языка, так и от страны или региона, например схема формы, цвет или приглашения должны быть переведены и соответствовать традициям конкретного региона.

Создание культурно-зависимых ресурсов

Создавая дополнительные файлы ресурсов для формы, не забудьте установить свойство Localizable равным True. Имеется три способа создания файлов ресурсов для конкретных культур.

- Если вы знаете идентификатор культуры, для которой хотите генерировать файл ресурсов, можете просто сохранить resx-файл с именем *имя_файла.идентификатор_культуры*.resx. Например, если вы переводите файл ресурсов Form1.resx на австралийский английский, сохраните его под именем Form1.en-AU.resx. Заметим, что при этом Visual Studio удаляет исходный resx-файл из решения и добавляет в него новый resx-файл, специфичный для конкретной культуры. Чтобы вложенными в узел Form1 оказались оба файла, следует исключить новый resx-файл, обновить вид решения (закрыв и заново открыв его), а затем поместить оба файла обратно, в проект.
- Visual Studio поддерживает существенно лучший способ создания культурно-зависимых файлов ресурсов для форм. В окне Properties формы можно выбрать Language. Имя этого свойства несколько обманчиво, так как оно позволяет обновить не только язык, но и страну или регион формы. Это свойство изначально установлено равным (Default) и должно всегда возвращаться к этому значению после того, как вы завершите генерацию или модификацию файлов ресурсов для конкретных культур. Для генерации файла ресурсов для австралийского английского выберите из раскрывающегося списка Language значение English (Australia) и внесите соответствующие изменения в форму. Завершив настройку новой планировки формы, сохраните ее и установите значение свойства Language равным (Default).
- Последний способ генерации культурно-зависимых файлов ресурсов заключается в использовании утилиты WinRes.exe. Хотя она и не добавляется в меню Start (Пуск), она доступна из папки Windows SDK (местоположение которой — C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin) и представляет собой графическую утилиту для генерации файлов ресурсов для форм. Эта утилита может загрузить существующий файл ресурсов, модифицировать свойства

всех управляющих элементов формы, а затем сохранить изменения в файл ресурсов для конкретной культуры. Перед тем как открыть файл ресурсов формы с помощью указанной утилиты, убедитесь, что свойство `Localizable` установлено равным `True`; в противном случае файл не будет корректно загружен.

Загрузка культурно-зависимых файлов ресурсов

Теперь вас может заинтересовать вопрос о взаимодействии файлов ресурсов и о том, должны ли файлы ресурсов создаваться и компилироваться одновременно с основным приложением. Ответ на оба эти вопроса кроется в механизме загрузки ресурсов классом `ResourceManager`.

Этот процесс имеет три уровня, основанные на текущей культуре пользовательского интерфейса (user interface culture — UI culture). Получить к ней доступ в коде можно с помощью свойства `CultureInfo.CurrentUICulture`. Учтите, что это свойство отлично от `CultureInfo.CurrentCulture`, которое представляет собой текущую культуру, используемую при сравнении строк, для форматирования дат и т.п. В отличие от текущей культуры, которая основана на региональных настройках компьютера (изменить которые можно, воспользовавшись командой меню Control Panel⇨Regional Settings (Панель управления⇨Язык и региональные стандарты)), культура пользовательского интерфейса по умолчанию зависит от выбранного языкового пакета пользовательского интерфейса Windows. Вы не можете изменить культуру пользовательского интерфейса по умолчанию для своего приложения, если только у вас не установлен Windows Multilingual User Interface Pack.

Хотя вы и не в состоянии изменить культуру пользовательского интерфейса по умолчанию, вы можете изменить это свойство в коде. Однако следует предупредить: без установленного пакета интерфейса некоторые культуры могут отображаться некорректно.

```
Thread.CurrentThread.CurrentUICulture = New CultureInfo("en-US")
```

Используя текущую культуру пользовательского интерфейса, механизм пытается локализовать ресурсы на основе соответствия культур. Например, если культура пользовательского интерфейса — `en-US`, механизм осуществляет поиск ресурса, соответствующего как языку (английский), так и стране (США). Если ресурс не найден, выполняется поиск только по языку (английский). Если ресурс все равно не найден, выполняется откат к *инвариантной культуре*.

Сопутствующие культурно-зависимые ресурсы

До сих пор мы упоминали только о том, как ресурс может быть преобразован для новой культуры и добавлен в приложение. Хотя этот метод дает вам полный контроль над тем, какие культуры будут распространяться вместе с приложением, было бы лучше, если бы вам не пришлось перестраивать все вое приложение при необходимости модифицировать ресурс для некоторой культуры, или при добавлении поддержки новой культуры.

Когда Visual Studio 2010 компилирует культурно-зависимые ресурсы, разделение файлов ресурсов можно описать как звезду, центром которой является основная сборка с инвариантным ресурсом. Затем для каждой культуры, для которой имеется файл ресурсов, создаются сопутствующие сборки-лучи. Имена сопутствующих сборок имеют вид `MyApp.resources.dll` и располагаются в подкаталогах, именованных в

соответствии с культурами. Хотя и имеется неявная связь между конкретными и нейтральными культурами (например, между `en-US` и `en`), сопутствующие сборки для обоих типов должны находиться в подкаталоге основного каталога.

Другой способ заключается в том, что основная сборка и/или сопутствующие сборки инсталлируются в глобальный кэш сборки (Global Assembly Cache — GAC). В этом случае каждая сборка должна быть названа таким образом, чтобы в кэше не было совпадающих имен.

Ясно, что рассматриваемый механизм должен работать со сборками как в GAC, так и в подкаталогах. Следовательно, для каждого уровня культуры (конкретного, нейтрального и инвариантного) первым проверяется GAC, а за ним — подкаталог культур. Наконец, если ресурс не найден, генерируется исключение.

Заметим, что культурно-зависимые файлы ресурсов не обязаны содержать все ресурсы, определенные в файле ресурсов по умолчанию. Рассмотренный механизм загружает ресурс из файла ресурсов по умолчанию, если он не найден в конкретном культурном файле, так что в культурно-зависимых файлах ресурсов имеет смысл хранить только те ресурсы, которые для данной культуры отличаются от ресурсов по умолчанию.

Доступ к ресурсам

В .NET Framework имеется масса средств, облегчающих выполнение стандартных задач по обращению к ресурсам. Сюда входят однострочная загрузка изображений, межсборочные ссылки, а также применение класса `ComponentResourceManager`.

Загрузка изображений и пиктограмм

Изображения и пиктограммы представляют собой два распространенных типа данных, хранимых в файлах ресурсов. Поэтому и класс `Bitmap`, и класс `Icon` имеют конструктор, который может создать экземпляр непосредственно из ресурса, не прибегая к помощи менеджера ресурсов. Например, если у вас есть изображение `MyImage.bmp`, которое вы включили в проект путем установки свойства Build Action равным `EmbeddedResource`, можете обратиться к этому изображению следующим образом:

```
Dim img As New Bitmap(GetType(ThisClass), "MyImage.bmp")
```

Здесь класс `ThisClass` может быть любым классом корневого пространства имен проекта, который содержит встроенный ресурс.

Межсборочные ссылки

В Visual Studio 2010 можно управлять уровнем доступности файлов ресурсов. Как показано на рис. 38.6, вы можете выбрать значение поля Access Modifier, определяющее, будет ли ресурс внутренним для сборки, в которой он определен (значение `Friend` [VB] или `Internal` [C#]), или общедоступным (`Public`).

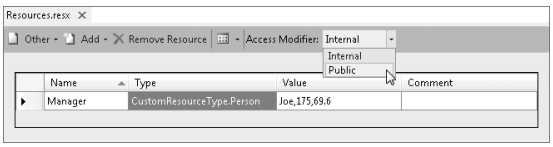


Рис. 38.6

При выборе значения Public к этому ресурсу можно обращаться из других сборок, указывая в качестве префикса имени ресурса имя сборки. Например, в приведенном далее фрагменте кода ресурс MyPerson находится в сборке CustomResourceType:

```
Dim p As Person = CustomResourceType.My.Resources.MyPerson
```

ComponentResourceManager

В первом примере в этой главе, после того как была включена локализация, для получения ресурсов, связанных с формой, использовался объект ComponentResourceManager. Он расширяет базовый объект ResourceManager, предоставляя дополнительную функциональность для получения и применения свойств компонента. Вот исходные строки, необходимые для установки свойств Button1:

```
Me.Button1.Location = New Point(71, 43)
Me.Button1.Size = New Size(185, 166)
Me.Button1.Text = "Button1"
Me.Button1.TabIndex = 0
```

При использовании ComponentResourceManager эти строки превращаются в одну:

```
resources.ApplyResources(Me.Button1, "Button1")
```

В предыдущих версиях Visual Studio код, сгенерированный при включении локализации, был существенно многословнее. Для каждого свойства выполнялся отдельный вызов ResourceManager для получения его по имени, как показано в приведенном далее фрагменте:

```
Me.Button1.Location = CType(resources.GetObject("Button1.Location"), Point)
Me.Button1.Size = CType(resources.GetObject("Button1.Size"), Size)
Me.Button1.TabIndex = CType(resources.GetObject("Button1.TabIndex"), Integer)
Me.Button1.Text = resources.GetString("Button1.Text")
```

Написание этого кода возможно и сейчас, поскольку метод GetObject объекта ComponentResourceManager остается доступным. Вопрос только в том, что каждое свойство, которое планируется локализовать, должно быть известно во время компиляции. Из-за этого каждое свойство каждого управляющего элемента добавляется в файл ресурсов. В результате в файл ресурсов добавляется чрезмерное количество свойств (даже те, значения которых не отличаются от значений по умолчанию). Это также приводит к большим накладным расходам при загрузке формы, поскольку каждое ее свойство устанавливается через свойство ресурса.

Метод ApplyResources класса ComponentResourceManager работает наоборот. Когда вы указываете имя управляющего элемента, которое должно быть единственным в пределах формы, извлекаются все ресурсы, имеющие данный префикс. После этого для определения свойства управляющего элемента используется полное имя ресурса. Например, для управляющего элемента Button1 будет извлечен ресурс Button1.Location, и его значение будет использовано для присвоения свойству Location этого управляющего элемента.

Этот процесс устраняет необходимость указывать все свойства в файле ресурсов. Он также требует, чтобы в файле ресурсов для конкретной культуры указывались дополнительные свойства, которые не определены в файле ресурсов по умолчанию.

Вас может заинтересовать вопрос о недостатках применения ComponentResourceManager. Для того чтобы установить свойство управляющего элемента с помощью имени этого свойства, для поиска ComponentResourceManager используется *отражение* (reflection). После получения свойство может быть активизировано. Каж-

дый поиск, выполняемый для установки свойства, относительно дорогостоящий. Однако сниженное количество устанавливаемых свойств, а также простота локализации без перекомпиляции основного приложения компенсируют эти недостатки.

Кодирование файлов ресурсов

В дополнение к богатому набору визуальных инструментов, предоставляемых Visual Studio 2010 для редактирования файлов ресурсов, для создания последних можно использовать код. .NET Framework предоставляет поддержку чтения и записи файлов ресурсов посредством двух интерфейсов: `IResourceReader` и `IResourceWriter`. После создания файлы ресурсов должны быть добавлены в приложение или связаны вручную, так, чтобы к ним можно было обратиться из приложения.

- `IResourceReader`. Гарантирует, что класс для чтения ресурса имеет следующие методы.
  - `GetEnumerator`. Получает объект `IDictionaryEnumerator`, который позволяет разработчику итеративно обойти все ресурсы в файле ресурсов.
  - `Close`. Используется для закрытия объекта для чтения ресурсов и освобождения всех связанных с ним ресурсов.
- `IResourceWriter`. Гарантирует, что класс для записи ресурса имеет следующие методы.
  - `AddResource`. Три перегрузки метода `AddResource` поддерживают добавление ресурсов в файл ресурсов. Обе реализации этого интерфейса в .NET Framework имеют либо дополнительную перегрузку данного метода, либо альтернативный метод для добавления ресурсов. Перегрузки, которые являются частью этого интерфейса, поддерживают добавление ресурсов парами "имя/значение". Каждый метод получает в качестве первого параметра имя, а в качестве второго — значение, такое как строка, массив байтов или объект. Последняя реализация, которая получает в качестве параметра объект, требует, чтобы он был сериализуем либо мог быть представлен в виде строки путем преобразования типов.
  - `Close`. Записывает ресурсы в поток перед его закрытием.
  - `Generate`. В отличие от метода `Close` просто записывает ресурсы в поток без его закрытия. После вызова этого метода вызов любого другого метода приводит к генерации исключения.

Реализации интерфейсов `ResourceReader` и `ResourceWriter`

`ResourceReader` и `ResourceWriter` представляют собой реализации интерфейсов `IResource...`, поддерживающие чтение и запись непосредственно из файлов ресурсов. Хотя чтение и запись в этом формате и являются наиболее прямым подходом, поскольку снижают необходимость в применении утилиты Resgen для генерации файла ресурсов, он ограничивает качество информации, которая может быть получена при чтении из файла. Каждый ресурс при этом рассматривается как последовательность байтов, тип которой неизвестен.

Реализации интерфейсов ResxResourceReader и ResxResourceWriter

ResxResourceReader и ResxResourceWriter — более универсальные реализации интерфейсов IResource. . . . В дополнение к поддержке указанных интерфейсов ResxResourceWriter поддерживает дополнительную перегрузку метода AddResource, с помощью которой может быть добавлен узел ResxDataNode. Класс ResxDataNode очень похож на запись словаря, так как имеет ключ (в нашем случае — свойство Name) и значение (которое вы должны указать при создании узла). Отличие же заключается в том, что данный узел может поддерживать дополнительные свойства, такие как комментарии или (в качестве альтернативы значению) ссылка на файл (например, с изображением, которое должно быть добавлено в файл ресурсов).

Как упоминалось ранее, в resx-файл можно добавлять ссылки на файлы, так что файл остается редактируемым. Соответствующий класс поддержки данной возможности называется ResxFileRef. Он может быть инстанцирован и добавлен в качестве ресурса посредством ResxResourceWriter. При этом в файл добавляется XML-узел наподобие приведенного ниже:

```
<data name="Figure_11_2"
      type="ResXFileRef, System.Windows.Forms">
  <value>.\Resources\CompanyLogo.tif;System.Drawing.Bitmap,
         System.Drawing,Version=4.0.0.0,Culture=neutral,
         PublicKeyToken=b03f5f7f11d50a3a </value>
</data>
```



Файлы ресурсов – наилучшее место для хранения статических данных приложений. Хотя они связываются с приложением в процессе компиляции, их содержимое легко извлечь и сделать удобочитаемым для человека. Однако в связи с этим файлы ресурсов не годятся для хранения секретных данных, таких как пароли или информация о кредитных картах.

Пользовательские ресурсы

Хотя Visual Studio предоставляет посредством файлов ресурсов неплохую поддержку разработки многоязычных приложений, иногда невозможно получить требуемый уровень контроля и управления, пользуясь только поведением по умолчанию. В этом разделе довольно подробно описывается, каким образом можно сериализовать пользовательские объекты для хранения в файлах ресурсов и как генерировать файлы проектировщика, обеспечивающие строго типизированный доступ к созданному файлу ресурсов.

Visual Studio 2010 позволяет хранить в файле ресурсов строки, изображения, пиктограммы и аудиофайлы. Все это можно сделать с помощью богатого пользовательского интерфейса. Для хранения в файле ресурсов более сложных типов данных их необходимо сериализовать в строковое представление, которое может быть включено в файл ресурсов.

Первый шаг при добавлении любого типа данных в файл ресурсов состоит в том, чтобы сделать этот тип данных сериализуемым. Этого легко добиться, пометив класс атрибутом Serializable. После этого объект можно добавлять в файл ресурсов с помощью реализации интерфейса IResourceWriter, например ResXResourceWriter:

VB

```
<Serializable()> _
Public Class Person
    Public Property Name As String
    Public Property Height As Integer
    Public Property Weight As Double
End Class

Dim p As New Person
p.Name = "Bob"
p.Height = 167
p.Weight = 69.5
Dim rWriter As New ResXResourceWriter("foo.resx")
rWriter.AddResource("DefaultPerson", p)
rWriter.Close()
```

C#

```
[Serializable()]
public class Person{
    public string Name   { get; set; }
    public int Height    { get; set; }
    public double Weight { get; set; }
}

var p = new Person(){
    Name = "Bob",
    Height = 167,
    Weight = 69.5
};
var rWriter = new ResXResourceWriter("foo.resx");
rWriter.AddResource("DefaultPerson", p);
rWriter.Close();
```

Однако сериализация объектов таким образом обладает парой недостатков.

- Чтобы включить файл ресурсов в приложение, сначала требуется воспользоваться кодом для его записи. Понятно, что с точки зрения администрирования это очень неприятно, так как предполагает дополнительную стадию в процессе построения приложения.
- Кроме того, сериализованное представление класса представляет собой бинарный объект и неудобочитаемо для человека. Это — в предположении, что генерирующий код написан корректно. К сожалению, это бывает далеко не всегда, так что удобочитаемость содержимого в среде Visual Studio 2010 существенно облегчила бы жизнь.

Обходным путем для решения обеих проблем является определение `TypeConverter` для данного класса и использование представления класса в виде строки. Такой ресурс может быть отредактирован в редакторе ресурсов Visual Studio. `TypeConverter` предоставляет механизм, посредством которого .NET Framework может определить, возможно ли представление класса (в данном случае — класса `Person`) в виде другого типа данных (в данном случае — строки). Первый шаг заключается в создании `TypeConverter` с применением `ExpandableObjectConverter`:

VB

```
Imports System.ComponentModel
Imports System.ComponentModel.Design.Serialization
Imports System.Globalization
```

```
Public Class PersonConverter
  Inherits ExpandableObjectConverter

  Public Overrides Function CanConvertFrom(ByVal context As _
                           ITypeDescriptorContext, _
                           ByVal t As Type) As Boolean

    If t Is GetType(String) Then Return True
    Return MyBase.CanConvertFrom(context, t)
  End Function

  Public Overrides Function ConvertFrom( _
                  ByVal context As ITypeDescriptorContext, _
                  ByVal info As CultureInfo, _
                  ByVal value As Object) As Object
    If (TypeOf (value) Is String) Then
      Try
        If value Is Nothing Then Return New Person()
        Dim vals = CStr(value).Split(",")c)
        If vals.Length < > 3 Then Return New Person()
        Return New Person With {.Name = vals(0), _
                  .Height = Integer.Parse(vals(1)), _
                  .Weight = Double.Parse(vals(2))}
      Catch
        Throw New ArgumentException("Can not convert '" & _
                                    value.ToString & _
                                    "' to type Person")
      End Try
    End If
    Return MyBase.ConvertFrom(context, info, value)
  End Function

  Public Overrides Function ConvertTo( _
                  ByVal context As ITypeDescriptorContext, _
                  ByVal culture As CultureInfo, _
                  ByVal value As Object, _
                  ByVal destType As Type) As Object
    If (destType Is GetType(String) And _
        TypeOf (value) Is Person) Then
      Dim c = TryCast(value, Person)
      Return c.Name & "," & c.Height.ToString & _
             "," & c.Weight.ToString
    End If
    Return MyBase.ConvertTo(context,culture,value,destType)
  End Function
End Class
```

C#

```
public class PersonConverter : ExpandableObjectConverter{
  public override bool CanConvertFrom(
            ITypeDescriptorContext context,
            Type t){
    if (typeof(string) == t) return true;
    return base.CanConvertFrom(context, t);
  }
  public override object ConvertFrom(
            ITypeDescriptorContext context,
            CultureInfo culture, object value){
    if (value is string){
      try{
        if (value == null) return new Person();
        var vals = (value as string).Split( ' , ' );
```

```
        if (vals.Length != 3) return new Person();
        return new Person{
          Name = vals[0],
          Height = int.Parse(vals[1]),
          Weight = double.Parse(vals[2])
        };
      }
      catch (Exception){
        throw new ArgumentException("Can not convert ' " +
                                    value.ToString() +
                                    " ' to type Person");
      }
    }
    return null;
  }
  public override object ConvertTo(
        ITypeDescriptorContext context,
        CultureInfo culture, object value, Type destType){
    if (typeof(string) == destType & & value is Person){
      var c = value as Person;
      return c.Name + "," + c.Height.ToString() + "," +
             c.Weight.ToString();
    }
    return base.ConvertTo(context, culture,
                          value, destType);
  }
}
```

Представленный класс должен иметь атрибут `TypeConverter`:

VB

```
<System.ComponentModel.TypeConverter( _
          GetType(PersonConverter))> _
<Serializable()> _
Public Class Person
    Public Property Name As String
    Public Property Height As Integer
    Public Property Weight As Double
End Class
```

C#

```
[System.ComponentModel.TypeConverter(typeof(PersonConverter))]
[Serializable()]
public class Person{
    public string Name   { get; set; }
    public int Height    { get; set; }
    public double Weight { get; set; }
}
```

Теперь можно добавить элемент в файл ресурсов с использованием строкового представления класса. Например, запись в resx-файле может выглядеть следующим образом:

```
<assembly alias="CustomResourceType"
          name="CustomResourceType, Version=1.0.0.0,
                Culture=neutral, PublicKeyToken=null" />
<data name="Manager"
      type="CustomResourceType.Person, CustomResourceType">
  <value> Joe,175,69.5 </value>
</data>
```



Создание ресурсов пользовательских типов — сложная задача, поскольку Visual Studio 2010 не обновляет ваш `TypeConverter` после первой загрузки. Вы можете либо строго именовать сборку, в которой располагается `TypeConverter`, и увеличивать номер версии после каждого внесения изменений, либо перезапускать Visual Studio, чтобы внесенные изменения вступали в силу.

Резюме

В этой главе продемонстрирована важность файлов ресурсов в XML-формате при создании приложений как для доступа к статическим данным, так и для упрощения локализации, для использования с другими языками и культурами. Богатый пользовательский интерфейс, предоставляемый Visual Studio 2010, позволяет легко добавлять в приложения такие ресурсы, как изображения, пиктограммы, строки, аудиофайлы и файлы других типов.

Встроенная поддержка локализации форм и генерации сопутствующих сборок помогает разработчикам создавать приложения, нацеленные на глобальный рынок. Вы также узнали, что пользовательский интерфейс, предоставляемый Visual Studio 2010, расширяем, а это означает возможность его модификации для работы с пользовательскими типами ресурсов.

ЧАСТЬ IX
Отладка

- **ГЛАВА 39.** Использование окон отладчика
- **ГЛАВА 40.** Отладка с применением точек прерывания
- **ГЛАВА 41.** Подсказки по данным, прокси и визуализаторы
- **ГЛАВА 42.** Отладка веб-приложений
- **ГЛАВА 43.** Передовые методы отладки

39

Использование окон отладчика

В ЭТОЙ ГЛАВЕ...

- Базовые концепции отладки в Visual Studio, включая точки прерывания и подсказки по данным (DataTips)
- Окна отладчика в Visual Studio
- Использование и развертывание исключений в процессе сеанса отладки

Отладка приложений — одна из самых сложных задач, стоящих перед разработчиками. Однако, корректно используя отладчик Visual Studio 2010, можно упростить анализ состояния приложения и причин его некорректной работы. В этой главе рассматриваются окна, которые Visual Studio 2010 предоставляет для поддержки программиста в ходе создания и отладки приложений.

Окно Редактирования кода

Наиболее важным окном для отладки является окно редактирования кода. Предоставляя возможность установки точек прерывания и пошагового выполнения кода, это окно является начальным пунктом почти для всех действий по отладке приложения. На рис. 39.1 показан простой фрагмент кода с установленной точкой прерывания и видимой текущей точкой выполнения.

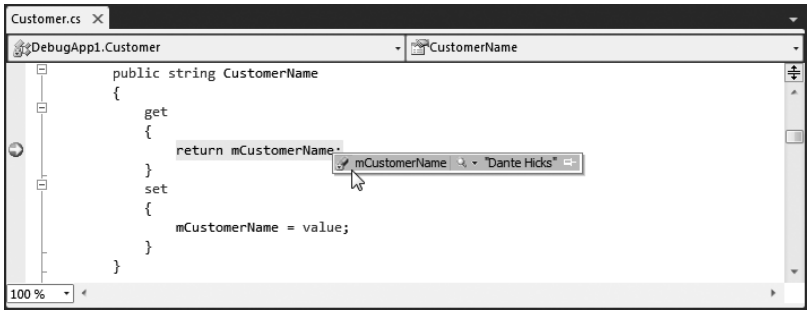


Рис. 39.1

Точки прерывания

Первый этап отладки приложения обычно заключается в поиске области кода, вызывающей ошибку. Это делается путем установки точек прерывания и постепенного прохождения кода. Более подробно установка точек прерывания и работа с текущей точкой выполнения рассматриваются в следующей главе. Хотя это и не видно на самом рис. 39.1, точки прерывания маркируются в окне кода красной точкой на полях страницы и красной подсветкой строки кода.

Когда при выполнении приложения встречается точка прерывания, текущая точка выполнения маркируется желтой стрелкой на поле, и соответствующая строка кода также подсвечивается желтым цветом. Как вы узнаете из следующей главы, этот маркер может перемещаться вперед и назад для управления порядком выполнения. Однако это должно делаться нечасто и с пониманием, поскольку такие действия модифицируют поведение приложения.

Подсказки DataTip

По достижении точки прерывания приложение приостанавливает работу и переходит в *режим прерывания* (break mode). В этом режиме можно получить информацию о текущем состоянии переменных, просто установив на имя переменной указатель мыши. Как показано на рис. 39.1, значением переменной `mCustomerName` является строка `"Dante Hicks"`. Эта отладочная подсказка называется *подсказкой по данным* (DataTip) и может использоваться для просмотра не только значений простых типов данных, таких как строки или целые числа, но и объектов более сложных типов данных, таких как объекты, созданные из нескольких вложенных классов.



Подсказки по данным используются и для запроса, и для редактирования значения переменной.

Из главы 41 вы узнаете о том, как можно перенастроить эти подсказки.

Окно точек прерывания

В ходе отладки сложного приложения для выявления проблемы можно установить ряд точек прерывания. К сожалению, этот подход имеет два побочных действия. Первое — выполнение приложения затруднено, так как вам то и дело приходится на-

жимать <F5> для продолжения работы. Второе, более важное, — выполнение приложения существенно замедляется из-за наличия *условных точек прерывания*, которые позволяют указать выражение, определяющее, должно ли приложение быть приостановлено. Чем более сложными являются условия точек прерывания, тем медленнее работает приложение. Так как точки могут быть разбросаны по нескольким исходным файлам, становится сложно найти и удалить те точки прерывания, которые больше не нужны.

Получить доступ к окну Breakpoints, показанному на рис. 39.2, можно с помощью меню Debug⇨Windows⇨Breakpoints. Оно предоставляет полезную информацию обо всех точках прерывания, установленных в настоящее время в приложении. Пользуясь этим окном, можно легко переходить от одной точки прерывания к другой, отключать их или удалять.

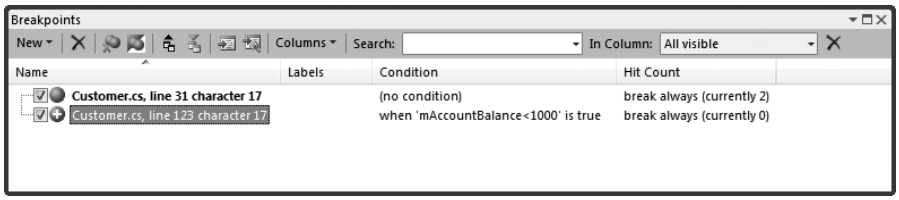


Рис. 39.2

На рис. 39.2 показаны две активные в настоящий момент точки прерывания в файле `Customer.cs`. Первая из них представляет собой обычную точку прерывания без условий. Вторая же имеет условие, которое гласит, что приложение должно приостанавливаться только тогда, когда значение переменной `mAccountBalance` меньше 1000. Это условие выделено шрифтом, поскольку приложение в настоящий момент находится в режиме паузы в данной точке прерывания.

Окно Breakpoints, подобно большинству других окон отладки, состоит из двух частей — панели инструментов и списка точек прерывания. На панель инструментов в Visual Studio 2010 добавлено несколько новых функций, включая поиск и импорт и экспорт точек прерывания. Эти функции рассматриваются в главе 40.

Каждый элемент списка точек прерывания представлен переключателем, указывающим, активна ли данная точка прерывания, пиктограммой и дескриптором точки прерывания, а также некоторым количеством столбцов, показывающих свойства точки прерывания. Эти столбцы можно выбирать из раскрывающегося списка Columns на панели инструментов. Установить дополнительные свойства точек прерывания можно, щелкнув правой кнопкой мыши на соответствующей точке прерывания.

Окно вывода

Одним из первых окон, с которым вы встречаетесь при первом запуске приложения, является окно Output. По умолчанию оно появляется всякий раз при сборке приложения и показывает течение процесса сборки. На рис. 39.3 показана успешная сборка демонстрационного приложения. В последней строке в окне Output подводятся итоги — в нашем случае это три успешно собранных проекта. В окне вывода имеется также информация о предупреждениях и ошибках, происшедших в процессе сборки (в данном случае ошибок нет). Хотя окно Output и может оказаться полезным, если

по каким-то причинам сборка неожиданно окажется неудачной, лучше для информации о сообщениях об ошибках и предупреждениях использовать список Error List.

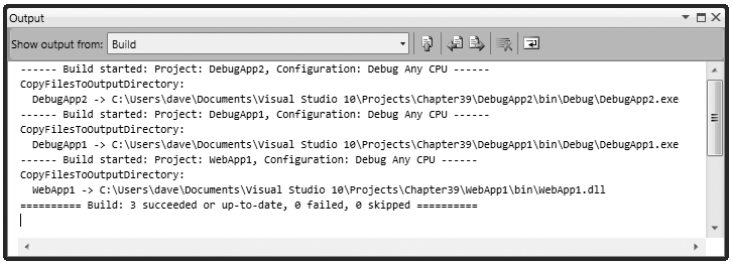


Рис. 39.3

Окно Output играет второстепенную роль в качестве стандартного средства вывода при выполнении приложения. Раскрывающийся список в левой части панели инструментов можно использовать для переключения между источниками вывода. На рис. 39.3 показан вывод процесса сборки, но при выполнении других действий в системе Visual Studio в этом списке появляются и другие элементы. Например, в отладочном режиме система Visual Studio создает запись Debug, после выбора которой видны все сообщения, выводимые кодом с помощью метода `Debug.Write` или `Debug.WriteLine`. Аналогично элемент Refactor создается для вывода результатов последней выполненной операции рефакторинга.



Вывод внешних инструментов, таких как `.bat` и `.com`-файлы, обычно направляется в окно Command. Этот вывод может быть показан и в окне Output при установке опции Use Output Window в диалоговом окне Tools⇨External Tools.

Прочие пиктограммы панели инструментов позволяют переходить к источнику сообщения, перемещаться к предыдущему или следующему сообщению, очищать содержимое окна и переключать режим переноса на новую строку в окне Output.

Окно Immediate

Достаточно часто при написании кода или отладке приложения возникает желание вычислять простые выражения — для тестирования некоторой части функциональности или для напоминания самому себе, как работает тот или иной код. В этом случае оказывается полезным окно Immediate (Debug⇨Windows⇨Immediate). Оно позволяет выполнять вводимые выражения. На рис. 39.4 показан ряд инструкций — от простого присваивания и операций вывода до более сложных создания объекта и манипуляций им.

На рис. 39.4 показан новый объект `Customer`, созданный в проекте C# в окне Immediate. В проекте Visual Basic нельзя явно объявлять переменные (например, `Dim x as Integer`). Вместо этого можно объявить переменную неявно путем присваивания.

Одна из наиболее полезных возможностей окна Immediate заключается в его применении при написании кода. Когда вы создаете объекты в окне Immediate в процессе проектирования, вызов конструктора и создание экземпляра объекта происходят без выполнения остального кода вашего приложения.

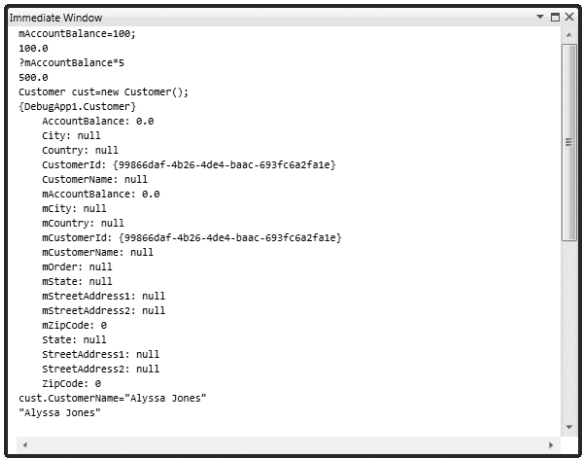


Рис. 39.4

Если вы вызываете метод или свойство, которое содержит активную точку прерывания, Visual Studio переходит в режим отладки и приостанавливает выполнение. Это особенно полезно при работе с определенным методом, который вы хотите протестировать, не запуская всего приложения.

Окно Immediate частично поддерживает технологию IntelliSense, и вы можете воспользоваться клавишами управления курсором, чтобы просмотреть историю команд.



Технология IntelliSense поддерживается в окне Immediate только в режиме отладки (не в процессе проектирования).

Окно Immediate позволяет также выполнять команды Visual Studio. Для передачи такой команды следует ввести знак “больше” (>) в начале строки. Доступен огромный набор команд; практически любое действие, которое может быть выполнено в системе Visual Studio, доступно как команда. К счастью, технология IntelliSense облегчает работу со списком доступных команд.

Имеется также почти сто предопределенных синонимов команд. Один из чаще всего используемых синонимов — символ “?”, который используется в качестве синонима команды `Debug.Print` для вывода значения переменной. Полный список предопределенных синонимов можно получить, введя в окне `>alias`, как показано на рис. 39.5.

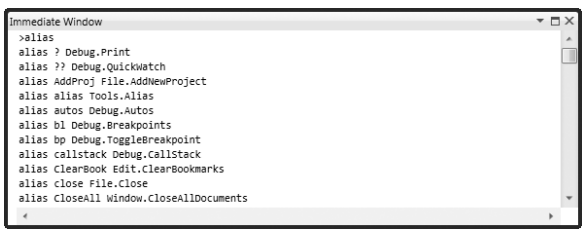


Рис. 39.5

Окна Watch

Ранее в этой главе вы видели, как можно использовать подсказки DataTip в окне редактирования кода для изучения значений переменных путем установки на их имена указателя мыши. Когда структура объекта становится более сложной, использовать

только DataTip становится трудно. У Visual Studio 2010 имеется ряд окон Watch, которые могут применяться для вывода переменных с простым интерфейсом, позволяющим изучать их структуру.

Окно QuickWatch

Окно QuickWatch (Debug⇨QuickWatch) представляет собой модальное диалоговое окно, которое может быть вызвано щелчком правой кнопки мыши в окне кода. Все выделенное в окне кода вносится в поле Expression диалогового окна, как показано на рис. 39.6, где вы видите объект `Customer`. В раскрывающемся списке Expression можно найти введенные ранее выражения.

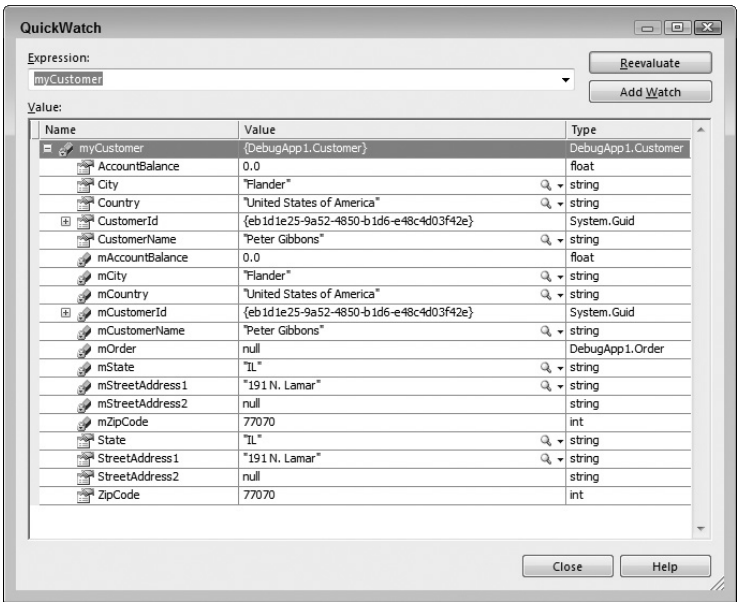


Рис. 39.6

Схема размещения дерева Value в окне QuickWatch аналогична подсказкам DataTip. Каждая строка содержит имя переменной, текущее значение и тип объекта. Значение переменной можно изменить, введя его в столбце Value.

Кнопка Add Watch позволяет добавить текущее выражение в одно из окон Watch. За этими переменными ведется постоянное наблюдение.

Окна Watch 1–4

В отличие от окна QuickWatch, которое является модальным и показывает значение переменной в определенной точке выполнения, окна Watch могут использоваться для отслеживания значений переменной в процессе пошагового выполнения кода. Хотя имеется четыре таких окна Watch, в большинстве случаев достаточно и одного. Наличие четырех отдельных окон означает, что вы можете иметь различные наборы переменных в разных окнах, что может оказаться полезным при работе со сложными приложениями со многими классами.

На рис. 39.7 показаны классы `Order` и `Customer` в окне Watch (Debug⇨Windows⇨Watch 1 (2–4)). Как и в случаях рассматривавшихся ранее окна QuickWatch и подсказок DataTip, пользовательский интерфейс может применяться для детального изучения более сложных типов данных.

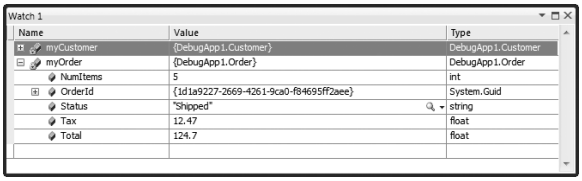


Рис. 39.7

Чтобы отслеживать дополнительные переменные, либо введите их в столбец Name в пустой строке, либо щелкните правой кнопкой мыши на имени такой переменной в окне кода и выберите в контекстном меню Add Watch.

Окна Autos и Locals

Окна Autos и Locals представляют собой два специальных окна Watch, в которые переменные автоматически добавляются отладчиком. Окно Autos (Debug⇨Windows⇨Autos) содержит переменные, использующиеся в текущей, предшествующих и последующих строках кода. Аналогично окно Locals (Debug⇨Windows⇨Locals) показывает все переменные, использующиеся в текущем методе. Если не считать автоматического заполнения этих окон, они ничем не отличаются от окон Watch.

Окна выполнения кода

В дополнение к изучению содержимого переменных во время сеанса отладки вы можете внимательно проверить логику своего кода и убедиться, что все работает так (и в том порядке), как вы и ожидали. В Visual Studio 2010 есть группа окон отладчика, которая показывает, что именно загружено и выполнено к моменту приостановки работы приложения. Это помогает лучше понять поведение кода при выполнении и быстро обнаружить логические ошибки.

Окно Call Stack

С ростом сложности приложений становится все сложнее отслеживать пути выполнения. Использование высоких деревьев наследования и интерфейсов часто запутывают путь выполнения. В этом случае на помощь приходит окно стека вызовов. Каждый путь выполнения должен иметь конечное количество записей в стеке (кроме случаев циклического вызова, когда неминуемо происходит переполнение стека). Просмотреть содержимое стека можно с помощью окна Call Stack (Debug⇨Windows⇨Call Stack), показанного на рис. 39.8.

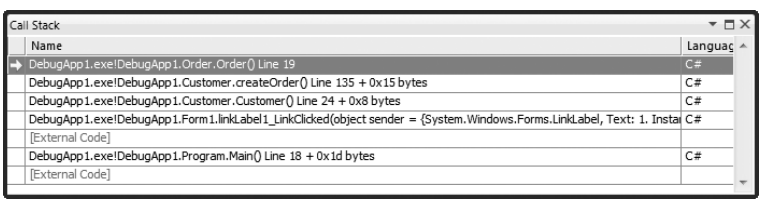


Рис. 39.8

Используя окно Call Stack, легко пройти по пути выполнения и определить, как и где был вызван текущий метод. Это можно сделать, щелкая на строках стека вызовов, известных как *кадры стека* (stack frame). Другие действия доступны через контекстное меню, открывающееся после щелчка правой кнопкой мыши. В него входят дизассемблирование определенного кадра стека, установка точек прерывания и просмотр различной информации.

Окно Threads

Большинство современных приложений используют несколько потоков выполнения. Что касается приложений Windows, то для повышения скорости отклика пользовательского интерфейса задачи, потребляющие большое количество процессорного времени, часто работают в отдельных потоках выполнения, будучи, таким образом, отделенными от основного приложения. Конечно, параллельное выполнение потоков делает отладку более сложной, в особенности когда потоки обращаются к одним и тем же классам и методам.

На рис. 39.9 показано окно Threads (Debug⇨Windows⇨Threads), в котором перечислены все активные потоки некоторого приложения. Обратите внимание, что в дополнение к потокам, создаваемым в коде, отладчиком создаются дополнительные потоки. Для простоты все потоки, используемые приложением, включая поток главного пользовательского интерфейса, получают имена, так что каждый поток легко отличить от других.

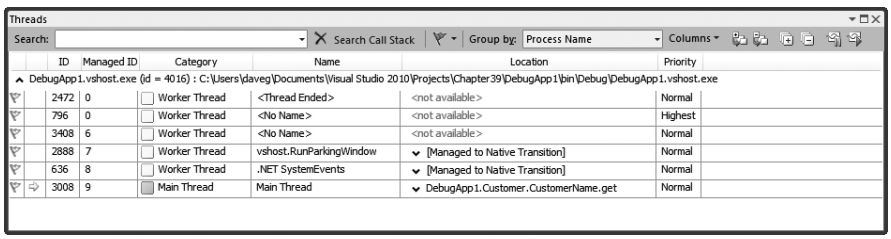


Рис. 39.9

Окно Threads выводит желтую стрелку, указывающую просматриваемый в настоящее время в окне кода поток. Для перехода к другому потоку просто дважды щелкните на нем, чтобы текущее местоположение в этом потоке было выведено в окне кода, а окно стека вызовов содержало информацию о данном потоке.

В режиме прерывания прерывается выполнение всех потоков приложения. Однако при пошаговом проходе вашего кода в отладчике очередная выполняемая инструкция может находиться как в интересующем вас потоке, так и в некотором другом. Если вас интересует путь выполнения одного потока, а выполнение других может быть приостановлено, щелкните правой кнопкой мыши на потоке в окне Threads и выберите в контекстном меню команду Freeze. Чтобы возобновить работу приостановленного потока, выберите в том же меню команду Thaw.

Отладка многопоточных приложений рассматривается в главе 43.

Окно Modules

Окно Modules (Debug⇨Windows⇨Modules), показанное на рис. 39.10, выводит список сборок, к которым обращается данное приложение. Для этих сборок, которые

составляют приложение, также выполняется загрузка отладочной информации, а это означает, что их можно отлаживать без дизассемблирования. Это окно особенно полезно, если вы хотите выяснить, какая версия сборки загружена и откуда.

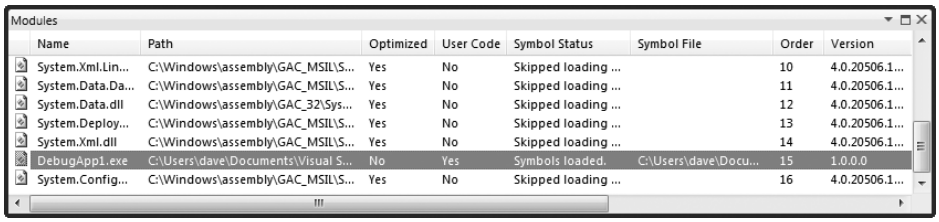


Рис. 39.10

На рис. 39.10 отладочная информация загружена для приложения `DebugApp1.exe`. Все прочие сборки пропущены, поскольку не содержат пользовательского кода и оптимизированы. Если доступен соответствующий файл с символьной информацией, его можно загрузить для сборки с помощью команды Load Symbols контекстного меню, вызываемого щелчком правой кнопкой мыши.

Окно Processes

Создание многоуровневых приложений может быть очень сложным, и зачастую требует одновременной работы всех уровней. Система Visual Studio 2010 позволяет запустить несколько проектов одновременно, обеспечивая сквозную отладку приложений. Альтернативное решение заключается в подключении к другим процессам для отладки приложений. Каждый раз, когда Visual Studio подключается к процессу, этот процесс добавляется к списку в окне Processes (Debug⇨Windows⇨Processes). На рис. 39.11 показано решение, включающее два приложения Windows и веб-приложение.

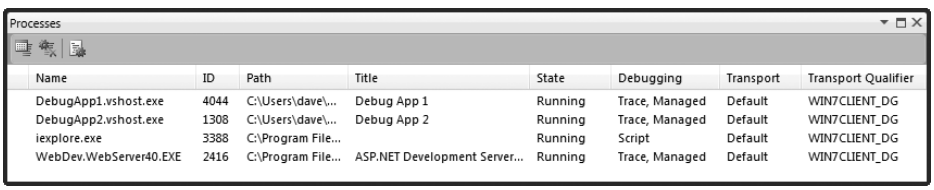


Рис. 39.11

Панель инструментов в верхней части окна Processes позволяет отсоединить или завершить присоединенный процесс, или подключиться к другому процессу.

Окна памяти

Три следующие окна обычно используются для низкоуровневой отладки, когда исчерпаны все прочие способы. Работа непосредственно с ячейками памяти, использование дизассемблирования или просмотр значений регистров процессора требуют фундаментальных знаний и терпения для анализа и использования низкоуровневой информации такого рода. Отладка на таком низком уровне требуется достаточно редко и представляет собой скорее исключение, чем правило.

Окна Memory 1–4

Четыре окна Memory могут использоваться для просмотра содержимого памяти по определенному адресу. В то время как окна Watch, Autos и Locals позволяют просмотреть содержимое переменных, хранящихся в определенных положениях в памяти, окно Memory показывает общую картину всего, что хранится в памяти.

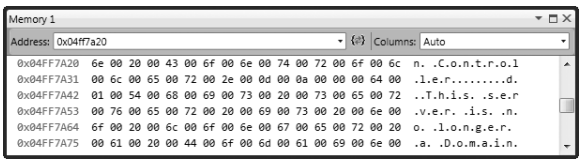


Рис. 39.12

Каждое из четырех окон Memory (Debug⇨Windows⇨Memory 1 (2–4)) позволяет просматривать содержимое разных адресов памяти, что облегчает процесс отладки приложения. На рис. 39.12 приведен пример информации, которая может быть предоставлена данным окном. Полоса прокрутки в правой части окна может использоваться для перемещения вперед и назад по адресам памяти и просмотра того, что в ней хранится.

Disassembly

Периодически возникают интересные споры об относительной производительности двух разных блоков кода. Иногда такие дискуссии превращаются в споры о количестве использованных инструкций в промежуточном представлении, и выяснение, насколько быстрее работает код, имеющий на одну инструкцию меньше. Что ж, если некоторый блок кода выполняется миллионы раз, то и одна инструкция может дать существенный выигрыш в производительности. Однако гораздо чаще небольшой высокоуровневый рефакторинг приводит к существенно большей экономии времени и вызывает существенно меньшее количество споров. На рис. 39.13 показано окно Disassembly (Debug⇨Windows⇨Disassembly) с фрагментом кода обработки щелчка мышью на LinkLabel; этот код как раз готов создать объект `Customer`. Вы видите инструкции, выполняющие это действие.

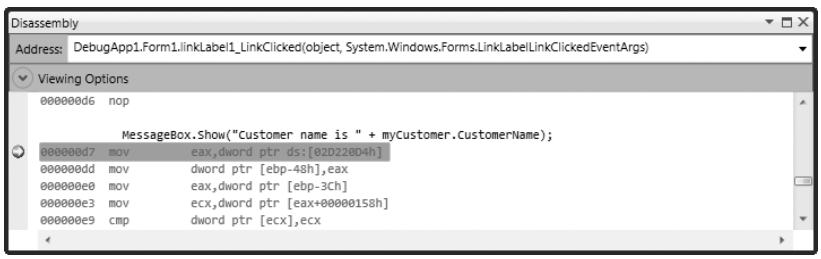


Рис. 39.13

На рис. 39.13 показано, что точка прерывания установлена на вызове конструктора и что точка выполнения совпадает с точкой прерывания. Оставаясь в этом окне, вы можете пройти по коду и увидеть, какие машинные инструкции при этом выполняются.

Окно Registers

При пошаговом проходе по машинным инструкциям в окне Disassembly очень сложно отследить, как загружается и перемещается разная информация, а также просмотреть содержимое регистров процессора. Окно Registers (Debug⇨Windows⇨Registers), показан-

ное на рис. 39.14, позволяет отследить содержимое разных регистров. Изменяющиеся значения регистров выделены красным цветом, что позволяет сразу увидеть, что происходит в каждой строке окна Disassembly.

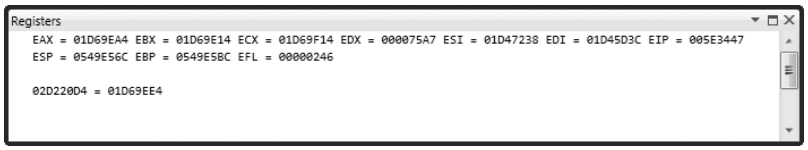


Рис. 39.14

Инструмент IntelliTrace (только в версии Ultimate edition)

Одной из весьма интересных новых возможностей Visual Studio является инструмент IntelliTrace. Одно из ограничений традиционных отладчиков заключается в том, что они показывают только снимок состояния приложения в отдельный момент времени. Инструмент IntelliTrace в Visual Studio собирает информацию в процессе сеанса отладки, тем самым позволяя вам вернуться в более раннюю точку и просмотреть состояние приложения в тот момент времени.



Инструмент IntelliTrace можно рассматривать как аналог "черного ящика" самолета в отладчике, который занимается записью всего происходящего при отладке.

IntelliTrace имеет два уровня сбора информации. По умолчанию он собирает информацию только о диагностических событиях, таких как вход в режим прерывания, пошаговый проход по коду в отладчике или генерация исключения. Вы можете также настроить IntelliTrace для сбора очень подробной информации, такой как детали каждого вызова функции, включая переданные ей параметры и возвращаемые значения.

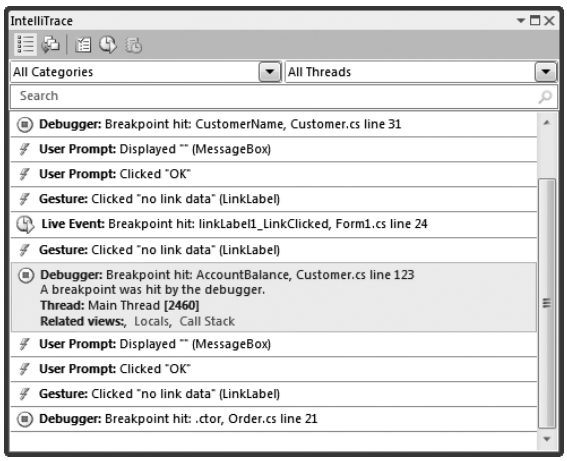


Рис. 39.15

Окно IntelliTrace Events (Debug⇨Windows⇨IntelliTrace Events), показанное на рис. 39.15, позволяет перемещаться по предыдущим диагностическим сообщениям. После щелчка на прошедшем событии точка выполнения в окне кода сменяет метку с желтой стрелки на красную с пиктограммой секундомера. Стек вызовов также обновляется, отображая соответствующее состояние приложения.

Если включен уровень сбора детальной информации, можете использовать окна Autos и Locals для изучения содержимого переменных.

Уровень детализации собираемых данных можно изменить (или полностью отключить сбор) на вкладке IntelliTrace диалогового окна, вызываемого командой меню Tools⇨Options. Можно также настроить IntelliTrace так, чтобы при сборе информации исключить некоторые сборки из данного процесса.



Собирая детальную информацию, необходимо ожидать определенного снижения производительности приложения. Следует также убедиться в наличии достаточного дискового пространства для собираемых данных. При сборе детальной информации, кроме того, отключается функциональность Edit and Continue.

Инструмент IntelliTrace позволяет работать и с журналами, создаваемыми новыми инструментами Visual Studio — Test and Lab Manager. Дополнительную информацию о IntelliTrace можно найти в главе 55.

Окна параллельной отладки

В настоящее время практически невозможно приобрести новый компьютер с одним процессором. Тенденция использования *многоядерных* процессоров (необходимая из-за достигнутых физических ограничений в области архитектуры процессоров) как основной способ получения более быстрых компьютеров, очевидно, будет иметь место и в дальнейшем.

К сожалению, программное обеспечение, которое не предназначено специально для многопроцессорных систем, не работает быстрее на многоядерном процессоре. Это становится проблемой для массы пользователей, ожидающих, что на новом компьютере их приложения будут работать быстрее.

Решение заключается в том, чтобы гарантировать параллельное выполнение различных частей кода на нескольких процессорах. Традиционный подход состоит в разработке программного обеспечения, работающего с несколькими потоками или процессами. К сожалению, написание и отладка многопоточных приложений очень сложна и чревата ошибками даже для опытного разработчика.

Компания Microsoft, понимая возникающие перед разработчиками проблемы, добавила ряд новых возможностей в Visual Studio 2010 и .NET Framework 4.0, которые призваны упростить создание такого программного обеспечения. Библиотека *Task Parallel Library* (TPL) — это набор расширений платформы .NET Framework, предоставляющих соответствующую функциональность. Библиотека TPL включает новые языковые конструкции, такие как циклы `Parallel.For` и `Parallel.ForEach`, а также новые коллекции, разработанные специально для параллельного обращения, включающие `ConcurrentDictionary` и `ConcurrentQueue`.

В новом пространстве имен `System.Threading.Tasks` есть несколько новых классов, которые существенно уменьшают усилия, прикладываемые при написании многопоточного и асинхронного кодов. Класс `Task` очень похож на поток, однако он более простой и лучше работает.

Написание параллельных приложений — это лишь часть всего цикла разработки. Требуются также эффективные средства отладки параллельных приложений. Поэтому Visual Studio 2010 вводит два новых окна отладки — Parallel Stacks и Parallel Tasks.

Окно Parallel Stacks

Ранее в данной главе рассказывалось об окне Call Stacks, которое можно использовать для просмотра пути выполнения текущей строки кода при отладке. Одним из ограничений данного окна является то, что одновременно можно видеть только один стек вызовов. Чтобы просмотреть стеки вызовов других потоков, следует воспользоваться окном Threads или панелью инструментов Debug Location для переключения отладчика на другой поток.

Окно Parallel Stacks (Debug⇨Windows⇨Parallel Stacks), показанное на рис. 39.16, является одним из наиболее полезных окон при отладке многопоточных и распараллеленных приложений. Оно предоставляет не только способ одновременного просмотра нескольких стеков вызовов, но и графическую визуализацию выполнения кода, включая демонстрацию того, как связаны несколько потоков и их совместные пути выполнения.

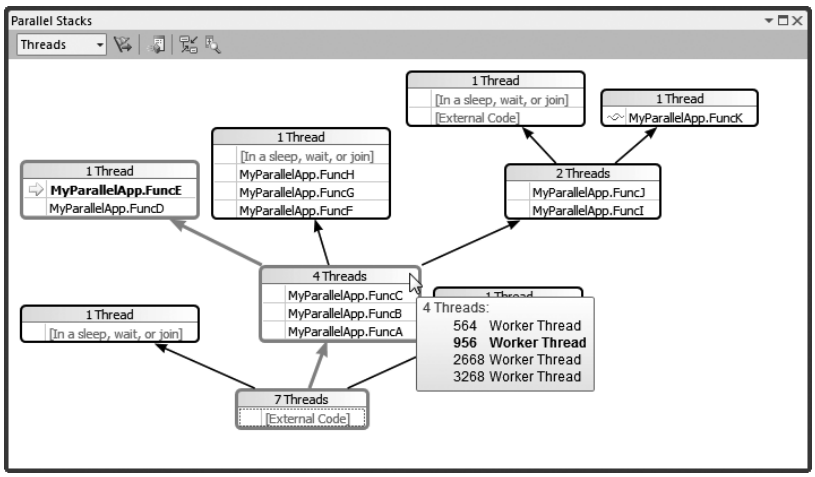


Рис. 39.16

На рис. 39.16 в окне Parallel Stacks показано приложение, в котором в настоящий момент работают семь потоков. Диаграмма вызовов читается снизу вверх. Поток `Main` находится в одном прямоугольнике, а четыре других потока сгруппированы в другом. Причина этого в том, что данные четыре потока совместно используют один стек вызовов (т.е. каждый поток вызвал функцию `FuncA` , которая затем вызвала `FuncB`, а та, в свою очередь, — `FuncC`). После вызова `FuncC` их пути кодов разошлись. Один поток вызвал `FuncD`, которая затем вызвала `FuncE`. Другой поток вызвал сначала `FuncF`, `FuncG`, а затем — `FuncH`. Два оставшихся потока вызвали `FuncI`, которая вызвала `FuncJ`, и т.д. Как видите, одновременная визуализация всех стеков вызовов дает не просто историю одного потока, а гораздо лучшее понимание состояния приложения в целом и того, как это состояние достигнуто.

На этом экране используется ряд других пиктограмм. Точка выполнения текущего потока показана желтой стрелкой. На рис. 39.16 она находится у функции `FuncE` в прямоугольнике в левой части диаграммы. Каждый прямоугольник, через который проходит соответствующий путь выполнения, выделен голубым цветом. Волнистые линии около `FuncK` в правой верхней части диаграммы указывают, что это текущая точка выполнения потока, не являющегося текущим.

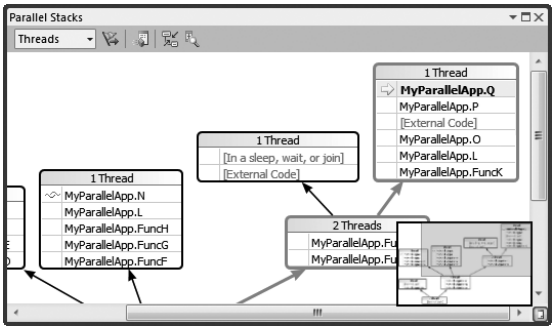


Рис. 39.17

Как видно из рис. 39.16, при наведении указателя мыши на метку с количеством потоков на вершине прямоугольника появляется информация об идентификаторах потоков. Можно также щелкнуть правой кнопкой мыши на каждой записи в стеке вызовов, чтобы получить доступ к различным функциям, таким как переход к соответствующей строке исходного текста в редакторе кода или переключение визуализации на другой поток.

В процессе работы с приложением, которое использует ряд потоков или задач или имеет очень глубокий стек, может оказаться, что визуализация Parallel Stacks не помещается в одно окно. В таком случае можно щелкнуть на пиктограмме в нижнем правом углу окна для вывода схемы окна, которая позволяет легко перемещаться по всей визуализации (рис. 39.17).

Окно Parallel Tasks

В начале данного раздела мы говорили о новой библиотеке Task Parallel Library в .NET Framework 4.0, которая включает класс `Task` в сборке `System.Threading.Tasks` и новые циклы `Parallel.For`. Окно Parallel Tasks (Debug⇨Windows⇨Parallel Tasks), показанное на рис. 39.18, помогает в отладке приложений, использующих эти новые возможности, путем вывода списка с состояниями всех текущих задач.

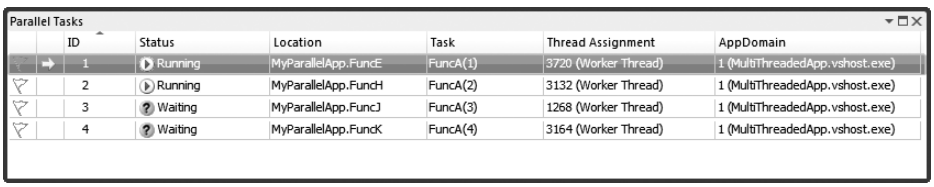


Рис. 39.18

Приостановленное приложение, показанное на рис. 39.18, создало четыре задачи, две из которых работают, а две находятся в состоянии ожидания. Вы можете щелкнуть на пиктограмме флажка, чтобы отметить одну или более задач для отслеживания.



Циклы `Parallel.For`, `Parallel.ForEach` и библиотека Parallel LINQ (PLINQ) используют класс `System.Threading.Tasks.Task` как часть своей реализации.

Исключения

Система Visual Studio 2010 содержит интеллектуальный обработчик исключений, который предоставляет большое количество полезной информации. На рис. 39.19 показано диалоговое окно Exception Assistant, появляющееся при генерации исключения. В дополнение к предоставлению детальной информации, в нем также выво-

дится ряд возможных действий. Список Actions зависит от типа сгенерированного исключения. В данном случае в нем два пункта — детальный просмотр исключения и копирование информации о нем в буфер обмена.

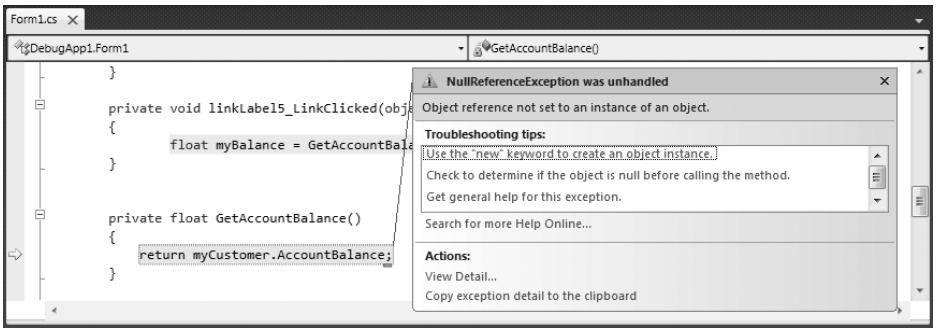


Рис. 39.19

Если вы выберете в списке действие View Detail, то будет выведено модальное диалоговое окно, предоставляющее анализ сгенерированного исключения. На рис. 39.20 показаны атрибуты исключения, включая состояние стека Stack Trace, которое может быть просмотрено полностью после щелчка на стрелке, направленной вниз, в правой части окна.

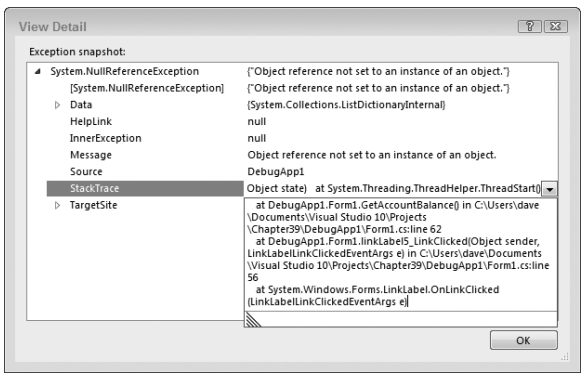


Рис. 39.20

Конечно, иногда исключения используются преднамеренно, для управления путем выполнения приложения. Например, некоторый пользовательский ввод может не соответствовать ограничениям формата, и применение регулярного выражения для получения информации не приводит к требуемым результатам. В таком случае функция анализа пользовательского ввода генерирует исключение, которое может быть легко перехвачено без прерывания всего приложения.

По умолчанию все исключения перехватываются отладчиком, так как предполагается, что при нормальной работе приложения исключения генерироваться не должны. В специально оговоренных ситуациях — как, например, неверный пользовательский ввод — может потребоваться игнорировать определенные типы исключений. Это можно сделать с помощью окна Exceptions, доступного из меню Debug.

На рис. 39.21 показано окно Exceptions (Debug⇨Exceptions), в котором перечислены все типы исключений, имеющиеся в .NET Framework. У каждого исключения имеются два переключателя. Отладчик может остановить выполнение при генерации исключения независимо от того, обработано ли оно. Если установлен флажок User-unhandled, отладчик будет останавливать выполнение приложения при любом исключении, не обработанном в пределах пользовательского кода. Дополнительную информацию по этому вопросу можно получить в главе 41, где рассматриваются атрибуты отладки.

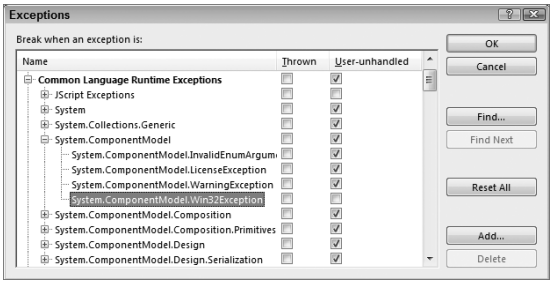


Рис. 39.21

К сожалению, в окне Exceptions не встречается ни одно из пользовательских исключений, которые может создать программист, но их можно добавить вручную, с помощью кнопки Add в правом нижнем углу окна. Нужно убедиться в том, что вводится полное имя класса, включая пространство имен; в противном случае отладчик не будет останавливать приложение при обработанном исключении. Понятно, что необработанные исключения будут приводить приложение к краху.

Настройка инструмента Exception Assistant

Как и в случае многих иных настраиваемых частей Visual Studio 2010, информация, выводимая инструментом Exception Assistant, хранится в XML-файле (`C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\Exception AssistantContent\1033\DefaultContent.xml`). Этот файл можно модифицировать как для изменения информации о существующих типах исключений, так и для добавления собственных пользовательских типов исключений. Если у вас есть собственные типы исключений, лучше создать свой XML-документ. Просто поместите его в тот же каталог, что и файл `DefaultContent.xml`; этого достаточно для регистрации его Visual Studio при очередной отладке вашего приложения. В приведенном далее фрагменте показан пример такого XML-файла:

```
<?xml version="1.0" encoding="utf-8" ?>
<AssistantContent Version="1.0" xmlns="urn:schemas-microsoft-com:xml-msdata:
  exception-assistant-content">
  <ContentInfo>
    <ContentName> Additional Content </ContentName>
    <ContentID> urn:exception-content-microsoft-com: visual-studio-7-
defaultcontent
    </ContentID>
    <ContentFileVersion> 1.0 </ContentFileVersion>
    <ContentAuthor> David Gardner </ContentAuthor>
    <ContentComment>My Exception Assistant Content for Visual Studio
    </ContentComment>
  </ContentInfo>
  <Exception>
    <Type> DebugApp1.myException </Type>
    <Tip HelpID="http://www.professionalvisualstudio.com/MyExceptionHelp.htm">
      <Description>Глупая ошибка, вам следует подучиться...
      </Description>
    </Tip>
  </Exception>
</AssistantContent>
```

В этом примере регистрируется справочная информация для типа исключения `myException`. Атрибут `HelpID` используется для ссылки на дополнительную информацию об исключении. При генерации данного исключения отладчик выводит окно, показанное на рис. 39.22.

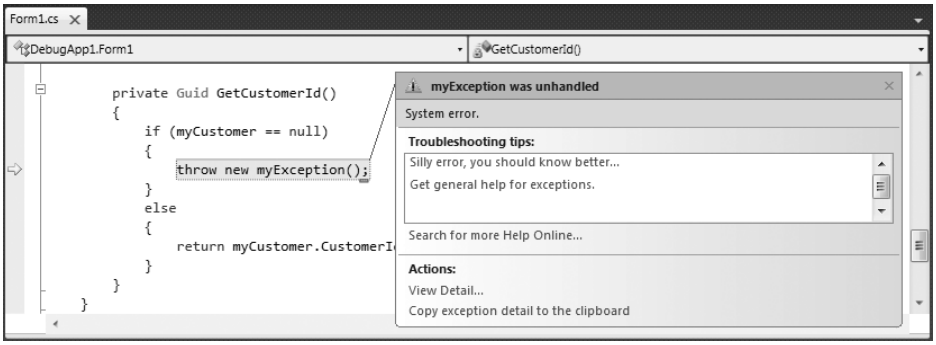


Рис. 39.22

Развертывание исключения

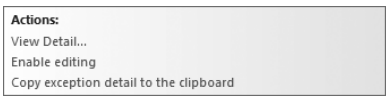


Рис. 39.23

На рис. 39.23 в списке Actions имеется дополнительный элемент – Enable editing. Это возможность развернуть выполнение приложения непосредственно перед генерацией исключения. Другими словами, вы можете отладить исключение, не перезапуская сеанс отладки.

Пункт Enable editing появляется, только если вы настроили систему Visual Studio таким образом, что при генерации исключения выполняется останов приложения. Как и многие другие возможности отладки, и Exception Assistant, и возможность развертывания исключения можно отключить на вкладке Debugging окна Options.



Альтернативным способом развертывания исключения является выбор команды Unwind to This Frame из контекстного меню, появляющегося после щелчка правой кнопкой мыши в окне Call Stack после генерации исключения. Эта возможность может быть полезна для проверки состояния приложения непосредственно перед генерацией приложения. Вы можете развернуть приложение только в том случае, если оно обработано (т.е. сгенерировано в блоке `try...catch`). Нужно также убедиться, что отладчик приостанавливает выполнение приложения при генерации исключения. Это можно сделать в окне Debug⇨Exceptions.

Резюме

В этой главе описаны все окна отладчика системы Visual Studio. Хотя, на первый взгляд, количество окон может показаться слишком большим, каждое из них решает свою отдельную задачу либо предоставляет доступ к определенной информации о работающем приложении. Вы можете легко перемещаться между окнами, переходя к тому из них, которое предоставляет наиболее важную в данный момент информацию.

Из следующей главы вы узнаете о том, как можно настраивать отладочную информацию, включая изменение информации, выводимой подсказками DataTip, и визуализацию информации о более сложных переменных.

40

Отладка с применением точек прерывания

В ЭТОЙ ГЛАВЕ...

- Использование точек прерывания, условных точек прерывания и точек трассировки для приостановки выполнения кода
- Управление выполнением программы в процессе отладки путем пошагового прохода кода
- Модификация кода в процессе работы с помощью функциональной возможности Edit and Continue

Давно минули те времена, когда отладка подразумевала вывод множества дополнительных сообщений для того, чтобы обнаружить точку, в которой наблюдается некорректная работа приложения. Система Visual Studio 2010 предоставляет богатую интерактивную среду отладки, которая оснащена такими средствами, как точки прерывания, точки трассировки и функциональная возможность Edit and Continue. Из данной главы вы узнаете, как использовать эти средства для отладки своего приложения.

Точки прерывания

Точка прерывания (breakpoint) применяется для приостановки выполнения приложения в определенном месте кода. Приложение переходит в режим прерывания (break mode), и при этом активизируется ряд окон Visual Studio 2010. Например, для просмотра значений переменных можно воспользоваться окном Watch. На рис. 40.1 показана точка прерывания, добавленная в конструктор класса Customer. Приложение останавливается в указанной строке при вызове конструктора класса Customer.

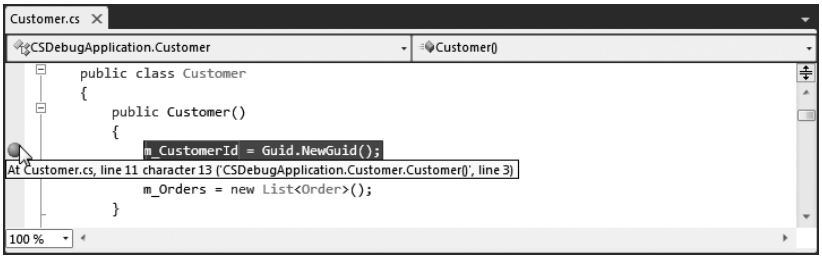


Рис. 40.1

Установка точки прерывания

Точки прерывания можно устанавливать с помощью либо меню Debug (используя команду Breakpoint в вызываемом правой кнопкой мыши контекстном меню), либо клавиши <F9>. Редактор кода Visual Studio 2010 позволяет также устанавливать точки прерывания, щелкая кнопкой мыши на полях.

Приложение может быть приостановлено только в строке выполняемого кода. Это означает, что точка прерывания, устанавливаемая на комментарии или объявлении переменной, будет перемещена при выполнении приложения к следующей строке выполняемого кода.

Простые точки прерывания

Точку прерывания можно установить на строке кода, расположив на этой строке указатель мыши и воспользовавшись любым из перечисленных ниже методов.

- Выберите команду Toggle Breakpoint меню Debug.
- Выберите Breakpoint⇨Insert Breakpoint в контекстном меню, которое появляется после щелчка правой кнопкой мыши.
- Воспользуйтесь клавишей <F9>.
- Однократно щелкните мышью на поле окна с кодом. На рис. 40.1 показано положение указателя мыши непосредственно после установки точки прерывания таким образом.

Если выбрать команду Breakpoint⇨Location из контекстного меню, открывающегося после щелчка правой кнопкой мыши на строке кода, будет выведено диалоговое окно File Breakpoint, показанное на рис. 40.2. На рисунке показана точка прерывания, установленная в строке 11 файла `Customer.cs`. Показан также номер символа, который выводится в случае наличия в одной строке нескольких инструкций.

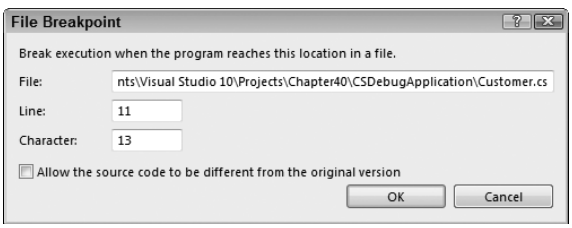


Рис. 40.2

Точки прерывания функций

Еще одним типом точек прерывания являются точки прерывания функций. Обычно установить точку прерывания функции можно, выбрав ее сигнатуру и нажав клавишу <F9> или воспользовавшись для создания точки прерывания мышью. При наличии перегрузок требуется найти их все и установить соответствующие точки прерывания. Установка точек прерывания функций позволяет устанавливать точки прерывания путем указания имени функции.

Для установки точки прерывания функции выберите команду меню Debug⇨ New Breakpoint⇨Break at Function. Появится диалоговое окно New Breakpoint, показанное на рис. 40.3, в котором можно указать имя функции, после вызова которой будет приостанавливаться выполнение приложения. Здесь при установке соответствующего переключателя можно воспользоваться технологией IntelliSense. Рекомендуется оставлять этот флажок установленным, так как без поддержки технологии IntelliSense корректно установить точку прерывания очень сложно.

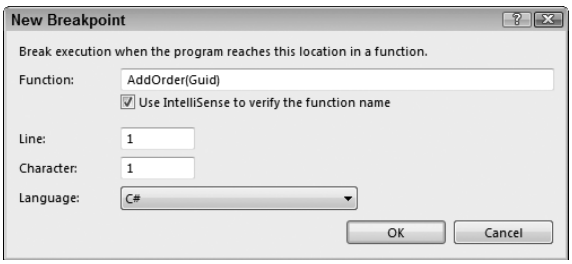


Рис. 40.3

К сожалению, в отличие от других окон отладки, здесь вы не получаете истинные возможности технологии IntelliSense при вводе имени. Однако, если вы выберете имя функции в окне кода перед тем, как приступить к созданию точки прерывания, оно будет автоматически вставлено в диалоговое окно.

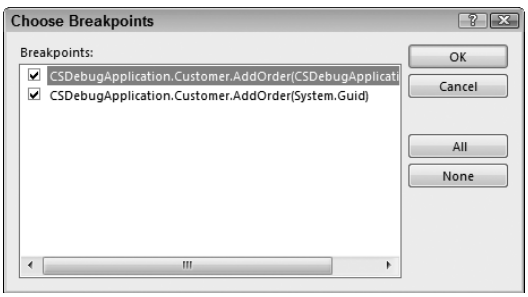


Рис. 40.4

При установке точки прерывания функции можно указать либо точную перегрузку функции, для которой вы хотите установить точку прерывания, либо имя функции. На рис. 40.3 выбрана перегрузка функции с одним параметром `Guid`. Обратите внимание, что, в отличие от полной сигнатуры метода, когда требуется имя параметра, для выбора конкретной перегрузки функции нужно указать только тип этого параметра. Если вы опустите информацию о параметрах, и при этом имеется несколько перегрузок функции, то получите приглашение выбрать перегрузку, для которой следует создать точку прерывания, как показано на рис. 40.4.

Точка прерывания по адресу

Еще один способ установки точек прерывания — с помощью окна Call Stack. Когда приложение находится в режиме прерывания, в стеке вызовов отображается текущий

список вызовов функций. После выбора любой строки в стеке вызовов точка прерывания может быть установлена точно так же, как и точка прерывания в файле, как описывалось ранее (меню Debug⇨Toggle Breakpoint, клавиша <F9> или команду Breakpoint⇨Insert Breakpoint контекстного меню). На рис. 40.5 показан небольшой стек вызовов с вновь установленной точкой прерывания на событии формы `Form1`.

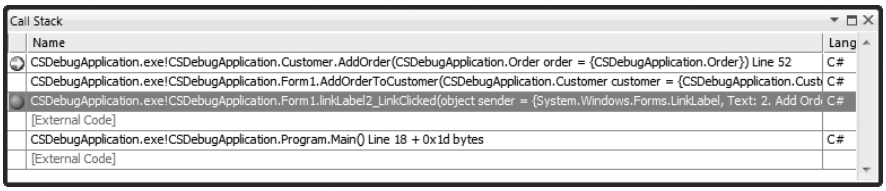


Рис. 40.5

Стек вызовов генерируется с использованием адресов функций, так что устанавливаемые точки прерывания являются точками прерывания по адресу. Этот тип точек прерывания полезен только в пределах одного сеанса отладки, так как адреса функций при внесении изменений и повторной сборке приложения, скорее всего, будут совершенно иными.

Добавление условий прерывания

Хотя точки прерывания и полезны для прерывания работы приложения в определенной точке и просмотра значений переменных и потока выполнения, если вы работаете с определенным сценарием, еще более полезной может оказаться возможность прерывания приложения только при соблюдении некоторых условий. Точки прерывания могут быть настроены на останов при выполнении определенных условий, при выполнении определенного количества итераций и т.п.

Условие

Условие точки прерывания может быть указано путем выбора команды Breakpoint⇨Condition в контекстном меню точки прерывания, появляющимся после щелчка правой кнопкой мыши. При этом открывается диалоговое окно Breakpoint Condition, показанное на рис. 40.6. Оно принимает логическое выражение, которое определяет, должна ли сработать точка прерывания. Если результатом вычисления выражения является значение `false`, приложение продолжает выполняться.

На рис. 40.6 точка прерывания установлена в классе `Order`, а условие заключается в том, что значение `OrderTotal` должно быть больше 1000. Как и в большинстве окон отладки, поле Condition поддерживает технологию IntelliSense, помогающую записать корректное условие. В случае указания неверного условия отладчик выводит соответствующее сообщение об ошибке, а приложение останавливается при первом же достижении этой точки прерывания.

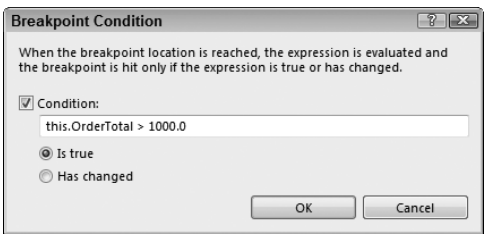


Рис. 40.6

Когда точке прерывания назначается условие (или количество проходов, как показано в следующем разделе), ее внешний вид изменяется — в красном кружке появляется белый крестик. Когда указатель мыши находится поверх этого кружка, выводится подсказка с полезной информацией, как показано на рис. 40.7.

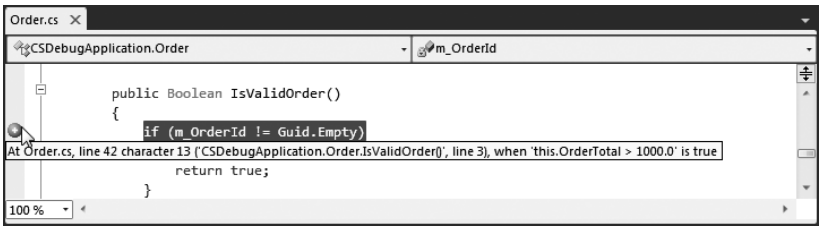


Рис. 40.7

Зачастую важнее знать, когда происходит изменение условия, чем когда оно истинно. Выбрав переключатель Has Changed, можно остановить приложение при изменении состояния условия. Если этот переключатель выбран, при первом достижении точки прерывания приложение не прекратит работу, поскольку значение условия сравнить не с чем.



Использование нескольких точек прерывания со сложными условиями может существенно замедлить выполнение вашего приложения, так что рекомендуется удалять все ненужные точки прерывания, чтобы они не мешали работе приложения.

Количество проходов точки прерывания

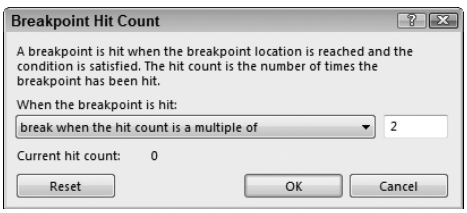


Рис. 40.8

Пожалуй, эта настройка точки прерывания не такая полезная, как условие, но и она может пригодиться для остановки приложения после определенного количества проходов точки прерывания. Для этого выберите в контекстном меню команду Breakpoint⇨Hit Count. На рис. 40.8 показано диалоговое окно Breakpoint Hit Count, которое можно использовать для указания, когда должна сработать точка прерывания.

После запуска приложения значение счетчика сбрасывается. Его значение может быть сброшено и вручную с помощью кнопки Reset. Для каждой точки прерывания имеется свой счетчик. Условие может быть одного из четырех видов.

- **Всегда.** Прерывание приложения не зависит от значения счетчика.
- **Равно.** Прерывание происходит, когда значение счетчика равно указанному.
- **Кратно.** Прерывание выполняется, когда значение счетчика кратно указанному (как показано на рис. 40.8).
- **Больше или равно.** Прерывание происходит, когда значение счетчика больше указанного или равно ему.

На рис. 40.9 показано окно Breakpoints, которое предоставляет дополнительную информацию о статусе каждой точки прерывания. В нашем случае точка прерывания настроена для приостановки работы приложения при каждом втором проходе. В настоящий момент значение счетчика равно 2.

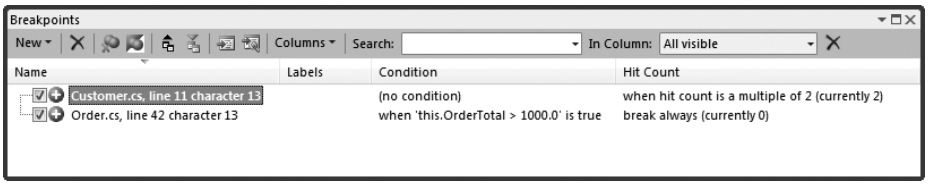


Рис. 40.9

Фильтр

Одно решение может состоять из нескольких приложений, которые должны работать одновременно. Это обычный сценарий при создании многоуровневого приложения. Когда приложение работает, отладчик может подключиться ко всем процессам, позволяя их отлаживать. По умолчанию по достижении точки прерывания приостанавливаются все процессы. Данным поведением отладчика можно управлять в окне Options, доступном из команды Options меню Tools. Уберите отметку переключателя Break All Processes When One Process Breaks, и процессы можно будет отлаживать по одному.

Если точка прерывания установлена в библиотеке классов, используемой более чем одним процессом, то по достижении этой точки будет приостановлено выполнение каждого процесса. Для того чтобы выполнить отладку только одного из этих процессов, можете добавить к точке прерывания фильтр, который ограничит ее действие только теми процессами, которые вас интересуют. Если вы отлаживаете приложение на нескольких машинах, можете создать фильтр, работающий по имени машины.

Фильтрация может использоваться даже в многопоточном приложении, если вы хотите ограничить точки прерывания конкретным потоком. Хотя точка прерывания срабатывает, только если поток отвечает критериям фильтра, приостановлены при этом будут все потоки. На рис. 40.10 показано диалоговое окно Breakpoint Filter и возможные условия фильтрации.



Рис. 40.10

Работа с точками прерывания

Зачастую необходимо изменить точку прерывания, поскольку она может находиться в неверной позиции или стать ненужной. В большинстве случаев проще удалить точку прерывания, но в некоторых ситуациях — например, при наличии сложного условия — может оказаться необходимым внесение изменений в существующую точку прерывания.

Удаление точек прерывания

Чтобы удалить ненужную точку прерывания, выберите ее либо в редакторе кода, либо в окне Breakpoints и удалите с помощью команды Toggle Breakpoint меню Debug. Можно также воспользоваться командой Delete Breakpoint контекстного меню, вызываемого щелчком правой кнопкой мыши, или пиктограммой Delete Breakpoint панели инструментов окна Breakpoints.

Отключение точек прерывания

Вместо удаления точки прерывания зачастую ее достаточно просто отключить с помощью команды Disable Breakpoint контекстного меню, вызываемого щелчком правой кнопки мыши. Можно также воспользоваться переключателем напротив точки прерывания в окне Breakpoints. На рис. 40.11 показана отключенная точка прерывания в окне кода.

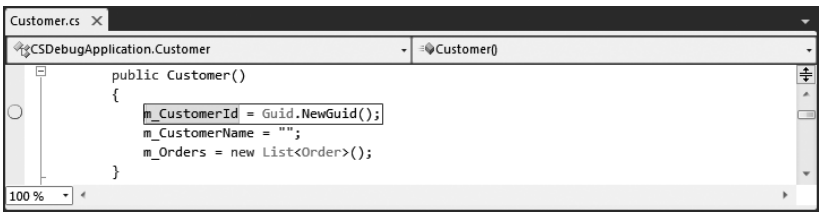


Рис. 40.11

Изменение местоположения точки прерывания

Местоположение точки прерывания может быть изменено путем выбора команды Breakpoint⇨Location контекстного меню, вызываемого щелчком правой кнопкой мыши. В зависимости от типа установленной точки прерывания диалоговое окно показывает ее местоположение либо как строку и символ в файле или функции, либо как адрес. Если местоположение представляет собой позицию в файле или функции, точку прерывания можно легко перенести в нужное место. Точку прерывания по адресу перенести сложнее, поскольку требуется гарантия того, что новый адрес является корректным местоположением точки прерывания.

Назначение метки точке прерывания

Одной из новых возможностей Visual Studio 2010 является назначение метки точке прерывания. Это особенно полезно, если вы хотите сгруппировать набор связанных точек прерывания. После присвоения метки точке прерывания вы можете выполнять поиск точки прерывания по имени или некоторые действия со всеми точками прерывания с определенной меткой.



Рис. 40.12

Для присвоения метки точке прерывания щелкните на последней правой кнопкой мыши и выберите в контекстном меню команду Edit Labels. При этом будет выведено диалоговое окно Edit Breakpoint Labels, показанное на рис. 40.12, в котором можно назначить точке прерывания одну или несколько меток.

После присвоения метки точкам прерывания можете выполнять над ними действия с помощью окна Breakpoints (Debug⇨Windows⇨Breakpoints). Это окно, показанное на рис. 40.13, позволяет фильтровать список, вводя метку в поле Search и нажимая <Enter>. Затем вы можете выбрать одно из действий на панели инструментов, такое как включение или отключение всех точек прерывания, отвечающих текущему критерию поиска.

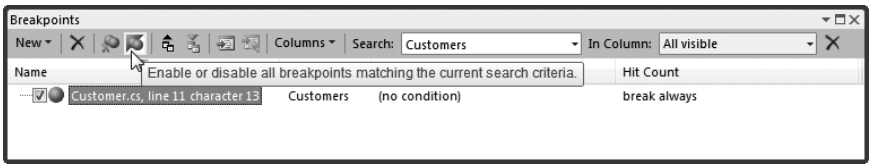


Рис. 40.13



По умолчанию поиск будет выполняться по всем столбцам, показанным в окне Breakpoint. Вы можете ограничить поиск определенными столбцами, заменив в раскрывающемся списке In Column пункт All Visible определенным столбцом.

Импорт и экспорт точек прерывания

Еще одной новой возможностью отладки в Visual Studio 2010 является импорт и экспорт точек прерывания. Эта возможность позволяет сохранять и восстанавливать точки прерывания, а также обмениваться ими с другими разработчиками. Эта функция ранее была недоступна, поскольку местоположения точек прерывания хранились в бинарном файле пользовательских настроек решения (`.suo`) вместе с другой пользовательской информацией.

Экспорт точек прерывания выполняется из окна Breakpoints (Debug⇨Windows⇨Breakpoints). Если вы хотите экспортировать только подмножество своих точек прерывания, сначала отфильтруйте список, введя критерий поиска. После того как будет выведен список точек прерывания, которые вы хотите экспортировать, щелкните на кнопке Export All Breakpoints Matching the Current Search Criteria панели инструментов.

Импорт точек прерывания также выполняется из окна Breakpoints путем щелчка на соответствующей кнопке панели инструментов.

Точки трассировки

Точка трассировки (tracepoint) отличается от точки прерывания тем, что при проходе через нее выполняется дополнительное действие. В действительности для таких целей, как применение фильтров, условий или счетчиков проходов, точка трассировки может рассматриваться как точка прерывания.

Точки трассировки можно сравнить с инструкциями наподобие `Debug` или `Trace`, но точки трассировки могут быть установлены динамически в процессе отладки кода и никак не влияют на последний.

Создание точки трассировки

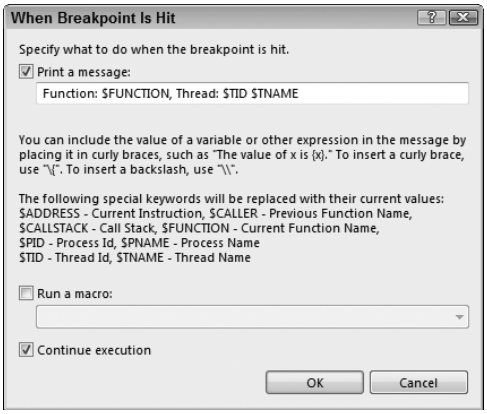


Рис. 40.14

Точки трассировки могут создаваться либо из существующих точек прерывания, либо из контекстного меню Breakpoint. Для создания точки трассировки из существующей точки прерывания выберите команду Breakpoint⇨When Hit из контекстного меню, вызываемого щелчком правой кнопкой мыши. Вы увидите диалоговое окно, показанное на рис. 40.14, которое позволяет вывести сообщение на консоль или запустить макрос. Кроме того, для создания точки трассировки в новом месте можно выбрать команду Breakpoint⇨Insert Tracepoint из контекстного меню, вызываемого щелчком правой кнопкой мыши. При этом вызывается то же диалоговое окно, изображенное на рис. 40.14, и вы можете настроить в нем действие точки трассировки.

После того как точка трассировки установлена, строка кода в окне кода изменяется, подтверждая этот факт. Это проиллюстрировано на рис. 40.15, на котором точка трассировки помечена на поле ромбом (хотя это и не видно на рисунке, ромб окрашен в красный цвет).

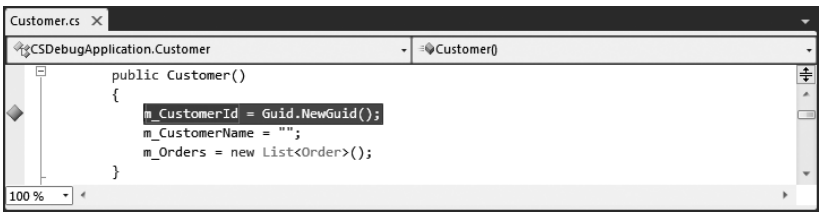


Рис. 40.15

Действия точек трассировки

При достижении точки трассировки могут выполняться два типа действий: вывод сообщения в консольное окно и запуск макроса. В диалоговом окне, показанном на рис. 40.14, можно указать, какое действие следует выполнить по достижении точки трассировки. Если не выбрано ни одно из действий, точка трассировки превращается в обычную точку прерывания.

По умолчанию флажок Continue execution установлен, так что приложение в точке трассировки не приостанавливается. Если этот флажок снят, то приложение по достижении данной точки трассировки приостанавливается, как если бы это была точка прерывания. Определенное для точки трассировки действие выполняется до прерывания приложения. Внешний вид такой точки трассировки не отличается от внешнего вида точки прерывания, так как визуальная подсказка указывает, остановит ли отладчик выполнение приложения в точке трассировки, а не связано ли с данной точкой трассировки некоторое действие.

Вывод сообщения

Как показано на рис. 40.14, в вашем сообщении трассировки может использоваться ряд ключевых слов. Однако здесь пропущена пара ключевых слов: `$FILEPOS`, которое указывает местоположение в текущем файле, и `$TICKS`, которое может использоваться в качестве индикатора относительного времени.

Макрос

Точки трассировки могут выполнять любые макросы Visual Studio, включая макросы, созданные вами. Поскольку макросы могут использоваться для модификации исходного текста, будьте осторожны при выполнении макросов точками трассировки. Модификация кода в процессе отладки приложения может привести к тому, что исходный текст не будет соответствовать работающему приложению. Макросы Visual Studio рассматриваются в главе 52.

Управление выполнением

По достижении точки прерывания зачастую полезно иметь возможность пройти код шаг за шагом и просмотреть как его выполнение, так и значения переменных. Visual Studio 2010 позволяет не только шаг за шагом пройти код, но и осуществить откат точки выполнения или даже повтор операций. Строка кода, которая должна быть выполнена, подсвечивается, а слева от нее выводится стрелка, как показано на рис. 40.16.

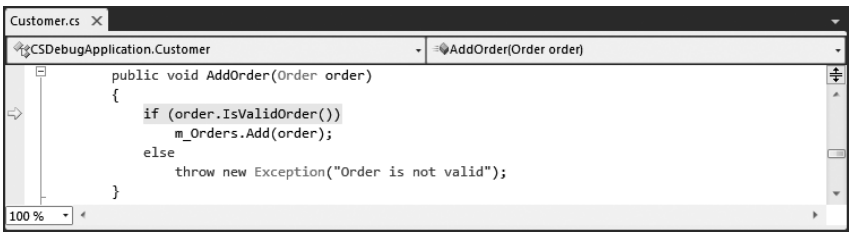


Рис. 40.16

Пошаговое выполнение кода

Первая возможность работы с точкой выполнения заключается в простом пошаговом проходе кода в ожидаемом порядке выполнения. При продвижении отладчика по коду возможны шаги трех размеров. Важно помнить, что при пошаговом проходе код реально выполняется, так что при этом могут изменяться значения переменных.

Команда Stepping Over (F10)

При выборе команды Stepping Over происходит полное выполнение строки и переход к следующей строке в текущем блоке кода. По достижении конца блока кода происходит возврат в вызывающий блок кода.

Команда Stepping Into (F11)

Команда Stepping Into ведет себя так же, как и Stepping Over, когда строка представляет собой простой оператор, такой как числовая операция или преобразование типов.

Если строка более сложная, команда Stepping Into вынуждает отладчик пройти через весь пользовательский код. Например, в приведенном ниже фрагменте нажатие <F10> в методе TestMethod приводит к проходу только по строкам кода в TestMethod. Нажатие же <F11> при достижении вызова MethodA заставляет отладчик пройти код метода MethodA шаг за шагом перед возвращением в метод TestMethod.

C#

```
public void TestMethod()
{
    int x = 5 + 5;
    MethodA();
}

private void MethodA()
{
    Console.WriteLine("Выполнен метод MethodA");
}
```

Команда Stepping Out (Shift+F11)

Случайно попав внутрь длинного метода, хочется иметь возможность выйти из него, не проходя каждую строку и не устанавливая дополнительную точку прерывания в его конце. Выберите команду Stepping Out, чтобы переместить курсор из текущего метода в точку его вызова. Рассмотрим предыдущий фрагмент. Если вы вошли в MethodA, то, нажав <Shift+F11>, вы немедленно вернете курсор в конец метода TestMethod.

Команда Step Filtering

Очень полезной возможностью, введенной в Visual Studio 2008 (Service Pack 1), является автоматический проход свойств и операторов. Во многих случаях открытые свойства представляют собой просто "обертки" закрытых переменных-членов, так что пошаговое их прохождение при отладке ничего не дает. Данная настройка отладчика особенно полезна, если вы вызываете метод, который получает несколько свойств в качестве параметров, как, например, метод в следующем фрагменте:

C#

```
printShippingLabel(cust.name, shipTo.street, shipTo.city,
shipTo.state, shipTo.zipCode);
```

Если включена настройка Step Over Properties and Operators, то отладчик после нажатия <F11> сразу переносит вас в первую строку метода printShippingLabel. При необходимости можно вручную открыть определенное свойство, щелкнув правой кнопкой мыши в окне редактора кода и выбрав в контекстном меню команду Step Into Specific. При этом перед вами появится подменю, в котором перечисляются все доступные свойства, как показано на рис. 40.17.

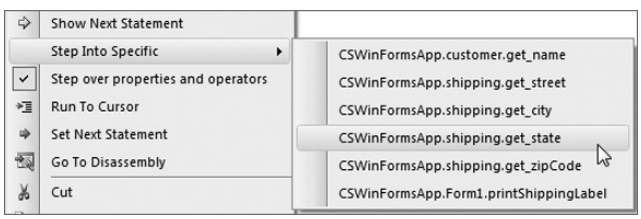


Рис. 40.17

Настройка Step Over Properties and Operators включена по умолчанию. Вы можете включать и отключать ее в процессе отладки, щелкнув правой кнопкой мыши в любом месте окна кода и выбрав эту команду в контекстном меню либо в диалоговом окне Options (Tools⇨Options с последующим выбором команды Debugging в левой части дерева).

Перемещение точки выполнения

По мере знакомства с возможностями отладки и пошагового выполнения кода вы можете обнаружить, что случайно "проскочили" интересующий вас метод. В этом случае вам, конечно, хотелось бы вернуться и пересмотреть свое последнее действие. Хотя вернуть приложение в предыдущее состояние невозможно, вы можете переместить точку выполнения так, чтобы метод был пройден повторно.

Для перемещения точки выполнения выберите и перетащите желтую стрелку рядом с текущей выполняемой строкой (рис. 40.16) в нужное место вперед или назад в текущем методе. Используйте этот прием осторожно, так как в результате можно получить незапланированное поведение кода и значений переменных.

Режим Edit and Continue

Одна из самых полезных возможностей отладки Visual Studio 2010 — режим Edit and Continue. Его поддерживают как C#, так и Visual Basic, позволяя вносить изменения в приложение "на лету". Когда ваше приложение приостановлено, можете внести в его код изменения и продолжить выполнение приложения. Новый или модифицированный код будет автоматически добавлен к вашему приложению, оказывая на него непосредственное воздействие.

Ограничения

Наверняка после прочитанного выше вас интересует вопрос "Накладываются ли какие-нибудь ограничения на изменения, которые можно внести в код?" Да, накладываются; имеется небольшое количество существенных правок (rude edits), которые требуют остановки приложения и сборки его заново. Полный список таких правок можно найти в справочной системе Visual Studio 2010, в разделе, посвященном возможности Edit and Continue. Здесь мы укажем только важнейшие из них.

- Изменения в текущей, активной инструкции
- Изменения глобальных имен (такие, как добавление новых типов или методов) или сигнатур методов, событий или свойств
- Изменения атрибутов

Отмена внесения изменений

При внесении изменений в исходный текст во время приостановки приложения Visual Studio должен внести эти изменения в работающее приложение. В зависимости от сложности сделанных изменений на это может потребоваться существенное количество времени. Если вы хотите отменить данное действие, выберите команду Stop Applying Code Changes в меню Debug.

Резюме

Большинство разработчиков, использующих Visual Studio 2010, применяют точки прерывания для изучения работы своего приложения. Из этой главы вы узнали, как оптимизировать использование точек прерывания для сокращения времени работы с ними.

В следующей главе вы познакомитесь с подсказками по данным и узнаете, как создавать прокси-типы и визуализаторы для отладки. Это позволит вам настраивать процесс отладки и экономить затрачиваемое на нее время.

41

Подсказки по данным, прокси и визуализаторы

В ЭТОЙ ГЛАВЕ...

- Изучение содержимого переменных с помощью подсказок по данным
- Применение атрибутов к классам и переменным-членам для настройки поведения отладчика
- Создание прокси-типов и визуализаторов для удобного представления сложных переменных и типов данных

Помимо написания кода, отладка, пожалуй, отнимает больше всего времени разработчика приложения. Если вы учтете все время, которое затрачено на пошаговое прохождение кода, просмотр значений переменных в окнах Watch и тому подобное, то без труда убедитесь в этом сами.

В предыдущих главах нас интересовало, как использовать различные окна отладчика для получения информации о текущем состоянии приложения и как устанавливать точки прерывания и точки трассировки для генерации отладочной информации. Из этой главы вы узнаете, как настроить процесс отладки, чтобы сократить время, затрачиваемое на проход ненужных строк кода.

Используя отладочные прокси-типы и визуализаторы, вы можете представлять сложные переменные и типы данных в отладчике так, как вам удобно. Эта возможность позволяет отфильтровывать ненужную информацию и сосредоточиваться на наиболее важных свойствах объекта, тем самым упрощая определение момента, когда приложение начинает функционировать некорректно, и выяснение причин этого.

Подсказки по данным



Рис. 41.1

В Visual Studio есть множество способов получить информацию о значениях переменных во время отладки. Для многих типов простейший способ исследовать переменную — просто навести на нее указатель мыши и прочесть значение переменной в появившемся окне подсказки по данным (*DataTip*). На рис. 41.1 показана подсказка для строкового свойства.

В дополнение к просмотру значений переменных вы можете щелкнуть на подсказке правой кнопкой мыши и выполнить ряд действий, в том числе копирование выведенного значения, добавление его в окно Watch и даже редактирование текущего значения переменной в случае простых типов данных наподобие строк или целых чисел.

Одной из новый функциональных возможностей системы Visual Studio 2010 являются закрепленные и плавающие подсказки. Их можно рассматривать как электронные эквиваленты стикеров (самоклеящихся листочков бумаги) для системы Visual Studio. Чтобы создать закрепленную подсказку, щелкните на пиктограмме кнопки в правой части окна подсказки. После этого оно будет прикреплено к данной строке кода в исходном файле в редакторе кода и будет видимым все время сеанса отладки.

На рис. 41.2 показана рабочая область системы Visual Studio с двумя закрепленными подсказками для переменных `c.CustomerName` и `o1.Total`. Меню появляется, когда вы располагаете указатель мыши над закрепленной подсказкой. Щелчок на пиктограмме с двойной стрелкой приводит к выводу текстового поля, в которое можно ввести текст. Вы можете также щелкнуть на пиктограмме кнопки в меню, чтобы преобразовать закрепленную подсказку в плавающую. Подсказка для переменной `c` на рис. 41.2 — плавающая.

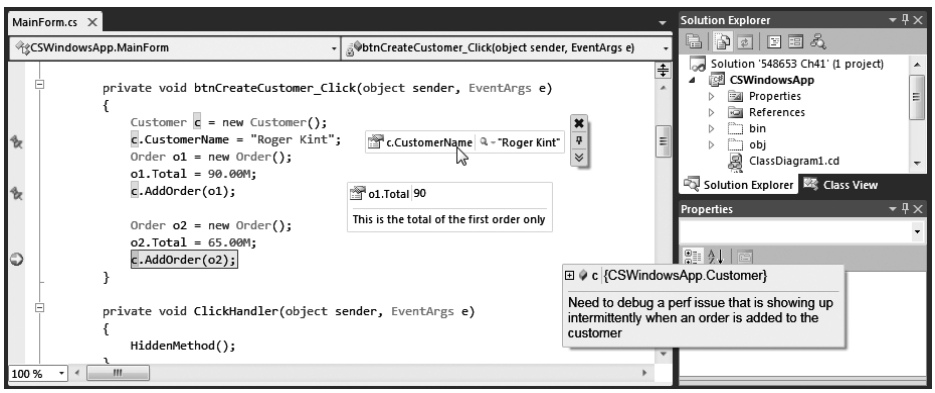


Рис. 41.2

Закрепленную подсказку можно перетащить в любую строку кода исходного файла, к которому она прикреплена, но не везде вне окна редактора кода. Закрепленные подсказки исчезают с экрана при переключении на другой исходный файл. Плавающие же подсказки в процессе сеанса отладки всегда видимы и могут быть перенесены в любое место на вашем мониторе.

Для каждой закрепленной подсказки на поле редактора кода имеется пиктограмма в виде синей кнопки. Эта пиктограмма остается видимой и по окончании сеанса отладки; если навести на нее указатель мыши, то появится подсказка со значением переменной, которое было во время последнего сеанса отладки.

Отдельные закрепленные или плавающие подсказки можно закрыть, щелкнув на пиктограмме с крестиком. Закрыть их все можно, выбрав команду меню Debug⇨ Clear All DataTips.

Наконец подсказки можно импортировать и экспортировать во внешний XML-файл, что удобно для резервного копирования или совместного использования подсказок несколькими разработчиками. Это делается с помощью команд Import DataTips и Export DataTips меню Debug.

Атрибуты отладки

В этом разделе вкратце рассказывается об атрибутах отладки, которые могут быть применены к коду для влияния на способ прохода отладчика. Некоторые из этих атрибутов могут использоваться и для настройки внешнего вида ваших типов данных при проходе над ними указателя мыши в режиме прерывания.



Классы атрибутов отладки находятся в пространстве имен `System.Diagnostics`. Вместо указания полного пространства имен для каждого атрибута в примерах кода в данной главе предполагается, что оно добавлено путем импорта.

Атрибут DebuggerBrowsable

Первый атрибут, который можно применить к полям и свойствам, — `DebuggerBrowsable`. В платформе .NET Framework 2.0 он интерпретировался только отладчиком C# и никак не проявлял себя в коде Visual Basic. Это ограничение в более новых версиях платформы .NET Framework отсутствует. Атрибут `DebuggerBrowsable` получает единственный параметр, который определяет, каким образом член будет выводиться в дереве переменных. В приведенном далее фрагменте кода поле `Orders` определено как `Collapsed`:

C#

```
public class Customer
{
    [DebuggerBrowsable(DebuggerBrowsableState.Collapsed)]
    public List < Order > Orders;
}
```

VB

```
Public Class Customer
    <DebuggerBrowsable(DebuggerBrowsableState.Collapsed)> _
        Public Orders As List(Of Order)
End Class
```

На рис. 41.3 (слева) показан этот фрагмент кода с атрибутом `DebuggerBrowsable`, установленным равным `Collapsed` (или не указанным). На рис. 41.3 (в центре) показан тот же фрагмент кода с атрибутом `DebuggerBrowsable`, установленным равным

RootHidden; при этом сам элемент Orders не виден, выводится только содержимое коллекции. Наконец, на рис. 41.3 (справа) атрибут DebuggerBrowsable установлен равным Never; в этом случае член Orders не виден вообще.

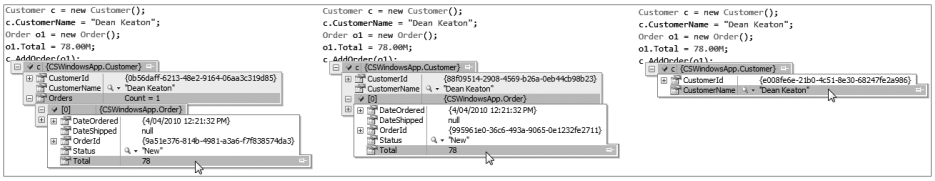


Рис. 41.3

Атрибут DebuggerDisplay

Когда указатель мыши оказывается над переменной в режиме прерывания, первое, что вы видите в подсказке, — тип объекта, расположенного под указателем. На рис. 41.3 указатель мыши изначально находился над классом Customer, за которым следует класс Order. Эта информация не особенно полезна, поскольку обычно вы и так знаете, с объектами каких типов работаете. Было бы лучше, если бы эта строка содержала более полезную информацию об объекте (как в случае более простых типов наподобие строк или целых чисел, когда выводится значение переменной).

Атрибут DebuggerDisplay может использоваться для изменения однострочного представления объекта (которое по умолчанию представляет собой полное имя класса). Этот атрибут получает единственный параметр, являющийся строкой. Данная строка может использовать вставку членов с использованием синтаксиса String.Format. Например, атрибуты, применяемые к классам Customer и Order, могут быть следующими:

C#

```
[DebuggerDisplay("Customer {CustomerName} "
                 "has {Orders.Count} orders")]
public class Customer
[DebuggerDisplay("Order made on {DateOrdered} "
                 "which is worth ${Total}")]
public class Order
```

VB

```
<DebuggerDisplay( _
    "Customer {CustomerName} has {Orders.Count} orders") > _
Public Class Customer
<DebuggerDisplay( _
    "Order made on {DateOrdered} which is worth ${Total}") > _
Public Class Order
```

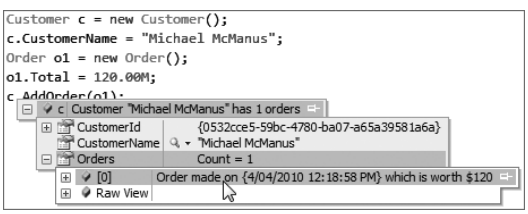


Рис. 41.4

При этом вывод отладчика имеет вид, показанный на рис. 41.4, указывающий, что у клиента Michael McManus есть один заказ, сделанный 4 апреля на сумму 120 долл.

Вернемся к синтаксису атрибута DebuggerDisplay. Вы видите, что выходная строка состоит из статического

текста и поля, а также информации, получаемой от свойства объекта. Например, свойство CustomerName объекта Customer в статический текст вносится как {CustomerName}.

Атрибут DebuggerHidden

Атрибут DebuggerHidden может быть добавлен к коду, который при отладке не должен проходиться шаг за шагом. Код, маркированный этим атрибутом, проходится как единое целое; в нем будет невозможно установить точку прерывания. Если этот код осуществляет вызов другого метода, отладчик проходит его шаг за шагом. В приведенном ниже фрагменте точка прерывания может быть установлена как в методе ClickHandler, так и в методе NotSoHiddenMethod:

C#

```
private void ClickHandler(object sender, EventArgs e)
{
  HiddenMethod();
}

[DebuggerHidden()]
public void HiddenMethod()
{
  Console.WriteLine("Здесь нельзя установить точку прерывания");
  NotSoHiddenMethod();
}

public void NotSoHiddenMethod()
{
  Console.WriteLine("Здесь точку прерывания установить можно!");
}
```

VB

```
Private Sub ClickHandler(ByVal sender As Object, _
                         ByVal e As EventArgs)
  HiddenMethod()
End Sub

<DebuggerHidden()> _
Public Sub HiddenMethod()
  Console.WriteLine("Здесь нельзя установить точку прерывания")
  NotSoHiddenMethod()
End Sub

Public Sub NotSoHiddenMethod()
  Console.WriteLine("Здесь точку прерывания установить можно!")
End Sub
```

При пошаговом проходе этого кода отладчик из вызова HiddenMethod в методе ClickHandler попадает прямо в метод NotSoHiddenMethod. Стек вызовов в этот момент имеет вид, показанный на рис. 41.5, и, как вы можете видеть, метод HiddenMethod в стеке не показан.

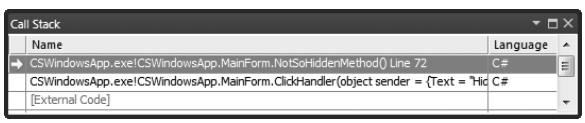


Рис. 41.5

Как и в случае других атрибутов `System.Diagnostic`, при генерации исключения атрибут будет проигнорирован и метод будет показан в стеке вызовов.

Атрибут DebuggerStepThrough

Подобно атрибуту `DebuggerHidden`, при применении к части кода атрибута `DebuggerStepThrough` проход этого кода выполняется как единое целое. Но в данном случае это делается независимо от того, вызывает ли данный код другие методы.



Рис. 41.6

Аналогично атрибуту `DebuggerHidden`, точки прерывания в блоке, маркированном атрибутом `DebuggerStepThrough`, не устанавливаются. Однако если точка прерывания установлена в коде, который вызывается маркированным кодом, то в стеке вызовов этот маркированный код будет указан как *внешний* (external code). Это проиллюстрировано на рис. 41.6, где показан код из предыдущего раздела. Однако в этом случае атрибут `DebuggerStepThrough` установлен для метода `HiddenMethod`, а не `DebuggerHidden`.

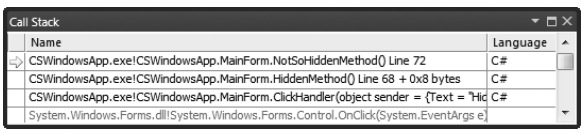


Рис. 41.7

Система Visual Studio 2010 поддерживает команду Just My Code, настраиваемую в узле Debugging в диалоговом окне Options (команда меню Tools⇨Options). Отключение этой команды показывает в стеке вызовов весь код, содержащийся в вашем приложении, как показано на рис. 41.7. Сюда включен весь код, сгенерированный проектировщиком и иными инструментами, который вы можете не хотеть отлаживать. Когда команда отключена, можете устанавливать точки прерывания и в блоках кода, маркированных рассматриваемым атрибутом.



Вы можете щелкнуть на стеке вызовов правой кнопкой мыши и выбрать пункт Show External Code для открытия любого скрытого или сгенерированного кода.

Атрибут DebuggerNonUserCode

Атрибут `DebuggerNonUserCode` объединяет атрибуты `DebuggerHidden` и `DebuggerStepThrough`. В конфигурации системы Visual Studio по умолчанию код, маркированный этим атрибутом, в стеке вызовов представлен как внешний. Как и в случае атрибута `DebuggerStepThrough`, вы не можете устанавливать точки прерывания в блоках кода, маркированных этим атрибутом. Пошаговый проход вызываемого кода выполняется так же, как и в случае применения атрибута `DebuggerHidden`.

Атрибут DebuggerStepperBoundary

Атрибут `DebuggerStepperBoundary` наиболее запутанный из всех атрибутов `Debugger`, поскольку действует только при определенных условиях. Он используется

для избежания ошибочной отладки, которая может произойти при переключении контекста потока в пределах атрибута `DebuggerNonUserCode`. При этом вполне возможно, что очередной пользовательский код, который выполняется шаг за шагом, на самом деле не связан с отлаживаемым процессом. Чтобы избежать такого неверного поведения отладчика, атрибут `DebuggerStepperBoundary` в таком сценарии не позволит пошаговое прохождение кода и восстановит его нормальное выполнение.

Прокси-типы

К этому времени вы узнали, как можно модифицировать подсказки для того, чтобы они показывали более существенную для отладки приложения информацию. Однако рассматривавшиеся до сих пор атрибуты ограничены в своем управлении тем, какая информация будет представлена в раскрытом дереве. Атрибут `DebuggerBrowsable` позволяет скрыть некоторые члены, но способа добавить дополнительные члены с помощью этого атрибута нет. Для этого можно использовать атрибут `DebuggerTypeProxy`, который предоставляет полный контроль над форматом подсказки.

Другой сценарий, при котором полезен прокси-тип, — когда свойство класса изменяет значение в классе. Например, приведенный ниже фрагмент класса `Customer` отслеживает количество вызовов свойства `OrderCount`. При обращении к подсказке свойство `CountAccessed` увеличивается на единицу.

C#

```
public class Customer
{
    private int m_CountAccessed;
    public int OrderCount
    {
        get
        {
            m_CountAccessed++;
            return this.Orders.Count;
        }
    }
    public int CountAccessed
    {
        get
        {
            return this.m_CountAccessed;
        }
    }
}
```

На рис. 41.8 показана всплывающая подсказка, которая должна показывать содержимое класса `Customer`. Вместо полного списка заказов в ней выводится общее их количество, максимальная и минимальная сумма заказа и список заказанного.

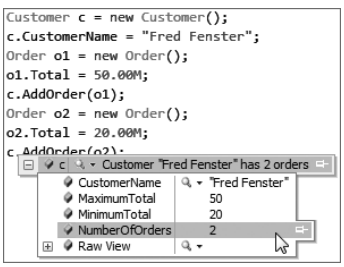


Рис. 41.8

Первая строка подсказки такая же, как и создаваемая с помощью атрибута `DebuggerDisplay`. Для генерации остальной части подсказки необходимо создать дополнительный класс, который будет работать в качестве заместителя при представлении этой информации. Затем класс `Customer` нужно маркировать атрибу-

том DebuggerTypeProxy, чтобы отладчик знал, что при выводе подсказки вместо класса Customer следует использовать прокси. Приведенный далее фрагмент показывает класс CustomerProxy, вложенный в класс Customer:

C#

```
[DebuggerDisplay("Customer {CustomerName} "
                 "has {Orders.Count} orders")]
[DebuggerTypeProxy(typeof(Customer.CustomerProxy))]
public class Customer
{
    private int m_CountAccessed;
    public int OrderCount
    {
        get
        {
            m_CountAccessed++;
            return this.Orders.Count;
        }
    }
    public int CountAccessed
    {
        get
        {
            return this.m_CountAccessed;
        }
    }
    public class CustomerProxy
    {
        public string CustomerName;
        public int NumberOfOrders;
        public decimal MaximumTotal = decimal.MinValue;
        public decimal MinimumTotal = decimal.MaxValue;
        public CustomerProxy(Customer c)
        {
            this.CustomerName = c.m_CustomerName;
            this.NumberOfOrders = c.m_Orders.Count;
            foreach (Order o in c.m_Orders)
            {
                this.MaximumTotal =
                    Math.Max(o.Total, this.MaximumTotal);
                this.MinimumTotal =
                    Math.Min(o.Total, this.MinimumTotal);
            }
        }
    }
}
```

Имеется очень немного причин для создания вложенных открытых классов, и одна из них — создание прокси-типа, поскольку он должен быть открытым, чтобы его можно было указать в атрибуте DebuggerTypeProxy, и вложенным, чтобы иметь возможность обращаться к закрытым членам класса Customer без применения открытых методов доступа.

Узел Raw View

Иногда может потребоваться игнорировать прокси, например при использовании компонента стороннего производителя, имеющего прокси, который искажает структуру основного типа. Если в процессе работы с таким типом возникают проблемы, для поиска их источника, возможно, придется просмотреть его содержимое.

В нижней части всплывающей подсказки на рис. 41.8 виден узел, помеченный как Raw View. После его раскрытия выводится отладочная подсказка в том виде, который она имела бы при отсутствии прокси.

Кроме того, в Visual Studio вы можете отключить все прокси-типы, воспользовавшись меню Tools⇨Options. В узле Debugging включите переключатель Show Raw Structure of Objects in Variables Windows. Это предотвратит вывод отладчиком информации от прокси.

Визуализаторы

Эта часть главы посвящена функциональной возможности Visual Studio 2010, которая может использоваться для помощи в отладке более сложных структур данных. Два наиболее распространенных типа данных, используемых программистами, — строки и `DataTable`. Строки часто оказываются длиннее, чем может показать всплывающая подсказка, а структура объекта `DataTable` не подходит для вывода в подсказке даже с применением прокси. В обоих случаях для просмотра данных в приемлемом виде можно создать визуализатор.

Когда визуализатор для некоторого типа создан, в первой строке всплывающей подсказки отладчика появляется пиктограмма в виде лупы. После щелчка на ней отображается визуализатор. На рис. 41.9 показано диалоговое окно Text Visualizer.

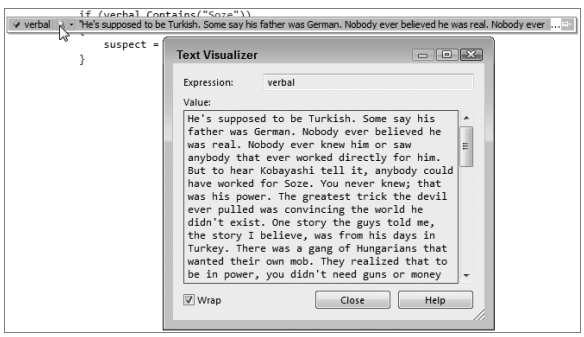


Рис. 41.9

Перед тем как приступить к написанию визуализатора, следует добавить ссылку на пространство имен `Microsoft.VisualStudio.DebuggerVisualizers`. Для этого щелкните правой кнопкой мыши на проекте в Solution Explorer и выберите из контекстного меню команду Add Reference. Следует также добавить это пространство имен как импорт во все классы, для которых планируется создание визуализаторов отладчика.



Версия `Microsoft.VisualStudio.DebuggerVisualizers`, которая поставляется с Visual Studio 2010, работает только с проектами с целевой версией Microsoft .NET Framework 4.0.

Обычно визуализатор состоит из двух частей: класса, который выступает в роли принимающего для визуализатора и на который ссылается атрибут `DebuggerVisualizer`, применяемый к визуализируемому классу, и формы, которая затем используется для вывода, или визуализации, класса. На рис. показана простая форма `CustomerForm`, которая может использоваться для представления информации о клиенте. Это просто обычная форма с управляющими элементами TextBox, DataGridView и Button. Единственным отличием является то, что она маркирована как `Serializable`, и ее конструктор принимает объект типа `Customer`, из которого извлекается выводимая информация, как показано в приведенном далее фрагменте кода.

C#

```
[Serializable()]
public partial class CustomerForm : Form
{
    public CustomerForm(Customer c)
    {
        InitializeComponent();
        this.txtCustomerId.Text   = c.CustomerId.ToString();
        this.txtCustomerName.Text = c.CustomerName;
        this.dgOrders.DataSource  = c.Orders;
    }

    private void btnOk_Click(object sender, EventArgs e)
    {
        this.DialogResult = DialogResult.OK;
        this.Close();
    }
}
```

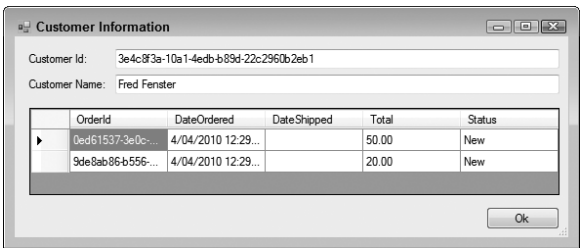


Рис. 41.10

Следующий шаг состоит в назначении этой формы визуализатором класса `Customer`. Это делается путем создания вложенного класса `CustomerVisualizer`, наследующего абстрактный класс `DialogDebuggerVisualizer`, как показано в следующем фрагменте кода:

C#

```
[Serializable()]
[DebuggerDisplay("Customer {CustomerName} "
                 "has {Orders.Count} orders")]
[DebuggerTypeProxy(typeof(Customer.CustomerProxy))]
[DebuggerVisualizer(typeof(Customer.CustomerVisualizer))]
public class Customer
{
  //...
  public class CustomerVisualizer : DialogDebuggerVisualizer
  {
    protected override void Show(
              IDialogVisualizerService windowService,
              IVisualizerObjectProvider objectProvider)
    {
      Customer c = (Customer)objectProvider.GetObject();
      CustomerForm cf = new CustomerForm(c);
      windowService.ShowDialog(cf);
    }
  }
}
```

В отличие от прокси-типа, который взаимодействует с реальным отлаживаемым объектом `Customer`, визуализаторы должны быть способны сериализовать отлажи-

ваемый класс, чтобы переместить его из отлаживаемого процесса в процесс отладки, а затем показать в визуализаторе. А потому и класс Customer, и класс Order должны быть маркированы атрибутом Serializable.

Метод Show класса CustomerVisualizer решает три задачи. Для вывода отлаживаемого объекта Customer сначала требуется получить ссылку на этот объект. Это делается с помощью метода GetObject объекта ObjectProvider. Поскольку связь между двумя процессами осуществляется посредством потока, этот метод выполняет трудную работу по десериализации объекта, чтобы с ним можно было работать.

Затем нужно передать объект Customer новому экземпляру CustomerForm. Наконец, следует использовать метод ShowDialog объекта WindowService для вывода формы. Важно вывести форму именно с помощью данного объекта, поскольку это гарантирует, что форма будет выведена верным потоком выполнения пользовательского интерфейса.

Наконец обратите внимание на то, что класс CustomerVisualizer указывается в атрибуте DebuggerVisualizer, заставляя отладчик использовать этот класс для загрузки визуализатора объекта Customer.

В качестве примечания, если вы создаете компоненты и хотите поставлять их визуализаторы отдельно от самих компонентов, эти визуализаторы могут быть инсталлированы путем помещения соответствующей сборки либо в каталог C:\Program Files\Microsoft Visual Studio 10.0\Common7\Packages\Debugger\Visualizers, (Program Files (x86) на 64-битных версиях Windows), либо в каталог Documents\Visual Studio 2010\Visualizers.

Усовершенствованные методы

До сих пор в этой главе рассматривалось, как выводить и визуализировать отлаживаемые объекты. Из предыдущих глав вы узнали, как модифицировать значения полей и свойств отлаживаемых объектов с помощью подсказок по данным. Осталось выяснить, каким образом можно редактировать более сложные объекты данных. В последнем разделе главы речь пойдет о том, как расширить свой визуализатор так, чтобы он мог сохранять изменения в объекте Customer.

Сохранение изменений в объекте

Когда создавался объект класса CustomerVisualizer, объект класса Customer извлекался из коммуникационного потока с использованием метода GetObject. Так вы получали клон отлаживаемого объекта класса Customer для применения в визуализаторе. Чтобы сохранить любые изменения, которые могут быть сделаны в объекте класса CustomerVisualizer, требуется отправить новый объект класса Customer обратно отлаживаемому процессу. Это можно сделать с помощью метода ReplaceObject объекта ObjectProvider, предоставляемого классом CustomerVisualizer.

Перед тем как вызвать метод ReplaceObject, требуется внести некоторые изменения, чтобы передавать модифицированный объект класса Customer назад визуализатору. При первой передаче в класс нужно сохранить объект класса Customer во внутренней переменной и обеспечить работу с ней через свойство, предназначенное только для чтения. Все это показано в приведенном далее фрагменте кода:

C#

```
[Serializable()]
public partial class CustomerForm : Form
{
    public CustomerForm(Customer c)
    {
        InitializeComponent();
        this.txtCustomerId.Text = c.CustomerId.ToString();
        this.txtCustomerName.Text = c.CustomerName;
        this.dgOrders.DataSource = c.Orders;
        m_ModifiedCustomer = c;
    }

    private Customer m_ModifiedCustomer;
    public Customer ModifiedCustomer
    {
        get
        {
            m_ModifiedCustomer.CustomerId   =
                    new Guid(txtCustomerId.Text);
            m_ModifiedCustomer.CustomerName =
                    txtCustomerName.Text;
            m_ModifiedCustomer.Orders       =
                    (List<Order>)dgOrders.DataSource;
            return m_ModifiedCustomer;
        }
    }

    private void btnOk_Click(object sender, EventArgs e)
    {
        this.DialogResult = DialogResult.OK;
        this.Close();
    }
}
```

Теперь можно легко обратиться к модифицированному объекту `Customer` и сохранить изменения, вызвав метод `ReplaceObject`, как показано далее:

C#

```
[Serializable()]
[DebuggerDisplay("Customer {CustomerName} "
                 "has {Orders.Count} orders")]
[DebuggerTypeProxy(GetType(Customer.CustomerProxy))]
[DebuggerVisualizer(GetType(Customer.CustomerVisualizer))]
public class Customer
{
  ...

  public class CustomerVisualizer : DialogDebuggerVisualizer
  {
    protected override void Show(
                 IDialogVisualizerService windowService,
                 IVisualizerObjectProvider objectProvider)
    {
      Customer c = (Customer)objectProvider.GetObject();
      CustomerForm cf = new CustomerForm(c);
      if (windowService.ShowDialog(cf) ==
          System.Windows.Forms.DialogResult.OK)
          objectProvider.ReplaceObject(cf.ModifiedCustomer);
    }
  }
}
```



Альтернативный метод заключается в применении связывания данных для всех полей `Customer` в форме с объектом `BindingSource`. Этот объект `BindingSource` может быть доступен с помощью открытого модификатора, что делает его доступным из класса визуализатора. Все, что теперь необходимо, — это установить объект `Customer` как `DataSource` объекта `BindingSource` классом визуализатора, и все изменения будут автоматически синхронизированы с исходным объектом `Customer`.

Резюме

Отладка приложений представляет собой один из наиболее пожирающих время и нервы разработчика видов деятельности в цикле создания приложения. Из этой главы вы узнали о том, как можно несколько упростить отладку в системе Visual Studio 2010, настраивая способы выведения информации.

Применение прокси-типов и визуализаторов позволяет управлять способом представления интересующей вас информации в процессе отладки, а значит, вам будет легче найти место и причину некорректной работы приложения и исправить допущенные огрехи.

42

Отладка веб-приложений

В ЭТОЙ ГЛАВЕ...

- Использование Visual Studio для отладки серверного кода ASP.NET и клиентского кода JavaScript в веб-браузере
- Включение и просмотр журнальных файлов трассировки ASP.NET для отдельных веб-страниц или целых приложений
- Настройка Health Monitoring для получения уведомлений о проблемах в приложении ASP.NET

В системе Visual Studio 2010 отладка решений для веб стала такой же обычной задачей, как и отладка Windows-приложений. Вы можете использовать большинство тех же окон отладки, которые рассматривались в предыдущих главах, а также исправлять ошибки с помощью инструмента Exception Assistant. Однако имеются некоторые отличия и дополнительные функциональные возможности, специфичные для веб-приложений, которые можно использовать для отладки, более тесно связанной с веб-парадигмой.

В дополнение к стандартным методам отладки ASP.NET предоставляет комплексные возможности трассировки, а также возможности мониторинга системы, позволяющие убедиться, что все работает так, как и ожидалось, либо быстро локализовать проблемы в противном случае.



При использовании Windows Vista или Windows 7, а с целью отладки — серверов IIS вместо встроенного сервера для веб-разработки, необходимо запускать систему Visual Studio с правами администратора. Щелкните правой кнопкой мыши на ярлыке Visual Studio 2010 и выберите Run as Administrator. Чтобы всегда выполнять запуск с правами администратора, щелкните правой кнопкой мыши на ярлыке Visual Studio 2010, выберите Properties и вкладку Compatibility и установите флажок Run This Program as an Administrator.

Отладка серверного кода ASP.NET

Перед выполнением любого уровня отладки веб-приложения необходимо убедиться, что в вашем веб-приложении или проекте веб-сайта включена отладка ASP.NET. В случае проекта веб-приложения включить отладку можно, щелкнув правой кнопкой мыши на записи проекта в окне Solution Explorer и выбрав узел Properties. Затем выберите вкладку Web и убедитесь, что в группе Debuggers установлен флажок ASP.NET, как показано на рис. 42.1.

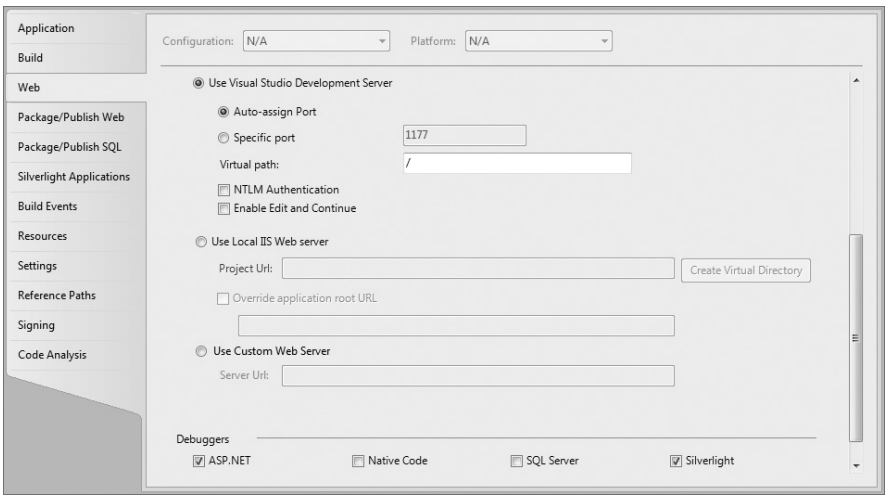


Рис. 42.1

Если вы хотите иметь возможность включать автономный код, хранимые процедуры или приложения для платформы Silvernight в свою отладку веб-приложений, можете на этой же вкладке активизировать переключатели Native Code, SQL Server и Silverlight. Опции Native code и SQL Server будут рассмотрены в следующей главе, а Silverlight — позже в данной главе.



В таких проектах веб-приложений, как ASP.NET Web Service или приложения ASP.NET MVC, отладка включается так же, как и в стандартных веб-приложениях ASP.NET. Фактически с точки зрения отладки между всеми этими типами проектов нет никакой разницы.

Поскольку проекты веб-сайтов не имеют файла проекта, для включения отладки следует использовать немного отличающуюся процедуру. Отладка включается в результате щелчка правой кнопкой мыши на записи проекта в окне Solution Explorer и выбора в контекстном меню команды Property Pages. Когда диалоговое окно Property Pages появится на экране, перейдите на страницу Start Options и убедитесь, что в группе Debuggers установлен флажок ASP.NET, как показано на рис. 42.2.

Как и в случае проектов веб-приложений, можете настроить запуск проекта веб-сайта, в том числе таким образом, чтобы не загружалась определенная страница, а сервер ожидал запроса от другого приложения.

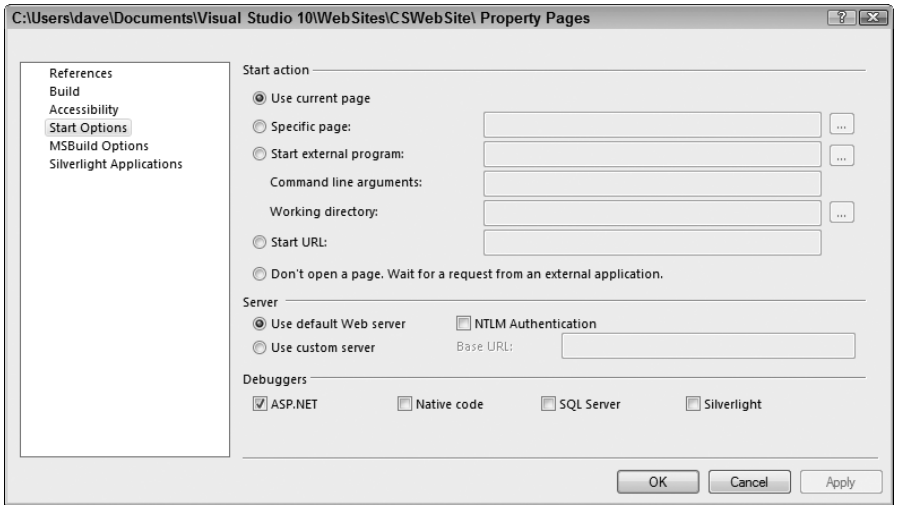


Рис. 42.2

Кроме отладчика ASP.NET на страницах свойств следует также включить опцию отладки для компилятора в файле `web.config`. Найдите узел `compilation` в элементе `system.web` и установите атрибут `debug` равным `true`. Ниже приведен минимальный файл `web.config` с включенной отладкой, готовый для подключения отладчика к приложению.

```
<?xml version="1.0"?>
<configuration>
    <system.web>
        <compilation debug="true" targetFramework="4.0" />
    </system.web>
    <system.webServer>
        <modules runAllManagedModulesForAllRequests="true" />
    </system.webServer>
</configuration>
```

Обратите внимание, что даже если активизирован отладчик ASP.NET на странице Start Options, без установки атрибута `debug` равным `true` вы не сможете отлаживать приложение. Однако система Visual Studio обнаружит это противоречие и представит вам диалоговое окно, информирующее о необходимости внести изменения в файл `web.config`. В нем также имеется возможность указать системе Visual Studio изменить указанный атрибут вместо вас.



Не распространяйте приложение ASP.NET с установленным атрибутом `debug="true"` в файле `web.config`. Это приведет к замедленной работе вашего приложения и использованию большего количества памяти и не позволит кэшировать некоторые объекты.

Исключения веб-приложений

По умолчанию ваше веб-приложение, столкнувшись с исключением, выводит страницу ошибки сервера ASP.NET, показанную на рис. 42.3. Именуемая в просторечии “желтым экраном смерти”, эта страница выводит детальную информацию об исключении, в том числе о состоянии стека.

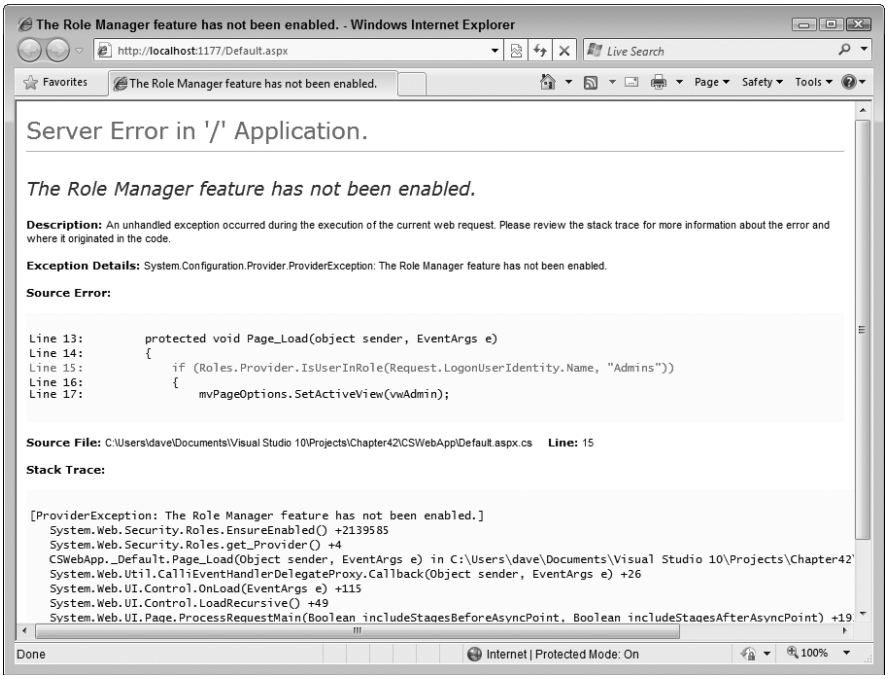


Рис. 42.3

Страница ошибки сервера генерируется как при отладочном, так и при обычном выполнении. Хотя эта страница весьма полезна при отладке, это совсем не то, что хочется показывать конечному пользователю. К счастью, имеется простой способ настройки перенаправления исключений, в том числе стандартных ошибок HTTP, путем редактирования раздела `customErrors` файла `web.config`.

Модифицировав уже рассмотренный нами файл `web.config` и включив в него перенаправление для ошибок 403 (доступ запрещен) и 404 (страница не найдена), мы получим файл наподобие приведенного:

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
    <customErrors mode="RemoteOnly"
                  defaultRedirect="GenericErrorPage.htm">
      <error statusCode="403" redirect="AccessDenied.html" />
      <error statusCode="404" redirect="PageNotFound.html" />
    </customErrors>
  </system.web>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true" />
  </system.webServer>
</configuration>
```

Атрибут `mode` раздела `customErrors` определяет три опции вывода пользовательской страницы об ошибке вместо страницы ошибки сервера по умолчанию.

- **On.** Всегда выводится пользовательская страница об ошибке.
- **Off.** Всегда выводится страница ошибки сервера по умолчанию.

➤ **RemoteOnly.** Выводится страница ошибки сервера по умолчанию, если запрос пришел от браузера с локального компьютера; в противном случае выводится пользовательская страница об ошибке.



Рис. 42.4

Страница ошибки сервера полезна в промышленном сценарии, когда вы не можете запустить приложение в режиме отладки. Однако при отладке гораздо более полезно приостановить выполнение приложения, как только было сгенерировано исключение. Добиться этого можно, включив опцию Break When an Exception Is Thrown для Common Language Runtime. На рис. 42.4 показано, как данная опция устанавливается в диалоговом окне Exceptions, вызываемом командой меню Debug⇨Exceptions.

После включения этой опции, когда возникает исключение, система Visual Studio возвращается в интегрированную среду и позиционирует рабочую область таким образом, чтобы соответствующая инструкция стала видимой. Подобно Windows-приложениям, система Visual Studio может помочь вам при возникновении ошибки, используя инструмент Exception Assistant. Как показано на рис. 42.5, с его помощью выводится детальная информация о веб-ошибке и о том, какая часть инструкции привела к ней.

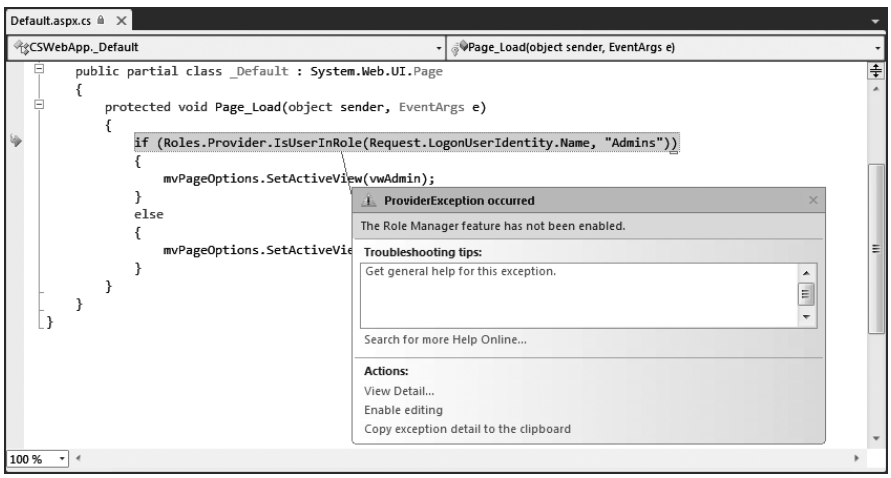


Рис. 42.5

Вы можете собрать дополнительную информацию об ошибке, щелкнув на ссылке View Detail.

Режим Edit and Continue

Функциональная возможность Edit and Continue, которая позволяет модифицировать код приостановленного в сеансе отладки приложения, по умолчанию для веб-приложений ASP.NET отключена. Ее можно включить, щелкнув правой кнопкой мы-

ши на записи проекта в окне Solution Explorer и выбрав узел Properties. На вкладке Web выберите опцию Enable Edit and Continue. Эта функциональная возможность работает только со встроенным сервером для веб-разработки Visual Studio.

Проекты веб-сайтов не поддерживают возможность Edit and Continue, однако в силу их естественной поддержки итеративного стиля разработки это не такая уж большая потеря. Более подробно функциональная возможность Edit and Continue рассматривается в главе 40.

Обработка ошибок

Хотя отладка приложений значительно упрощается при использовании предоставляемых Visual Studio 2010 инструментов, всегда лучше избегать ошибок, а не исправлять их. Вы можете использовать в веб-приложениях структурированную обработку исключений `try-catch`. Кроме того, можете включить в приложение код для обработки любых ошибок, выходящих за рамки всех условий `catch`.



Обратите внимание на использование термина *обработка ошибок*, а не *обработка исключений*. Это связано с тем, что обработка ошибок включает не только обработку исключений, но и, например, ошибки HTML, такие как "страница не найдена" или "требуется аутентификация".

Ошибки можно перехватывать на двух уровнях. На уровне отдельной страницы можно перехватить неожиданную ошибку и сгенерировать собственную пользовательскую встроенную ошибку; можно также перехватывать ошибки на уровне приложения путем реализации программы для перехвата ошибок в файле `global.asax`.

Ошибки на уровне страниц

Для обработки ошибки на отдельной странице необходимо реализовать программу обработки события `MyBase.Error`. При возникновении этого события можно выполнять любые требующиеся при неожиданной ошибке действия. Типичная подпрограмма может иметь следующий вид:

C#

```
void Page_Error(object sender, EventArgs e)
{
    Response.Write("Произошла неожиданная ошибка.");
    Server.ClearError();
}
```

VB

```
Private Sub Page_Error(ByVal sender As Object, _
                       ByVal e As System.EventArgs) _
    Handles MyBase.Error
    Response.Write("Произошла неожиданная ошибка.")
    Server.ClearError()
End Sub
```

Как говорилось ранее, вы можете также назначить перенаправления стандартных ошибок HTTP в файле `web.config`, так что такой метод следует использовать только для ошибок, которые еще не были обработаны и специфичны для данной конкретной страницы.

Ошибки на уровне приложения

На уровне веб-приложения ошибки можно перехватывать и с помощью файла `global.asax`. По умолчанию веб-проекты Visual Studio 2010 не включают этот файл, так что сначала нужно добавить его в проект с помощью диалогового окна Add New Item. Выберите элемент Global Application Class, оставьте неизменным имя `global.asax` и щелкните на кнопке Add, чтобы добавить файл в свой проект.

Когда этот класс добавляется в проект, шаблон включает "заглушки" для наиболее распространенных событий, в том числе для события ошибки. Для обработки любых ошибок, не перехваченных где-либо в ином месте проекта, добавьте свой код обработки в подпрограмму `Application_Error`:

C#

```
protected void Session_End(object sender, EventArgs e)
{
    Server.Transfer("UnexpectedError.aspx");
}
```

VB

```
Sub Application_Error(ByVal sender As Object, _
                      ByVal e As EventArgs)
    Server.Transfer("UnexpectedError.aspx")
End Sub
```

Приведенная подпрограмма просто направляет пользователя на страницу с сообщением об ошибке, определяемую информацией из свойства `Server.GetLastError`.

Отладка клиентского кода JavaScript

Одна из наиболее полезных возможностей Visual Studio 2010 для веб-разработчиков — поддержка отладки кода JavaScript на стороне клиента. В сочетании с поддержкой технологии IntelliSense для языка JavaScript эта возможность существенно упрощает разработку кода на языке JavaScript.



Отладка JavaScript работает только в том случае, когда в качестве веб-браузера в сеансе отладки используется Internet Explorer.

Установка точек прерывания в коде JavaScript

Установка точек прерывания в коде JavaScript не отличается от установки других точек прерывания. В окне редактора все точки прерывания в коде JavaScript выводятся в виде кружка с белым центром, как показано на рис. 42.6.

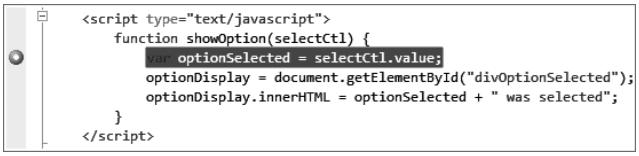


Рис. 42.6.



Отладка JavaScript будет отключена, если включена отладка Silverlight.



Точки прерывания JavaScript обладают той же функциональностью, что и стандартные точки прерывания. К ним относятся установка условий, счетчик проходов и даже макросы как часть точек трассировки.

Когда отладчик достигает точки прерывания, он приостанавливает выполнение и выводит HTML-код, который выведен для клиента (рис. 42.7). Это полноценная отладка кода, включающая все элементы на стороне клиента, такие как the ViewState, и серверные управляющие элементы, визуализированные в HTML.

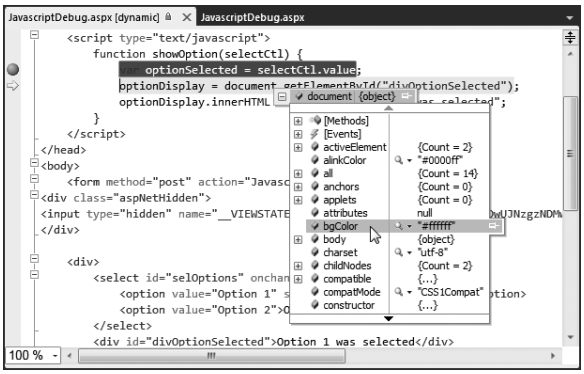


Рис. 42.7

Система Visual Studio 2010 обладает также сложными визуализаторами для элементов на стороне клиента. На рис. 42.7 это продемонстрировано всплывающей подсказкой, которая показывает свойства и методы объекта документа.

В одно и то же время на одной и той же странице можно установить точки прерывания как в клиентском коде JavaScript, так и в серверном коде Visual Basic или C#. Это позволяет вам шаг за шагом проходить как серверный, так и клиентский коды в одном сеансе отладки.

Отладка динамически генерируемого JavaScript

Существует несколько сценариев, в которых платформа ASP.NET отсылает клиенту динамически сгенерированный сервером код JavaScript. Например, управляющие элементы ASP.NET AJAX, такие как Update Panel, генерируют клиентские файлы JavaScript, которые хранятся в качестве ресурсов в ScriptManager.

Когда вы запускаете веб-приложение в режиме отладки, в окне Solution Explorer выводится список всех ссылок на сценарии, которые загружает отлаживаемая страница, как показано на рис. 42.8. Двойной щелчок на любой из ссылок в узле Script Documents позволяет вывести соответствующий код JavaScript и установить точки прерывания в этих сценариях.

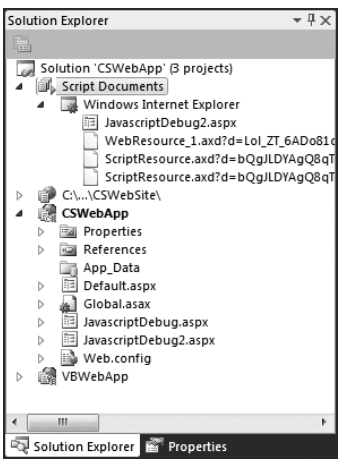


Рис. 42.8

Отладка кода на языке JavaScript с помощью библиотеки ASP.NET AJAX

Библиотека ASP.NET AJAX предоставляет как отладочную, так и окончательную, выпускную, версии клиентских библиотек на языке JavaScript. Выпускная версия оптимизирована в смысле производительности и минимизирует размер кода JavaScript, который должен загружаться клиентом. Отладочная версия более многословна и предоставляет дополнительные отладочные возможности во время работы, такие как проверка типов и аргументов.

Если в файле `web.config` включена отладка, библиотека ASP.NET AJAX использует отладочную версию клиентских библиотек. Вы можете обеспечить загрузку отладочной версии для отдельных страниц, установив атрибут `ScriptMode="Debug"` в свойствах элемента управления ScriptManager.

Кроме того, библиотека ASP.NET AJAX содержит класс `Sys.Debug`, который можно применить для добавления отладочных инструкций в свой клиентский код на языке JavaScript. Этот класс может использоваться для вывода свойств объектов во время выполнения кода, генерации сообщений трассировки или проверки утверждений (assertions).

Отладка приложений Silverlight



Рис. 42.9

Система Visual Studio 2010 содержит отладчик, который упрощает отладку приложений на платформе Silverlight. Создавая приложение Silverlight, система Visual Studio приглашает вас либо сгенерировать тестовую страницу HTML для размещения приложения Silverlight, либо использовать существующий или новый веб-проект, как показано на рис. 42.9.

На рис. 42.9 мы выбираем создание тестовой страницы для размещения приложения Silverlight в проекте веб-приложения, являющегося частью существующего решения, и включаем в этом веб-приложении отладку приложения Silverlight. Если вы выберете любой из двух других вариантов, вам не потребуется выполнять дополнительные шаги для включения отладки приложения Silverlight.

В существующем проекте веб-приложения можно всегда включить поддержку отладки приложения Silverlight или вывести информацию о ее состоянии на соответствующей странице свойств проекта.

Если отладка приложения Silverlight включена, можно устанавливать точки прерывания в коде в файлах классов страниц XAML. Когда отладчик встречается с такой точкой прерывания, сеанс приостанавливается и выводится текущая строка кода, как показано на рис. 42.10. Вы можете шаг за шагом проходить код, просматривать стек

вызовов и свойства объектов так же, как это делается для любого веб-приложения или приложения Windows Forms.

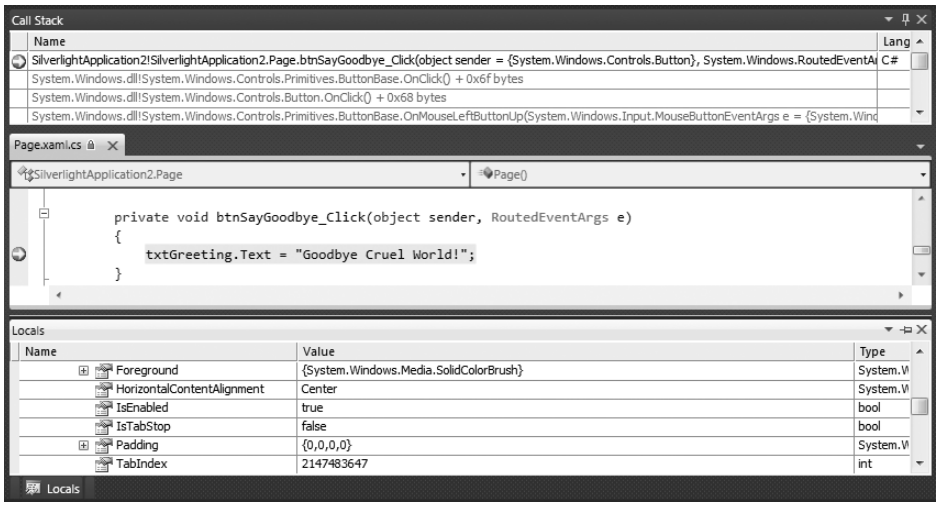


Рис. 42.10

Единственное существенное ограничение в случае отладки приложения Silverlight заключается в отсутствии поддержки режима Edit and Continue и отключенной отладке JavaScript.

Трассировка

Если ваше веб-приложение работает не так, как ожидалось, в дополнение к его отладке можно реализовать трассировку приложения ASP.NET для просмотра информации, генерируемой запросами к отдельным страницам. Трассировка позволяет добавлять в код отладочные инструкции, видимые только локально; когда веб-приложение установлено на удаленный сервер, пользователям эта информация трассировки будет недоступна.

Информация трассировки может включать переменные и простые объекты, помогающие вам определить состояние конкретного запроса и то, как он выполнялся. Обратите внимание, что трассировка приложения ASP.NET отличается от использования класса `Trace` в обычном приложении Windows тем, что вывод выполняется на реальную веб-страницу ASP.NET или для автономного просмотрщика трассировочной информации, а не в окно вывода, используемое командами `Trace`.

Трассировка на уровне страницы

Для реализации трассировки на уровне страницы необходимо включить атрибут `trace` в директиву `@Page` в начале страницы, которая должна трассироваться. Простая ASPX-страница с активизированной трассировкой может иметь следующий вид:

```
<%@ Page Language="C#" AutoEventWireup="true" Trace="true"
         TraceMode="SortByCategory"
         CodeBehind="ShowTrace.aspx.cs"
```

```
        Inherits="CSWebApp.ShowTrace" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title> Trace Example Page </title>
</head>
<body>
    <form runat="server">
        <div> Hello! </div>
    </form>
</body>
</html>
```

В дополнение к этому с помощью атрибута `TraceMode` можно указать, как должны выводиться сообщения трассировки, связанные с запросом страницы. Значение этого атрибута можно установить равным `SortByTime` для вывода сообщений трассировки в порядке их генерации или `SortByCategory` для их разделения по разным типам. На рис. 42.11 показан вывод трассировки для простой страницы, определенной в предыдущем коде при сортировке по категориям.

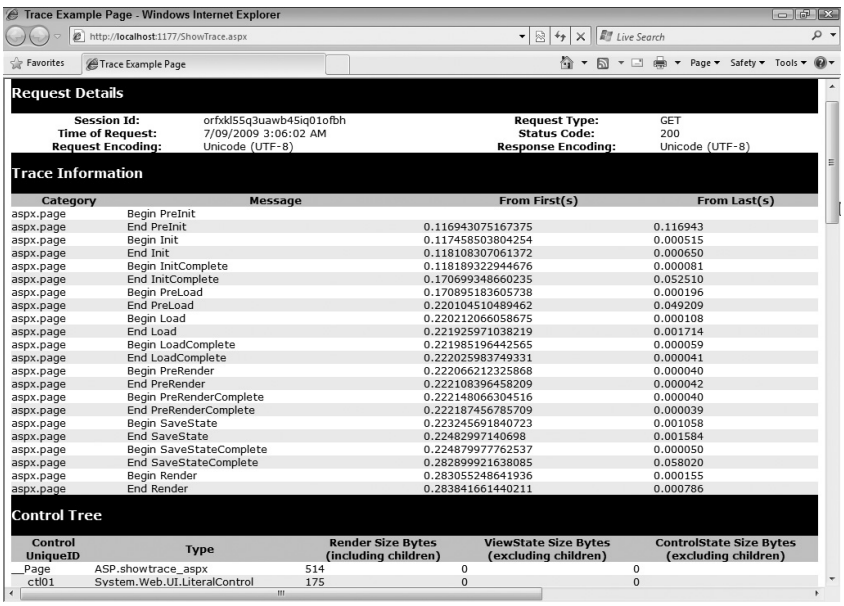


Рис. 42.11

Трассировка на уровне приложения

Трассировка на уровне приложения может быть включена в файле `web.config`. В узел `system.web` необходимо включить узел `trace` с атрибутом `enabled`, значение которого равно `true`. При использовании трассировки на уровне приложения можно управлять ее выводом с помощью атрибута `pageOutput`. Если его значение равно `true`, вы получаете информацию трассировки в нижней части каждой страницы (как на рис. 42.11), в то время как значение `false` гарантирует, что информация трассировки выводиться на странице не будет, а доступна она будет только через отладчик

Trace Viewer (рассматривается позже в этой главе). Вы также можете ограничить количество трассируемой информации с помощью атрибута `requestLimit`. Включение узла `trace` в файл `web.config`, с которым вы уже встречались в этой главе, даст нам конфигурацию наподобие следующей:

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
    <customErrors mode="RemoteOnly"
                  defaultRedirect="GenericErrorPage.htm">
      <error statusCode="403" redirect="AccessDenied.html" />
      <error statusCode="404" redirect="PageNotFound.html" />
    </customErrors>
    <trace enabled="true" pageOutput="false"
           traceMode="SortByCategory"/>
  </system.web>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true" />
  </system.webServer>
</configuration>
```

Вывод трассировки

Трассировка приводит к обильному выводу. Простой пример страницы `Hello`, рассмотренный ранее, дает почти три печатные страницы информации, включая данные следующих категорий.

- **Детали запроса.** Детальная информация о текущем сеансе, времени запроса, его типе и HTTP-код, возвращенный браузеру.
- **Информация трассировки.** Полный листинг каждого события, его начала и конца, включая время, потребовавшееся на обработку каждого события.
- **Дерево управляющих элементов.** Список всех управляющих элементов, определенных на странице, включая сам объект страницы и HTML-элементы. Для каждого объекта указан его размер, что позволяет определить, влияют ли аномальные размеры объектов на производительность вашего приложения.
- **Состояния сеанса и приложения.** Здесь перечисляются ключи и их значения как для отдельного сеанса, так и для приложения в целом.
- **Коллекции запрошенных и возвращенных идентификационных файлов (cookies).** Список всех известных ASP.NET-запросов и ответов с идентификационными файлами для системы, к которой может обращаться ваше приложение.
- **Коллекция заголовков.** Список заголовков HTTP, включенных в страницу.
- **Коллекция заголовков ответа.** Заголовки HTTP, связанные с ответом и указывающие тип возвращаемого объекта.
- **Коллекция форм.** Список всех определенных на странице форм.
- **Коллекция строк запроса.** Список всех строк, использованных в запросе страницы.
- **Переменные сервера.** Список всех серверных переменных, известных серверу ASP.NET и текущему выполняемому приложению.

Как вы можете видеть, при реализации трассировки для веб-страницы или приложения вы получаете доступ к огромному количеству информации, которую можно использовать для выяснения, как работает ваше приложение. Вы можете выяснить, нет ли отсутствующих или излишних данных в некоторых коллекциях, а также проанализировать, нет ли каких-либо событий с аномально большим временем обработки.

Обработчик Trace Viewer

Инструмент Trace Viewer представляет собой пользовательский обработчик, включаемый в веб-приложение при активизации его трассировки. При трассировке на уровне приложения вы можете перейти к странице и просмотреть весь вывод трассировки. Чтобы обратиться к обработчику Trace Viewer, откройте страницу `trace.axd` в корневом каталоге веб-сайта.

Обработчик Trace Viewer предоставляет вам сводную таблицу всех запросов, сделанных в приложении, вместе со временем, когда был сделан запрос, и кодом состояния HTTP, возвращенным в ответ. Предоставляется также ссылка на детальную информацию по каждому запросу (это та же информация, которую можно увидеть в случае трассировки на уровне страницы и которая была рассмотрена ранее), как показано на рис. 42.12.

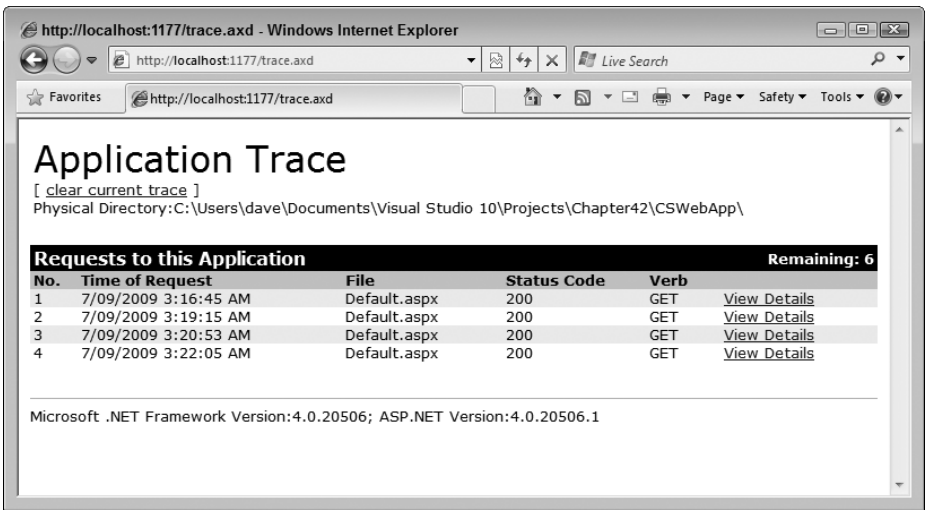


Рис. 42.12

Пользовательский вывод трассировки

Вы можете добавить к информации трассировки по умолчанию собственные сообщения с помощью методов `Trace.Warn` и `Trace.Write`. Они оба имеют одно и то же множество синтаксических перегрузок, и единственное реальное их отличие заключается в том, что сообщения, выводимые методом `Warn`, отображаются красным цветом.

Простейший вид данных команд включает только одну строку сообщения наподобие

```
Trace.Warn("Имеется проблема")
```

Однако вы можете добавить к своим предупреждениям и сообщениям категории, используя перегруженные методы, которые включают категорию и необязательный объект ошибки:

```
Trace.Warn("MyApp Error Category", "Имеется проблема", myAppException)
```

Инструмент Health Monitoring

Платформа ASP.NET включает встроенное средство для генерации и перехвата событий с целью мониторинга веб-приложения. Эта функциональная возможность, именуемая Health Monitoring, позволяет более активно управлять веб-приложениями, уведомляя обо всех возникших проблемах.



Инструмент Health Monitoring предоставляет гораздо бо́льшие возможности, чем просто уведомление о происшедших исключениях. Вы можете также оснастить свой код и генерировать сигналы о пользовательских событиях, например, если пользователь не может войти в систему или пытается обратиться к данным с ограниченным доступом.

Инструмент Health Monitoring включается посредством файла `web.config`. В узел `system.web` необходимо включить узел `healthMonitoring`, содержащий атрибут `enabled` со значением `true`. Этот узел содержит также детальную информацию о том, какого провайдера использовать, и правила обработки различных событий. Дополняя рассматривавшийся ранее файл `web.config`, мы создаем SMTP-провайдер и правило, согласно которому детальная информация обо всех необработанных исключениях пересылается веб-мастеру электронной почтой. В файл `web.config` добавлена ссылка на SMTP-сервер, так что провайдер может отправлять такие электронные уведомления.

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
    <customErrors mode="RemoteOnly"
                  defaultRedirect="GenericErrorPage.htm">
      <error statusCode="403" redirect="AccessDenied.html" />
      <error statusCode="404" redirect="PageNotFound.html" />
    </customErrors>
    <trace enabled="true" pageOutput="false"
           traceMode="SortByCategory"/>
    <healthMonitoring enabled="true">
      <providers>
        <add name="SMTPProvider"
         type="System.Web.Management.SimpleMailWebEventProvider"
         from="server@yourdomain.com"
         to="webmaster@yourdomain.com"
         subjectPrefix="Exception on WebApp:"
         bufferMode="Critical Notification"/>
      </providers>
      <rules>
        <clear />
        <add name="All Errors Default"
             eventName="All Errors"
             provider="SMTPProvider" />
      </rules>
    </healthMonitoring>
  </system.web>
```

```
  <system.net>
    <mailSettings>
      <smtp><network host="mail.yourdomain.com"/> </smtp>
    </mailSettings>
  </system.net>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true" />
  </system.webServer>
</configuration>
```

Когда все изменения внесены, при генерации исключения, которое остается необработанным, по указанному адресу высылается соответствующее электронное сообщение. В нем содержится большое количество полезной информации, включая детальную информацию об исключении и состояние стека. На рис. 42.13 приведен пример такого сообщения.

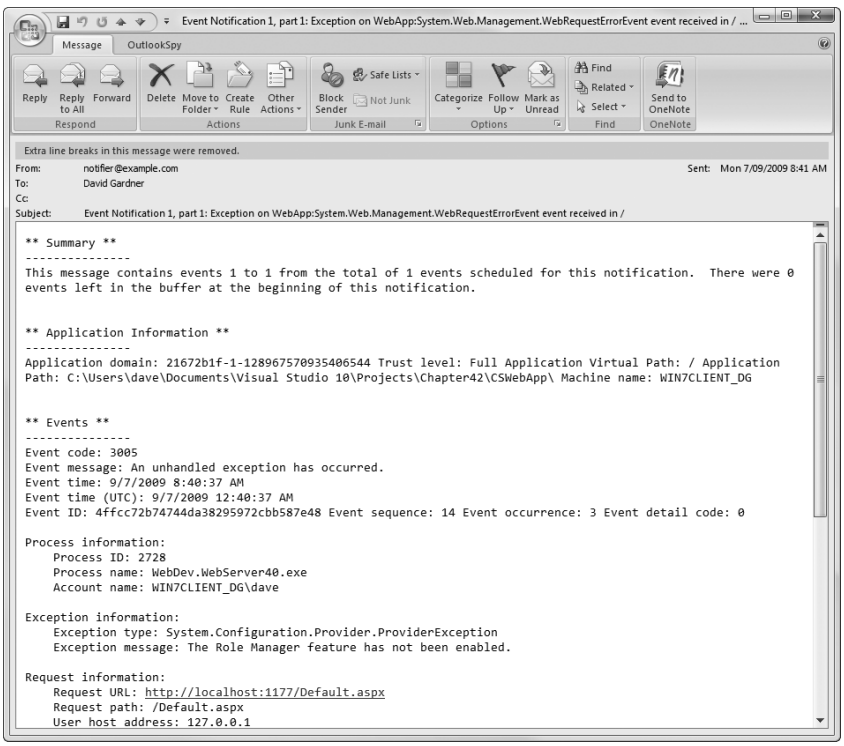


Рис. 42.13

В дополнение к SMTP-провайдеру имеются также провайдеры Event Log, WMI и SQL Server. Можно описать достаточно сложные правила направления уведомлений одному или нескольким из этих провайдеров. Если ни один из них не отвечает вашим требованиям, можете создать собственный провайдер.

Резюме

Комбинация серверных возможностей системы Visual Studio 2010 и технологии ASP.NET предоставляет большой набор инструментов, помогающий следить за веб-решениями. Эти инструменты расширяют имеющийся набор функциональных возможностей по отладке обычных приложений Windows такими специфичными функциями, как отладка кода на языке JavaScript и приложений Silverlight, обработка ошибок на уровне страницы и приложения, а также возможность трассировки кода, которую можно использовать для мониторинга работы ваших веб-приложений без помех для конечных пользователей.

Кроме того, средство Health Monitoring технологии ASP.NET позволяет активно следить за веб-приложениями, отправляя уведомления о возникших проблемах.

43

Передовые методы отладки

В ЭТОЙ ГЛАВЕ...

- Добавление отладочных действий в код с применением классов `Debug` и `Trace`
- Методы отладки приложений, работающих на удаленном компьютере
- Отладка многопоточных приложений, хранимых процедур SQL Server и смешанных приложений

Как вы уже знаете, Visual Studio 2010 предлагает много средств отладки и запуска приложений, включая перехват ошибок и вывод информации о них (вы можете вовремя предпринять необходимые меры), методы эффективной отладки веб-приложений, а также другие возможности, такие как точки прерывания и визуализация ошибок.

В Visual Studio можно найти и другие функциональные возможности для отладки проектов, баз данных и даже самого .NET Framework. В этой главе вы познакомитесь с передовыми методами отладки проектов безотносительно к языку или технологии.

Действия при старте

Система Visual Studio предоставляет несколько способов запуска приложений в начале сеанса отладки. Для большинства проектов достаточно запуска по умолчанию, что в случае Windows-приложения означает непосредственный запуск программы. В случае веб-приложения Visual Studio открывает веб-браузер по умолчанию и загружает текущую страницу либо переходит к корневому пути веб-приложения при отсутствии активных страниц.

В некоторых сценариях может потребоваться иное действие в процессе сеанса отладки, например всегда открывать определенную веб-страницу при запуске веб-приложения. В таких ситуациях вы можете изменить настройки старта на страницах свойств проекта Debug или Web. На рис. 43.1 показаны действия при запуске проекта Windows Forms.

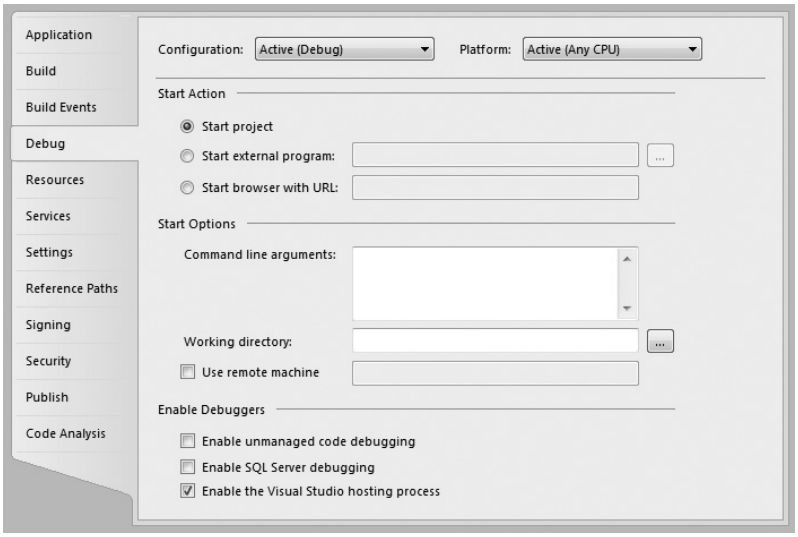


Рис. 43.1

В дополнение к непосредственному запуску проекта вы можете выбрать старт внешней программы, которая затем вовлечет ваш проект в процесс выполнения. В качестве альтернативы можно выбрать запуск веб-браузера вашей системы по умолчанию с определенным URL, в предположении, что этот URL приведет к выполнению вашего проекта.

Зачастую приложения создаются таким образом, чтобы вести себя по-разному в зависимости от аргументов командной строки. Если ваш проект таков, и вам нужно протестировать различные его конфигурации, можете воспользоваться текстовым полем Command Line Arguments, в котором указываются аргументы командной строки для выполняемого проекта. Аргументы командной строки необходимо ввести точно так же, как это будет сделано конечным пользователем вашего приложения.

Вы можете изменить каталог по умолчанию, из которого запускается ваше приложение, установив новое значение поля Working Directory. Это та же настройка, которую вы устанавливаете, редактируя ярлык Windows. Кроме того, можно указать другую машину, которая будет управлять процессом отладки приложения, активизировав опцию Use Remote Machine. Обратите внимание, что здесь нужно явно указать путь к удаленному компьютеру.

Последний раздел страницы Debug относится к другим видам отладки, которые могут выполняться одновременно с вашим приложением. По умолчанию в среде Visual Studio активен только процесс отладки управляемого кода, но вы можете включить и отладку неуправляемого кода или хранимых процедур SQL Server. Эти отладчики рассматриваются далее в данной главе.



Настройки Configuration и Platform доступны только при активизации настройки Show Advanced Build Configurations, которую можно найти на странице Projects and Solutions⇨General options и которая по умолчанию включена для всех конфигураций, кроме Visual Basic.

Действия при старте веб-приложений ASP.NET находятся на странице Web свойств проекта, показанной на рис. 43.2. По умолчанию веб-сайт запускается с текущей открытой в редакторе кода или в веб-проектировщике страницей. Можно изменить эту настройку и всегда использовать определенную страницу или URL. Еще один вариант состоит в запуске внешней программы или ожидании запроса от внешнего приложения. Это особенно полезно при отладке веб-сервиса, вызываемого другим приложением.

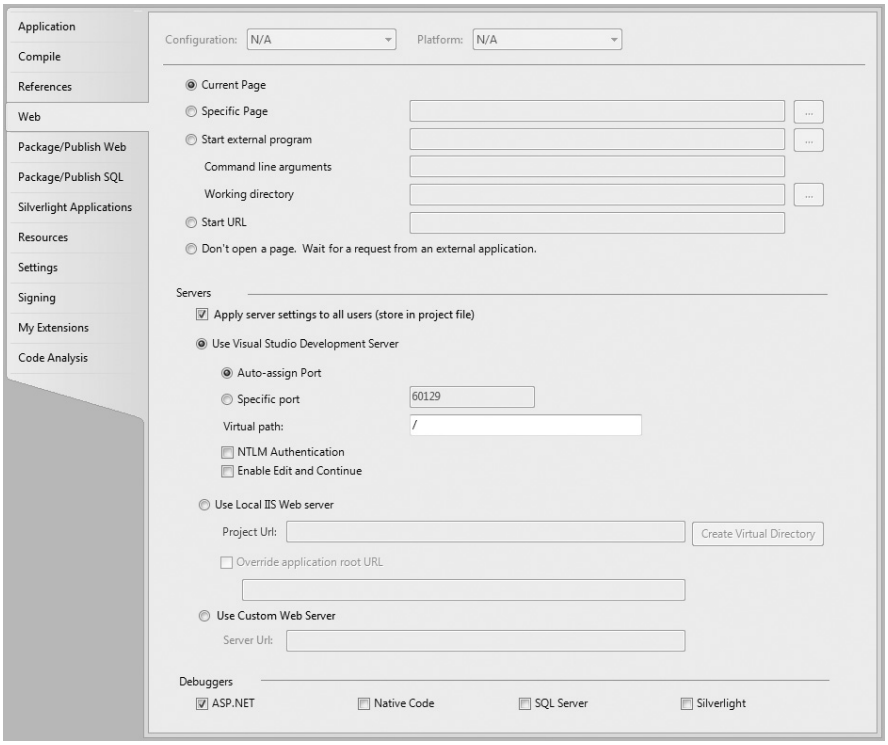


Рис. 43.2

Для проекта веб-приложения ASP.NET можно выбрать один из трех вариантов веб-сервера. Наиболее удобен встроенный Visual Studio Development Server, так как он не требует ни установки, ни настройки. В отличие от IIS, Visual Studio Development Server поддерживает функциональную возможность Edit and Continue. Опция Custom Web Server позволяет указать для отладки удаленный веб-сервер.

Отладка с применением кода

В .NET Framework в пространстве имен `System.Diagnostics` имеются три класса, которые могут использоваться для встраивания поддержки отладки в код — `Debug`, `Debugger` и `Trace`. При корректном использовании эти классы предоставляют весьма мощное средство взаимодействия с отладчиком.

Функциональность, предоставляемая всеми тремя классами, обеспечивается посредством легко добавляемых в код методов и свойств.

Класс Debugger

Класс `Debugger` предоставляет программный доступ к некоторым функциям отладчика в системе Visual Studio. Например, в приведенном далее фрагменте система проверяет, работает ли приложение под управлением отладчика, и, если не работает, запускает его и присоединяет к процессу:

C#

```
if (!Debugger.IsAttached)
{
    Debugger.Launch();
}
```

VB

```
If Not Debugger.IsAttached() Then
    Debugger.Launch()
End If
```

При выполнении этого кода вне системы Visual Studio выполнение программы приостанавливается и выводится диалоговое окно, подобное показанному на рис. 43.3. После выбора New Instance of Visual Studio 2010 приложение загружается в Visual Studio и выполнение приложения продолжается в режиме отладки.

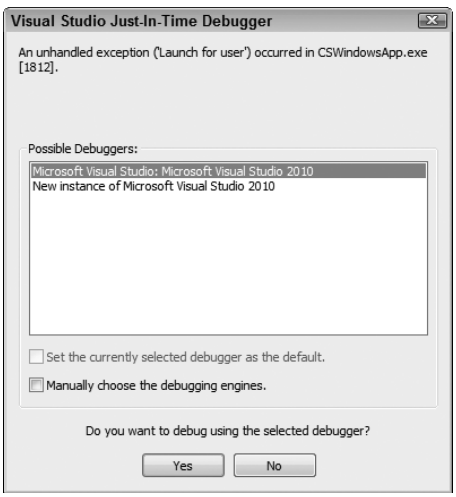


Рис. 43.3

Классы Debug и Trace

Классы `Debug` и `Trace` используются для вывода отладочной информации и трассировки пути выполнения вашего приложения. Большинство свойств и методов обоих классов одинаковы, что может показаться избыточным. Однако в реализации этих методов и представлении результатов имеется ключевое отличие.

Класс `Debug` должен использоваться только для вывода информации в режиме отладки. Класс `Trace` может применяться, если вы хотите, чтобы вывод осуществлялся и в отладочной, и в окончательной версиях. При отладке приложения в процессе разработки вывод обоих классов направляется в окно Output системы Visual Studio. Однако в окончательной версии все инструкции `Debug` подавляются компилятором и

в процессе работы вывод не осуществляется. Это позволяет включить большое количество отладочного кода в приложение, и это никак не повлияет на размер и производительность окончательной версии приложения.



Возможность использования инструкций `Trace` и `Debug` в разных конфигурациях определяется директивами компилятора. В системе Visual Studio вы можете включить или отключить эти директивы на странице свойств проекта. Для проектов на языке C# эти настройки находятся на странице свойств Build, а для проектов на языке Visual Basic открываются после щелчка на кнопке Advanced Compiler Options на странице свойств Compile.

Методы классов `Debug` и `Trace` для вывода отладочных сообщений перечислены в табл. 43.1.

Таблица 43.1. Методы для вывода отладочных сообщений

Метод	Вывод
`Write`	Текст или строка и необязательная категория
`WriteIf`	Текст и необязательная категория, если переданное в качестве аргумента условие равно `true`
`WriteLine`	Текст, за которым следует символ возврата каретки, и необязательная категория
`WriteLineIf`	Текст, за которым следует символ возврата каретки, и необязательная категория, если переданное в качестве аргумента условие равно `true`

Можно также организовать отступы при выводе сообщений, увеличивая или уменьшая их с помощью методов `Indent` и `Unindent`.

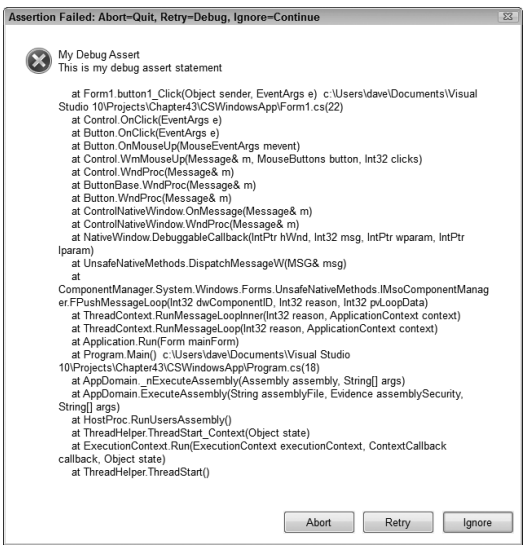


Рис. 43.4

Можно воспользоваться методом `Assert` классов `Debug` и `Trace` для создания так называемых утверждений, которые тестируют условие, переданное им в качестве аргумента. Если условие оказывается истинным, никакие действия не выполняются. Если же условие ложно, то при работе в отладочном режиме выполнение программы приостанавливается и выводится диалоговое окно, показанное на рис. 43.4.

Щелчок на кнопке Abort приводит к завершению выполнения приложения; Retry останавливает программу в данном месте, а Ignore продолжает выполнение программы.

В режиме отладки весь вывод классов `Debug` и `Trace` направляется в окно Output. Однако в окончательном выпуске весь вывод трассировки собирается в слушателе, который представляет собой объект, получающий весь вывод трассировки и записывающий его на выходное устройство. Этим устройством может быть текстовый файл, журнал событий Windows или некоторое иное хранилище информации.

Наконец, вам доступен переключатель Trace Switches, который позволяет включать, выключать и фильтровать вывод трассировки. Этот переключатель включается в файле `app.config` данного приложения.

Отладка работающих приложений

Иногда требуется выполнить отладку приложения, работающего вне системы Visual Studio. На то имеется множество причин, в частности — дефект, обнаруживающийся только в выпущенной окончательной версии. К счастью, система Visual Studio обеспечивает простой метод подключения и отладки приложения Windows или веб-приложения, которое в настоящий момент уже выполняется.

Подключение к процессу Windows

Подключение к работающему процессу Windows — очень простая задача в системе Visual Studio. В идеальном случае в системе Visual Studio должен открыться исходный текст, так что вы получаете возможность отлаживать процесс так, как если бы он был запущен в режиме отладки в системе Visual Studio.



Если вы отлаживаете выполняемый файл без доступа к исходному тексту, возможности отладки будут весьма ограничены. Если выполняемый файл собран без отладочной или символьной информации, вряд ли вам удастся получить при такой отладке сколько-нибудь полезные результаты. Поэтому рекомендуется при сборке окончательной версии на самом деле делать две сборки: одну — без отладочной информации, а другую — с ней. Соответствующая информация должна храниться где-то в безопасном месте, чтобы к ней можно было обратиться при необходимости подключения к работающему процессу или при отладке дампа памяти.

Выберите в меню Debug команду Attach to Process. При этом будет выведено диалоговое окно Attach to Process, показанное на рис. 43.5, из которого можно выбрать все активные процессы. Найдите в списке Available Processes приложение, которое вы хотите отлаживать, и щелкните на кнопке Attach.

Поскольку подключение к приложению требует выполнения этого действия вручную, оно не слишком хорошо подходит для решения проблемы, возникающей при запуске. Кроме того, отлаживая приложение, которое не требует пользовательского ввода и быстро завершается, вы можете просто не успеть к нему подключиться. В обоих случаях лучше либо запустить приложение в отладочном режиме из системы Visual Studio, либо создать сборку с инструкцией `Debug.Break()` в коде запуска приложения.

По завершении отладки нужно отсоединиться от процесса, воспользовавшись командой Debug⇨Detach All. Можете также завершить приложение командой Debug⇨Terminate All.

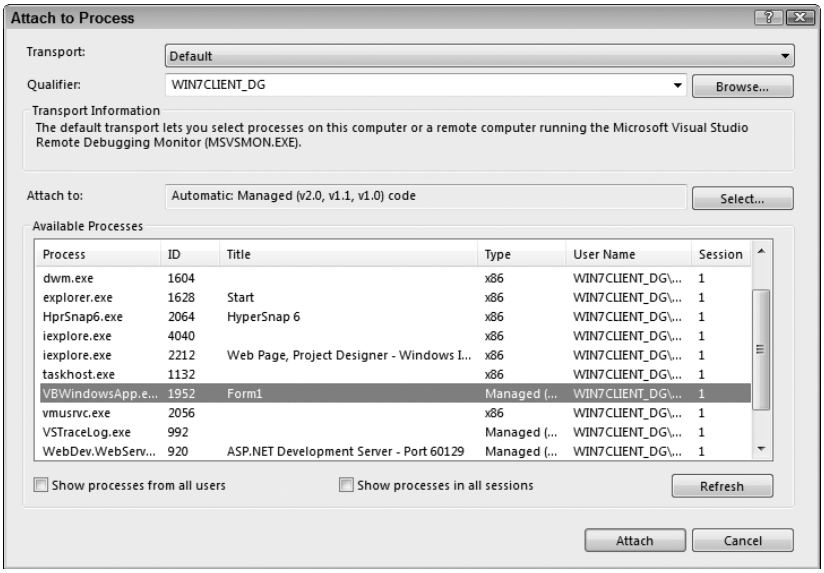


Рис. 43.5

Подключение к веб-приложению

Подключиться к веб-приложению ASP.NET почти так же просто, как и к приложению Windows. Однако, прежде чем подключиться к веб-приложению, нужно убедиться, что в файле `web.config` приложения включена возможность отладки. Найдите узел `Compilation` в элементе `system.web` и установите его атрибут `debug` равным `true`. Приведенный далее листинг содержит минимальный файл `web.config` с включенной отладкой, готовый для подключения отладчика к приложению:

```
<configuration>
  <appSettings/>
  <connectionStrings/>
  <system.web>
    <compilation debug="true" />
  </system.web>
</configuration>
```

Технология ASP.NET автоматически обнаруживает любые изменения в настройках `web.config` и немедленно их применяет. Следовательно, нет необходимости в перезапуске компьютера или IIS, чтобы внесенные изменения вступили в силу. Как говорилось в главе 42, это изменение может влиять на производительность, так что в окончательной, поставляемой пользователям версии этот атрибут оставлять не стоит.

После включения возможности отладки можете подключиться к веб-приложению. Процесс, к которому необходимо подключиться, представляет собой процесс ASP.NET, который является процессом либо на сервере IIS (он называется `w3wp.exe` в IIS 6.0 или в более поздней версии либо `aspnet_wp.exe` в более старых версиях IIS), либо во встроенном сервере разработки Visual Studio 2010 `WebDev.WebServer.exe`.



Поскольку обычно процесс IIS работает от имени учетной записи `ASPNET` или `NETWORK SERVICE`, запускать систему Visual Studio нужно с правами администратора, чтобы иметь возможность подключить отладчик.

Чтобы начать отладку, выберите команду Attach to Process меню Debug. Установите флажок Show Processes in All Sessions, если вы подключаетесь к системе ASP.NET под управлением сервера IIS. Найдите нужный процесс ASP.NET в списке Available Processes и щелкните на кнопке Attach. Как показано на рис. 43.6, вас могут попросить перезапустить систему Visual Studio с правами более высокого уровня.

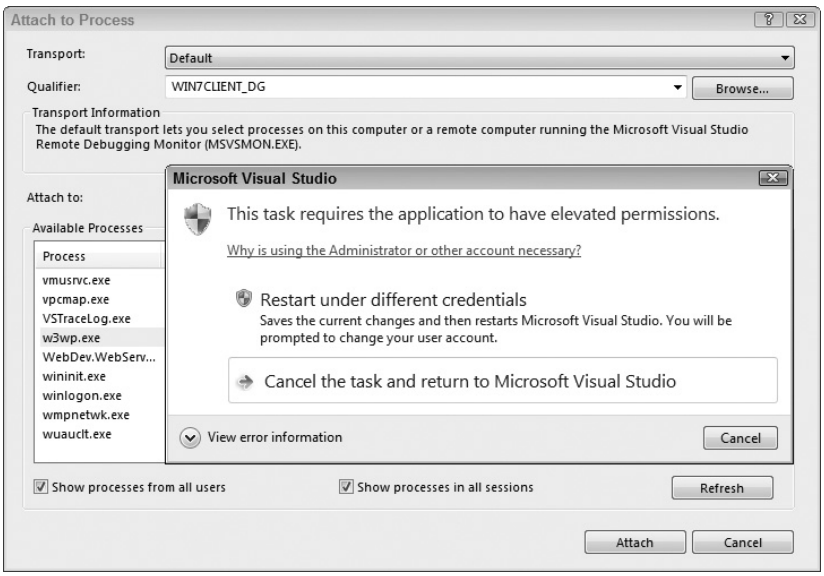


Рис. 43.6

Удаленная отладка

Удаленная отладка позволяет подключиться к процессу, работающему на другой машине. Это может быть полезно в тех случаях, когда ошибка проявляет себя только на "чужих" компьютерах либо когда необходимо отладить Windows Service или веб-приложение ASP.NET, работающее на промышленном сервере.

Отладка удаленного приложения не отличается от отладки локального приложения. Подключившись к удаленному приложению, можно устанавливать точки прерывания, просматривать переменные и проходить код шаг за шагом. Однако перед тем как подключиться к удаленному процессу, нужно убедиться, что на отлаживаемой машине работает программа Remote Debugging Monitor.

Remote Debugging Monitor (`msvsmon.exe`) — это небольшая программа, поставляемая с Visual Studio 2010. По умолчанию ее 32-разрядная версия устанавливается в каталоге `C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\Remote Debugger\x86`.



Версия x64 `msvsmon.exe` по умолчанию не устанавливается вместе с Visual Studio 2010, если вы не работаете с 64-разрядной версией Windows. Версия `msvsmon.exe` IA-64 доступна только в Visual Studio Team System.

Можно просто скопировать эту папку на удаленную машину и запустить программу локально, а можно создать совместно используемый ресурс и запустить программу с

применением UNC-пути. Можно также инсталлировать Remote Debugging Monitor на удаленной машине, запустив установочный MSI-файл, который имеется на Visual Studio DVD в каталоге `Remote Debugger`.

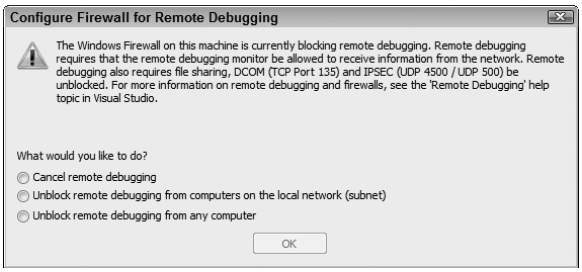


Рис. 43.7

Когда вы первый раз запускаете программу `msvsmon.exe` на удаленном компьютере, она пытается настроить брандмауэр Windows таким образом, чтобы открыть порты, необходимые для удаленной отладки. В некоторых средах, таких как Windows Server 2003, программа предложит вам внести необходимые изменения, как показано на рис. 43.7. На машине разработчика Visual Studio вносит необходимые изменения в брандмауэр, обеспечивая возможность подключения к удаленной машине.

После запуска программа Remote Debugging Monitor просто прослушивает сеть в ожидании входящих запросов отладки. По умолчанию удаленные запросы должны быть аутентифицированными, и только пользователи, которые являются администраторами, имеют необходимые права для подключения и отладки приложений. Настройки безопасности можно изменить с помощью команды меню Tools⇨Options в диалоговом окне, показанном на рис. 43.8.

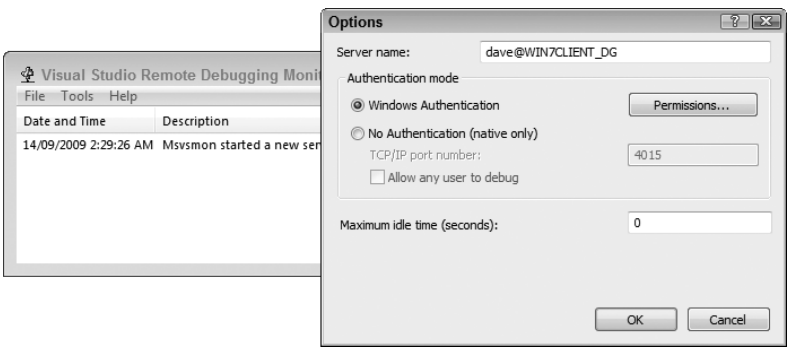


Рис. 43.8



При включенном режиме No Authentication ваша машина будет открыта для любого пользователя в сети. Удаленный пользователь сможет запускать на вашем компьютере приложения, обращаться к данным и даже выполнять деструктивные действия посредством отладчика. Вы предупреждены!

После того как на удаленной машине запущена программа Remote Debugging Monitor, вы можете подключиться к приложению на этой машине с помощью команды меню Debug⇨Attach to Process. Введите имя компьютера или IP-адрес удаленной машины в поле Qualifier. Visual Studio подключится к Remote Debugging Monitor, аутентифицирует вас и выведет список процессов, работающих на удаленной машине. Просто выберите процесс, к которому вы подключаетесь, и вы сможете отлаживать его так же, как если бы это был локальный процесс.

Исходные тексты платформы .NET Framework

Одной из интересных тенденций, демонстрируемых в последние годы Microsoft, является возрастающая открытость и даже готовность перейти к открытым исходным текстам. Технология ASP.NET MVC Framework, рассмотренная в главе 21, — хороший пример данной тенденции, поскольку ее исходные тексты распространяются в виде собираемого решения Visual Studio.

Однако, пожалуй, еще более важным примером является выпуск исходных текстов большого количества базовых библиотек классов .NET Framework. Доступные только для чтения по лицензии Microsoft Reference License, они позволяют шаг за шагом проходить код .NET Framework и отлаживать его, как часть сеанса отладки. Хотя вы всегда можете вывести намерения программиста с помощью надстройки Reflector, это нельзя сравнить с просмотром реального исходного текста, *включая оперативную документацию*. Действительно хорошей новостью является то, что эта документация практически всеобъемлюща.



Исходный текст недоступен ни для каждой сборки, поставляемой как часть .NET Framework, ни для каждой версии. Для сборок с доступными исходными текстами зачастую имеется задержка между выпуском сборки и доступностью ее кода. Список доступных в настоящий момент сборок можно найти по адресу `http://referencesource.microsoft.com/netframework.aspx`.

Первый шаг для обеспечения возможности обращения к исходным текстам состоит в конфигурировании некоторых настроек Debugger. Выберите команду меню Tools⇨Options и категорию Debugging. Если вы используете Visual Basic Profile, выберите опцию Show All Settings, чтобы увидеть все возможные настройки. Убедитесь, что у вас установлен флажок Enable .NET Framework Source Stepping, как показано на рис. 43.9. После того как вы установите данный флажок, вам могут быть представлены два приглашения; первое указывает, что опция Enable Just My Code отключена, а второе сообщает о том, что установлено значение местоположения символьного кэша по умолчанию.

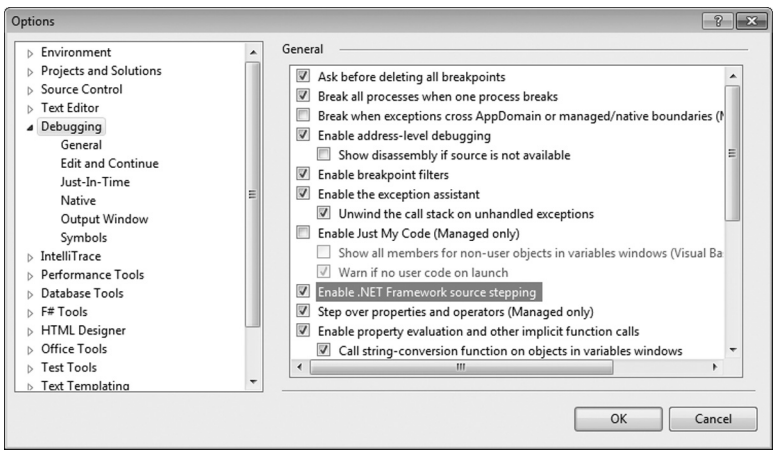


Рис. 43.9

Далее перейдите к категории Symbols диалогового окна Options (рис. 43.10) и проверьте автоматически установленное значение местоположения символьного кэша. При необходимости можете изменить указанное значение, но сначала убедитесь в наличии полного доступа для чтения и записи данного каталога. Если вы настраиваете этот параметр во время работы в режиме отладки, можете немедленно загрузить символьную информацию, щелкнув на кнопке Load all symbols. В противном случае, если вы не находитесь в режиме отладки, символьная информация к текущему проекту будет загружена, как только вы щелкнете на кнопке OK.

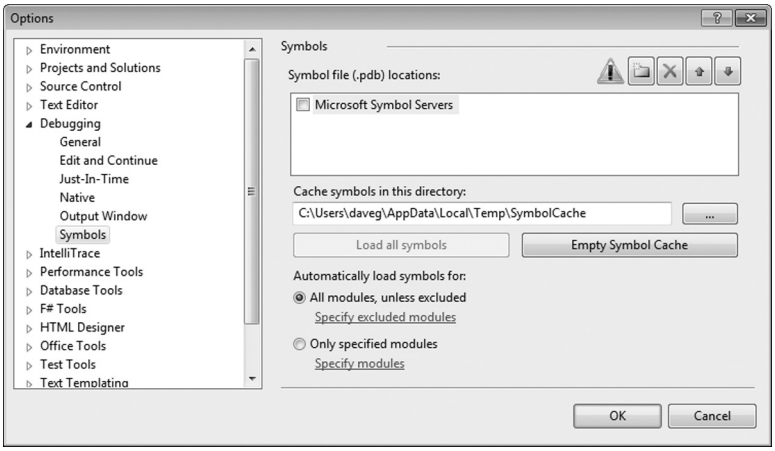


Рис. 43.10

Теперь вы можете шаг за шагом проходить и отлаживать библиотеки базовых классов .NET Framework в процессе сеанса отладки. Установите точку прерывания в коде вашего приложения и запустите его в режиме отладки. По достижении точки прерывания откройте окно Call Stack (Debug⇨Windows⇨Call Stack), чтобы просмотреть путь выполнения. Если символьная информация загружена, код, доступный для отладки, не будет заблокирован, и вы сможете дважды щелкнуть на нем в окне Call Stack или открыть доступ к исходному тексту в сеансе отладки, как показано на рис. 43.11. Если это первый просмотр кода, будет выведена лицензия Microsoft Reference Library.

Отладка многопоточных и распараллеленных приложений

Многопоточные приложения традиционно обладают дурной славой трудно отлаживаемых. По-видимому, причиной этого является необходимость решения таких фундаментальных задач, как отслеживание того, какой поток выполнения в настоящее время изучается и чем заняты все прочие потоки. К счастью, Visual Studio 2010 обладает усовершенствованной поддержкой отладки многопоточных приложений.

В главе 39 мы рассматривали окно отладки Threads, в котором перечисляются все активные потоки определенного приложения. Доступная в этом окне функциональность включает возможность определить для потока дружественное имя. Вы также можете устанавливать флажки для отдельных потоков, которые позволяют тратить меньше времени на отслеживание потоков с заданными идентификаторами.

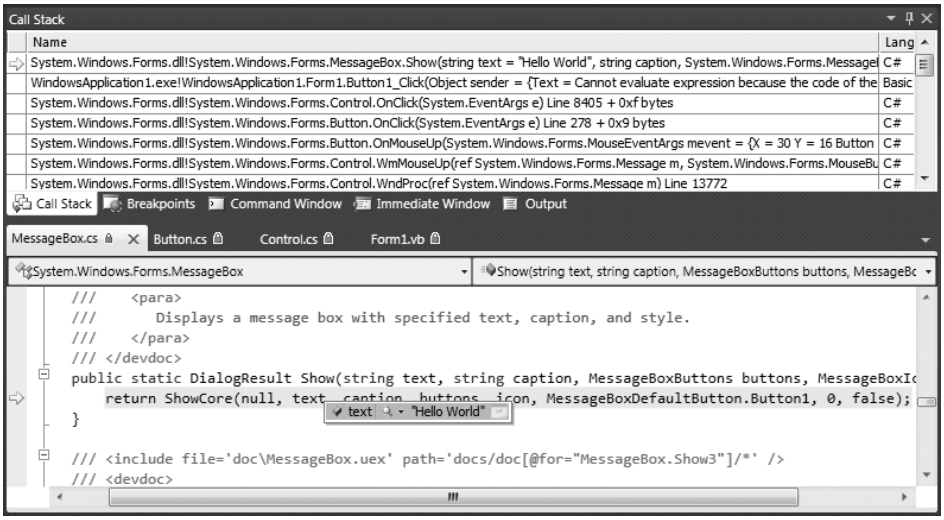


Рис. 43.11

Дальнейшее усовершенствование отладки связано с возможностью идентификации каждого потока в окне редактирования исходного текста. Вы можете щелкнуть в окне Threads правой кнопкой мыши на любой записи и выбрать в контекстном меню команду Show Threads in Source. Результат этого действия показан на рис. 43.12, где на полях показана пиктограмма потока (на которой изображены красная и синяя волнистые линии). Пиктограмма указывает, что поток или несколько потоков остановлены в данном месте. Если навести на пиктограмму указатель мыши, будет выведена подсказка, указывающая, какой именно поток остановлен. В качестве имен потоков используются дружественные имена, введенные вами в окне Threads.

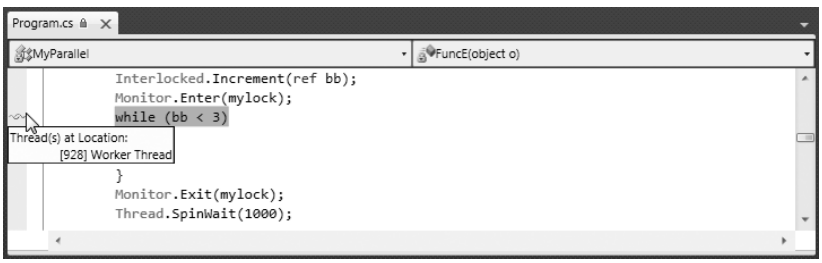


Рис. 43.12

С помощью панели инструментов Debug Location, показанной на рис. 43.13, вы можете перемещаться между потоками. Когда в раскрывающемся списке Thread вы выбираете поток, отличный от текущего, стек вызовов обновляется и содержит путь выполнения выбранного потока, а точка выполнения перемещается к текущей позиции в исходном тексте. Диаграмма вызовов в окне Parallel Stacks также обновляется, отражая тот факт, что в качестве текущего выбран новый поток.

Вы также можете отмечать флажками потоки и задачи в окнах Threads и Parallel Tasks. Пометка с помощью флажков позволяет отслеживать поток или задачу в пределах сеанса отладки и отфильтровывать задачи или потоки, которые вас не ин-

тересуют. На рис. 43.14 флажком помечены первые две задачи в окне Parallel Tasks. Выбрав опцию Show Only Flagged на панели инструментов окна Parallel Stacks, мы фильтруем диаграмму вызовов, заставляя ее скрывать не интересующие нас задачи.

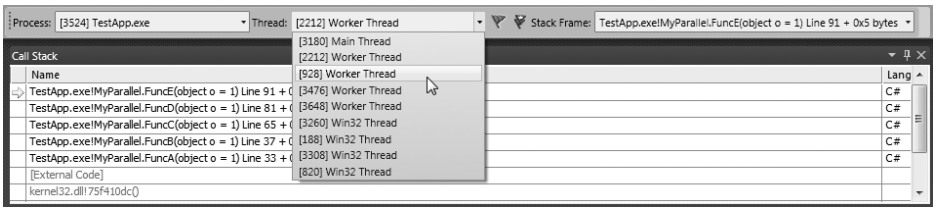


Рис. 43.13

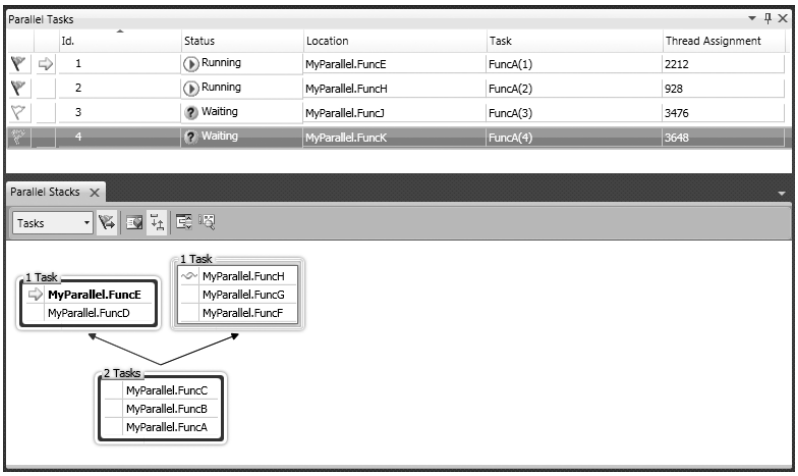


Рис. 43.14

Хотя отладка многопоточных и распараллеленных приложений остается нетривиальной задачей, рассмотренные возможности облегчают изучение определенных потоков или задач, а также отфильтровывают не интересующую нас информацию из окон отладчика Visual Studio.

Отладка хранимых процедур SQL Server

Другой очень полезной возможностью модели отладки в Visual Studio 2010 является способность отлаживать хранимые процедуры в базах данных SQL Server. Сначала следует установить флажок Enable SQL Server Debugging на вкладке Debug страницы свойств проекта, как показано на рис. 43.15. После активизации, когда ваш код сталкивается с хранимой процедурой, вы можете отлаживать ее код наряду с вашим собственным.

Вы можете даже включать в хранимую процедуру точки прерывания, так что можно отлаживать код SQL Server без остановки выполнения кода приложения.

Для отладки хранимых процедур ваша учетная запись Windows должна быть членом группы системных администраторов SQL Server.

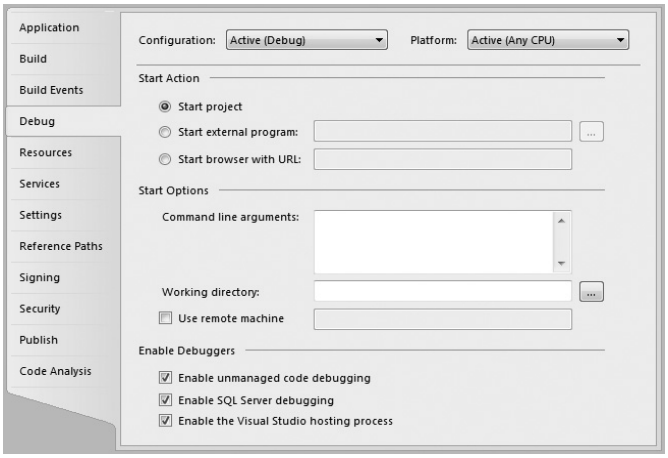


Рис. 43.15

Отладка в смешанном режиме

Приложение смешанного режима (mixed-mode application) — это приложение, в котором объединяется управляемый код (Visual Basic, C#, управляемый C++ и т.д.) и машинный код (обычно C++). Отладка приложений смешанного режима не сильно отличается от отладки приложения с управляемым кодом, однако сначала следует настроить приложение так, чтобы поддерживалась отладка машинного кода. На рис. 43.15 из предыдущего раздела показано включение отладчика неуправляемого кода вместе с включением отладчика SQL Server.

Отладка в смешанном режиме имеет определенные ограничения, о которых вам следует знать. Во-первых, она доступна только в Windows 2000 или более поздних операционных системах. Во-вторых, при отладке в смешанном режиме вы столкнетесь с тем, что некоторые операции, такие как пошаговое прохождение кода, работают очень медленно. Это можно исправить, убрав флажки Enable Property Evaluation и Other Implicit Function Calls на странице настройки отладчика.



В связи с тем, что стеки вызовов машинного и управляемого кодов различны, отладчик не может всегда предоставлять единый стек вызовов для смешанного кода. Хотя и редко, в стеке вызова возможны определенные несовместимости. Подробнее об этом можно прочесть в библиотеке MSDN.

“Посмертная” отладка

Несмотря на самое качественное тестирование приложений, в них неизбежно будут присутствовать скрытые ошибки, которые проявятся уже после выпуска программы. К счастью, можно отладить многие ошибки после того, как они произошли на компьютерах пользователей.

“Посмертная” отладка включает исследование дампа памяти приложения, полученного в момент проявления ошибки или неожиданного поведения. Это может произойти в момент генерации необработанного исключения или при “зависании” про-

граммы, или просто когда приложение демонстрирует поведение, указывающее на наличие утечки памяти.

В прошлом для отладки дампов памяти .NET-приложений использовались такие инструменты, как WinDbg с расширением Son of Strike (SOS). Однако WinDbg разработан для отладки машинного кода, и даже с дополнительной поддержкой SOS решение задачи соответствия MSIL исходному коду остается весьма сложным делом.

Система Visual Studio 2010 и платформа .NET Framework 4.0 привносят новую функциональность, которая облегчает отладку дампов памяти .NET-приложений.



Описанная здесь посмертная отладка работает только для приложений и веб-сайтов .NET версии 4.0. К тому же гораздо лучше, если ваше приложение компилируется в отладочной конфигурации. В противном случае вы не получите доступ к большому количеству полезной информации.

Генерация файлов дампов

Имеется несколько способов генерации файлов дампов памяти, в том числе с помощью приложения Windows Task Manager, отладчика WinDbg и самой системы Visual Studio. В системе Windows Vista или более поздних операционных системах простейший способ заключается в щелчке правой кнопкой мыши на процессе в окне Windows Task Manager и выборе в контекстном меню команды Create Dump File.

Одним из более функциональных инструментов для генерации дампов является сценарий `adplus.vbs`, который представляет собой интерфейс командной строки для обращения к отладчику WinDbg. Этот сценарий и программа WinDbg инсталлируются вместе с пакетом Debugging Tools for Windows, который можно найти по адресу `http://www.microsoft.com/whdc/DevTools/Debugging/`. Вам нужно инсталлировать версию, соответствующую архитектуре процессора целевой машины (x86, x64, Itanium).

Для генерации файла дампа следует открыть окно командной строки, заменить текущий каталог каталогом, в котором установлен пакет Debugging Tools, и ввести команду

```
adplus -hang -pn processname.exe
```

Эта команда подключается к процессу *`processname.exe`* в неинвазивном режиме, генерирует дамп и отключается. Приложение после этого продолжает свою работу.

Если вы отлаживаете "зависшее" приложение, приложение, использующее большое количество памяти, или приложение с неожиданным поведением, вам нужно получить один или более дампов в подходящее время. Возможно, придется действовать методом проб и ошибок, чтобы получить дамп с полезной информацией.

Если вы отлаживаете определенное исключение, генерируемое приложением, то следует использовать аргумент `-c` для передачи файла, конфигурирующего `adplus` для генерации дампа в момент генерации исключения.

Можно воспользоваться Visual Studio 2010 для генерации дампа в процессе сеанса отладки. Для этого приостановите выполнение приложения и выберите команду меню Debug⇨Save Dump As.

Отладка файлов дампов

Для получения максимальной пользы от "посмертной" отладки, необходимо настроить систему Visual Studio для загрузки ваших символьных файлов. Символьные файлы имеют расширение PDB и генерируются как часть отладочной сборки. Вы можете найти их в соответствующем каталоге; для каждой сборки имеется по одному такому файлу.

Выберите в среде Visual Studio сначала команду меню Tools⇨Options, а затем — подкатегорию Symbols в категории Debugging. Вы можете указать либо URL, либо локальный каталог в качестве местоположения ваших символьных файлов. Команда Microsoft Symbol Servers будет уже включена; добавьте собственные каталоги, как показано на рис. 43.16.

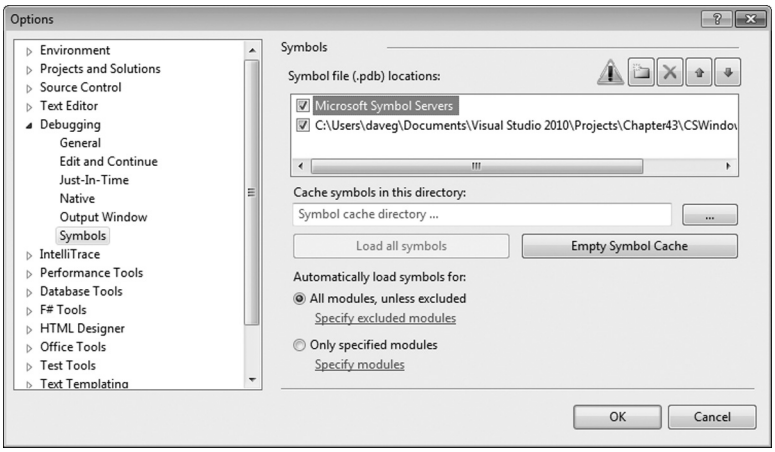


Рис. 43.16

Теперь, когда у вас есть сгенерированный файл дампа и настроены символы, вы используете для начала "посмертной" отладки в системе Visual Studio. Выберите команду меню File⇨Open⇨File и укажите местоположение файла дампа. После его открытия система Visual Studio выведет сводную страницу, как показано на рис. 43.17.

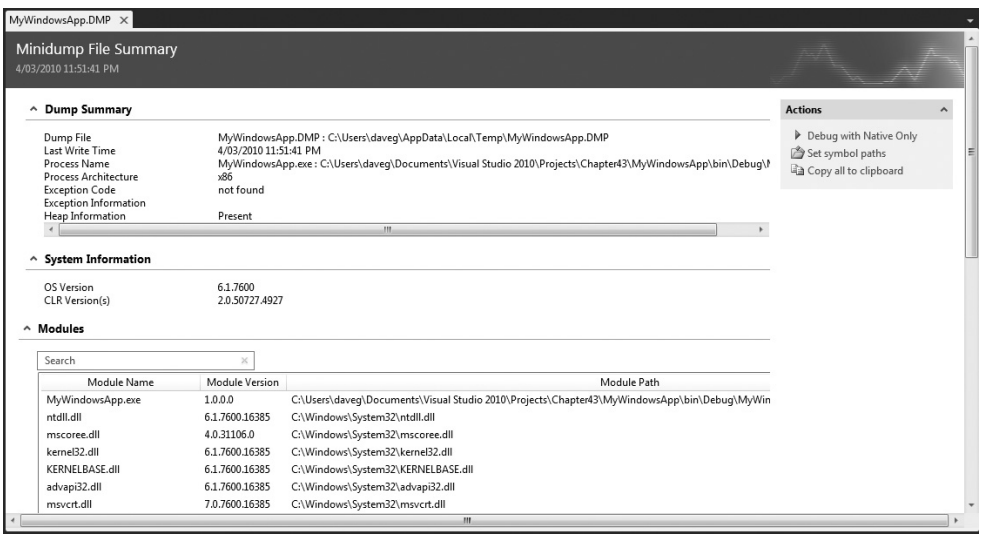


Рис. 43.17

Щелкните на ссылке Debug with Mixed для загрузки дампа и всех символов. Эта ссылка выводится только тогда, когда дамп получен от управляемого приложения .NET Framework 4.0; в противном случае вы можете воспользоваться только вариантом Debug with Native Only.

Отладка файла дампа похожа на все прочие сеансы отладки — вы можете вывести стек вызовов, просмотреть значения переменных и изучить потоки. Главное ограничение состоит в том, что, поскольку вы видите моментальный снимок, а не "живое" приложение, вы не можете шаг за шагом выполнять исходный код.

Резюме

В этой главе рассмотрение отладки ваших проектов и приложений завершается описанием передовых методов отладки. Система Visual Studio 2010 позволяет выполнить широкий спектр сценариев отладки, таких как отладка многопоточных приложений, хранимых процедур, неуправляемого кода и даже самой платформы .NET Framework. Эти методы предоставляют в ваше распоряжение очень эффективные варианты отладки для решения любых вопросов независимо от конкретного языка программирования или технологии приложения.

ЧАСТЬ X

Построение и развертывание

- **ГЛАВА 44.** Переход на систему Visual Studio 2010
- **ГЛАВА 45.** Настройка сборки приложений
- **ГЛАВА 46.** Контроль версий и подписи сборок
- **ГЛАВА 47.** Умышленное запутывание, мониторинг и управление приложениями
- **ГЛАВА 48.** Упаковка и развертывание
- **ГЛАВА 49.** Развертывание веб-приложений

44

Переход на систему Visual Studio 2010

В ЭТОЙ ГЛАВЕ...

- Использование преимуществ новой среды при работе со старыми проектами
- Обновление проектов для работы с последней средой времени выполнения и библиотеками

Всякий раз, когда выходит новая версия системы Visual Studio, возникает задержка перед началом ее использования разработчиками. В первую очередь, она связана с необходимостью одновременного обновления существующих приложений до новой версии платформы .NET Framework. Например, для перехода от версии Visual Studio 2003 к версии Visual Studio 2005 пришлось обновлять приложения до версии .NET Framework 2.0. В случае системы Visual Studio 2008 переход к последней версии интегрированной среды разработки можно было выполнить независимо от перехода к новой версии .NET Framework. Это особенно важно, если вам необходимо создавать приложения для старых версий Windows, для которых в новейших версиях .NET Framework поддержка отсутствует.

Из этой главы вы узнаете, как можно перенести существующие приложения .NET в систему Visual Studio 2010. Это делается в два этапа: обновление до системы Visual Studio 2010 и обновление используемой приложением версии до .NET Framework 4.0.

Переход к системе Visual Studio 2010 от CBCNTVS Visual Studio 2008

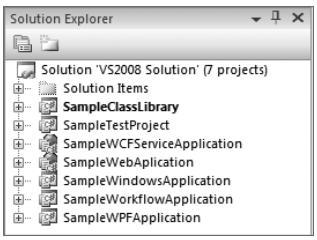


Рис. 44.1

Начнем с решения, которое содержит "винегрет" из типов приложений. На рис. 44.1 показано решение для системы Visual Studio 2008, содержащее проекты Class Library, Unit Test, WCF Web Site, Web Application, Windows Application, Workflow Application и WPF Application. Проекты WCF Web Site, Web Application, Unit Test и Workflow Application ссылаются на проект Class Library, а проекты Windows Application и WPF Application — на проект WCF Service Application.

Обновление этого решения такое же простое, как и его открытие в системе Visual Studio 2010. При этом автоматически вызывается мастер Visual Studio Conversion Wizard, показанный на рис. 44.2. Этот мастер относительно прост, с единственной опцией — нужно ли создать резервную копию перед обновлением. Пожалуй, лучше согласиться и сделать резервную копию, чтобы у вас осталась рабочая версия, если что-то пойдет не так.

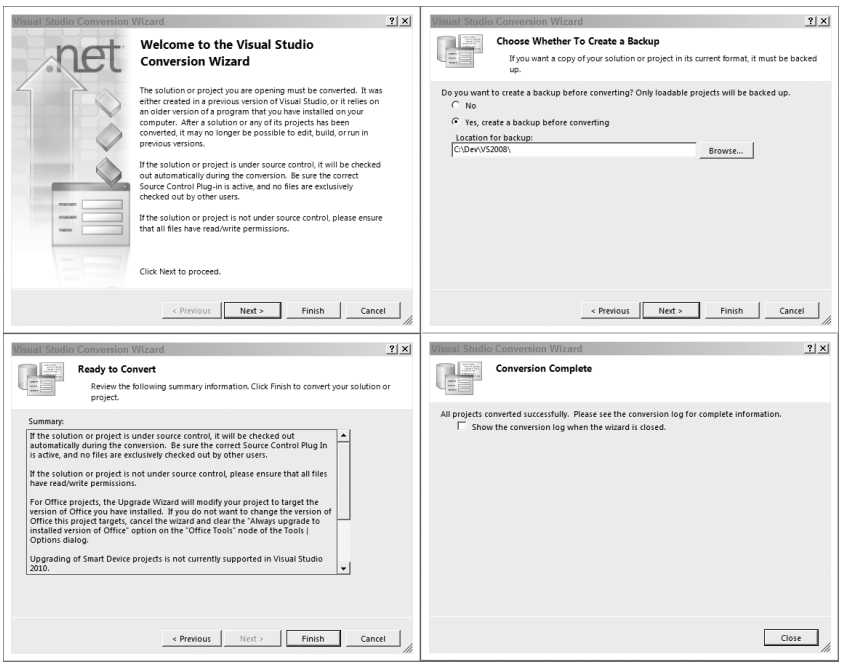


Рис. 44.2



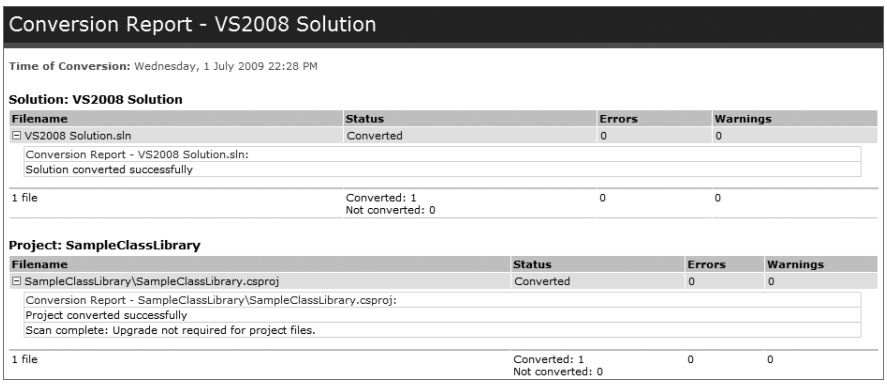
Инструменты, необходимые для разработки приложений для мобильных устройств, изначально в систему Visual Studio 2010 включены не были. Это означает, что вы не можете обновлять существующие приложения для устройств от версии Visual Studio 2008 до версии Visual Studio 2010.

Экран Summary напоминает, что в ваше решение могут быть внесены изменения и что в некоторых случаях частью процесса обновления могут быть изменения схем и ссылок. Список проектов, которые будут обновлены, на копии экрана не показан.



Если в вашем решении имеется проект Web Site, вам будет предложено обновить его до версии .NET Framework 4.0. Если вы предпочитаете не делать этого сейчас, вы сможете сделать это позже.

Даже если мастер Conversion Wizard сообщает об отсутствии ошибок, все равно лучше просмотреть журнал преобразования по завершении работы. Этот журнал (`UpgradeLog.xml` в папке решения) выглядит так, как показано на рис. 44.3, и обычно в нем перечисляются только те файлы решений и проектов, которые были обновлены. Выполняя обновление старых (до версии Visual Studio 2005) решений, вы можете столкнуться с проблемами, поскольку между версиями имеются определенные несовместимости. Несмотря на способность создавать приложения для нескольких версий платформы .NET Framework, система Visual Studio 2010 ограничена версией 2.0 и более поздними. Поэтому, если у вас имеется решение, использующее более ранние версии, мастер Conversion Wizard попытается его обновить.



Conversion Report - VS2008 Solution

Time of Conversion: Wednesday, 1 July 2009 22:28 PM

Solution: VS2008 Solution

Filename	Status	Errors	Warnings
⊟ VS2008 Solution.sln	Converted	0	0
Conversion Report - VS2008 Solution.sln:			
Solution converted successfully			
1 file	Converted: 1 Not converted: 0	0	0

Project: SampleClassLibrary

Filename	Status	Errors	Warnings
⊟ SampleClassLibrary\SampleClassLibrary.csproj	Converted	0	0
Conversion Report - SampleClassLibrary\SampleClassLibrary.csproj:			
Project converted successfully			
Scan complete: Upgrade not required for project files.			
1 file	Converted: 1 Not converted: 0	0	0

Рис. 44.3.

Возможность создания многоцелевых приложений в системе Visual Studio 2010 приводит к тому, что обновление решения не означает обновления версии .NET Framework, для которой вы создаете свои приложения. Так что вопрос в том, что именно изменено и к каким последствиям это может привести. Ответ на него заключается в минимальных изменениях в файлах как решений, так и проектов. В файле решения изменения могут касаться только заголовка. Например, строки

```
Microsoft Visual Studio Solution File, Format Version 10.00
# Visual Studio 2008
```

станут строками

```
Microsoft Visual Studio Solution File, Format Version 11.00
# Visual Studio 2010
```

Изменения в файле проекта включают дополнительные изменения и атрибуты. Например, код



```
<Project
  ToolsVersion="3.5"
  DefaultTargets="Build"
```

```
    xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <PropertyGroup>
      <Configuration Condition=" ' $(Configuration) ' == ' '
">Debug</Configuration>
      <Platform Condition=" ' $(Platform) ' == ' ' ">AnyCPU</Platform>
      <ProductVersion> 9.0.30729 </ProductVersion>
      <SchemaVersion> 2.0 </SchemaVersion>
      <ProjectGuid> {8735A946-4A21-4921-A4F9-E9645ABCF9ED}
      </ProjectGuid>
      <OutputType> Library </OutputType>
      <AppDesignerFolder> Properties </AppDesignerFolder>
      <RootNamespace> ClassLibrary1 </RootNamespace>
      <AssemblyName> ClassLibrary1 </AssemblyName>
      <TargetFrameworkVersion> v3.5 </TargetFrameworkVersion>
      <FileAlignment> 512 </FileAlignment>
    </PropertyGroup>
...
```

VS2008Solution\ClassLibrary1\ClassLibrary1.csproj

превращается в



```
<Project
  ToolsVersion="4.0"
  DefaultTargets="Build"
  xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug
    </Configuration>
    <Platform Condition=" '$(Platform)' == '' ">
        AnyCPU
    </Platform>
    <ProductVersion>9.0.30729</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{8735A946-4A21-4921-A4F9-E9645ABCF9ED}
    </ProjectGuid>
    <OutputType>Library</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>ClassLibrary1</RootNamespace>
    <AssemblyName>ClassLibrary1</AssemblyName>
    <TargetFrameworkVersion>v3.5</TargetFrameworkVersion>
    <FileAlignment>512</FileAlignment>
    <FileUpgradeFlags>
    </FileUpgradeFlags>
    <OldToolsVersion>3.5</OldToolsVersion>
    <UpgradeBackupLocation />
  </PropertyGroup>
...
```

Upgraded\ClassLibrary1\ClassLibrary1.csproj



Если у вас есть разработчики, работающие и в системе Visual Studio 2008, и в системе Visual Studio 2010, то они могут иметь общий набор файлов проектов, и два файла решений — по одному для каждой версии Visual Studio. Даже после обновления проекта до версии Visual Studio 2010 его файлы могут открываться в обеих версиях среды разработки. К сожалению, этого нельзя сказать о системе MSBuild — после обновления своих проектов вам придется обновить и процесс сборки, чтобы использовать систему MSBuild версии 4.0.

Обновление до версии .NET Framework 4.0

Если вы перенесли свои приложения в систему Visual Studio 2010 и навели порядок в среде построения приложений, подумайте о дальнейшем обновлении — до версии .NET Framework 4.0. В отличие от обновления до версии .NET Framework 2.0, где имелся ряд несовместимых изменений, обновление до версии 4.0 выполняется относительно безболезненно. Хотя вам и может понадобиться выполнить некоторые изменения, основная часть вашего приложения должна обновиться без проблем.

В большинстве случаев обновление приложения сводится к изменению свойства проекта Target Framework. На рис. 44.4 показано диалоговое окно свойств для проекта C# Class Library. На вкладке Application имеется раскрывающийся список доступных целевых версий .NET Framework.

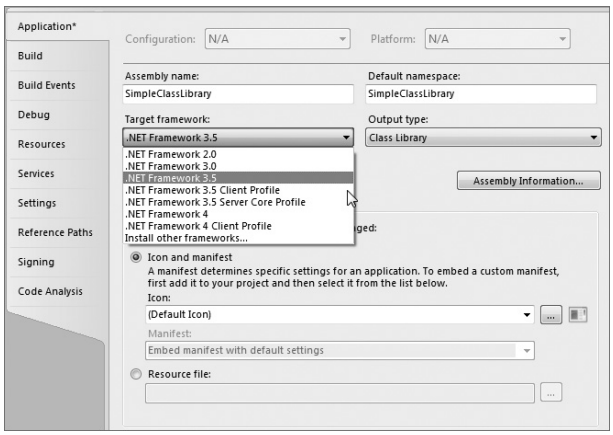


Рис. 44.4

Как только вы выберете новую версию .NET Framework, появится диалоговое окно, изображенное на рис. 44.5. Если вы выберете Yes, все ожидающие подтверждения изменения будут внесены, проект будет закрыт, обновлен и вновь открыт уже с новой целевой версией .NET Framework. Рекомендуется немедленно собрать проект, чтобы убедиться в его компилируемости.

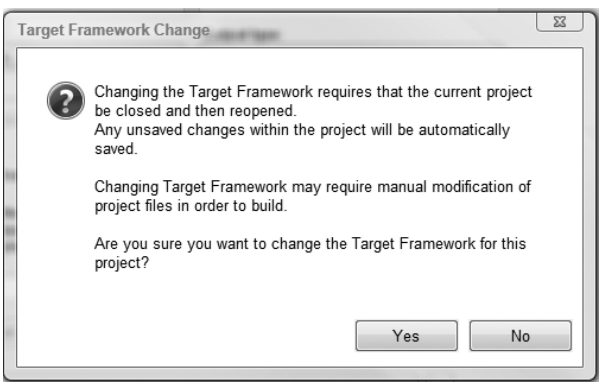


Рис. 44.5



В случае проектов VB этот раскрывающийся список находится в диалоговом окне Advanced Compile Options, к которому можно обратиться из вкладки Compile проектировщика свойств проекта.

Резюме

Из этой главы вы узнали, как обновить существующие приложения .NET до версий Visual Studio 2010 и .NET Framework версии 4.0. Последний набор инструментов и версия .NET Framework повышают производительность, функциональность и удобство ваших приложений. Однако не следует забывать об ограничениях, накладываемых применением последних версий .NET Framework. Если ваш целевой рынок использует старые операционные системы, такие как Windows 2000, вы можете предпочесть остановиться на версии .NET Framework 2.0, поддерживаемой этими платформами. Система Visual Studio 2010 позволяет получить лучшее из двух миров, выполняя обновление только тогда, когда вам это нужно.

45

Настройка сборки приложений

В ЭТОЙ ГЛАВЕ...

- Настройка среды
- Действия в начале и в конце сборки
- Создание пользовательских сценариев MSBuild

Хотя большинство проектов можно создать с помощью настроек компиляции, заданных по умолчанию, иногда для достижения конкретных целей может потребоваться модификация некоторых аспектов процесса построения приложения. В этой главе рассматриваются настройки построения приложений на языках Visual Basic и C# и способы их использования для достижения своих целей.

Кроме того, вы узнаете, как система Visual Studio 2010 использует процессор MSBuild для выполнения компиляции и как с помощью конфигурационных файлов можно управлять компиляцией своих проектов.

Общие настройки

Еще до того, как начать работу над проектом, вы можете изменить некоторые настройки на странице Options системы Visual Studio 2010. Они применимы ко всем проектам и решениям, открываемым в интегрированной среде разработки, и как таковые могут использоваться для настройки компиляции ваших проектов.

Первое место, которое следует посетить профессиональным программистам на языке Visual Basic, – страница General группы Projects and Solutions. По умолчанию настройки интегрированной среды разработки Visual Basic скрывают от просмотра

некоторые из глобальных настроек, так что единственный способ их увидеть — активизировать опцию Show Advanced Build Configurations.

Когда это сделано, на страницах My Project интегрированная среда разработки выводит настройки Build Configuration. Кроме того, становится доступной команда меню Build⇨Configuration Manager. В средах других языков это делать не нужно, так как соответствующие настройки активизированы изначально (хотя вы, конечно же, можете их отключить, если не хотите, чтобы они засоряли ваши меню и страницы).

Две другие настройки на этой странице относятся к построению проектов. Одна из них позволяет Visual Studio при запуске сборки автоматически показывать окно Output, а вторая автоматически выводит окно Error, если в процессе сборки возникают ошибки. По умолчанию в конфигурациях всех языков эти настройки включены.

Страница настроек Build and Run (показанная на рис. 45.1) из группы Projects and Solutions содержит гораздо больше доступных опций, позволяющих настроить процесс построения приложений.

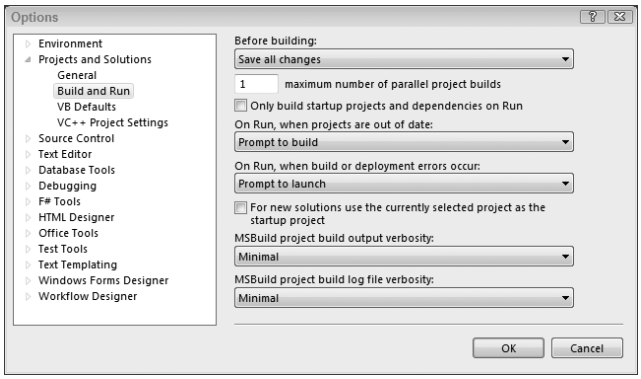


Рис. 45.1

На показанной странице этого не видно, но некоторые из настроек применимы только к проектам на языке C#, так что имеет смысл просмотреть все настройки, чтобы знать, для чего они предназначены и на какие языки действуют.

- Before Building. Сообщает системе Visual Studio, как обрабатываются изменения, внесенные в любую часть проекта до процесса сборки. Имеется четыре варианта.
  - Save All Changes. Автоматически сохраняет все изменения, не спрашивая вашего разрешения. Пожалуй, это наилучший выбор, при котором вы никогда не забудете сохранить свою работу. Это настройка по умолчанию.
  - Save Changes to Open Documents Only. Также выполняет автоматическое сохранение, но только открытых документов. Тем самым исключаются некоторые изменения в файлах решений и проектов.
  - Prompt to Save All Changes. Позволяет сохранить все изменения перед началом сборки. Когда процесс сборки начинается, выводится диалоговое окно с вопросом, следует ли сохранить все внесенные изменения. Если вы откажетесь от сохранения изменений, то сборка продолжится, но будет использоваться последняя сохраненная версия файла. Эта настройка хороша тем, что вам известны сделанные (возможно, нечаянно) изменения в исходном тексте.

- Don't Save Any Changes. Как следует из названия, не сохраняет изменения ни в одном из файлов, как открытых в редакторе, так и всех прочих.

- Maximum Number of Parallel Project Builds.Управляет максимальным количеством одновременно работающих процессов сборки (в предположении, что компилируемое решение состоит из нескольких проектов).

- Only Build Startup Projects and Dependencies on Run. Выполняется сборка только тех частей решения, которые непосредственно связаны со стартовыми проектами. Это означает, что любые проекты, которые не связаны с стартовыми проектами, исключаются из процесса сборки. Эта настройка активна по умолчанию, так что, если у вас имеется решение со многими проектами, вызываемыми стартовыми посредством позднего связывания или иных подобных средств, они не будут собраны автоматически. Вы можете либо деактивизировать эту настройку, либо собирать такие проекты отдельно вручную.

- On Run, When Projects Are Out of Date. Используется только для проектов на языке C++ и предоставляет три варианта для проектов, которые изменились со времени последней сборки. По умолчанию используется выбор Always to Build, который заставляет всегда выполнять сборку при запуске приложения. Вариант Never Build всегда использует предыдущую сборку обновленных проектов, а Prompt to Build позволяет заново собрать каждый обновленный проект. Заметим, что описанная настройка применима только к команде Run, и если вы выполняете сборку с помощью меню Build, то проекты будут собраны повторно в соответствии с другими настройками конфигурации.

- On Run, When Build or Deployment Errors Occur. Управляет действием, предпринимаемым в случае ошибки. Вопреки официальной документации эта настройка влияет на поведение сборки и в среде Visual Basic, и в среде C#. Варианты выбора в случае этой настройки следующие: вариант по умолчанию Prompt to Launch, выводящий диалоговое окно, которое запрашивает вас о том, какие действия следует предпринять; Do Not Launch, который не запускает решение и возвращает вас в режим проектирования; и Launch Old Version, который игнорирует ошибки компиляции и запускает последнюю успешную сборку проекта.

  Вариант запуска старой сборки позволяет проигнорировать ошибки в подчиненных проектах и все равно запустить ваше приложение; однако, поскольку при этом предупреждения о происшедших ошибках не выводятся, есть риск не разобраться, с какой версией проекта вы, собственно, имеете дело.

  Заметим, что при использовании варианта Prompt to Launch, если впоследствии вы установите флажок Do Not Show This Dialog Again в диалоговом окне запроса, эта настройка будет обновлена либо до Do Not Launch, либо до Launch Old Version в зависимости от того, выберете ли вы продолжение запуска.



Рекомендуется установить для этого свойства значение Do Not Launch, так как это может повысить эффективность написания и отладки кода — одним окном будет меньше!

- For New Solutions Use the Currently Selected Project as the Startup Project. Полезна при создании решения с несколькими проектами. Когда собирается решение, процесс сборки в системе Visual Studio полагает, что текущий выбранный проект

является запускаемым (startup) и определяет все зависимости и точку запуска исходя из этого.

- MSBuild Project Build Output Verbosity. Система Visual Studio 2010 использует для компиляции процессор MSBuild, создающий собственный набор выводимых сообщений компиляции, обеспечивающих информацией о состоянии каждого проекта по мере их построения. Указанным выводом можно управлять с помощью следующих опций.
  - По умолчанию количество выводимой процессором MSBuild информации равно Minimal, что приводит к очень небольшому количеству информации о каждом проекте; вывод можно отключить совсем, выбрав опцию Quiet, или, напротив, увеличить, выбрав один из вариантов более детального вывода.
  - Вывод MSBuild направляется в окно Output, доступ к которому можно получить с помощью команды меню View⇨Other Windows⇨Output (при определенных настройках это может быть View⇨Output). Если вы не видите вывода информации о сборке, убедитесь, что в поле Show Output From выбрано значение Build (рис. 45.2).

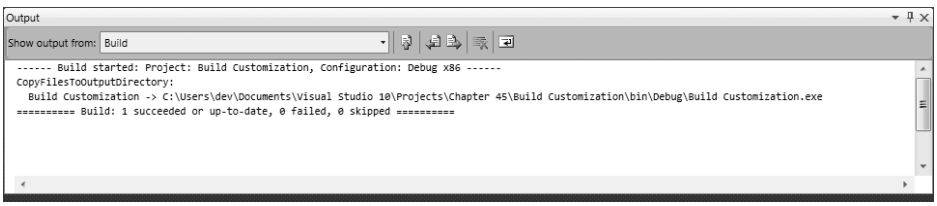


Рис. 45.2

- MSBuild Project Log File Verbosity. Система Visual Studio, собирая проект на языке C++, генерирует текстовый журнальный файл с информацией о действиях процессора MSBuild, а также со всеми сообщениями, поступающими в окно Output. Количество информации, поступающей в этот текстовый файл, может независимо управляться с помощью этой опции. Так, вы можете воспользоваться предоставляемой возможностью и вывести более подробную информацию в журнальный файл, ограничившись минимальным набором сообщений в окне Output (если что-то пойдет не так, вы всегда сможете обратиться за подробной информацией к журнальному файлу). Если такой отдельный файл вам не нужен, отключите его, воспользовавшись командой Projects and Solutions⇨VC++ Project Settings⇨Build Logging.

Стоит взглянуть и на другие страницы в категории Projects and Solutions, поскольку они управляют настройками компилятора Visual Basic по умолчанию (Option Explicit, Option Strict, Option Compare и Option Infer), а также настройками сборки, специфичными для C++. Примечание для разработчиков на C++: имеется возможность указать переменные среды PATH для компонентов разных типов (таких, как выполняемые или заголовочные файлы) для сборок для разных платформ, а также следует ли использовать журнальный файл для информации, выводимой в процессе сборки.

Ручная настройка зависимостей

Система Visual Studio 2010 в состоянии обнаружить зависимости между проектами, которые ссылаются друг на друга. Эти зависимости затем используются для определения порядка сборки проектов. К сожалению, при некоторых условиях система Visual Studio не в состоянии определить эти зависимости, в частности если у вас в процессе сборки имеются пользовательские этапы. К счастью, можно вручную определить зависимости между проектами, и указать, как они связаны между собой. Вы можете воспользоваться диалоговым окном, показанным на рис. 45.3, которое вызывается с помощью команды меню Project⇨Project Dependencies или Project⇨Build Order.

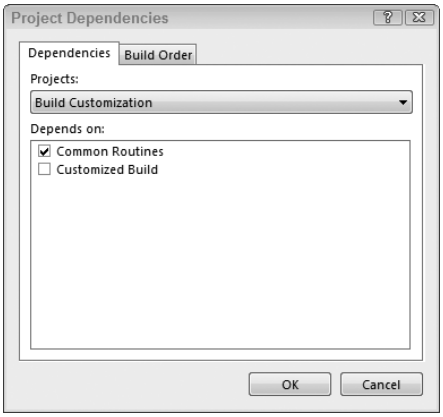


Рис. 45.3.



Обратите внимание, что данные команды меню доступны, только когда в интегрированной среде разработки открыто решение, состоящее из нескольких проектов.

Вначале из раскрывающегося списка выбирается проект, который зависит от других проектов, а затем в списке в нижней части диалогового окна отмечаются те проекты, от которых он зависит. Все зависимости, автоматически обнаруженные системой Visual Studio 2010, появляются в этом списке уже отмеченными. Вкладка Build Order может использоваться для подтверждения порядка сборки проектов.

Страница Compile в среде Visual Basic

Проекты Visual Basic имеют дополнительный набор настроек, управляющих тем, как будет протекать процесс сборки. Для доступа к настройкам компиляции откройте My Project, дважды щелкнув на его записи в Solution Explorer. Когда будет выведена страница Options, перейдите на вкладку Compile, выбрав ее в левой части диалогового окна, как показано на рис. 45.4.

Поле Build Output Path указывает, где будет храниться выполняемая версия (приложение или DLL) вашего проекта. В среде Visual Basic значением по умолчанию является каталог `bin\Debug\` или `bin\Release\` (в зависимости от текущей конфигурации), но вы можете изменить его, перейдя в нужное вам место.



Рекомендуется включить настройку Treat All Warnings as Errors, которая в большинстве случаев заставит вас писать лучший, менее подверженный ошибкам код.

Следует предупредить вас о двух дополнительных множествах скрытых настроек. Кнопка Build Events в правом нижнем углу доступна для разработчиков Visual Basic, которые хотят запускать действия или сценарии перед сборкой или после нее (они будут вскоре рассмотрены). Вторая кнопка называется Advanced Compile Options.

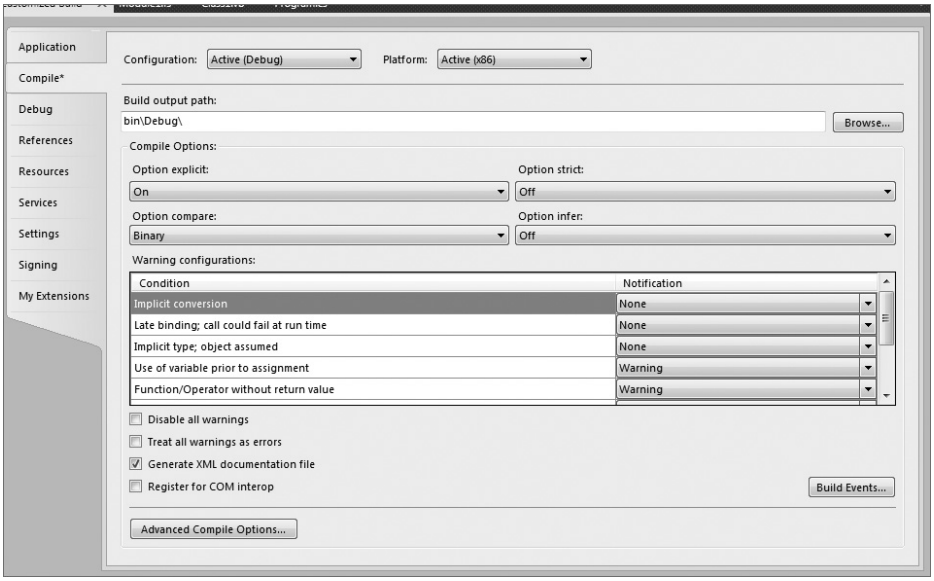


Рис. 45.4.

Окно Advanced Compiler Settings

После щелчка на кнопке Advanced Compile Options выводится диалоговое окно Advanced Compiler Settings, показанное на рис. 45.5, в котором можно выполнить тонкую настройку процесса сборки выбранного проекта. Настройки разделены на две большие группы: Optimizations и Compilation Constants.

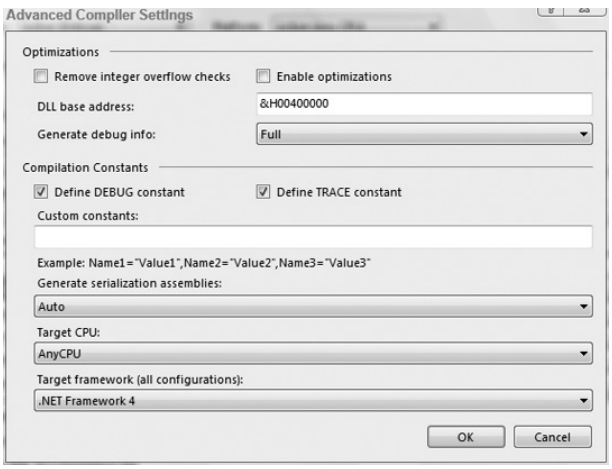


Рис. 45.5.

Группа Optimizations

Настройки в группе Optimizations управляют выполнением компиляции, чтобы процесс сборки прошел быстрее или чтобы минимизировать размер выходных фай-

лов. Обычно эти настройки не изменяют, но если вам необходимо уточнить процесс компиляции, можете воспользоваться такими опциями.

- Remove Integer Overflow Checks. По умолчанию ваш код проверяется на наличие переполнения целых чисел, что потенциально может вызвать утечку памяти. Отключение этой опции приводит к устранению таких проверок и делает код быстрее за счет снижения его безопасности.
- Enable Optimizations. Оптимизация может дать более быстрый код, обычно за счет увеличения размера файлов и времени сборки.
- DLL Base Address. Позволяет указать базовый адрес DLL в шестнадцатеричном формате. Если проект не создает DLL, данная настройка недоступна.
- Generate Debug Info. Управляет генерацией отладочной информации в выходных файлах. По умолчанию данная опция в отладочной конфигурации генерирует отладочную информацию, которая позволяет подключать отладчик к работающему приложению. Вы можете отключить отладочную информацию полностью или сгенерировать `.pdb`-файлы; последний выбор означает, что вы сможете отлаживать приложение при запуске из Visual Studio 2010, но при попытке подключиться к работающему приложению вы сможете увидеть только дизассемблированный код.

Константы компиляции

Константы компиляции могут использоваться для управления тем, какая информация включается в выходные файлы и даже какой код будет компилироваться. Настройка Compilation Constants имеет следующие значения.

- Define DEBUG Constant and Define TRACE Constant. На основе флагов `DEBUG` и `TRACE` включает в скомпилированное приложение соответственно отладочную информацию и информацию трассировки.
- Custom Constants. Если для сборки вашего приложения необходимы пользовательские константы, можете указать их здесь в виде `ConstantName="Value"`. Если у вас имеется несколько констант, они разделяются запятыми.

Три последние опции не являются константами компиляции, но допускают дальнейшую настройку сборки вашего проекта.

- Generate Serialization Assemblies. По умолчанию имеет значение Auto, что позволяет процессу сборки самому определить, требуются ли сборки сериализации, но при желании его можно заменить значением On или Off.



Сериализация создается с помощью утилиты командной строки `Sgen.exe`. Этот инструмент генерирует сборку, содержащую `XmlSerializer` для сериализации (и десериализации) определенного типа. Обычно эти сборки генерируются во время выполнения при первом применении `XmlSerializer`. Предварительная генерация во время компиляции может повысить производительность приложения при первом запуске. Сборки сериализации имеют имена вида *`TypeName`*`.XmlSerializers.dll`. Дополнительную информацию по этому вопросу можно найти в документации по `Sgen.exe`.

- Target CPU. В зависимости от известных вашей системе типов процессоров эта опция позволяет оптимизировать создаваемое приложение для конкретной платформы. Значение по умолчанию – AnyCPU; оно создает приложение, способное работать на любом процессоре, поддерживающем .NET Framework.
- Target Framework. Это единственная настройка в этом диалоговом окне, которая применима ко всем конфигурациям и используется для определения того, какая версия библиотек базовых классов применяется при компиляции.

События сборки



Рис. 45.6

Вы можете выполнять дополнительные действия до или после процесса сборки, добавляя их в список событий. Щелкните на кнопке Build Events на странице My Project Compile для вывода диалогового окна Build Events. На рис. 45.6 показано событие после сборки, которое запускает собранное приложение после каждого успешного построения.

Каждое действие, которое вы хотите выполнить, должно располагаться в отдельной строке и может быть добавлено непосредственно либо в текстовое поле Pre-build Event Command Line, либо в Post-build Event Command Line; кроме того, вы можете воспользоваться кнопками Edit Pre-build и Edit Post-build для доступа к предопределенным синонимам, которые могут использоваться в действиях.



Если ваши события сборки представляют собой пакетные файлы, предваряйте их инструкцией `call`. Например, чтобы вызывать `archive_previous_build.bat` перед каждой сборкой, в поле Pre-build Event Command Line введите текст `call archive_previous_build.bat`. Кроме того, любой путь, содержащий пробелы, должен быть заключен в двойные кавычки. Это справедливо даже тогда, когда источником пути с пробелами является встроенный макрос.

Показанное на рис. 45.7 диалоговое окно Event Command Line включает список макросов, которые можно применять при создании событий. Для каждого макроса выводится его текущее значение, так что вы можете увидеть, какой текст будет включен в вашу командную строку при использовании того или иного макроса.

В приведенном примере разработчик создал командную строку `$(TargetDir)$(TargetFileName)$(TargetExt)`, полагая, что она будет запускать приложение по окончании сборки. Однако, проанализировав значения макросов, нетрудно заметить, что расширение будет включено дважды, что легко исправить, либо просто удалив макрос `$(TargetExt)`, либо заменив все выражение одним макросом `$(TargetPath)`.

В нижней части диалогового окна Build Events имеется возможность указать, при каких условиях должно выполняться действие Post Build Event. Допустимы следующие значения.

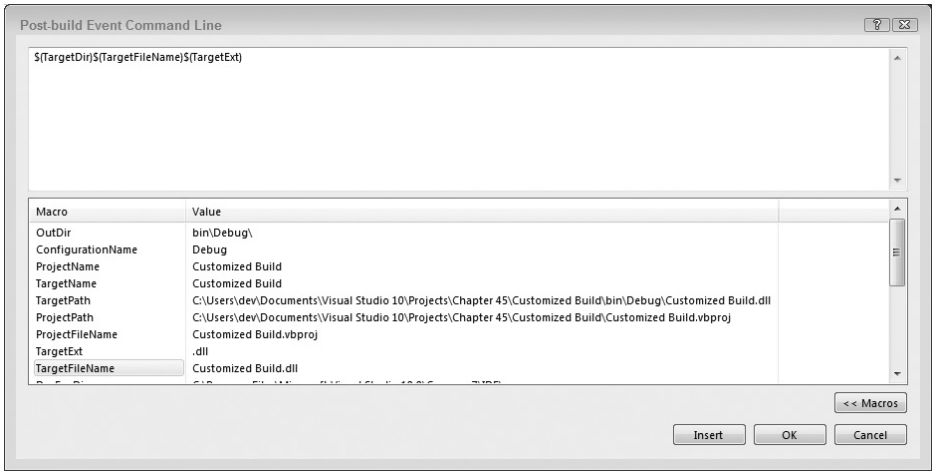


Рис. 45.7

- Always. После выбора этого значения сценарий Post Build Event будет запущен даже в том случае, если сборка завершилась неудачно. Не забывайте, что нет никакой гарантии, что Visual Studio создаст какие-либо файлы, поэтому ваш сценарий должен уметь обрабатывать такую ситуацию.
- On Successful Build. Это выбор по умолчанию. Сценарий Post Build Event будет запущен только в том случае, если процесс сборки завершится успешно. Это означает, что действие будет выполнено, даже если ваш проект не изменялся (а значит, и сборка заново не выполнялась).
- When the Build Updates the Project Output. Этот выбор очень похож на выбор On Successful Build, но сценарий Post Build Event будет запущен только в том случае, если выходной файл проекта был изменен. Это отличный выбор для поддержки локального кэша архивных сборок проектов, так как он означает, что в архив будут копироваться только те файлы, которые были изменены со времени последней сборки.

Для действий Pre-Build Event нет никаких фильтров, определяющих условия их выполнения.

Страницы Build C#

Среда C# предоставляет собственный набор настроек сборки. В общем случае эти настройки те же, что и доступные для проектов Visual Basic, но расположены они в других местах, поскольку программисты C# более склонны к тонкой настройке программ, чем разработчики на Visual Basic, которых обычно интересует скорость разработки, а не работы приложения.

Вместо одной страницы Compile на страницах свойств проекта у C# имеются страницы Build и Build Events. Страница Build Events работает точно так же, как и диалоговое окно Build Events Visual Basic, так что за информацией о ней обратитесь к предыдущему разделу главы.

Как показано на рис. 45.8, многие настройки на странице Build коррелируют с настройками на странице Compile или в области Advanced Compiler Settings Visual Basic. Некоторые настройки, такие как Define DEBUG Constant и Define TRACE Constant, идентичны их аналогам в среде Visual Basic.

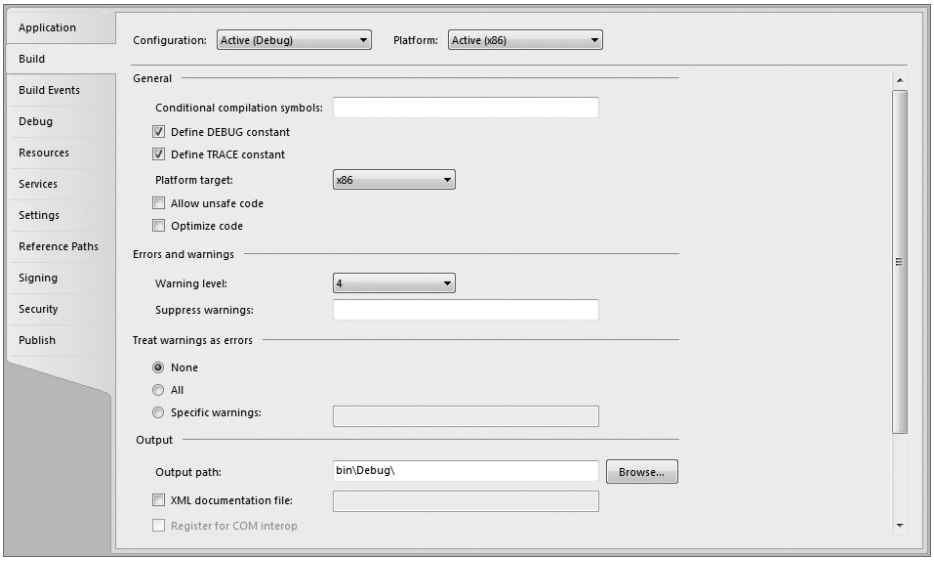


Рис. 45.8

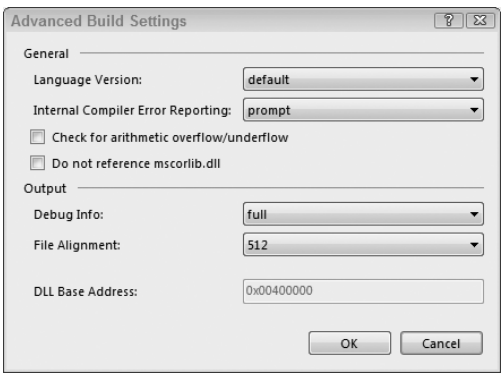


Рис. 45.9

Однако некоторые из настроек переименованы, чтобы отвечать словарю языка программирования C; так, например, настройка Optimize code эквивалентна настройке Enable optimizations. Как и в случае настроек компиляции Visual Basic, вы можете определить, будут ли предупреждения трактоваться, как ошибки, а также указать уровень выдачи предупреждений.

Щелчок на кнопке Advanced на странице Build вызывает показанное на рис. 45.9 диалоговое окно Advanced Build Settings, которое включает настройки, недоступные разработчикам на Visual Basic. Эти настройки позволяют управлять процессом сборки, включая информацию о внутренних ошибках, происшедших в процессе компиляции, и о том, какие отладочные сведения должны быть сгенерированы.

Функции большинства настроек понятны из их названий, так что далее приводится только краткий перечень их влияния на процесс сборки.

- Language Version. Определяет используемую версию языка программирования C#. По умолчанию применяется текущая версия. В Visual Studio 2010 другими вариантами являются ISO-1 и ISO-2, которые ограничивают возможности языка только определенными в соответствующем стандарте.

- Internal Compiler Error Reporting. Если в процессе компиляции произошла ошибка (не ошибка компиляции в исходном тексте, а ошибка самого процесса компиляции), вы можете отправить информацию о ней специалистам из Microsoft, чтобы они исправили ее в следующей версии компилятора. По умолчанию используется значение Prompt, т.е. будет выведен запрос, нужно ли отправлять информацию в Microsoft.
- Другие значения — None (информация не отправляется), Send (информация отправляется автоматически) и Queue (информация ставится в очередь для отправки позже).
- Check for Arithmetic Overflow/Underflow. Проверка ошибок переполнения, которые могут привести к небезопасной работе приложения. Ошибка потери значимых разрядов возникает, когда точность числа слишком высока для данной системы.
- Do Not Reference mscorlib.dll. По умолчанию ваш проект автоматически ссылается на библиотеку `mscorlib.dll`, которая определяет пространство имен `System`, но вы можете воспользоваться этой настройкой для создания собственного пространства имен `System` и связанных с ним объектов.
- Debug Info. Идентична настройке Visual Basic Generate debug info.
- File Alignment. Используется для установки границ разделов в выходном файле и позволяет управлять внутренним размещением скомпилированного выходного файла. Значения указываются в байтах.
- DLL Base Address. Идентична настройке Visual Basic с тем же именем.

Указанные настройки позволяют более точно управлять процессом сборки. Однако в Visual Studio 2010 есть другой вариант, который непосредственно редактирует сценарии сборки. Это возможно благодаря тому, что Visual Studio 2010 для компиляции применяет MSBuild.

Процессор MSBuild

Система Visual Studio 2010 использует процессор MSBuild, который представляет собой процессор компиляции Microsoft, изначально появившийся в системе Visual Studio 2005. Он использует конфигурационные файлы в формате XML, в которые включены все рассматривавшиеся ранее в этой главе настройки, а также файлы, которые должны быть включены в процесс компиляции.

Фактически система Visual Studio начиная с версии Visual Studio 2005 использует конфигурационные файлы MSBuild в качестве файлов определения проекта вместо старых форматов файлов проектов, использовавшихся в предыдущих версиях Visual Studio. Это позволяет автоматически использовать MSBuild при компиляции приложений в интегрированной среде разработки, поскольку и для определения проекта в интегрированной среде разработки, и для процесса сборки применяются одни и те же файлы.

Как Visual Studio использует процессор MSBuild

Как уже упоминалось, содержимое файлов проектов в системе Visual Studio 2010 основано насхеме MSBuild XML Schema и может редактироваться непосредственно

в Visual Studio. Таким образом, вы можете настраивать загрузку и компиляцию своего проекта.

Однако для редактирования файла проекта требуется убрать активный статус проекта в окне Solution Explorer. Щелкните правой кнопкой мыши в окне Solution Explorer на проекте, который вы хотите отредактировать, и выберите из контекстного меню команду Unload Project.

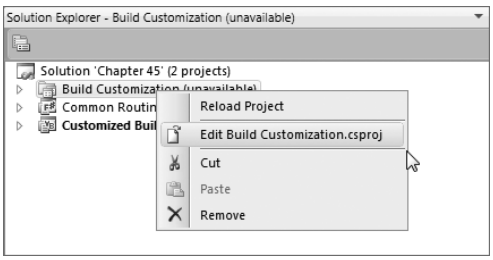


Рис. 45.10

Проект в окне Solution Explorer будет свернут и помечен как недоступный. Кроме того, любые открытые файлы, которые принадлежат проекту, будут при этом закрыты. Щелкните правой кнопкой мыши на проекте еще раз, и вы увидите в контекстном меню команду для редактирования файла проекта (рис. 45.10).

После ее выбора в XML-редакторе системы Visual Studio 2010 будет открыт XML-файл проекта. Приведенный далее листинг содержит пример файла проекта MSBuild для пустого проекта C#:



```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" DefaultTargets="Build"
  xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Configuration Condition="'$(Configuration)' == ''">
      Debug
    </Configuration>
    <Platform Condition="'$(Platform)' == ''">x86</Platform>
    <ProductVersion>8.0.30703</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{04ABE6E2-5500-467B-BB01-0BBF0258E94A}
    </ProjectGuid>
    <OutputType>Exe</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>ConsoleApplication</RootNamespace>
    <AssemblyName>ConsoleApplication</AssemblyName>
    <TargetFrameworkVersion>v4.0</TargetFrameworkVersion>
    <TargetFrameworkProfile>Client</TargetFrameworkProfile>
    <FileAlignment>512</FileAlignment>
  </PropertyGroup>
  <PropertyGroup
    Condition="'$(Configuration)|$(Platform)' == 'Debug|x86'">
    <PlatformTarget>x86</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup
    Condition="'$(Configuration)|$(Platform)' == 'Release|x86'">
    <PlatformTarget>x86</PlatformTarget>
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
```

```
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="System"/>
    <Reference Include="System.Core"/>
    <Reference Include="System.Xml.Linq"/>
    <Reference Include="System.Data.DataSetExtensions"/>
    <Reference Include="Microsoft.CSharp"/>
    <Reference Include="System.Data"/>
    <Reference Include="System.Xml"/>
  </ItemGroup>
  <ItemGroup>
    <Compile Include="Program.cs"/>
    <Compile Include="Properties\AssemblyInfo.cs"/>
  </ItemGroup>
  <Import
    Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets"/>
  <!-- To modify your build process, add your task inside one
       of the targets below and uncomment it. Other similar
       extension points exist, see Microsoft.Common.targets.
    <Target Name="BeforeBuild">
    </Target>
    <Target Name="AfterBuild">
    </Target>
  -->
</Project>
```

ConsoleApplication\ConsoleApplication.csproj

XML-текст содержит информацию о сборке. Большинство узлов в приведенном исходном тексте непосредственно связаны с настройками, которые вы видели ранее на страницах Compile и Build, но включают также требующиеся пространства имен .NET Framework. Первый элемент `PropertyGroup` содержит свойства проекта, применимые ко всем конфигурациям сборки. За ним следуют два условных элемента, определяющих свойства для каждой из двух конфигураций сборки — Debug и Release. Остальные элементы служат для ссылок проекта и импорта пространств имен.

Когда проект включает дополнительные файлы, такие как формы и пользовательские управляющие элементы, каждый из них определяется в файле проекта со своим набором узлов. Например, приведенный далее исходный текст демонстрирует дополнительный XML-код, включенный в стандартный проект Windows Application. Этот код идентифицирует форму, ее файл кода проектировщика и дополнительные файлы приложения, необходимые для Windows-приложения.



```
<ItemGroup>
  <Compile Include="Form1.cs">
    <SubType>Form</SubType>
  </Compile>
  <Compile Include="Form1.Designer.cs">
    <DependentUpon>Form1.cs</DependentUpon>
  </Compile>
  <Compile Include="Program.cs"/>
  <Compile Include="Properties\AssemblyInfo.cs"/>
  <EmbeddedResource Include="Properties\Resources.resx">
    <Generator>ResXFileCodeGenerator</Generator>
    <LastGenOutput>Resources.Designer.cs</LastGenOutput>
    <SubType>Designer</SubType>
  </EmbeddedResource>
  <Compile Include="Properties\Resources.Designer.cs">
    <AutoGen>True</AutoGen>
```

```
      <DependentUpon>Resources.resx</DependentUpon>
    </Compile>
    <None Include="Properties\Settings.settings">
      <Generator>SettingsSingleFileGenerator</Generator>
      <LastGenOutput>Settings.Designer.cs</LastGenOutput>
    </None>
    <Compile Include="Properties\Settings.Designer.cs">
      <AutoGen>True</AutoGen>
      <DependentUpon>Settings.settings</DependentUpon>
      <DesignTimeSharedInput>True</DesignTimeSharedInput>
    </Compile>
  </ItemGroup>
```

WindowsFormsApplication\WindowsFormsApplication.csproj

В процесс сборки можно добавить дополнительные задачи, включив их в узлы `Target` с атрибутами `Name` со значениями `BeforeBuild` и `AfterBuild`. Однако эти действия не появятся в диалоговом окне Build Events Visual Studio 2010, рассмотренном ранее. Альтернативный путь состоит в использовании узла `PropertyGroup`, который включает записи `PreBuildEvent` и `PostBuildEvent`. Например, чтобы запустить приложение после успешной сборки, добавьте приведенный ниже XML-блок непосредственно перед закрывающим дескриптором `</Project>`:

```
<PropertyGroup>
  <PostBuildEvent>"$(TargetDir)$(TargetFileName)"
  </PostBuildEvent>
</PropertyGroup>
```

По окончании редактирования XML-файл проекта необходимо заново включить, щелкнув правой кнопкой мыши в окне Solution Explorer и выбрав в контекстном меню команду Reload Project. Если ваш файл проекта все еще открыт, система Visual Studio попросит вас сначала его закрыть.

Схема MSBuild Schema

Подробное рассмотрение процессора MSBuild выходит за рамки данной книги. Однако желательно понимать, что делают разные компоненты, составляющие файл проекта MSBuild, чтобы иметь возможность просматривать и обновлять свои проекты.

Основу файла проекта образуют четыре главных типа: элементы, свойства, цели и задачи. Вы можете совместно использовать их для создания конфигурационного файла, который описывает проект целиком, как показано в предыдущем примере файла проекта C#.

Элементы

Элементы — это узлы, которые определяют входные данные для сборки и проекта. Они определены как узлы, дочерние по отношению к узлу `ItemGroup`, и наиболее распространенным элементом является узел `Compile`. Он сообщает процессору MSBuild, что некоторый файл должен быть включен в компиляцию. Приведенный далее фрагмент кода файла проекта демонстрирует элемент, определенный для файла `Form1.cs` приложения Windows:

```
<ItemGroup>
    <Compile Include="Form1.cs">
        <SubType>Form</SubType>
    </Compile>
</ItemGroup>
```

Свойства

Узлы PropertyGroup используются для хранения всех свойств, определенных для данного проекта. Свойства обычно представляют собой пары "ключ–значение". Они могут содержать единственное значение и использоваться для хранения настроек проекта, к которым вы можете обратиться на страницах Build и Compile в интегрированной среде разработки.

Узлы PropertyGroup могут содержать необязательный атрибут Condition, как показано в приведенном далее фрагменте кода:

```
<PropertyGroup
  Condition="'$(Configuration)|$(Platform)' == 'Release|x86'">
  <DebugType>pdbonly</DebugType>
  <Optimize>true</Optimize>
  <OutputPath>bin\Release\</OutputPath>
  <DefineConstants>TRACE</DefineConstants>
  <ErrorReport>prompt</ErrorReport>
  <WarningLevel>4</WarningLevel>
</PropertyGroup>
```

В этом фрагменте определен узел PropertyGroup, который будет включен в сборку, если проект создается как Release для платформы x86. Каждый из шести узлов в PropertyGroup использует в качестве имени узла имя свойства.

Цели

Элементы Target позволяют упорядочить задачи (будут рассмотрены в следующем разделе) последовательно. Каждый элемент Target должен иметь атрибут Name, идентифицирующий его; он может быть вызван непосредственно, тем самым обеспечив несколько входных точек в процесс сборки. Приведенный далее фрагмент кода определяет Target с именем BeforeBuild:

```
<Target Name="BeforeBuild">
</Target>
```

Задачи

Элементы Task определяют действия, которые процессор MSBuild будет выполнять при определенных условиях. Вы можете определить свои задачи или воспользоваться множеством встроенных задач, таких как Copy. Как показано в приведенном далее фрагменте кода, Copy может копировать один или более файлов из одного места в другое:

```
<Target Name="CopyFiles">
  <Copy
    SourceFiles="@(MySourceFiles)"
    DestinationFolder="\\PDSERVER01\SourceBackup\"
  />
</Target>
```

Контроль версий с помощью задач MSBuild

Одним из аспектов автоматизированных систем сборки является контроль версий. Из этого раздела вы узнаете, как настроить процесс сборки вашего проекта, чтобы он мог принимать внешний номер версии. Этот номер версии будет использоваться для обновления файла AssemblyInfo, который впоследствии будет влиять на версию сборки. Начнем с рассмотрения файла AssemblyInfo.cs, который обычно содержит информацию о версии сборки, такую как приведенная далее:



```
[Assembly: AssemblyVersion("1.0.0.0")]
```

CustomizedBuild\Properties\AssemblyInfo.cs

Настройка процесса сборки должна заменять номер версии по умолчанию номером, передаваемым как часть процесса. Для этого мы воспользуемся библиотекой MSBuild стороннего производителя — MSBuildTasks (проект Tigris, `http://msbuildtasks.tigris.org/`). Она включает задачу `FileUpdate`, которая может быть использована для проверки соответствия регулярному выражению. Прежде чем воспользоваться этой задачей, необходимо импортировать файл Targets библиотеки MSBuildTasks. Этот файл по умолчанию инсталлируется в каталог расширений MSBuild:



```
<Project ToolsVersion="4.0" DefaultTargets="Build"
xmlsn="http://schemas.microsoft.com/developer/msbuild/2003">
  <!-- Импорт для юзания MSBuild Community Tasks -->
<Import Project="$(MSBuildExtensionsPath)\MSBuildCommunityTasks\
MSBuild.Community.Tasks.Targets"/>
  <PropertyGroup>
    ...
```

CustomizedBuild\CustomizedBuild.csproj

Поскольку нам нужно обновить файл `AssemblyInfo` до начала сборки, добавим вызов задачи `FileUpdate` в цель `BeforeBuild`. Правда, это усложнит дальнейшую поддержку и отладку. Поэтому гораздо лучше сначала создать цель для задачи `FileUpdate`, а затем сделать цель `BeforeBuild` зависящей от нее:



```
<Import
    Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
<Target Name="BeforeBuild"
        DependsOnTargets="UpdateAssemblyInfo">
</Target>

<Target Name="UpdateAssemblyInfo">
  <Message Text="Build Version: $(BuildVersion)" />
  <FileUpdate Files="Properties\AssemblyInfo.cs"
              Regex="\d+\.\d+\.\d+\.\d+"
              ReplacementText="$(BuildVersion)" />
</Target>
```

CustomizedBuild\CustomizedBuild.csproj

Можно заметить, что мы используем здесь свойство `$(BuildVersion)`, которого пока что не существует. Если сейчас мы запустим MSBuild для этого проекта, он заменит номер версии в нашем файле `AssemblyInfo` пустой строкой. К сожалению, компиляция такого текста не будет выполнена. Мы можем просто определить указанное свойство с некоторым значением по умолчанию наподобие следующего:

```
<PropertyGroup>
    <BuildVersion> 0.0.0.0 </BuildVersion>
    <Configuration Condition="'$(Configuration)' == ' '">
    Debug
    </Configuration>
```

Это сработает, но при сборке проекта в Visual Studio 2010 он будет иметь одну и ту же версию. К счастью, библиотека MSBuildTasks имеет другую задачу, `Version`, которая генерирует для нас номер версии. Вот соответствующий код:



```
<Target Name="BeforeBuild"
        DependsOnTargets="GetVersion;UpdateAssemblyInfo">
</Target>
...
<Target Name="GetVersion" Condition="$(BuildVersion) == ''">
```

```
    <Version BuildType="Automatic" RevisionType="Automatic"
             Major="1" Minor="3">
      <Output TaskParameter="Major" PropertyName="Major" />
      <Output TaskParameter="Minor" PropertyName="Minor" />
      <Output TaskParameter="Build" PropertyName="Build" />
      <Output TaskParameter="Revision" PropertyName="Revision" />
    </Version>
    <CreateProperty Value="$(Major).$(Minor).$(Build).$(Revision)">
      <Output TaskParameter="Value" PropertyName="BuildVersion" />
    </CreateProperty>
  </Target>
```

CustomizedBuild\CustomizedBuild.csproj

Новая цель `GetVersion` будет выполняться только в том случае, когда свойство `$(BuildVersion)` не указано. Она вызывает задачу `Version` из библиотеки MSBuild-Tasks, которая устанавливает старший номер версии равным 1, а младший — 3 (конечно, вы можете настраивать эти параметры, а не жестко их кодировать). Числа `Build` и `Revision` автоматически генерируются в соответствии с простым алгоритмом. Эти компоненты версий собираются вместе в задаче MSBuild `CreateProperty`, создавая полное свойство `$(BuildVersion)`, требующееся нам. Наконец, эта задача добавляется в список целей, от которых зависит цель `BeforeBuild`.

Теперь при сборке проекта в Visual Studio 2010 мы получим автоматически генерируемый номер. В вашем автоматизированном процессе сборки можно указать номер версии в качестве аргумента вызова MSBuild:

```
MSBuild CustomizedBuild.csproj /p:BuildVersion=2.4.3154.9001
```

Резюме

Процесс сборки по умолчанию может быть настроен с помощью огромного количества настроек системы Visual Studio 2010 благодаря мощи и гибкости процессора MSBuild. В файл проекта можно включить дополнительные действия, выполняемые до и после самого процесса сборки, а также дополнительные файлы.

46

Контроль версий и подписи сборок

В ЭТОЙ ГЛАВЕ...

- Контроль версий сборок
- Подпись и строгое имя сборки
- Управление именованными сборками в глобальном кэше сборок

Создавая сборку .NET, вы можете подписать ее для назначения ей строгого имени. Сборка без строгого имени идентифицируется именем файла, чего зачастую недостаточно для ее уникальной идентификации. Это означает, что другие проекты, которые зависят от вашей сборки, не могут гарантированно получить доступ к корректной версии. Именованные же сборки могут быть идентифицированы единственным образом не только зависимыми проектами, но и системными администраторами, которые могут применять к вашей сборке политики безопасности.

Из этой главы вы узнаете, как использовать систему Visual Studio 2010 для установки номера версии сборки и применять цифровую подпись для того, чтобы подписать свою сборку так, чтобы ее никто не смог изменить. Кроме того, в результате вы получите строго именованную сборку, которая может быть добавлена в глобальный кэш сборок (Global Assembly Cache).

Именование сборок

Каждая сборка .NET, будь то выполняемый файл или библиотека классов, содержит манифест с информацией о сборке. В первую очередь, это имя и номер версии сборки, но сюда включаются также культура и открытый ключ, если это строго именованная сборка. Эту информацию легко просмотреть, открыв сборку в программе Red Gate's .NET Reflector, как показано на рис. 46.1.

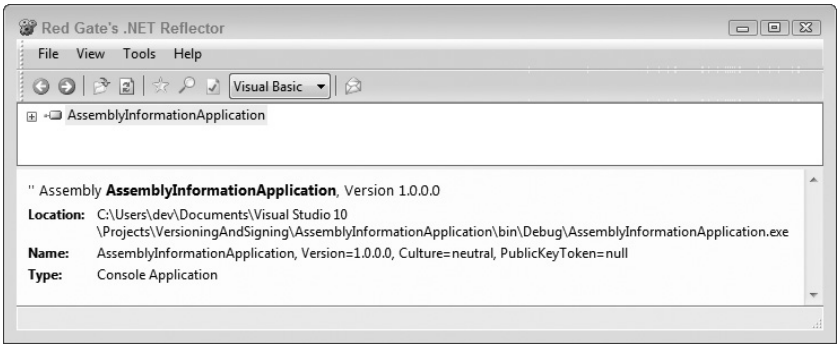


Рис. 46.1

На рис. 46.1 сборка `AssemblyInformationApplication.exe` не имеет открытого ключа. Другие сборки, такие как `System.Data`, имеют полное имя вида

```
System.Data, Version=2.0.0.0, Culture=neutral, PublicKeytoken=b77a5c561934e089
```

Вы указываете имя своей сборки в системе Visual Studio 2010 через редактор свойств проекта, как показано на рис. 46.2. На этом рисунке можно видеть поле Assembly Name на вкладке Application, а также поля Assembly Version, доступные после щелчка на кнопке Assembly Information.

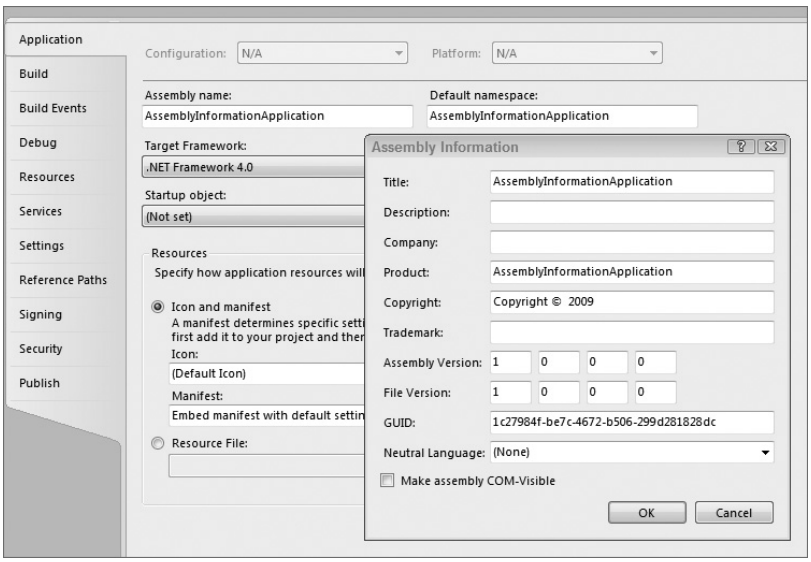


Рис. 46.2

Свойства сборки, представленные в диалоговом окне на рис. 46.2, содержатся в файле `AssemblyInfo`, добавляемом к проекту по умолчанию.



Если вы используете проект C#, то можете обнаружить файл `AssemblyInfo.cs`, развернув элемент Properties проекта в Solution Explorer. Если вы работаете в среде VB, то можете найти файл `AssemblyInfo.vb` в окне My Project (если установите флажок проекта Show All Files).

Приведенный далее фрагмент кода иллюстрирует атрибуты сборок `AssemblyVersion` и `AssemblyFileVersion`, которые используются для определения версии сборки и версии файла сборки:

VB

```
' Информация о версии сборки состоит из четырех частей:
'
' Major Version
' Minor Version
' Build Number
' Revision
'
' Можно указать все значения либо использовать значения
' Build и Revision по умолчанию, заменив их '*':
' <Assembly: AssemblyVersion("1.0.*")>
<Assembly: AssemblyVersion("1.0.0.0")>
<Assembly: AssemblyFileVersion("1.0.0.0")>
```

C#

```
// Поясняющие комментарии удалены
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Если вы не понимаете, в чем разница между версией и версией файла, то примите к сведению, что все зависит от их использования. Информация о версии сборки применяется платформой .NET Framework при разрешении информации о сборках и типах. Версия же файла — это то, что выводит программа Windows Explorer при просмотре свойств файла.



Постоянно обсуждается вопрос, должны ли совпадать версия сборки и версия файла, но окончательный ответ — за вами. Некоторые разработчики предпочитают их синхронизировать, поскольку это позволяет увидеть версию сборки в окне Windows Explorer. С другой стороны, отдельные организации используют версию файла для представления изменений в сборке (например, небольшой рефакторинг или текущие исправления), в то время как версия сборки применяется для указания новых версий приложения.

Как поясняется в комментариях к фрагменту кода VB, версия сборки состоит из четырех компонентов — Major, Minor, Build и Revision. Вопрос о том, как вы будете увеличивать значения этих компонентов, целиком в вашей власти. Фактически в Visual Studio 2010 вы можете просто выбрать символ * для полей Build и/или Revision. Одна широко распространенная стратегия заключается в том, что значения Major и Minor используются для представления версии продукта, над которой вы работаете. Увеличение только значения Minor при этом представляет минимальные исправления и новую функциональность (аналог пакета обновлений), в то время как значение Major представляет новую базовую функциональность.

Значения же Build и Revision могут использоваться в связи с процессом сборки. Например, номер Build может представлять номер недели при разработке текущего выпуска, а номер Revision — последний номер обновления в хранилище исходных текстов. Это последнее значение в таком случае становится весьма важным, поскольку его можно использовать отдельно для получения из хранилища исходного текста, использовавшегося для сборки определенной версии.

Согласованность версий

Конфигурация проекта по умолчанию не годится для поддержания согласованных номеров версий всех проектов в решении. Однако, используя возможность включения в проект связываемых файлов, вы можете заставить систему Visual Studio 2010 предоставлять согласованные номера версий. Это особенно важно, если вы применяете автоматизированную систему сборки, которая автоматически увеличивает номер версии. Вместо того чтобы обновлять каждый номер в своем файле `AssemblyInfo`, можно просто модифицировать единственный файл, и при этом версии всех проектов будут обновлены.

Начать следует с создания в папке решения дополнительного файла `AssemblyInfo`, скажем — `GlobalAssemblyInfo.vb`. Для этого щелкните правой кнопкой мыши на узле решения и выберите в контекстном меню пункт Add New Item. Новый элемент будет добавлен в папку вашего решения. В этот файл необходимо перенести атрибуты `AssemblyVersion` и `AssemblyFileVersion` из файла `AssemblyInfo` ваших проектов (кроме того, требуется импортировать пространство имен `System.Reflection`, если только вы не используете полностью квалифицированные имена атрибутов).

Сделав все это, добавьте вновь созданный файл в каждый из своих проектов. Это делается с помощью команды Add Existing Item контекстного меню, вызываемого после щелчка правой кнопкой мыши на проекте в инструментальном окне Solution Explorer. Когда вы указываете местоположение файла `GlobalAssemblyInfo.vb` или `GlobalAssemblyInfo.cs`, убедитесь, что вы выбрали в раскрывающемся списке Add пункт Add As Link, как показано на рис. 46.3.

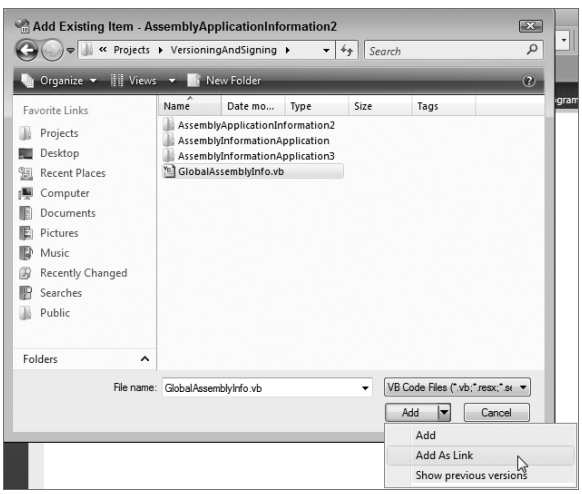


Рис. 46.3

Этот файл `GlobalAssemblyInfo` может применяться в любом количестве проектов; единственным ограничением является то, что он специфичен для используемого языка программирования — VB или C#. Если у вас имеется решение с проектами как на VB, так и на C#, вам нужен файл `GlobalAssemblyInfo` для каждого из языков. Но это все равно лучше, чем поддерживать отдельный файл с информацией о версии для каждого проекта. Заметим, что вы можете включать и другие атрибуты сборок в эти центральные файлы, например такие, как `AssemblyCopyright`, `AssemblyCompany` и `AssemblyTrademark`.

Строго именованные сборки

Строгое, или уникальное, имя (strong name) состоит из частей, которые однозначно идентифицируют сборку. В него входит обычное текстовое имя и номер версии. К этим элементам добавляются открытый ключ (public key) и цифровая подпись. Они генерируются с применением соответствующего закрытого ключа. Благодаря

системе открытого и закрытого ключей и цифровой подписи, рассматриваемые имена являются совершенно уникальными. Подписав сборки, вы предохраняете их от злоумышленного внесения изменений в их код. Сборки .NET относительно просты для анализа, восстановления исходного текста, модификации и перекомпиляции. Хеш-значение, создаваемое как часть процесса подписания, изменяется при модифицировании сборки и тем самым предоставляет механизм защиты от неавторизованных модификаций.

Использование строгих имен может также гарантировать корректность версии поставляемой сборки. Никакие модификации не могут быть сделаны без влияния на сигнатуру, что тут же приводит к несовместимости со сгенерированным строгим именем.

Как упоминалось ранее, применение строгих имен дает администратору возможность явно устанавливать политики безопасности для ваших решений, используя их строгие имена. Это дает уверенность в том, что будучи установленным, программное обеспечение будет работать так, как ожидается, поскольку его нельзя изменить без воздействия на подпись строгого имени.



После того как вы начали применять строго именованные сборки в своем решении, обязательно используйте строго именованные файлы во всей цепочке ссылок, так как, допустив неподписанную сборку как часть цепочки, вы тем самым разрушаете систему безопасности, для реализации которой и предназначено строгое именование сборок.

Глобальный кэш сборок

Каждый компьютер, на котором установлена платформа .NET Framework, имеет системный кэш, именуемый глобальным кэшем сборок (Global Assembly Cache — GAC), который может применяться для хранения сборок, совместно используемых несколькими приложениями. Сборки, добавленные в кэш GAC, доступны из любого приложения .NET в той же системе. Это может существенно облегчить жизнь в организациях, где имеется общая функциональность, которую вы хотите позволить совместно использовать многим приложениям.

В этом кэше (который обычно хранится в папке, в каталоге `Windows`) находятся общие компоненты времени выполнения языка программирования, а также другие глобально зарегистрированные бинарные файлы, которыми вы можете воспользоваться, как и любой другой. Если сборка предназначена для применения единственным приложением, она должна размещаться в папке приложения.



Важно отметить, что добавлять сборки в GAC не рекомендуется, кроме тех случаев, когда вам действительно нужно обеспечить их совместное использование приложениями и они слишком велики для того, чтобы поставлять их с каждым приложением отдельно.

Если вы решите предоставить сборку в совместное пользование приложениям, вам надо знать, как сохранить ее в кэше GAC. Ваша сборка должна быть строго именованной. В этом плане у вас нет выбора, так как кэш проверяет целостность всех хранящихся в нем файлов. Как именовать свои сборки, вы узнаете позже в данной главе.

Если у вас имеется строго именованная сборка, можете добавить ее в GAC с помощью утилиты командной строки `gacutil.exe`:

```
gacutil.exe /i AssemblyInformationApplication.dll
```

Если сборка с таким именем уже существует, можете ее переустановить, воспользовавшись аргументом командной строки `/f`. Для деинсталляции сборки используется команда

```
gacutil.exe /u AssemblyInformationApplication
```



`Gacutil.exe` является частью пакета Microsoft .NET Framework Software Developer Kit (SDK) и не входит в стандартную поставку. Это означает, что полагаться на наличие этой утилиты можно только при наличии сред разработки. Для добавления в кэш GAC на клиентских машинах следует использовать MSI-файл. Более подробно данный вопрос рассматривается в главе 48, "Упаковка и развертывание".

Подписание сборок

Ранее для подписания сборок в Visual Studio требовалась генерация файла ключа строгого имени (`.snk`) с помощью внешней утилиты с последующим редактированием атрибутов сборки в конфигурационном файле приложения. К счастью, система Visual Studio обладает встроенной поддержкой подписания всех проектов с управляемым кодом с помощью вкладки Signing в редакторе свойств проекта, как показано на рис. 46.4.

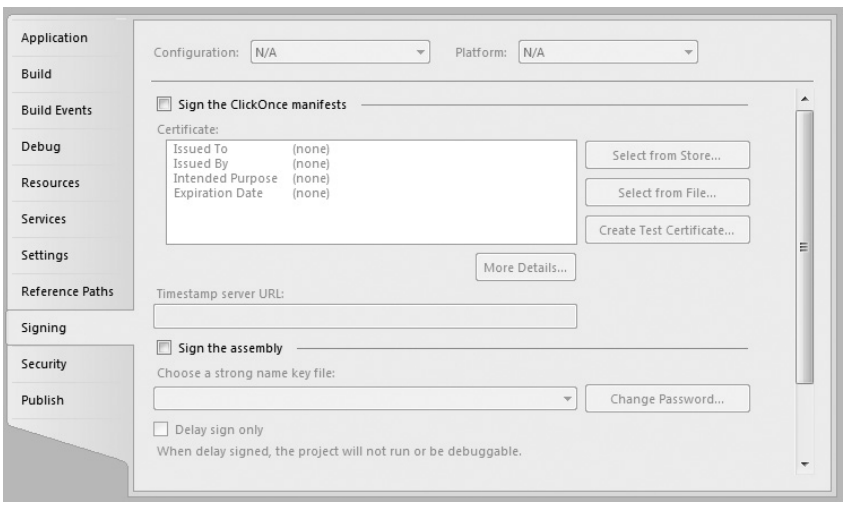


Рис. 46.4

Вкладка Signing в нижней части страницы позволяет подписывать сборку. Сначала необходимо установить флажок Sign the assembly, чтобы указать, что вы будете генерировать строгое имя. Затем следует выбрать файл ключа строгого имени для использования в процессе подписания.

Могут использоваться файлы ключей как в старом формате `.snk`, так и в новом формате `.pfx`. Выберите в раскрывающемся списке Browse для указания местополо-

жения файла в вашей файловой системе и щелкните на кнопке OK в диалоговом окне для сохранения файла ключа вместе с прочими настройками страницы Signing.



Рис. 46.5

Можете также создать файл ключа строгого имени, выбрав из раскрывающегося списка пункт New. После выбора New можете создать файл в формате `.pfx`. На рис. 46.5 показано диалоговое окно Create Strong Name Key. Вы можете просто выбрать имя файла для ключа и защитить его паролем. Если вы решите добавить пароль, то вам будет предложено ввести его при первой сборке приложения на любом другом компьютере. После этого система Visual Studio запомнит указанный пароль.

В любом случае, после того как вы создадите и выберете файл ключа, он будет добавлен в ваш проект в окне Solution Explorer, позволяя вам легко включать его в распространяемые проекты.

Одна из основных причин, по которым вы можете захотеть подписать свои сборки, заключается в защите их от модификации. Поэтому большинство организаций возводят высокие барьеры безопасности вокруг файлов ключей строгих имен, используемых для подписания их сборок. Вряд ли вы захотите разрешить доступ к закрытому ключу для успешного подписания сборок, но и в этой ситуации вам все равно нужно потребовать цифровой подписи приложения. Однако вместо предоставления полного файла ключа строгого имени, который содержит информацию и об открытом, и о закрытом ключах, вы предоставляете только файл с открытым ключом и устанавливаете флажок Delay Sign Only. Позже, скорее всего, в качестве части процесса сборки приложения, вам потребуется подписывать сборки с использованием полного ключа:

```
sn –R AssemblyInformationApplication MyOrganisationsStrongkey.snk
```

Выбрав отложенное подписание своих сборок, вы не сможете отлаживать и даже запускать приложение, поскольку оно не сможет пройти процесс верификации, являющийся частью проверок .NET Framework перед запуском приложения. В действительности это не совсем так, поскольку можно зарегистрировать свою сборку (фактически любую сборку, подписанную тем же открытым ключом), так что этап верификации будет пропущен:

```
sn – Vr AssemblyInformationApplication.exe
```



Регистрируйте сборки для пропуска верификации только на тех машинах, на которых выполняется разработка. В дальнейшем вы сможете дерегистрировать сборку (или все сборки, подписанные тем же открытым ключом) с помощью утилиты `sn` с параметром командной строки `–Vu`.

Резюме

Задача строгого именования сборок и их защита таким образом от некорректного применения в настоящее время решается достаточно просто и может быть реализована полностью в рамках интегрированной среды разработки Visual Studio 2010. Страница Signing предоставляет возможность создавать и назначать файлы ключей без непосредственного редактирования атрибутов сборки приложения.

47

Умышленное запутывание, мониторинг и управление приложениями

В ЭТОЙ ГЛАВЕ...

- Возможности Dotfuscator Software Services — Community Edition, бесплатного инструмента для послесборочного совершенствования и мониторинга приложений, поставляемых с Visual Studio
- Умышленное запутывание кода для предотвращения простой декомпиляции сборок
- Защита приложений от несанкционированных модификаций
- Настройка даты окончания работы приложения, по истечении которой приложение перестанет запускаться
- Настройка отслеживания для определения, какие приложения и возможности используются

Если заглянуть за кулисы и посмотреть, как работают сборки .NET, то можно увидеть, что весь исходный код на платформе .NET компилируется в промежуточный язык Microsoft (Microsoft Intermediary Language – MSIL, или просто IL для краткости), а не в машинный код (независимый от используемого языка программирования). Затем этот промежуточный язык компилируется *по запросу* (just-in-time), когда его нужно выполнить. Такой подход имеет ряд значительных преимуществ, например возможность динамического запроса информации о типах и методах, выполняемого сборкой. Однако это обоюдоострый меч: подобная гибкость означает, что скрытые в коде алгоритмы и логика могут быть легко вскрыты и модифицированы – как законно, так и нет. В этой главе вы познакомитесь с инструментами, которые помогут вам защитить свой исходный текст от любопытных глаз и отследить использование ваших приложений.

Дизассемблер MSIL

Прежде чем перейти к вопросу о защите вашего кода и отслеживании его поведения "на воле", рассмотрим пару инструментов, которые помогут вам создавать лучшие приложения. Первый инструмент — Microsoft .NET Framework IL Disassembler, или IL Dasm. Он включен в пакет Microsoft Windows SDK, который по умолчанию инсталлируется одновременно с Visual Studio 2010. Вы можете найти его в меню Start⇨All Programs⇨Microsoft Visual Studio 2010⇨Microsoft Windows SDK Tools⇨IL Disassembler. На рис. 47.1 с помощью этого инструмента открыта небольшая библиотека классов, и тут же показаны пространство имен и информация о классах, содержащихся в этой сборке.

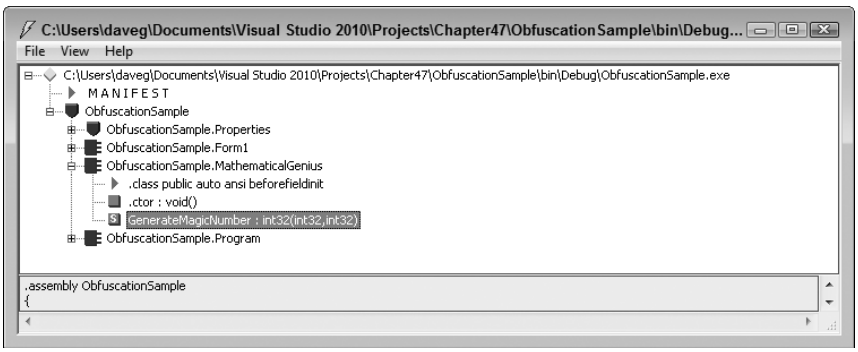


Рис. 47.1

Чтобы сравнить генерируемый код IL, возьмем следующий исходный текст класса `MathematicalGenius`, генерирующий "магическое число":

C#

```
namespace ObfuscationSample
{
  public class MathematicalGenius
  {
    public static Int32 GenerateMagicNumber(Int32 age,
                                            Int32 height)
    {
      return (age * height) + DateTime.Now.DayOfYear;
    }
  }
}
```

VB

```
Namespace ObfuscationSample
  Public Class MathematicalGenius
    Public Shared Function GenerateMagicNumber( _
            ByVal age As Integer, _
            ByVal height As Integer) As Integer
      Return (age * height) + Today.DayOfYear
    End Function
  End Class
End Namespace
```

Дважды щелкните на методе `GenerateMagicNumber` в программе IL Dasm, и получите дополнительное окно, в котором показан IL-текст этого метода. На рис. 47.2 показан этот IL-текст метода `GenerateMagicNumber`, который представляет

ваш сверхсекретный патентованный алгоритм. Но любой, кто готов потратить часик-другой на изучение интерпретации MSIL, может легко понять, что этот метод просто перемножает два параметра типа `int32`, `age` и `height`, а затем добавляет к произведению номер текущего дня в году.

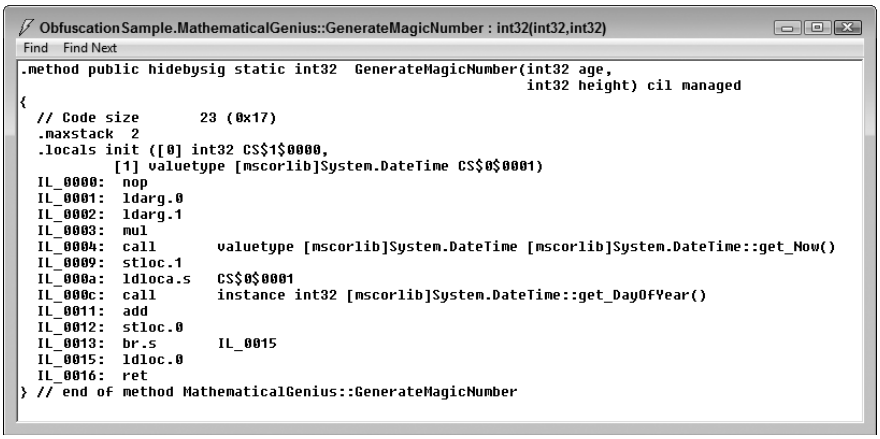


Рис. 47.2

Для тех, у кого нет времени на изучение MSIL, декомпилятор может конвертировать этот IL-текст в один или несколько языков программирования .NET.

Декомпиляторы

Одним из наиболее широко используемых декомпиляторов является программа .NET Reflector от Red Gate Software (которую можно найти по адресу `www.red-gate.com/products/reflector/`). Эту программу можно применять для декомпиляции любых сборок .NET в C#, Visual Basic Managed C11 и даже Delphi. На рис. 47.3 показано, как выглядит в программе Reflector только что рассмотренный нами в IL Dasm код.

В левой части рис. 47.3 показана информация о пространствах имен, типе и методе, подобно тому, как ее представляет IL Dasm. Двойной щелчок на методе должен открыть справа панель Disassembler, которая выводит содержимое метода на языке, выбранном на панели инструментов. В данном случае получен код C#, практически идентичный использованному нами исходному коду.



Можно заметить, что на рис. 47.3 перечислены некоторые из сборок библиотеки базовых классов .NET Framework, включая `System`, `System.Data` и `System.Web`. Поскольку к ним не было применено запутывание кода, они точно так же легко могут быть декомпилированы с помощью программы Reflector. Однако компания Microsoft предоставляет разработчикам большую часть исходных текстов платформы .NET Framework, так что вы можете просмотреть исходные тексты данных сборок, включая комментарии, без применения декомпилятора. Об этом рассказывалось в главе 43.

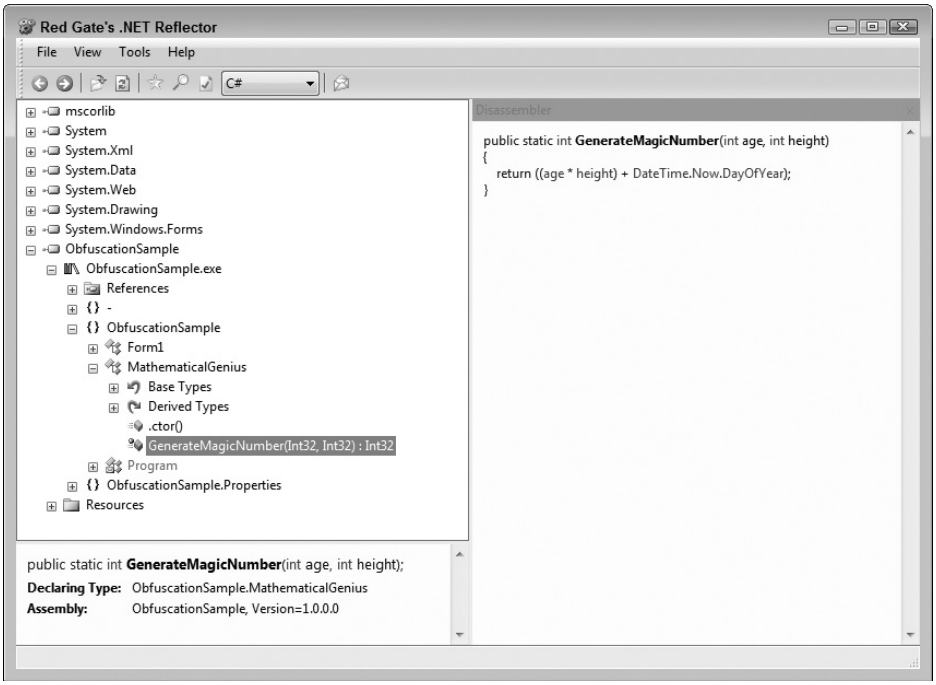


Рис. 47.3

Если бы функция генерации "магического числа" представляла собой реальный секрет, на котором ваша организация делает деньги, возможность декомпиляции приложения представляла бы собой существенный риск. Все было бы еще хуже при использовании дополнения Reflector.FileDisassembler, написанного Дэнисом Бауэром (Denis Bauer), которое можно найти по адресу `www.denisbauer.com/NETTools/FileDisassembler.aspx`. При его использовании сборка может быть целиком декомпилирована в исходные файлы вместе с файлом проекта.

Запутывание кода

До сих пор в данной главе речь шла об оптимальной защите логики вашего приложения. *Умышленное запутывание* (obfuscation) представляет собой переименование символов и модификацию путей кода в сборке, так что логика становится скрытой и не может быть легко понята при декомпиляции. Закрывать код может ряд продуктов, каждый из которых использует собственные способы для того, чтобы результат получился как можно более запутанным. Visual Studio 2010 поставляется с инструментарием Community Edition of Dotfuscator Software Services от PreEmptive Solutions, который используется в этой главе в качестве примера применения технологии запутывания кода.



Запутывание не предотвращает декомпиляцию кода; оно просто делает получающийся в результате декомпиляции код более сложным для понимания. Применение запутывания приводит к последствиям, которые следует учитывать при использовании отражения или строгого именования своего приложения.

Инструмент Dotfuscator Software Services

Хотя программу Dotfuscator можно вызвать из меню Tools системы Visual Studio 2010, это отдельный продукт с собственной лицензией. Версия Community Edition (CE) содержит только часть функциональных возможностей коммерческого продукта Dotfuscator Suite. Если вы действительно заинтересованы в сокрытии функциональности своего приложения, подумайте о приобретении коммерческой версии, дополнительную информацию о которой можно найти по адресу `www.preemptive.com/products/dotfuscator/compare-editions`.

Программа Dotfuscator CE использует собственный формат проекта для отслеживания того, какие сборки закрываются и какие настройки вами указаны. После запуска программы Dotfuscator из меню Tools она открывается с новым несохраненным проектом. Выберите узел Input Assemblies в дереве навигации, а затем щелкните на кнопке с эллипсисом (...) под списком Assembly Name для добавления сборок .NET, которые вы хотите запутать. На рис. 47.4 показан новый проект Dotfuscator, в который добавлена ранее рассматривавшаяся в этой главе сборка.

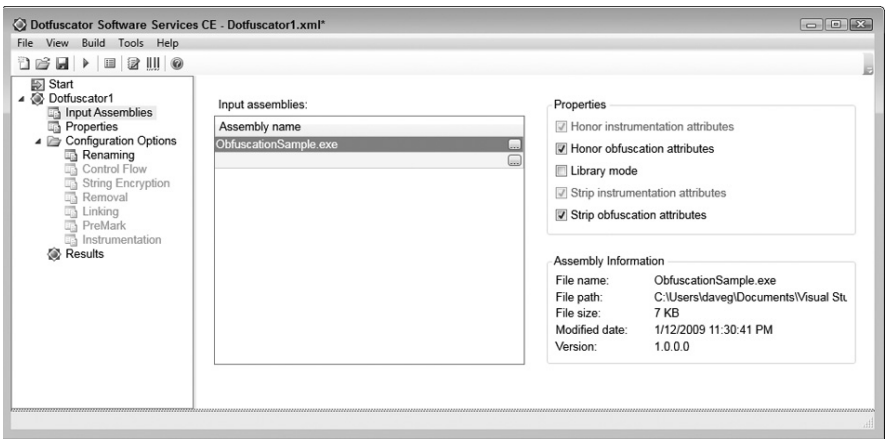


Рис. 47.4



В отличие от других действий, связанных с процессом сборки, которые обычно базируются на исходных файлах, запутывание представляет собой действие, выполняемое после сборки и работающее с уже скомпилированным набором сборок. Программа Dotfuscator получает существующее множество сборок и применяет алгоритм запутывания к IL-коду, генерируя множество новых сборок.

Если нет необходимости обновлять иные настройки, для запутывания приложения можно выбрать команду меню Build⇨Build Project или щелкнуть на кнопке “play” (четвертая слева) панели инструментов. Если проект Dotfuscator был сохранен, закрытые сборки будут добавлены в папку нижележащего уровня `Dotfuscated`, находящуюся в папке, где хранится проект. Если проект не был сохранен, его вывод будет направлен в папку `c:\Dotfuscated`.

Открыв сгенерированную сборку с помощью программы Reflector, как показано на рис. 47.5, можно увидеть, что метод `GenerateMagicNumber` переименован вместе с

входными параметрами. Кроме того, удалена иерархия пространств имен, а классы переименованы. Хотя это и очень простой пример, можно заметить, как методы с похожими именами, ничего не говорящими об их функциональности, могут ввести в заблуждение того, кто просматривает декомпилированный код, и сделать исходный текст очень сложным для понимания.

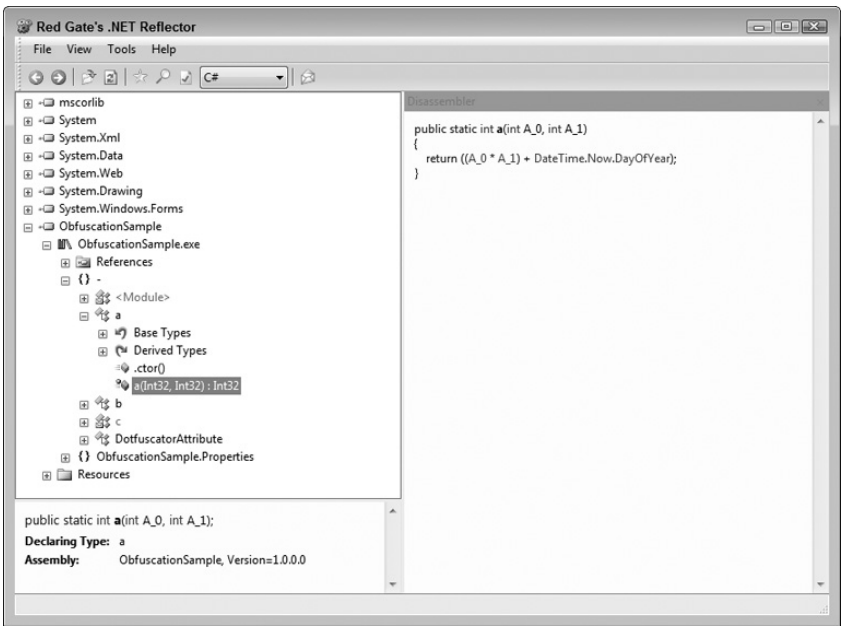


Рис. 47.5



Бесплатная версия Dotfuscator закрывает сборки только с помощью переименования классов, переменных и функций. Коммерческая версия использует несколько дополнительных методов, таких как изменение потока управления в сборке или шифрование строк. Во многих случаях поток управления начинает вызывать в декомпиляторах невосстановимые исключения, препятствуя тем самым автоматической декомпиляции.

Предыдущий пример запутывания открытого метода класса работает только в том случае, если метод вызывается только из сборок, запутанных вместе со сборкой, содержащей определение класса. Однако, если имеется запутанная библиотека классов или функция API, к которой обращаются другие, не запутанные приложения, вы увидите список классов, которые не имеют четкой структуры, взаимосвязей и даже соглашений об именовании. Это делает работу с данной сборкой очень трудной. К счастью, программа Dotfuscator позволяет управлять процессом переименования при запутывании. Перед продолжением работы необходимо выполнить небольшой рефакторинг кода, вынеся функциональность из открытых методов. Если вы не сделаете этого и исключите такой метод из процесса запутывания, ваш секретный алгоритм останется открытым. Выделяя логику в отдельные методы, вы сможете запутать ее, оставив при этом открытый интерфейс неизменным. Код после описанного рефакторинга выглядит примерно так:



C#

```
namespace ObfuscationSample
{
  public class MathematicalGenius
  {
    public static Int32
        GenerateMagicNumber(Int32 age, Int32 height)
    {
      return SecretGenerateMagicNumber(age, height);
    }

    private static Int32
        SecretGenerateMagicNumber(Int32 age, Int32 height)
    {
      return (age * height) + DateTime.Now.DayOfYear;
    }
  }
}
```

Сниппет кода MathematicalGenius.cs



VB

```
Namespace ObfuscationSample
  Public Class MathematicalGenius
    Public Shared Function GenerateMagicNumber( _
                        ByVal age As Integer, _
                        ByVal height As Integer) As Integer
      Return SecretGenerateMagicNumber(age, height)
    End Function

    Private Shared Function SecretGenerateMagicNumber( _
                        ByVal age As Integer, _
                        ByVal height As Integer) As Integer
      Return (age * height) + Today.DayOfWeek
    End Function
  End Class
End Namespace
```

Сниппет кода MathematicalGenius.vb

После новой сборки приложения заново откройте проект `Dotfuscator`, выбрав его из списка Recent Projects. Существует несколько способов избирательного закрытия сборки. Во-первых, можно применить режим Library к конкретным сборкам, установив соответствующий флажок в окне Input Assemblies (рис. 47.4). В результате такой установки пространство имен, имя класса и все открытые свойства и методы останутся неизменными, а все закрытые методы и переменные будут переименованы. Во-вторых, можно вручную выбрать элементы, которые не должны быть переименованы при работе программы Dotfuscator. Для этого откройте элемент Renaming дерева навигации, показанного на рис. 47.6.

Диалоговое окно Renaming открывается на вкладке Exclusions, где вы можете увидеть знакомое дерево сборок с перечисленными атрибутами, пространствами имен, типами и методами. Как следует из названия вкладки, это дерево позволяет исключить определенные элементы из процесса переименования. На рис. 47.6 метод `GenerateMagicNumber`, как и содержащий его класс, исключен (в противном случае вы бы получили что-то наподобие `b.GenerateMagicNumber`, где `b` — переименованный класс). В дополнение к явному выбору исключенных элементов вы можете определить пользовательские правила, включающие регулярные выражения.

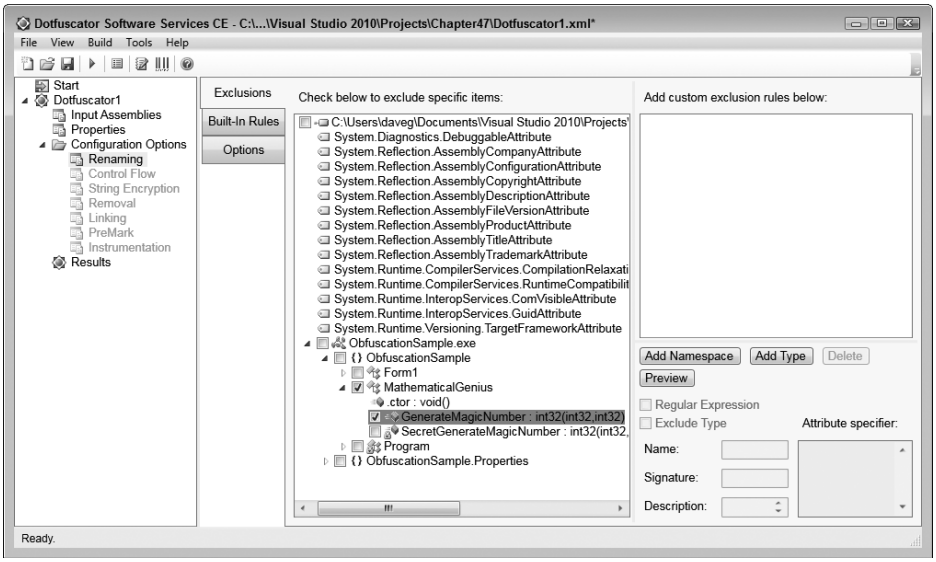


Рис. 47.6

После сборки проекта `Dotfuscator` щелкните на элементе Results в дереве навигации. На экране вы увидите действия, которые Dotfuscator выполнил во время закрытия. Новые имена каждого класса, свойства и метода выводятся как подузлы каждого переименованного элемента дерева. На рис. 47.7 показано, что класс `MathematicalGenius` и метод `GenerateMagicNumber` остались непереименованными.

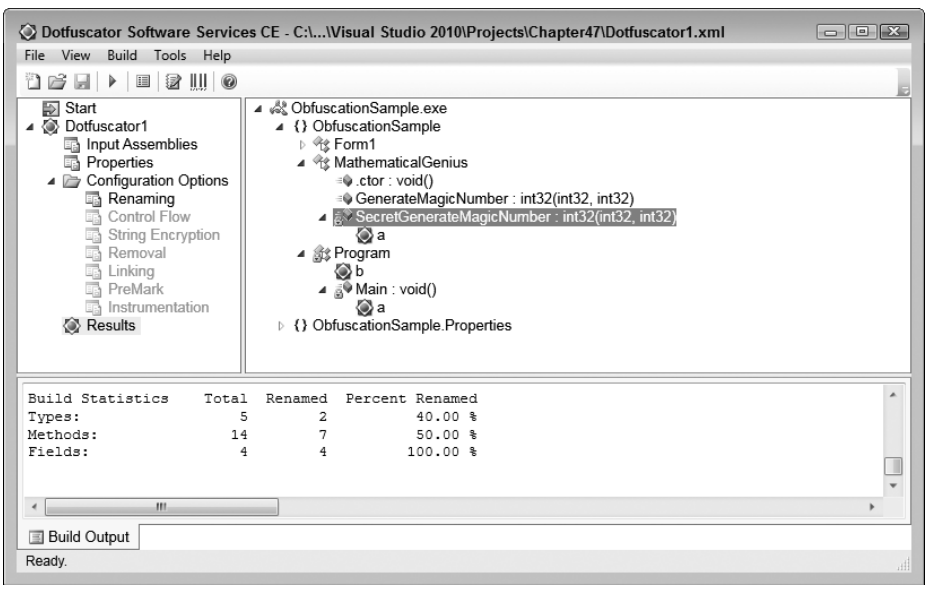


Рис. 47.7

Метод `SecretGenerateMagicNumber` переименован в `a`, как указано в поддереве с пиктограммой Dotfuscator.

Атрибуты запутывания

В предыдущем примере вы увидели, как выбирать типы и методы для закрытия с помощью программы Dotfuscator. Конечно, работая с другим продуктом, его нужно настроить для исключения открытых методов из процесса запутывания. Было бы удобнее иметь возможность аннотировать код атрибутами, указывающими, должен ли быть запутан определенный символ. Это можно сделать, применяя атрибуты `Obfuscation` и `ObfuscationAssemblyAttribute` из пространства имен `System.Reflection`.

Поведение по умолчанию в Dotfuscator заключается в перекрытии исключений, определенных в проекте, исключениями, указанными атрибутами запутывания. На рис. 47.4 показан ряд переключателей для каждой добавленной в проект сборки, один из которых — Honor Obfuscation Attributes. Снимая флажки Honor Obfuscation Attributes для отдельных сборок, вы можете изменить поведение по умолчанию так, чтобы любые установленные в проекте исключения имели более высокий приоритет.

Атрибут ObfuscationAssemblyAttribute

Атрибут `ObfuscationAssemblyAttribute` может быть применен к сборке для управления тем, как она должна рассматриваться — как библиотека классов или как закрытая сборка. Отличие заключается в том, что для библиотеки классов ожидаются ссылки на ее открытые типы и методы со стороны других классов. А потому необходимо указать инструменту запутывания, что эти символы не должны переименовываться.

Напротив, в закрытых сборках каждый символ потенциально может быть переименован. Далее приведен синтаксис `ObfuscationAssemblyAttribute`:

C#

```
[assembly: ObfuscateAssemblyAttribute(false, StripAfterObfuscation=true)]
```

VB

```
<Assembly: ObfuscateAssemblyAttribute(False, StripAfterObfuscation:=True)>
```

Два аргумента, принимаемые данным атрибутом, указывают, является ли сборка закрытой и следует ли раскрывать атрибуты после запутывания. В приведенном фрагменте указывается, что это не закрытая сборка, так что открытые символы не должны быть переименованы. Кроме того, указано, что атрибут запутывания должен быть удален после выполнения запутывания — в конце концов, чем меньше информации доступно всем подряд, тем лучше.

Добавление этого атрибута в файл `AssemblyInfo.cs` или `AssemblyInfo.vb` автоматически сохраняет имена всех открытых символов в приложении `ObfuscationSample`. Это означает, что можно удалить исключение, созданное ранее для метода `GenerateMagicNumber`.

Атрибут ObfuscationAttribute

Недостаток атрибута `ObfuscationAssemblyAttribute` заключается в том, что он делает видимыми все открытые типы и методы независимо от того, существуют они для всеобщего или только для внутреннего использования. С другой стороны, атрибут `ObfuscationAttribute` можно применять к отдельным типам и методам, так что он предоставляет возможность более тонкого контроля над тем, что требуется закрыть. Для иллюстрации применения этого атрибута выполним рефакторинг нашего

примера, чтобы включить в него дополнительный открытый метод EvaluatePerson и переместить логику в другой класс, HiddenGenius:



C#

```
namespace ObfuscationSample
{
  [System.Reflection.ObfuscationAttribute(ApplyToMembers=true, Exclude=true)]
  public class MathematicalGenius
  {
    public static Int32 GenerateMagicNumber(Int32 age, Int32 height)
    {
      return HiddenGenius.GenerateMagicNumber(age, height);
    }
    public static Boolean EvaluatePerson(Int32 age, Int32 height)
    {
      return HiddenGenius.EvaluatePerson(age, height);
    }
  }
  [System.Reflection.ObfuscationAttribute(ApplyToMembers=false,
                                                          Exclude=true)]
  public class HiddenGenius
  {
    public static Int32 GenerateMagicNumber(Int32 age, Int32 height)
    {
      return (age * height) + DateTime.Now.DayOfYear;
    }
    [System.Reflection.ObfuscationAttribute(Exclude=true)]
    public static Boolean EvaluatePerson(Int32 age, Int32 height)
    {
      return GenerateMagicNumber(age, height) > 6000;
    }
  }
}
```

Сниппет кода MathematicalGenius.cs



VB

```
Namespace ObfuscationSample
  <System.Reflection.ObfuscationAttribute(ApplyToMembers:=
                                        True,Exclude:=True)> _
  Public Class MathematicalGenius
    Public Shared Function GenerateMagicNumber(ByVal age As Integer, _
                                          ByVal height As Integer)
                                                    As Integer
      Return HiddenGenius.GenerateMagicNumber(age, height)
    End Function

    Public Shared Function EvaluatePerson(ByVal age As Integer, _
                                        ByVal height As Integer) As Boolean
      Return HiddenGenius.EvaluatePerson(age, height)
    End Function
  End Class

  <System.Reflection.ObfuscationAttribute(ApplyToMembers:=False,
                                                        Exclude:=True)> _
  Public Class HiddenGenius
      Public Shared Function GenerateMagicNumber(ByVal age As Integer, _
                                           ByVal height As Integer)
                                                      As Integer
      Return (age * height) + Today.DayOfWeek
  End Function
```

```
    <System.Reflection.ObfuscationAttribute(Exclude:=True)> _
    Public Shared Function EvaluatePerson(ByVal age As Integer, _
                                          ByVal height As Integer) As Boolean
      Return GenerateMagicNumber(age, height) > 6000
    End Function
  End Class
End Namespace
```

Сниппет кода MathematicalGenius.vb

В данном примере `MathematicalGenius` — это класс, который должен быть виден за пределами библиотеки. Поэтому необходимо предотвратить запутывание этого класса и всех его методов, применив атрибут `ObfuscationAttribute` с параметрами `Exclude` и `ApplyToMembers`, равными `True`.

Второй класс, `HiddenGenius`, запутывается частично. По мнению разработчиков класса, метод `EvaluatePerson` должен быть доступен за пределами класса, а все остальные методы должны быть запутаны. В результате к классу вновь применяется атрибут `ObfuscationAttribute`, так что класс не запутывается. Однако на этот раз нам нужно, чтобы поведение по умолчанию было таким, чтобы символы, содержащиеся в классе, были запутаны. Поэтому параметр `ApplyToMembers` устанавливается равным `False`. Кроме того, к методу `EvaluatePerson` применяется атрибут `Obfuscation`, так что этот метод остается доступным.

Предостережения

В некоторых местах стоит остановиться и подумать о том, что произойдет после применения запутывания, точнее — переименования, и как оно может повлиять на работу приложения.

Отражение

Платформа .NET Framework реализует богатую модель отражения, которая обеспечивает возможность запросов типов и динамического создания экземпляров. К сожалению, некоторые из методов отражения используют просмотр строк для имен типов и членов. Понятно, что применение переименования препятствует успешной работе поиска, так что единственное решение — не изменять никаких символов, которые могут быть задействованы при использовании отражения. Обратите внимание, что запутывание потока управления не имеет такого нежелательного побочного действия. Возможность Dotfuscator под названием *интеллектуальное запутывание* (smart obfuscation) пытается автоматически определить ограниченное множество символов для исключения на основе информации о применении отражения приложением. Например, если вы используете поля имен перечислимого типа, то интеллектуальное запутывание обнаружит вызов отражения для получения имени поля перечисления и автоматически исключит эти поля из переименования.

Строго именованные сборки

Строгие имена присваиваются сборкам для предотвращения вмешательства в их работу. К сожалению, запутывание основано на возможности изменения имен и потока кода в существующей сборке, а это означает, что сборка перестает иметь корректное строгое имя. Чтобы позволить запутывание, необходимо отложить подписание сборки, установив флажок Delay Sign Only во вкладке Signing окна Project Properties, показанного на рис. 47.8.

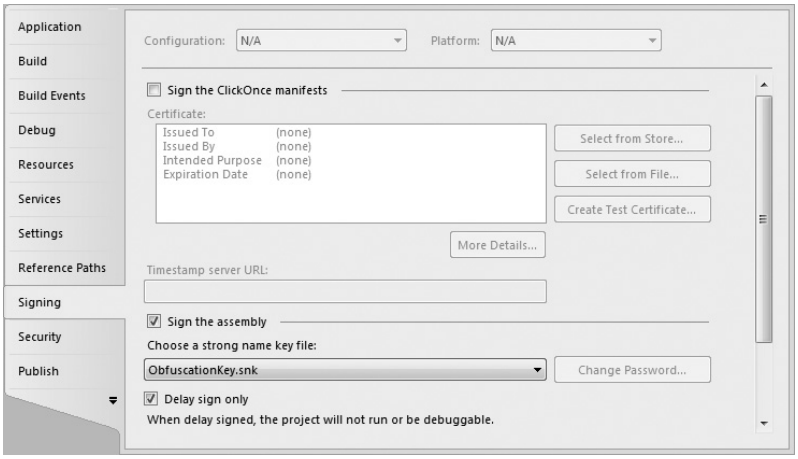


Рис. 47.8

Построенную сборку можно запутать обычным способом. Единственное отличие состоит в том, что после запутывания нужно подписать запутанную сборку, что можно сделать вручную с использованием утилиты Strong Name, как показано в следующем примере:

```
sn -R ObfuscationSample.exe ObfuscationKey.snk
```



Утилита Strong Name не включена в путь по умолчанию, так что нужно либо вызывать ее из приглашения командной строки Visual Studio (Start⇨All Programs⇨Microsoft Visual Studio 2010⇨Visual Studio Tools), либо вводить полный путь к `sn.exe`.

Отладка с отложенной подписью

Установка флажка Delay Sign Only в окне Project Properties не позволяет запускать и отлаживать приложение. Это связано со сбоем процесса верификации строгого имени. Для того чтобы разрешить отладку приложения с отложенной подписью, можете зарегистрировать соответствующие сборки для пропуска процесса верификации. Это также делается с использованием утилиты Strong Name. Например, приведенный далее код опускает верификацию приложения `ObfuscationSample.exe`:

```
sn -Vr ObfuscationSample.exe
```

Следующий же фрагмент восстанавливает ее:

```
sn -Vu ObfuscationSample.exe
```

Выполнять эти действия вручную всякий раз при сборке приложения — сплошная головная боль, так что можно просто добавить к событиям после сборки приложения следующие строки:

```
"$(DevEnvDir)..\..\..\Microsoft SDKs\Windows\v7.0A\bin\sn.exe" -Vr
"$(TargetPath)"
"$(DevEnvDir)..\..\..\MicrosoftSDKs\Windows\v7.0A\bin\sn.exe" -Vr
"$(TargetDir)$(TargetName).vshost$(TargetExt)"
```



В зависимости от конкретной среды может потребоваться изменить эти строки, указав другой корректный путь к `sn.exe`.

Первая строка опускает верификацию для скомпилированного приложения. Однако Visual Studio использует дополнительный файл `vshost` для загрузки приложения при запуске. Это требуется регистрацией для пропуска верификации при запуске сеанса отладки.

Мониторинг приложения и управление им

Версия программы Dotfuscator, поставляемая с Visual Studio 2010, обладает новыми функциональными возможностями, связанными с мониторингом приложений и управлением ими во время их выполнения. Как и в случае запутывания, эта возможность добавляется в приложение после сборки, а это означает, что обычно вам не нужно каким-либо образом модифицировать свой код, чтобы воспользоваться ее преимуществами.

Новая функциональная возможность мониторинга и управления включает следующее.

- **Защита от вмешательства.** В случае неавторизованной модификации вашего приложения завершает его работу и (необязательно) уведомляет об этом факте.
- **Срок действия приложения**. Настройка даты истечения срока действия приложения, после которой оно перестает работать.
- **Отслеживание использования приложения.** Оснащает ваш код средствами для отслеживания его использования путем включения соответствующей функциональной возможности в ваше приложение.

Для того чтобы оснастить свое приложение описанной функциональной возможностью, выполните соответствующую настройку проекта `Dotfuscator`. Найдите элемент Instrumentation в дереве навигации и выберите вкладку Options. Установите флажок Enable Instrumentation, как показано на рис. 47.9.

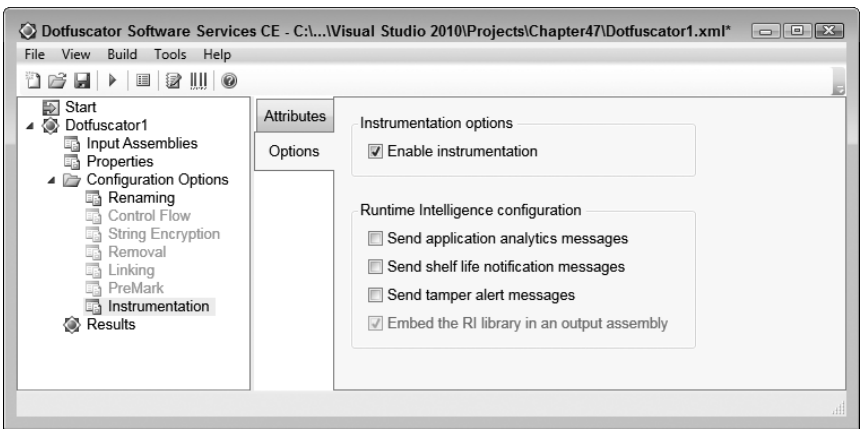


Рис. 47.9

После этого можете указать новую функциональность, которая будет добавлена в ваше приложение, с помощью атрибутов проекта `Dotfuscator` — либо добавляя пользовательские атрибуты в свой код, либо через пользовательский интерфейс `Dotfuscator`.

Защита от вмешательства

Защита от вмешательства предоставляет возможность обнаружения неавторизованной модификации вашего приложения. В то время как запутывание представляет собой *превентивную* меру, направленную на снижение рисков от неавторизованной декомпиляции, защита от вмешательства представляет собой *детективную* меру, направленную на снижение рисков от неавторизованной модификации ваших управляемых сборок. Соединение превентивных и детективных мер представляет собой широко распространенный способ управления рисками, например противопожарные мероприятия, включающие превентивные меры по недопущению возгорания, и детективные меры по обнаружению возгорания.

Защита от вмешательства применима на уровне методов и осуществляется во время работы приложения, при выполнении защищенного метода.

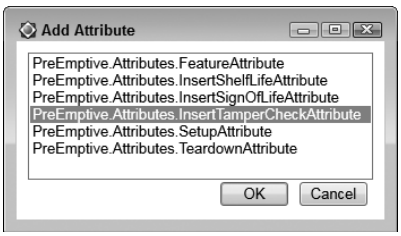


Рис. 47.10

Чтобы добавить защиту от вмешательства в свое приложение, выберите в меню навигации команду Instrumentation, а затем — вкладку Attributes. Вы увидите дерево, содержащее сборки, добавленные в проект `Dotfuscator`, с иерархией классов и методов, имеющихся в этих сборках. Перейдите к функции `HiddenGenius.GenerateMagicNumber` и выберите в контекстном меню, вызываемом щелчком правой кнопкой мыши, команду Add Attribute. После этого будет отображено диалоговое окно со списком доступных атрибутов проекта `Dotfuscator`, показанное на рис. 47.10.

Выберите атрибут `InsertTamperCheckAttribute` и щелкните на кнопке OK. Атрибут будет добавлен к выбранному методу, а его свойства будут перечислены в правой части окна, как показано на рис. 47.11. Наконец выберите свойство `ApplicationNotificationSinkElement` и измените его значение на `DefaultAction`.

Теперь можете создать проект `Dotfuscator` для вставки в свое приложение функции защиты от вмешательства.

Чтобы помочь протестировать эту функцию, программа Dotfuscator поставляется с простой утилитой, которая имитирует неавторизованное вмешательство. Она называется TamperTester и находится в том же каталоге, в который инсталлирована программа Dotfuscator (по умолчанию это `C:\Program Files\Microsoft Visual Studio 10.0\PreEmptive Solutions\Dotfuscator Community Edition`). Утилита запускается из командной строки с именем сборки и выходной папкой в качестве аргументов:

```
tampertester ObfuscationSample.exe c:\tamperedapps
```



Убедитесь, что вы передали утилите TamperTester сборку, сгенерированную программой Dotfuscator, а не исходную сборку, созданную системой Visual Studio.



Рис. 47.11

По умолчанию в случае внесения модификаций ваше приложение немедленно завершит работу. Вы можете настроить программу Dotfuscator для генерации уведомления. Коммерческая версия программы Dotfuscator включает два основных расширения CE-версии. Они позволяют добавить пользовательский обработчик, который вызывается при обнаружении вмешательства и поддерживает пользовательский обработчик реального времени вместо выхода по умолчанию. Кроме того, компания PreEmptive Solutions предоставляет службу уведомлений, которая автоматически уведомляет вашу организацию о происшедшем вмешательстве.

Технологии Runtime Intelligence

Термин *Runtime Intelligence* (RI) означает технологии и службы для сбора, интеграции, анализа и представления информации об использовании приложения. Программа Dotfuscator CE в системе Visual Studio 2010 может внедрять средства RI в произвольную точку сборки для передачи данных о сеансах и использовании возможностей. Далее описываются применение инструментария Dotfuscator's Runtime Intelligence и некоторые доступные бесплатные и коммерческие аналитические возможности.

Для использования инструментария Dotfuscator CE следует включить его в своем проекте `Dotfuscator`. Щелкните на элементе Instrumentation в дереве навигации и выберите вкладку Options. Убедитесь, что все флажки группы Runtime Intelligence Configuration установлены, как показано на рис. 47.12.

Затем добавьте несколько атрибутов в свои сборки, чтобы убедиться, что они могут быть однозначно идентифицированы, а данные инструментария доступны. На вкладке Attributes узла Instrumentation имеется иерархия классов всех сборок, добавленных в проект `Dotfuscator`. Щелкните правой кнопкой мыши на узле верхнего уровня (узле, который содержит полный путь к вашей сборке), выберите команду Add Attribute и добавьте новый атрибут `BusinessAttribute`.

Из списка свойств атрибута выберите `CompanyKey`. Этот атрибут предоставляет уникальный идентификатор для вашей компании и должен быть одним и тем же во всех ваших сборках. Можете щелкнуть на кнопке с троеточием для генерации нового `CompanyKey`. Кроме того, введите значение свойства `CompanyName`, которое будет использоваться в отчетах.

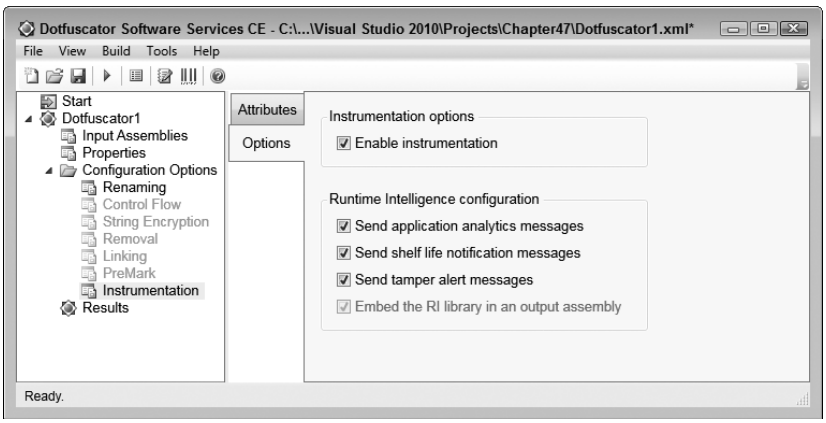


Рис. 47.12

Повторите эти действия для добавления нового атрибута `ApplicationAttribute` в свои сборки. Свойство `GUID` этого атрибута должно содержать уникальный идентификатор, одинаковый для всех сборок приложения. Как и в случае свойства `CompanyKey`, можете сгенерировать значение для свойства `GUID`, щелкнув на кнопке с троеточием. Также следует ввести значения свойств `Name` и `Version`, но свойство `ApplicationType` может оставаться пустым.

После добавления этих атрибутов ваш проект будет выглядеть так, как на рис. 47.13.

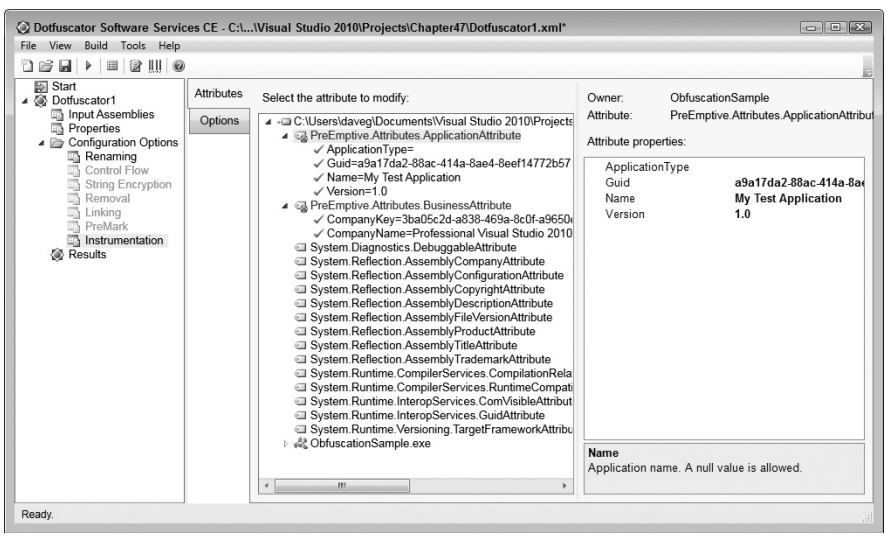


Рис. 47.13

Последний шаг состоит в добавлении в ваше приложение атрибутов SetupAttribute и TeardownAttribute. Их можно добавить к любому методу, и обычно они определяются в приложении один раз, хотя последнее и не строго необходимо, если ваше приложение имеет несколько точек входа и выхода. Атрибут SetupAttribute должен быть помещен в метод, который вызывается вскоре после запуска приложения. Аналогично атрибут TeardownAttribute следует добавить в метод, который вызывается перед выходом из приложения. Заметим, что можно создать методы специально для этих атрибутов.

В случае приложения C# Windows Forms можно поместить эти атрибуты в метод Main; можно также модифицировать класс Program.cs, добавив методы AppStart и AppEnd, как показано в приведенном далее фрагменте кода:



C#

```
static class Program
{
  static void Main()
  {
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    AppStart();
    Application.Run(new Form1());
    AppEnd();
  }

  static void AppStart()
  {
  }

  static void AppEnd()
  {
  }
}
```

Сниппет кода Program.cs

В случае приложения VB Windows Forms можно воспользоваться для указания методов Startup и Shutdown функциональной возможностью Application Events, как показано в приведенном далее фрагменте кода:



VB

```
Imports Microsoft.VisualBasic.ApplicationServices
Namespace My
  Partial Friend Class MyApplication
    Private Sub MyApp_Startup(ByVal sender As Object, _
                              ByVal e As StartupEventArgs) _
                              Handles Me.Startup
    End Sub

    Private Sub MyApp_Shutdown(ByVal sender As Object, _
                               ByVal e As System.EventArgs) _
                               Handles Me.Shutdown
    End Sub
  End Class
End Namespace
```

Сниппет кода ApplicationEvents.vb

После того как вы определили в своем приложении методы startup и shutdown, можете добавить атрибуты SetupAttribute и TeardownAttribute. Найдите в дере-

ве метод `startup`, щелкните на нем правой кнопкой мыши и выберите Add Attribute. Затем выберите `SetupAttribute` и щелкните на кнопке OK. В свойствах атрибута укажите значение свойства `CustomEndpoint`, уведомляющее программу Dotfuscator, куда следует отправлять служебные сообщения. Щелкните на кнопке с троеточием и выберите в списке PreEmptive's Free Runtime Intelligence Services.



Не стоит собирать информацию о применении приложения без разрешения пользователя, поскольку ваш код может быть квалифицирован как шпионящее программное обеспечение (spyware). Для настройки запроса `SetupAttribute` предоставляет три свойства — `OptInSourceElement`, `OptInSourceName` и `OptInSourceOwner`.

Runtime Intelligence Service — это управляемая служба, которую предоставляет компания PreEmptive Solutions и которая накапливает данные, собранные службой Runtime Intelligence, и управляет данными Runtime Intelligence, генерируемыми вашим приложением, и веб-портал, включающий анализ времени выполнения. Компания PreEmptive Solutions предлагает бесплатную и коммерческую версии своей службы Runtime Intelligence Service.



Бесплатная версия PreEmptive's Runtime Intelligence Service используется только для предварительного тестирования, поскольку в ней отсутствуют достаточные средства безопасности и нет никаких гарантий относительно периода работоспособности или сохранности данных. Для промышленных приложений необходимо приобрести коммерческую версию либо найти или создать альтернативное решение.

Наконец, добавьте атрибут `TeardownAttribute` в соответствующий метод и соберите проект `Dotfuscator`. Пару раз запустите приложение для генерации инструментальных сообщений. Можете также воспользоваться утилитой TamperTester, описанной в предыдущем разделе, чтобы создать "взломанную" версию приложения, генерирующую уведомление о неавторизованном вмешательстве.

После нескольких запусков приложения откройте свой браузер и посетите вебсайт `http:/free.runtimeintelligence.com`. Можете войти в портал, используя уникальный идентификатор, сгенерированный ранее для свойства `CompanyKey`.

На экране приветствия, выводимом после входа в портал, показано количество уведомляющих сообщений, ожидающих загрузки в базу данных. Вам может потребоваться подождать, пока все сообщения не будут загружены, перед тем как вы сможете просматривать данные.

В меню Application Overview можно просмотреть результирующие диаграммы, такие как время работы приложения, версии операционной системы и платформы .NET Framework, применяемые вашими пользователями. Отчет Application Scorecard, показанный на рис. 47.14, выводит список всех приложений, зарегистрированных с этим ключом `CompanyKey`, а также ключевые параметры их применения.

Позже вы узнаете, как отслеживание использования приложения можно расширить возможностями, специфичными для вашего приложения.

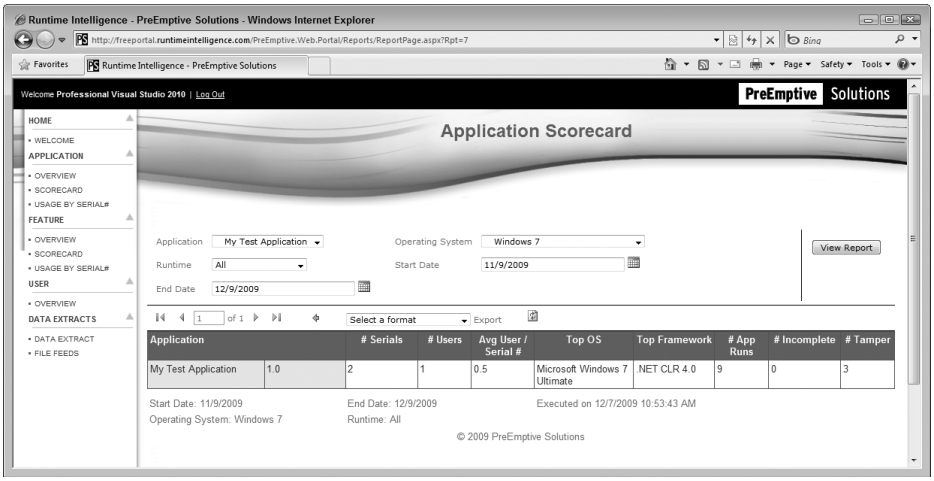


Рис. 47.14

Срок действия приложения

Возможность указания срока действия приложения, известная также как *Shelf Life*, позволяет указать дату истечения срока действия приложения, после которой оно перестанет работать. Это может быть полезно в ряде сценариев, таких как выпуск пробных версий приложения.

Для указания срока действия приложения необходим ключ Shelf Life Activation Key (SLAK). Этот ключ от компании PreEmptive можно запросить по адресу `http://www.preemptive.com/products/shelflife`.

Реализовать указание срока действия приложения помогают два атрибута. Атрибут `InsertShelfLifeAttribute` добавляет дату, гарантируя, что приложение перестанет работать после определенного дня, а также при попытках запуска приложения с истекшим сроком годности отправляет уведомляющее сообщение. Атрибут `InsertSignOfLifeAttribute` отправляет уведомление службе Runtime Intelligence всякий раз при запуске приложения. Это позволяет выяснить, как часто запускается приложение.

Перед добавлением указанных атрибутов оснастите свою сборку атрибутами `BusinessAttribute`, `ApplicationAttribute`, `SetupAttribute` и `TeardownAttribute`, как описано в предыдущем разделе о Runtime Intelligence.

Неплохо добавить атрибуты срока действия в метод, который вызывается вскоре после запуска приложения. Вы можете также добавить их в регулярно вызываемый метод на случай, если пользователи оставят ваше приложение работающим после наступления указанной даты. На рис. 47.15 атрибуты `InsertShelfLifeAttribute` и `InsertSignOfLifeAttribute` добавлены в метод `Form1.InitializeComponent`. Это гарантирует, что проверка срока действия будет выполняться всякий раз при работе с формой `Form1`.

Оба атрибута требуют ключ Shelf Life Activation Key. Получив ключ от компании PreEmptive, сохраните его на локальном диске и укажите путь к нему в свойстве `ActivationKeyFile`. Установите значение свойства `ExpirationDate`, указав прошедшую дату для проверки описываемой функциональной возможности.

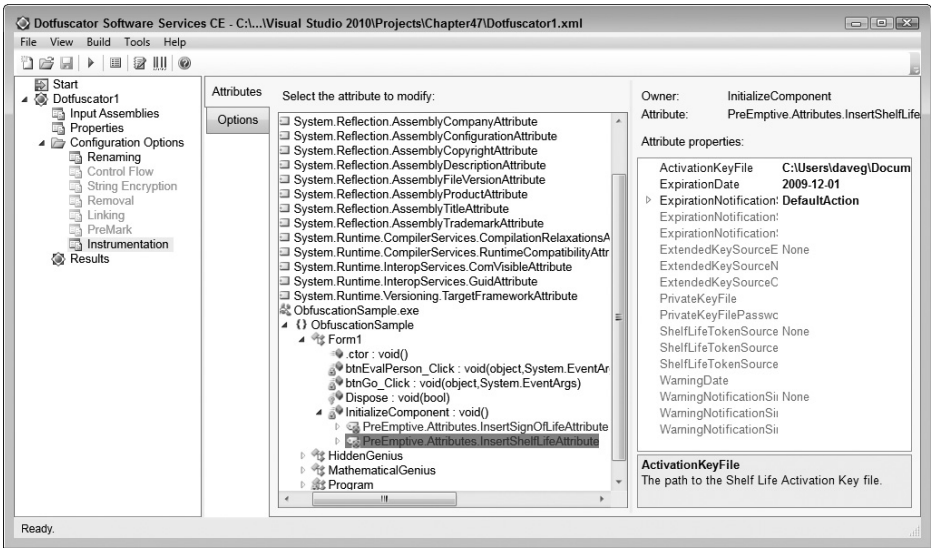


Рис. 47.15

Когда истекает срок работы приложения, его поведение определяется двумя настройками. Во-первых, если на вкладке Instrumentation Options установлен флажок Send Shelf Life Notification Messages, в указанное место будет отправлено уведомляющее сообщение. Во-вторых, если вы установите для свойства `ExpirationNotificationSinkElement` значение `DefaultAction`, приложение будет немедленно завершать работу.

Коммерческая версия программы Dotfuscator позволяет указать дату предупреждения и добавить пользовательские обработчики, запускаемые, когда достигнуты дата предупреждения и дата истечения срока действия. Их можно использовать для отключения некоторых функциональных возможностей или для вывода предупреждающего сообщения пользователю.

Коммерческая версия Dotfuscator позволяет вашему приложению также получать информацию из внешнего источника, такого как веб-служба или конфигурационный файл. Так, вы можете применить другой сценарий, например прекращение работоспособности через месяц с момента установки приложения.

Отслеживание использования приложения

Ранее в этой главе было показано, как в ваше приложение добавляются атрибуты `SetupAttribute` и `TeardownAttribute`. При их добавлении в приложение можно отправлять уведомляющие сообщения и тем самым отслеживать данные об использовании приложения и о его системном окружении. Эти атрибуты могут использоваться и для определения устойчивости приложения, так как отсутствие уведомления Teardown говорит о возможном аварийном останове (или о том, что пользователь оказался настолько разочарованным вашим приложением, что прекратил выполнение).

Программа Dotfuscator позволяет не только отслеживать запуск и завершение работы приложения, но и оснащать код специальными возможностями для отслеживания его использования. В версии Dotfuscator CE можно добавить в свои методы до де-

сяти атрибутов `FeatureAttribute`, каждый из которых указывает одно и то же (или разные) свойство. Так можно агрегировать методы вашего приложения в логические высокоуровневые группы, не зависящие от реальной иерархии классов в вашем коде.

На рис. 47.16 можно увидеть атрибут `FeatureAttribute`, добавленный к методам `EvaluatePerson` и `GenerateMagicNumber` класса `MathematicalGenius`. Эти возможности описывают имя, которое выводится при просмотре отчета об использовании приложения. Данные атрибуты имеют также свойство `FeatureEventType` со значением `Tick`, которое указывает, что отслеживается сам факт использования функциональной возможности.

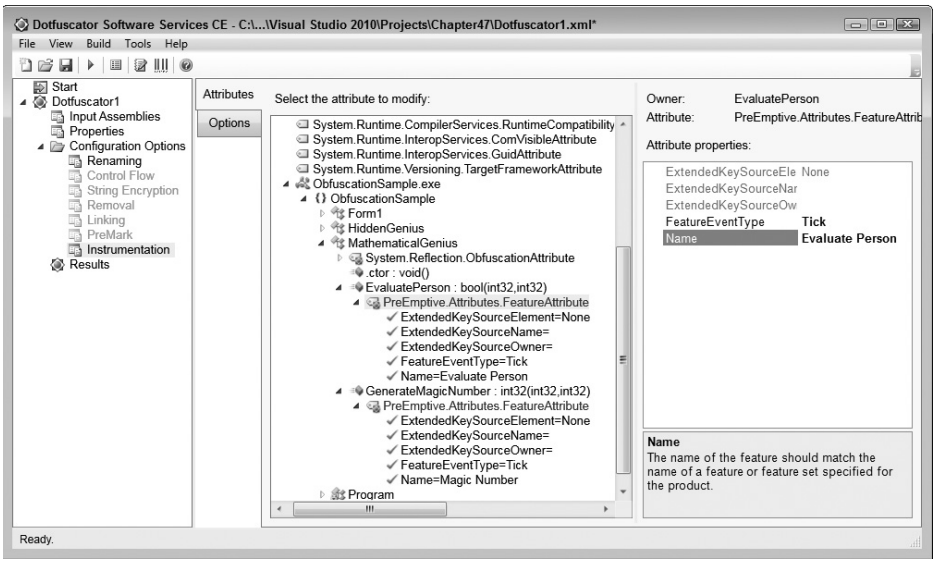


Рис. 47.16

Кроме факта использования, можно отслеживать количество времени работы функциональной возможности. В этом случае нужно добавить два атрибута `FeatureAttribute`: у одного свойство `FeatureEventType` имеет значение `Start`, а у другого — `Stop`. Так генерируются два служебных сообщения, позволяя аналитической службе Runtime Intelligence вычислять продолжительность использования функциональной возможности.

Коммерческая версия программы Dotfuscator включает:

- неограниченную возможность отслежвания;
- применение пакета Microsoft WMI SDK для работы с аппаратным и программным стеками;
- расширенное накопление данных, включая значения пользовательских данных;
- возможность SSL-пересылки данных.

Данные Runtime Intelligence могут использоваться для усовершенствования процесса разработки (предоставляя информацию по использованию приложением различных операций) и как дополнительный источник данных о бизнес-активности и производительности. У компании Microsoft есть собственная аналогичная програм-

ма сбора пользовательских данных Customer Experience Improvement Program (CEIP) (ее подробное описание можно найти по адресу `http://www.microsoft.com/products/ceip/`).

По мере перемещения управляемого кода за пределы настольных систем в веб-клиенты и интернет (Silverlight, Azure), а также в мобильные устройства, служба Runtime Intelligence играет все более важную роль в качестве составной части инструментария для управления жизненным циклом приложения.

Резюме

В этой главе рассмотрены инструменты IL Dasm и Reflector, продемонстрировано, насколько просто решаются задачи реверсивного проектирования сборок .NET, и приведена информация об их работе. Вы также узнали о том, как применять службу Dotfuscator Software Services для:

- защиты вашей интеллектуальной собственности;
- усложнения несанкционированного вмешательства в ваши приложения;
- мониторинга и оценки интенсивности использования приложения с помощью средств Runtime Intelligence;
- обеспечения срока действия вашего приложения.

48

Упаковка и развертывание

В ЭТОЙ ГЛАВЕ...

- Создание инсталляторов для проектов
- Настройка процесса инсталляции
- Проверка соответствия требованиям приложения
- Установка служб Windows
- Развертывание проектов через веб с помощью технологии ClickOnce
- Обновление проектов ClickOnce

Одна из тем разработки программного обеспечения, которая часто остается неосвещенной, — развертывание готовых приложений. Создание инсталлятора — простой процесс, который может превратить ваше приложение из любительской утилиты в профессиональный инструмент. В этой главе рассматривается создание инсталлятора Windows Installer для любого типа приложений .NET.

Система Visual Studio 2010 включает также поддержку развертывания ClickOnce, которое можно использовать для создания динамически обновляемых приложений. Это особенно важно для периодически обновляемых приложений, так как решает задачу развертывания таких обновлений.

Инсталляторы Windows

Windows Installer стал стандартной частью Windows начиная с Windows 2000. Это компонент, который управляет инсталляцией, поддержкой и удалением программных пакетов. Visual Studio 2010 содержит богатый пользовательский интерфейс для создания инсталляционных пакетов, которые предназначены для инфраструктуры Windows Installer, для множества различных приложений. В этой главе демонстрируется построение инсталляторов для настольных приложений и служб Windows. В следующей главе содержится информация о пакетах и развертывании веб-приложений.

Создание инсталлятора

Для создания инсталлятора с помощью системы Visual Studio 2010 необходимо добавить дополнительный проект в приложение, которое вы намерены распространять. На рис. 48.1 показаны доступные типы проектов для настройки и развертывания. Проект Setup Project должен использоваться для приложений Windows Forms или служб, а Web Setup Project – для веб-сайтов или веб-служб `ASP.NET`. Если вы хотите создать инсталлятор, который будет интегрирован в больший инсталлятор, начните с проекта Merge Module Project. CAB Project можно использовать для создания пакета, который будет распространяться через веб-браузер. Мастер Setup Wizard шаг за шагом проведет вас через процесс создания корректного проекта для типа приложения, которое вы намерены распространять.

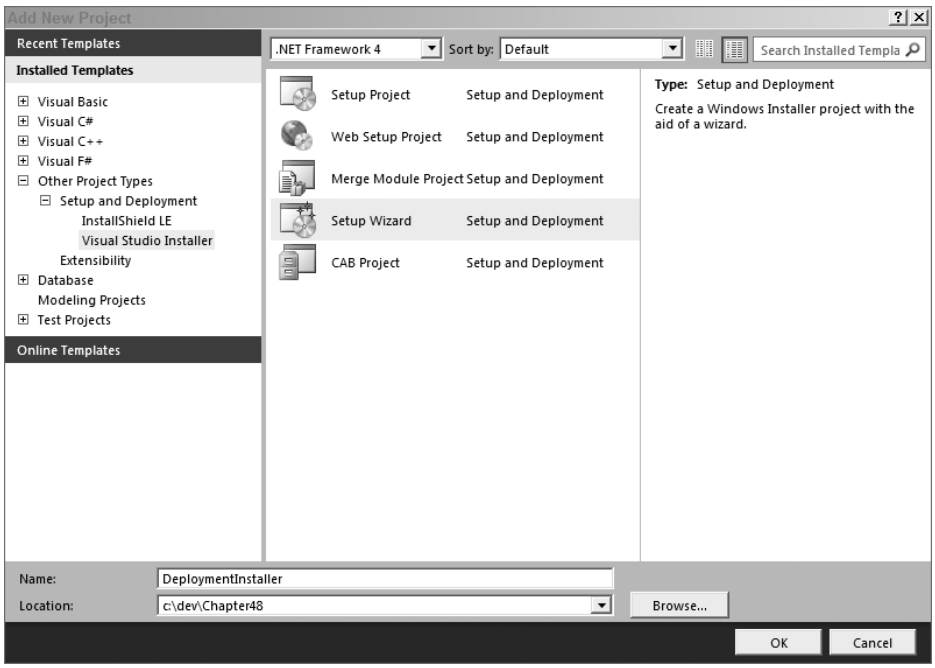


Рис. 48.1



Web Setup Project рассматривается в следующей главе.

В данном случае воспользуемся Setup Wizard для создания инсталлятора для простого приложения C# Windows Forms – CallCentre. После заставки Setup Wizard первое, что нужно предпринять, – сделать выбор между инсталлятором и свободно распространяемым пакетом. При выборе инсталлятора следует определиться, будет ли это инсталлятор приложения Windows или веб-приложения. Основное отличие в том, что инсталлятор приложения Windows помещает приложение в соответствующую папку в `Program Files`, в то время как инсталлятор веб-приложения создает виртуальный каталог в корневом каталоге указанного веб-сайта. В случае свободно распространяемого пакета необходимо делать выбор между модулем, который может быть интегрирован в больший инсталлятор, и CAB-файлом.



Merge Module — это специальный тип инсталлятора, который содержит инструкции по установке совместно используемого компонента. После того как такой модуль создан, он может быть включен во многие другие инсталляторы. Сами по себе такие модули не устанавливаются и должны обязательно быть включены в файл инсталлятора.



Проект CAB создает коллекцию .cab-файлов, содержащую множество сжатых файлов. Эти .cab-файлы могут загружаться или открываться с разных дисков по отдельности. Обычно содержимое .cab-файлов копируется при работе инсталлятора в локальную файловую систему.

Независимо от типа создаваемого проекта, следующий шаг мастера Setup Wizard наиболее важный, поскольку он определяет множество распространяемых файлов. На рис. 48.2 показан третий экран мастера Setup Wizard, который предлагает выбрать, какие файлы следует включить в проект. В данном случае выбраны выходные файлы проекта CallCentre Application, так как требуется включить главный выполняемый файл и все сборки, от которых он зависит. Выбран также элемент Content Files, который включает все файлы с действиями сборки. Последними выбираются файлы, которые не являются частью какого-либо проекта. Например, это могут быть примечания к выпуску, информация о лицензии, документация или файлы README.

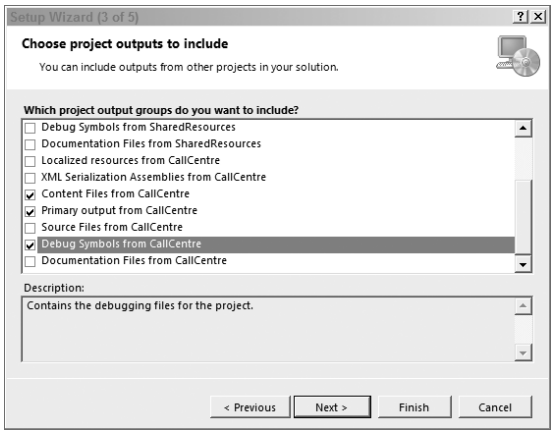


Рис. 48.2



Иногда в пакет включаются файлы с отладочной информацией, которые могут выявить ошибки в распространяемом приложении. Однако обычно этого не происходит, поскольку в инсталлятор придется добавлять множество диагностических инструментов.

Созданный проект добавляется в окно Solution Explorer, как показано на рис. 48.3. Хотя вы не можете явно добавить в проект файлы из библиотеки классов SharedResources, они добавляются при обнаружении зависимостей. Если части, от которых зависит распространяемый проект, гарантированно содержатся на целевом компьютере, их можно вручную удалить из проекта, выбрав команду Exclude контекстного меню, вызываемого щелчком правой кнопкой мыши. Например, если это дополнение к другому приложению, которое содержит копию

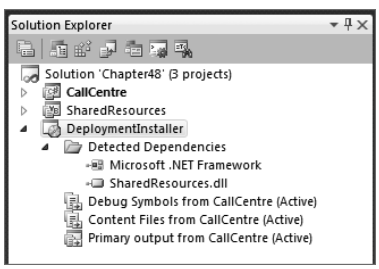


Рис. 48.3

сборки SharedResources, ее можно исключить из списка зависимостей. В результате инсталлятор может быть меньшего размера и проще распространяться.



Если зависимости между вашими проектами изменяются, возможно, их придется пересмотреть. Это можно сделать, выбрав команду Refresh Dependencies контекстного меню, вызываемого щелчком правой кнопкой мыши на узле Detected Dependencies.

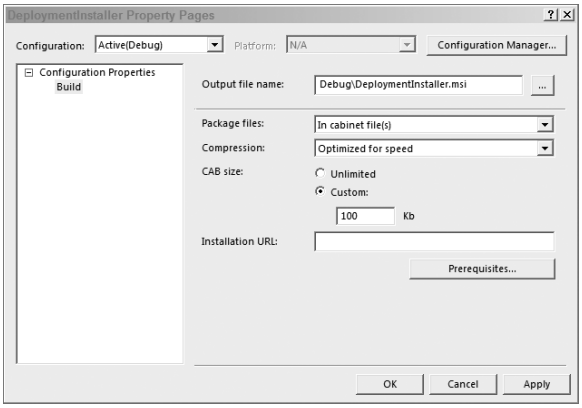


Рис. 48.4

После выбора проекта DeployментInstaller в верхней части окна Solution Explorer появляется ряд новых пиктограмм, как показано на рис. 48.3. В отличие от других типов проектов, когда свойства проекта появляются в области редактирования, после щелчка на первой пиктограмме (Properties) открывается диалоговое окно Property Pages, показанное на рис. 48.4. Оно может использоваться для настройки процесса сборки модуля развертывания. Обратиться к этому диалоговому окна можно также посредством элемента Properties контекстного меню, вызываемого щелчком правой кнопкой мыши на проекте в окне Solution Explorer.

По умолчанию свойство Package Files установлено равным In setup file, так что все выполняемые файлы и файлы, от которых они зависят, помещаются в создаваемый .msi-файл. Проект развертывания создает также файл Setup.exe, который проверяет соответствие минимальным требованиям, таким как наличие платформы .NET Framework, до вызова .msi-файла для установки приложения. Хотя для оптимизации размера файла применяется сжатие, для больших проектов включение всего необходимого в единственный распространяемый файл может стать проблемой. В качестве альтернативы, как показано на рис. 48.4, можно упаковать приложение в ряд CAB-файлов. В этом случае размер CAB-файла ограничен 100 Кбайт, что облегчает развертывание в случае медленной сети. Еще один сценарий, когда такая возможность оказывается полезной, — развертывание приложения на CD или DVD при суммарном объеме информации, превышающем размер одного диска.

Последнее свойство на этой странице — Installation URL. Если вы планируете распространять свое приложение через веб-сайт, можете предпочесть упаковку всего необходимого в единственный файл, и в этом случае вам не надо указывать Installation URL, так как вы просто добавляете ссылку на файл Setup.exe на соответствующем веб-сайте, и пользователь может инсталлировать приложение, просто щелкнув на ссылке. В качестве альтернативы можно упаковать приложение в меньшие модули, которые можно загружать постепенно. Для этого следует указать Installation URL, из которого будет выполняться установка.

Как только что говорилось, проект развертывания по умолчанию создает файл Setup.exe. Кнопка Prerequisites открывает диалоговое окно, подобное показанному на рис. 48.5, в котором можно настроить поведение этого файла. Вы можете указать,

что файл `Setup.exe` не должен создаваться, и в этом случае приложение можно инсталлировать, дважды щелкнув на `.msi`-файле. При этом, конечно же, убирается проверка инсталляции платформы .NET Framework.

В дополнение к платформе .NET Framework вы можете указать и другие компоненты, такие как SQL Server Compact 3.5, которые должны быть установлены. При выполнении этих проверок пользователю будет предложено инсталлировать все отсутствующие компоненты до вызова основного файла инсталлятора. В зависимости от того, как вы хотите распространять свое приложение, хранение всех необходимых программ в том же месте, где находится ваше основное приложение, может оказаться полезным и сберечь много времени, которое пришлось бы потратить на их поиск и загрузку.

Вернемся к окну Solution Explorer и к нашему проекту `DeploymentInstaller`. Имеется дополнительное окно Properties для проектов развертывания, которое может быть открыто после выбора соответствующего проекта и нажатия <F4>. При этом откроется стандартное окно Properties, показанное на рис. 48.6, которое может использоваться для указания деталей развертывания инсталлируемого приложения.

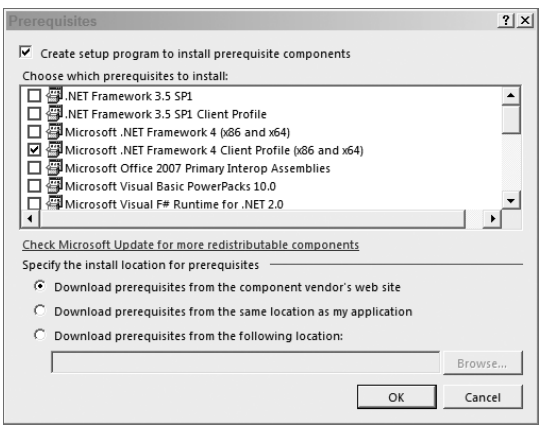


Рис. 48.5

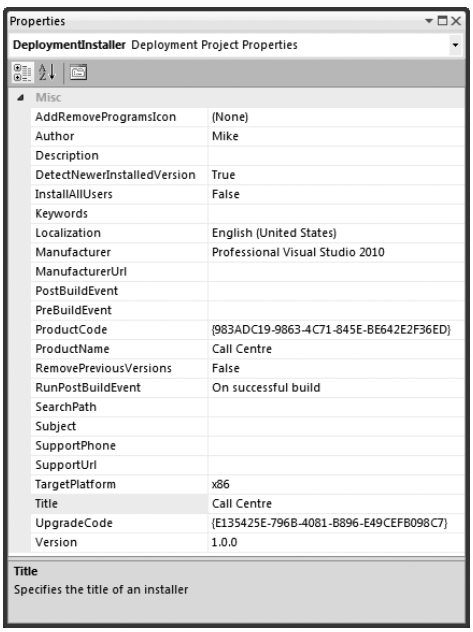


Рис. 48.6

Свойства проекта развертывания, показанные на этом экране, позволяют настроить внешний вид, пиктограммы и поведение мастера инсталляции. Настоятельно рекомендуется настроить эти свойства таким образом, чтобы ваше приложение легко идентифицировалось в диалоговом окне Add/Remove Programs и инсталляция выглядела профессионально, а не наполовину незаконченной. Как можно видеть на рис. 48.7, некоторые из этих свойств используются для настройки диалогового окна инсталлятора.

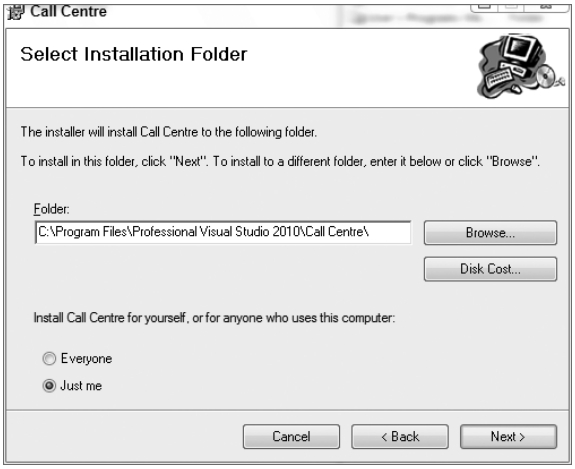


Рис. 48.7

После того как приложение инсталлировано, некоторые из его свойств появляются в диалоговом окне Programs and Features (Add/Remove Programs в Windows 7), доступном через Control Panel, как показано на рис. 48.8. Здесь вы можете увидеть свойства AddRemoveProgramsIcon, ProductName и Manufacturer. Дополнительные свойства можно увидеть после выбора команды More в контекстном меню, вызываемом щелчком правой кнопкой мыши на полосе заголовка.

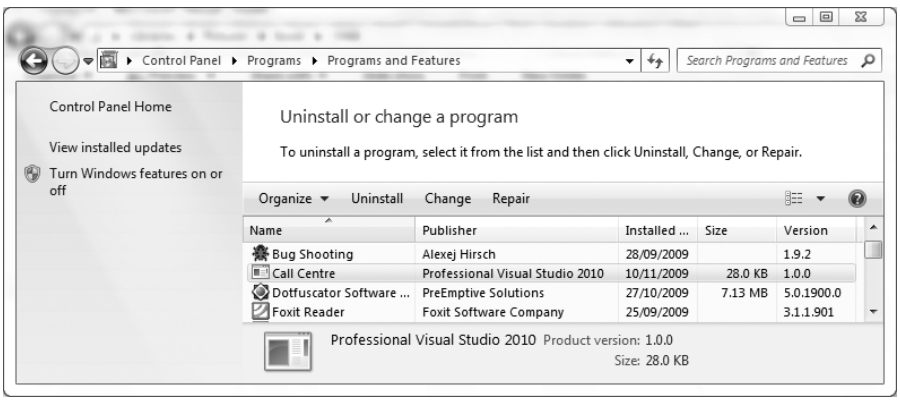


Рис. 48.8



Чтобы протестировать свой инсталлятор, можете выбрать команду Install (а позже — Uninstall) контекстного меню, вызываемого щелчком правой кнопкой мыши на проекте в окне Solution Explorer. Если эта команда в меню отключена, сначала выполните сборку проекта.

Настройка инсталлятора

Оставшиеся пиктограммы в верхней части окна Solution Explorer используются для настройки того, что будет включено в распространяемый пакет. Кроме пиктограмм быстрого вызова, можно использовать команду View контекстного меню, вызываемого щелчком правой кнопкой мыши. Начнем с представления File System, которое указывает, куда на целевом компьютере будут инсталлированы файлы. По умолчанию для приложений Windows используется узел Application Folder, как показано на рис. 48.9. Выберите этот узел и взгляните на окно Properties, указывающее, что место расположения этой папки по умол-

чанию — [ProgramFilesFolder][Manufacturer]\[ProductName]. Оно состоит из трех предопределенных переменных инсталляции, ProgramFilesFolder, Manufacturer и ProductName, значения которых будут определены и скомбинированы в процессе инсталляции. Как показано на рис. 48.7, мастер позволяет пользователям изменять это местоположение в процессе инсталляции.

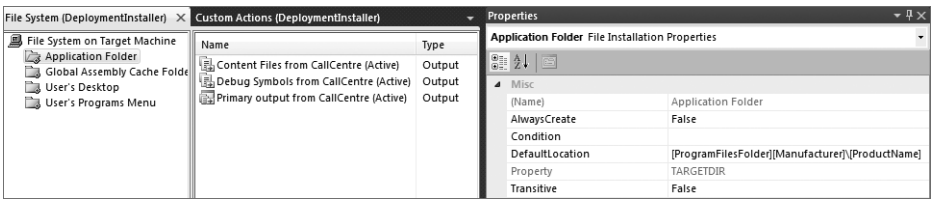


Рис. 48.9

Вы уже знаете, что проект CallCentre Application зависит от сборки SharedResources. На рис. 48.9 эта сборка удалена из закладки Application Folder и перемещена в папку Global Assembly Cache Folder. При инсталляции приложения основной выполняемый файл помещается в соответствующий подкаталог Program Files, а сборка SharedResources инсталлируется в кэш Global Assembly Cache, так что она доступна для любого приложения .NET. Чтобы достичь этого, сначала надо создать папку в представлении File System, выбрав закладку Global Assembly Cache Folder из элемента Add Special Folder контекстного меню, которое появляется после щелчка правой кнопкой мыши. В качестве части общей инсталляции можно устанавливать файлы в ряд других специальных папок. Следующий шаг состоит в перемещении сборки SharedResources в глобальный кэш сборок путем выбора сборки на правой панели представления File System и замене значения свойства Folder с Application Folder значением Global Assembly Cache Folder. Можно также перетащить элемент из закладки Application Folder в закладку Global Assembly Cache Folder.

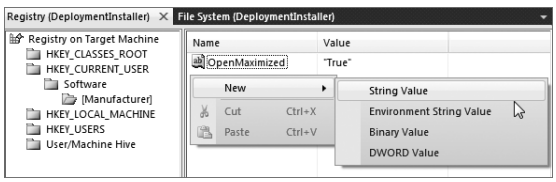


Рис. 48.10

В дополнение к инсталляции файлов на целевом компьютере можно добавить ключи в реестр. Одни разработчики выступают за использование реестра, другие — против. Хотя он и представляет собой удобное хранилище для информации конфигурации на уровне пользователей, настройки приложения с областью видимости пользователя позволяют легче ими управлять. Представление Registry, как показано на рис. 48.10, может использоваться для добавления ключей и значений в реестр. Чтобы добавить новый ключ, щелкните правой кнопкой мыши в дереве Registry и выберите команду Add Key из контекстного меню. Чтобы добавить новое значение, выберите соответствующий ключ в дереве Registry и тип значения из команды New контекстного меню, вызываемого щелчком правой кнопкой мыши, на правой панели представления, как показано на рис. 48.10. Элементы Name и Value могут затем быть установлены с помощью окна Properties.

На рис. 48.11 показано представление File Types распространяемого проекта. Это представление используется для добавления расширений инсталлируемых файлов. Например, в данном случае вы устанавливаете расширение .call. Можете указать

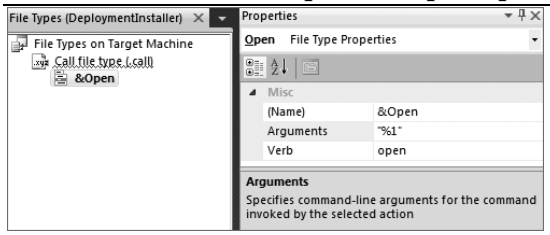


Рис. 48.11

для этого типа пиктограмму, а также определить выполняемый файл, который будет вызываться для файлов данного типа. В большинстве случаев это будет основной файл вашего приложения. Чтобы добавить новый тип файлов, щелкните правой кнопкой мыши на корневом узле дерева File Types и выберите из контекстного меню Add File Types. При этом будет создан узел для нового типа файлов и для действия по умолчанию (выделено полужирным шрифтом) для этого типа. Для расширения `.call` действием по умолчанию является Open, и оно может быть выполнено с помощью двойного щелчка на файле соответствующего типа. Команда Open также выводится (вновь выделенное полужирным шрифтом) в контекстном меню, вызываемом щелчком правой кнопкой мыши на файле с расширением `.call`. Можете добавить для этого типа файлов и другие действия, выбрав команду Add Action контекстного меню, вызываемого щелчком правой кнопкой мыши, для данного типа файлов. Действие по умолчанию можно сменить, выбрав команду Set as Default контекстного меню, вызываемого щелчком правой кнопкой мыши на новом действии. Можно также изменить порядок вывода действий в контекстном меню, перемещая их вверх и вниз по дереву.

Приложения .NET могут быть автономными, и их список зависимостей содержит только платформу .NET Framework. Однако веб-приложения требуют наличия системы IIS, а более сложные приложения могут требовать установки сервера SQL Server. Можете убедиться в выполнении этих требований с помощью проверок, устанавливаемых в представлении, которое показано на рис. 48.12. По умолчанию платформа .NET Framework изначально добавлен к рассматриваемым условиям. Ранее вы видели, что `Setup.exe` проверяет наличие платформы .NET Framework и инсталлирует его, если он не установлен. Условия запуска внедрены в `.msi`-файл и, в отличие от условий в файле `Setup.exe`, проверяются даже при непосредственной установке файла `.msi`. Единственным ограничением является то, что условия запуска всего лишь выводят предупреждение и URL, указывающий на более подробную информацию. В случае условия запуска для платформы .NET Framework и сообщение, и URL изначально вам предоставлены.

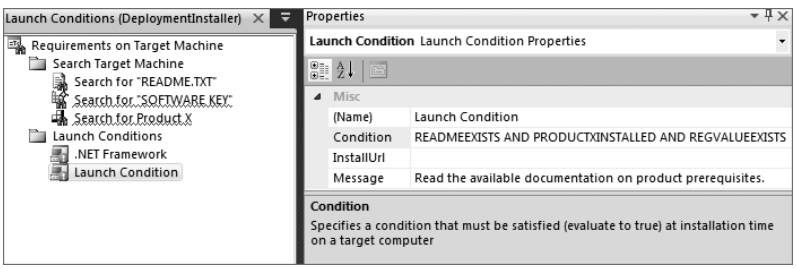


Рис. 48.12



Когда файл `Setup.exe` проверяет наличие платформы .NET Framework, он ограничен поиском или установкой конкретной версии. Условие запуска платформы .NET Framework также может быть настроено таким образом, чтобы обнаруживать конкретную версию, но, если ваше приложение способно работать с несколькими версиями, можете установить для свойства Version условия запуска значение Any. При этом ваше приложение будет устанавливаться при наличии любой установленной версии платформы .NET Framework.

Дерево в левой части рис. 48.12 на самом деле разделено на две части. Верхняя половина дерева используется для указания, какие поиски должны быть выполнены на целевой машине. Могут выполняться поиски файлов, инсталлированных компонентов или приложений, а также значений реестра. Свойство поиска файла включает папку поиска, версию и дату модификации, а также размер файла. Для поиска установленных компонентов следует знать идентификатор Component ID, который указывается в `.msi`-файле инсталлируемого программного обеспечения. Эта информация может быть получена с помощью такого продукта, как Orca, включенного в состав Windows SDK Components for Windows Installer Developers, который можно загрузить с веб-сайта Microsoft (`www.microsoft.com/downloads/`). Поиск в реестре требует свойств, указывающих искомые ключ, имя и значение. В каждом из этих случаев поиску должен быть назначен идентификатор `Property`. Если поиск успешен, свойство инсталлятора с этим идентификатором получает значение `True`.

Идентификаторы `Property`, назначаемые поискам на целевой машине, могут использоваться в условиях запуска в нижней части дерева. На рис. 48.12 показаны условие для проверки наличия .NET Framework и пользовательское условие запуска. Свойство `Condition` установлено равным логическому И всех трех результатов поиска. Таким образом, если любой из поисков будет неудачным, соответствующий идентификатор получит значение `False`, а значит, и все логическое выражение будет ложным. Это предотвратит инсталляцию приложения, и пользователю будет выведено предупреждающее сообщение.

Заметим, что в некоторых других представлениях ряд узлов дерева имеют свойство `Condition`. Например, в представлении File System каждый файл имеет свойство `Condition`, для которого может быть указано условие. Если это условие ложно, файл на целевой машине не устанавливается. В каждом из случаев синтаксис свойства `Condition` должен быть корректен для функции `MsiEvaluateCondition`, вызываемой как часть процесса инсталляции. Эта функция принимает стандартные операторы сравнения, такие как равно (`=`), не равно (`<>`), меньше (`<`), больше (`>`), а также логические операторы `NOT`, `AND`, `OR` и `XOR`. Имеется также ряд предопределенных свойств инсталлятора Windows, которые могут быть включены в свойство условия. Далее приведено подмножество полного списка, который можно найти в документации Windows Installer SDK.

- `ComputerName`: имя целевого компьютера
- `VersionNT`: версия Windows целевого компьютера
- `VersionNT64`: то же значение, что и `VersionNT`, но данное свойство устанавливается только для 64-битных операционных систем

- `ServicePackLevel`: установленный пакет обновлений
- `LogonUser`: имя текущего пользователя
- `AdminUser`: имеет ли текущий пользователь привилегии администратора
- `COMPANYNAME`: имя компании, указанное в мастере инсталляции
- `USERNAME`: имя пользователя, указанное в мастере инсталляции

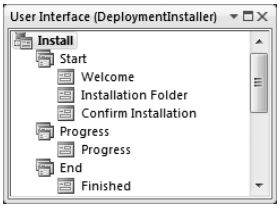


Рис. 48.13

Одна из основных причин создания инсталляторов — желание сделать процесс развертывания приложения более простым, удобным и унифицированным. Для этого необходимо создать простой пользовательский интерфейс, в который конечный пользователь сможет вводить значения. Это может быть каталог для инсталляции или иные параметры, требующиеся для настройки приложения. Ясно, что чем меньше шагов в инсталляторе, тем проще для пользователя процесс инсталляции. Однако, пожалуй, лучше задать пользователю больше вопросов при инсталляции, чем заставить его гадать после инсталляции, почему приложение не работает. Представление User Interface, показанное на рис. 48.13, позволяет настроить экраны, которые будут выводиться пользователю в процессе инсталляции.

В этом приложении определены два пользовательских интерфейса: стандартная инсталляция и административная инсталляция (не видна). Оба процесса следуют одной и той же структуре: Start, где обычно собирается информация от пользователя перед установкой продукта; Progress, когда пользователю выводится визуальная информация о состоянии процесса инсталляции; и End, где пользователь получает резюмирующую информацию о выполненной инсталляции. Административная инсталляция обычно используется, когда требуется настройка сети, и может запускаться с помощью вызова `msiexec` с параметром `/a`.

Настраивать процесс инсталляции можно добавлением и/или удалением диалоговых окон из дерева пользовательского интерфейса. Для добавления нового диалогового окна щелкните правой кнопкой мыши на любой из трех стадий процесса инсталляции и выберите команду Add Dialog контекстного меню. При этом будет выведен список предопределенных диалоговых окон, из которых вы можете выбрать нужное вам. Каждое из диалоговых окон имеет различную схему размещения элементов; одни из них используются для получения ввода от пользователя, а другие — для вывода сообщений пользователю. Управляющие элементы для ввода информации передают введенные значения для последующего использования в процессе инсталляции через идентификаторы свойств. Например, для указания того, какие из инструментов продукта должны быть установлены, можно воспользоваться кнопками-флагами. Условие же может быть помещено в представление File System, так что инсталлироваться будут только те элементы, для которых установлены соответствующие флажки.

Добавление пользовательских действий

Зачастую требуется выполнить те или иные действия до или после инсталляции. Для этого можете создать пользовательское действие, выполняемое как часть процесса инсталляции или деинсталляции. Добавление пользовательского действия влечет созда-

ние выполняемого кода и привязку к соответствующему событию инсталлятора, чтобы этот код был выполнен. Пользовательские действия применяют модель событий, аналогичную той, которая используется компонентами Windows, для связывания создаваемого вами кода с соответствующими событиями инсталлятора. Чтобы добавить пользовательское действие к событию инсталлятора, необходимо создать класс, наследующий базовый класс `Installer`. Этот базовый класс предоставляет ряд событий, для которых вы можете написать обработчики. Поскольку написание пользовательских действий инсталлятора — достаточно сложная задача, диалоговое окно Add New Item включает шаблон Installer Class в узле General. Новый класс (добавленный к проекту SharedResources) открывается с применением проектировщика компонентов, как показано на рис. 48.14.

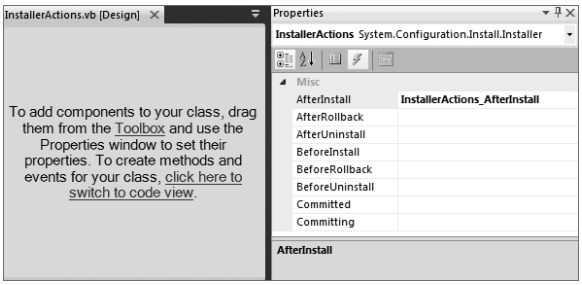


Рис. 48.14

На вкладке Events окна Properties выберите событие инсталлятора, для которого вы хотите добавить обработчик. Если обработчика события не существует, будет создан и открыт в окне редактора кода новый обработчик. Приведенный далее фрагмент кода автоматически генерируется при создании обработчика события. В него внесен код вызова окна сообщения, информирующего о завершении работы обработчика события `AfterInstall`:



C#

```
using System.ComponentModel;
using System.Configuration.Install;
using System.Windows.Forms;

public partial class InstallerActions
{
  public InstallerActions()
  {
    InitializeComponent()
  }

  private void InstallerActions_AfterInstall(
                 object sender, InstallEventArgs e)
  {
    MessageBox.Show("Процесс инсталляции завершен!");
  }
}
```

Сниппет кода SharedResources\InstallerActions.cs



VB

```
Imports System.ComponentModel
Imports System.Configuration.Install
Imports System.Windows.Forms

Public Class InstallerActions
  Public Sub New()
    MyBase.New()
    InitializeComponent()
  End Sub
```

```
  Private Sub InstallerActions_AfterInstall( _
              ByVal sender As Object, _
              ByVal e As InstallEventArgs) _
              Handles Me.AfterInstall
    MessageBox.Show("Процесс инсталляции завершен!")
  End Sub
End Class
```

Сниппет кода SharedResources\InstallerActions.vb

Как и в случае форм и других компонентов, остальная часть этого класса хранится в файле класса проектировщика. Класс `InstallerActions` наследует класс `Installer` и имеет атрибут `RunInstaller`. Это гарантирует, что класс может быть обработчиком событий, генерируемых инсталлятором.

Только что созданный класс `InstallerActions` добавлен в сборку `SharedResources`. Что касается событий, связанных с классом `InstallerActions`, то инсталлятор должен знать о классе, содержащем пользовательские действия. Для осуществления такой связи добавьте сборку `SharedResources` в представление Custom Actions проекта развертывания, щелкнув правой кнопкой мыши на любом из узлов, которые показаны на рис. 48.15, и выбрав Add Custom Action из контекстного меню. В нашем случае мы будем подключать `SharedResources`. На рис. 48.15 эта связь создана только для действия Install. Если вы хотите связать класс `CustomAction` со всеми действиями, добавьте пользовательское действие к корневому узлу Custom Actions.

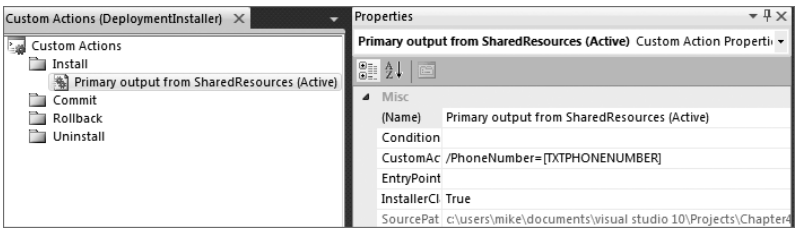


Рис. 48.15

Чтобы завершить рассмотрение, обратите внимание на важность возможности передачи пользовательскому действию информации, полученной от пользователя на этапе Start процесса инсталляции. К сожалению, поскольку пользовательское действие вызывается после завершения работы инсталлятора, необходимо применить специальный канал передачи свойств инсталлятора пользовательскому обработчику. В представлении Custom Actions (рис. 48.15) выберите команду Properties Window контекстного меню, вызываемого щелчком правой кнопкой мыши на узле Primary output. Свойство `CustomActionData` используется для определения пар "имя–значение", которые будут пересланы пользовательскому инсталлятору. Например, у вас может быть `/PhoneNumber="+14250010001"`, и в этом случае обработчик события может обратиться к указанному значению следующим образом:



C#

```
private void CustomActions_AfterInstall(object sender,
                                        InstallEventArgs e)
{
MessageBox.Show("Number: " +
         this.Context.Parameters["PhoneNumber"].ToString());
}
```

Сниппет кода SharedResources\CustomActions.cs



VB

```
Private Sub CustomActions_AfterInstall( _
                ByVal sender As Object, _
                ByVal e As InstallEventArgs) _
                Handles Me.AfterInstall
  MessageBox.Show("Number: " &
            Me.Context.Parameters("PhoneNumber").ToString)
End Sub
```

Сниппет кода SharedResources\CustomActions.vb

Конечно, жестко закодированные значения — идея не самая хорошая; было бы лучше использовать значение, указанное пользователем. Для применения свойства, определенного в пользовательском интерфейсе инсталлятора, замените указанную строку идентификатором свойства в квадратных скобках. Например, `/PhoneNumber=[TXTPHONENUMBER]` приведет к использованию текста из поля `TXTPHONENUMBER`.

Инсталлятор службы

Инсталлятор для службы Windows можно создать так же, как и для приложения. Однако инсталлятор службы Windows должен не только установить файлы в необходимые места, но и зарегистрировать службу с тем, чтобы она появилась в списке служб. Это можно сделать с помощью компонентов `ServiceInstaller` и `ServiceProcessInstaller` из пространства имен `System.ServiceProcess` (вероятно, вам придется добавить их на панель инструментов, поскольку по умолчанию они невидимы). Как показано на рис. 48.16, следует перетащить каждый из указанных компонентов на поверхность проектировщика пользовательского инсталлятора.

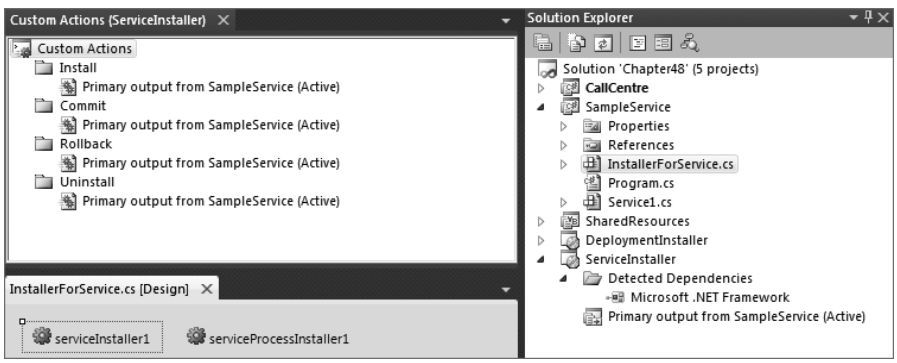


Рис. 48.16

Класс `ServiceInstaller` используется для указания выводимого имени (имени службы, которое будет выведено в списке служб Windows), имени службы (имени класса службы, который будет выполняться при запуске службы) и типа запуска (запускается ли служба вручную или автоматически при запуске Windows). Для каждой службы, которую вы хотите установить, следует создать отдельный экземпляр класса `ServiceInstaller`, указывая различные выводимые имена и имена служб. Необходим единственный экземпляр класса `ServiceProcessInstaller`, который будет использован для указания информации об учетной записи, от имени которой будет работать служба (или службы). В приведенном далее фрагменте кода конструктор

InstallerForService указывает, что класс Service1 должен быть инсталлирован как служба и что он должен автоматически запускаться с использованием учетной записи NetworkService:



C#

```
[RunInstaller(true)]
public partial class InstallerForService :
    System.Configuration.Install.Installer
{
  const string SERVICE_DISPLAY_NAME = "My Generic Service";
  const string START_AFTER_INSTALL = "STARTAFTERINSTALL";
  const string NET_PROCESS_NAME = "Net";
  const string NET_START = "Start \"{0}\"";
  const int NET_WAIT_TIMEOUT = 5000;
  const string NET_WAIT_ERROR = "ВНИМАНИЕ: процесс занял времени больше,
   чем " +"ожидалось, так что его может потребоваться перезагрузить вручную";

  public InstallerForService()
  {
    InitializeComponent();
    serviceInstaller1.DisplayName = SERVICE_DISPLAY_NAME;
    serviceInstaller1.ServiceName = typeof(Service1).ToString();
    serviceInstaller1.StartType   = ServiceStartMode.Automatic;
    serviceProcessInstaller1.Account =
                ServiceAccount.NetworkService;
  }

  private void InstallerForService_AfterInstall(
        object sender, InstallEventArgs e)
  {
    var startString = Context.Parameters[START_AFTER_INSTALL];
    if (startString == "") return;
    var shouldStart = Boolean.Parse(startString);
    if (!shouldStart) return;
    var proc = Process.Start(CreateNetStartProcessInfo());
    if (!proc.WaitForExit(NET_WAIT_TIMEOUT))
      MessageBox.Show(NET_WAIT_ERROR);
  }

  private ProcessStartInfo CreateNetStartProcessInfo()
  {
    return new ProcessStartInfo(NET_PROCESS_NAME,
            String.Format(NET_START, SERVICE_DISPLAY_NAME))
    {
      WindowStyle = ProcessWindowStyle.Hidden
    };
  }
}
```

Сниппет кода Windows Service\InstallerForService.cs



VB

```
Public Class InstallerForService
  Const SERVICE_DISPLAY_NAME = "My Generic Service"
  Const START_AFTER_INSTALL  = "STARTAFTERINSTALL"
  Const NET_PROCESS_NAME = "Net"
  Const NET_START = "Start ""{0}"""
  Const NET_WAIT_TIMEOUT = 5000
  Const NET_WAIT_ERROR = "ВНИМАНИЕ: процесс занял времени больше, чем " &
    "ожидалось, так что его может потребоваться перезагрузить вручную"
```

```
  Public Sub New()
    MyBase.New()
    ' Этот вызов требуется Component Designer.
    InitializeComponent()

    ' Добавление кода инициализации
    ' после вызова InitializeComponent
    ServiceInstaller1.DisplayName    = SERVICE_DISPLAY_NAME
    ServiceInstaller1.ServiceName    = GetType(Service1).ToString
    ServiceInstaller1.StartType      = ServiceStartMode.Automatic
    ServiceProcessInstaller1.Account =
            ServiceAccount.NetworkService
  End Sub

  Private Sub InstallerForService_AfterInstall( _
    ByVal sender As System.Object, _
    ByVal e As System.Configuration.Install.InstallEventArgs) _
    Handles MyBase.AfterInstall

    Dim startString = Context.Parameters(START_AFTER_INSTALL)
    If startString = "" Then Return

    Dim shouldStart = Boolean.Parse(startString)
    If Not shouldStart Then Return

    Dim proc = Process.Start(CreateNetStartProcessInfo())

    If Not proc.WaitForExit(NET_WAIT_TIMEOUT) Then
      MessageBox.Show(NET_WAIT_ERROR)
    End If
  End Sub

  Private Function CreateNetStartProcessInfo() _
    As ProcessStartInfo

    Dim startInfo = New ProcessStartInfo(NET_PROCESS_NAME,
        String.Format(NET_START, SERVICE_DISPLAY_NAME))
    startInfo.WindowStyle = ProcessWindowStyle.Hidden
    Return startInfo
  End Function
End Class
```

Сниппет кода Windows Service\InstallerForService.vb

В приведенный листинг включен также обработчик события `AfterInstall`, использующийся для запуска службы по завершении процесса инсталляции. По умолчанию, даже если запуск службы определен как автоматический, служба инсталлятором не запускается. Однако при деинсталляции службы инсталлятор пытается ее остановить.

Пользовательский интерфейс данного проекта развертывания включает диалоговое окно Checkboxes (A) с помощью представления проекта User Interface (см. пользовательский интерфейс по умолчанию на рис. 48.13). Щелкните правой кнопкой мыши на узле Start и выберите из контекстного меню команду Add Dialog. В окне Add Dialog выберите диалоговое окно Checkboxes (A) и щелкните на кнопке OK. Так новое диалоговое окно будет вставлено в конец процесса инсталляции. Порядок диалоговых окон может быть изменен с помощью элементов Move Up/Down контекстного меню, вызываемого щелчком правой кнопкой мыши на узлах в окне User Interface.

Выберите команду Properties Window контекстного меню, вызываемого щелчком правой кнопки мыши на новом диалоговом окне, что приведет к выводу окна Properties. Установите идентификатор свойства `Checkbox1` равным `STARTAFTERINSTALL`, а затем установите свойство `Visible` остальных переключателей равным `false`. Как говорилось ранее в этой главе, необходимо также добавить `/STARTAFTERINSTALL=[STARTAFTERINSTALL]` к свойству `CustomActionData` сборки в представлении Custom Actions проекта развертывания. Этот ввод пользователя позволит вам решить, следует ли запускать службу по завершении инсталляции.

Технология ClickOnce

Применение инсталлятора Windows — практичный подход для развертывания любого приложения. Однако распространить инсталлятор по тысячам машин, а затем потенциально их обновлять — задача нелегкая. Хотя специализированные продукты управления помогают упростить развертывание приложения, веб-приложения часто заменяют приложения Windows, поскольку могут динамически обновляться, что оказывает воздействие на всех пользователей системы. Технология *ClickOnce*, впервые вошедшая в состав .NET Framework версии 2.0, позволяет создавать самообновляющиеся приложения Windows. Из этого раздела вы узнаете, как использовать Visual Studio 2010 для создания приложений, которые могут распространяться и обновляться с применением ClickOnce.

Развертывание одним щелчком

Для демонстрации функциональности развертывания с применением ClickOnce в этом разделе используется то же приложение, что и для создания Windows Installer, CallCentre, которое выводит пустую форму. Для развертывания этого приложения с применением ClickOnce выберите команду Publish контекстного меню, вызываемого щелчком правой кнопкой мыши на проекте. После этого откроется мастер Publish Wizard, который проведет вас через начальную настройку ClickOnce для вашего проекта.

Первый шаг Publish Wizard позволяет выбрать место для развертывания приложения. Можете выбрать размещение на локальном веб-сайте, FTP-сервере, в совместно используемой папке или даже в локальной папке вашего компьютера. Щелчок на кнопке Browse приведет к открытию диалогового окна Open Web Site, которое помогает указать искомое местоположение.

На следующем шаге мастер спросит вас, откуда ожидается установка приложения пользователем. По умолчанию предполагается, что пользователь будет выполнять инсталляцию с компакт-диска или с диска DVD-ROM. Зачастую установка выполняется из совместно используемой папки в корпоративной сети или с веб-сайта в Интернете. Заметим, что места, в которых вы размещаете приложения и из которых их будут инсталлировать пользователи, могут быть различными. Это может оказаться полезным при тестировании новых выпусков.

Содержимое последнего шага будет изменяться в зависимости от выбранных настроек инсталляции. Если ваше приложение будет устанавливаться с компакт-диска или DVD-ROM, на этом шаге у вас спросят, должно ли приложение автоматически проверять наличие обновлений. Если эта настройка включена, вы должны указать проверяемое приложением местоположение. В случае инсталляции из общей папки

или с веб-сервера предполагается, что обновление будет выполняться оттуда же, откуда выполнялась инсталляция. Последний вопрос связан с доступностью приложения в автономном режиме. Если выбран автономный режим, в меню Start добавляется ярлык приложения, и оно может быть удалено с помощью диалогового окна Add/Remove programs операционной системы. Пользователь сможет запустить инсталлятор, даже если изначальное его местоположение более недоступно. Если приложение доступно только по сети, то ярлык не создается, и пользователь должен посещать местоположение установки всякий раз при запуске приложения.

Последний экран мастера позволяет проверить конфигурацию перед опубликованием приложения. После опубликования приложения вы можете запустить файл запуска `Setup.exe`, который создается для инсталляции приложения. При выбранной инсталляции с веб-сайта генерируется также файл `default.htm`. Этот файл, показанный на рис. 48.17, использует JavaScript для обнаружения зависимостей и предоставляет кнопку Install для запуска `Setup.exe`.

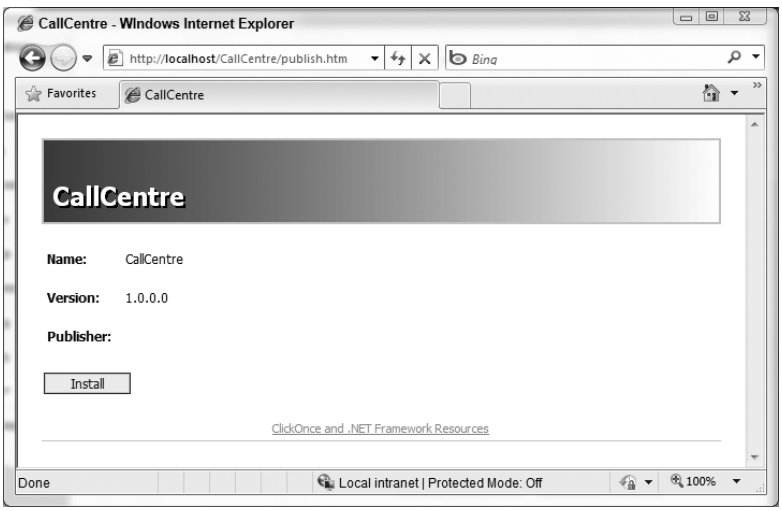


Рис. 48.17

После щелчка на кнопке Install выводится диалоговое окно, которое предлагает запустить или сохранить `Setup.exe`. Выбор команды Run (или запуск `Setup.exe` другого типа инсталлятора) приводит к появлению диалогового окна Launching Application, показанного на рис. 48.18, на то время, пока из местоположения инсталляции загружаются компоненты вашего приложения.

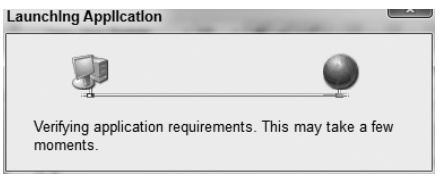


Рис. 48.18

После того как вся необходимая информация будет загружена, выводится предупреждение системы безопасности, показанное на рис. 48.19. В данном случае предупреждение связано с тем, что хотя манифест установки и подписан, но подписан он с использованием сертификата, неизвестного машине, на которой выполняется установка.

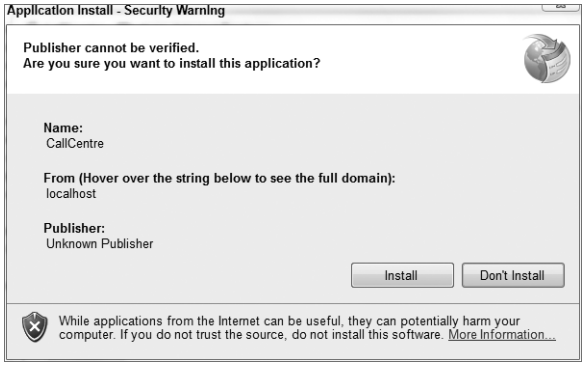


Рис. 48.19



Манифест установки приложения ClickOnce представляет собой XML-файл, описывающий устанавливаемое приложение вместе со ссылкой на текущую версию. Хотя это и не требуется, каждый манифест развертывания может быть подписан издателем, что обеспечивает манифест строгим именем. Тем самым предотвращаются возможные злоумышленные вмешательства в него.

При подписании манифеста возможны три варианта. По умолчанию система Visual Studio 2010 создает тестовый сертификат для подписания манифеста, имеющий формат *имя приложения*_TemporaryKey.pfx, и автоматически добавляет его в решение (это делается при первой публикации приложения с помощью кнопки Publish Now). Хотя этот сертификат и может использоваться в процессе разработки, для развертывания он не рекомендуется. Второй вариант состоит в покупке сертификата сторонней компании наподобие VeriSign или в использовании сервера сертификатов Windows Server для создания внутреннего сертификата.

Преимущество сертификата от хорошо известной компании заключается в том, что он может быть автоматически проверен любой машиной. Использование тестового или внутреннего сертификата требует его передачи в соответствующее хранилище сертификатов. На рис. 48.20 показана вкладка Signing окна Project Properties, на которой представлен манифест ClickOnce, подписанный с помощью сертификата, сгенерированного на локальном компьютере. Существующий сертификат можно выбрать из хранилища или из файла. В качестве альтернативы может быть создан другой тестовый сертификат.

Если вы хотите, чтобы ваше приложение инсталлировалось с известным издателем, добавьте тестовый сертификат в корневое хранилище сертификатов на компьютере, на котором инсталлируется продукт. Поскольку это также машина развертывания, добиться этого можно, щелкнув на кнопке More Details. При этом откроется диалоговое окно с детальным описанием сертификата, включая тот факт, что он не может быть аутентифицирован. (Если вы используете сертификат, созданный по умолчанию системой Visual Studio 2010, воспользуйтесь кнопкой Select from File для повторного выбора сгенерированного сертификата, а затем щелкните на кнопке More Details. Дело в том, что по непонятным причинам без этого дополнительного шага окно детальной информации не выводит кнопку Install Certificate.) Щелчок на Install Certificate позволяет указать, что этот сертификат должен быть установлен в хра-

нилище Trusted Root Certification Authorities. Это хранилище не является хранилищем по умолчанию, так что вам нужно к нему перейти. Поскольку наш сертификат тестовый, можно пока что проигнорировать предупреждение о том, что его не следует использовать в производстве, но помнить об этом следует. Теперь, когда вы опубликуете свое приложение и попытаетесь его инсталлировать, вы увидите, что диалоговое окно изменилось и выглядит так, как на рис. 48.21.

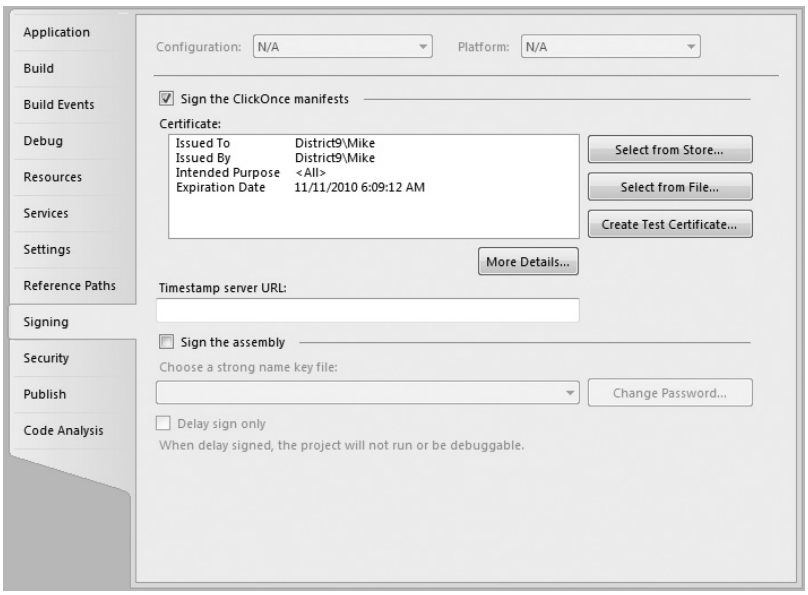


Рис. 48.20

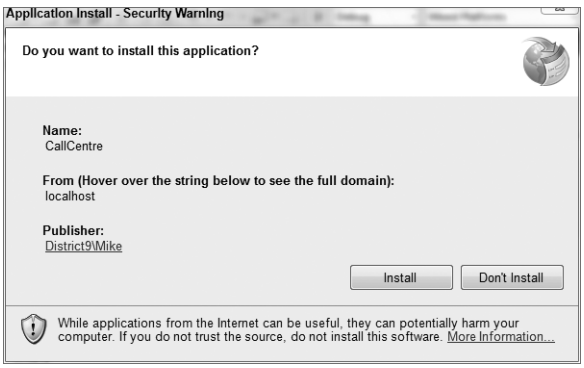


Рис. 48.21

Хотя вы и знаете издателя, вы все равно будете предупреждены о том, что приложению для работы требуются дополнительные права. Щелчок на ссылке More Information приведет к открытию диалогового окна, показанного на рис. 48.22. Здесь используется то же цветовое кодирование уровней безопасности, что и в системе Windows Server 2008 и в системе Windows 7: диалоговое окно содержит три пиктограммы: зеленую — для обеспеченной безопасности, красную — для потенциально слабой системы безопасности и желтую — для информации или советов по улучшению системы безопасности.

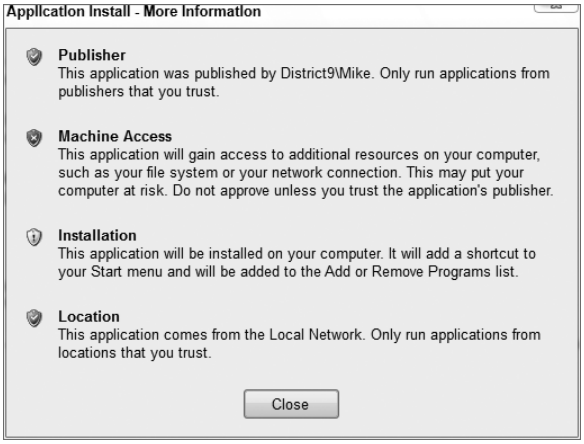


Рис. 48.22

Манифесты ClickOnce оцениваются по четырем измерениям безопасности. Только что вы узнали, как указать хорошо известного издателя, что критично для безопасной инсталляции приложения. По умолчанию ClickOnce публикует приложения как полностью проверенные, предоставляя им максимальный контроль над локальным компьютером. Это необычно, поскольку в большинстве случаев компания Microsoft на первое место ставит безопасность. Для работы с полным доступом приложению требуются дополнительные права. Приложение Sample Application будет доступно как в оперативном, так и в автономном режимах; и хотя это и не самый большой риск для системы безопасности, оно модифицирует локальную файловую систему. Наконец, место, из которого выполняется инсталляция приложения, при определении возможного уровня опасности приложения почти всегда так же важно, как и его издатель. В нашем случае приложение опубликовано в локальной сети, так что непохоже, чтобы оно представляло угрозу безопасности.

Поскольку на самом деле это приложение ничего не делает, можно снизить требующийся приложению уровень доверия (trust level). Как показано на рис. 48.23, данное приложение сделано приложением с частичным доверием, основанным на зоне Local Intranet. Это приводит к изменению пиктограммы Machine Access на зеленую, и желтой остается только одна пиктограмма — Installation. К сожалению, единственный способ сделать зеленой и ее — не разрешать инсталлировать приложение, что означает его недоступность в автономном режиме.

В идеальном случае было бы неплохо иметь возможность пропустить диалоговое окно Application Install и обеспечить приложение необходимыми правами автоматически. Это можно сделать, добавив сертификат в хранилище Trusted Publishers. Чтобы приложение могло быть автоматически инсталлировано, даже в случае хорошо известных сертификатов их следует добавить в указанное хранилище. По завершении этого действия все, что вы будете видеть после загрузки приложения, — диалоговое окно с индикатором выполнения, и никаких окон с предупреждениями наподобие показанного на рис. 48.21.

После инсталляции приложение может быть запущено путем либо возврата к URL инсталляции (рис. 48.17), либо выбора ярлыка с тем же именем, что и у приложения, из вновь созданной папки `Start Menu`.

Обновление одним щелчком

В некоторый момент в будущем вы можете внести изменения в свое приложение, например добавить кнопку в созданную ранее простую форму. ClickOnce поддерживает процесс обновления, позволяющий опубликовать новую версию вашего приложения так же, как и раньше, и существующие версии будут обновлены в следующий раз при подключении к сети. Пока вас устраивает текущий набор настроек, процесс обновления представляет собой просто процесс публикации. При использовании мас-

тера Publish Wizard для обновления существующего приложения оказываются предварительно настроенными все ранее использованные для публикации значения.

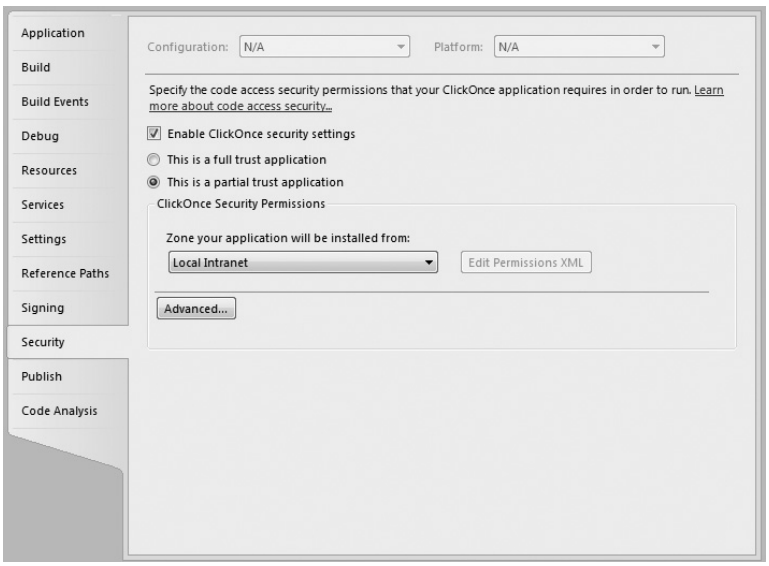


Рис. 48.23

Вы можете проверить эти настройки на вкладке Publish проектировщика Project Properties (рис. 48.24). Проектировщик показывает местоположение публикации, местоположение инсталляции, а также режим инсталляции приложения. Существует также вариант Publish Version. Это значение не демонстрируется мастером Publish Wizard, но по умолчанию номер версии начинается с 1.0.0.0 и увеличивается начиная с крайнего справа знака каждый раз при публикации приложения.

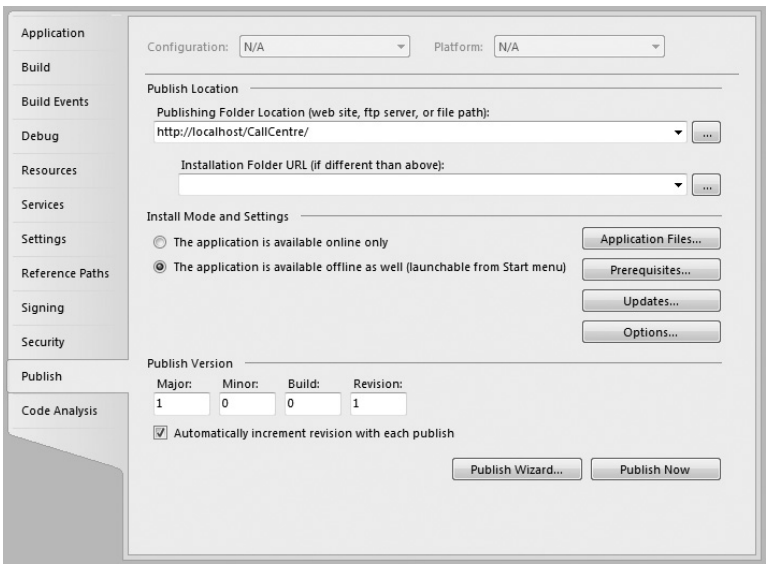


Рис. 48.24

Справа имеется ряд кнопок для более сложных настроек, большинство из которых мастером не отображается. Диалоговое окно Application Updates (рис. 48.25) позволяет настроить процесс обновления приложением самого себя. На рис. 48.25 приложение будет выполнять обновление раз в месяц после запуска. Можно также указать минимально требуемую версию для приложения, что не позволит старым клиентам работать с приложением, пока оно не будет обновлено.

При таком изменении, когда вы опубликуете новую версию приложения, всем существующим пользователям будет предложено обновить его до последней версии, как показано на 48.26.

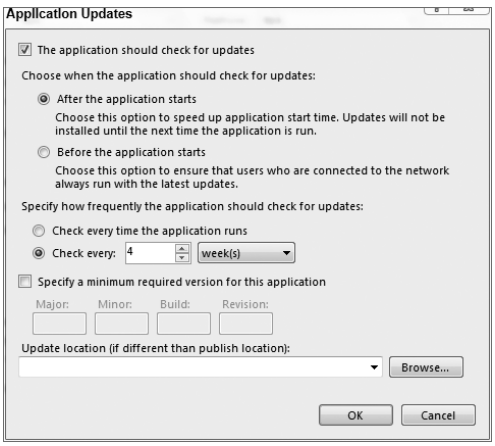


Рис. 48.25

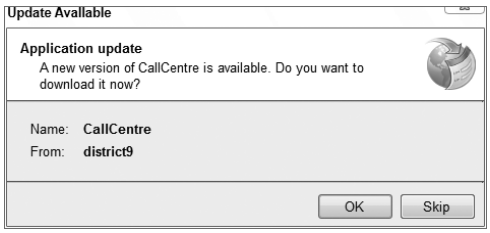


Рис. 48.26

Одна из наиболее мощных возможностей развертывания ClickOnce заключается в том, что оно отслеживает предыдущую версию инсталлированного приложения. Это означает, что на любом этапе можно не только корректно деинсталлировать приложение, но и выполнить откат к ранней версии. Приложение может быть деинсталлировано или возвращено к предыдущей версии в списке Programs and Features из Control Panel.



Заметим, что для обновления пользователи должны иметь связь с исходным URL развертывания приложения, когда оно выполняет проверку наличия новой версии (в нашем случае — при запуске приложения). Вы можете также вынудить всех пользователей выполнить обновление до определенной версии (т.е. без выводимого пользователю предложения), указав минимально необходимую версию в диалоговом окне Application Updates (рис. 48.25).

Резюме

В этой главе вы познакомились с деталями построения инсталляторов для приложений различных типов. Наличие качественного инсталлятора во многом определяет, насколько профессионально выглядит приложение. ClickOnce предлагает иной подход для тех, кто планирует распространять свое приложение среди большой аудитории пользователей. Начиная с версии платформы .NET Framework 3.5, этот метод применим для гораздо большего количества типов приложений.

49

Развертывание веб-приложений

В ЭТОЙ ГЛАВЕ...

- Публикация проектов Web Site и Web Application
- Публикация сценариев баз данных с веб-приложениями
- Копирование изменений Web Site на удаленный сервер
- Создание пакетов Web Application для развертывания с помощью Web Deployment Tool
- Управление конфигурационными файлами для сред с множественным развертыванием
- Поддержание уровня компьютера с применением Web Platform Installer
- Расширение Web Platform Installer для включения собственных приложений

Из предыдущей главы вы узнали, как распространять и устанавливать свои приложения Windows с помощью инсталляторов и технологии ClickOnce. Но как развертываются веб-приложения? В этой главе рассматривается развертывание проектов Web Site и Web Application. В ней также речь идет об упаковке веб-приложений для удаленного развертывания с помощью нового инструмента Web Deployment Tool и их интеграции с Web Platform Installer.

Одним из наиболее важных аспектов создания приложения является его упаковка для последующего развертывания. Большая часть веб-приложений предназначена для внутреннего использования, когда достаточно простого копирования сценариев. Но если вы хотите сделать свое веб-приложение доступным другим для покупки и использования, сосредоточьтесь на максимальном упрощении процесса его развертывания.



Компания Microsoft после системы Visual Studio 2005 выпустила проекты Web Deployment Projects в качестве более "продвинутого" средства управления развертыванием проектов Web Site и Web Application. При выпуске системы Visual Studio 2008 эти проекты были обновлены, но не включены в окончательный выпуск. На момент написания этой книги компания Microsoft не анонсировала никаких планов по включению проектов Web Deployment Projects в Visual Studio 2010.

Развертывание веб-сайта

Веб-проекты, создаваемые с помощью системы Visual Studio 2010, делятся на две большие категории: проекты Web Application и проекты Web Site. В этом разделе демонстрируются инструменты, предназначенные для развертывания и сопровождения проектов Web Site.

Публикация веб-сайта

Простейший способ развертывания веб-сайта, созданного в системе Visual Studio 2010, заключается в его публикации с помощью команды Publish Web Site меню Build. После выбора этой команды появится диалоговое окно, показанное на рис. 49.1. В нем есть лишь несколько базовых настроек, которые позволяют опубликовать отладочную информацию, выполнить обновление веб-сайта "на месте", а также обеспечивают применение различных стратегий именования и требований безопасности. Обычно используется только поле ввода Target Location, в котором указывается местоположение публикации. Это место может быть локальным экземпляром IIS, FTP-сайтом, некоторым местом в файловой системе или удаленным экземпляром IIS. Щелчок на кнопке с эллипсисом возле поля Target Location на рис. 49.1 приводит к вызову диалогового окна для указания конкретного места расположения публикации, показанное на рис. 49.2.

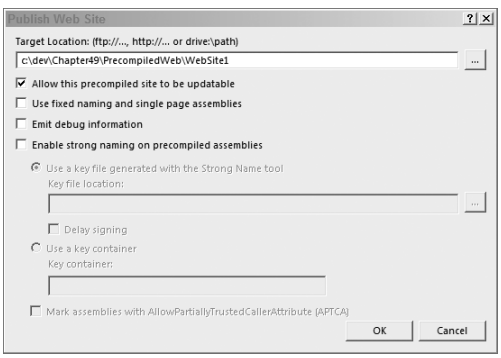


Рис. 49.1

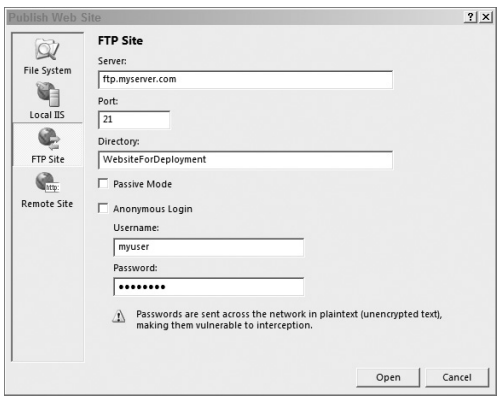


Рис. 49.2



Здесь показана публикация с применением персональной учетной записи FTP, и если это первая публикация данного сайта, для его корректного функционирования может потребоваться определить эту папку как приложение IIS.

Копирование веб-сайта

После того как веб-сайт опубликован, важно его обновить. Один из вариантов обновления состоит в повторном выполнении его публикации. Однако при этом будет опубликован весь веб-сайт, даже если обновление коснулось единственного файла. Альтернативой полной публикации является использование инструмента Copy Web Site (рис. 49.3) для синхронизации файлов между вашим проектом и веб-сайтом. Можете обратиться к этому инструменту из контекстного меню, вызываемого щелчком правой кнопкой мыши в Solution Explorer, либо через меню веб-сайта.

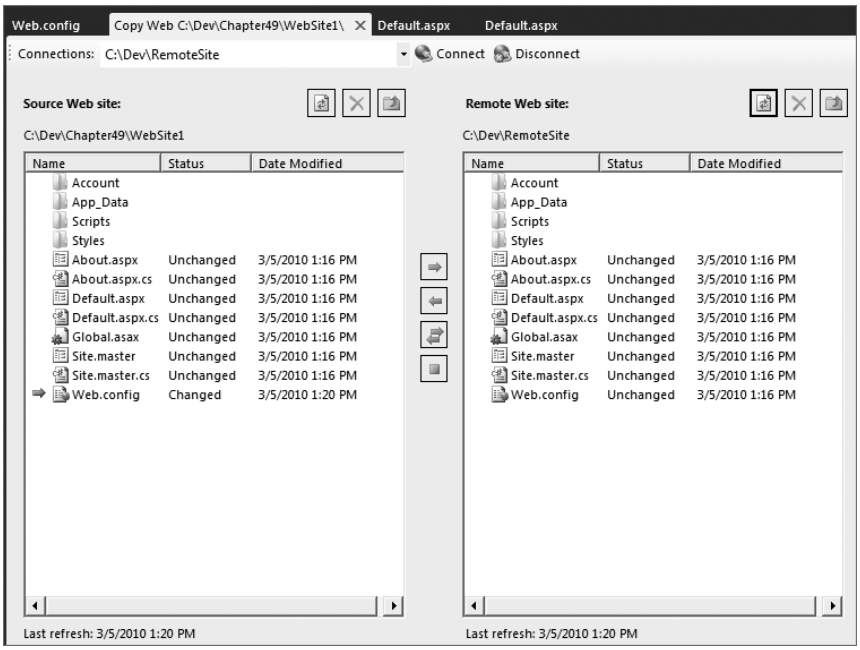


Рис. 49.3

Чтобы просмотреть имеющиеся на удаленном сайте файлы, либо выберите подключение из числа последних, выводимых в соответствующем раскрывающемся списке, либо щелкните на кнопке Connect. После этого откроется диалоговое окно, подобное показанному на рис. 49.2, в котором вы можете указать, как подключиться к удаленному веб-сайту. После подключения можно увидеть, какие файлы не синхронизированы, и воспользоваться левой и правой стрелками для того, чтобы переместить файлы между вашим локальным проектом и удаленным веб-сайтом.

Развертывание веб-приложения

Проекты веб-приложений существенно отличаются от проектов веб-сайтов и имеют свой набор инструментов для развертывания. В системе Visual Studio 2010 появилась возможность развертывания с помощью нового инструмента Web Deployment Tool, который используется для упрощения импорта и экспорта приложений IIS вместе с их зависимостями — такими, как метаданные IIS и базы данных — из командной строки, кон-

соли управления IIS 7.0, встроенных команд (cmdlets) оболочки Powershell или непосредственно из системы Visual Studio. В нем имеется также возможность управления несколькими версиями конфигурационных данных для различных сред без дублирования.

Публикация веб-приложения

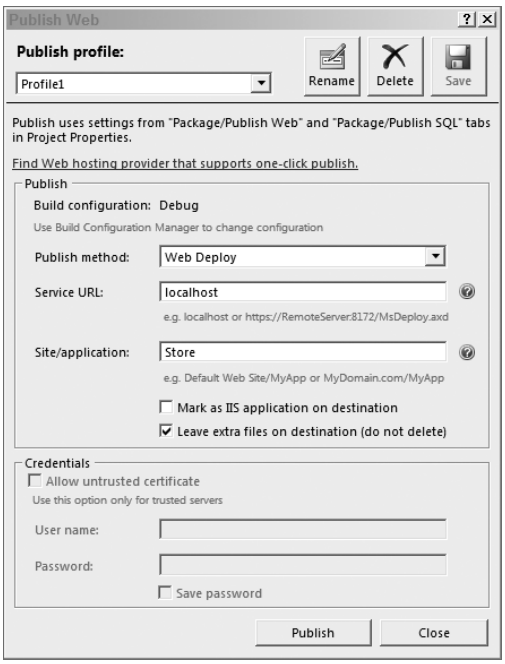


Рис. 49.4

Наиболее быстрый способ развертывания проекта Web Application заключается в простой публикации непосредственно из системы Visual Studio. Выберите команду Publish контекстного меню, вызываемого щелчком правой кнопкой мыши в окне Solution Explorer, для вывода показанного на рис. 49.4 диалогового окна Publish Web. При первом запуске диалогового окна Publish Web управляющий элемент Publish Profile представляет собой поле ввода (а не раскрывающийся список), позволяющее дать профилю значащее имя. Всякий раз при развертывании это делается с определенным профилем, который инкапсулирует настройки целевой среды. Проект Web Application может поддерживать коллекцию профилей, которая позволяет развернуть одно веб-приложение в ряде целевых сред, храня при этом настройки для каждой среды отдельно.

Четыре варианта выбора Publish Method определяют, что вы увидите в нижней части диалогового окна: Web Deploy FTP, File System или Front-Page Server Extensions (FPSE). File System и Front-Page Server Extensions позволяют ввести целевое местоположение публикации веб-приложения. Вариант с FTP позволяет сделать то же самое, но при этом добавляются параметры учетной записи для FTP. Все они после щелчка на кнопке с эллипсисом предоставляют урезанную версию рис. 49.2. Единственной дополнительной настройкой является выбор между простой заменой файлов в целевом местоположении и полным удалением всех файлов из целевого местоположения перед развертыванием.

Последний вариант метода публикации — инструмент `MSDeploy.exe` (известный также как Web Deployment Tool), призванный помогать администраторам и разработчикам более просто упаковывать и развертывать проекты веб-приложений. Это выполняется путем упаковки всех необходимых для вашего приложения файлов вместе со всеми необходимыми для инсталляции и конфигурирования метаданными в единый zip-файл. Затем этот файл может быть установлен с помощью интерфейса IIS7.0, из командной строки, с применением встроенных команд оболочки Powershell или непосредственно из самой системы Visual Studio.

Для развертывания непосредственно на сервер с работающим инструментом MSDeploy необходима следующая информация.

- Ссылка на службу, прослушивающую инструмент MSDeploy
- Сайт и имя приложения, использующиеся для идентификации удаленного приложения
- Имя пользователя и пароль для удаленного экземпляра инструмента MSDeploy

Вся эта информация предоставляется администратором вашего сервера или провайдером хостинга.

После по меньшей мере одного успешного опубликования вашего приложения станет доступной панель инструментов Web One Click Publish. Эта панель, показанная на рис. 49.5, позволяет выбрать профиль публикации и выполнить публикацию одним щелчком на кнопке Publish. Если панель инструментов не видна, включите ее из главного меню, выбрав View⇨Toolbars⇨Web One Click Publish.



Рис. 49.5



При запуске процесса публикации из главного меню или из контекстного меню, вызываемого щелчком правой кнопкой мыши, вам до развертывания приложения будет предоставлена возможность ознакомиться со свойствами профиля. Однако при использовании панели инструментов Web One Click Publish развертывание начинается немедленно.

Упаковка веб-приложения

Даже если у вас нет непосредственного доступа к целевой среде, вы все равно можете создавать пакеты, которыми можно управлять посредством Web Deployment Tool. Это так же просто, как выбрать команду Create Package контекстного меню, вызываемого щелчком правой кнопкой мыши на проекте Web Application в окне Solution Explorer.

По умолчанию при этом в папке `obj\Debug\Package\` генерируются zip-файл и файлы поддержки, включающие `.cmd`-файл с файлом параметров, которые могут использоваться для инсталляции пакета из командной строки на целевом сервере.



Эти пакеты представляют собой в точности то, что развертывается на удаленном сервере при использовании инструмента MSDeploy.

Настройка пакетов веб-приложений

Важным аспектом развертывания и упаковки веб-приложений является настройка того, что было развернуто. Получить показанную на рис. 49.6 вкладку Package/Publish Web можно как на обычной странице свойств проекта, так и выбрав команду Package/Publish Settings из контекстного меню, вызываемого щелчком правой кнопкой мыши на проекте в окне Solution Explorer.

Первая настройка определяет, какие именно файлы должны быть развернуты. Имеется три варианта.

- **Only files needed to run the application.** Включает только те файлы, которые действительно необходимы для работы приложения, исключая файлы, для которых отсутствует действие сборки. Эта наиболее “исключающая” настройка используется по умолчанию.

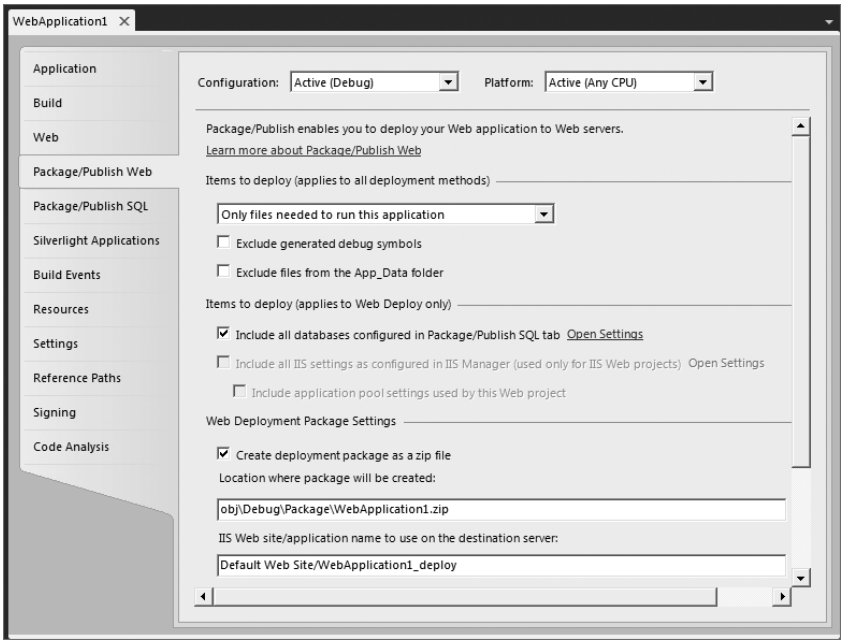


Рис. 49.6

- **All files in this project.** Включает все файлы, которые являются частью проекта в окне Solution Explorer.
- **All files in this project folder.** Включает все файлы, которые имеются в папке проекта, даже если они не являются частью такового. Это наименее ограничивающий выбор.

Имеется также возможность исключить генерируемые файлы с отладочной информацией и файлы в папке `App_Data`. Для развертывания баз данных используйте настройки Package/Publish SQL, которые будут рассмотрены в следующем разделе.

Инструмент Web Deployment Package может выполнить копирование метаданных IIS, если только ваш проект разрабатывался на сервере IIS. Для выполнения соответствующей установки перейдите на страницу Web properties и измените настройку Servers на Use Local IIS Web Server.

Наконец можете указать необходимость создания распространяемого пакета и его местоположение, а также указать веб-сайт/имя приложения для пакета при его установке на сервер IIS.

Упаковка данных SQL Server

Распространенной проблемой при развертывании веб-приложений является то, что они зачастую используют сервер баз данных для хранения данных, так что процесс установки должен найти способ управления развертыванием схемы базы данных, а потенциально — и самих данных. Инструмент Web Deployment Packages позволяет указать сценарии базы данных для настройки или обновления ваших баз данных, а инструмент Web Deployment Tool управляет процессом запуска сценариев на сервере баз данных в процессе установки.

Чтобы добавить в распространяемый пакет базу данных, воспользуйтесь страницей свойств Package/Publish SQL, показанной на рис. 49.7. Кнопки Add и Remove в правой части обеспечивают поддержку списка развертываемых подключений. Если у вас уже имеется набор строк подключения в файле `web.config`, можете их импортировать непосредственно в указанный список с помощью кнопки Import from Web.Config. Можно также воспользоваться флажками Deploy для того, чтобы решить, следует ли включать в пакет ту или иную строку подключения.

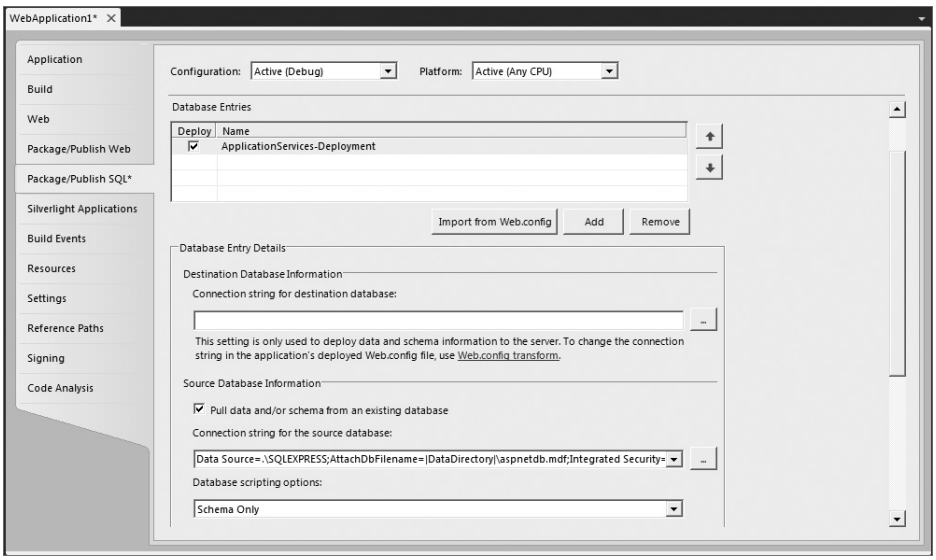


Рис. 49.7

При наличии в верхней части выделенной строки подключения, в нижней части выводятся связанные с ней детали развертывания. Первая настройка позволяет предоставить для пакета развертывания другую строку подключения.

Настройка, следующая за ней, позволяет воспользоваться при создании пакета развертывания схемой и/или данными из существующей базы данных. Для этого просто нужно предоставить строку подключения и выбрать развертываемую схему и/или данные. В дополнение к автоматически генерируемым сценариям базы данных можете добавить собственные сценарии. Это может оказаться полезным при использовании инструмента, который создает сценарии изменения вашей базы данных.



При использовании системы Visual Studio для извлечения информации о схеме или данных из существующей базы данных SQL Server генерируются сценарии для развертывания новой базы данных. Для обновления существующей базы данных создайте собственные сценарии и добавьте их в пакет.

Функциональная возможность Web.config Transformation

Еще одной проблемой для многих проектов оказывается поддержка различных версий конфигурации приложения для разных сред. Обычно эта проблема решается с помощью нескольких копий файла `web.config` — по одной для каждой среды —

и переименования при развертывании приложения файла с подходящей для данной среды версией. Даже если вы сделаете такой подход частью процесса создания приложения, проблемы все равно останутся — хотя бы потому, что любое изменение файла `web.config` потенциально должно быть внесено в каждый из конфигурационных файлов для разных поддерживаемых сред. Система Visual Studio 2010 оснащена новой функциональной возможностью, Web.config Transformation, которая позволяет иметь единый конфигурационный файл и указывать для каждой среды только специфичные для нее отличия.

При первоначальном создании проект приложения включает `web.config` с двумя преобразованиями, по одному для каждой конфигурации сборки — Debug и Release. Каждое преобразование связано с единым конфигурационным файлом и располагается в окне Solution Explorer под файлом `web.config` (рис. 49.8). Для создания собственной конфигурации выберите из выпадающего меню Configuration команду Configuration Manager, а затем из раскрывающегося списка Active solution configuration — команду New. Введите имя своей новой конфигурации и (необязательно) выберите, из какой более ранней конфигурации должны быть скопированы ее значения.



Рис. 49.8

После того как вы получите новую конфигурацию, новое преобразование можно добавить путем выбора команды Add Config Transforms контекстного меню, вызываемого щелчком правой кнопкой мыши на файле `web.config` в окне Solution Explorer. При этом будут автоматически добавлены все отсутствующие преобразования из коллекции конфигураций решения.

Файл преобразования конфигурации представляет собой стандартный конфигурационный файл, но с добавленным новым пространством имен XML для указания преобразований. Это пространство имен имеет два атрибута, `Transform` и `Locator`, которые могут быть добавлены к любому элементу конфигурационного файла. `Transform` используется для указания изменения, которое будет сделано в исходном конфигурационном файле. `Locator` же применяется для определения конкретных узлов, к которым будут применяться преобразования. Вот несколько примеров вместе с описанием того, что они делают. Более подробную информацию можно найти в справочном руководстве библиотеки MSDN.

```
<appSettings>
    <add xdt:Transform="Remove"xdt:Locator="Condition(@name=
                                                ' UseMockDatabase ' )"/>
</appSettings>
```

Здесь выполняются поиск и удаление из выходного конфигурационного файла узла `add` в элементе `appSettings` с атрибутом `name`, равным `UseMockDatabase`.

```
<connectionStrings>
  <add name="AdventureWorks"
  xdt:Transform="Replace"
  xdt:Locator="Match(name)"
  connectionString="Data Source=UAT_DB;
  Initial Catalog=AdventureWorks; ~CA
  Integrated Security=true"/>
</connectionStrings>
```

Здесь выполняется обновление строки подключения `AdventureWorks` к серверу баз данных UAT.

```
<system.web>
  <compilation xdt:Transform="RemoveAttributes(debug)" />
</system.web>
```

Здесь из узла compilation элемента system.web удаляется атрибут debug.

```
<authorization>
  <deny users=" * " xdt:Transform="Insert"/>
</authorization>
```

Здесь в нижнюю часть элемента authorization выполняется вставка нового узла, запрещающего доступ ко всем пользователям.

Инсталляторы веб-проектов

В дополнение к инструменту Web Deployment Tool можно создать стандартный пакет Windows Installer для управления развертыванием веб-приложения или веб-сайта. Для этого нужно создать проект Web Setup Project из категории Other Project Types Setup и узел Deployment в диалоговом окне New Project. По сути, это обычный проект настройки, который конфигурируется с соответствующей выходной папкой (см. представление File System) и пользовательским интерфейсом (см. представление User Interface) для развертывания веб-приложения. После создания упомянутого проекта добавьте соответствующие выходы проекта. В левой части рис. 49.9 показано диалоговое окно Add Project Output Group для добавления в проект веб-сайта. Поскольку веб-сайт содержит только файлы содержимого, это единственная доступная в этом диалоговом окне настройка.

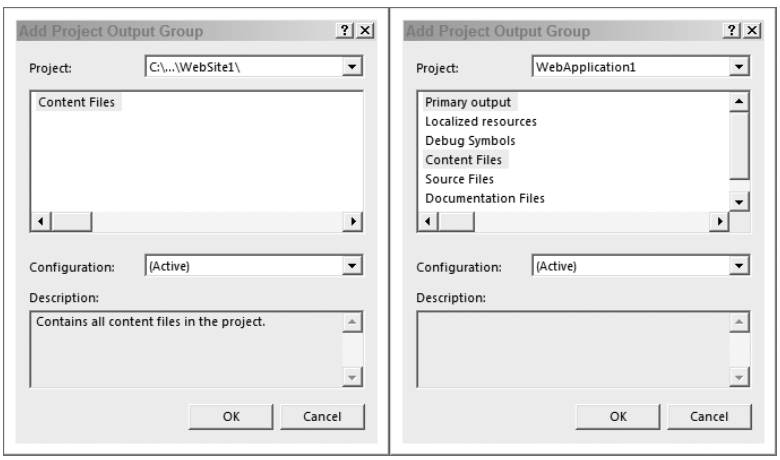


Рис. 49.9

Напротив, при использовании Web Application Project вы можете выбирать как основной выход проекта, так и файлы содержимого, как показано в правой части рис. 49.9.

Одной из уникальных возможностей инструмента Web Setup Project является экран, который добавляется к пользовательскому интерфейсу для того, чтобы позволить пользователю определить веб-сайт, виртуальный каталог и пул приложения, которые будут использоваться инсталлируемым приложением. На рис. 49.10 показано это диалоговое окно, являющееся частью процесса инсталляции.

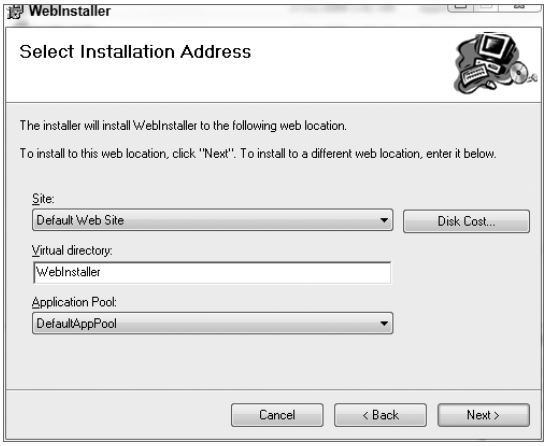


Рис. 49.10

В старых версиях системы Visual Studio мастер установки веб-сайта запрашивал у пользователя только имя виртуального каталога, в который инсталлировалось приложение, и затем этот каталог создавался на веб-сайте по умолчанию. Если на одном и том же сервере находится несколько веб-сайтов (что довольно распространено для таких продуктов, как SharePoint), то это могло привести к тому, что приложение устанавливалось не на тот веб-сайт, для которого предназначалось. Возможность указать веб-сайт в процессе инсталляции снижает объем необходимого постинсталляционного администрирования, которое требовалось в прошлом.

Программа Web Platform Installer

Зачастую для корректной работы веб-приложения используется множество технологий и инструментов — как в процессе работы приложения, так и при его разработке. Даже если ваша среда корректно настроена для одного приложения, следует понимать его и уметь управлять его связями и зависимостями. Наконец, всегда имеются новые инструменты, библиотеки и приложения, доступные в интернете, которые вы можете собрать при создании собственных проектов. По мере того как ваша среда становится все более сложной, поддерживать ее работоспособной и обновленной становится все труднее.

Программа Microsoft Web Platform Installer, показанная на рис. 49.11, представляет собой простой инструмент, разработанный для управления программным обеспечением, установленным на ваших веб-серверах и машине для разработки.

После загрузки программы Web Platform Installer с адреса `http://www.microsoft.com/web` вы можете запускать его столько раз, сколько вам нужно. Этот инструмент в состоянии обнаружить, какие компоненты уже установлены на вашей машине, и вы можете устанавливать и снимать соответствующие флажки для добавления и удаления соответствующих компонентов. Имеется возможность учета зависимостей между компонентами и установка всего необходимого.

Программа Web Platform Installer способна управлять компонентами и за пределами компонента Web Platform. Доступна также коллекция приложений Microsoft Web Application Gallery, которую можно найти по адресу `http://www.microsoft.com/web/gallery`. Эти приложения указаны в разных категориях на вкладке Web Applications. Подобно компонентам Web Platform, эти приложения могут иметь собственные предварительные условия, и программа Web Platform Installer примет меры по их обеспечению при установке.

Если вы уже упаковали свое приложение для развертывания с помощью инструмента MSDeploy, оно готово и для развертывания с использованием программы Web Platform Installer. Ваше приложение может быть добавлено в коллекцию Web Applica-

tion Gallery, если вы заполните соответствующую форму на веб-портале Microsoft. Если ваше приложение получит одобрение, оно будет готово к инсталляции на любой машине с установленной программой Web Platform Installer.

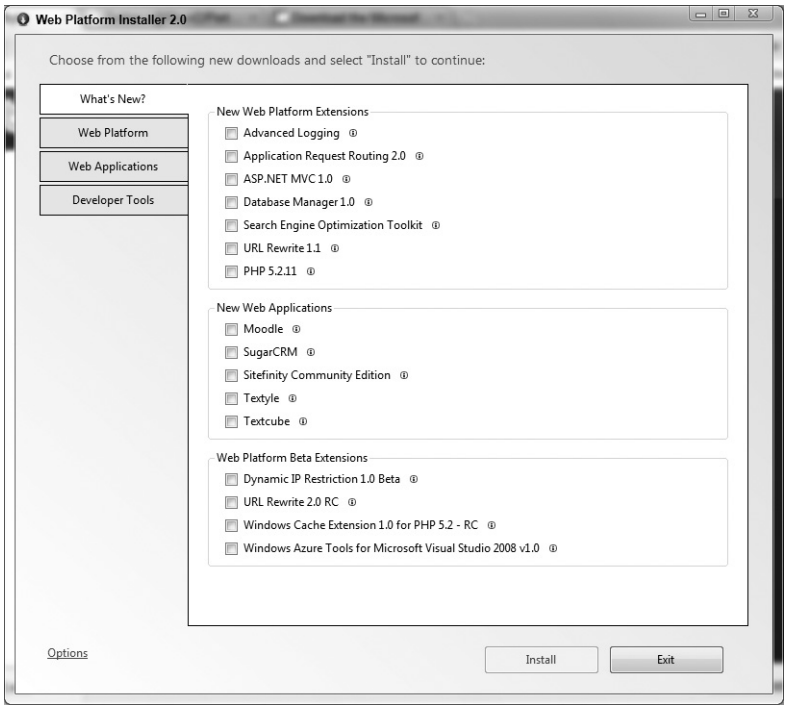


Рис. 49.11

Расширение Web Platform Installer

Как упоминалось в предыдущем разделе, можно достаточно просто включить приложение в коллекцию Web Application Gallery, чтобы сделать его доступным широкой аудитории. Имеется ряд сценариев использования преимуществ программы Web Platform Installer, не делая при этом свое приложение общедоступным. Это может быть связано с применением приложения только в рамках вашей компании, с неготовностью к окончательному выпуску и желанием протестировать процедуру развертывания.

Программа Web Platform Installer для пополнения списка инсталлируемых компонентов и продуктов последними данными использует ленту новостей в формате Atom. Каждая запись такой ленты соответствует приложению или компоненту в пользовательском интерфейсе Web Platform Installer. Вкладки Web Platform и Web Application обновляются разными лентами — по адресам `http://www.microsoft.com/web/webpi/2.0/WebProductList.xml` и `http://www.microsoft.com/web/webpi/2.0/WebApplicationList.xml` соответственно. В дополнение к этим двум лентам каждая инсталляция программы Web Platform Installer может указывать дополнительные ленты со ссылками на другие компоненты.

Вот пример ленты для простого веб-приложения хронометражного листа:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <version>1.0.0</version>
  <title>AdventureWorks Product WebPI Feed</title>
  <link href="http://www.professionalvisualstudio.com/SampleProductFeed.xml"/>
  <updated>2009-11-01T16:30:00Z</updated>
  <author>
    <name>Adventure Works</name>
    <uri>http://www.professionalvisualstudio.com</uri>
  </author>
  <id>http://www.professionalvisualstudio.com/SampleProductFeed.xml</id>

  <entry>
    <productId>TimeSheets</productId>
    <title>Adventure Works Timesheets</title>

    <summary>The Adventure Works corporate Timesheeting system</summary>
    <longSummary>The Adventure Works corporate Timesheeting system</longSummary>
    <productFamily>Human Resources</productFamily>

    <version>1.0.0</version>
    <images>
      <icon>c:\AdventureWorksIcon.png</icon>
    </images>
    <author>
      <name>Adventure Works IT</name>
      <uri>http://www.professionalvisualstudio.com</uri>
    </author>
    <published>2009-11-01T12:30:00Z</published>

    <discoveryHint>
      <or>
        <discoveryHint>
          <registry>

<keyPath>HKEY_LOCAL_MACHINE\SOFTWARE\AdventureWorks\Timesheets</keyPath>
            <valueName>Version</valueName>
            <valueValue>1.0.0</valueValue>
          </registry>
        </discoveryHint>
        <discoveryHint>
          <file>
            <filePath>%ProgramFiles%\AdventureWorks\Timesheets.exe</filePath>
          </file>
        </discoveryHint>
      </or>
    </discoveryHint>

    <dependency>
      <productId>IISManagementConsole</productId>
    </dependency>

    <installers>
      <installer>
        <id>1</id>
        <languageId>en</languageId>
        <architectures>
          <x86/>
        </architectures>
        <osList>
          <os>
            <!-- the product is supported on Vista/Windows Server SP1 + -->
            <minimumVersion>
```

```
              <osMajorVersion>6</osMajorVersion>
              <osMinorVersion>0</osMinorVersion>
              <spMajorVersion>0</spMajorVersion>
            </minimumVersion>
            <osTypes>
              <Server/>
              <HomePremium/>
              <Ultimate/>
              <Enterprise/>
              <Business/>
            </osTypes>
          </os>
        </osList>
        <eulaURL>http://www.professionalvisualstudio.com/eula.html</eulaURL>

        <installerFile>
          <!-- size in KBs -->
          <fileSize>1024</fileSize>

<installerURL>http://www.professionalvisualstudio.com/Timesheets_x86.msi
          </installerURL>
          <sha1>111222FFF000BBB444555EEEAAA777888999DDDD</sha1>
        </installerFile>

        <installCommands>
          <msiInstall>
            <msi>%InstallerFile%</msi>
          </msiInstall>
        </installCommands>
      </installer>
    </installers>
  </entry>

  <tabs>
    <tab>
      <groupTab>
        <id>AdventureWorksHRTab</id>
        <name>Adventure Works Human Resources</name>
        <description>Adventure Works HR Apps</description>
        <groupingId>HRProductFamilyGrouping</groupingId>
      </groupTab>
    </tab>
  </tabs>

  <groupings>
    <grouping>
      <id>HRProductFamilyGrouping</id>
      <attribute>productFamily</attribute>
      <include>
        <item>Human Resources</item>
      </include>
    </grouping>
  </groupings>
</feed>
```

В первой части указывается стандартная информация о самой ленте, включая дату последнего обновления и информацию об авторе. Эта информация нужна для того, чтобы лента могла быть без проблем воспринята программой для чтения новостных лент. Далее следует единственный узел `entry`, содержащий информацию о самом приложении. Программа Web Platform Installer в состоянии использовать значение

productId для ссылок на данное приложение в других местах, включая указание его в качестве зависимости для других компонентов.

Узел discoveryHint используется для определения того, установлено ли данное приложение. В данном случае определить, установлено ли приложение, можно либо по определенному значению ключа системного реестра, либо по наличию файла с определенным именем. Если найден хотя бы один из указанных элементов, Web Platform Installer считает, что данное приложение уже установлено. В дополнение к этим двум типам подсказки можно использовать подсказку msiProductCode для проверки наличия приложений, устанавливаемых инсталлятором MSI.

Данное простое приложение хронометражного листа зависит от консоли IIS Management Console. Каждый компонент, требующийся вашему приложению, может быть указан с помощью его значения productId. Если этот компонент на целевой машине не установлен, программа Web Platform Installer его инсталлирует. В дополнение к зависимостям можно указать значение incompatibilities для своего приложения, которое предотвращает одновременную установку несовместимых приложений.

Последний компонент записи для приложения — элемент installers. Для каждого инсталлятора, который вы хотите предоставлять пользователям, должен быть один элемент installer, и все такие элементы должны иметь разные идентификаторы. Каждый инсталлятор может быть нацелен на определенный диапазон языков, операционных систем и архитектур CPU. Если целевая среда не попадает в указанный диапазон, инсталлятор пользователю показан не будет. Каждая запись инсталлятора должна определять файл инсталлятора, который будет загружен в локальный кэш перед тем, как с ним будет выполнена команда installCommands.



Файл инсталлятора требует указания размера и хеш-значения SHA1, чтобы программа Web Platform Installer могла убедиться в том, что файл загружен корректно. Компания Microsoft предоставляет инструмент File Checksum Integrity Verifier (fciv.exe), который можно использовать для генерации указанного хеш-значения. Найти и загрузить этот инструмент можно по адресу http://download.microsoft.com.

Последние два элемента указывают, что будет выведено в пользовательском интерфейсе программы Web Platform Installer. Каждый элемент tab добавляется к списку вкладок в левой части окна. В приведенном примере мы добавляем вкладку для группы продуктов, определенной ниже, в элементе groupings, используя атрибут productFamily.

Для добавления ленты в экземпляр Web Platform Installer щелкните на ссылке Options в нижнем левом углу, и откроется страница Options. Введите идентификатор URL ленты в текстовое поле и щелкните на кнопке Add Feed. После щелчка на кнопке OK Web Platform Installer обновит все ленты и перезагрузит все приложения, включая новое приложение Adventure Works, как показано на рис. 49.12.

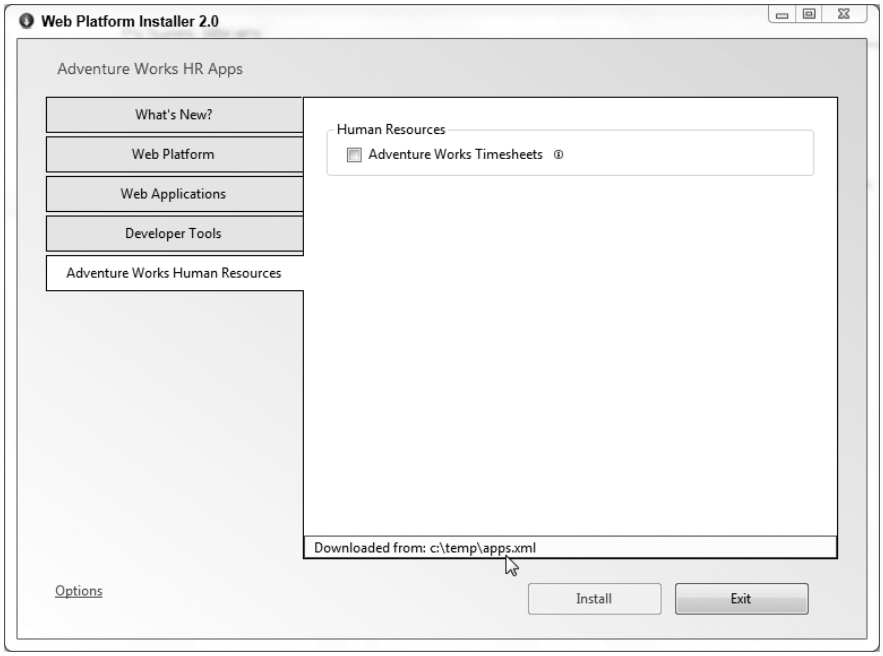


Рис. 49.12

Резюме

Из этой главы вы узнали, как использовать возможности системы Visual Studio 2010 для упаковки веб-приложений, предназначенных для развертывания. Новый инструмент Web Deployment Tool делает развертывание для разных сред и машин быстрым и безболезненным, а панель инструментов One Click Publish позволяет легко управлять разными профилями публикаций. Наконец, программа Web Platform Installer предоставляет простой способ как донести свои приложения большому количеству потенциальных клиентов, так и управлять собственным набором приложений уровня предприятия.

ЧАСТЬ XI

Настройка и расширение системы Visual Studio

- **ГЛАВА 50.** Модель автоматизации
- **ГЛАВА 51.** Надстройки
- **ГЛАВА 52.** Макросы
- **ГЛАВА 53.** Библиотека Managed Extensibility Framework (MEF)

50

Модель автоматизации

В ЭТОЙ ГЛАВЕ...

- Настройки расширений системы Visual Studio
- Работа с моделью автоматизации системы Visual Studio

Зачастую, работая в системе Visual Studio, приходится выполнять повторяющиеся действия, которые хотелось бы объединить в единый автоматически выполняющийся пакет. Такая возможность могла бы упростить выполнение сложных задач, оптимизировать работу, уменьшить неприятные ощущения от неинтеллектуальной монотонной работы и в конечном итоге повысить производительность вашего труда. Возможно, вы также хотели бы добавить новую функциональную возможность в среду Visual Studio и поделиться ею с другими разработчиками в вашей компании (а может, и во всем мире). К счастью, система Visual Studio обладает большими возможностями расширения — фактически множество возможностей системы Visual Studio, которые вы считаете встроенными, на самом деле представляют собой расширения! Такая расширяемость позволяет легко добавлять в систему Visual Studio новую функциональную возможность, отвечающую вашим требованиям. К таким возможностям относятся автоматизация задач, добавление новых инструментальных окон, новых функций в редактор кода и новых пунктов меню (включая команды контекстного меню редактора кода), создание визуализаторов отладки, собственных мастеров, расширение существующих диалоговых окон и даже добавление собственных редакторов/проектировщиков и языков программирования! В этой главе рассматриваются возможности расширения системы Visual Studio и модель автоматизации, используемая как макросами, так и надстройками системы Visual Studio.

Варианты расширений системы Visual Studio

К сожалению, история о расширениях системы Visual Studio достаточно печальна, поскольку имеется множество различных средств расширения Visual Studio, и иногда

сложно определить, какой метод должен использоваться для достижения поставленной цели. Ниже перечислены варианты расширений системы Visual Studio и контекст, в котором они наиболее часто применяются.

- **Макросы.** Представляют собой простейший способ автоматизации системы Visual Studio и могут рассматриваться как язык сценариев интегрированной среды разработки. Макросы лучше всего подходят для быстрой автоматизации задач (таких, как работа с текстом в редакторе кода или автоматизация повторяющегося набора задач в интегрированной среде разработки). Возможности макросов весьма ограничены и подходят для решения только простых задач автоматизации. В качестве языка макросов поддерживается только VB. Макросы могут совместно использоваться разработчиками, но при условии совместного использования файлов макросов (включая их исходный текст). Разработке макросов для системы Visual Studio 2010 посвящена глава 52.
- **Надстройки** (add-ins). Более мощные по сравнению с макросами (несмотря на то, что оба работают с моделью автоматизации системы Visual Studio) и позволяют создавать также окна инструментов и мастера и интегрировать другие возможности в самой интегрированной среде разработки. Надстройки являются компилируемыми проектами (на предпочитаемом вами языке программирования платформы .NET или Visual C++), так что вы можете передавать другим разработчикам бинарный код, а не исходные тексты. Разработке надстроек для Visual Studio 2010 посвящена глава 51.
- **Пакеты VSPackages.** Представляют собой часть пакета Visual Studio SDK (загружается и инсталлируется отдельно) и предоставляют еще более мощные возможности, чем надстройки. Пакеты VSPackages позволяют обратиться ко внутренним интерфейсам ядра системы Visual Studio, а потому идеально подходят для интеграции в нее собственных редакторов, проектировщиков и языков программирования. Однако рассмотрение пакетов VSPackages выходит за рамки данной книги. Дополнительную информацию по пакетам VSPackages можно найти в книге Кейвана Наири (Keyvan Nayyeri) *Professional Visual Studio Extensibility*.
- **Части компонентов Managed Extensibility Framework (MEF).** Позволяют расширять новый редактор кода системы Visual Studio 2010 на основе технологии WPF, изменяя его внешний вид и поведение. Если вы хотите добавить возможности в редактор кода, то это лучший выбор. О том, как разрабатывать расширения редактора кода в системе Visual Studio 2010, рассказывается в главе 53.

В нескольких следующих главах речь идет о способах расширения системы Visual Studio, включая применение надстроек, макросов и механизм Managed Extensibility Framework (MEF). Однако в этой главе мы продолжим рассмотрение модели автоматизации системы Visual Studio 2010, с которой работают как макросы, так и надстройки.

Модель автоматизации системы Visual Studio

Модель автоматизации системы Visual Studio, известная также как расширение инструментов разработки (Development Tools Extensibility — DTE), является объектной моделью системы Visual Studio, которая может использоваться для взаимодействия с интегрированной средой разработки. Эта объектная модель позволяет выполнять в системе Visual Studio многие действия, обеспечивающие желаемое поведение, обра-

ботку событий, генерируемых системой Visual Studio (например, события активизации некоторой команды), и многие другие функции, такие как вывод пользовательских окон инструментов в интегрированной среде разработки Visual Studio.

Эта объектная модель является средством взаимодействия надстроек и макросов с интегрированной средой разработки Visual Studio, так что в этом разделе будут более глубоко рассмотрены ее структура и предоставляемая его функциональность.

Обзор модели автоматизации

Модель автоматизации системы Visual Studio (DTE) представляет собой основанную на COM-технологии объектную модель, обновляемую в каждой новой версии, что со временем сделало ее сложной и запутанной.

Модель DTE состоит из набора интерфейсов COM и связанных с ними реализаций, охватывающих различные аспекты функциональных возможностей системы Visual Studio. Так как конкретные классы в основном реализуют соответствующие интерфейсы, можно ожидать наличия массы пар классов — интерфейсов и их реализаций. Например, корневой объект представляет собой класс DTE, который реализует интерфейс `_DTE`.

В силу самой их природы, интерфейсы не поддерживают расширяемости и никогда не должны изменяться, поскольку любое изменение их структуры разрушит структуры всех классов, реализующих такой исходный интерфейс. По мере выхода новых версий системы Visual Studio (каждая из которых требует добавления новой функциональной возможности к существующим классам объектной модели) это порождало проблемы. Компания Microsoft не могла обновлять существующие интерфейсы, так как это привело бы к проблемам с существующими надстройками, так что вместо этого было принято решение о создании в каждой новой версии системы Visual Studio новой версии интерфейсов, наследующей предыдущую, и о добавлении к ней новых требований. Эти новые интерфейсы использовали суффиксы, указывающие номер редакции, так что их имена не совпадают с именами предшественников, что привело к сегодняшней запутанной и недружелюбной модели, в которой одну и ту же часть объектной модели представляет несколько интерфейсов/классов.

Например, взглянем на интерфейсы `Debugger`, `Debugger2`, `Debugger3`, `Debugger4` и `Debugger5`. Интерфейс `Debugger` был частью Visual Studio 2003 и представляет собой исходный интерфейс. `Debugger2` — обновленная версия `Debugger` для Visual Studio 2005, `Debugger3` — для Visual Studio 2008, `Debugger4` — для Visual Studio 2008 SP1, а `Debugger5` — для Visual Studio 2010. Корневой интерфейс DTE также имеет редакцию DTE2, и обычно используется именно она, а не ее предшественница.

С практической точки зрения это означает сложность навигации по иерархии объектной модели. Модель демонстрирует методы классов более раннего ее воплощения, но для обращения к ее функциям требуется приведение объекта к более позднему интерфейсу. Например, первая итерация объекта `Solution` не предоставляет возможности создания папки решения. Такая возможность появилась только с методом `AddSolutionFolder` интерфейса `Solution2`. Поэтому приведенный далее код макроса работать *не* будет:

```
Public Sub AddSolutionFolder()
    DTE.Solution.AddSolutionFolder("TestFolder") ' Неверно
End Sub
```

Зато такой макрос вполне работоспособен:

```
Public Sub AddSolutionFolder()
    Dim solution As Solution2 = DirectCast(DTE.Solution, Solution2)
    solution.AddSolutionFolder("TestFolder")
End Sub
```

Как видите, упомянутые выше обстоятельства делают модель автоматизации сложной и запутанной.

Из-за того, что модель автоматизации базируется на COM-технологии и мы используем для взаимодействия с ней управляемый код, нам необходимо использовать межоперационные сборки для обеспечения моста между нашим управляемым кодом и объектной моделью COM. К сожалению, как и сама объектная модель, они также запутанны. Дополнительная межоперационная сборка добавляется с каждой версией системы Visual Studio, так что ваш проект должен ссылаться на каждую такую сборку — от базовой до выпущенной с наименьшей версией системы Visual Studio, которую поддерживает вашу надстройку или макрос. Например, надстройка или макрос, поддерживающий только систему Visual Studio 2010, должны будут работать с `EnvDTE.dll` (из версии Visual Studio 2003), `EnvDTE80.dll` (из версии Visual Studio 2005), `EnvDTE90.dll` (из версии Visual Studio 2008), `EnvDTE90a.dll` (из версии Visual Studio 2008 SP1) и `EnvDTE100.dll` (из версии Visual Studio 2010).

Давайте рассмотрим некоторые из различных функциональных областей Visual Studio, которые модель автоматизации предоставляет для нас, включая решения и проекты, документы и окна, команды, отладчики и события. Все они располагаются под корневым объектом `DTE` (который должен быть приведен к `DTE2` для того, чтобы получить доступ к последней редакции этого объекта).



Приводимые примеры кода — коды макросов; сами макросы подробно рассматриваются в главе 52. Большинство примеров выводит информацию с использованием команды `Debug.Print`, после чего ее можно увидеть в окне Output в Macros IDE.

Решения и проекты

Объект `DTE.Solutions` позволяет автоматизировать открытое в настоящий момент решение, решая такие задачи, как перечисление содержащихся в нем проектов, создание нового проекта в решении (или удаление старого проекта), добавление папки решения, получение и установка конфигурации и свойств проекта и даже открытие нового решения в интегрированной среде разработки Visual Studio и работа с ним. Приведенный далее код демонстрирует перечисление проектов в решении, а также вывод в окно Output их имен и количества элементов в каждом проекте:

```
Public Sub EnumerateProjects()
  For Each project As Project In DTE.Solution.Projects
    Debug.Print(project.Name & " contains " & _
                project.ProjectItems.Count.ToString() & "project items")
  Next project
End Sub
```



Выполнить перечисление проектов в активном решении можно также с помощью коллекции `DTE.ActiveSolutionProjects`.

Можно автоматизировать и проекты в решении. Сюда включены элементы проектов и содержащие их файлы. Можно также получить или обновить конфигурацию и свойства проекта и добавить или удалить элементы из проекта:

```
Public Sub EnumerateProjectsItems()
  Dim project As Project = _
    DTE.Solution.Projects.Item(1) ' Первый проект
  For Each projectItem As ProjectItem In project.ProjectItems
    Debug.Print(projectItem.Name)
  Next projectItem
End Sub
```

Окна и документы

Окна в системе Visual Studio представляют собой либо окна инструментов (таких, как Solution Explorer, Tasks и т.д.), либо окна документов (файлов, открытых в редакторе кода или проектировщике). Работа с окнами любого типа относительно проста.

Можно перечислить все открытые окна и получить о них детальную информацию следующим образом:

```
Public Sub EnumerateOpenWindows()
  ' Открываются как окна инструментов, так и окна документов
  For Each window As Window2 In DTE.Windows
    Debug.Print(window.Caption & " | _
                State = " & window.WindowState.ToString())
  Next window
End Sub
```

Рассмотрим, как можно работать с окнами инструментов. Воспользуйтесь приведенным далее фрагментом кода для получения ссылки на окно (независимо от того, открыто ли оно) и работы с ним (его активизации, вывода или сокрытия, свертывания и т.д.):

```
Public Sub ShowandDockTaskListWindow()
  Dim window As Window2 = _
    DTE.Windows.Item(Constants.vsWindowKindTaskList)
  window.Visible = True      ' Вывод окна
  window.IsFloating = False ' Закрепление окна
  window.AutoHides = False   ' Отмена автосокрытия
  window.Activate()
End Sub
```

Можно получить ссылку на конкретное окно инструментов (такое, как Task List) и работать с его функциональной возможностью (например, добавлять задачи в окно Task List):

```
Public Sub AddTaskToTaskList()
  Dim tasksWindow As TaskList = DTE.ToolWindows.TaskList
  tasksWindow.TaskItems.Add("", "", "Created by a macro")
End Sub
```

Как видите, работа с окнами инструментов достаточно проста. Теперь посмотрим, как работать с окнами документов. Получить ссылку на активное окно в интегрированной среде разработки можно следующим образом:

```
Dim window As Window2 = DTE.ActiveWindow
```

Можно даже получить ссылку на само окно интегрированной среды разработки Visual Studio:

```
Dim window As Window2 = DTE.MainWindow
```

Можно автоматизировать окна документов в систему Visual Studio, включая открытие или закрытие окна документа, его активизацию и получение элемента проекта, открытого в окне документа. В приведенном далее примере перечисляются окна открытых документов с выводом их имен файлов и путей в окне вывода:

```
Public Sub EnumerateOpenDocuments()
  For Each document As Document In DTE.Documents
    Debug.Print(document.Name & ", Path=" & document.Path)
  Next document
End Sub
```

Для получения ссылки на окно активного документа используйте код

```
Dim document As Document = DTE.ActiveDocument
```

Для управления конфигурацией окон в интегрированной среде разработки можно воспользоваться коллекцией `DTE.WindowConfigurations`.

Команды

Каждое выполняемое действие в системе Visual Studio представлено командой. Например, все элементы меню после выбора выполняют команды. Каждая команда имеет уникальное имя, числовой идентификатор (в пределах группы) и идентификатор GUID, назначенный группе. Система Visual Studio имеет тысячи команд, в чем можно убедиться, просмотрев коллекцию `DTE.Commands` следующим образом:

```
Public Sub EnumerateCommands()
  For Each command As Command In DTE.Commands
    Debug.Print(command.Name & ", _
        ID=" & command.ID & ", GUID=" & command.Guid)
  Next command
End Sub
```

Для выполнения действий в системе Visual Studio с помощью своих надстроек или макросов нужно получить ссылку на соответствующую команду и выполнить ее. Например, вы хотите закомментировать выбранный текст в редакторе кода. Соответствующая команда называется `Edit.CommentSelection`, и вы можете выполнить ее с помощью следующего кода:

```
Public Sub ExecuteCommentSelectionCommand()
  DTE.ExecuteCommand("Edit.CommentSelection")
End Sub
```



С учетом количества имеющихся команд поиск имени команды для выполнения того или иного действия может оказаться сложным делом. Простейший способ найти имя команды — записать макрос (см. главу 52) с выполненным действием, а затем просмотреть код сгенерированного макроса.

Вы также можете прослушивать выполняемые команды, которые запускаются системой Visual Studio, как события, которые можно обрабатывать. Одно событие генерируется перед выполнением команды, а другое — после ее завершения. Например, вы можете выполнять некоторые действия, когда осуществляется вставка текста в редактор кода (т.е. в ответ на команду `Edit.Paste`). Обработка событий рассматривается позже в данной главе.

Отладчик

Управлять функциями отладчика Visual Studio можно с помощью объекта автоматизации DTE.Debugger. Так, вы можете работать с точками останова, управлять выполнением кода и исследовать различные аспекты отлаживаемого приложения (включая процессы и потоки).

Приведенный далее фрагмент кода демонстрирует перечисление всех точек останова в текущем решении:

```
Public Sub EnumerateBreakpoints()
  For Each breakpoint As Breakpoint2 _
        In DTE.Debugger.Breakpoints
    Debug.Print(breakpoint.Name & " | File: " & "breakpoint.File & "_
                  " | Function: " & breakpoint.FunctionName &_
                  " | Line: " & breakpoint.FileLine)
  Next breakpoint
End Sub
```

Вы можете также управлять выполнением кода при отладке приложения, в частности запускать и прекращать отладку, шаг за шагом проходить код, выполнять программу до текущего местоположения курсора и т.д. Приведенный далее код демонстрирует запуск текущего решения в отладчике:

```
Public Sub RunApplicationInDebugger()
    DTE.Debugger.Go()
End Sub
```

События

Модель автоматизации позволяет прослушивать действия Visual Studio и отвечать на них запуском событий, которые вы можете обрабатывать. Эти события классифицируются в соответствии с их функциональной областью в виде подобъектов DTE.Events, включая DocumentEvents, WindowEvents, BuildEvents, SolutionEvents, ProjectsEvents, DebuggerEvents и многие другие. В главе 51 демонстрируется обработка событий в надстройках. В случае макросов модуль EnvironmentEvents, автоматически добавляемый в каждый проект макроса, определяет переменные событий, для которых вы можете создавать методы обработки событий (с применением ключевого слова Handles). Приведенный далее фрагмент кода перехватывает событие DocumentOpened в объекте DTE.Events.DocumentEvents (экземпляр которого был предварительно создан и сохранен в переменной DocumentEvents в модуле по умолчанию EnvironmentEvents). Поместите его в модуль EnvironmentEvents, сохраните модуль и закройте интегрированную среду разработки Macros:

```
Private Sub DocumentEvents_DocumentOpened( _
        ByVal Document As EnvDTE.Document) _
        Handles DocumentEvents.DocumentOpened
    MsgBox("Document opened: " & Document.Name & _
           " at " & DateTime.Now)
End Sub
```

Этот обработчик событий будет запускаться при открытии нового документа в интегрированной среде разработки Visual Studio.

Резюме

Из этой главы вы узнали о средствах расширения функциональных возможностей системы Visual Studio 2010 и познакомились со структурой и возможностями модели автоматизации системы Visual Studio, которую надстройки и макросы используют для ее расширения. В следующих двух главах рассматриваются два упомянутых средства расширения системы Visual Studio с применением указанной объектной модели.

51

Надстройки

В ЭТОЙ ГЛАВЕ...

- Структура надстроек
- Создание надстроек
- Тестирование и отладка надстроек
- Распространение надстроек

Как описано в главе 50, надстройки (add-ins) системы Visual Studio представляют собой компоненты, работающие в ней и расширяющие ее функциональные возможности с помощью модели автоматизации.

В этой главе будет рассмотрен процесс создания надстроек для системы Visual Studio, которые внедряются в интегрированную среду разработки Visual Studio для открытия окна инструментов (которое позволяет хранить некоторые заметки), выполнения действий в системе Visual Studio (копирование выбранного текста из редактора кода) и обработки событий системы Visual Studio (перехват команд вырезания и копирования в редакторе кода).

Разработка надстройки

При создании проекта надстройки Visual Studio запускается мастер Add-in Wizard, который помогает создать подходящую структуру и базовую функциональность вашей надстройки на основе ваших ответов на задаваемые им вопросы. После этого на созданном мастером фундаменте вы можете надстраивать необходимую вам реализацию функциональной возможности, создавая тем самым полезный инструмент.

Мастер Add-in Wizard

Начнем создание проекта с использования шаблона Visual Studio Add-in в подкатегории проектов Extensibility (в категории Other Project Type), как показано на рис. 51.1.

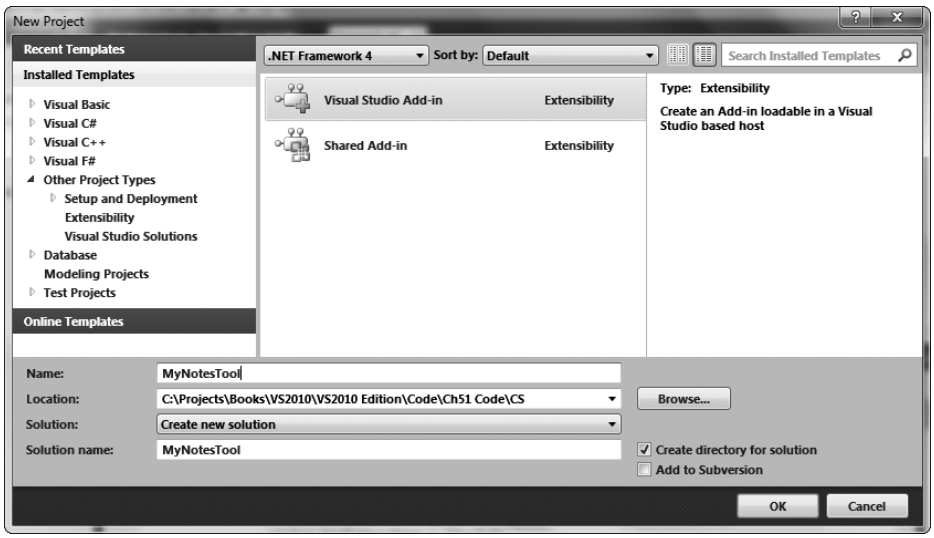


Рис. 51.1



Обратите внимание, что категория Extensibility содержит также шаблон Shared Add-in. Этот шаблон подобен шаблону Visual Studio Add-in, но используется для создания надстроек для разных приложений пакета Microsoft Office, а не системы Visual Studio.

Щелчок на кнопке OK запускает мастер Add-in Wizard. Работа мастера состоит из семи этапов (включая этап приветствия и резюме). В этом разделе мы пройдем через каждый этап работы мастера и рассмотрим предоставляемые им возможности.

В начале работы отображается экран приветствия с кратким описанием мастера, показанный на рис. 51.2.

На следующем этапе работы мастера Add-in Wizard, как показано на рис. 51.3, необходимо выбрать язык разработки для надстройки (поскольку шаблон Visual Studio Add-in не находится в определенной языковой категории в диалоговом окне New Project). Вы можете выбирать из четырех вариантов — Visual C#, Visual Basic, Visual C++/CLR и Visual C++/ATL. Система Visual Studio генерирует проект на выбранном вами языке программирования.

Теперь выберите приложение для своей надстройки, как показано на рис. 51.4. Имеется два варианта — Visual Studio IDE и Visual Studio Macros IDE (который рассматривается в следующей главе). Можете установить или снять флажок для каждого приложения. Разработанное надстройка будет доступно для указанных на этом этапе приложений.



Рис. 51.2

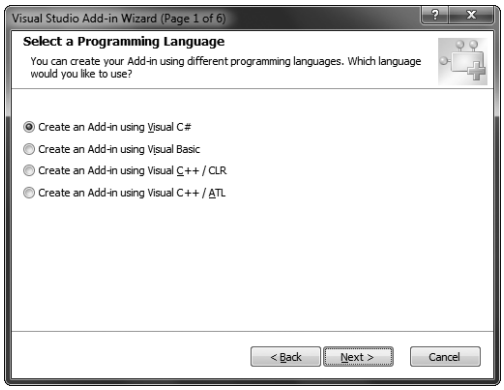


Рис. 51.3

Введите имя и описание своего надстройки, как показано на рис. 51.5. Эта информация — именно то, что конечные пользователи будут видеть в диалоговом окне Add-in Manager системы Visual Studio для вашей надстройки.

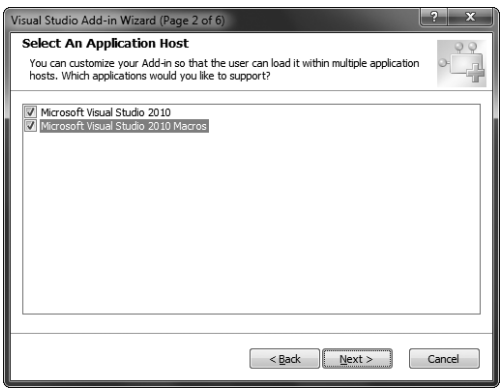


Рис. 51.4



Рис. 51.5

Следующий этап содержит настройки загрузки надстройки и ее взаимодействия с системой Visual Studio. Можете установить три флажка, показанные на рис. 51.6. Первый указывает, что надстройка будет иметь отдельную команду в меню Tools, которую можно использовать для ее активизации. Второй указывает, что вы хотите загружать свою надстройку при запуске интегрированных сред Visual Studio IDE или Visual Studio Macros IDE, а третий используется для указания, что ваша надстройка не использует модальный пользовательский интерфейс, а потому может применяться со сборкой из командной строки.

Следующий этап (рис. 51.7) позволяет выводить информацию в диалоговом окне About системы Visual Studio, что весьма полезно при выпуске вашей надстройки в качестве программного продукта.

На последнем этапе, как показано на рис. 51.8, вы видите итоговую информацию о том, что было выбрано вами в процессе работы мастера. Можете либо вернуться назад и изменить свои настройки, либо щелкнуть на кнопке Finish, чтобы продолжить работу и сгенерировать решение и первоначальный текст надстройки.

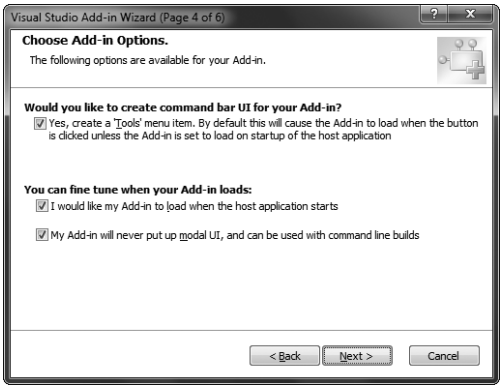


Рис. 51.6

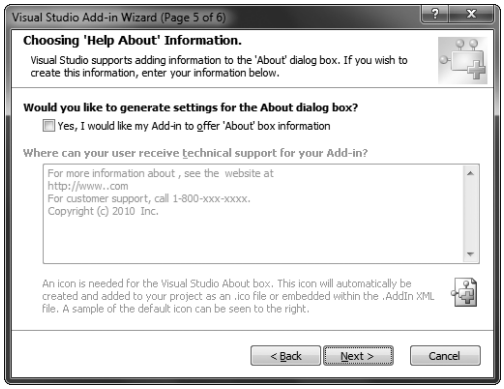


Рис. 51.7

После щелчка на кнопке Finish система Visual Studio генерирует решение со всеми необходимыми файлами, созданными для вашей надстройки, сконфигурированное в соответствии с выбранными вами настройками.

Структура проекта

После создания проекта его структура имеет вид, показанный на рис. 51.9.

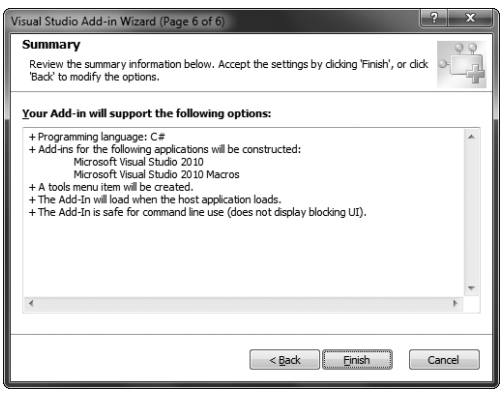


Рис. 51.8

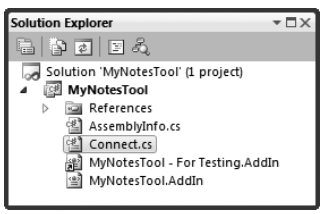


Рис. 51.9

Как видите, проект состоит из файла `Connect.cs` (или `Connect.vb`) и двух файлов с расширением `.AddIn`.

Файл `Connect.cs/Connect.vb` содержит основной класс, управляющий надстройками. Файлы `.AddIn` используются для того, чтобы информировать систему Visual Studio о надстройке, с тем чтобы она могла быть загружена. Один из файлов располагается в вашей папке проекта, а второй, как вы можете заметить, представляет собой связный файл (`MyNotesTool - For Testing.AddIn`), который находится в папке `My Documents\Visual Studio 2010\Addins` вашего пользовательского профиля Windows. Как указывает данное имя, этот файл используется для того, чтобы система Visual Studio могла обнаружить надстройку во время ее тестирования и отладки. Причина, по которой файл располагается в указанной папке, заключается

в том, что система Visual Studio просматривает ее в поисках файлов надстроек. Если вы откроете оба файла, то обнаружите, что они идентичны с единственным исключением — узел `Assembly` связного файла включает полный путь к скомпилированной сборке надстройки, в то время как второй файл включает только имя сборки (ожидается, что она находится в той же папке, что и файл `.AddIn`). Более близко с файлами `.AddIn` вы познакомитесь позже в этой главе.

Тестирование надстройки

Начните с проверки, все ли корректно работает при простом запуске вашего проекта. Запустите новый экземпляр системы Visual Studio 2010, в котором можете тестировать и отлаживать свою надстройку. Если вы выберете в мастере настройку, отвечающую за ее автоматический запуск одновременно с интегрированной средой разработки и за создание команды в меню Tool, то в меню Tools появится элемент меню, соответствующий вашей надстройке, вместе с пиктограммой по умолчанию в виде улыбающегося лица (которую вы можете заменить собственной пиктограммой), как показано на рис. 51.10.

Если вы не выбрали автоматическую загрузку при запуске интегрированной среды разработки, можете запустить надстройку, используя инструмент Add-in Manager (меню Tools⇨Add-in Manager), поставив отметку возле его имени, как показано на рис. 51.11.

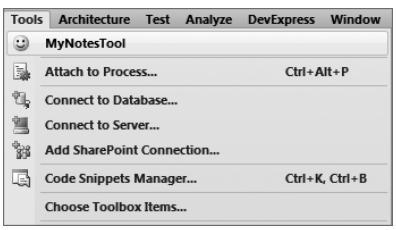


Рис. 51.10

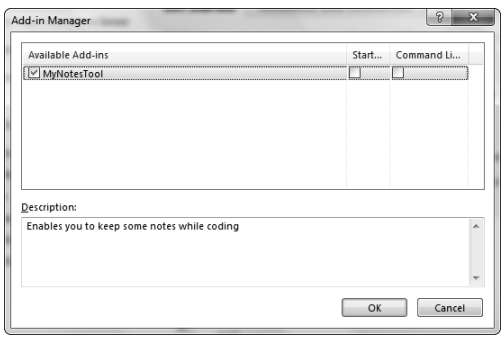


Рис. 51.11



Если ваша надстройка не появляется в окне Add-in Manager, система Visual Studio не может найти `.AddIn`-файл. Выберите команду меню Tools⇨Options и категорию Add-in/Macro Security (в категории Environment, как показано на рис. 51.12). Убедитесь, что путь, где располагается тестируемый файл `.AddIn`, имеется в списке Add-in File Paths. Возможно также, что использованные в данном диалоговом окне переменные среды в вашей системе не объявлены, — проверьте и такую возможность.

Закройте отладочный экземпляр системы Visual Studio, чтобы завершить отладку надстройки.

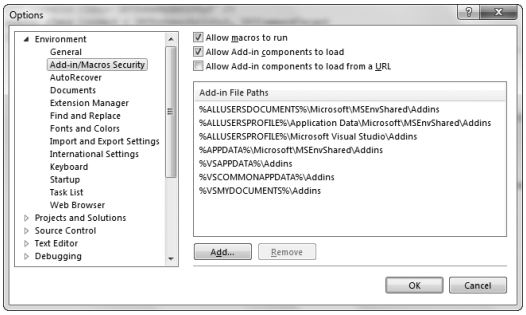


Рис. 51.12

Файл с расширением `.AddIn`

Как упоминалось ранее в этой главе, в вашем решении имеется два `.AddIn`-файла: один — в папке проекта, а второй, связный файл, — в папке `My Documents\Visual Studio 2010\Addins` на вашей машине.

В ранних версиях системы Visual Studio нужно было зарегистрировать компонент COM надстройки на своей машине вручную, что усложняло развертывание (надстройка не могла быть установлена простым копированием). `.AddIn`-файлы разработаны для того, чтобы упростить процесс развертывания надстроек. Поместив `.AddIn`-файл в папку, которую просматривает система Visual Studio, вы позволяете ей воспользоваться им для того, чтобы найти и загрузить вашу надстройку (не беспокоясь о необходимости его регистрации). По сути, `.AddIn`-файлы указывают Visual Studio, где находятся ваши надстройки (которые обычно находятся на том же пути, что и `.AddIn`-файл).

`.AddIn`-файлы представляют собой файлы в формате XML, и в дополнение к указанию системе Visual Studio на размещение надстройки они содержат информацию о конфигурации, такую как доступность надстройки для различных программ (включая разные версии Visual Studio), какое описание надстройки должно выводиться в окне Add-in Manager, а также настройки запуска надстройки.

Открыв `.AddIn`-файл, вы найдете в нем XML-код наподобие приведенного ниже:

```
<?xml version="1.0" encoding="UTF-16" standalone="no"?>
<Extensibility xmlns="http://schemas.microsoft.com/AutomationExtensibility">
  <HostApplication>
    <Name>Microsoft Visual Studio Macros</Name>
    <Version>10.0</Version>
  </HostApplication>
  <HostApplication>
    <Name>Microsoft Visual Studio</Name>
    <Version>10.0</Version>
  </HostApplication>
  <Addin>
    <FriendlyName>My Notes</FriendlyName>
    <Description>My Notes</Description>
    <Assembly>MyNotesTool.dll</Assembly>
    <FullClassName>MyNotesTool.Connect</FullClassName>
    <LoadBehavior>5</LoadBehavior>
    <CommandPreload>1</CommandPreload>
    <CommandLineSafe>1</CommandLineSafe>
  </Addin>
</Extensibility>
```

Особое внимание следует обратить на узлы `HostApplication`, перечисляющие имена приложений и их версии, для которых должна быть доступна надстройка. Приведенный файл делает надстройку доступной для системы Visual Studio 2010 и для среды Visual Studio

Macros 2010 IDE. Если вы хотите сделать свою надстройку доступной для других версий Visual Studio, просто добавьте дополнительные узлы `HostApplication` с соответствующими номерами версий Visual Studio (Visual Studio 2008 = 9.0, Visual Studio 2005 = 8.0). Конечно, при этом убедитесь, что вы не используете возможностей, специфичных для системы Visual Studio 2010, и удалите ссылки на библиотеки EnvDTE версий, больших, чем минимально поддерживаемая.



Если вы обновляете надстройку из предыдущей версии Visual Studio, добавьте в существующий `.AddIn`-файл новый узел `HostApplication` со значением `10.0` в узле `Version`, чтобы надстройка могла работать в системе Visual Studio 2010.

Класс `Connect`

В этом разделе рассматривается структура основного класса, управляющего надстройками. Класс `Connect.cs` (или `Connect.vb`) управляет жизненным циклом надстройки, и вы можете найти ряд методов из интерфейсов `IDTExtensibility2` и `IDTCommandTarget`, которые обрабатывают уведомления событий и реализованы этим классом.

Интерфейс `IDTExtensibility2` предоставляет обработчики событий, генерируемых системой Visual Studio, которые уведомляют надстройку о каждой точке ее жизненного цикла. Приведенные далее методы составляют интерфейс `IDTExtensibility2`.

- **`OnConnection`**. Вызывается, когда надстройка загружается система Visual Studio.
- **`OnStartupComplete`**. Вызывается, когда система Visual Studio завершает загрузку надстройки.
- **`OnAddInsUpdate`**. Вызывается, когда изменяется коллекция надстроек для системы Visual Studio.
- **`OnBeginShutdown`**. Вызывается, когда система Visual Studio завершает работу.
- **`OnDisconnection`**. Вызывается, когда надстройка выгружается из системы Visual Studio.

Интерфейс `IDTCommandTarget` предоставляет обработчики событий для именованных команд, используемых надстройками. Приведенные далее методы составляют интерфейс `IDTCommandTarget`.

- **`Exec`**. Вызывается, когда команда, используемая надстройками, вызывается из системы Visual Studio (как, например, когда выбирается команда меню, созданная для надстройки в меню Tools). Система Visual Studio передает этому методу имя команды, чтобы вы могли отреагировать на нее соответствующим образом.
- **`QueryStatus`**. Вызывается, когда система Visual Studio запрашивает состояние команды (такое, как, например, доступна ли она).

Создание окна инструмента

Теперь, когда вы ознакомились со структурой и жизненным циклом надстройки, самое время добавить в него немного функциональности для взаимодействия со средой Visual Studio IDE и реализовать некоторое полезное поведение. Пример, с которым мы работаем в данной главе, создает прикрепляемое окно инструмента в системе Visual Studio, которое будет позволять размещать примечания при работе. К сожале-

нию, мастер Add-in Wizard не оказывает помощи в создании вашего собственного окна инструмента (которое представляет собой одно из наиболее распространенных требований при написании надстроек), так что вам придется заняться этим самостоятельно. В этом разделе мы шаг за шагом рассмотрим создание такого окна.



Несмотря на то что система Visual Studio 2010 имеет пользовательский интерфейс для применения технологии WPF, вы все еще должны при создании окна инструмента надстройки применять подсистему Windows Forms. Для использования технологии WPF для окон инструментов следует применять пакеты VSPackages из набора Visual Studio 2010 SDK.

Добавим в проект новый элемент Windows Forms User Control и назовем его `NotesUserControl` (`.cs` или `.vb`). Добавим к пользовательскому управляющему элементу управляющий элемент RichTextBox с именем `rtbNotes`, установим свойство `BorderStyle` равным `None` и закрепим его так, чтобы заполнить область управляющего элемента.

Теперь вернемся к файлу `Connect.cs` (или `.vb`) и добавим в него следующий метод для упрощения процесса создания окна инструмента:

VB

```
Private toolWindow As Window2 = Nothing
Private Function CreateToolWindow( _
            ByVal guid As String, _
            ByVal windowTitle As String, _
            ByVal classPath As String) As Object
    Dim windowObject As Object = Nothing

    Dim windows As Windows2 = DirectCast(applicationObject.Windows, Windows2)
    Dim assemblyLocation As String = Assembly.GetCallingAssembly().Location

    toolWindow = DirectCast(windows.CreateToolWindow2(addInInstance,
         assemblyLocation, classPath, windowTitle, guid, windowObject), Window2)

    Return windowObject
End Function
```

C#

```
Private Window2 toolWindow = null;

private object CreateToolWindow(string guid, string windowTitle,
                                string classPath)
{
    object windowObject = null;

    Windows2 windows = (Windows2)_applicationObject.Windows;
    string assemblyLocation = Assembly.GetCallingAssembly().Location;

    toolWindow = (Window2)windows.CreateToolWindow2(addInInstance,
      assemblyLocation, classPath, windowTitle, guid, ref windowObject);

    return windowObject;
}
```



На уровне класса необходимо поддерживать ссылку на пользовательский управляющий элемент, поскольку окно надстройки в процессе жизненного цикла последнего не уничтожается, а просто скрывается.

Окно инструмента создается после выбора соответствующей команды меню Tools. Вас уведомляют об этом в методе Exec, и (обратите внимание) мастер создает код для ответа на это уведомление (хотя этот код и не выполняет никаких действий). Воспользуйтесь приведенным исходным текстом для создания окна инструмента и его вывода в системе Visual Studio (добавленный в метод код выделен полужирным шрифтом):

VB

```
Private notesUserControl As NotesUserControl

Public Sub Exec(ByVal commandName As String,
                ByVal executeOption As vsCommandExecOption,
                ByRef varIn As Object, ByRef varOut As Object,
                ByRef handled As Boolean)
                        Implements IDTCommandTarget.Exec
  handled = False
  If executeOption =
    vsCommandExecOption.vsCommandExecOptionDoDefault Then _
    If commandName = "MyNotesTool.Connect.MyNotesTool" Then
      ' Идентификатор окна инструмента
      Dim windowID As String =
            "{fb9e4681-681d-4216-9a28-0f09f3528360}"

      ' Создание окна, если оно еще не создано
      If toolWindow Is Nothing Then
        notesUserControl = DirectCast(CreateToolWindow(
                    windowID, "My Notes",
                   "MyNotesTool.NotesUserControl"),
                   NotesUserControl)
      End If

      ' Если окно скрыто, делаем его видимым
      toolWindow.Visible = True

      handled = True
      Return
    End If
  End If
End Sub
```

C#

```
private NotesUserControl notesUserControl;
public void Exec(string commandName,
                 vsCommandExecOption executeOption,
                 ref object varIn, ref object varOut,
                 ref bool handled)
{
  handled = false;
  if (executeOption ==
        vsCommandExecOption.vsCommandExecOptionDoDefault)
  {
    if (commandName == "MyNotesTool.Connect.MyNotesTool")
    {
      // Идентификатор окна инструмента
      string windowID =
          "{fb9e4681-681d-4216-9a28-0f09f3528360}";

      // Создание окна, если оно еще не создано
      if (toolWindow == null)
      {
        notesUserControl = ( NotesUserControl)
          CreateToolWindow(windowID, "My Notes" ,
```

```
                                "MyNotesTool.NotesUserControl");
            }

            // Если окно скрыто, делаем его видимым
            toolWindow.Visible = true;

            handled = true;
            return;
        }
    }
}
```

Как видно из исходного текста, процесс создания окна относительно прост. Вы передаете ранее созданному методу `CreateToolWindow` идентификатор, уникально идентифицирующий это окно инструмента, заголовок окна и квалифицированное имя класса пользовательского управляющего элемента, и он создает окно инструмента в системе Visual Studio.

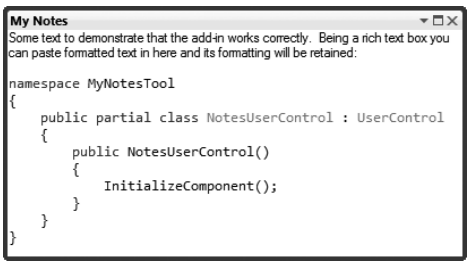


Рис. 51.13

Теперь запустите свой проект и выберите свою надстройку в меню Tools. В качестве окна инструмента будет выведен пользовательский управляющий элемент (рис. 51.13), который затем вы сможете перемещать и закреплять в интегрированной среде разработки, как и любое другое окно инструмента.



Система Visual Studio запоминает местоположение окна (используя уникальный идентификатор для хранения и получения информации), так что при следующей загрузке системы Visual Studio окно появится в том месте, где оно находилось в последний раз (хотя это работает, только когда надстройка не отлаживается). Однако для того, чтобы окно выводилось при запуске системы Visual Studio, его следует создавать при запуске надстройки, а не при выборе команды меню.

Доступ к модели автоматизации Visual Studio

Теперь вы можете добавить к окну инструмента собственную функциональную возможность (в пользовательский управляющий элемент), такую как сохранение текста в текстовый файл (при желании), как при программировании стандартного приложения. Однако этот пример не демонстрирует интеграцию с функциональной возможностью системы Visual Studio и его событиями, так что добавим в надстройку функциональную возможность для демонстрации интеграции путем создания кнопки для получения выделенного в редакторе кода текста и вставки его в примечания в текущей позиции курсора.

Для обращения к объектной модели системы Visual Studio из пользовательского управляющего элемента следует создать статическую переменную класса `_applicationObject` в классе `Connect` и обеспечить к ней открытый доступ с помощью свойства, как показано в приведенном далее фрагменте кода:

VB

```
Private Shared _applicationObject As DTE2

Public ReadOnly Property ApplicationObject() As DTE2
    Get
        Return _applicationObject
    End Get
End Property
```

C#

```
private static DTE2 _applicationObject;

public static DTE2 ApplicationObject
{
    get { return Connect._applicationObject; }
}
```

Добавим к пользовательскому управляющему элементу управляющий элемент ToolStrip с кнопкой, которая будет копировать выделенный в редакторе кода текст и вставлять его после щелчка в текстовое поле. В обработчик событий для этой кнопки добавим следующий код:

VB

```
Private Sub btnCopy_Click(ByVal sender As System.Object,
                          ByVal e As System.EventArgs)
                          Handles btnCopy.Click
  If Not Connect.ApplicationObject.ActiveDocument Is Nothing Then
    Dim selection As TextSelection = DirectCast(
      Connect.ApplicationObject.ActiveDocument.Selection,
        TextSelection)
      rtbNotes.SelectedText = selection.Text
  End If
End Sub
```

C#

```
private void btnCopy_Click(object sender, EventArgs e)
{
  if (Connect.ApplicationObject.ActiveDocument != null)
  {
    TextSelection selection =
        Connect.ApplicationObject.ActiveDocument.Selection as TextSelection;
    rtbNotes.SelectedText = selection.Text;
  }
}
```

Этот код получает выделенный текст из активного документа редактора кода и вставляет его в текущей позиции пользовательского управляющего элемента. Заметим, что при вставке код оказывается неотформатированным (т.е. без выделения цветом синтаксических элементов). В качестве альтернативы можете использовать для копирования и вставки текста приведенный далее код — при этом сохраняется цвет синтаксиса, но теряется текущее содержимое системного буфера обмена:

VB

```
Private Sub btnCopy_Click(ByVal sender As System.Object,
                          ByVal e As System.EventArgs)
                          Handles btnCopy.Click
  If Not Connect.ApplicationObject.ActiveDocument Is Nothing Then
    Connect.ApplicationObject.ActiveDocument.Selection.Copy()
    rtbNotes.Paste()
  End If
End Sub
```

C#

```
private void btnCopy_Click(object sender, EventArgs e)
{
  if (Connect.ApplicationObject.ActiveDocument != null)
  {
    Connect.ApplicationObject
           .ActiveDocument.Selection.Copy();
    rtbNotes.Paste();
  }
}
```

Обработка событий системы Visual Studio

В качестве последнего примера обработаем событие, генерируемое системой Visual Studio. Мы будем обрабатывать события команд Cut и Copy (перед тем, как команды будут выполнены), получая выделенный текст из редактора кода и автоматически вставляя его в наш пользовательский управляющий элемент.

Сначала получим ссылку на команды, события которых будут перехватываться (Cut и Copy), а затем — и сами объекты событий команд. Разработчики на C# добавляют также обработчик события `BeforeExecute` для каждой команды:

VB

```
Private WithEvents cutEvent As CommandEvents = Nothing
Private WithEvents copyEvent As CommandEvents = Nothing

Private Sub EnableAutoCopy()
  ' Включение прослушивания команд Cut и Copy
  Dim cmdCut As Command =
      Connect.ApplicationObject.Commands.Item("Edit.Cut", 0)
  Dim cmdCopy As Command =
      Connect.ApplicationObject.Commands.Item("Edit.Copy", 0)

  cutEvent = Connect.ApplicationObject.Events.CommandEvents(cmdCut.Guid,
                                                            cmdCut.ID)
  copyEvent = Connect.ApplicationObject.Events.CommandEvents(cmdCopy.Guid,
                                                             cmdCopy.ID)
End Sub
```

C#

```
private CommandEvents cutEvent = null;
private CommandEvents copyEvent = null;

private void EnableAutoCopy()
{
  // Включение прослушивания команд Cut и Copy
  Command cmdCut =
    Connect.ApplicationObject.Commands.Item("Edit.Cut", 0);

  cutEvent =
    Connect.ApplicationObject.Events.get_CommandEvents(cmdCut.Guid,
                                                       cmdCut.ID);

  cutEvent.BeforeExecute +=
    new _dispCommandEvents_BeforeExecuteEventHandler(OnBeforeCutCopy);

  Command cmdCopy = Connect.ApplicationObject.Commands.Item("Edit.Copy", 0);

  copyEvent = Connect.ApplicationObject.Events.get_CommandEvents(cmdCopy.Guid,
                                                                 cmdCopy.ID);

  copyEvent.BeforeExecute +=
    new _dispCommandEvents_BeforeExecuteEventHandler(OnBeforeCutCopy);
}
```

Теперь можно определить метод обработчика события, который будет обрабатывать событие BeforeExecute для обеих команд, получая выделенный текст из редактора кода и передавая его в пользовательский управляющий элемент:

VB

```
Private Sub OnBeforeCutCopy(ByVal guid As String,
                            ByVal id As Integer,
                            ByVal customIn As Object,
                            ByVal customOut As Object,
                            ByRef cancel As Boolean)
                            Handles cutEvent.BeforeExecute,
                            copyEvent.BeforeExecute
  Dim codeWindow As TextWindow = TryCast(
    Connect.ApplicationObject.ActiveWindow.Object, EnvDTE.TextWindow)
  If Not codeWindow Is Nothing Then
    rtbNotes.SelectedText = codeWindow.Selection.Text &
            Environment.NewLine & Environment.NewLine
  End If
End Sub
```

C#

```
private void OnBeforeCutCopy(string guid, int id,
                             object customIn,
                             object customOut,
                             ref bool cancel)
{
  TextWindow codeWindow = Connect.ApplicationObject.ActiveWindow.Object
    as EnvDTE.TextWindow;
  if (codeWindow != null)
  {
    rtbNotes.SelectedText = codeWindow.Selection.Text +
                Environment.NewLine + Environment.NewLine;
  }
}
```

Наконец следует выполнить очистку при выгрузке надстройки и освободить все обработчики событий (метод CloseToolWindow вызывается из класса Connect в методе OnDisconnection):

C#

```
private void DisableAutoCopy()
{
  if (cutEvent != null)
      Marshal.ReleaseComObject(cutEvent);
  if (copyEvent != null)
      Marshal.ReleaseComObject(copyEvent);

  cutEvent = null;
  copyEvent = null;
}

public void CloseToolWindow()
{
  DisableAutoCopy();
}
```

Развертывание надстройки

Несмотря на то что надстройка представляет собой COM-компонент (который обычно требует регистрации в Windows), развертывать надстройки в системе Visual

Studio очень просто благодаря .AddIn-файлу, который позволяет Visual Studio обнаружить надстройку и использовать ее.

Как говорилось ранее в данной главе, система Visual Studio просматривает все пути, перечисленные в диалоговом окне Options (рис. 51.12), и ищет в них файлы с расширением .AddIn. Поэтому при развертывании своей надстройки поместите .AddIn-файл и сборку надстройки (т.е. .dll-файл) в один из указанных путей (обычно — в папку профиля пользователя My Documents\Visual Studio 2010\Addins), что позволит системе Visual Studio при запуске находить и загружать ваши надстройки.

Вы можете использовать для развертывания своей надстройки на другой пользовательской машине простую операцию XCOPY, но лучше создать программу установки, которая сделает все необходимое вместо вас. Можете использовать стандартный пакет инсталлятора Windows (.msi), но для данного экземпляра лучше, пожалуй, использовать пакет *Visual Studio Content Installer*. К сожалению, пакет Visual Studio Content Installer создается вручную, но это достаточно простая задача. По сути, он состоит из упакованных в zip-архив файлов, но с расширением .vsi, и специальным образом отформатированного XML-файла (также включенного в zip-файл) с расширением .vscontent, в котором содержится детальная информация об инсталлируемых файлах (из состава zip-файла) и места, куда они должны быть установлены.

Начнем с создания XML-файла с именем MyNotesTool.vscontent для нашего проекта и добавим в него следующее содержимое:

```
<VSContent
  xmlns="http://schemas.microsoft.com/developer/vscontent/2005">
  <Content>
    <FileName> MyNotesTool.Addin </FileName>
    <FileName> MyNotesTool.dll </FileName>
    <DisplayName> My Notes </DisplayName>
    <Description>
      Позволяет создавать примечания в окне
      инструмента при работе в Visual Studio.
    </Description>
    <FileContentType> Addin </FileContentType>
    <ContentVersion> 2.0 </ContentVersion>
  </Content>
</VSContent>
```

Теперь с помощью проводника Windows Explorer (или вашего любимого инструмента для работы с zip-архивами) запакуйте в zip-файл файлы MyNotesTool.AddIn, MyNotesTool.dll и файл MyNotesTool.vscontent, и присвойте ему имя MyNotesTool.vsi (не давайте ему расширение .zip). Теперь, когда кто-то дважды щелкнет на .vsi-файле, надстройка будет автоматически установлена и готова к работе при следующем запуске Visual Studio.

Резюме

В этой главе вы познакомились с надстройками для системы Visual Studio, а также прошли процесс создания надстройки, которое выводит закрепляемое окно инструмента, получает текст из редактора кода и отвечает на некоторые события последнего. Наконец вы узнали, каким образом лучше всего разворачивать надстройки для других разработчиков.

52

Макросы

В ЭТОЙ ГЛАВЕ...

- Понятие макроса
- Создание макросов
- Тестирование и отладка макросов
- Развертывание макросов

Подобно надстройкам, макросы представляют собой еще один вариант расширений, встроенных в систему Visual Studio. Как говорилось в главе 50, макросы имеют как много преимуществ по сравнению с надстройками (быстрота создания, возможность записи действий Visual Studio в макрос), так и много недостатков (они должны быть написаны на VB, не могут компилироваться и обладают ограниченными возможностями). Как следует воспользоваться средством автоматизации системы Visual Studio, зависит от ваших конкретных требований. Если вы выполняете один и тот же набор действий в Visual Studio, макросы существенно облегчат вашу задачу, сводя весь набор к одному действию.

В этой главе будет рассмотрен процесс записи макроса, его повторное выполнение, создание макроса "с нуля" и передача созданных макросов другим разработчикам.

Понятие макроса

Макрос системы Visual Studio (в действительности называется VSMacro, но среди разработчиков известен просто как макрос) представляет собой код, который может быть запущен в среде Visual Studio IDE для автоматизации некоторой задачи. Этот код не компилируется, а интерпретируется, т.е. по сути он написан на языке сценариев системы Visual Studio. Макрос реализуется как открытый метод модуля, не принимающий параметров и не возвращающий значения. Подобно надстройкам, макросы используют интерфейс программирования приложений Development Tools Extensibility (DTE) API

для автоматизации интегрированной среды разработки Visual Studio. Макрос не может выводить на экран элементы пользовательского интерфейса системы Visual Studio (такие, как окно инструмента), так что, если вам нужен этот вид функциональных возможностей, рассмотрите вопрос о применении надстроек.

Макросы существуют в виде модулей кода, а эти модули содержатся в проектах макросов. Файлы проектов макросов имеют расширение `.vsmacros` и концептуально очень похожи на стандартные файлы проектов системы Visual Studio. Однако в этом случае различные модули и связанные с ними файлы встраиваются в файл проекта, а не являются отдельными файлами. Кроме того, модуль кода может содержать несколько макросов.

Если вы разработчик на языке VB, у вас не будет проблем с кодированием макросов, поскольку при их создании используется именно этот язык программирования. К сожалению, C# в качестве языка программирования макросов не поддерживается, так что разработчикам на C# потребуется изучить VB или создавать надстройки вместо макросов.

Инструментальное Окно Macro Explorer

Перед тем как перейти к рассмотрению инструментов для создания макросов, взглянем на инструментальное окно Macro Explorer, которое помогает управлять макросами и запускать доступные макросы. Если окно инструмента невидимо, воспользуйтесь командой меню View⇨Other Windows⇨Macro Explorer для вывода этого окна в интегрированной среде разработки.

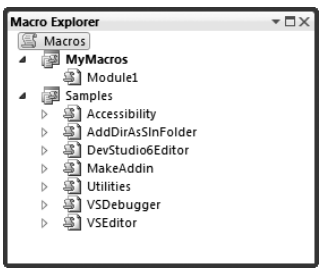


Рис. 52.1

Как видно из рис. 52.1, имеется узел верхнего уровня Macros, под которым располагаются проекты макросов. Перечисляются также все макросы в этих проектах.

Это диалоговое окно очень полезно для выбора и запуска макроса (просто дважды щелкните на макросе для его запуска), а также для редактирования макроса (щелкните на макросе правой кнопкой мыши и выберите из контекстного меню команду Edit). Если вы выберете редактирование макроса, его код будет открыт в интегрированной среде разработки Macros, которая рассматривается в следующем разделе главы.

Обратите внимание, что в Macro Explorer уже выводятся два проекта: MyMacros и Samples. MyMacros представляет собой проект по умолчанию, в котором вы создаете свои макросы (хотя при необходимости можно создавать дополнительные проекты макросов). Проект макросов Samples состоит из примеров макросов, которые вы можете использовать при изучении макросов и для работы с моделью автоматизации Visual Studio.

Интегрированная среда разработки Macros

Как показано на рис. 52.2, интегрированная среда разработки Visual Studio Macros очень похожа на интегрированную среду разработки Visual Studio, несмотря на то что это две совершенно разные программы. Изучив пользовательский интерфейс одной, вы будете лучше ориентироваться в другой.

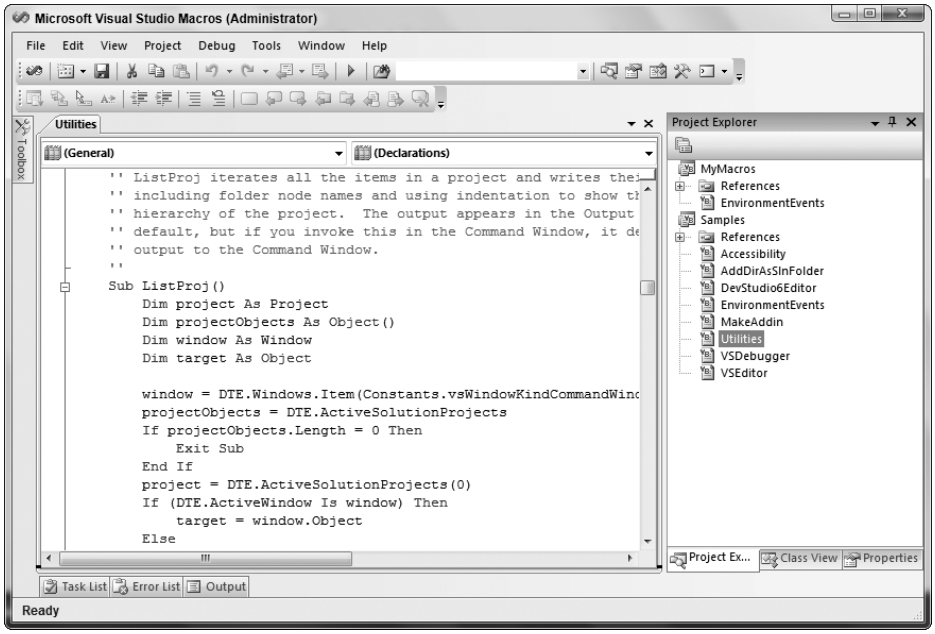


Рис. 52.2

Особый интерес вызывает окно Project Explorer (напоминающее окно Solution Explorer в интегрированной среде разработки Visual Studio) в правой части экрана. Оно позволяет переходить к разным проектам макросов и содержащимся в них модулям и файлам. Как можно видеть, в отличие от окна Macro Explorer, в основной интегрированной среде разработки Visual Studio, где выводятся доступные макросы каждого проекта, это окно выводит для каждого проекта макроса его модули и файлы. Окно Macro Explorer скрывает модули, выводя только функции из всех модулей проекта. Когда вы открываете модуль, это равноценно открытию исходного файла в среде Visual Studio.



Интегрированная среда разработки Visual Studio Macros не может запускаться, как автономная программа, и может загружаться только посредством системы Visual Studio.

Создание макроса

Имеется два способа создания макросов: либо создавать их “с нуля” и писать код самостоятельно, либо записывать набор действий системы Visual Studio с использованием макрорегистратора (macro recorder) для последующего воспроизведения. В этом разделе мы рассмотрим оба метода, начав с записи макросов, поскольку это более простой способ приступить к созданию макросов.

Запись макросов

Если вы хотите просто автоматизировать набор действий в среде Visual Studio, то примите к сведению, что простейший путь создания макроса для выполнения этих действий заключается в применении макрорегистратора для записи выполняемых действий с последующим воспроизведением при необходимости. Макрорегистратор записывает все действия, выполняемые вами в интегрированной среде разработки Visual Studio. При необходимости можно вручную внести изменения в записанный код, чтобы сгенерированный макрос работал именно так, как требуется вам. Так можно сэкономить время на изучение модели автоматизации системы Visual Studio и написание кода макросов вручную. Это также неплохой способ изучить, как создаются макросы.



Рис. 52.3

Для записи макроса воспользуйтесь командой меню Tools⇨Macros⇨Record Temporary Macro (или комбинацией клавиш <Ctrl+Shift+R>). При этом немедленно начнется запись всех выполняемых вами действий в интегрированной среде разработки Visual Studio. Будет также выведена показанная на рис. 52.3 панель инструментов, позволяющая сделать паузу, остановить, отменить или продолжить запись макроса.

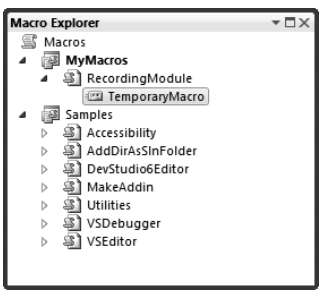


Рис. 52.4

После выполнения действий, которые вы хотите записать, щелкните на кнопке остановки записи на панели инструментов макрорегистратора (или вновь воспользуйтесь комбинацией клавиш <Ctrl+Shift+R>). Теперь, если вы посмотрите на окно Macro Explorer, то увидите модуль, созданный (по умолчанию) в проекте `MyMacros` под названием `RecordingModule`, и макрос `TemporaryMacro` (рис. 52.4).

Затем необходимо переименовать макрос, выбрав команду Rename контекстного меню, которое вызывается после щелчка правой кнопкой мыши на макросе (можно также выбрать макрос и нажать клавишу <F2>), и введя новое имя макроса; так вы избежите перезаписи последнего при новой записи макроса. Теперь вы можете запустить макрос, дважды щелкнув на его имени, и протестировать его работу.



Изменить проект для записи макроса можно с помощью щелчка правой кнопкой мыши на проекте в окне Macro Explorer и выбора команды меню Set as Recording Project. Однако модуль, в который записывается макрос, изменить нельзя — это всегда модуль `RecordingModule`.

Хотя макрорегистратор и помогает при создании макроса, он не записывает все выполняемые вами в среде Visual Studio действия, и вряд ли в состоянии записать логику ваших действий. Поэтому обычно приходится вручную модифицировать код макроса и добавлять в него необходимые действия и логику.

Разработка макросов

При создании макроса вы можете как выбрать создание нового проекта (полезно, если вы хотите распространять этот макрос среди других разработчиков), так и использовать уже существующий. Если вы хотите создать новый проект макроса, то пусть вас не удивляет то, что вы не найдете возможности такого создания в интегрированной среде разработки Macros.

Вместо этого необходимо вернуться в интегрированную среду разработки Visual Studio либо из окна Macro Explorer, щелкнув правой кнопкой мыши на корневом узле Macros и выбрав New Macro Project из контекстного меню, либо с помощью команды меню Tools⇨Macros⇨New Macro Project. После этого появится диалоговое окно, в котором можно присвоить имя проекту, а затем проект будет создан вместе с модулем по умолчанию с именем Module1 (которое следует заменить чем-то более вразумительным). В противном случае можно воспользоваться существующим проектом макроса (вероятно, наилучшим выбором будет проект MyMacros). После этого можно начать добавление открытых функций в код модуля — каждая из них становится макросом.



Написание макросов можно ускорить. Один из способов — ознакомиться с образцами из проекта Samples и разобраться, как они работают. Попробуйте найти среди них тот, который решает задачу, напоминающую вашу. Другой способ заключается в том, чтобы начать с записи макроса, а затем разобраться в сгенерированном макрорекордером коде. Вы сможете использовать фрагменты этого кода при работе над своим макросом.

При необходимости можно добавить дополнительные модули, щелкнув правой кнопкой мыши на проекте макроса и выбрав из контекстного меню команду Add⇨Add New Item. Можно также импортировать в свой проект существующие модули (например, готовые модули из стандартного проекта VB), что делается с помощью команды Add⇨Add Existing Item контекстного меню, вызываемого щелчком правой кнопкой мыши на проекте, после чего из появившегося диалогового окна выбирается модуль, который нужно импортировать. Конечно, можно указать и сборку .NET (написанную на любом языке .NET), содержащую функции, которые вы хотели бы использовать в своем макросе, — с помощью команды Add Reference контекстного меню, вызываемого щелчком правой кнопкой мыши на узле References проекта макроса, с последующим выбором сборки в появившемся диалоговом окне. Однако, если вы хотите распространять свой макрос среди других разработчиков, подумайте о создании надстройки вместо макроса.

Создадим очень простой макрос, который вставляет некоторый текст в текущей позиции в редакторе кода для указания того, что код под ним модифицирован (текст имеет вид `// Modified by [Name] at [DateTime]`, где вместо значений в квадратных скобках указаны соответствующие значения).

Поскольку планируется развертывание макроса на других машинах, создадим отдельный проект макроса под названием CodeNotes (щелкните правой кнопкой мыши на узле Macros в окне Macro Explorer и выберите из контекстного меню команду New Macro Project). Переименуйте модуль по умолчанию Module1 в CodeModificationNotes (щелкните на модуле правой кнопкой мыши и выберите в контекстном меню команду Rename). Дважды щелкните на модуле CodeModificationNotes, чтобы открыть его в интегрированной среде разработки Macros. Добавьте в модуль следующую функцию (это и будет макрос):

```
Public Sub InsertCodeModificationNote()
  Const MY_NAME As String = "Chris Anderson"
  Dim textSelection As EnvDTE.TextSelection

  textSelection = CType(DTE.ActiveDocument.Selection(), EnvDTE.TextSelection)
  textSelection.Text = "// Modified by " & MY_NAME & " at " & DateTime.Now
End Sub
```

Теперь можете отладить свой макрос, помещая курсор в функцию и нажимая <F5> (Run). Интегрированная среда разработки Visual Studio становится активной, а после выполнения макроса вы вновь возвращаетесь в интегрированную среду разработки Macros. Если вы вернетесь в интегрированную среду разработки Visual Studio и посмотрите на окно Macro Explorer, то увидите, что теперь ниже модуля появился макрос, который может быть запущен двойным щелчком мыши на его имени.

Выполнение макроса

Имеется несколько способов запуска созданного макроса. В частности, можно

- дважды щелкнуть на имени макроса в окне Macro Explorer;
- воспользоваться клавишей <F5> в интегрированной среде разработки Macros, когда курсор находится в исходном тексте макроса.

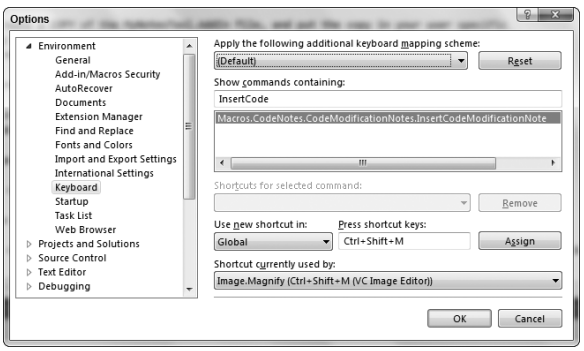


Рис. 52.5

Кроме того, можно назначить комбинацию клавиш для запуска регулярно используемого макроса, чтобы ускорить его запуск. Воспользуйтесь командой меню Tools⇨ Options и найдите узел Keyboard в категории Environment. Найдите свой макрос по имени (введя его в текстовое поле Show Commands Containing) и выберите его из списка. Выберите текстовое поле Press Shortcut Keys и нажмите комбинацию клавиш, которая должна активизировать макрос (рис. 52.5). Убедитесь, что комбинация клавиш в настоящее время не используется (в противном случае вы ее перезапишете), и щелкните на кнопке Assign. Теперь, когда вы нажмете упомянутую комбинацию клавиш в интегрированной среде разработки Visual Studio, ваш макрос будет выполнен.

В случае макросов с продолжительным временем выполнения в нижнем правом углу интегрированной среды разработки и на панели инструментов появится пиктограмма ленты. После щелчка правой кнопкой мыши на этой пиктограмме вы можете с помощью контекстного меню остановить любой выполняющийся в настоящее время макрос.

Отладка макросов аналогична отладке приложения .NET любого другого типа. Просто запустите свой проект макроса в режиме отладки, нажав клавишу <F5> или выбрав соответствующие команды меню в интегрированной среде разработки Macros. В вашем распоряжении будут те же инструменты отладки, что и в среде Visual Studio, позволяющие устанавливать точки останова, шаг за шагом проходить код и просматривать значения переменных, наводя на них указатель мыши.

Развертывание макросов

Поскольку макросы не компилируются, при их распространении вы фактически распространяете исходный текст, видимый любому пользователю. Поэтому перед распространением макроса убедитесь, что в нем нет исходного текста, который вы хотели бы убрать подальше от чужих глаз. Вы не можете распространять отдельный макрос — это должен быть проект макроса целиком. Если в проекте есть другие макросы, которые вы не хотели бы распространять, перенесите интересующий вас макрос в новый проект и распространяйте уже его.

Чтобы макрос был автоматически доступен разработчику в системе Visual Studio, проект должен находиться в папке `My Documents\Visual Studio 2010\Projects\VSMacros80` пользовательского профиля Windows.

Как и в случае надстроек, вы можете передать макрос на другую машину с помощью простой операции XCOPY в виде пакета инсталлятора Windows (`.msi`) или пакета Visual Studio Content Installer. Пожалуй, наилучшим выбором является создание пакета Visual Studio Content Installer, и создается для макросов он так же, как для надстроек. Рассмотрим развертывание проекта макроса с применением пакета Visual Studio Content Installer.

Создадим там же, где находится `.vsmacros`-файл, текстовый файл `CodeNotes.vscontent` со следующим содержимым:

```
<?xml version="1.0" encoding="utf-8"?>
<VSContent xmlns="http://schemas.microsoft.com/developer/vscontent/2005">
  <Content>
    <FileName>CodeNotes.vsmacros</FileName>
    <DisplayName>Code Notes</DisplayName>
    <Description>A macro project to put notes in code.</Description>
    <FileContentType>Macro Project</FileContentType>
    <ContentVersion>1.0</ContentVersion>
  </Content>
</VSContent>
```

Теперь в проводнике Windows Explorer (или в вашем любимом инструменте для работы с zip-архивами) запакуйте в zip-файл файлы `CodeNotes.vsmacros` и `CodeNotes.vscontent` и присвойте ему имя `CodeNotes.vsi` (не давайте ему расширение `.zip`). Теперь, когда кто-то дважды щелкнет на `.vsi`-файле, макрос будет автоматически установлен и готов к работе при следующем запуске системы Visual Studio.

Резюме

В этой главе вы познакомились с макросами системы Visual Studio и их применением для автоматизации повторяющихся действий и узнали, как они записываются или создаются “с нуля” с использованием модели автоматизации Visual Studio. Теперь вы умеете поддерживать макросы в системе Visual Studio, а также создавать, запускать и передавать их другим разработчикам.

53

Библиотека Managed Extensibility Framework (MEF)

В ЭТОЙ ГЛАВЕ...

- Архитектура расширяемых приложений
- Использование библиотеки Managed Extensibility Framework в приложениях
- Компоненты редактора Visual Studio 2010 Editor
- Расширение редактора Visual Studio 2010 Editor
- Импорт служб Visual Studio Services

Создание слабо связанных приложений, которые могут быть расширены после развертывания, — процесс не из простых. Вам придется принять множество проектных решений, включая такие, как идентификация и загрузка расширений, которые должны быть развернуты, а также создание служб приложений, доступных для загрузки расширений. Упростить создание расширяемых приложений призвана разработанная компанией Microsoft библиотека с открытыми исходными текстами Managed Extensibility Framework (MEF). Она позволяет выносить повторно используемые части приложения в надстройки или расширения, которые обнаруживаются и загружаются во время выполнения приложения, и создавать приложения в очень слабо связанном стиле.

Система Visual Studio 2010 использует библиотеку MEF для предоставления точек расширения редактора, но ожидается, что в будущем применение этой библиотеки будет существенно шире.

Эта глава состоит из трех разделов. В первом содержится введение в библиотеку MEF и описываются ее работа и применение в приложениях. Во втором разделе описываются компоненты нового управляющего элемента Visual Studio 2010 Editor и их взаимодействие. В последнем разделе речь идет о расширении редактора с помощью библиотеки MEF и содержится пример с выделением некоторых типов комментариев в вашем коде.



На момент написания книги библиотека MEF все еще находилась на стадии разработки, так что некоторые описанные детали могут быть изменены. Свежую информацию о библиотеке MEF можно получить на веб-сайте по адресу `http://mef.codeplex.com`.

Основы работы с библиотекой MEF

В этом разделе будет создано простое приложение, демонстрирующее способ использования библиотеки MEF. Эта библиотека содержится в сборке `System.ComponentModel.Composition`, инсталлируемой в кэше GAC в качестве части платформы .NET Framework 4.0.

Ключевым компонентом MEF является `CompositionContainer`, находящийся в пространстве имен `System.ComponentModel.Composition.Hosting`. *Контейнер композиции* отвечает за создание *компонуемых частей* приложения, которые в реализации библиотеки MEF по умолчанию представляют собой обычные объекты платформы .NET. Эти части могут как содержать фундаментальные аспекты вашего приложения, так и поступать из внешних сборок расширений, динамически загружаемых во время работы приложения.

Каждая часть может предоставлять одну или более точек *экспорта*, в которых нуждаются другие компонуемые части, и требовать от других частей предоставления точек *импорта*. Точки импорта и экспорта могут быть простыми свойствами или полями, а могут быть и целыми классами. При запросе части из контейнера композиции он пытается обнаружить запрашиваемую часть и удовлетворить все зависимости от импорта, которые та может иметь. Каждый импорт предоставляется (экспортируется) другими частями, о которых осведомлен контейнер и которые, в свою очередь, могут иметь свои требования к импорту, которые также должны быть удовлетворены.

Для создания пустого приложения MEF создайте проект командной строки, добавьте в него ссылку на сборку `System.ComponentModel.Composition` и замените содержимое файла `Program.cs` (C#) или `Module1.vb` (VB) следующим кодом.



C#

```
using System.ComponentModel.Composition;
using System.ComponentModel.Composition.Hosting;

namespace GettingStartedCS
{
  class Program
  {
    static void Main(string[] args)
    {
      var app = new ApplicationRoot();
      app.Run();
    }
  }

  class ApplicationRoot
  {
    public void Run()
    {
      Compose();
    }
    private void Compose()
```

```
    {
      var compositionContainer = new CompositionContainer();
      compositionContainer.ComposeParts(this);
    }
  }
}
```

Сниппет кода GettingStarted\Program.cs



VB

```
Imports System.ComponentModel.Composition
Imports System.ComponentModel.Composition.Hosting

Module Module1

  Sub Main()
    Dim app As New ApplicationRoot
    app.Run()
  End Sub

End Module

Class ApplicationRoot
  Sub Run()
    Compose()
    Console.WriteLine("OK")
  End Sub

  Private Sub Compose()
    Dim compositionContainer As New CompositionContainer
    compositionContainer.ComposeParts(Me)
  End Sub
End Class
```

Сниппет кода GettingStarted\Module1.vb



Метод `ComposeParts` является методом расширения в пространстве имен `System.ComponentModel.Composition`, так что, если это пространство имен не будет включено, приведенный код не будет компилироваться.

Приведенный пример создает контейнер `CompositionContainer`, а затем требует от него *скомпоновать* класс `ApplicationRoot`. Метод `ComposeParts` удовлетворяет требования импорта компонуемых частей. Если он не в состоянии этого сделать, генерируется исключение `System.ComponentModel.Composition.CompositionException`. Поскольку никаких требований к импорту у класса `ApplicationRoot` нет, приложение просто выводит на консоль `OK` и завершает работу. Это не очень впечатляюще, но это основа, к которой можно добавлять другую функциональность.

Импорт и экспорт

В предыдущем примере исходного текста контейнер должен удовлетворять требования импорта класса `ApplicationRoot`. Перед тем как добавить к классу требования импорта, вы должны экспортировать класс, удовлетворяющий зависимостям. Класс `ApplicationRoot` выводит сообщение о состоянии дел по завершении компоновки. Вы можете делегировать эту функциональность другому классу и запросить у контейнера экземпляр этого класса в процессе компоновки.

Чтобы сделать часть доступной остальным частям вашей программы, ее можно экспортировать с помощью атрибута Export. Приведенный далее фрагмент кода создает и экспортирует простой класс:



C#

```
[System.ComponentModel.Composition.Export]
class StatusNotificationService
{
  public void ShowStatus(string statusText)
  {
    System.Console.WriteLine(statusText);
  }
}
```

Сниппет кода GettingStarted\StatusNotificationService.cs



VB

```
<System.ComponentModel.Composition.Export()>
Public Class StatusNotificationService
  Public Sub ShowStatus(ByVal statusText As String)
    System.Console.WriteLine(statusText)
  End Sub
End Class
```

Сниппет кода GettingStarted\StatusNotificationService.vb

После добавления атрибута Export в класс StatusNotificationService библиотека MEF рассматривает последний как компонуемую часть. Заметим, однако, что атрибут Export — это просто метаданные, так что библиотека MEF все еще не знает о существовании этой части и не будет ее использовать. Простейший способ сделать часть доступной для библиотеки MEF во время компоновки частей — передать экземпляр экспортируемого класса методу ComposeParts. Изменим метод Compose класса ApplicationRoot так, чтобы он создавал экземпляр класса StatusNotification и передавал его методу ComposeParts в качестве второго параметра.

Наконец, чтобы указать, что класс ApplicationRoot требует экземпляр этой части, в класс ApplicationRoot надо добавить свойство с атрибутом Import. Приведенный далее фрагмент кода представляет собой полный листинг класса ApplicationRoot. После вызова Compose в методе Run используется импортированная часть:



C#

```
class ApplicationRoot
{
  public void Run()
  {
    Compose();
    NotifcationService.ShowStatus("Composition Complete");
  }

  public void Compose()
  {
    var compositionContainer = new CompositionContainer();
    var statusNotificationService =
        new StatusNotificationService();
    compositionContainer.ComposeParts(this,
                         statusNotificationService);
  }
```

```
  [System.ComponentModel.Composition.Import]
  public StatusNotificationService NotifcationService{
                                          get; set; }
}
```

Сниппет кода GettingStarted\Program.cs



VB

```
Class ApplicationRoot
  Sub Run()
    Compose()
    NotificationService.ShowStatus("Composition Complete")
  End Sub

  Private Sub Compose()
    Dim compositionContainer As New CompositionContainer
    Dim statusNotificationService _
                          As New StatusNotificationService
    compositionContainer.ComposeParts(Me, _
                          statusNotificationService)
  End Sub

  <System.ComponentModel.Composition.Import()>
  Property NotificationService() As StatusNotificationService
End Class
```

Сниппет кода GettingStarted\Module1.vb

Контракты

Когда в процессе компоновки контейнер пытается разрешить зависимости, он использует для сопоставления импорта и экспорта строку, именуемую *контрактом*. По умолчанию при отсутствии контракта библиотека MEF будет использовать в качестве контракта полностью квалифицированное имя типа экспортируемого элемента. Вы можете перекрывать контракт, передавая строку или тип конструктору атрибута `Import` или `Export`. Приведенный далее фрагмент кода демонстрирует три экспорта, имеющих один и тот же контракт:



C#

```
class Settings
{
  [Export]
  public string Username;

  [Export(typeof(string))]
  public string Password;

  [Export("System.String")]
  public string Server;
}
```

Сниппет кода GettingStarted\Settings.cs



VB

```
Public Class Settings
  <Export()>
  Dim Username As String

  <Export(GetType(String))>
  Dim Password As String
```

```
  <Export("System.String")>
  Dim Server As String
End Class
```

Сниппет кода GettingStarted\Settings.vb



Рекомендуется использовать для конракта тип, поскольку полностью квалифицированное имя типа окажется уникальным с большей вероятностью. Если вам необходимо использовать строковые контракты, примите меры, обеспечивающие их уникальность.

При необходимости можете использовать контракт, отличающийся от экспортируемого типа. Основная причина для этого — тип, реализующий интерфейс или наследующий абстрактный базовый класс. В приведенном далее примере класс `SaveOperation` ничего не знает о конкретном отправителе сообщений, который будет использоваться, а потому импортирует абстракцию — `IMessageService`. Класс `CommandLineMessageService` экспортирует сам себя с контрактом интерфейса `IMessageService`. Таким образом, класс `SaveOperation` способен воспользоваться возможностью отправки сообщений, не беспокоясь о деталях того, каким образом эти сообщения будут пересылаться. Если вы захотите впоследствии изменить способ работы приложения, то сможете реализовать новый интерфейс `IMessageService` и заменить конкретный тип, экспортируемый контрактом.



C#

```
public interface IMessageService
{
  void SendMessage(string message);
}

[Export(typeof(IMessageService))]
public class CommandLineMessageService : IMessageService
{
  public void SendMessage(string message)
  {
    Console.WriteLine(message);
  }
}

public class SaveOperation
{
  [Import]
  public IMessageService MessageService { get; set; }

  public void DoSave()
  {
    MessageService.SendMessage("Saving...");
    // Выполнение операции сохранения
    MessageService.SendMessage("Saved");
  }
}
```

Сниппет кода GettingStarted\SaveOperation.cs



VB

```
Public Interface IMessageService
  Sub SendMessage(ByVal message As String)
End Interface
```

```
<Export(GetType(IMessageService))>
Public Class CommandLineMessageService
  Implements IMessageService

  Public Sub SendMessage(ByVal message As String) _
    Implements IMessageService.SendMessage
      Console.WriteLine(message)
    End Sub
  End Class

  Public Class SaveOperation
  <Import()>
  Public Property MessageService As IMessageService

  Public Sub DoSave()
    MessageService.SendMessage("Saving...")
    ' Выполнение операции сохранения
    MessageService.SendMessage("Saved")
  End Sub
End Class
```

Сниппет кода GettingStarted\SaveOperation.vb



Экспорт абстракций и строк потенциально может привести к проблемам. При наличии нескольких экспортов с одним и тем же контрактом MEF не будет знать, какой из них использовать для удовлетворения данного импорта. В этой ситуации можно импортировать для контракта перечислимую коллекцию вместо единственного экземпляра с помощью атрибута `ImportMany`. Можно также присоединить к экспорту дополнительные метаданные для уточнения импорта. Дополнительную информацию по этому вопросу можно найти по адресу `http://mef.codeplex.com`.

Каталоги

В приведенном примере исходного текста имеется только один способ уведомить `CompositionContainer` о компонуемых частях — передать экземпляры методу `ComposeParts`. Это означает, что ваше приложение должно знать о каждой добавляемой в контейнер части и не будет работать с расширениями, добавляемыми после выпуска приложения. Кроме того, со временем это становится утомительным.

Обнаружение частей — работа *каталога*, который может быть передан конструктору `CompositionContainer`. Если контейнер конструируется с каталогом, он может обращаться к последнему при необходимости обнаружить экспорт. Библиотека MEF поставляется с четырьмя каталогами.

- `TypeCatalog` создается со списком типов частей. Экземпляры этих частей создаются по требованию контейнера для удовлетворения требований импорта в процессе компоновки частей.
- `AssemblyCatalog` подобен `TypeCatalog` за исключением того, что сканирует в поисках типов частей всю сборку.
- `DirectoryCatalog` сканирует структуру папок в поисках сборок, которые просматриваются на предмет наличия типов частей.

- `AggregateCatalog` коллекционирует части из ряда других каталогов. Это оказывается полезным, так как конструктор контейнера способен принять только один каталог.

Приведенный далее фрагмент кода демонстрирует создание контейнера, который ищет части в текущей выполняемой сборке и во всех сборках в папке `/Extensions`:



C#

```
var assemblyCatalog =
        new AssemblyCatalog(Assembly.GetExecutingAssembly());
var directoryCatalog =
        new DirectoryCatalog(@".\Extensions\");
var aggregateCatalog =
        new AggregateCatalog(assemblyCatalog, directoryCatalog);
var compositionContainer =
        new CompositionContainer(aggregateCatalog);
```



VB

```
Dim assemblyCatalog As New
AssemblyCatalog(Assembly.GetExecutingAssembly())
Dim directoryCatalog As New DirectoryCatalog(".\Extensions\")
Dim aggregateCatalog As New AggregateCatalog(assemblyCatalog,
directoryCatalog)
Dim compositionContainer As New CompositionContainer(AggregateCatalog)
```



Можно создать собственный каталог, создав класс, наследующий `ComposablePartCatalog` и перекрывающий свойство `Parts`.

Возможности библиотеки MEF

Библиотека MEF поддерживает ряд передовых возможностей, которые могут оказаться полезными при создании расширяемых приложений, а также надстроек и расширений для других приложений. К ним относятся следующие:

- экспорт свойств, полей и методов;
- импорт полей, методов и аргументов конструкторов;
- импорт коллекций;
- пакетная компоновка и перекомпоновка;
- отложенный импорт;
- фильтрация каталога;
- время жизни частей;
- импорт и экспорт пользовательских метаданных.

Дополнительную информацию по этим и другим вопросам, связанным с MEF, можно найти в MEF Programming Guide по адресу `http://mef.codeplex.com`.

Редактор Visual Studio 2010 Editor

Одним из наиболее значительных изменений в системе Visual Studio 2010 является новый редактор кода и текста, написанный на управляемом коде. Этот новый редак-

тор использует библиотеку MEF для управления своей структурой, что означает импорт множества предопределенных контрактов. В дополнение к этому он экспортирует ряд служб с предопределенными контрактами, которые предоставляют доступ к уровню представления и лежащей в его основе модели редактора. Новый редактор собран из четырех основных подсистем.

Подсистема модели текста

Подсистема *модели текста* (Text Model) используется для представления текста и допускает его модификацию. Это сугубо логическая модель, которая никоим образом не связана с выводом на экран.

Основным компонентом этой системы является интерфейс `ITextBuffer`, представляющий последовательность символов, которую редактор должен выводить на экран. Интерфейс `ITextBuffer` может быть сохранен в файловой системе как объект класса `ITextDocument`, но это необязательно. Представление может полностью располагаться в оперативной памяти. Для создания новых экземпляров класса `ITextBuffer` можно использовать класс `ITextBufferFactoryService`. Вносить изменения в объект класса `ITextBuffer` может любое количество потоков, пока один из них не вызовет метод `TakeThreadOwnership`.

При изменении объекта `ITextBuffer` создается новая версия. Каждая версия представлена как неизменяемый экземпляр класса `ITextSnapshot`. Так как эти снимки не могут изменяться, к ним может безопасно обращаться любое количество потоков, даже если исходный объект класса `ITextBuffer` и продолжает изменяться.

Для внесения изменения в объект класса `ITextBuffer` можно использовать метод `CreateEdit`, чтобы создать экземпляр интерфейса `ITextEdit`. Интерфейс `ITextEdit` позволяет заменить фрагмент текста в буфере новым набором символов. Экземпляр класса `ITextEdit` может быть применен к объекту класса `ITextBuffer` путем вызова метода `Apply`. Его можно удалить путем вызова метода `Cancel` или `Dispose`. В любой момент для класса `ITextBuffer` может быть создан только один экземпляр класса `ITextEdit`, и если буфером владеет некоторый конкретный поток, то только он может создавать эти экземпляры.



Интерфейс `ITextBuffer` содержит методы `Insert`, `Replace` и `Delete`, которые служат "оберткой" для создания и применения экземпляров `ITextEdit`.

Все операции одного объекта класса `ITextEdit` выполняются относительно начального состояния объекта класса `ITextBuffer` в момент создания этого экземпляра класса `ITextEdit`. Из-за этого вы не можете вставить некоторый текст, а затем снова удалить его в пределах единственного экземпляра класса `ITextEdit`.

В процессе применения интерфейса `ITextEdit` создаются новые экземпляры классов `ITextVersion` и `ITextSnapshot` и генерируется событие `Changed`. `ITextVersion` представляет изменения между текущим состоянием `ITextBuffer` и предыдущим состоянием, в то время как `ITextSnapshot` является представлением `ITextBuffer` только для чтения после применения `ITextEdit`. Изменения в `ITextVersion` представлены в виде списка экземпляров `ITextChange`, каждый из которых, будучи применен к снимку, дает очередной снимок в последовательности. Для последней версии эта коллекция всегда пуста (имеет значение `null`).

Подсистема представления текста

Подсистема *представления текста* (Text View) отвечает за управление выводом текста на экран. Сюда входит определение того, какие строки должны выводиться и как должен форматироваться выводимый текст. Она также отвечает за дополнительные визуальные расширения, такие как волнистая линия, указывающая на ошибку компиляции. Наконец, эта подсистема управляет полями, на которых может располагаться дополнительная информация.

Главной частью данной подсистемы является интерфейс `ITextView`. Его экземпляры используются для визуального представления текста на экране. Они применяются как для главного окна редактора, так и для, например, текста всплывающей подсказки. `ITextView` отслеживает три различных текстовых буфера посредством свойства `TextViewModel`. Это

- буфер данных, который представляет фактический текст;
- буфер редактирования, в котором выполняется редактирование текста;
- визуальный буфер, выводимый на экран.

Текст форматируется с помощью *классификаторов* (см. раздел "Подсистема классификации") и декорируется так называемыми украшениями, которые поступают от провайдера украшений, присоединенного к представлению текста.

Выводимая на экран часть текста представляет собой *область просмотра* (view port). Эта область определяет логическую систему координат, в которой точка (0,0) находится в верхнем левом углу текста. Если редактор не "раскрыт" и не преобразован каким-то способом, единица расстояния в представлении эквивалентна одному пикселю. Каждая строка текста, выводимая на экран, представляет собой экземпляр интерфейса `ITextViewLine`. Этот интерфейс может использоваться для обратного отображения пикселей на символы.

Наконец, весь редактор, все украшения и поля содержатся в `IWpfTextViewHost`.



Подсистема представления текста поступает в двух частях. Первая технологически независима и располагается в сборке `Microsoft.Visual Studio.Text.UI`. Вторая же представляет собой реализацию технологии WPF и располагается в сборке `Microsoft.VisualStudio.Text.UI.WPF`. В большинстве случаев в именах элементов, специфичных для технологии WPF, содержится текст "`Wpf`".

Подсистема классификации

Подсистема классификации управляет распознаванием и форматированием различных типов текста. Она также отвечает за оснащение текста дополнительными метаданными, которые могут использоваться подсистемой представления текста для добавления различных визуальных элементов, а также для выделения цветом текста и его схемы вывода (например, вывод части кода в свернутом виде).

Операционная подсистема

Операционная подсистема определяет поведение и команды редактора. Она также обеспечивает возможность отказа от внесенных изменений.

Расширение редактора

Расширения редактора представляют собой `.vsix`-пакеты, которые экспортируют контракты, импортируемые компонентами системы Visual Studio. Когда система Visual Studio загружает эти пакеты, их содержимое добавляется в каталог `MEF`, который затем используется для компоновки частей редактора. Пакет Visual Studio Integration SDK поставляется с рядом шаблонов, позволяющих создавать редакторы. Они находятся на странице Extensibility диалогового окна New Project, показанного на рис. 53.1.

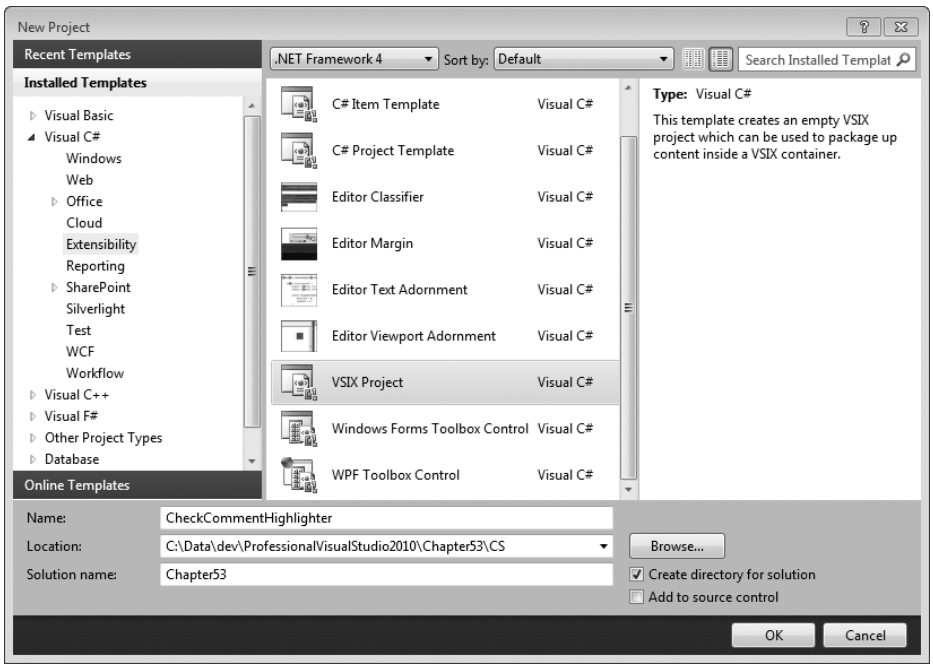


Рис. 53.1



Пакет Visual Studio 2010 SDK не устанавливается вместе с системой Visual Studio 2010. Вы можете найти этот пакет по адресу `http://msdn.microsoft.com/en-us/vsx/default.aspx`.

Воспользуйтесь шаблоном VSIX Project. Для доступа к расширениям редактора посредством данного пакета отредактируйте файл `source.extension.vsixmanifest` и воспользуйтесь кнопкой Add Content для добавления текущего проекта в качестве MEF Component, как показано на рис. 53.2.

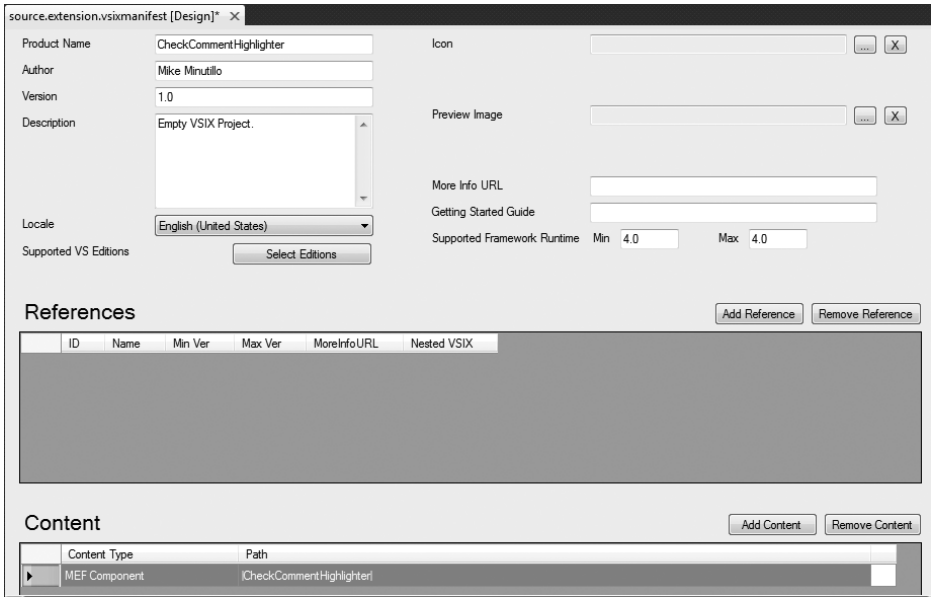


Рис. 53.2

После того как ваш проект настроен как содержащий MEF-содержимое, все, что вам нужно, — это создать классы, которые экспортируют известные контракты расширений, и система Visual Studio автоматически найдет их. В дополнение к этому вы можете импортировать контракты служб из системы Visual Studio, которые обеспечат вам доступ ко всем возможностям редактора.

В процессе разработки расширения редактора могут запускаться и отлаживаться в экспериментальном экземпляре системы Visual Studio (Experimental Instance of Visual Studio). Этот экспериментальный экземпляр ведет себя как отдельная инсталляция системы Visual Studio со своими настройками и реестром. Он также управляет отдельным набором расширений. Когда вы будете готовы к развертыванию своего расширения для обычного экземпляра системы Visual Studio, дважды щелкните на пакете `.vsix`, созданном как часть процесса сборки. Этот пакет самодостаточный, так что его можно использовать и для передачи вашего расширения на другие компьютеры.

Точки расширения редактора

Редактор Visual Studio 2010 Editor ищет множество контрактов, которые используются для расширения поведения редактора во время его работы. Обычно необходимо создать как минимум два класса для каждого типа расширения. Один класс выполняет всю необходимую работу, а второй обычно импортируется системой Visual Studio и по требованию предоставляет экземпляры основного класса расширения.

Типы содержимого

Каждому `ITextBuffer` при создании назначается *тип содержимого*, который идентифицирует тип содержащегося в нем текста. Примеры типов содержимого включают `Text`, `Code`, `CSharp` и `Basic`. Типы содержимого используются в качестве фильтров

для различных расширений редактора, которые можно создавать путем добавления атрибута `ContentType` в экспортируемое расширение. Примером может служить провайдер IntelliSense, работающий только с XML-содержимым.



Несмотря на то что тип содержимого назначается `ITextBuffer` при создании последнего, этот тип может быть изменен с помощью метода `ChangeContentType`.

Вы можете создавать собственные типы содержимого путем экспорта свойства или поля с контрактом `ContentTypeDefinition`. Каждый тип содержимого может иметь несколько родительских типов содержимого, которые определены путем добавления атрибута `BaseDefinition` к экспортируемому типу содержимого для каждого родительского типа. Для получения полного списка типов содержимого можно импортировать интерфейс `IContentTypeRegistryService`, который поддерживает список зарегистрированных типов содержимого.



Тип содержимого может быть связан с расширением файла с помощью атрибута `FileExtension`. Обратите внимание, что расширение файла должно быть одним из уже зарегистрированных расширений Visual Studio. Дополнительную информацию по этому вопросу вы можете найти в документации MSDN, выполнив запрос “ProvideLanguageExtension Attribute Class”.

Типы и форматы классификации

Тип классификации представляет собой метаданные, которые могут быть применены к любому фрагменту текста. Примерами типов классификации могут служить “ключевое слово” и “комментарий”, которые наследуют тип классификации “код”. Можно создавать собственные типы классификации, экспортируя свойство или поле класса `ClassificationTypeDefinition`. Так вы можете назначать тексту пользовательское поведение.



Типы классификации отличаются от типов содержимого. Каждый объект класса `ITextBuffer` имеет единственный тип содержимого, но может содержать фрагменты с несколькими классификациями.

Типы классификации назначаются фрагментам текста с использованием экземпляра `IClassifier`. *Агрегатор классификаторов* собирает классификации от различных классификаторов текстового буфера и создает для этого буфера уникальное неперекрывающееся множество классификаций. По сути агрегатор классификаторов сам по себе является классификатором, поскольку он также предоставляет классификации для фрагмента текста. Чтобы получить агрегатор классификаторов для конкретного экземпляра класса `ITextBuffer`, можно импортировать интерфейс `IclassificationAggregatorService` и вызвать его метод `GetClassifier`, передав ему текстовый буфер.

Можно определить формат для определенного типа классификации, порождая новый класс из `ClassificationFormatDefinition` и экспортируя его с контрактом `EditorFormatDefinition`. Базовый класс содержит ряд свойств, которые можно использовать для изменения способа вывода текста на экран. Определение формата свя-

зывается с типом классификации с помощью атрибута `ClassificationType` экспортируемого класса. Этот атрибут принимает строку, которая представляет собой список с разделителем-запятой типов классификации, к которым применим данный формат. Можно использовать и атрибуты `DisplayName` и `UserVisible` для вывода этого формата на странице Fonts and Settings диалогового окна Options. Можно также указать атрибут `Priority`, который помогает определить, когда применяется данный формат.

Поля

Поле (margin) представляет собой часть пользовательского интерфейса вокруг границ главного окна редактора. Поля имеют четыре предопределенных имени, `Top`, `Bottom`, `Left` и `Right`, и выступают в роли контейнеров для других полей, которые вы можете определить. Вы можете, например, определить поле, которое становится красным при открытии сгенерированного файла, предупреждая таким образом, что этот файл не следует редактировать.

Для создания поля необходимо написать класс, реализующий интерфейс `IWpfTextViewMargin`, который содержит свойства как для размера поля, так и для фактического элемента `UIElement`, который выводит поля экрана. Для регистрации поля в системе Visual Studio необходимо экспортировать класс с контрактом интерфейса `IWpfTextViewMarginProvider`. Этот интерфейс содержит единственный метод, который должен возвращать экземпляр класса `IWpfTextViewMargin`. В дополнение к экспорту библиотеки MEF провайдер полей может предоставлять следующее:

- атрибут `Name`, который используется для назначения провайдеру удобочитаемого имени;
- атрибут `ContentType`, идентифицирующий тип содержимого, для которого поле должно быть сделано доступным;
- атрибуты `Order` и `MarginContainer`, использующиеся для определения того, где должны выводиться поля. Порядок определяется путем передачи строкового имени другого поля, выводимого до или после данного (значение `Before` или `After`). Контейнер идентифицирует границу, у которой должно выводиться поле, — верхнюю, нижнюю, левую или правую.

Дескрипторы

Дескриптор (tag) представляет собой часть метаданных, примененных к определенному фрагменту текста. Примеры дескрипторов включают `SquiggleTag`, `TextMarkerTag` и `OutliningRegionTag`. Дескрипторы связаны с фрагментами текста с помощью экземпляров интерфейса `ITagger`. Для регистрации `ITagger` необходимо создать класс, который реализует интерфейс `ITaggerProvider`, заместить метод `CreateTagger` и экспортировать новый класс с контрактом `ITaggerProvider`. Ваш провайдер должен быть также маркирован атрибутом `TagType`, который определяет тип генерируемого дескриптора.



Классификация является частным случаем назначения дескрипторов, предоставляемых классом `ClassifactionTag`.

Обрамление

Обрамление (adornment) представляет собой специальный эффект, который может быть применен к фрагменту текста или к самой поверхности редактора. Вы можете определять собственные элементы обрамления, которые представляют собой просто стандартные классы `UIElement` технологии WPF. Каждый тип украшения выводится в отдельном слое, так что разные типы украшений не взаимодействуют друг с другом. Для указания слоя, которому принадлежит обрамление, класс обрамления должен экспортировать свойство `AdornmentLayerDefinition` с атрибутами `Name` и `Order`. Атрибут `Order` может быть определен как `Before` или `After` по отношению к одному из четырех встроенных слоев — `Selection`, `Outlining`, `Caret` и `Text`. Когда обрамление выводит само себя, оно запрашивает экземпляр класса `IAdornmentLayer` у интерфейса `IWpfTextView` по имени. Этот интерфейс предоставляет методы для добавления элемента `UIElement` к слою и для очистки слоя от всех обрамлений.

Для создания обрамления необходимо экспортировать класс с контрактом `IWpfTextViewCreationListener`. Этот класс должен просто создавать новый экземпляр обрамления всякий раз при создании текстового представления. Класс может быть маркирован с помощью стандартного атрибута `ContentType` для фильтрации типов содержимого, с которыми он может работать. Атрибут `TextViewRole` определяет, для какого типа текстового представления предназначен этот класс. Список `PredefinedTextViewRoles` содержит корректные значения этого атрибута.

Процессоры мыши

Процессоры мыши (mouse processors) перехватывают события мыши. Каждый процессор мыши должен быть производным от класса `MouseProcessorBase` и перекрывать обработчики тех событий, которые вы хотите обрабатывать. Чтобы система Visual Studio могла работать с вашим процессором мыши, следует экспортировать класс с контрактом интерфейса `IMouseProcessorProvider`. Кроме того, необходимо применить атрибут `ContentType` для идентификации типов содержимого, для которых доступен процессор мыши.

Обработчики перетаскивания

Обработчики перетаскивания (drop handlers) настраивают поведение редактора Visual Studio при перетаскивании в него содержимого. Каждый обработчик должен реализовывать интерфейс `IDropHandler`, а кроме того, для предоставления обработчика в системе Visual Studio нужен провайдер `IDropHandlerProvider`. Класс провайдера должен экспортировать контракт `IDropHandlerProvider` вместе со следующими метаданными.

- Атрибут `Name`, идентифицирующий обработчик.
- Атрибут `DropFormat`, указывающий формат текста, для которого пригоден обработчик. Поддерживаются 23 встроенных формата, обрабатываемых в определенном порядке. Полный список можно найти в документации MSDN.
- Атрибут `Order` для определения, когда среди прочих обработчиков должен вызываться указанный. Это делается с помощью компонентов `Before` и `After` атрибута `Order`. Каждый компонент представляет собой текстовое имя обработ-

чика. Обработчик по умолчанию, предоставляемый Visual Studio, имеет имя `DefaultFileDropHandler`.

Настройки редактора

Настройки редактора позволяют настраивать ваше расширение. Эти настройки могут быть импортированы в другие компоненты системы и использоваться для изменения их поведения. Этот тип экспорта используется для передачи значения вашей настройки другим компонентам, но система Visual Studio ничего не делает для того, чтобы настройки были доступны пользователю. Если вы хотите, чтобы пользователь мог управлять настройками расширения, создайте собственный пользовательский интерфейс.

Для создания настройки редактора следует выполнить наследование одного из трех абстрактных классов, `EditorOptionDefinition<T>`, `ViewOptionDefinition<T>` или `WpfViewOptionDefintion<T>`, и указать тип создаваемого значения настройки (т.е. `Boolean` или `String`). Эти базовые классы предоставляют абстрактные свойства для реализации содержащейся информации о настройке, включая текущее значение и значение по умолчанию. Чтобы сделать настройку редактора доступной для Visual Studio, следует экспортировать ее с контрактом `EditorOptionDefinition`.

Технология IntelliSense

IntelliSense — это термин, использующийся для описания набора возможностей предоставления контекстуальной информации и служб завершения выражений. Независимо от типа предоставляемого вами расширения системы IntelliSense, ее компоненты и процессы всегда одинаковы:

- *брокер* управляет процессом в целом;
- *сеанс* представляет последовательность событий, которая обычно начинается с действия пользователя, запускающего презентатор, и заканчивается подтверждением выбора или отказом от него;
- *контроллер* определяет, когда должен начаться и закончиться сеанс, и решает, чем должен завершиться сеанс;
- *источник* предоставляет содержимое сеанса IntelliSense и определяет наилучшее соответствие для вывода на экран;
- *презентатор* отвечает за вывод содержимого сеанса.

При определении расширения системы IntelliSense рекомендуется предоставлять как минимум источник и контроллер. Презентатор предоставляется только в том случае, когда вы хотите настроить вывод своей функциональной возможности.

Чтобы представить источник системы IntelliSense, создайте класс, реализующий один или более следующих интерфейсов: `ICompletionSource`, `IQuickInfoSource`, `ISignatureHelpSource` и `ISmartTagSource`. Каждый из этих интерфейсов определяет методы, обеспечивающие вас контекстом сеанса и позволяющие вернуть выводимую информацию.

Для каждого из реализованных интерфейсов требуется другой класс, реализующий интерфейс соответствующего провайдера: `ICompletionSourceProvider`, `IQuickInfoSourceProvider`, `ISignatureHelpSourceProvider` или `ISmartTagSourceProvider`. Этот класс провайдера должен быть экспортирован с применением

интерфейса его провайдера в качесве контракта. В дополнение к экспорту можно указать атрибуты `Name`, `Order` и `ContentType`.

Для предоставления контроллера IntelliSense требуется класс, который реализует интерфейс `IIntellisenseController`. Этот интерфейс предоставляет методы для подключения к контроллеру и отключения от него буферов `ITextBuffer`. Когда контроллер обнаруживает событие, которое должно начать сеанс IntelliSense, он запрашивает его у брокера соответствующего типа: `ICompletionBroker`, `IQuickInfoBroker`, `ISignatureHelpBroker` или `ISmartTagBroker`. Простейший способ получить доступ к брокеру заключается в его импорте в провайдер контроллера и передаче конструктору контроллера IntelliSense.

Наконец, требуется провайдер контроллера `IIntellisenseControllerProvider`, который экспортируется с атрибутами `Name`, `Order` и `ContentType`.

Службы редактора

Система Visual Studio предоставляет большое количество служб редактора с хорошо известными контрактами, которые можно импортировать в свои классы расширений. Вот несколько наиболее часто используемых служб (полный список можно найти в документации MSDN).

- Интерфейс `IContentTypeRegistryService` управляет коллекцией типов содержимого, которые могут назначаться `ITextBuffer`. Эта служба позволяет добавлять и удалять типы содержимого, а также запрашивать типы содержимого, зарегистрированные в настоящий момент.
- Интерфейс `ITextDocumentFactoryService` предоставляет возможность создания новых документов и загрузки существующих из файловой системы. Кроме того, имеет события создания и уничтожения `ITextDocument`.
- Интерфейс `IClassifierAggregatorService` содержит единственный метод `GetClassifier`, возвращающий классификатор для данного `ITextBuffer`. Создает и кэширует классификаторы, если они еще не существуют.
- Интерфейс `ITextSearchService` отвечает за поиск некоторого текста в определенной области текста. Имеет методы для поиска как всех экземпляров, так и только одного следующего экземпляра.
- Интерфейс `IWpfKeyboardTrackingService` позволяет переключать отслеживание клавиатуры в редакторе в системе WPF. Обычно система Visual Studio выполняет собственное отслеживание клавиатуры, так что если вы используете управляющие элементы WPF, прослушивающие клавиатурные события, то последние никогда не будут обнаружены. Эта служба позволяет переключать эту возможность так, чтобы система WPF первой обнаруживала обрабатываемые клавиатурные события. Клавиатурные события, оставшиеся необработанными системой WPF, передаются для обычной обработки в среду Visual Studio.

Расширение Check Comment Highlighter

В этом разделе рассматривается полный исходный текст учебного расширения и даются пояснения его работы. В нашем офисе, когда мы сталкиваемся с чем-то, что представляется не вполне верным, мы добавляем комментарий с запросом на уточне-

ние, в котором используем специальное слово `check:`, за которым следуют несколько предложений, поясняющих, что именно показалось сомнительным. Обычно, встретив фрагмент исходного текста с таким комментарием и имея ответ на поставленный вопрос, мы пытаемся выполнить рефакторинг, чтобы ответ на поставленный вопрос стал очевидным, либо добавляем комментарий с пояснениями (а в тех редких случаях, когда комментарий указывает на ошибку, исправляем ее). Этот метод позволяет содержать код в более поддерживаемом, удобочитаемом и понятном состоянии. У нас имеются инструменты, позволяющие извлечь список всех комментариев такого вида, но гораздо удобнее было бы выделять их прямо в окне редактора. Расширение Check Comment Margin Highlighter делает именно это, добавляя на левое поле (где обычно указываются точки прерывания) специальный знак, указывающий строки с комментариями, содержащие `check:`.

Код состоит из двух частей: создатель дескрипторов и фабрика символов. Начнем с исходного текста создателя дескрипторов.



C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel.Composition;
using Microsoft.VisualStudio.Text;
using Microsoft.VisualStudio.Text.Classification;
using Microsoft.VisualStudio.Text.Editor;
using Microsoft.VisualStudio.Text.Tagging;
using Microsoft.VisualStudio.Utilities;

namespace CheckCommentHighlighter
{
  class CheckCommentTag : IGlyphTag { }
  class CheckCommentTagger : ITagger <CheckCommentTag>
  {
    private readonly IClassifier _classifier;
    public CheckCommentTagger(IClassifier classifier)
    {
      _classifier = classifier;
    }
    public IEnumerable < ITagSpan <CheckCommentTag>>
        GetTags(NormalizedSnapshotSpanCollection spans)
    {
      foreach (var span in spans)
      {
        foreach (var classification in
                  _classifier.GetClassificationSpans(span))
        {
          var isComment = classification.ClassificationType
            .Classification
            .ToLower()
            .Contains("comment");
          if (isComment)
          {
            var index = classification.Span.GetText()
              .ToLower().IndexOf("check:");
            if (index != -1)
            {
              var tag = new CheckCommentTag();
              var snapshotSpan = new SnapshotSpan(
                      classification.Span.Start + index, 6);
              yield return
                  new TagSpan < CheckCommentTag > (
                                  snapshotSpan,
```

```
                                        tag);
            }
          }
        }
      }
    }

    public event EventHandler < SnapshotSpanEventArgs >
        TagsChanged;
  }

  [Export(typeof(ITaggerProvider))]
  [TagType(typeof(CheckCommentTag))]
  [ContentType("code")]
  class CheckCommentTaggerProvider : ITaggerProvider
  {
    [Import]
    private IClassifierAggregatorService AggregatorService;
    public ITagger < T >
        CreateTagger < T > (ITextBuffer buffer)
        where T : ITag
    {
      if( buffer == null )
        throw new ArgumentNullException("buffer");
      var classifier =
            AggregatorService.GetClassifier(buffer);
      return new CheckCommentTagger(classifier)
            as ITagger < T > ;
    }
  }
}
```

Сниппет кода CheckCommentHighlighter\CheckCommentTagger.cs



VB

```
Imports System.ComponentModel.Composition
Imports Microsoft.VisualStudio.Text
Imports Microsoft.VisualStudio.Text.Tagging
Imports Microsoft.VisualStudio.Text.Editor
Imports Microsoft.VisualStudio.Text.Classification
Imports Microsoft.VisualStudio.Utilities

Friend Class CheckCommentTag
  Inherits IGlyphTag
End Class

Friend Class CheckCommentTagger
  Implements ITagger(Of CheckCommentTag)

  Private m_classifier As IClassifier

  Friend Sub New(ByVal classifier As IClassifier)
    m_classifier = classifier
  End Sub

  Private Function GetTags(ByVal spans _
    As NormalizedSnapshotSpanCollection)
    As IEnumerable(Of ITagSpan(Of CheckCommentTag))
    Implements ITagger(Of CheckCommentTag).GetTags

    Dim Tags As New List(Of ITagSpan(Of CheckCommentTag))
    For Each span As SnapshotSpan In spans
      For Each classification As ClassificationSpan In
        m_classifier.GetClassificationSpans(span)
        If classification.ClassificationType _
```

```
            .Classification.ToLower()
          .Contains("comment") Then

          Dim index As Integer = _
            classification.Span.GetText().ToLower()
          .IndexOf("check:")

          If index < > -1 Then
            Dim snapshotSpan As _
                New SnapshotSpan(classification.Span.Start
              + index, 6)

            Dim tag As New CheckCommentTag
              Tags.Add(New TagSpan(Of _
                        CheckCommentTag)(snapshotSpan, tag))
          End If
        End If
      Next classification
    Next span
    Return Tags
  End Function

  Public Event TagsChanged _
        As EventHandler(Of SnapshotSpanEventArgs)
    Implements ITagger(Of CheckCommentTag).TagsChanged

End Class

<Export(GetType(ITaggerProvider)), ContentType("code"),
    TagType(GetType(CheckCommentTag))>
Friend Class CheckCommentTaggerProvider
  Implements ITaggerProvider

  <Import()>
  Friend AggregatorService As IClassifierAggregatorService

  Public Function CreateTagger(Of T _
            As ITag)(ByVal buffer As ITextBuffer)
    As ITagger(Of T) Implements ITaggerProvider.CreateTagger

    If buffer Is Nothing Then
      Throw New ArgumentNullException("buffer")
    End If

    Dim Classifier = AggregatorService.GetClassifier(buffer)
    Dim tagger As New CheckCommentTagger(Classifier)

    Return TryCast(tagger, ITagger(Of T))
  End Function
End Class
```

Сниппет кода CheckCommentHighlighter\CheckCommentTagger.vb

Здесь определены три класса. Первый класс — `CheckCommentTag`. Он наследует класс `IGlyphTag`, но собственной реализации не имеет. Это просто маркер для идентификации того, что определенный фрагмент текста должен быть отмечен специальным символом. Мы можем указать некоторые свойства класса дескриптора для передачи информации фабрике символов, которая может использоваться для влияния на вывод `UIElement`.

Второй класс — `CheckCommentTagger`. Он отвечает за идентификацию фрагментов текста, к которым должен применяться дескриптор `CheckCommentTag`. Он делает это путем реализации интерфейса `ITagger<CheckCommentTag>`. Этот интерфейс состоит из метода `GetTags` и события `TagsChanged`. Метод `GetTags` принимает коллекцию фрагментов и возвращает коллекцию `ITagSpans`. В этой реализации он находит все

комментарии с помощью классификатора и ищет подстроку `check:`. Если такая подстрока имеется, он создает новый элемент `TagSpan<CheckCommentTag>`, который применяется только к фрагменту текста с подстрокой `check:`.

Последний класс — `CheckCommentTaggerProvider`, содержащий экспортируемые метаданные MEF, которые Studio ищет в расширении. Этот класс экспортируется с применением контракта `ITaggerProvider`, а это означает, что Visual Studio добавит его ко внутреннему списку провайдеров для последующего вызова при необходимости. К этому классу присоединены две другие части метаданных. Класс `TagTypeAttribute` указывает тип дескрипторов, которые производят генераторы дескрипторов, создаваемые данным провайдером. Класс `ContentTypeAttribute` поставляет фильтр типов содержимого, для которых этот генератор должен использоваться. В данном случае атрибут указывает, что данный провайдер генераторов дескрипторов должен вызываться только тогда, когда редактор содержит код, являющийся обычным базовым типом содержимого редактора.

Класс провайдера генераторов дескрипторов выдвигает требование импорта `IClassifierAggregatorService`. Эта служба используется при конструировании генераторов, происходящем в методе `CreateTagger<T>`. Этот метод получает объект класса `ITextbuffer`, для которого предоставляется генератор. Он использует сначала класс `AggregatorService` для получения классификатора, а затем классификатор — для конструирования объекта класса `CheckCommentTagger`, определенного в предыдущем фрагменте кода.

Этого кода достаточно, чтобы позволить системе Visual Studio маркировать комментарии с подстрокой `check:` как требующие выделения символом, но если вы развернете свое расширение в том виде, в котором оно находится в настоящее время, то не увидите ничего из-за отсутствия компонентов, предназначенных для вывода объекта класса `CheckCommentTag`. Для этого вам нужна *фабрика символов*, которая представляет собой вторую половину расширения. Ниже приведен ее код.



C#

```
using System.ComponentModel.Composition;
using System.Windows;
using System.Windows.Media;
using System.Windows.Shapes;
using Microsoft.VisualStudio.Text.Editor;
using Microsoft.VisualStudio.Text.Formatting;
using Microsoft.VisualStudio.Text.Tagging;
using Microsoft.VisualStudio.Utilities;

namespace CheckCommentHighlighter
{
  class CheckCommentGlyphFactory : IGlyphFactory
  {
    public UIElement GenerateGlyph(IWpfTextViewLine line,
                                   IGlyphTag tag)
    {
      var validTag = tag as CheckCommentTag != null;
      if (!validTag)
        return null;
      return new Polygon
      {
        Fill = Brushes.LightBlue,
        Stroke = Brushes.DarkBlue,
        StrokeThickness = 2,
        Points = new PointCollection
```

```
{
          new Point(0, 0),
          new Point(16, 8),
          new Point(0, 16)
        }
      };
    }
  }

  [Export(typeof(IGlyphFactoryProvider))]
  [TagType(typeof(CheckCommentTag))]
  [Name("CheckCommentGlyph")]
  [ContentType("code")]
  [Order(After="VSTextMarker")]
  class CheckCommentGlyphFactoryProvider
    :IGlyphFactoryProvider
  {
    public IGlyphFactory GetGlyphFactory(IWpfTextView view,
                                         IWpfTextViewMargin margin)
    {
      return new CheckCommentGlyphFactory();
    }
  }
}
```

Сниппет кода CheckCommentHighlighter\CheckCommentGlyphFactory.cs



VB

```
Imports System.ComponentModel.Composition
Imports System.Windows
Imports System.Windows.Media
Imports System.Windows.Shapes

Imports Microsoft.VisualStudio.Text.Editor
Imports Microsoft.VisualStudio.Text.Formatting
Imports Microsoft.VisualStudio.Text.Tagging
Imports Microsoft.VisualStudio.Utilities

Friend Class CheckCommentGlyphFactory
    Implements IGlyphFactory
  Public Function GenerateGlyph(ByVal line As IWpfTextViewLine,
                                ByVal tag As IGlyphTag)
           As UIElement Implements IGlyphFactory.GenerateGlyph
    If tag Is Nothing OrElse Not (TypeOf tag Is CheckCommentTag) Then
      Return Nothing
    End If

    Dim triangle As New System.Windows.Shapes.Polygon()

    With triangle
      .Fill = Brushes.LightBlue
      .Stroke = Brushes.DarkBlue
      .StrokeThickness = 2
      .Points = New PointCollection()
      With .Points
        .Add(New Point(0, 0))
        .Add(New Point(16, 8))
        .Add(New Point(0, 16))
      End With
    End With

    Return triangle
  End Function
End Class
```

```
<Export(GetType(IGlyphFactoryProvider)),
        Name("CheckCommentGlyph"),
        Order(After:="VsTextMarker"),
        ContentType("code"),
        TagType(GetType(CheckCommentTag))>
Friend NotInheritable Class TodoGlyphFactoryProvider
       Implements IGlyphFactoryProvider

  Public Function GetGlyphFactory(
            ByVal view As Microsoft.VisualStudio _
                          .Text.Editor.IWpfTextView,
            ByVal margin As Microsoft.VisualStudio _
                          .Text.Editor.IWpfTextViewMargin)
            As Microsoft.VisualStudio.Text.Editor.IGlyphFactory
    Implements Microsoft.VisualStudio.Text.Editor _
               .IGlyphFactoryProvider.GetGlyphFactory
    Return New CheckCommentGlyphFactory()
  End Function
End Class
```

Сниппет кода CheckCommentHighlighter\CheckCommentGlyphFactory.vb

Так же, как и ранее, здесь работают два класса: класс, создающий графические символы, и класс, по требованию предоставляющий экземпляры фабрики символов Visual Studio. Фабрика `CheckCommentGlyphFactory` очень проста. Она выполняет проверку, чтобы убедиться, что дескриптор — корректного типа, и создает выводимый визуальный элемент. Это может быть любой `UIElement` системы WPF. В данной реализации это голубой треугольник с синей границей, указывающий вправо.

Второй класс представляет собой шлюз в системе Visual Studio. Он экспортируется с использованием контракта `IGlyphFactoryProvider`, связанного с определенным дескриптором и типом содержимого. Он также определяет имя, которое упрощает его идентификацию. Наконец, он указывает, что должен выводиться после элементов в слое “VSTextMarker”; это означает, что он будет выведен поверх элементов этого слоя. Фактическая реализация этого класса представляет собой простой метод фабрики для экземпляров класса `CheckCommentGlyphFactory`.

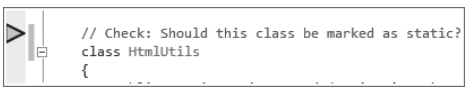


Рис. 53.3

При запуске это расширение будет выполняться в экспериментальном экземпляре системы Visual Studio. Загрузите файл с исходным текстом, добавьте в него комментарий, начинающийся с `Check:`, и вы увидите на левом поле голубой треугольник, как показано на рис. 53.3.

Резюме

Библиотека Managed Extensibility Framework упрощает процесс создания расширяемых приложений, позволяя рассматривать приложение как коллекцию компонуемых частей, каждая из которых осуществляет экспорт и требует импорта. Расширения могут добавляться в приложение путем создания соответствующих каталогов частей с их последующей передачей контейнеру композиции. Библиотека MEF имеет гораздо бо́льшие возможности, чем описанные в данной главе. За дополнительной информацией по этому вопросу обратитесь к справочнику MEF Programming Guide по адресу `http://mef.codeplex.com`.

Система Visual Studio 2010 позволяет создавать расширения времени выполнения с применением возможностей библиотеки MEF. Он отслеживает расширения с известными экспортируемыми контрактами, которые затем использует при компоновке нового управляющего элемента WPF Editor, позволяя вам легко расширять его поведение. Кроме того, система Visual Studio экспортирует ряд служб с хорошо известными контрактами, которые можно импортировать для использования в своих расширениях. Дополнительную информацию о новом компоненте Visual Studio Editor и его расширении с применением библиотеки MEF можно найти в разделе документации MSDN, посвященном компоненту Visual Studio 2010 Editor; там же представлено множество конкретных примеров расширений.

ЧАСТЬ XII

Редакция Visual Studio Ultimate

- **ГЛАВА 54**. Редакция Visual Studio Ultimate для архитекторов
- **ГЛАВА 55.** Редакция Visual Studio Ultimate для разработчиков
- **ГЛАВА 56.** Редакция Visual Studio Ultimate для тестировщиков
- **ГЛАВА 57.** Система Team Foundation Server

54

Редакция Visual Studio Ultimate для архитекторов

В ЭТОЙ ГЛАВЕ...

- Создание моделей вашего решения
- Обеспечение архитектуры приложения
- Исследование существующей архитектуры

Редакция VSTS Architecture предлагала четыре диаграммы для проектирования распределенных систем, которые почти никогда не использовались в промышленных масштабах. Авторы системы Visual Studio 2010 отказались от этих четырех диаграмм, и все функциональные возможности для архитектурных задач были реализованы заново. Эти функциональные возможности можно разделить на две большие категории.

Первая новая возможность — шаблон проекта Modeling Projects, который позволяет создавать UML-диаграммы для построения модели приложения. Имеется также новая диаграмма, которую можно использовать для определения и обеспечения соотношений между элементами кода в проектах.

Другие новые возможности связаны с навигацией и осмыслением существующего кода. Имеются в виду способность генерировать диаграммы последовательностей из методов на языках C# и VB, а также графы зависимостей различных компонентов решения. Наконец, система Visual Studio 2010 содержит новый инструмент — Architecture Explorer, который применяется для быстрой навигации по вашему решению.

Проекты моделирования

Модель в терминологии программного обеспечения — это абстрактное представление некоторого процесса или объекта. Модели создаются для лучшего понимания способа работы различных частей приложения и передачи другим этого понимания. В системе Visual Studio 2010 все модели поддерживаются в виде единого проекта Modeling Project. Проекты моделей можно найти на своей странице в диалоговом окне Add New Project. Можно также создать проект путем добавления в решение диаграммы с помощью команды меню Architecture⇨New Diagram. Это приводит к выводу диалогового окна Add New Diagram, показанного на рис. 54.1. В нижней его части находится выпадающий список, позволяющий выбрать существующий проект или создать новый.

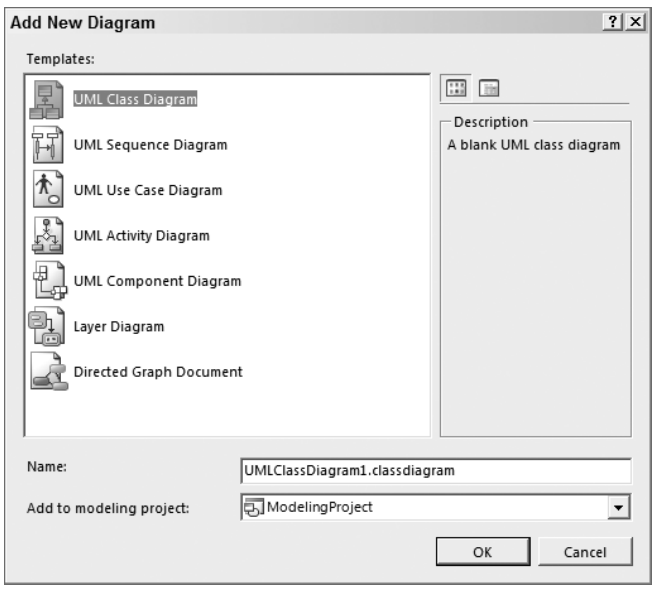


Рис. 54.1

Многие диаграммы в проекте модели могут быть легко подключены к набору рабочих элементов Work Items в системе Team Foundation Server, что делает его отличным инструментом для связи с другими членами команды.



Возможность создания проектов моделей и связанных с ними диаграмм в редакции Visual Studio 2010 Ultimate ограничена. Версия Premium включает возможность просмотра проектов и диаграмм, созданных кем-то другим.

UML-диаграммы

Унифицированный язык моделирования (Unified Modeling Language — UML) является промышленным стандартом для создания моделей в виде диаграмм. Система Visual Studio

2010 способна создавать большинство распространенных UML-диаграмм, включая такие диаграммы, как Activity Diagrams, Component Diagrams, Class Diagrams, Sequence Diagrams и Use Case Diagrams.



UML-диаграммы Visual Studio 2010 отвечают стандарту UML 2.0.

Диаграмма Use Case Diagram

Диаграмма Use Case Diagram (рис. 54.2) определяет пользователей системы и задачи, которые им необходимо решать с помощью системы. Как показано на рис. 54.2, каждая задача может состоять из подчиненных задач. Диаграмма Use Case Diagrams обычно представляет собой диаграмму очень высокого уровня.

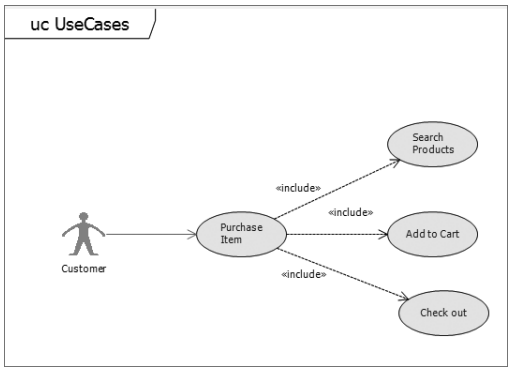


Рис. 54.2

Моделирование задач позволяет сфокусироваться на целях конечных пользователей и гарантировать, что их потребности будут удовлетворены разрабатываемым вами приложением. Кроме того, оно позволяет определить рамки вашего приложения в плане требований пользователей, что способствует пониманию области охвата создаваемого вами проекта. Прецеденты (Use Cases) в системе TFS обычно связаны с пользовательскими сценариями (User Story) и тестами (Test Case).

Диаграмма Activity Diagram

Диаграмма Activity Diagram (рис. 54.3) описывает действия и решения, которые участвуют в выполнении единой задачи. Диаграммы Activity и Use Case часто используются для того, чтобы представить разными способами одну и ту же информацию. Диаграмма Use Cases зачастую лучше показывает иерархическую природу задач, выполняемых пользователем, в то время как диаграммы Activity демонстрируют использование каждой подзадачи.

Диаграмма Activity начинается с небольшого черного кружка, за которым следуют стрелки, в конечном итоге достигающие кружка с точкой в центре. Каждый прямоугольник со скругленными углами представляет действие, а ромб — решение о том, какое действие следует предпринять следующим. Небольшая пиктограмма
в виде разветвления в нижнем правом углу прямоугольника Search Products идентифицирует данное действие как вызывающее другое действие.

Диаграмма может демонстрировать и параллельные потоки действий, как показано на рис. 54.4. Здесь также показано асинхронное отправление и получение событий.

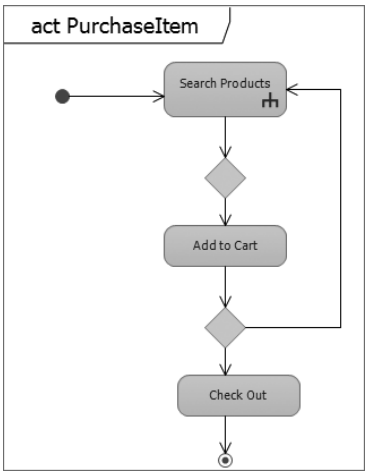


Рис. 54.3

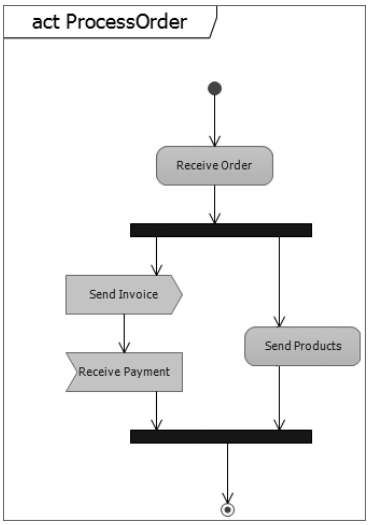


Рис. 54.4



Обычно такие диаграммы используются для представления алгоритмов.

Диаграмма Sequence Diagram

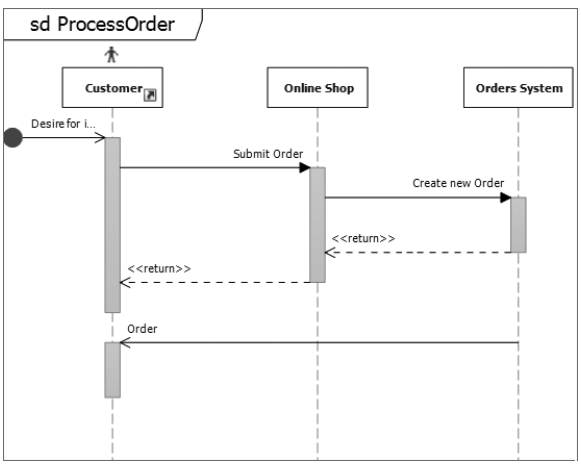


Рис. 54.5

Диаграмма Sequence Diagram (рис. 54.5) показывает сообщения, передаваемые между различными компонентами системы или между системами при активности большего масштаба. Диаграмма Sequence Diagram используется, когда необходимо показать поток действий от одного пользователя к другому в пределах системы.

В верхней части диаграммы находятся прямоугольники, представляющие вовлеченных в процесс пользователей. Пунктирные линии от каждого пользователя ведут к контекстам исполнения, указывающим, где именно в общем процессе каждый пользователь выполняет некоторую операцию. При чтении диаграммы ось времени направлена вниз. Линии в горизонтальном направлении представляют сообщения, передаваемые между пользователями системы. Сообщения могут быть синхронными (заполненные стрелки) или асинхронными (открытые стрелки). Они также могут быть найденными, что означает, что их источник неизвестен, или потерянными, когда неизвестно, куда они идут. Обычно такие сообщения появляются на границах моделируемой деятельности.

Диаграмма Interaction Use (рис. 54.6) представляет собой часть диаграммы Sequence Diagram, отделенной от нее и могущей использоваться повторно. Для создания диаграммы Interaction Use выберите эту диаграмму на панели инструментов и перетащите прямоугольник, охватывая то, что должно быть включено в диаграмму. После того как диаграмма Interaction Use будет создана, ее можно использовать для генерации другой диаграммы последовательности или для ее связи с существующей. Двойной щелчок на диаграмме Interaction Use открывает ее.

Иногда требуется сгруппировать несколько контекстов исполнения и сообщений. Примером может служить необходимость повторения взаимодействия в цикле. Для этого необходимо создать диаграмму Combined Fragment (рис. 54.7), выбирая включаемые в нее элементы и один из вариантов настройки Surround With.

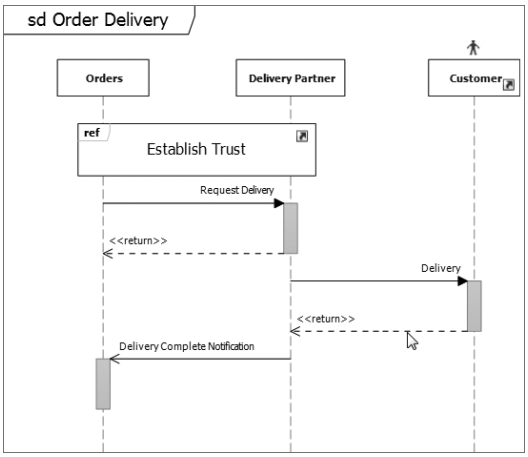


Рис. 54.6

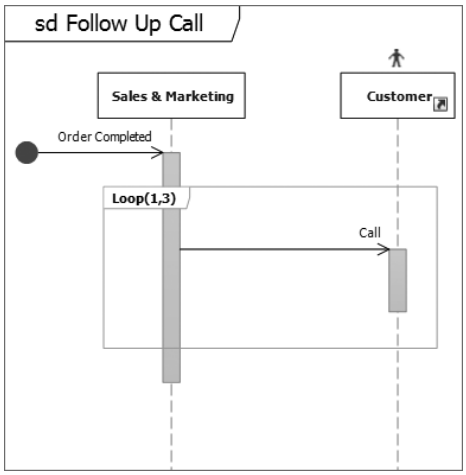


Рис. 54.7



Хотя на диаграммах UML Sequence и диаграммах .NET Sequence используются одни и те же обозначения, не путайте их. Диаграммы UML Sequence могут создаваться только в проектах Modeling Project и включать элементы из других частей модели. Диаграммы .NET Sequence генерируются из существующего .NET-кода и не являются частью модели.

Диаграмма Component Diagram

Компонент представляет собой единый модуль функциональности, который может быть замещен в своей среде. Каждый компонент скрывает свою внутреннюю структуру, но указывает интерфейсы, которые могут использоваться другими компонентами для доступа к функциональным возможностям данного компонента. Кроме того, каждый компонент может указать набор интерфейсов, необходимых для выполнения своих задач. Диаграмма Component Diagram (рис. 54.8) демонстрирует компоненты системы вместе с их открытыми и требуемыми интерфейсами. Она также показывает, как открытые интерфейсы соотносятся с требуемыми.

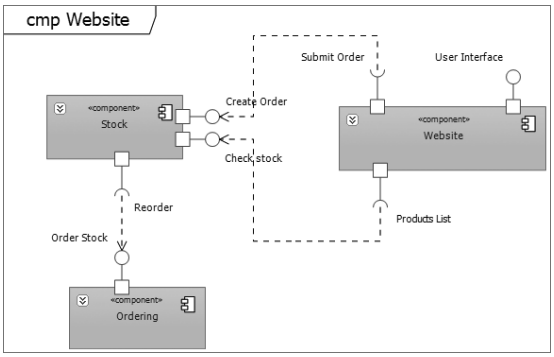


Рис. 54.8

Моделирование компонентов помогает рассматривать части приложения как дискретные единицы. Это, в свою очередь, снижает связанность в вашем проекте, делая приложение проще в обслуживании и развитии. Обычно взаимодействия системных компонентов (или частями в пределах компонента) моделируются с помощью диаграмм Sequence Diagram. Для моделирования интерфейсов компонента с передаваемыми между интерфейсами данными (параметрами) можно использовать диаграмму Class Diagram.

Можно также использовать Class Diagram для описания классов, из которых состоят части компонентов. Наконец, для моделирования внутренней работы компонентов используют диаграмму Activity Diagram.

Диаграмма Class Diagram

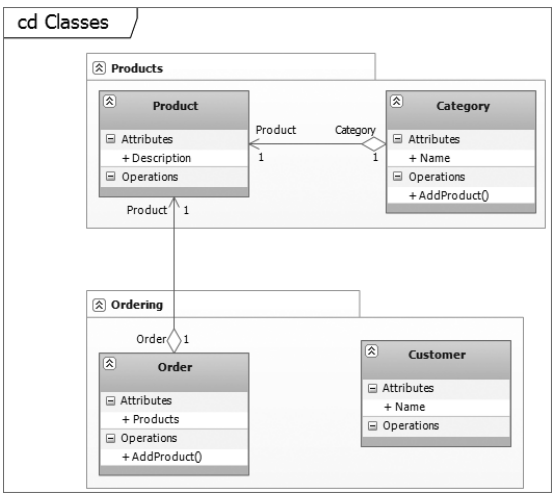


Рис. 54.9

Диаграмма Class Diagram (рис. 54.9) позволяет моделировать типы, применяемые в вашей системе, и взаимоотношения между ними. Пожалуй, это наиболее часто используемый тип UML-диаграмм. Вы можете определять классы, интерфейсы и перечисления. Каждый из этих элементов может быть связан с другим путем наследования, композиции, агрегирования или просто ассоциации. Каждый элемент может иметь атрибуты и операции, определенные над ним. Наконец, элементы могут группироваться в пакеты.



Несмотря на лежащие в их основе одинаковые обозначения, не следует путать диаграммы UML Class Diagrams и .NET Class Diagrams. UML Class Diagram используется для определения и организации элементов модели. .NET Class Diagram играет подобную роль для кода .NET. Изменение .NET Class Diagram приводит к изменению соответствующего .NET-кода.

UML Model Explorer

Каждая из диаграмм UML представляет разные точки зрения на одну и ту же лежащую в ее основе модель. Чтобы рассмотреть всю модель в целом, можно воспользо-

ваться окном UML Model Explorer, показанным на рис. 54.10. При добавлении содержимого в модель с использованием различных диаграмм каждый элемент будет выводиться в окне UML Model Explorer.

Добавлять элементы непосредственно в модель можно с помощью контекстных меню узлов в окне UML Model Explorer. Можно также перетаскивать элементы из окна Model Explorer непосредственно на поверхность многих диаграмм. Так создается связь между исходным элементом и его представлением в диаграмме. Любой элемент можно удалить как из диаграммы, так и из диаграммы, и из модели.

Применение Layer Diagram для проверки архитектуры приложения

Диаграмма Layer Diagram (рис. 54.11) представляет собой инструмент, который должен помочь вам в указании высокоуровневой структуры программного решения. Она составлена из различных областей или слоев приложения и определяет взаимоотношения между ними.

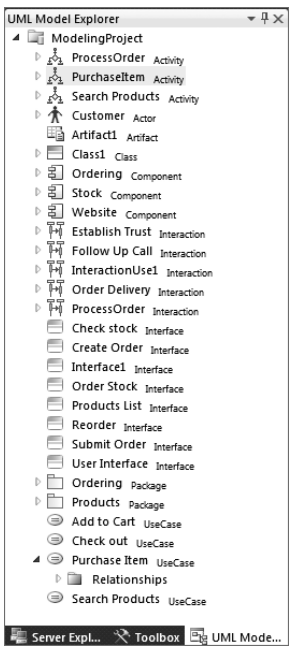


Рис. 54.10

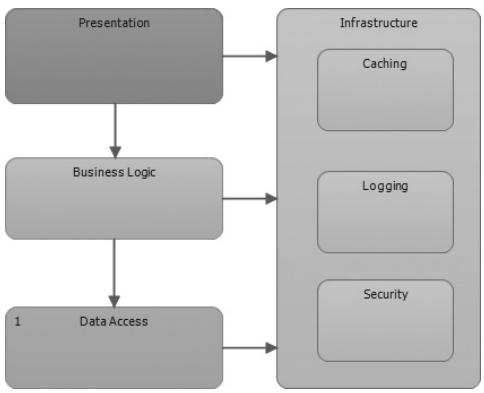


Рис. 54.11

Каждый слой представляет собой логическую группу классов с общей технической ответственностью, такую как группа классов, отвечающая за доступ к данным или за представление на экране.

Создав диаграмму, можете перетаскивать каждый уровень на поверхность и давать ему свое имя. Между слоями можно указывать однонаправленные или двунаправленные связи. Слой зависит от другого слоя тогда, когда отдельные его компоненты содержат прямые ссылки на некоторые компоненты слоя, от которого зависит данный слой. При отсутствии явной зависимости предполагается, что компонентов, соответствующих данному описанию, не существует.



Как видно из рис. 54.11, слои могут быть вложены один в другой.

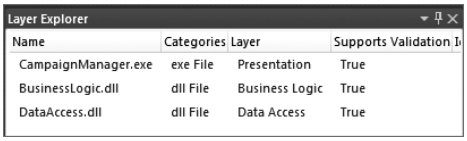


Рис. 54.12

Созданную диаграмму можно использовать для выявления связей между слоями вашего компилируемого приложения и для проверки соответствия связей первоначальному проекту. Прежде чем приступить к описанным действиям, следует с каждым слоем связать проекты путем перетаскивания их из окна Solution Explorer в свой слой. Когда вы сделаете это, в окно Layer Explorer (рис. 54.12) будут добавлены новые записи, а числа внутри каждого слоя будут обновлены, отражая количество связанных с ним элементов.



Новые слои создаются путем перетаскивания проектов из окна Solution Explorer на поверхность диаграммы Layer Diagram.

После того как диаграмма слоев будет связана со сборками, можете заполнить ее недостающими зависимостями путем выбора команды Generate Dependencies контекстного меню, вызываемого щелчком правой кнопкой мыши на поверхности диаграммы. При этом будет выполнен анализ связанных сборок, а также при необходимости — сборка проекта и заполнение всех недостающих зависимостей. Заметим, что этот инструмент не удаляет неиспользуемые зависимости.

Если диаграмма содержит все слои и только те зависимости, которые, как вы считаете, должны быть, можете проверить свое приложение на соответствие проекту, определенному диаграммой. Для этого воспользуйтесь командой Validate Architecture контекстного меню, вызываемого щелчком правой кнопкой мыши на поверхности диаграммы. Этот инструмент анализирует структуру вашего решения, и все найденные нарушения будут показаны как ошибки построения, как показано на рис. 54.13. Двойной щелчок на такой ошибке приводит к открытию диалогового окна Directed Graph, указывающего взаимоотношения между различными проектами.

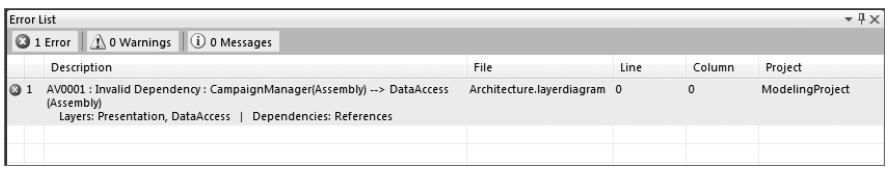


Рис. 54.13



Не все артефакты, которые могут быть связаны с диаграммой, поддерживают проверку. В окне Layer Explorer имеется столбец Supports Validation, который может помочь вам определить, нет ли у вас таких артефактов.

Проекты моделирования имеют свойство `ValidateArchitecture` логического типа, которое используется для определения того, должны ли все диаграммы Layer Diagram быть проверены по окончании построения проекта. Можно также запросить проверку архитектуры от системы Team Foundation Build, если добавить свойство `ValidateArchitectureOnBuild` в свой файл `TfsBuild.proj` или шаблон Process Template и установить его значение равным `true`.

Связь с системой Team Foundation Server

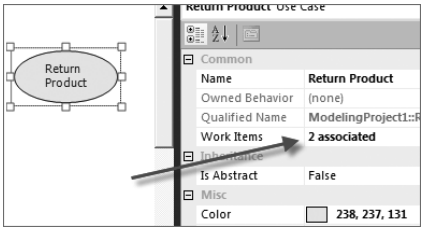


Рис. 54.14

Каждый из элементов диаграммы в проекте модели, как и сами диаграммы, может быть связан с элементами Work Items в системе Team Foundation Server. Это можно сделать из контекстного меню, вызываемого щелчком правой кнопкой мыши на элементе, который вы хотите связать, выбрав в нем команду Create Work Item либо Link to Work Item. Когда элемент модели будет связан с Work Items, он появится в окне свойств этого элемента (рис. 54.14). Можно также получить список элементов Work Items, с которыми связан данный элемент, выбрав команду View Work Items контекстного меню элемента.

Изучение кода

Многие возможности системы Visual Studio разработаны для того, чтобы помочь вам понять структуру существующего кода и перемещаться по нему. Ориентированные графы обеспечивают вас высокоуровневым представлением взаимосвязей между различными типами компонентов в вашем проекте. Инструмент Architecture Explorer позволяет глубоко погрузиться в код, помогая при этом не заблудиться в нем и всегда отдавать себе отчет в том, где именно вы находитесь. Возможность генерации диаграмм Sequence Diagram позволяет быстро разобраться в поведении конкретного метода, а также в его связях с другими методами и классами.

Инструмент Architecture Explorer

Одним из наиболее сложных аспектов при работе с новым кодом является понимание того, где именно вы находитесь относительно других частей кода. В окне Architecture Explorer (рис. 54.15) можно очень быстро перемещаться по структуре кода с помощью одного щелчка мышью, оставляя при этом “следы”, которые позволяют быстро понять, как именно вы сюда попали. Некоторые элементы можно перетащить из окна Architecture Explorer непосредственно на поверхность многих диаграмм из числа описанных в данной главе.

На рис. 54.15 показано продвижение по столбцам, в каждом из которых содержится ряд узлов. Всякий раз, когда вы щелкаете на узле в столбце, открывается новый столбец, основанный на выбранном узле. Между столбцами имеется свернутый столбец Action, который разворачивается после щелчка на нем (рис. 54.16). Выбор другого действия позволяет вам изменить следующий шаг на пути навигации. В некоторых случаях действия выполняют некоторые задачи вне окна Architecture Explorer. Двойной щелчок на узле зачастую приводит к открытию в окне редактора.



Вы можете выбрать несколько узлов в столбце, и тогда в следующем столбце будет находиться объединение результатов для выбранных столбцов.

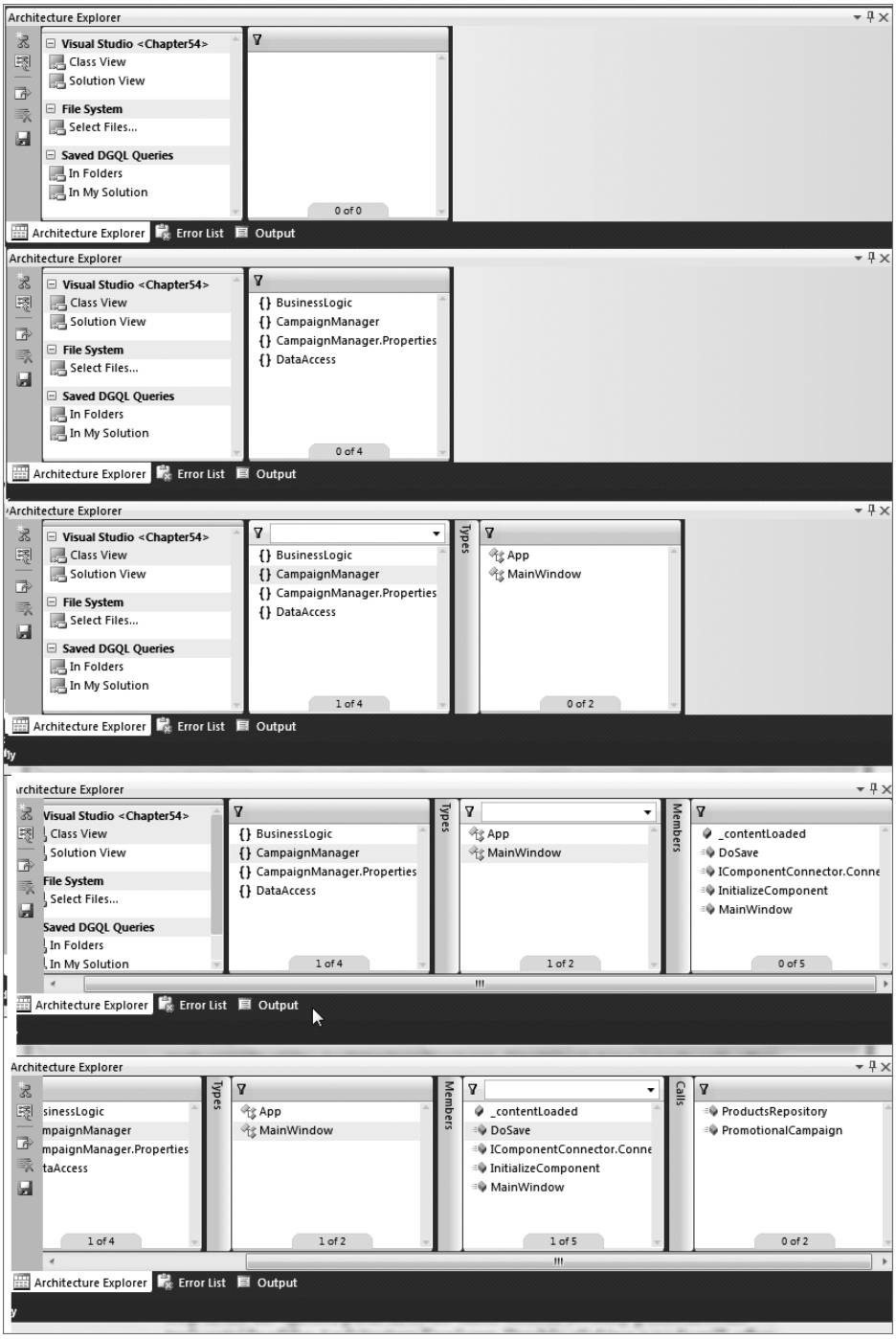

Architecture Explorer
Visual Studio <Chapter54>
Class View
Solution View
File System
Select Files...
Saved DGQL Queries
In Folders
In My Solution
0 of 0
Architecture Explorer
Error List
Output
{} BusinessLogic
{} CampaignManager
{} CampaignManager.Properties
{} DataAccess
0 of 4
Types
App
MainWindow
1 of 4
0 of 2
Members
_contentLoaded
DoSave
IComponentConnector.Conne
InitializeComponent
MainWindow
1 of 2
0 of 5
Calls
ProductsRepository
PromotionalCampaign
1 of 5
0 of 2

Рис. 54.15

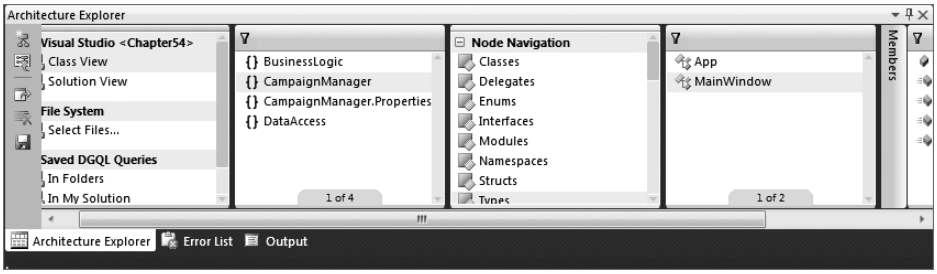


Рис. 54.16

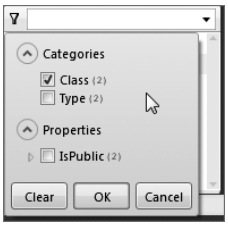


Рис. 54.17.

Каждый столбец в окне Architecture Explorer может быть независимо отфильтрован двумя способами. Простейший путь заключается во вводе в текстовом поле в верхней части столбца некоторого текста и нажатии клавиши <Enter>. Так на основании введенной вами информации фильтруется содержимое столбца. Другой способ состоит в использовании диалогового окна Category and Property Filter, показанного на рис. 54.17, которое вызывается после щелчка на пиктограмме фильтра в верхней левой части столбца. При фильтрации столбца его фоном становится изображение большой пиктограммы фильтрования.

Первый столбец — Architecture, который представляет собой специальный столбец, содержащий представления, каждое из которых принадлежит домену. Домен Solution предлагает представление Class View, основанное на логическом представлении ваших классов, и представление Solution View, основанное на физическом размещении ваших файлов. Домен File System позволяет загрузить скомпилированные сборки с диска и проанализировать их в проводнике Architecture Explorer.

Домен Saved DGQL Queries предлагает доступ к ранее сохраненным запросам. Эти запросы могут быть сохранены как часть вашего решения или располагаться в вашей папке `Documents\Visual Studio 10\ArchitectureExplorer\Queries`. Для сохранения запроса воспользуйтесь кнопкой Save Query to Favorites в левой части Architecture Explorer.

Графы зависимостей

Когда вы получаете в свое распоряжение существующий код, одной из наиболее сложных задач является выявление зависимостей между различными сборками, пространствами имен и классами. Граф зависимости — Dependency Graph — позволяет визуализировать зависимости между элементами на разных уровнях рассмотрения. Простейший способ создания графа зависимостей заключается в использовании команды меню Architecture⇨Generate Dependency Graph. Он позволяет создать граф зависимостей для сборок, пространств имен, классов или в соответствии с некоторым определяемым вами пользовательским критерием. Имеются четыре основных варианта размещения графа в зависимости от направления его стрелок: сверху вниз, слева направо, снизу вверх и справа налево. На рис. 54.18 показан пример, в котором стрелки направлены слева направо. Имеется также представление Quick Clusters, которое пытается расположить элементы так, чтобы они находились как можно ближе к другим связанным с ними элементам.



Рис. 54.18

В дополнение к этим представлениям имеется представление Dependency Matrix (рис. 54.19), которое использует цветные квадраты для того, чтобы показать, что элемент в строке зависит от элемента в столбце.

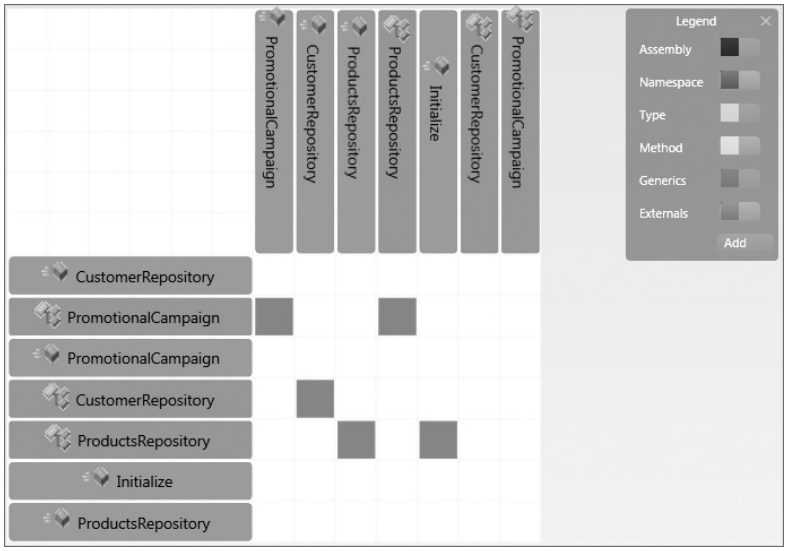


Рис. 54.19

Система Visual Studio 2010 включает три анализатора графов зависимостей, которые могут помочь найти потенциальные проблемы в вашем коде. Они включаются и отключаются в контекстном меню Analyzers и используют окрашивание элементов, отвечающих критерию, определенному анализатором.

- **Циклические ссылки.** Их достаточно сложно обнаружить при росте количества компонентов. Анализатор обнаруживает строго связанные элементы и выделяет их цветом.
- **Показ концентраторов.** Концентратор (hub) — это компонент, от которого зависят многие другие компоненты. Его сложно изменять, так как внесенные в него изменения затрагивают практически всю систему.
- **Узлы без ссылок.** Это узлы, от которых не зависят другие узлы. Это может указывать на наличие проблем, например на то, что элемент, который представлен данным узлом, не используется в приложении, или на то, что зависящие от него узлы на диаграмме отсутствуют.

Генерация диаграммы последовательности

Инструмент .NET Sequence Diagram позволяет моделировать реализацию метода на языке C# или VB. Наилучший способ создания диаграммы последовательности .NET — щелкнуть правой кнопкой мыши на сигнатуре метода в окне редактора и выбрать команду Generate Sequence Diagram для вывода одноименного диалогового окна, показанного на рис. 54.20.

После того как вы щелкнете на кнопке OK, Visual Studio проанализирует ваш проект и сгенерирует диаграмму .NET Sequence Diagram, как показано на рис. 54.21. Вы можете вносить в нее изменения, не влияя на лежащий в ее основе код, так что ее можно применять для быстрого испытания разных идей относительно работы кода.

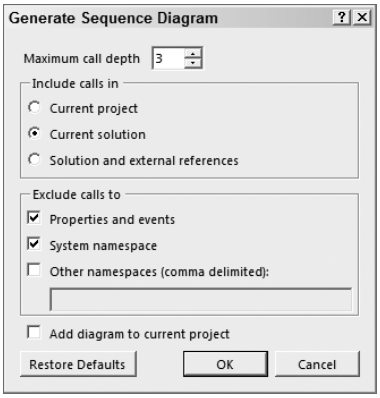


Рис. 54.20

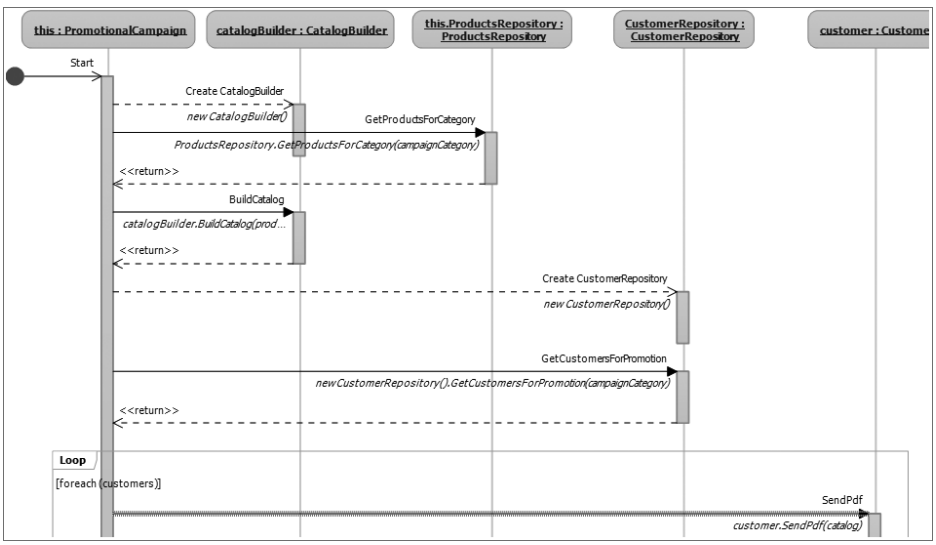


Рис. 54.21



С помощью диалогового окна Add New Item можно добавить диаграмму .NET Sequence Diagram в любой проект .NET. При добавлении таким образом диаграмма не будет связана с конкретным методом .NET.

Резюме

Функциональная возможность Modeling Projects предоставляет эффективные способы работы с дизайном проекта. Вы можете использовать диаграммы Use Case Diagram, Activity Diagram и Sequence Diagram для моделирования пользовательских требований с

разных точек зрения. Диаграммы Component Diagram, Class Diagram и Layer Diagram предназначены для моделирования структуры приложения. Возможность проверки с помощью диаграммы Layer Diagram того, насколько ваше приложение соответствует разработанной архитектуре, является отличным инструментом поддержки высокого качества проекта и гарантией того, что принятые архитектурные решения не окажутся заброшенными в процессе написания кода.

Быстрое изучение существующего кода может оказаться нелегкой задачей. При ее решении вам могут существенно помочь графы, указывающие взаимоотношения между разными частями приложения. Новый инструмент Architecture Explorer позволяет быстро перемещаться по соединениям между компонентами системы в поисках нужных элементов. Наконец, возможность генерации диаграммы последовательности на основе существующего метода помогает быстро понять, как метод взаимодействует с другими методами и классами приложения.

55

Редакция Visual Studio Ultimate для разработчиков

В ЭТОЙ ГЛАВЕ...

- Анализ потенциальных проблем кода
- Профилирование приложений в поисках узких мест
- Разработка проектов баз данных

Редакции Visual Studio 2010 Premium и Ultimate предоставляют разработчикам множество передовых возможностей, в первую очередь направленных на повышение качества и упрощение разработки баз данных. Инструментарий для повышения качества включает метрики кода, статический анализ и профилирование. Конечно же, нельзя утверждать, что невозможно создать качественный код с помощью других редакций Visual Studio 2010 или что указанные инструменты гарантируют высокое качество кода. Очевидно, что для этого требуется нечто большее, чем применение нескольких инструментов, но они могут оказать огромную помощь и существенно сократить время решения таких задач, как обзор и отладка кода. Наиболее интересный новый инструмент отладки — IntelliTrace. Он позволяет собирать информацию о среде, связанную с непройденными контрольными примерами.

В главе 26 уже рассматривались некоторые инструменты для работы с базами данных, доступные в системе Visual Studio 2010. В этой главе рассматриваются две большие области: разработка по технологии SQL-CLR и инструменты, помогающие командам работать с базами данных. Первая область связана с разработкой и развертыванием .NET-кода, располагающегося в системе SQL Server. Вторая помогает в работе с разными версиями схем, изолируя разработчиков и позволяя им сравнивать изменения, автоматически генерировать данные и совместно использовать вносимые ими модификации с другими разработчиками или администраторами баз данных.

Метрики кода

Метрики кода служат в качестве справочного материала о том, насколько поддерживаемым является разработанный вами код. Система Visual Studio 2010 предоставляет пять метрик для вашего исходного кода, которые можно просмотреть в окне Code Metrics, показанном на рис. 55.1. Чтобы открыть это окно, воспользуйтесь командами меню Analyze⇨Calculate Code Metrics или Analyze⇨Windows⇨Code Metric Results. После того как окно будет выведено, можете использовать кнопку панели инструментов вверху слева для пересчета метрик.

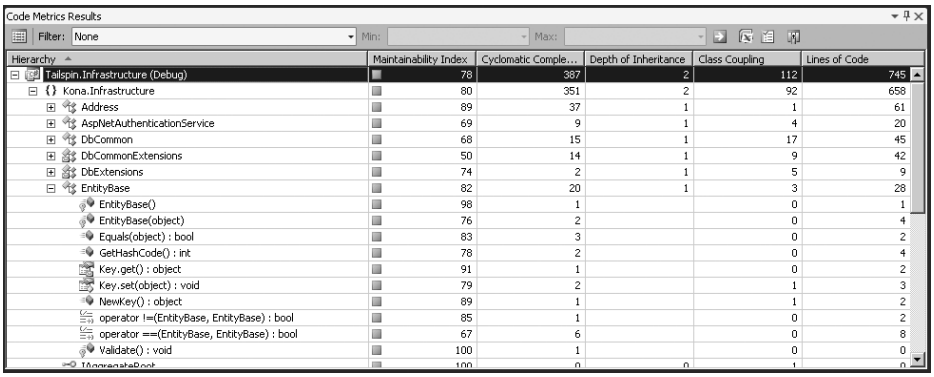


Рис. 55.1

Непосредственно в списке можно фильтровать любые метрики (чтобы увидеть только те методы, которые попадают в определенный диапазон), экспортировать данные в таблицы Excel, настраивать столбцы для удаления метрик или создавать рабочий элемент системы Team Foundation Server. Экспорт в таблицу Excel в особенности полезен для генерации отчетов с помощью сводных таблиц или для применения фильтров и сортировки. Например, если вы хотите просмотреть методы более чем с 15 строками кода, учтите, что их фильтрование непосредственно в окне Code Metrics даст вам большое количество пространств имен и типов, и вам придется затратить время на их просмотр, чтобы найти среди них методы; в то же время в таблице Excel можно легко отфильтровать пространства имен и типы, оставив только методы.

При использовании метрик для принятия решений о своем коде учитывайте, что абсолютные значения не так важны, как относительные. Правило, которое гласит “все методы не должны содержать больше 25 строк кода”, не столь полезно, как правило с относительным утверждением “предпочитайте короткие методы”. Вы должны также рассматривать как важный фактор изменения значений, так что, если средний индекс поддержки падает, это может быть признаком того, что вам пора задуматься об изменении кода, чтобы сделать его более легко поддерживаемым.



Для всех метрик, кроме индекса поддержки (Maintainability Index), более низкие значения предпочтительнее.

Метрика Lines of Code

Название Lines of Code (строки кода) говорит само за себя. Тем не менее стоит упомянуть, что назначение этой метрики — только проинформировать о сложности кода, и она *не* должна использоваться в качестве меры выполненной работы. Очевидно, что метод из пяти строк, вызывающий другие методы, проще метода из 25 строк, в который внесен код вызываемых методов:

```
public class OrdersGenerator
{
    public void GenerateOrder(Order order)
    {
        IsUnderCreditLimit(order);
        IsCustomerBlocked(order.Customer);
        AreProductsInStock(order);
        IsCustomerBlocked(order);
        SaveOrder(order);
    }
    // Остальные методы пропущены.
}
```

Если сравнить класс с шестью методами, показанный в приведенном выше фрагменте кода, с классом с той же функциональностью, но в котором весь код собран в одном методе, то последний будет содержать все 25 строк кода. В предположении, что каждый метод состоит из пяти строк, вариант класса с шестью методами состоит из 30 строк, но при этом он проще для понимания. С этой метрикой следует быть осторожными: более длинный класс может оказаться лучше более короткого.



Для снижения этого показателя можно воспользоваться рефакторингом Extract Method, рассматривавшимся в главе 8. Подумайте, не лучше ли перенести извлеченный метод в новый класс.

Метрика Depth of Inheritance

Эта метрика подсчитывает количество базовых классов; обычно рекомендуется, чтобы ее значение было менее шести. Однако, как и в случае других метрик, подходить к значению данной метрики нужно с осторожностью, поскольку оно зависит, в частности, от того, какие классы наследуются. Если вы наследуете класс `LinkLabel`, то получите глубину наследования, равную 4, но ваши базовые классы будут изменяться, скорее всего, меньше, чем при наследовании класса `ProviderXComponent` и глубине наследования 1. Более вероятно, что класс `ProviderX` будет изменять свой компонент и влиять на ваш класс, в то время как компания Microsoft вряд ли пойдет на такой шаг. Однако сами вы, вероятно, не будете обновлять библиотеку, содержащую класс `ProviderX`. Главный вывод из сказанного в том, что данная метрика должна рассматриваться с учетом конкретных обстоятельств и конкретных базовых классов.

Метрика Class Coupling

Эта метрика подсчитывает зависимости элемента от других типов за исключением примитивных и встроенных типов наподобие `Int32`, `Object` или `String`. Чем больше у вас зависимостей, тем сложнее поддерживать данный элемент, поскольку тем более вероятны изменения в других типах, которые могут отрицательно сказаться на

вашем коде. Аналогично глубине наследования важность данной метрики зависит от того, какие именно зависимости у вас есть. Класс, обращающийся к системным библиотекам, существенно менее надежен, чем классы, обращающиеся к другим классам, находящимся в активной разработке. Вы можете просмотреть данную метрику на каждом уровне иерархии (проект, пространство имен, тип и член).

Cyclomatic Complexity

Цикломатическая сложность — это мера того, сколько путей выполнения имеется в вашем коде. Метод с высокой цикломатической сложностью труднее для понимания и сопровождения, чем метод с низким значением. Для этой метрики трудно указать рекомендуемое значение, поскольку оно зависит от уровня команды разработчиков и команды, которая будет сопровождать продукт. Гораздо важнее информация о тенденции. Так, постепенное повышение цикломатической сложности говорит о том, что ваш код становится сложнее для понимания. Тем не менее иногда сложное решение вполне оправдано.

Maintainability Index

Эта метрика вычисляется по формуле, учитывающей цикломатическую сложность, количество строк кода, а также количество операторов и операндов — как общее, так и индивидуальное. Вы получаете значение от 0 до 100, причем чем оно больше, тем легче поддерживать данный код.

Код, исключенный из анализа

Код, маркированный атрибутами `CompilerGenerated` и `GeneratedCode`, не рассматривается при подсчете метрик. Примерами кода с атрибутом `GeneratedCode` являются классы `Datasets` и `Web Service Proxies`, но другой генерируемый код (наподобие Windows Forms) не маркируется и учитывается при подсчете метрик.

Инструмент Managed Code Analysis Tool

Этот инструмент основан на FxCop, инструменте компании Microsoft для внутреннего использования, выпущенном для открытого использования несколько лет назад и, наконец, вошедшем в интегрированную среду разработки. Он позволяет выполнять статический анализ кода с применением набора правил, определяющих качественные решения, которые вы намерены применить к своему коду. Вы можете настраивать применяемый к каждому проекту набор правил на странице свойств проекта, показанной на рис. 55.2.

Для его использования щелкните правой кнопкой мыши на проекте и выберите в контекстном меню команду Run Code Analysis или, если вы выбрали пункт Enable Code Analysis on Build в окне свойств проекта, просто скомпилируйте последний. Будут выполнены необходимые вычисления в соответствии с правилами, и, если обнаружатся их нарушения (поверьте мне, рано или поздно это произойдет), вы получите набор предупреждений в окне Error List.

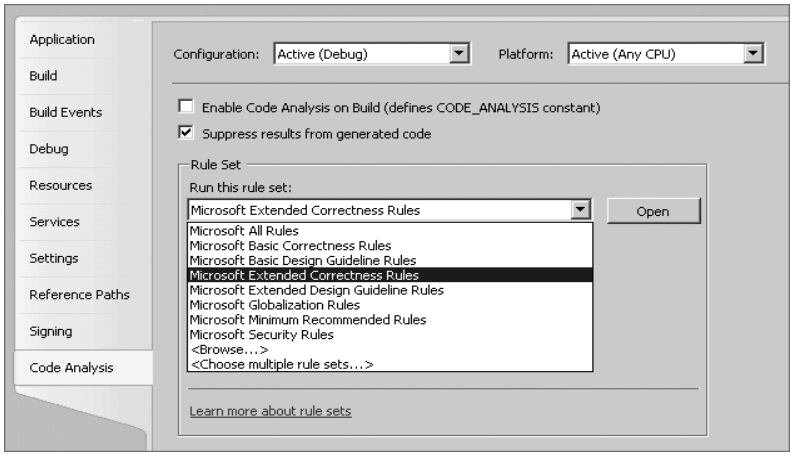


Рис. 55.2



По умолчанию о каждом нарушении правил сообщается с помощью предупреждения, но вы можете изменить это поведение.

Если вы щелкнете правой кнопкой мыши на предупреждении и выберете из контекстного меню команду Show Error Help, то получите описание правила, того, что произошло и как исправить ситуацию. Подавить вывод предупреждающих сообщений можно с помощью атрибута `System.Diagnostics.CodeAnalysis.SuppressMessage`, который может быть применен к проблемному члену или ко всей сборке в целом. Вы можете быстро и просто генерировать эти атрибуты, выбрав одну из команд подменю Suppress Message контекстного меню, вызываемого щелчком правой кнопкой мыши в окне Errors.

Более 200 правил собраны в 11 категорий, и при необходимости вы можете добавлять собственные пользовательские правила. В зависимости от проекта вы можете захотеть исключить некоторые категории или некоторые конкретные правила. Например, если вам не нужна глобализация и вы не планируете поддерживать ее в будущем, исключите данную категорию. Вы можете даже создать собственный набор правил (Add New Item⇨Code Analysis Rule Set), если предоставляемые компанией Microsoft правила не соответствуют вашим нуждам.

При первом запуске инструмента Code Analysis нужно включить все правила и при необходимости либо исключить их путем внесения изменений в код, либо подавить с помощью атрибута все предупреждения. Это отличный способ получить практический опыт. После нескольких итераций переписанный код будет существенно меньше нарушать правила. Если вы начинаете новый проект, можете добавить политику проверок, которая защитит ваш код с предупреждениями анализатора от последующих проверок.



Никогда не отключайте предупреждения без важных причин. Найти эти нарушения снова будет очень трудно.

Инструмент C/C++ code analysis tool

Этот инструмент похож на инструмент Managed Code Analysis Tool, но работает с неуправляемым кодом. Для его активизации просто перейдите в окно свойств своего проекта на языке C++, найдите узел Code Analysis в разделе Configuration Properties и выберите Yes для пункта Enable Code Analysis for C/C++ on Build. Всякий раз при компиляции проекта этот процесс будет перехватываться инструментом анализа, который будет пытаться анализировать каждый путь выполнения.

Это поможет вам обнаруживать потенциальные аварийные остановы, которые очень сложно найти с помощью других методов типа отладки, не говоря об огромном затрачиваемом при этом времени. Инструмент способен обнаружить утечки памяти, неинициализированные переменные, проблемы при работе с указателями, а также переполнения и исчерпания буферов.

Инструменты для профилирования

Инструменты профилирования позволяют обнаружить и исправить проблемы проекта, связанные с производительностью и памятью. Вы можете начать сеанс профилирования с выбора команды Launch Performance Wizard в меню Analyze. На первом шаге мастер предложит выбрать один из четырех методов профилирования.

- **Метод CPU Sampling** сообщает об использовании центрального процессора через регулярные промежутки времени работы вашего приложения. Этот тип профилирования хорошо подходит для начального анализа или для обнаружения проблем, связанных конкретно с использованием центрального процессора.
- **Метод Instrumentation** вносит дополнительные строки кода в вашу сборку для сообщения о продолжительности вызова каждого метода. Этот вид профилирования можно использовать для поиска того, что вызывает наибольшие затраты времени в вашем приложении.
- **Метод .NET Memory profiler** накапливает данные об объектах при их создании и при их уничтожении сборщиком мусора.
- **Метод Concurrency profiling** собирает информацию о многопоточных приложениях и предоставляет ряд визуализаций, которые помогут вам при изучении проблем, связанных с распараллеливанием.

Затем вы выбираете проект, выполняемый файл, DLL или веб-сайт, который намерены профилировать. После этого Performance Wizard создает сеанс профилирования и открывает окно Performance Explorer. Вы можете также создать пустой сеанс из Performance Explorer или из теста в окне Test Results.

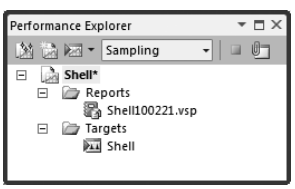


Рис. 55.3

В окне Performance Explorer (рис. 55.3) можно выбирать между первыми двумя указанными выше методами исследования производительности. Используя комбинированный управляющий элемент, вы можете запустить мастер или создать сеанс вручную. Хотя вы можете исследовать DLL, при этом вам понадобится точка входа вашего приложения, чтобы начать сеанс, так что убедитесь, что в качестве целевого приложения у вас включен выполняемый файл, веб-сайт или текстовый проект.



При охвате большого количества кода профилирование может дать вам хорошую наводку на методы с большим временем работы или с использованием большого количества памяти. Однако отнеситесь к указанной информации критически — влияние редко вызываемого долгого метода на общую производительность приложения может оказаться ничтожным по сравнению с влиянием очень быстрого, но очень часто вызываемого метода.

Настройка сеансов профилирования

Для настройки вашего сеанса щелкните правой кнопкой мыши и выберите из контекстного меню команду Properties. В разделе General можно выбирать между Sampling, Instrumentation и Concurrency (рис. 55.4), а также при желании активизировать профилирование использования памяти .NET и указывать, куда должны записываться отчеты.

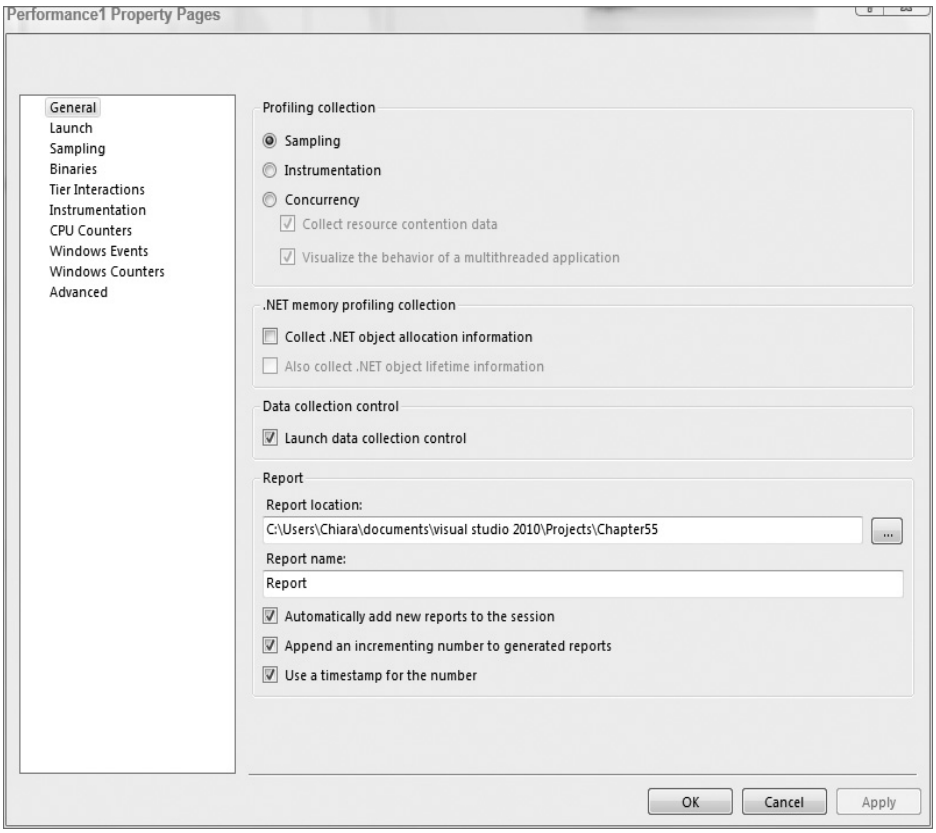


Рис. 55.4

В разделе Sampling можно указать, когда должна выполняться выборка данных; по умолчанию установлено значение 10000000 тактов процессора. В зависимости от того, что вы хотите отслеживать, можете изменить отбор проб на ошибки отсутствия страниц, системные вызовы или на иной специфичный счетчик производительности.

Включение профилирования взаимодействия уровней (Tier Interaction Profiling — TIP) позволяет собирать информацию о синхронных вызовах системы ADO.NET между вашим приложением и системой SQL Server. Сюда включаются количество выполнений каждого запроса и информация о том, сколько времени отнимает каждый из них. Если вы профилируете приложение ASP.NET WebForms, инструмент TIP также в состоянии предоставить данные о запросах страниц (количестве и времени генерации).

Раздел Instrumentation используется для указания событий перед измерениями и после них, например подпись сборки строгим именем. Эти настройки устанавливаются на уровне цели профилирования. Последний раздел страницы свойств, Advanced, также используется при выборе метода `Instrumentation`; здесь можно указать дополнительные аргументы команды. Чтобы просмотреть весь доступный список, выполните запрос "VSInstr" в документации MSDN.

Остальные разделы используются для указания коллекций различных счетчиков или событий. Раздел CPU Counters позволяет собирать дополнительную низкоуровневую информацию, выводимую в виде дополнительных столбцов разных отчетов (этот раздел доступен только для метода `Instrumentation`). Раздел Windows Counters предоставляет счетчики для системной производительности, а полученные ими результаты доступны в отчете Marks.

Раздел Windows Events позволяет указать провайдеров отслеживания событий. Чтобы получить информацию по событиям Windows, нужно самостоятельно получить текстовый отчет с помощью команды

```
Vsperfreport c:\<path>ReportName.vsp /calltrace /output:c:\<path>
```

Отчеты

После всех настроек можете запустить приложение, тест или веб-сайт из Performance Explorer. Все будет работать как обычно, но при этом будет выполняться сбор данных. По завершении работы приложения будет сгенерирован отчет. В табл. 55.1 приведено описание некоторых представлений отчета, а на рис. 55.5 показан его окончательный вид.

Таблица 55.1. Некоторые представления отчета

Представление	Описание
Summary	Показывает информацию о функциях. В случае режима Sampling показывает, какие функции выполняют наибольшую работу. В режиме Instrumentation показывает наиболее часто вызываемые функции с наибольшей выполняемой работой и функции с наибольшим временем работы. Отсюда вы можете перейти к представлению Functions. Если выбран пункт Collect .NET Object Allocation Information (рис. 55.4), показывает, какие функции запрашивают больше всего памяти и типы с наибольшим потреблением памяти и наибольшим количеством экземпляров
Call Tree	Содержит иерархию вызовов функций. Call Tree имеет функциональную возможность Hot Spot, которая указывает дочерние функции с наибольшим временем работы
Modules	Показывает информацию о модулях, для которых собиралась информация
Caller/Callee	Показывает, какие функции вызываются данной функцией и какие функции вызывают данную функцию
Functions	Представляет список всех функций, для которых собиралась информация. Двойной щелчок на функции позволяет перейти в окно Caller/Callee
Allocations	Показывает количество экземпляров и байтов, выделенных для каждого конкретного типа

Дополнительные отчеты могут быть сгенерированы с помощью инструмента командной строки `VSPerfReport`. Дополнительную информацию можно найти в документации MSDN.

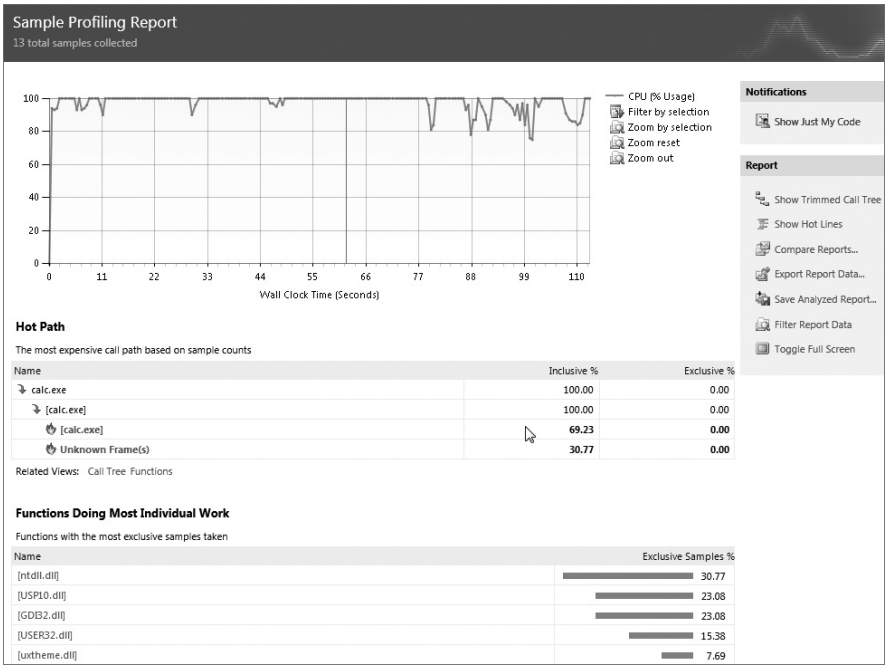


Рис. 55.5

На странице свойств проекта после выбора пунктов Collect .NET Object Allocation Information и Also Collect .NET Object Lifetime Information становятся доступными только Allocation и Object Lifetime соответственно. Некоторые из представлений отчета могут изменяться в зависимости от конкретных настроек. Чтобы увидеть описание отдельного столбца, просто наведите на его заголовок указатель мыши. Детальную информацию об отчетах можно найти в документации MSDN.

Во всех представлениях можно воспользоваться фильтром на панели инструментов, чтобы получить более конкретную информацию. Вы можете добавлять или удалять столбцы, щелкая правой кнопкой мыши на заголовке, и выполнять сортировку по определенному столбцу. Отчеты могут экспортироваться в форматах XML или CSV, а последовательные отчеты сравниваться друг с другом, позволяя отслеживать влияние вносимых изменений на производительность приложения.

Инструмент Standalone Profiler

Это инструмент, который запускается из командной строки и полезен для автономного профилирования приложения при отсутствии системы Visual Studio на машине, например в промышленных условиях. Для инсталляции этого инструмента нужно выполнить команду `vs_profiler.exe` из среды установки Visual Studio в папке `Standalone Profiler`. Инструмент будет установлен в каталог `%ProgramFiles%\Microsoft Visual Studio 10.0\Team Tools\Performance Tools`. Для частого его использования этот путь можно добавить в системную переменную `PATH`.

Приведенные далее команды выполняют профилирование приложения с использованием методики Sampling с настройками по умолчанию. Первая строка включает трас-

сировку. Следующая команда переключает профайлер для использования метода `CPU Sampling` и вывода отчета. В этом случае отчет будет сохранен в каталоге профилируемого приложения в файле с именем `Report.vsp`. Затем вы запускаете приложение, работаете с ним как обычно, а по окончании работы выключаете профайлер. После этого можете открыть файл отчета и проанализировать его в системе Visual Studio.

```
C:\ApplicationToProfile > vsperfclrenv /traceon
Enabling VSPerf Trace Profiling of managed applications(excluding allocation
profiling).
...
C:\ApplicationToProfile > vsperfcmd -start:sample -output:Report
Microsoft (R) VSPerf Command Version 10.0.30128 x86
...
C:\ApplicationToProfile > vsperfcmd -launch:Application.exe
Microsoft (R) VSPerf Command Version 10.0. 30128 x86
...
Successfully launched process ID:4144 Application.exe
C:\ApplicationToProfile > vsperfcmd -shutdown
Microsoft (R) VSPerf Command Version 10.0. 30128 x86
...
Shutting down the Profile Monitor
```

Инструмент IntelliTrace

IntelliTrace — новый инструмент в составе редакции Visual Studio 2010 Ultimate, который существенно упрощает отладку приложения. Его работа напоминает работу бортового регистратора, только не во время полета, а во время сеанса отладки, и позволяет просматривать значения и состояние приложения, какими они были в процессе отладки. В дополнение к этому сеансы работы инструмента IntelliTrace могут быть сохранены, а впоследствии вновь загружены. Тестеры, использующие инструмент Microsoft Test and Lab Manager, также могут собирать информацию о работе инструмента IntelliTrace в процессе тестирования, что позволяет знать о точном состоянии системы при обнаружении ошибки.



В настоящее время инструмент IntelliTrace поддерживает проекты на языках C# и VB, а также на языке F# в экспериментальном режиме. Использовать инструмент IntelliTrace в режиме подключения к процессу или при удаленной отладке нельзя.

После запуска сеанса отладки при включенном инструменте IntelliTrace (Tools⇨Options⇨IntelliTrace) выводится показанное на рис. 55.6 окно IntelliTrace. В нем предоставляется список диагностических событий, которые может обнаруживать инструмент IntelliSense.

Когда происходит новое диагностическое событие, система Visual Studio добавляет его в конец списка. Если вы приостанавливаете выполнение или если достигается точка прерывания, окно IntelliTrace становится активным. Если вы щелкнете на любом из диагностических событий, оно раскроется, и будет показано немного больше информации. Это раскрытое представление содержит список связанных представлений, обновляемый инструментом IntelliTrace для отражения состояния приложения в момент события. Вы можете проверять состояние стека вызовов, просматривать переменные и выполнять любые действия, которые могут выполняться в обычном сеансе отладки. Когда вы будете готовы продолжить выполнение приложения, щелкните на кнопке Return to Live Debugging.

Инструмент IntelliTrace в состоянии перехватывать в процессе сеанса отладки информацию двух типов. Режим IntelliTrace Events включен по умолчанию и содержит события отладчика Visual Studio, такие как запуск приложения или достижение точки останова. Генерация и перехват исключений также являются событиями инструмента IntelliTrace. Когда тестировщик собирает информацию о работе инструмента IntelliTrace, начало и конец теста вместе со всеми сбоями также входят в события IntelliTrace. Наконец, платформа .NET Framework сама по себе в состоянии генерировать некоторые диагностические события. Полный их список можно найти в настройках инструмента IntelliTrace.

Другой тип информации, которую может отслеживать инструмент IntelliTrace, — вызовы методов вместе с информацией об их параметрах. Чтобы воспользоваться этой информацией, ее нужно включить до начала сеанса отладки (Tools⇨Options⇨IntelliTrace). После активизации Call Information можно переключить окно IntelliTrace в режим Show Calls View (рис. 55.7), в котором показан каждый вызов метода и выход из него вместе с подсписком событий, происшедших в процессе его выполнения.

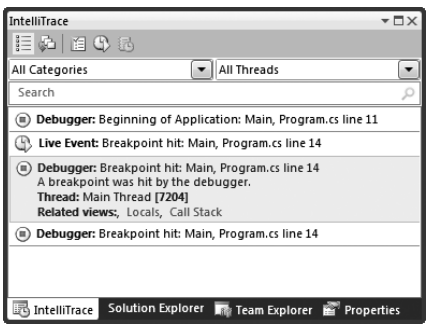


Рис. 55.6

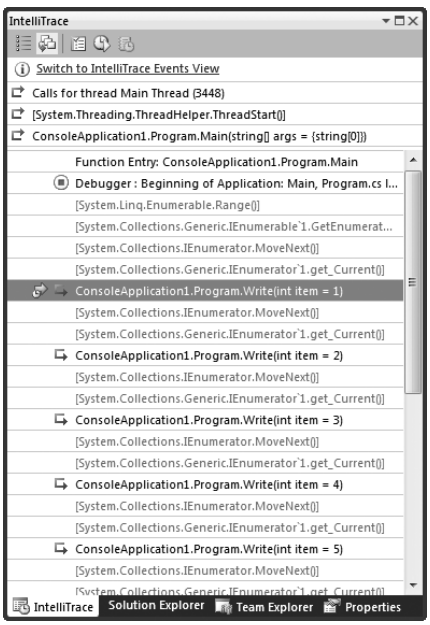


Рис. 55.7

Если вы находитесь в сеансе работы инструмента IntelliTrace с включенной опцией Call Information, в окне редактора добавляется новое поле с маркерами навигации IntelliTrace (рис. 55.8). Их можно использовать для навигации по информации, собранной инструментом IntelliSense, из редактора. Это поле при обычной отладке является светло-серым, но после активизации инструмента IntelliSense становится темно-серым.



Рис. 55.8

Файлы инструмента IntelliTrace (с расширением `.tdlog`) сохраняются на жестком диске в местах, заданных по умолчанию, и могут быть архивированы и заново открыты позже. Когда вы открываете файл с расширение `.tdlog`, появляется представление IntelliTrace Summary. Двойной щелчок на потоке или исключении приводит к открытию сеанса IntelliTrace в соответствующей точке и к началу новой отладки.



По умолчанию файлы инструмента IntelliTrace хранятся в папке `C:\ProgramData\Microsoft Visual Studio\10.0\TraceDebugging`. С помощью команды меню Tools⇨Options⇨IntelliTrace⇨Advanced можно в любой момент проверить, где хранятся эти файлы.

Инструменты для работы с базами данных

Большинству приложений для хранения не используемых в настоящий момент данных требуется тот или иной вид базы данных. Редакция Visual Studio 2010 Premium предоставляет все возможности из редакции VSTS Database и добавляет инструменты, которые ранее были доступны только как Power Tools.

Проект SQL-CLR Database

В диалоговом окне New Project узла Database имеются два узла для проектов SQL-CLR Database на языках программирования C# и VB. Они применяются для создания управляемых классов для использования в системе SQL Server. При создании проекта этого типа вам будет предложено указать подключение и включить отладку по технологии CLR. Используйте CLR-отладку только при разработке серверов баз данных и изолированно от других разработчиков, как будет рассказано позже.

Можете добавить в этот проект некоторые типы наподобие агрегатов данных, пользовательских функций, хранимых процедур, триггеров, пользовательских типов и классов. Данный проект можно рассматривать как обычный проект на языках VB/C#; вы можете добавлять в него классы, ссылки и даже веб-ссылки. Для своих методов можно создавать модульные тесты, описанные в главе 11, выполнять рефакторизацию кода и сборку — таким же образом, как и для других проектов библиотек. История отладки в этом случае несколько отличается, так как ваш код работает в контексте системы SQL Server. Для начала вы должны позволить системе VS удаленно отлаживать код. Это необходимо сделать только при первой отладке проекта. Однако точка входа в код находится в сценарии, который будет использовать ваши управляемые объекты. Как можно заметить при создании проекта, в нем имеется файл `Test.sql`, используемый для создания и тестирования ваших типов.

Тестовые сценарии быстро растут, а длинные сценарии трудно сопровождать. Вы можете добавлять новые тестовые сценарии и устанавливать их как сценарии отладки по умолчанию (Default Debug Script), щелкая на них правой кнопкой мыши. Это аналогично установке страницы ASP.NET в качестве стартовой. Можете установить точку останова в тестируемом сценарии или классе и нажать клавишу <F5> (или воспользоваться командой меню Debug⇨Start Debugging). Это приведет к построению вашего проекта, удалению сборки из системы SQL Server, если она ранее была развернута системой Visual Studio, и последующей регистрации сборки в системе SQL Server с применением процедуры Create Assembly. Наконец, она будет подключена к процессу `SqlServr.exe`, чтобы позволить вам отлаживать сценарии и управляемый код. Вы можете также создать тестовые модули базы данных, которые используют ваши SQL-объекты (включая SQL-CLR типы), как поясняется в соответствующем разделе главы 56.

Автономная схема базы данных

Существует еще один тип Database Project, который находится в узле Microsoft SQL Server, как показано на рис. 55.9. Как можно видеть, имеется по три варианта выбора для каждой из систем SQL Server 2005 и SQL Server 2008, но в основном это один и тот

же тип проекта. Отличия между ними заключаются в применении мастера и целевой версии SQL Server. Некоторые варианты доступны только при создании проекта мастером, так что лучше воспользоваться им. Данные проекты позволяют иметь автономное представление вашей схемы базы данных. Они создают файлы для каждого объекта схемы и вносят изменения в базу данных.

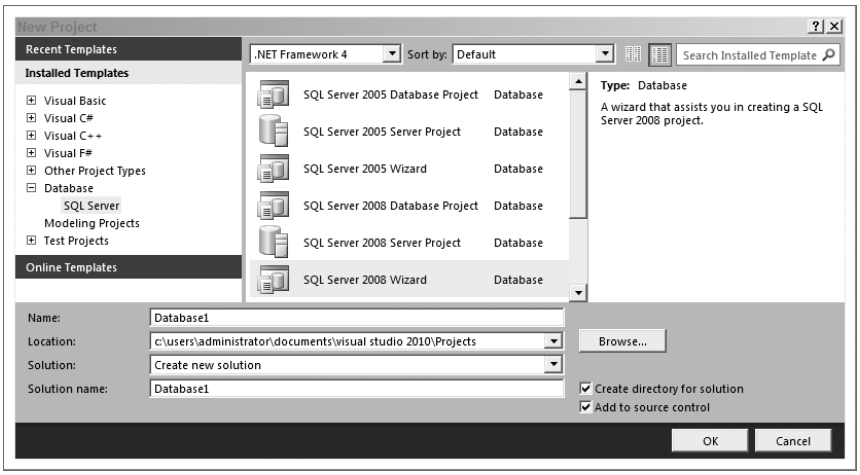


Рис. 55.9

Если вы запустите мастер для системы SQL Server 2008, то первый выбор, который вам придется сделать, — чем будет управлять ваш проект: единой базой данных или системой SQL Server. Это то же самое, что и выбор между Database Project и Server Project на рис. 55.9. Другой выбор на этой странице касается организации файлов в соответствии со схемой или типом объекта (таблицы, представления, хранимые пути и т.д.). Это важно, поскольку после создания проекта этот выбор не может быть изменен. Однако, ответив "Нет" организации в соответствии со схемой, вы все равно сможете отличать схему благодаря префиксу, добавляемому к каждому имени файла. Если же вы выберете организацию в соответствии со схемой, то в каждой из них у вас будут папки для каждого типа. На рис. 55.10 слева показано представление Schema View, отфильтрованное по схемам. Вы можете изменить эту настройку для организации всех объектов по типам. В окне Solution Explorer (на том же рисунке справа) показаны два проекта с разной физической организацией файлов.

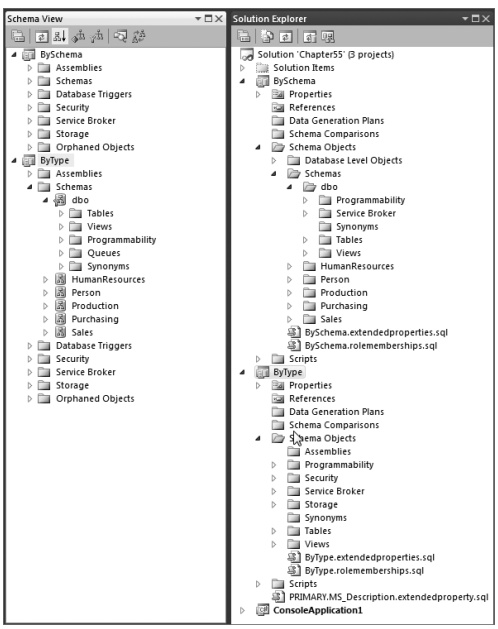


Рис. 55.10

На следующем шаге мастера выполняется настройка базы данных; если вы не знаете своих значений по умолчанию в этот момент, не бойтесь выбрать хоть что-нибудь,

чтобы впоследствии изменить свой выбор, получив необходимую информацию от администратора базы данных. Вы можете выбрать импорт этой конфигурации непосредственно из базы данных. Если вы будете иметь возможность подключения к производственной базе данных, то имеет смысл выполнить импорт настроек прямо сейчас. Для этого просто перейдите к следующему этапу работы с мастером и отметьте необходимые переключатели, как показано на рис. 55.11.

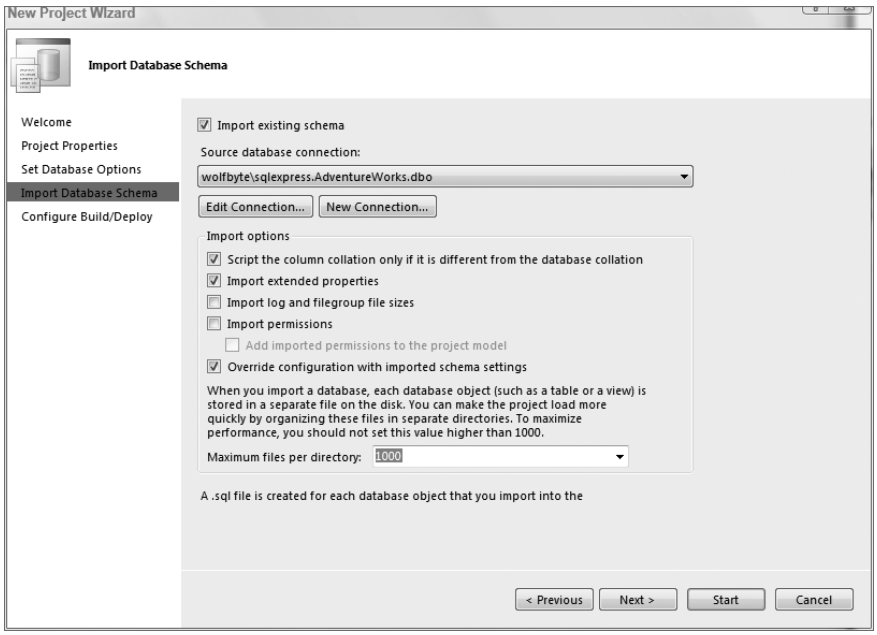


Рис. 55.11

Последний шаг состоит в настройке сборки и развертывания. Вам потребуется больше конфигураций развертывания, так что здесь вы выбираете только свою базу данных разработки. Учтите, что это необязательно должна быть та же база данных, что и на предыдущем этапе: здесь вы настраиваете свою базу данных разработки, а на предыдущем шаге выполняли импорт существующих объектов и конфигурации из промышленной базы данных (при ее доступности). Три необходимые конфигурации будут более подробно рассмотрены позже, а пока можете продолжить работу с настройками по умолчанию. Последний шаг может потребовать некоторого времени, в особенности при большом количестве импортируемых объектов.

По окончании работы мастера вы можете добавить или удалить объекты схемы прямо из окон Solution Explorer или Schema View (View⇨Schema View) — они будут представлены в виде файлов. При сборке проекта будет сгенерирован и выполнен `.sql`-файл, содержащий все инструкции DML, необходимые для воссоздания вашей схемы. Этот сценарий может быть затем развернут в системе SQL Server. Вы можете также выбрать развертывание при каждой сборке, так что ваша база данных всегда будет отражать последние внесенные изменения.

При работе с инструментом Source Control этот тип проекта становится весьма полезным. Рассмотрим сценарий, когда каждый разработчик имеет дело с собственным экземпляром сервера. Один из них вносит изменения в схему и .NET-код и развертывает

их на своем экземпляре. Затем он тестирует .NET-приложение, которое, как выясняется, отлично работает с новыми изменениями базы данных. После он регистрирует все свои изменения на сервере, и их получают и другие разработчики. Когда второй разработчик развертывает свой экземпляр базы данных, он обновляется в соответствии с последними изменениями, и разработчик получает новые изменения в коде вместе с новыми изменениями в схеме, примененные к его экземпляру системы SQL Server.

Генерация данных

В дополнение к управлению версиями схемы вы можете генерировать данные для тестовых целей. Для этого нужно добавить план генерации данных. Щелкните правой кнопкой мыши на проекте Offline Schema под меню Add, и вы найдете опцию Data Generation Plan. По умолчанию будут выбраны все таблицы схемы, но эту настройку можно изменить. В верхней части рис. 55.12 выбрана только таблица `Products`. Таблицы `ProductModel` и `ProductCategory` добавлены автоматически (обратите внимание на затененные переключатели) из-за наличия внешних ключей между таблицей `Products` и указанными.



Рис. 55.12



Помните, что планы генерации служат только для разработки и тестирования. Не выполняйте их на производственных серверах.

В плане можно выбрать отношение между связанными столбцами; так, на рис. 55.12 указано, что в каждой подкатегории должно быть десять продуктов (10:1).

В окне Column Details выбраны все столбцы, за исключением `ProductID`, который представляет собой первичный ключ. Вы можете отменить выбор столбцов при наличии в них значения по умолчанию или если они допускают отсутствие значений (`NULL`). В свойствах окна можно указать, какой процент значений `NULL` будет генерироваться.

Для каждого столбца нужно указать тип используемого генератора. По умолчанию генератор выбирается в соответствии с типом данных, или, в случае столбца с внешним ключом, значения выбираются из первичных ключей связанной таблицы. Вы можете использовать и другие генераторы. В примере на рис. 55.12 для столбца `Name` таблицы `Products` использовано регулярное выражение, объединяющее два имени. Еще один вариант — использование инструмента Data Bound Generator. В этом случае требуется указать строку подключения и запрос `select` для получения вставляемых данных. Можно также создать собственный генератор со своей логикой путем реализации базового класса и регистрации вашей сборки в качестве инструмента Customer Generator. Дополнительную информацию по этому вопросу можно найти в разделе "Creating Custom Generators" в документации MSDN.

Для выполнения плана Data Generation Plan просто выберите команду Generate Data из меню Data⇨Data Generator. Помните, что запускать этот план можно только для тестовой базы данных или базы данных разработки.

Рефакторинг базы данных

В главе 8 рассматривался рефакторинг обычного .NET-кода. В случае баз данных имеется только три варианта рефакторинга — переименование (Rename), перемещение схемы (Move Schema) и полностью квалифицированные имена (Fully-Qualify Names). Вы можете переименовывать таблицы, столбцы, хранимые процедуры, ограничения, индексы и другие объекты базы данных. Вы можете также переименовывать ссылки на другие серверы или базы данных, если таковые используются в сценариях.

Для переименования объекта из представления Schema View щелкните на нем правой кнопкой мыши и выберите команду Rename подменю Refactor. При переименовании объекта вы увидите предварительный вид изменений, показанный на рис. 55.13. В верхней части представлена старая версия, а в нижней — новая для всех зависимостей.

Переименование легко выполняется для автономной версии схемы, но в случае таблиц и столбцов его применение к базе данных может оказаться трудной задачей. Поскольку старый объект будет уничтожен и создан заново, это может привести к потере данных. Если у вас имеется план генерации данных, можете регенерировать их после применения изменений, но если вам необходимо сохранить имеющиеся данные, то следует либо модифицировать сценарий развертывания, либо выполнить рефакторинг вручную. Дополнительная информация по этому вопросу приведена в разделе "Protecting Data during a Renaming Operation" документации MSDN.

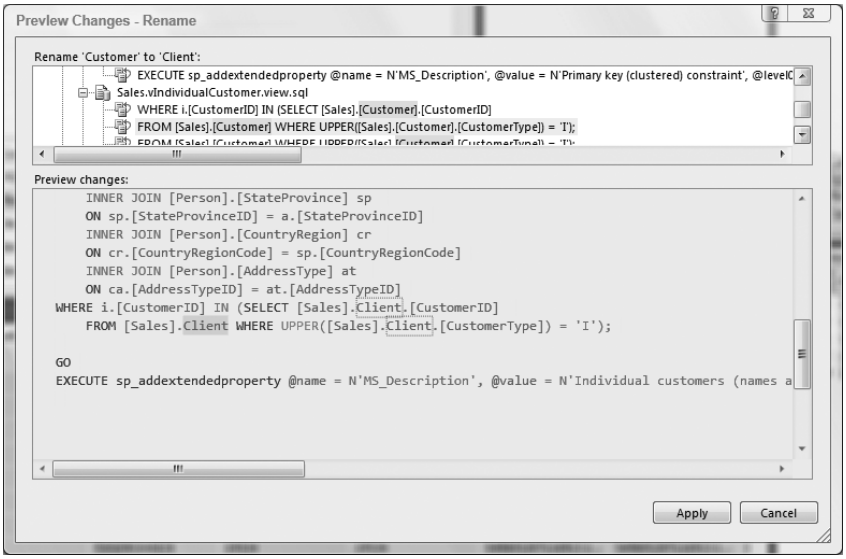


Рис. 55.13

Сравнение схем

Этот инструмент позволяет сравнивать схемы в разных базах данных или проектах Database Project. Для его использования выберите команду New Comparison меню Data⇨Schema Compare. Необходимо выбрать проект или базу данных в качестве источника (Source) и целевой проект или базу данных (Target). Сделав это, вы увидите окно, подобное показанному на рис. 55.14. На нижней панели будут представлены обе версии выбранного объекта, причем отличия будут выделены цветом. Удобно при этом воспользоваться фильтрами, доступными на панели инструментов; по умолчанию показаны все объекты. Чтобы увидеть только различия, выберите опцию Non Skip Objects.

Для каждого объекта базы данных отображено действие, которое будет к нему применено. Затем на панели инструментов можно выбрать либо опцию Write Updates для принятия всех изменений, либо опцию Export to Editor, чтобы получить инструкции SQL, которые будут использованы для обновления (если вы хотите просмотреть их или выполнить вручную).

Для настройки инструмента Schema Comparisons воспользуйтесь командой меню Tools⇨Options и перейдите к узлу Schema Compare в Database Tools. По умолчанию выбрана настройка Block Schema Updates if Data Loss Might Occur. Она рекомендована к применению, но в некоторых случаях может привести к сбою. Вы можете ее отключить, но сначала убедитесь, что вы работаете с тестовой базой данных и сможете сгенерировать данные заново. Прочие настройки, такие как Ignore White Space, могут пригодиться для сокращения количества дополнительных изменений.

Не забывайте, что если вы используете проект типа Database Project, то выбор развертывания приведет к записи изменений из проекта в базу данных, выбранную в настройках свойств проекта. Этот инструмент может быть полезен для просмотра изменений или выполнения настройки вручную.



Рис. 55.14

Сравнение данных

Этот инструмент полезен при копировании данных из одной базы данных в другую. Для запуска мастера воспользуйтесь командой меню Data⇨Data Compare⇨New Data Comparison. Выберите исходную (Source) и целевую (Target) базы данных и сравниваемые записи. В зависимости от того, чего именно вы хотите, можете выбрать различные варианты сравнения — Different Records для обновления, Only in Source для вставки, Only in Target для удаления и Identical Records для справки. Наконец, можете выбрать сравниваемые таблицы и столбцы.

Вам будут представлены все выбранные на последнем шаге объекты, но будут выделены только те, в которых имеются отличия. Рекомендуется использовать фильтр Selected Records, чтобы увидеть только их. Затем можете установить или снять отметки изменений, которые будут применяться на уровне таблицы или строки.

На рис. 55.15 показаны результаты сравнения двух версий простой базы данных Products. В верхней части показаны таблицы, а в нижней — записи выбранной таблицы. На вкладке Different Records нижней панели поочередно отображаются каждый столбец источника и целевой столбец, так что можно увидеть, где именно есть различия.



Инструмент Data Compare доступен для работы только с таблицами в схеме по умолчанию.

На панели инструментов можно выбрать либо Write Updates, либо (для внесения изменений вручную) Export to Editor.

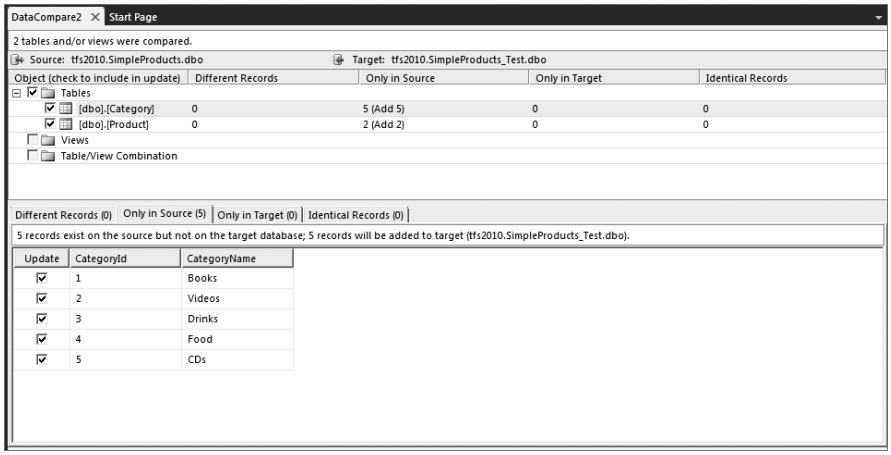


Рис. 55.15

Статический анализ

Редакции Visual Studio 2010 Premium и Ultimate так же, как и для кода, включают инструменты статического анализа для баз данных. Для их запуска воспользуйтесь командой меню Data⇨Static Analysis⇨Run. В настоящее время в разработке баз данных вам помогают 14 правил, разделенных на три категории — Design, Naming и Performance.

Редактор Transact-SQL

Этот редактор позволяет работать с кодом на языке Transact-SQL (T-SQL) непосредственно в системе Visual Studio. Для его открытия можно дважды щелкнуть мышью на `.sql`-файле в окне Solution Explorer или в представлении Schema View проекта Database Project. Еще один вариант — запустить пустой редактор. Для этого воспользуйтесь командой меню Data⇨Transact-SQL Editor и выберите New Query Connection. Теперь можете начать создание кода на языке T-SQL, пользуясь выделением цветом и большинством других возможностей системы Visual Studio, таких как закладки или поиск с заменой. С помощью панели инструментов или меню редактора можно проверять синтаксис, выполнять код, отключаться для автономной работы или подключаться к базе данных для выполнения запроса. При выполнении запросов результаты могут выводиться в виде таблицы, в текстовом формате или экспортироваться в файл. Это поведение также можно изменить с помощью соответствующей команды меню или панели инструментов.

Практические советы

Далее приводятся полезные советы — результат большого практического опыта работы с базами данных в небольших проектах и проектах среднего размера.

- Каждый разработчик должен использовать собственный локальный экземпляр базы данных: один — для разработки и другой — для тестирования. Это необходимо для того, чтобы отделить непринятые и непроверенные изменения и избежать влияния работы других разработчиков над той же базой данных

в то же самое время. Это строго необходимо для целей отладки управляемого кода, так как начало сеанса отладки приводит к остановке всех потоков управляемого кода. В свойствах проекта Database Project можно указать целевую базу данных для каждой конфигурации решения, но проект SQL-CLR может иметь единственную целевую базу данных.

- Каждый разработчик должен использовать две базы данных: одну — для разработки и другую — для тестирования, так как для каждой из них применяются различные данные.
- Используйте в качестве адреса машины `local` или `127.0.0.1` вместо, скажем, `MikesComputer` или `192.168.2.6`, которые будут работать только на одной машине.
- Если вы используете экземпляры базы данных, убедитесь, что у всех разработчиков экземпляры имеют одно и то же имя.
- У всех разработчиков должна быть одна и та же версия системы SQL Server. Хотя для проверок времени разработки и для тестирования можно использовать системы SQL Server Express, некоторые возможности, например индексирование текста, им не поддерживаются.
- Сбросьте флажок Block Incremental Deployment if Data Loss Might Occur в окне свойств проекта для конфигурации решения, используемого для тестовых баз данных. Поскольку у вас имеется план генерации данных, данные после внесения изменений в схему могут легко быть созданы заново.
- При развертывании для производственной базы данных сначала соберите проект Database Project, а затем модифицируйте сценарий сборки для его развертывания на сервере вручную. Для более детального просмотра вносимых изменений можно воспользоваться инструментом Schema Comparison.

Резюме

В этой главе вы познакомились с несколькими передовыми возможностями, являющимися частью Visual Studio 2010 Premium. Их назначение — повысить качество кода. Инструменты Code Metrics и Analysis Tool статически анализируют ваш код или бинарные файлы, собирая различные метрики и проверяя соответствие разным правилам. Эти метрики полезны для выяснения, насколько код сопровождаем. Для анализа кода применяются разные категории правил, помогающие убедиться в том, что код будет работать корректно. Инструменты Profiling Tools исследуют реальную работу кода во время выполнения приложения, а инструмент IntelliTrace позволяет изучать выполнение приложения в процессе сеанса отладки.

В этой главе были рассмотрены и некоторые важные возможности для разработчиков баз данных. Вы узнали, как легко разработать код для проектов SQL-CLR и как Offline Schema Projects помогает работать в команде, контролировать версии и объединять вносимые изменения. Такие передовые возможности, как рефакторинг и модульное тестирование, изменяют способ разработки баз данных, а инструменты наподобие Schema Compare, Data Compare, Data Generation и T-SQL Editor помогают вам в этом процессе.

56

Редакция Visual Studio Ultimate для тестировщиков

В ЭТОЙ ГЛАВЕ...

- Тестирование приложений веб и Windows
- Установка взаимоотношений между кодом и тестами
- Планирование, выполнение и координация задач тестирования
- Управление тестовыми средами

Тестировать приложения можно по-разному. В главе 11 вы ознакомились с концепцией модульных тестов, представляющих собой небольшие выполняемые фрагменты кода, которые проверяют определенные аспекты поведения какого-либо метода или класса. В этой главе рассматривается передовой инструментарий, встроенный в систему Visual Studio и доступный для решения других задач тестирования, включая тестирование веб-приложений и баз данных. Вы узнаете и о том, как отслеживать взаимосвязи между тестами и кодом.

В системе Visual Studio 2010 также содержится новый продукт под названием Test and Lab Manager. Этот инструмент для тестировщиков разработан для непосредственного взаимодействия с системой Team Foundation Server и управления планами, комплектами и контрольными примерами. Инструмент Test and Lab Manager доступен в редакции Visual Studio Ultimate и в качестве части отдельного пакета под названием Test Elements.

Автоматизированные тесты

Автоматизированный тест представляет собой часть кода, которая проверяет поведение приложения без пользовательского ввода или управления. Когда выполняет-

ся автоматизированный тест, система может быть оставлена без присмотра до тех пор, пока тест не будет завершен. Для создания автоматизированного теста в системе Visual Studio 2010 воспользуйтесь командой меню Test⇨New Test, которая выводит диалоговое окно Add New Test, показанное на рис. 56.1.

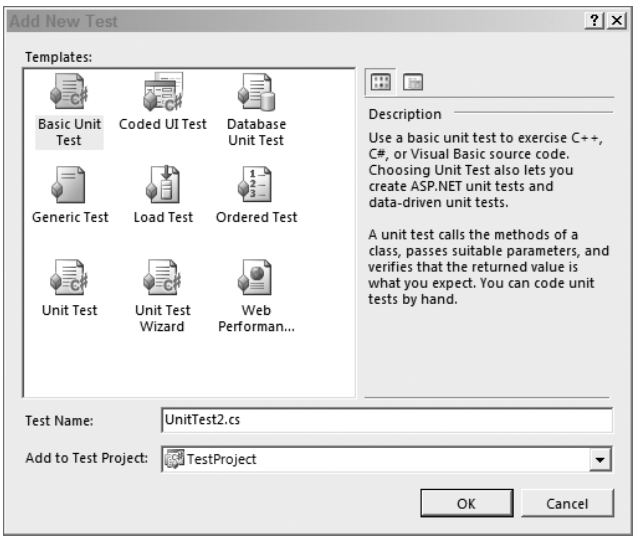


Рис. 56.1



В зависимости от редакции Visual Studio у вас может не быть всех показанных на рис. 56.1 тестов. Тесты Coded UI Test и Database Unit Test доступны только в редакциях Premium и Ultimate. Тесты Web Test и Load Test доступны только в редакции Ultimate. Остальные автоматизированные тесты доступны во всех трех редакциях.

Веб-тесты производительности

Этот тип автоматизированных тестов имитирует веб-запросы и позволяет проверять ответы и выполнять различные вычисления для проверки прохождения теста. Когда вы создаете веб-тест, браузер Internet Explorer открывается с включенной панелью Web Test Recorder, как показано на рис. 56.2. Вы можете посетить сайт, как обычный пользователь. По окончании просто щелкните на кнопке Stop. После этого откроется окно проектировщика Web Test, показанное на рис. 56.3. В нем можно настроить свой тест, добавив правила проверок и извлечения информации, параметры контекста, комментарии, источники данных и вызовы других тестов, а также вставив транзакции. Можно также указать целевое время ответа на свои запросы.



Рис. 56.2

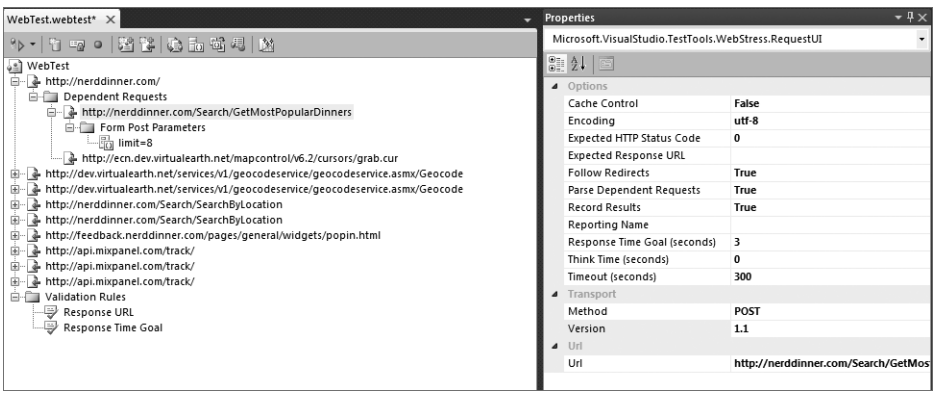


Рис. 56.3

Зачастую необходимо выполнять один и тот же набор тестов для разных веб-серверов. Для этого нужно указать тестируемый сервер в качестве параметра контекста. Щелкните правой кнопкой мыши в Web Test Designer на главном узле и выберите в контекстном меню команду Parameterize Web Servers. Система Visual Studio проверит идентификатор URL в каждом запросе и определит параметры контекста, которые необходимо создать.

Можно связать свои запросы с использованием вывода одного из них в качестве ввода для следующего. Для этого добавьте правила извлечения к определенному запросу. Можно извлекать информацию из полей, атрибутов, заголовков HTTP, скрытых полей и текста и даже использовать регулярные выражения. Результат извлечения определяет параметр контекста, который в дальнейшем может быть применен, например,

в качестве параметра формы или строки запроса. Вы можете сначала добавить продукт, а затем выполнить его поиск с помощью идентификатора в другом запросе.

Добавлять параметры форм и строк запросов можно из контекстного меню запроса. Выбирая параметр формы или строки запроса из окна свойств, можно установить его значение в качестве параметра контекста или связать его с источником данных.

Никакая тестовая схема не выполняется без проверок. Когда вы записываете тест, добавляется правило Response URL Validation Rule, которое утверждает, что ответный идентификатор URL должен быть тем же, что и записанный ответный идентификатор URL. Для большинства сценариев этого недостаточно. Из контекстного меню Web Test можно добавлять правила проверок. Вы можете проверять, есть ли в поле формы или в атрибуте определенное значение или включен ли некоторый дескриптор, находить определенный текст или убеждаться, что реакция на запрос не превышает определенного интервала времени.

Двойной щелчок на `.testrunconfig`-файле в окне Solution Explorer обеспечивает возможность дальнейшей настройки проекта Web Test. Можете выбрать количество итераций, тип браузера и сети и должен ли тестовый процессор имитировать время на раздумья. У вас может быть несколько конфигураций тестов, и в этом случае вы выбираете действующую конфигурацию из меню Test.

Тесты Web Test, как и любые другие типы тестов, выводятся в окне Test List Editor, которое было показано в главе 11. В этом окне можно запустить свой тест; его также можно запустить непосредственно из проектировщика Web Test Designer. После того как тест будет запущен, детальную информацию о нем можно просмотреть, дважды щелкнув на нем в окне Test Results. Чтобы открыть это окно, выберите команду Test Results в меню Test. Можно просмотреть состояние каждого запроса, общее время его выполнения и количество байтов. После выбора запроса вы увидите детали как выбранного запроса, так и полученного ответа, значения параметров контекста, правила проверок и извлечения информации, а также вид страницы в браузере. Пример окна показан на рис. 56.4.

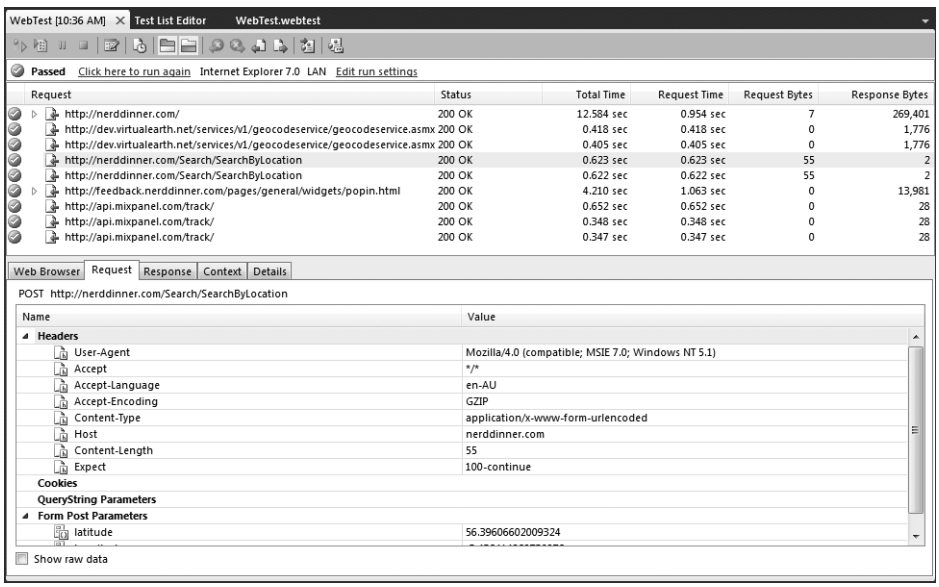


Рис. 56.4

Если вам нужна дополнительная гибкость, закодируйте тест Web Test с использованием технологий .NET и Web Testing Framework. Наилучший способ изучить работу этого программного обеспечения и приступить к работе над своими тестами — сгенерировать код записанного теста Web Test. Это можно сделать в контекстном меню Web Test.



Хотя система Visual Studio и предоставляет некоторые специфичные для системы ASP.NET возможности, тесты Web Test можно использовать и для сайтов, созданных с использованием иных технологий.

Тесты загрузки

Тест Load Test многократно выполняет набор тестов, так что вы можете посмотреть, как ведет себя приложение при разных нагрузках. При создании теста Load Test запускается мастер, который проводит вас через все этапы. Сначала необходимо создать сценарий. В нем определяется, нужно ли использовать имитацию времени на раздумья. При записи теста Web Test время между запросами также записывается и может использоваться в качестве значения времени на раздумья (его можно отредактировать для каждого запроса Web Test в окне свойств).

В качестве части сценария указывается шаблон загрузки; например, это может быть постоянная загрузка 100 пользователями или загрузка, возрастающая каждые 10 секунд на 10 пользователей, пока не достигнет 200 пользователей. Следующие этапы — Test, Browser и Network Mix — определяют, как тесты запускаются виртуальными пользователями, какие браузеры используются и какие типы сетей должны имитироваться. На этапе Test Mix можно добавить тесты Generic, Ordered и Web Test.

На этапе Counter Sets добавляются компьютеры, над которыми должен выполняться мониторинг, и интересующие вас показатели производительности. Например, можете отслеживать свой сервер баз данных и серверы системы IIS. На последнем этапе, Run Settings, можно указать продолжительность теста или количество итераций тестов, как часто должны выполняться пробы для счетчиков производительности, описание теста, сколько идентичных ошибок будет записано и уровень проверок. Уровень проверок определяется для каждого правила проверки веб-теста. Поскольку применение правил может оказаться дорогостоящей операцией, в тестах Load Test применяются только правила с уровнем не выше указанного уровня проверки.

После щелчка на кнопке Finish вы увидите проектировщик Load Test Designer, показанный на рис. 56.5. Здесь можно добавить дополнительные сценарии, наборы счетчиков или новые настройки работы.

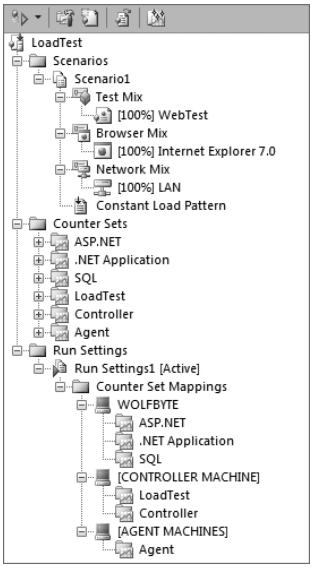


Рис. 56.5

Запустив тесты, вы увидите окно Load Test Monitor; по умолчанию оно показывает представление Graphs. В левой его части имеется представление счетчиков в виде дерева. Вы можете выбирать элементы, которые следует добавить к графикам. С помощью панели инструментов можно заменить данное представление на Summary или Tables, экспортировать данные в таблицу Excel или в файл формата CSV, а также добавить примечания. В нижней части представления Graphs имеется панель условных обозначений, как показано на рис. 56.6. На ней можно выбрать счетчики, которые должны быть показаны на графиках, или отменить сделанный выбор. Во время работы теста результаты мониторинга обновляются после каждой очередной пробы. В представлении Tables отображаются таблицы `Requests`, `Errors`, `Pages`, `SQL Trace`, `Tests`, `Thresholds` и `Transactions`.

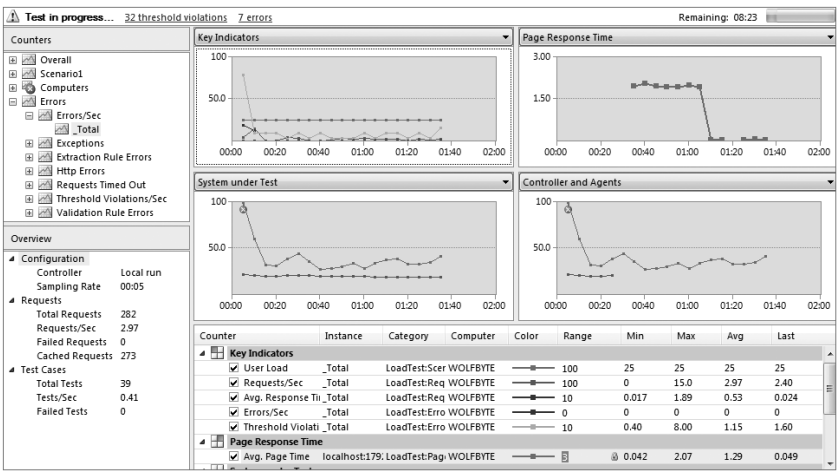


Рис. 56.6

Особенно важной является таблица `Thresholds`. В ней представлены значения каждого счетчика производительности, что позволяет выявить возможные проблемы. Кроме того, на графиках с помощью пиктограмм ошибок или предупреждений указываются точки, в которых нарушались правила.

Инструмент Test Load Agent

Для крупномасштабных приложений одного компьютера может быть недостаточно для имитации требуемой загрузки. Инструмент Visual Studio Team System 2010 Test Load Agent может распределять работу по разным машинам. Он в состоянии имитировать около тысячи пользователей на процессор. Этот продукт требует отдельной инсталля-

ции, одного контроллера и как минимум одного агента. Для настройки среды воспользуйтесь командой Administer Test Controller меню Test. Здесь можно выбрать контроллер и добавить агентов. Затем в окне Test Run Configuration в узле Controller and Agent можно выбрать удаленный запуск тестов и сконфигурированный контроллер.

Модульные тесты баз данных

Вы уже встречались с модульным тестированием в главе 11. В этом разделе данная тема распространяется на базы данных. Этот вид тестов полезен для проверки функциональности и проектирования объектов схемы и может работать вместе с планом Data Generation Plan и проектом Schema Database Project. Для создания теста Database Unit Test выберите из меню Test команду New Test и шаблон Database Unit Test, после чего при необходимости создайте проект. Для первого теста в проекте введите базу данных, с которой будет работать тест. Ее можно будет изменить позже, воспользовавшись командой Database Test Configuration меню Test. Крайне рекомендуется иметь для тестовых целей для каждого разработчика выделенную базу данных. Можно также выбрать развертывание проекта перед запуском тестов. Это гарантирует, что вы всегда будете тестировать последнюю версию схемы. Наконец вы можете использовать план Data Generation Plan для генерации данных для модульных тестов.

Для создания и редактирования тестов имеется проектировщик, показанный на рис. 56.7. Здесь вы можете выбирать существующие тестовые методы или создавать новые, а также добавлять новые условия тестирования. Доступны условия Empty ResultSet, Execution Time, Inconclusive, Not Empty ResultSet, Row Count и Scalar Value. Выбрав условие, настройте его в окне свойств, например, указав значение для количества строк. Можете также создать собственные условия тестирования. Условия Data Checksum и Expected Schema, которые ранее представляли собой часть пакета Power Tools, в настоящее время доступны по умолчанию.

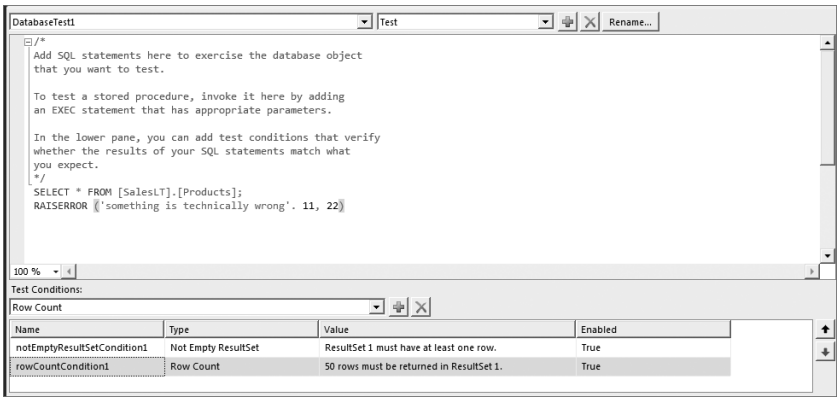


Рис. 56.7

Существует еще два способа проверки тестов. Один из них — генерация ошибок в коде на языке T-SQL. Функция RAISERROR получает сообщение об ошибке (оно будет выведено в окне Test Results в случае сбоя), серьезность ошибки и ее состояние. Если серьезность ошибки превышает 10, она вызывает неудачное завершение теста.

Другой способ проверки тестов выполняется из ваших сценариев на языках C# или VB.NET. Щелкните правой кнопкой мыши на тесте и выберите View Code, и вы увидите, что он структурирован, как обычный модульный тест из главы 11, и просто имеет метод, который вызывает сценарии SQL. Выполняя сценарий, вы получаете массив `ExecutionResult`; каждый объект `ExecutionResult` будет содержать множество данных `DataSet`, количество строк и время выполнения. Используя эти свойства, можете создавать собственные проверяемые утверждения. Метод `Execute` класса `TestService` также получает параметр `SqlParameter`, который может быть передан вашему сценарию и возвращен коду на языке C# для дальнейшей проверки.

Тест Coded UI Test

Иногда наилучший способ протестировать приложение заключается в управлении им извне в качестве пользователя.

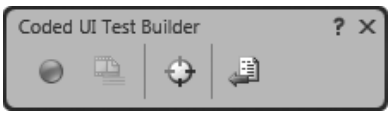


Рис. 56.8

Когда вы создаете тест типа Coded UI Test, он запускает инструмент Coded UI Test Builder, показанный на рис. 56.8. После щелчка на кнопке Start Recording инструмент Coded UI Test Builder отслеживает все действия, выполняемые вами с помощью мыши и клавиатуры.

Откройте свое приложение и воспользуйтесь им для перехода в то состояние, которое вы хотели бы протестировать, а затем щелкните на кнопке Generate Code. Вам будет предложено именовать свой записанный метод, который будет сохранен в тестовом проекте как часть отображения UI Map. Это отображение является описанием действий и утверждений, которые можно использовать для автоматизации и тестирования приложения.



Каждый тестовый проект содержит единственное отображение UI Map, которое совместно используется всеми тестами Coded UI Test.

Когда ваше приложение находится в требуемом состоянии, можете создать утверждения о разных частях пользовательского интерфейса. Для этого перетащите пиктограмму с перекрестием из окна Coded UI Test Builder в ту часть пользовательского интерфейса, для которой вы намерены создать утверждение. Когда вы отпустите кнопку мыши, будет выведено диалоговое окно Add Assertions, показанное на рис. 56.9.

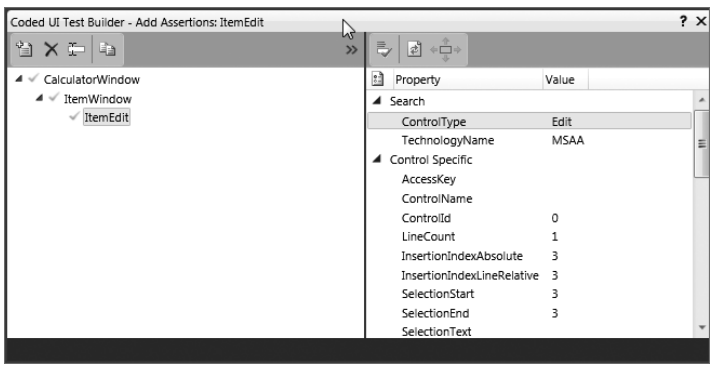


Рис. 56.9

В левой его части находится сворачиваемая панель, на которой показана иерархия всех управляющих элементов, идентифицированных к настоящему моменту. В правой части находится список свойств, которые инструмент Coded UI Test Builder в состоянии идентифицировать, вместе с их значениями. Чтобы сделать утверждение, касающееся одного из этих свойств, щелкните на нем правой кнопкой мыши и выберите из контекстного меню Add Assertion. Каждое утверждение содержит компаратор и значение, с которым производится сравнение.

Обобщенные тесты

Система Team System охватывает не все разновидности тестов, и потому Microsoft включила концепцию обобщенных тестов (Generic Test), так что вы можете легко применять пользовательские тесты, но при этом использовать остальные возможности, такие как Test Results, Test List, Assign Work Items и Publish Test Results.

Для настройки обобщенного теста укажите существующую программу и (необязательно) ее аргументы командной строки, дополнительные развертываемые файлы и переменные среды. Внешнее приложение может возвращать результаты тестов Team System двумя способами. Один из них — код возврата (Error Level), значение 0 которого указывает на успешное завершение, а все иные значения рассматриваются как ошибки. Другой способ заключается в возврате XML-файла, соответствующего схеме `SummaryResult.xsd`, которую можно найти в инсталляционном пути системы Visual Studio. Дополнительную информацию по этому вопросу — как о самой схеме, так и о том, как подробно сообщать об ошибках с помощью XML — можно найти в документации MSDN.

Упорядоченные тесты

Упорядоченный тест (Ordered Test) используется тогда, когда необходимо сгруппировать тесты и выполнить их как единое целое или когда тесты зависят один от другого и должны выполняться в определенном порядке. Установившаяся практика заключается в создании атомарного модульного теста, чтобы его можно было запускать изолированно и получать повторяемые результаты. Я не рекомендую прибегать к упорядоченным тестам только для того, чтобы справиться с зависимостями между модульными тестами. Причиной применения упорядоченного теста может служить создание сеанса выполнения более чем для одного теста.

В редакторе упорядоченных тестов Ordered Test Editor имеется список доступных тестов, которые можно добавлять в упорядоченный тест, причем один и тот же тест может быть добавлен более одного раза. Можете также указать, что тесты должны продолжать выполняться даже при неудачном завершении какого-либо из них. После запуска упорядоченный тест выполняет отобранные тесты в указанном порядке.

Связывание кода и тестов

Тесты и код очень сильно взаимосвязаны. Для существования тестов без тестируемого кода нет никакой причины, а код, не протестированный с помощью тестов, потенциально неверен. Система Visual Studio включает два инструмента, разработанные для явной связи тестов и кода. Инструмент Code Coverage в состоянии определяет, какие области вашего кода выполняются в процессе тестирования, о чем и сообщает

вам при необходимости добавления новых тестов в ваше решение. Инструмент Test Impact Analysis позволяет определить, какие тесты должны быть выполнены заново, основываясь на измененных вами частях кода.



Инструменты Code Coverage и Test Impact Analysis доступны в система Visual Studio 2010 только в редакциях Premium и Ultimate.

Инструмент Code Coverage

Это очень полезный инструмент. Он помогает выяснить, какие строки кода в действительности выполняются при тестировании. Для этого для начала в вашем решении должен быть проект Test Project, о котором вы можете подробнее прочесть в главе 11. Предположим, что у вас уже есть класс `SubscriptionTest` и тест `CurrentStatusTest` уже пройден; теперь вы можете активизировать инструмент Code Coverage.

Чтобы открыть окно свойств Test Run, дважды щелкните на файле `Local.testsettings` в окне Solution Explorer или воспользуйтесь меню Test⇨Edit Test Run Configurations и выберите свою активную конфигурацию. Настройки Code Coverage (рис. 56.10) располагаются на странице Data and Diagnostics.

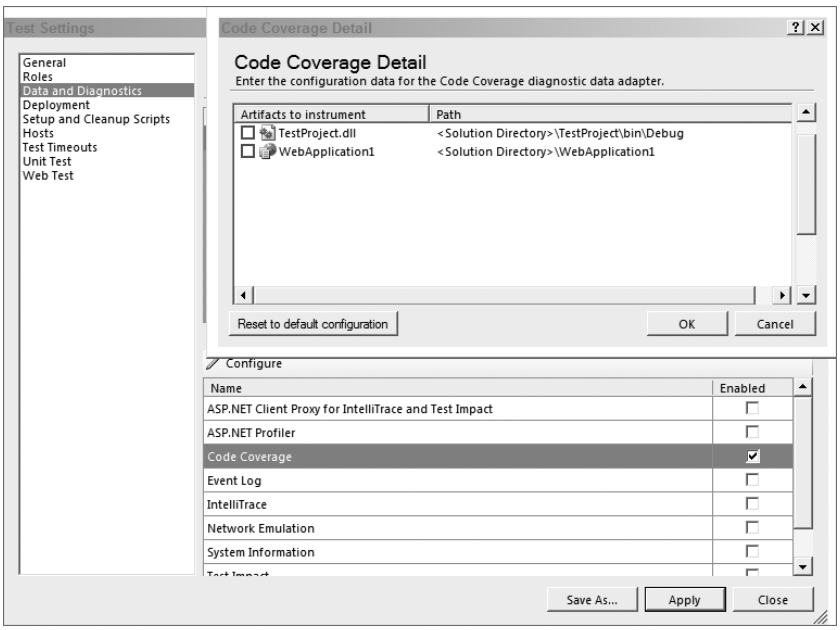


Рис. 56.10

Выберите сборки, с которыми вы намерены работать. Даже если они подписаны, подпишите их повторно, как и в случае профилирования.

Теперь просто запустите свой тест, в окне Test Results щелкните на нем правой кнопкой мыши и выберите в контекстном меню команду Code Coverage. На рис. 56.11 показано окно Code Coverage, указывающее охваченные и не охваченные тестом блоки каждой сборки, пространства имен, типа и члена. Двойной щелчок на члене

открывает файл кода, подсвечивая выполненные строки синим цветом (светло-серый фон на рисунке книги), а не выполненные — красным (темно-серый фон на рисунке книги), как показано на рис. 56.12.

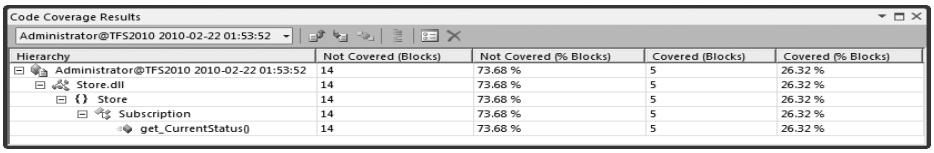


Рис. 56.11

```
public Status CurrentStatus
{
    get
    {
        if( !PaidUpTo.HasValue) return Status.Temporary;
        if (PaidUpTo.Value > DateTime.Now)
            return Status.Financial;
        else if( PaidUpTo.Value >= DateTime.Now.AddMonths(-3) )
            return Status.Unfincancial;
        else
            return Status.Suspended;
    }
}
```

Рис. 56.12

Как можно видеть на рис. 56.11, `get_CurrentStatus()` содержит 73.68% неохваченных тестом блоков. Первая же проверка в методе приводит к выходу из него, так что последующие строки кода не выполняются. Это указывает на необходимость дополнительных контрольных примеров. Обратитесь к разделу главы 11, в котором дано определение источника данных с дополнительными входными данными.

Наличие кода, который не был протестирован, заставляет задуматься о следующем.

- Это код, который в действительности не используется вообще и только мешает, снижая легкость сопровождения проекта. Решение: просто удалите его.
- Это код, который не был протестирован. Решение: создайте новый тест.
- Это код, который настолько прост, что не нуждается в тестировании. Дважды хорошенько подумайте, прежде чем принять такое решение даже в случае простого свойства или метода. Код, обратиться к которому можно из любого места приложения, в случае наличия ошибки может привести к проблемам в любом месте приложения. Этого достаточно, чтобы написать новый тест. Щелкните правой кнопкой мыши на члене и выберите из контекстного меню команду Find All References. Это поможет вам определить, не является ли код неиспользуемым, но не поможет найти обращения из свойств, связанных с данными, или из проектов, не находящихся в данном решении.

Нет необходимости обязательно доводить показатель охвата тестами до 100%. Во многих проектах это излишне, в особенности при наличии старого кода или если вы не приступали к модульному тестированию. Вам просто нужно не допускать снижения охвата своего кода. Итерация за итерацией, а еще лучше (при использовании непрерывной интеграции) проверка за проверкой, ваш процент охвата должен расти или хотя бы оставаться неизменным. Еще важнее, чем следить за процентами, следить за тем, чтобы не пропустить критично важные контрольные примеры.

Инструмент Test Impact Analysis

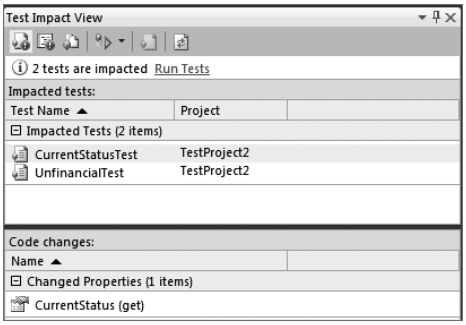


Рис. 56.13

Регулярное выполнение всех модульных тестов по мере разработки новых функциональных возможностей и исправления ошибок, чтобы убедиться в том, что остальной код при этом остался незатронутым и работоспособен, — хорошая практика. Тем не менее не все тесты необходимо запускать всякий раз при внесении изменений в код. Инструмент Test Impact Analysis в состоянии определить, на результаты каких тестов могут повлиять те или иные изменения, внесенные в ваш код. Окно Test Impact View (рис. 56.13) разделено на две части. В верхней части выводится список тестов, которые нужно выполнить заново, основанный на изменениях, внесенных в исходные тексты. После выбора теста (или тестов) в верхней части окна в нижней части выводятся измененные методы.

Данные о влиянии вносимых изменений на тесты не отслеживаются постоянно, так как это приводило бы к слишком высокой нагрузке и снижению производительности интегрированной среды разработки. Вместо этого указанные данные обновляются всякий раз при сборке вашего приложения и запуске тестов. Если данные о влиянии изменений на тесты устарели, становится доступной кнопка обновления в окне Test Impact View.



В окне Test Impact View имеется кнопка для запуска всех тестов, на которые повлияли внесенные изменения.

Управление тестами в системе Visual Studio

Простейший способ управления тестами в системе Visual Studio — использовать Test List Editor, о котором рассказывалось в главе 11. Здесь вы можете добавить тесты в списки, выбрать выполняемые тесты или списки, выполнить их сортировку, группирование или фильтрацию, а также включить их или отключить. Для запуска инструмента Test List Editor выберите соответствующую команду из меню Test Windows.

Другой возможностью является использование окна Test View, которое проще, чем редактор списков Test List Editor. Здесь вы также можете запускать и фильтровать тесты, но не можете группировать их в списки тестов.

Когда вы запускаете набор тестов, выводится окно Test Results. Если оно не появляется автоматически, откройте его с помощью меню Test⇨Windows. Здесь можно сортировать тесты и результаты и просматривать сообщения об ошибках. Вы можете выбирать, какие тесты следует выполнить повторно; по умолчанию будут отмечены тесты, не пройденные успешно, неубедительные и не запускавшиеся, а с успешно пройденных отметка будет снята. Поскольку список результатов может быть большим, можете использовать сортировку, фильтрацию и группирование и даже изменить представление для вывода на левой панели Test Lists с использованием той же организации, что и определенная в Test List Editor.

Из окна Test Results результаты можно экспортировать и сохранять в `.trx`-файл. Вы можете просто оправлять эти файлы каждому, кто сможет импортировать их в свое окно Test Results. Этот человек увидит те же результаты, что и тестировщик, и даже может перезапустить тесты, чтобы воспроизвести ошибки на своей машине. Это возможно, поскольку бинарные файлы включены вместе с выводом каждого теста.

Вместо передачи файлов с результатами тестов с одного компьютера на другой лучше опубликовать их в системе Team Foundation Server. Этот вариант действий доступен на панели инструментов окна Test Results. Вам будет предложено выбрать командный проект; этот проект должен быть с настроенной сборкой, о чем рассказывается в главе 57. Преимущества публикации, помимо того, что она обеспечивает доступ к данным для других людей, заключаются в возможности использовать данные для отчетов.

Инструмент Test and Lab Manager

Инструмент Test and Lab Manager (рис. 56.14) представляет собой новый инструмент для планирования, выполнения и отслеживания тестирования. Он интегрирован непосредственно в систему Team Foundation Server и позволяет создавать, обновлять и запрашивать рабочие элементы.

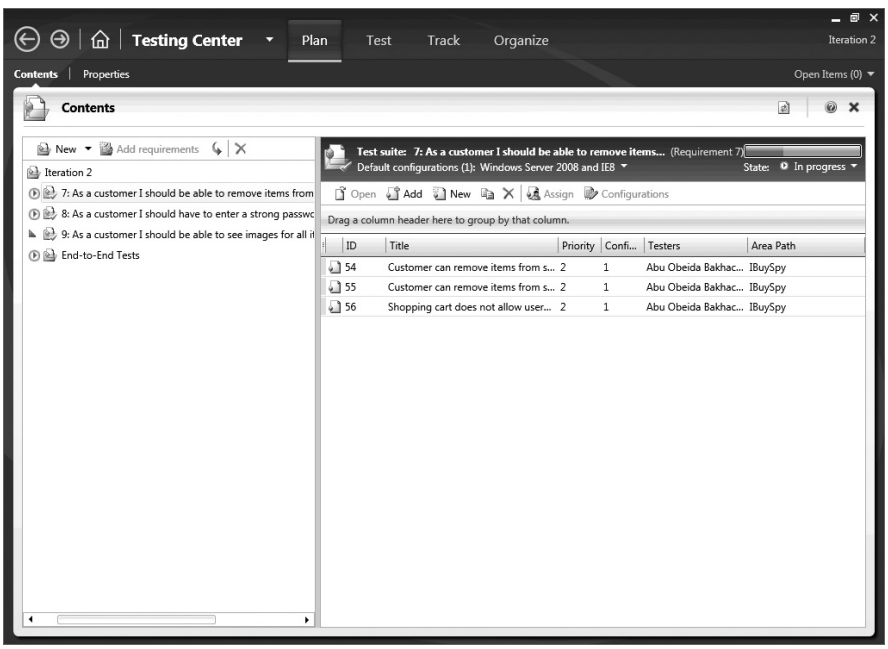


Рис. 56.14

У инструмента Test and Lab Manager есть два пользовательских интерфейса. Когда вы впервые открываете приложение, он выглядит так, как показано на рис. 56.14. Это окно Testing Center. В нем имеется четыре вкладки, связанные с основными видами деятельности интерфейса Testing Center по созданию и запуску комплектов тестов.

Окно Testing Center

При первом запуске окна Testing Center вам предлагается создать план. Все тесты в окне Testing Center организованы в планы, и рекомендуется создавать свой тестовый план для каждой выпускаемой версии вашего приложения. Если у вас уже есть план, вы увидите представление Contents View, показанное на рис. 56.14. Здесь можно создавать новые тесты, добавлять существующие и назначать своим тестам тестировщиков и конфигурации.



Каждый план имеет состояние In Planning, In Progress или Complete. Эта информация хранится в системе Team Foundation Server и может быть представлена в отчетах. Когда план находится в работе, доступна также информация о том, какая доля запланированных контрольных примеров выполнена.

Открыв существующий контрольный пример или создав новый, вы увидите окно Test Case (рис. 56.15). Каждый контрольный пример состоит из ряда действий, показанных в нижней части окна. Каждое действие сопровождается описанием, что именно должен сделать тестировщик, а также как проверить успешность завершения действия.

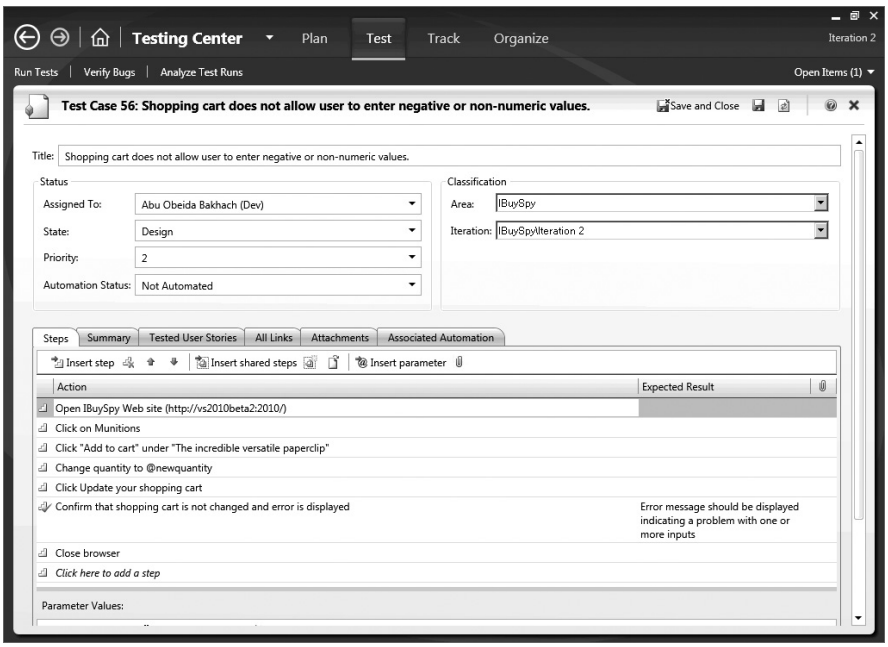


Рис. 56.15

Действия могут иметь параметры наподобие показанного на рис. 56.16 у четвертого действия. Каждый параметр может иметь несколько определенных значений; при этом выполняется по одной итерации теста для каждого из значений.

Для запуска контрольного примера выберите его на вкладке Test и щелкните на кнопке Run. Откроется окно Test Runner (рис. 56.16). В нем выводятся список шагов и информация о ходе тестирования. По завершении каждого шага можете отметить его как успешно пройденный или как не пройденный.

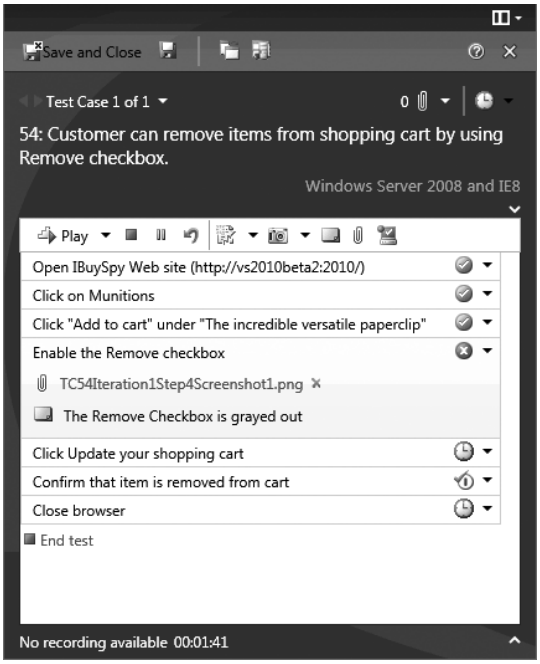


Рис. 56.16



Вы можете записывать шаги выполняемого вручную теста; это позволит вам автоматизировать процесс его выполнения, если потребуется пройти его позже еще раз.

На панели инструментов окна Test Runner имеются кнопки, которые позволяют связывать различные элементы с результатами выполнения теста, включая комментарии, снимки экрана, файлы и даже полный снимок состояния системы, который разработчик сможет использовать при отладке.

Окно Lab Center

Щелкнув на заголовке Testing Center, вы сможете переключиться на Lab Center (рис. 56.17). Lab Center используется для управления средами, в которых производится тестирование. Сюда может включаться информация как о физических, так и о виртуальных средах.

Для использования возможностей окна Lab Center инсталлируйте инструмент Test Controller и свяжите его с коллекцией проектов в системе Team Foundation Server. Если инструмент Test Controller доступен, используйте окно Lab Center для поддержки коллекции физических машин (Physical Machines), виртуальных машин (Virtual Machines) и шаблонов виртуальных машин (Virtual Machine Templates). Когда тестировщик запускает тест, он подключается к одной из этих машин, и если он обнаруживает ошибку, то может сделать снимок системы, который затем будет добавлен к отчету об ошибке. Разработчик, получив эту информацию, сможет подключиться к тестовой машине и вернуть ее в интересующее его состояние.

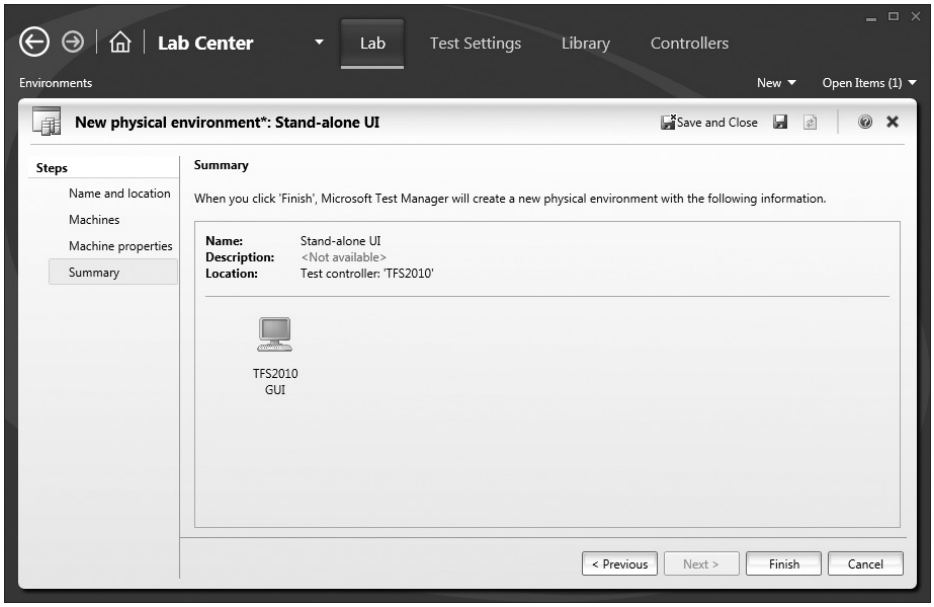


Рис. 56.17

Для конфигурирования данных, которые собираются на каждой машине в среде, воспользуйтесь страницей Data and Diagnostics на вкладке Test Settings (рис. 56.18). Можно накапливать множество разных видов данных — от щелчков мышью и нажатий клавиш клавиатуры до полного видео рабочего стола во время тестирования.

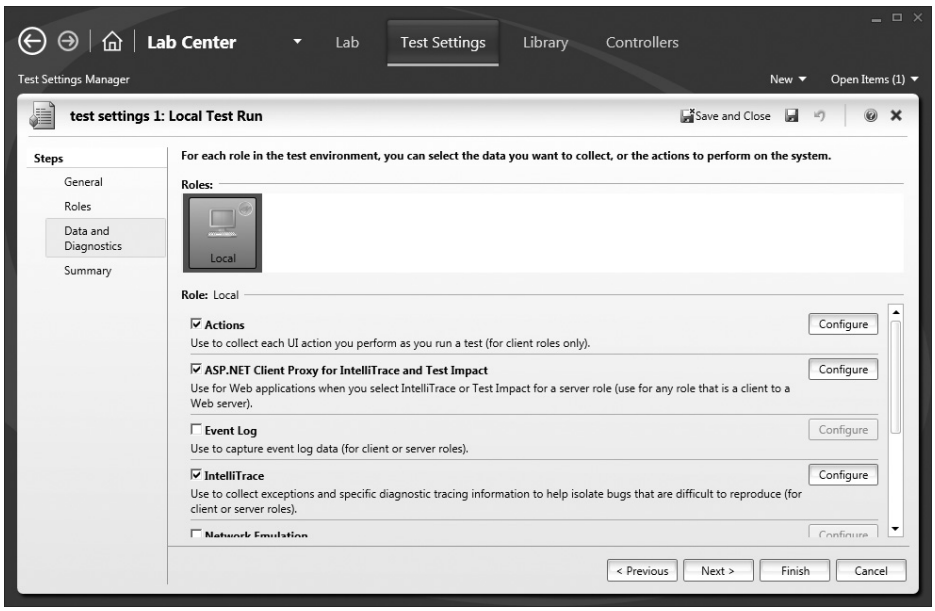


Рис. 56.18

Резюме

В этой главе вы ознакомились с типами автоматизированных тестов, включенных в систему Visual Studio 2010: от веб-тестов, которые позволяют воспроизводить набор запросов, до тестов загрузки, которые позволяют имитировать одновременную работу нескольких пользователей. Вы также ознакомились с автоматизацией приложения с помощью тестов Coded UI Test, что помогает протестировать способы взаимодействия пользователя с вашим приложением. Обобщенные тесты могут применяться для "обертывания" существующих тестов, использующих иные механизмы, а упорядоченные тесты помогают при последовательном запуске набора тестов. Вы узнали, как модульные тесты отображаются на тестируемый код с помощью инструментов Code Coverage и как эта информация используется для определения, какие тесты должны быть выполнены при изменении кода. Наконец вы ознакомились с возможностями управления тестами наподобие их группирования в списки и публикации результатов в системе Team Foundation Server.

Новый инструмент Test and Lab Manager предназначен для помощи тестировщикам в их нелегкой работе. Создав контрольные примеры и организовав их в планы, можно легко управлять задачами тестирования, а интеграция с системой Team Foundation Server упрощает отслеживание хода тестирования и обмена результатами с другими членами команды.

57

Система Team Foundation Server

В ЭТОЙ ГЛАВЕ...

- Управление задачами проектов
- Визуализация изменений в репозитории исходного кода
- Создание конфигураций сборок
- Отчеты о ходе выполнения работ
- Настройка шаблонов процессов

Программные проекты печально известны своей сложностью; очень малое их количество укладывается в сроки и бюджеты, не опускаясь ниже приемлемого уровня качества. Чем больше становятся программные проекты и чем бо́льших команд они требуют, тем сложнее становится процесс управления ими, — причем не только для менеджеров, но и для разработчиков, тестеров, архитекторов и даже потребителей. Было разработано множество разнообразных подходов для решения проблем управления программными проектами — включая такие модели, как CMMI (Capability Maturity Model-Integrated — интегрированная модель технологической зрелости организации), и такие методологии, как RUP (Rational Unified Process — рациональный унифицированный процесс разработки) и гибкие методологии Agile Practices, Scrum и Continuous Integration. Ясно, что желательно иметь инструментарий, который поддерживал бы управление программными проектами.

Базовым требованием к программному проекту, даже в случае небольшого проекта для одного разработчика, является наличие репозитория исходных текстов с управлением версиями. Для более сложных проектов требуются дополнительные функциональные возможности, такие как маркировка, архивация, ветвление и слияние. Задания по работе над проектом должны создаваться, упорядочиваться по приоритетам и назначаться, а их выполнение — отслеживаться, а в конце дня (еще лучше — при

каждом изменении, фиксируемом в репозитории) необходимо убедиться в том, что приложение собирается и проходит все тесты. Для облегчения этого процесса и усовершенствования обмена информацией в пределах команды также требуется соответствующий инструментарий.

Система Team Foundation Server (TFS) 2010 позволяет выполнять все указанные задачи. Из этой главы вы узнаете, как работает система контроля версий, как она интегрирована с отслеживанием рабочих элементов и как может быть проверена работоспособность каждого изменения кода перед его регистрацией в репозитории. Вы увидите, как менеджеры проекта могут использовать отчеты для лучшего понимания состояния дел и применять Excel и Project для распределения работ.



Система TFS 2010 имеет несколько новых функциональных возможностей, которые облегчают работу с ним, включая сниженные требования к среде и упрощенную инсталляцию. Вы можете даже установить его на клиентской операционной системе наподобие Windows 7.

Командный проект

Для начала работы с TFS необходимо создать *командный проект* (team project). В нем содержится вся информация о вашем проекте, включая исходный текст, задачи, которые необходимо выполнить, инструкции по сборке вашего приложения, документация, оценки качества и информация о планировании. Каждый проект может иметь собственный портал совместной работы SharePoint.

В системе Visual Studio 2010 командные проекты группируются в *коллекции командных проектов* (team project collections). Все проекты и коллекции командных проектов используют общую фундаментальную инфраструктуру, такую как хранилища данных, схемы рабочих элементов, и пул идентификаторов (для рабочих элементов и для нумерации массивов изменений). Если на вашем предприятии имеются логические группы проектов, для каждой из них стоит создать коллекцию командных проектов.



Выполнить откат и восстановить отдельные проекты невозможно. Это может делаться только на уровне коллекции проектов.

Шаблоны процессов

Создавая командный проект, выберите *шаблон процесса* (process template), который определяет способ использования данного инструмента. Выберите тот, который наилучшим образом подходит для принятой в вашей организации методологии. В поставку системы Team Foundation Server входят два шаблона, основанные на технологии Microsoft Solution Framework. Один из них нацелен на технологию Agile Development, а второй — на технологию CMMI Process Improvement. Оба варианта отлично подходят в качестве начальной точки, но если ваша компания уже имеет определенный процесс, его шаблон можно встроить в систему TFS или воспользоваться шаблоном стороннего производителя. На рис. 57.1 показан выбор шаблона процесса. В оставшейся части данной главы мы будем основываться на шаблоне MSF for Agile Software Development, а при необходимости — обращаться к версии CMMI.

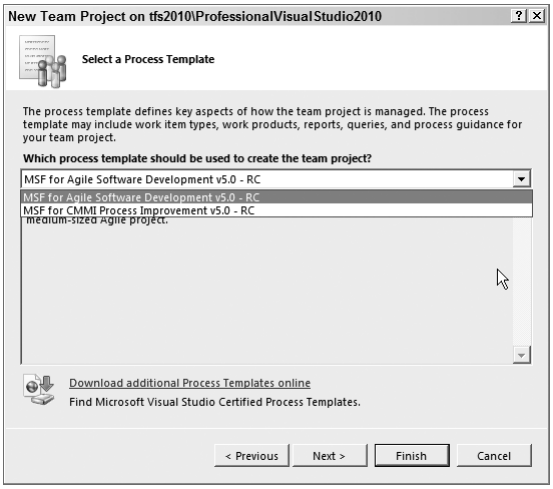


Рис. 57.1

Шаблон процесса создает среду для командного проекта. Это действие обычно включает определение типов рабочих элементов, создание групп и допусков по умолчанию, подготовку репозитория с контролем версий и настройку отчетов и портала SharePoint со структурой документов и руководством процессом. Другой процесс может пропустить некоторые из указанных или добавить собственные задачи.

По окончании работы мастер открывает страницу руководства, которая детально описывает используемый процесс, определяет ответственности ролей, описывает разные типы рабочих элементов и предоставляет пошаговое руководство по конкретным задачам наподобие "как создать Vision Statement". На рис. 57.2 показаны окна Project Guidance и Team Explorer.

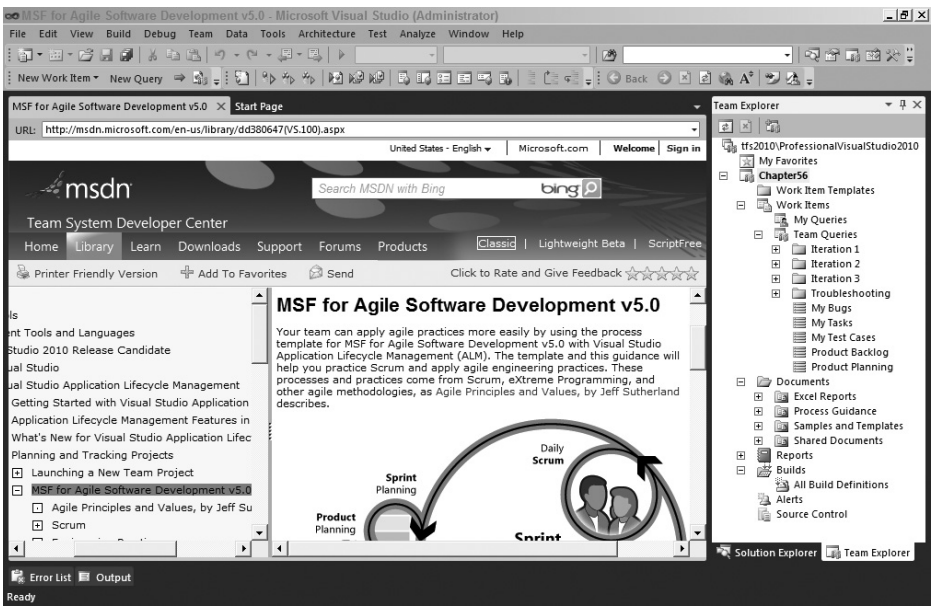


Рис. 57.2

Можете обратиться к различным возможностям системы TFS в окне Team Explorer. В нем содержится дерево Work Item Queries – удобный способ обращения к документам, хранящимся в портале Team Portal, ссылкам на отчеты, списку Team Builds и узлу Source Control.

Отслеживание рабочих элементов

Система Team Foundation Server позволяет управлять всей деятельностью с помощью *рабочих элементов* (work items). Как вы увидите в следующих разделах, можно находить рабочие элементы с использованием *запросов рабочих элементов* (work item queries) и управлять ими с применением Visual Studio, Excel, Project или Team System Web Access. Типы рабочих элементов определяются в вашем шаблоне процесса.



Одной из наиболее востребованных функциональных возможностей, приведшей к появлению системы TFS 2010, являются иерархические рабочие элементы. Сейчас система TFS предоставляет эту возможность, так что вы можете создавать подзадачи и родительские задачи. Можно также создавать ссылки на предшествующие и последующие рабочие элементы, что позволяет управлять зависимостями между задачами. Эти новые ссылки могут даже синхронизироваться с приложениями Microsoft Excel и Microsoft Project, обеспечивая еще большую гибкость в управлении рабочими элементами.

Запросы рабочих элементов

Рабочие элементы, показанные на рис. 57.3, являются задачами выбранного командного проекта. Искать интересующие вас рабочие элементы можно с помощью запросов из Team Explorer. Шаблон процесса включает 14 командных запросов (рис. 57.2), таких как Active Bugs, Open Issues или My Work Items.

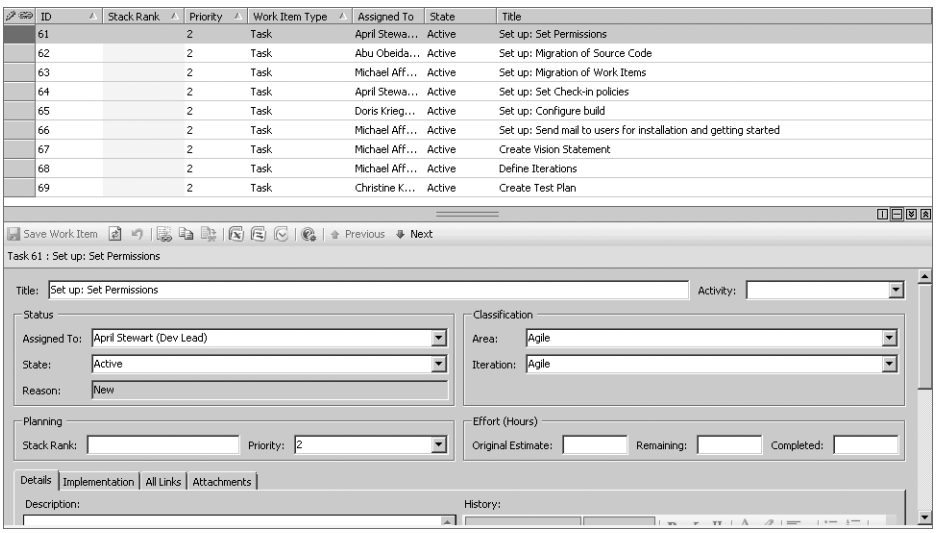


Рис. 57.3

В большинстве случаев этих запросов достаточно, однако вы можете создавать и новые запросы. Если вы — администратор проекта, можете добавлять новые командные запросы, делая их доступными для всех, кто работает с проектом. Если вы можете изменять шаблон процесса, то можете и добавлять новые запросы, так что

проекты, созданные с использованием отредактированных шаблонов, будут их содержать. Изменения в шаблонах не применимы к уже созданным командным проектам. Если таких прав у вас нет, вы все равно можете создавать персональные запросы.



Если вы замечаете, что постоянно создаете одни и те же запросы от проекта к проекту, добавьте их в свои шаблоны процессов. Со временем это снизит необходимость в создании пользовательских запросов.

Для создания запроса щелкните правой кнопкой мыши на узле My Queries и выберите в контекстном меню команду New Query (рис. 57.4).

Теперь можете визуально спроектировать свой запрос. В данном случае нас интересуют только рабочие элементы выбранного проекта, присвоенные текущему пользователю и в рамках итерации Iteration 1. Это указывается с помощью переменных `@me` и `@Project`. Можете также указать, какие столбцы должны быть видимыми в решетке, и настройки сортировки (рис. 57.5). Затем новый запрос можно выполнить для получения подсписка рабочих элементов.

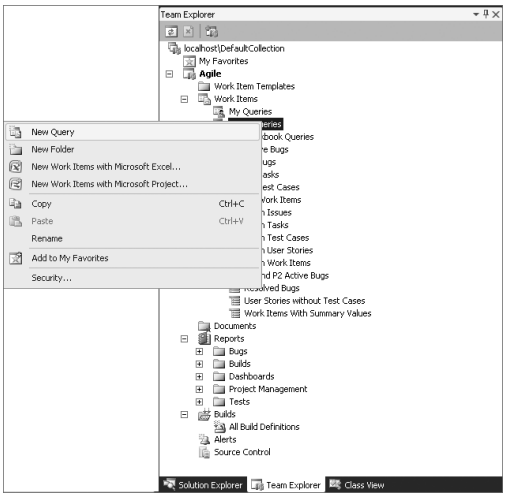


Рис. 57.4

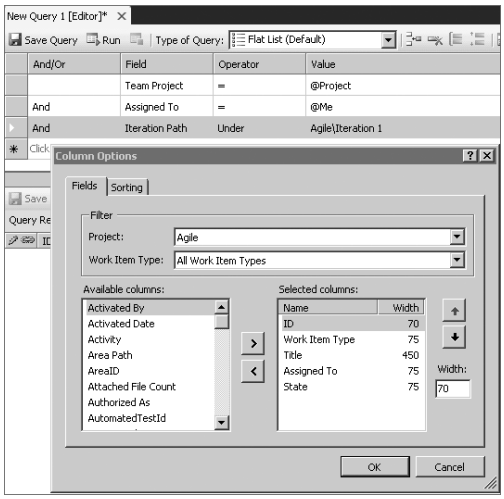


Рис. 57.5

В системе Team Foundation Server 2010 для показа непосредственно связанных рабочих элементов запросы могут воспользоваться преимуществами их новой иерархической структуры, позволяя вам видеть влияние удаления возможности или какие задачи следует выполнить для ее завершения. Можно также просмотреть результаты запроса в виде простого списка, списка рабочих элементов с их прямыми связями или дерева рабочих элементов. Каждый из них идентифицируется небольшой пиктограммой возле запроса в системе Team Explorer. Можете создать для своих запросов рабочих элементов структуру папок, и каждый запрос или папка при этом может быть защищена от несанкционированного доступа.



Хотя папка запросов рабочих элементов и может быть защищена, ничто не помешает неавторизованным пользователям дублировать запросы для себя лично.

Типы рабочих элементов

В инструменте MSF for Agile Development имеется шесть типов рабочих элементов: ошибки (bugs), вопросы (issues), общие этапы (shared steps), задачи (tasks), контрольные примеры (test cases) и пользовательские сценарии (user stories). Каждый рабочий элемент имеет различные поля в зависимости от его типа. Например, в случае ошибки это поля с тестовой и системной информацией, в то время как задача содержит информацию об оценочном рабочем времени для ее выполнения, а также оставшемся и использованном рабочем времени. В шаблоне проекта MSF for CMMI имеется отсутствующий в инструменте Agile рабочий элемент изменения запроса (change-request) и элемент работы над ошибкой (bugwork), но в этом случае он не так прост — здесь он включает информацию о действиях, необходимых для воспроизведения ошибки, а также другие поля, такие как критичность ошибки, ее приоритет и т.п. Все эти поля настраиваются на уровне либо шаблона, либо командного проекта.

Добавление рабочих элементов

Основной способ добавления рабочих элементов — использование команды меню Team⇨Create Work Item с последующим выбором типа добавляемого рабочего элемента. Еще одним способом добавления рабочих элементов является использование окна Test Results (рис. 57.6).

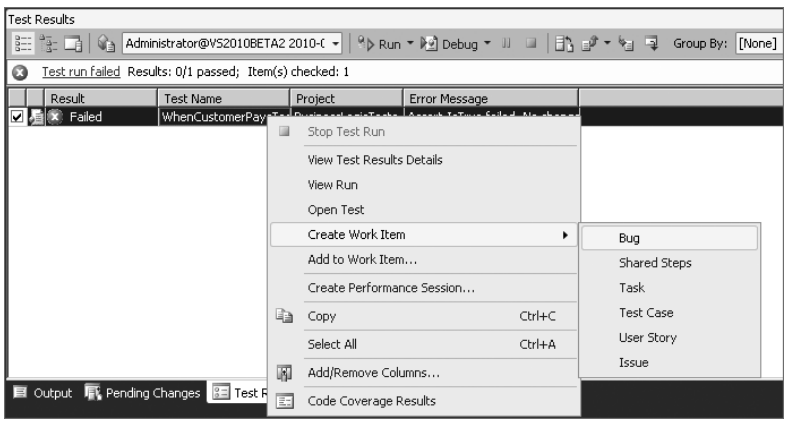


Рис. 57.6

При использовании этого способа обычно создаются элементы для ошибок, а также связи между ними и выделенным тестом. Вы можете переходить от ошибки к тесту или просматривать тест и связанные с ним рабочие элементы (рис. 57.7). Если тест вновь не пройден, можете просмотреть, какие рабочие элементы с ним связаны, и отследить их до связанных с ними множеств изменений, о чем будет рассказано позже. Система Visual Studio 2010 способна связать информацию IntelliTrace с рабочим элементом. Таким образом, разработчик, которому поручено исправление ошибки, может легко ее воспроизвести.

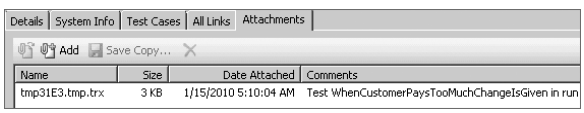


Рис. 57.7

Каждый рабочий элемент может быть связан с другими элементами. Система Team Foundation Server 2010 умеет работать с несколькими типами связей, включая такие отношения, как родительский элемент (Parent), дочерний (Child), предшественник (Predecessor) или преемник (Successor). Для добавления связи щелкните на вкладке All Links и на кнопке Link To. Можете также создать рабочий элемент, непосредственно связанный с данным, с помощью кнопки New Linked Work Item.

Состояние рабочего элемента

Вы ежедневно работаете над задачами, описываемыми назначенными вам рабочими элементами. Каждый рабочий элемент описывается простым состоянием машины, которое определяет для каждого данного состояния допустимые новые состояния. Это состояние машины является частью определения рабочего элемента, и решение о нем принимается шаблоном процесса. Когда новое состояние выбрано, можете указать причину перехода в него. Например, рабочий элемент ошибки в инструменте MSF for Agile может перейти из состояния активного (Active) из-за решения вопроса (Resolved) в состояние завершенного (Closed). Можно также вновь вернуться в активное состояние из-за отката (Regression) или повторного вызова (Reactivated). Поле причины позволяет отличать ошибки, активные потому, что они только что обнаружены, от ошибок, проявившихся вновь.

Интеграция приложений Excel и Project

Поиск, добавление или редактирование рабочих элементов может оказаться сложной задачей, которая плохо масштабируется при наличии сотен задач. Это может оказаться особенно проблематичным для менеджеров проекта, не работающих с системой Visual Studio. Они обычно предпочитают такие программы, как Excel или Project. Интеграция этого программного обеспечения благодаря предоставляемым дополнениям оказывается простой задачей.

Программа Excel

Щелкните на кнопке New List и выберите интересующий вас командный проект и запрос. При этом вся полученная от веб-службы информация будет выведена в таблицу Excel. Здесь вы можете ее отсортировать, отфильтровать и отредактировать, а затем отправить изменения на сервер, а также обновить данные с учетом изменений, сделанных другими, добавить связи и вложения и выбрать выводимые столбцы.

Еще один способ добиться этого — из окна Team Explorer. Выберите в контекстном меню запроса рабочего элемента команду Open in Microsoft Excel. Это приведет к созданию листа Excel с соответствующей информацией. На рис. 57.8 показаны оба описанных варианта, а на рис. 57.9 — результаты в таблице Excel.

Шаблон MSF for Agile создает ряд рабочих элементов стандартных рабочих книг Excel, которые можно найти в подузле Documents узла Excel Reports.

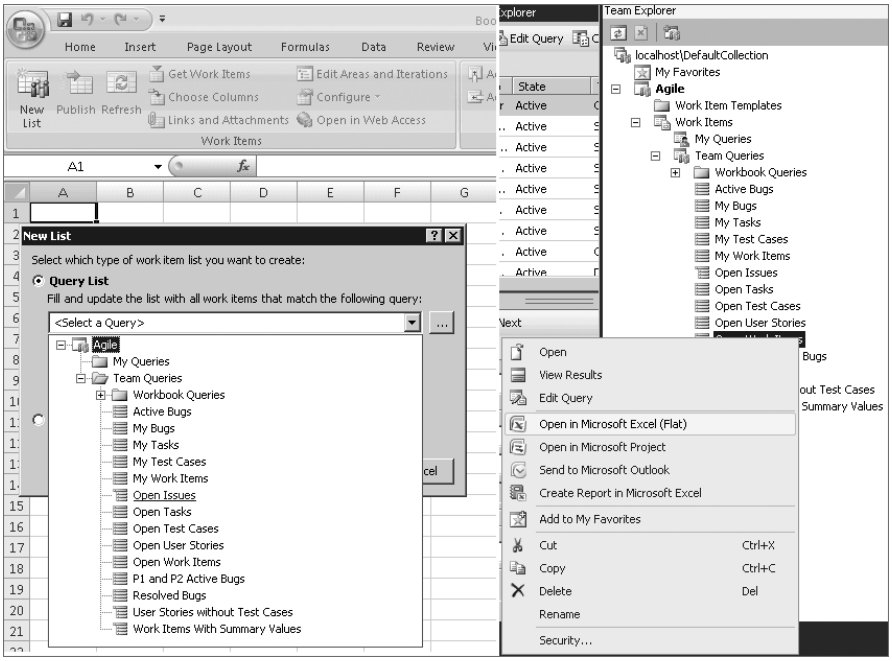


Рис. 57.8

	A	B	C	D	E	F
1	**Project:** Agile **Server:** localhost\DefaultCollection **Query:** Open Work Items **List type:** Flat					
2	ID	Stack Rank	Priority	Work Item Type	Assigned To	State
3	70		2	Bug	Administrator	Active
4	61		2	Task	April Stewart (Dev Lead)	Active
5	62		2	Task	Abu Obeida Bakhach (Dev)	Active
6	63		2	Task	Michael Affronti (PM)	Active
7	64		2	Task	April Stewart (Dev Lead)	Active
8	65		2	Task	Doris Krieger (Dev)	Active
9	66		2	Task	Michael Affronti (PM)	Active
10	67		2	Task	Michael Affronti (PM)	Active
11	68		2	Task	Michael Affronti (PM)	Active
12	69		2	Task	Christine Koch (Tester)	Active

Рис. 57.9

Программа Project

Имеется дополнение и для программы Project. Так же, как и при использовании программы Excel, можно подключиться к серверу, выбрать проект и запрос рабочих элементов, но вместо всего списка надо выбрать каждый из рабочих элементов, который вы хотите импортировать в свой проект, как показано на рис. 57.10.

После того как ваши рабочие элементы будут импортированы, можете редактировать их поля непосредственно в программе Project. Это возможно благодаря отображению между полями системы TFS и столбцами окна MS Project. Например, Resource Names в программе Project будут отображаться на поля Assigned To в системе TFS. Поля, имеющиеся только в системе Team System, будут отображаться на текстовые поля в программе Project; так, поле Work Item Type отображается в поле Text 24. Эти отображения заранее настроены в шаблоне процесса.

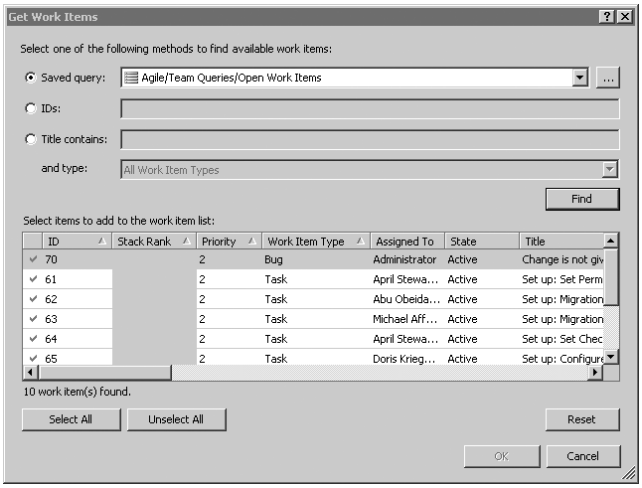


Рис. 57.10

Вы можете добавлять новые рабочие элементы, вкладывать их в итерации или области, назначать работникам, выбирать тип рабочего элемента, распределять ресурсы, просматривать отчеты, а также обновлять информацию с сервера после того, как разработчики отчитаются о проделанной работе, изменяя на сервере поля с информацией о состоянии рабочего элемента или, например, об объеме оставшейся до завершения задачи работы.

В отличие от предыдущих версий системы TFS, редакция 2010 в состоянии оперировать понятиями иерархических рабочих элементов и предшествующими и последующими задачами. Когда рабочий элемент загружается в программу Project, последний в состоянии воспользоваться преимуществами имеющихся связей при создании обычного проекта. При создании и обновлении новых ассоциаций программы Project даже в состоянии вернуть обновленные данные системе TFS в понятном для того виде.

Однако имеются и важные различия, которые делают интеграцию не столь простой.

- В программе Project для одного и того же действия можно иметь много ресурсов, в то время как система TFS принимает только одну запись. Один из подходов к решению этой проблемы заключается в добавлении пользовательских полей к типу рабочего элемента, однако отображение при этом не работает, так как вам нужно, чтобы один столбец в программу Project отображался на два поля системы TFS. Обходной путь заключается в добавлении действий для каждого ресурса.
- В поставке отсутствует интеграция Project server/TFS. Для нее вы можете воспользоваться инструментами, разрабатываемыми сторонними производителями. Такая интеграция появится в следующей версии системы Team System, как и усовершенствование управления проектами для распределения и балансирования ресурсов.



Если вы намерены использовать программу Project в сочетании с системой TFS, ознакомьтесь со статьей "*Quick Tips and Operational Differences when Tracking Tasks using Office Project and Team Foundation*", которую можно найти по адресу `http://msdn.microsoft.com/en-us/library/dd380701(VS.100).aspx`.

Управление версиями

Управление версиями (Version Control), которое иногда называют управлением исходными текстами (Source Control), представляет собой инструмент, который позволяет управлять, совместно использовать и аккумулировать историю изменений вашего исходного текста. Для взаимодействия с системой управления версиями TFS используется окно Source Control Explorer, показанное на рис. 57.11. Можете открыть его из окна Team Explorer или воспользовавшись командой меню View⇨Other Windows⇨Source Control Explorer.

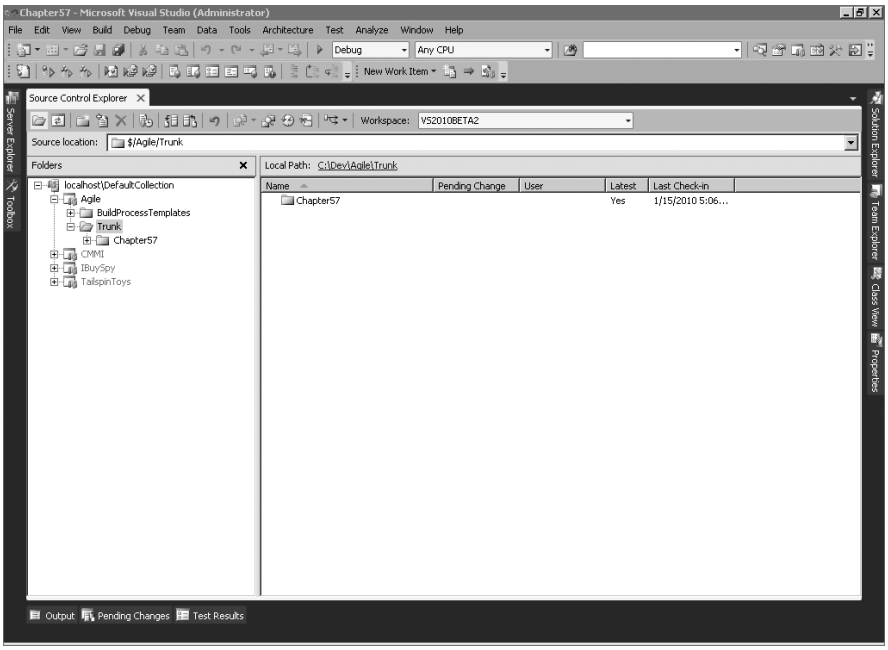


Рис. 57.11

Для работы с файлами на своей машине требуется *рабочее пространство* (workspace), которое определяет множество отображений между путями вашей локальной файловой системы и удаленной системы. Для каждого пути можно определить свою локальную папку, но лучше использовать только одно отображение. Это помогает хранить все решения и проекты вместе с помощью относительных путей даже в случае разных командных проектов. Для определения такого отображения откройте выпадающий список и выберите рабочее пространство.

После настройки рабочего пространства можете получить последнюю версию исходных текстов и начать работу с ними, добавляя файлы, помечая их как измененные, сохраняя внесенные изменения, просматривая историю изменений или сравнивая папки.



В предыдущих версиях TFS каждое рабочее пространство ограничивалось одним пользователем на одной машине. Система Visual Studio 2010 вводит концепцию общедоступного рабочего пространства, которое могут совместно применять несколько пользователей на одной и той же машине.

Работа с окном Solution Explorer

Создавая проект, можете добавить его в систему управления версиями. Система Team System автоматически связывает и добавляет его в соответствии с ранее определенным отображением. Вот почему необходимо настроить местоположение в папке в своем рабочем пространстве (локальный путь, на который выполняется отображение), как показано на рис. 57.12.

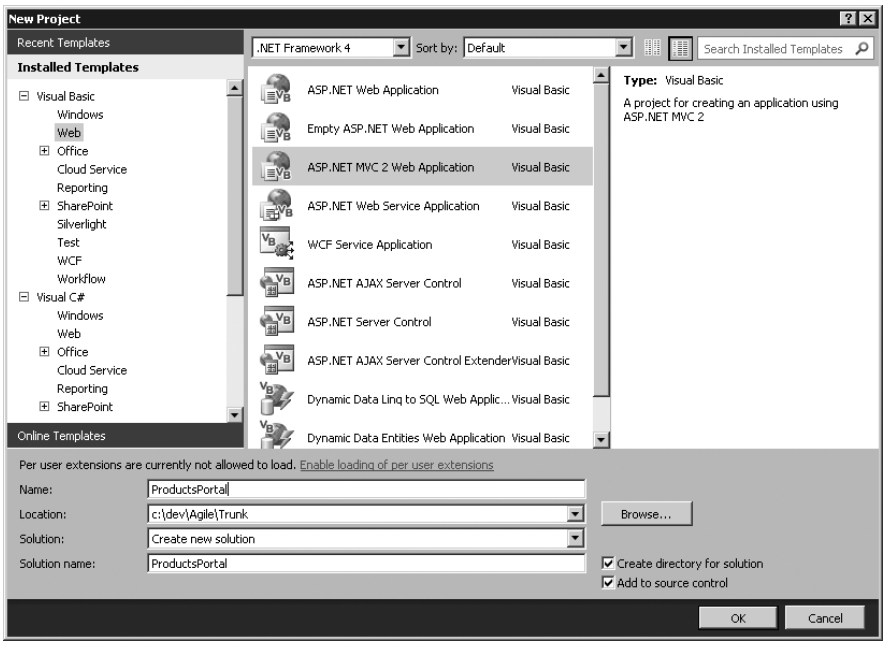


Рис. 57.12

Основным местом взаимодействия с системой управления версиями является окно Solution Explorer, с которым вы работаете в среде Visual Studio. Каждый раз при включении в решение нового файла он добавляется в систему управления версиями. Когда вы открываете файл и система Visual Studio обнаруживает, что вы его редактируете, файл автоматически маркируется как редактируемый. По окончании работы вы можете просто щелкнуть правой кнопкой мыши на решении и выбрать в контекстном меню Check In, чтобы сохранить свои изменения на сервере. Дополнительная информация по этому вопросу приведена в главе 13. Здесь же рассматривается специфика использования системы управления версиями при работе с системой Team Foundation Server 2010.



Хотя работа с системой управления версиями TFS из окна Solution Explorer – достаточно распространенное явление, оно обладает некоторыми недостатками, так как означает, что система Visual Studio в состоянии работать только с файлами, к которым обращается ваше решение. Если у вас есть другие управляемые элементы, не являющиеся частью решения, работать с ними следует в окне Source Control Explorer.

Извлечение файлов

Файлы под управлением версиями по умолчанию предназначаются только для чтения; в терминах TFS о них говорится, что они *записаны* (checked in). Чтобы начать работу с такими файлами, их следует отметить как редактируемые. Это делается автоматически при изменении их содержимого в интегрированной среде разработки Visual Studio. Если это текстовый файл (т.е. исходный текст C#, VB или XML-файл), интегрированная среда разработки отмечает файл как *совместно редактируемый* (shared check-out). Если же это бинарный файл (документ Word, SQL Server Compact Edition Database или какой-то иной ресурс), используется режим *исключительного редактирования* (exclusive check-out).

Совместное редактирование позволяет двум или более разработчикам одновременно редактировать такой файл, в то время как в исключительном режиме файл может изменяться только одним разработчиком. Исключительный режим можно назначить и для текстового файла, если его нужно защитить от модификации кем бы то ни было еще. Однако так поступать не рекомендуется, и использовать исключительный доступ следует только тогда, когда это действительно необходимо. Хорошим примером может служить обновление данных для ссылок WCF. Это текстовая информация, но поскольку должны быть обновлены одновременно многие файлы, слияние обновлений оказывается не таким простым. Исключительный режим гарантирует, что никто другой не будет модифицировать ссылки одновременно с вами.



При инсталляции системы TFS2010 Power Tools вы получаете возможность работать с файлами непосредственно из проводника Windows Explorer.

Запись файла

Чтобы сохранить внесенные изменения на сервере, нужно выполнить *запись* (check in) отредактированных файлов. Вы можете выбрать, какие файлы должны быть включены в *множество изменений* (changeset), добавить к ним комментарии, связать их с рабочими элементами и добавить примечания к регистрации (рис. 57.13).

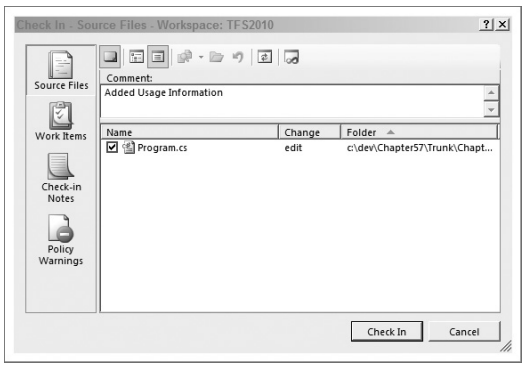


Рис. 57.13

В зависимости от определенных в командном проекте стратегий можно связать регистрацию с рабочим элементом, выполнить анализ кода, провести тесты или как минимум выполнить успешную сборку решения. Для изменения стратегий командного проекта откройте окно Source Control Settings (Team⇨Team Project Settings⇨Source Control) и перейдите на вкладку Check-in Policy. После того как ваши стратегии будут определены, вы можете получить предупреждение о несоответствии действий определенной стратегии Policy Warnings (рис. 57.14), которое может быть проигнорировано.

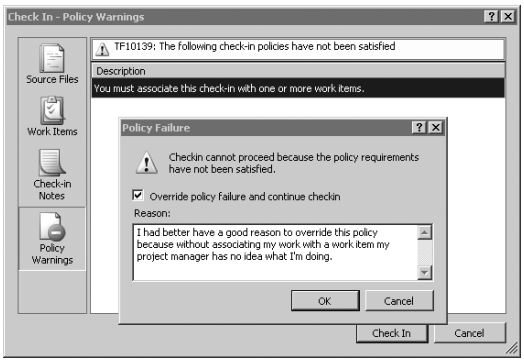


Рис. 57.14



Файлы, в которые были внесены связанные логически изменения, лучше регистрировать одновременно, а не по одному. Множество записанных файлов вместе со всеми примечаниями и соответствующими рабочими элементами образуют *множество изменений* (changeset). Множества изменений упрощают как управление историей проекта, так и выполнение слияний изменений.

Разрешение конфликтов

Хотя режим совместного редактирования позволяет нескольким разработчикам одновременно работать над одним и тем же файлом, это может привести к конфликтам. Их легко разрешить с помощью системы Visual Studio. В окне Pending Changes-Conflicts (рис. 57.15) можно сравнить версии и просмотреть все изменения, внесенные в файл. Для разрешения конфликтов можно воспользоваться функциональной возможностью Auto Merge и позволить системе Visual Studio выполнить слияние изменений вместо вас, отменить свои локальные изменения, отменить изменения на сервере или объединить изменения вручную.



Рис. 57.15

Если изменения внесены в разные части файла (например, в два разных метода), система Visual Studio автоматически разрешит изменения, но если модифицирована одна и та же строка кода, необходимо будет либо выбрать одну из версий, либо вручную объединить оба файла с помощью инструмента Merge Changes.



Система Visual Studio сравнивает тексты для определения перекрытия изменений, но это не гарантирует, что полученный в результате файл будет компилироваться или вести себя так, как ожидается. Возможность автоматического слияния полезна, но должна использоваться с осторожностью. Со временем вы приобретете достаточный опыт, чтобы разобраться, в каких файлах можно объединять изменения автоматически для экономии времени, а на какие нужно обратить особое внимание.

В инструменте Merge Changes (рис. 57.16) имеются три версии — "чужая" (т.е. версия на сервере), своя и объединенная. Вы можете легко перемещаться между изменениями и конфликтами. В случае конфликта можно вручную редактировать проблемные строки или выбирать версию для сохранения. Когда все конфликты будут разрешены, можете принять изменения, принять новый файл в качестве своей текущей версии и выполнить регистрацию изменений.

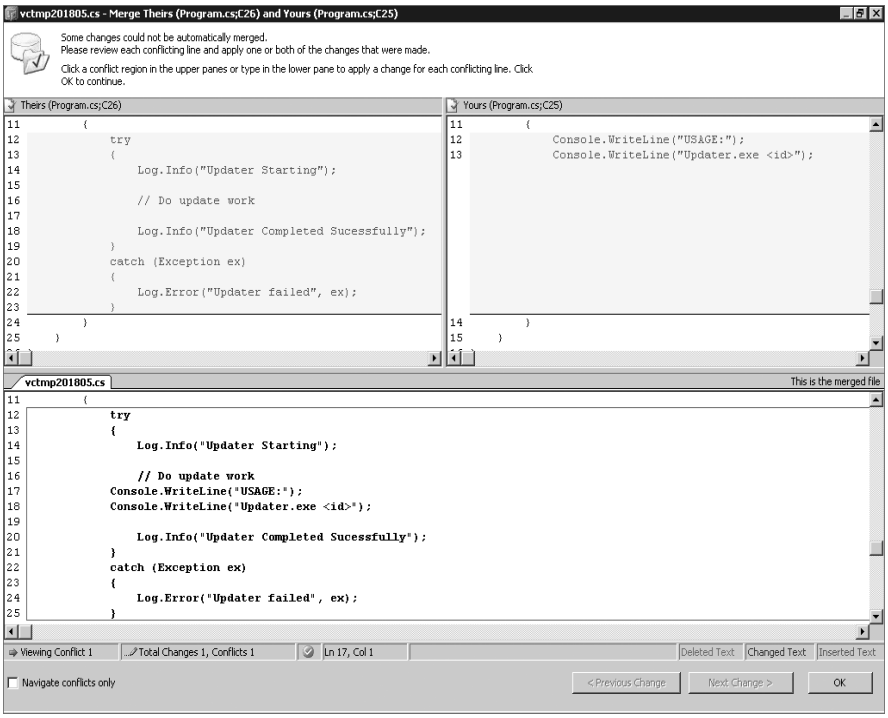


Рис. 57.16



После разрешения конфликтов рекомендуется выполнить автоматизированные тесты, чтобы убедиться, что вы не внесли никаких некорректных изменений. Как вы увидите позже, этот тест можно запускать на сервере автоматически перед каждой регистрацией, однако лучше получить результат проверок как можно раньше.

Работа в автономном режиме

Система Team Foundation Server использует протокол HTTP и веб-службы, так что он отлично работает через Интернет и обеспечивает совместную работу распределенных команд. Однако, если у вас нет доступных подключений, система Visual Studio позволяет работать в автономном режиме.

Все файлы под контролем системы управления версиями доступны только для чтения. Когда вы решите сохранить файл, появится предупреждение и вам нужно будет выбрать команду Overwrite. Когда соединение с системой TFS будет установлено заново, вы сможете подключиться из окна Solution Explorer, щелкнув правой кнопкой мыши на решении. Система Visual Studio выполнит поиск файлов решения без атрибута "только для чтения". Если этих файлов нет в окне Source Control, они будут добавлены; в противном случае они будут помечены как редактируемые.



Обнаружить изменения файлов, модифицированных вне решения, система Visual Studio не в состоянии, так что их нужно пометить вручную. Чтобы упростить эту задачу, предоставляется возможность сравнить свою локальную копию с последней версией, щелкнув правой кнопкой мыши на папке в окне Source Control Explorer.

Метки

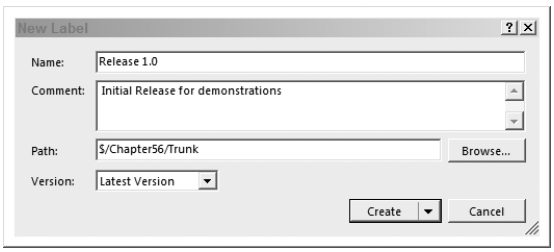


Рис. 57.17

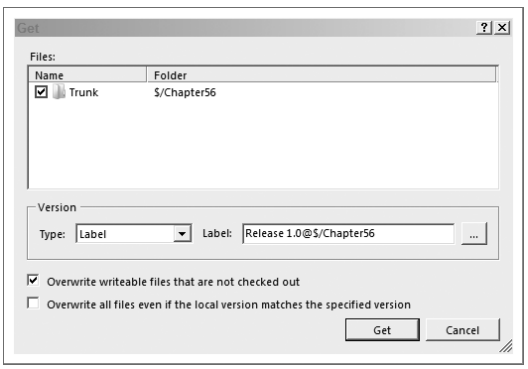


Рис. 57.18

Добавление *метки* упрощает ссылку на помеченную версию. Для создания метки просто щелкните правой кнопкой мыши в окне Source Control Explorer на папке, которую вы хотите маркировать, при необходимости добавьте дополнительные файлы и укажите имя (поле Name) и (необязательно) комментарий (поле Comment) (рис. 57.17). Аналогично можно обратиться к определенной версии с помощью метки. Это очень удобно при выпуске приложения.

Для получения помеченной версии щелкните правой кнопкой мыши на файле или папке в окне Source Control Explorer и выберите команду Get Specific Version из контекстного меню. В выпадающем списке Type окна Get (рис. 57.18) выберите пункт Label. Вы можете искать метки по имени, командному проекту и владельцу.

Для того чтобы разместить последнюю версию в другом месте, создайте ветвь. Об этом будет рассказано далее в данной главе.



Необходимо либо выполнить откат, либо зарегистрировать все незавершенные изменения перед тем, как обратиться к определенной версии кода, чтобы отделить последние изменения в своем рабочем пространстве от помеченной версии.

История

Каждое внесенное вами изменение сохраняется на сервере, и вы можете запросить любую версию файла. Лучше всего делать это в окне History. Просто щелкните правой кнопкой мыши на файле в окне Source Control или Solution Explorer и выберите в контекстном меню команду View History. Здесь вы можете проследить эволюцию файла со временем (рис. 57.19). Щелкнув правой кнопкой мыши на любой версии, вы можете сравнить ее со своей версией и открыть или просмотреть детальную информацию о множестве изменений (включая комментарии и связанные рабочие элементы).

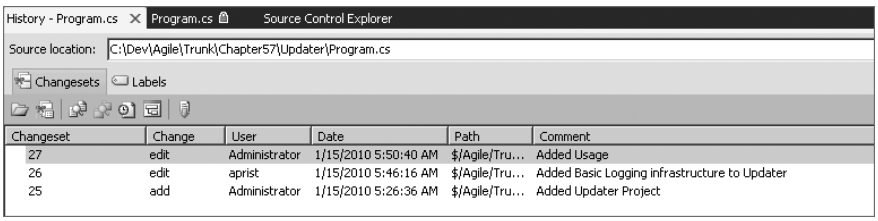


Рис. 57.19

В верхней половине рис. 57.19 имеется переключатель между вкладками Changesets и Labels. При переходе к представлению Labels, показанному на рис. 57.20, выводится список всех меток, которые назначались данному файлу. Это может быть полезно для быстрого просмотра отличий двух версий.

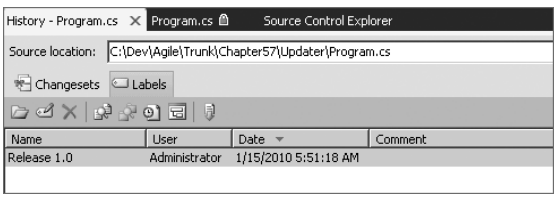


Рис. 57.20

Аннотация

Аннотация позволяет просмотреть, кто и когда редактировал каждую строку кода (рис. 57.21). Отталкиваясь от каждого сделанного изменения, можно получить его множество изменений для просмотра детальной информации, запросить конкретную версию, сравнить ее с предыдущей версией, найти файл в окне History или добавить аннотацию.

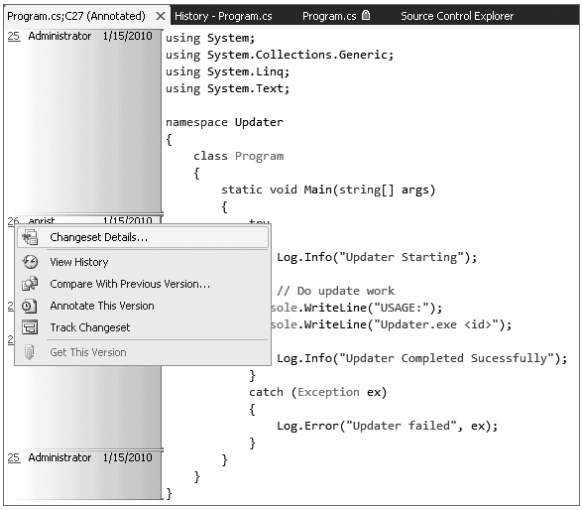


Рис. 57.21

Архивирование

Записанный файл автоматически становится доступным для других. Однако иногда нужно сохранить изменения так, чтобы они никому не были доступны, например при работе на чужом компьютере при желании сохранить изменения для дальнейшей работы или для передачи кому-то, или когда вы получаете новое срочное задание, не успев выполнить старое.

В процессе *архивирования* (shelving) ваши изменения сохраняются на сервере. Можете назначить *множество архивирования* (shelveset) рабочим элементам или добавить комментарии и примечания, как при обычной регистрации. Вы можете также (необязательно) применить стратегии регистрации перед архивированием и выбрать откат локальных изменений после нее. Последнее полезно в случае, когда вам необходимо работать над новым заданием, которому не должны мешать изменения. Архивирование изменений полезно и при переходе на другой компьютер, чтобы не выполнять одни и те же изменения на двух рабочих местах. Чтобы получить измененные файлы, вы выполняете *разархивирование* (unshelve) множества архивирования, и ваши файлы вновь передаются вам и отмечаются как редактируемые.

Каждое множество архивирования представляет собой мгновенный снимок файлов в момент его создания, доступный только для чтения. Из-за этого для множества архивирования управление версиями не поддерживается, так что при сохранении множества архивирования с тем же именем, которое уже имеется на сервере, вы просто переписываете имеющееся множество.



Множества архивирования уникальным образом идентифицируются комбинацией имени множества и имени создавшего его пользователя, так что, даже если ваша схема именования совпадает со схемой другого члена команды, вы не можете испортить работу друг друга.

Хотя по умолчанию вы можете просматривать только собственные множества архивирования, вы можете увидеть множества архивирования других людей и даже получить их. Это может быть полезным, если необходимо для того, чтобы ваш коллега ознакомился с вашим кодом еще до его записи, или для передачи работы другому разработчику, когда она еще не доведена до состояния, в котором ее можно регистрировать.

При разархивировании имеется возможность как удалить множество архивирования с сервера, так и оставить его нетронутым. Можно также удалить это множество, не получая его с сервера (соответствующая настройка имеется в диалоговом окне Unshelve). Лучше регулярно удалять ненужные множества архивирования с сервера хотя бы для того, чтобы упростить поиск используемых вами архиваций.

Ветвление

В терминах управления версиями *ветвь* (branch) представляет собой параллельную версию вашего кода. Эта возможность полезна в ряде ситуаций. Вот пара простейших примеров.

- Исправления или дополнения в текущей версии во время работы над новой. Выпустив версию 1.0, вы пометили все исходные тексты и приступили к работе над версией 2.0. Затем обнаружилась критическая ошибка, когда до выпуска версии 2.0 оставалось не менее пары месяцев. В этой ситуации вы можете создать ветвь из версии 1.0 (получить эту версию можно по ее метке), исправить ошибку в новой ветви и выпустить версию 1.1. Позже можно объединить сделанные изменения и интегрировать их в основную ветвь.
- Создание ветви из последней версии для выполнения большого рефакторинга или внесения серьезного изменения, полной уверенности в работоспособности которого нет, так что вы бы не хотели, чтобы эти изменения затронули основную ветвь. Если все прошло успешно, можно выполнить слияние с основной ветвью, а если нет — просто удалить неудачную ветвь.

Следует разумно подходить к выбору стратегии ветвления, наилучшей для вашей организации, типа продукта и процесса. В ряде случаев можно ограничиться меткой версии или архивацией изменения. Злоупотребление ветвлением может экспоненциально усложнить управление исходными текстами. На сайте Codeplex можно найти руководство с дополнительными сценариями, связанными с ветвлением (`http://www.codeplex.com/TFSBranchingGuideII`).

Для создания ветви щелкните правой кнопкой мыши на папке, от которой хотите отвести ветвь, и выберите в контекстном меню команду Branching and Merging⇨ Branch. Вам будет предложено указать версию, от которой следует выполнить ветвление, и где в репозитории управления версиями она должна храниться.



Ветви стали очень важной возможностью системы TFS 2010, так что имеется соответствующий инструментарий для того, чтобы воспользоваться всеми предоставляемыми ими преимуществами. Вы можете пометить существующую папку как ветвь, указав необходимые метаданные, так что ветви смогут расти из любых более ранних версий исходного текста.

Если ветвей несколько, используйте для просмотра их иерархии инструмент Branch Visualization (рис. 57.22), выбирая пункт View Hierarchy из выпадающего спи-

ска Branching and Merging в окне Source Control Explorer. В этом инструменте вы также можете инициировать слияние путем перетаскивания множества изменений из одной ветви в другую.

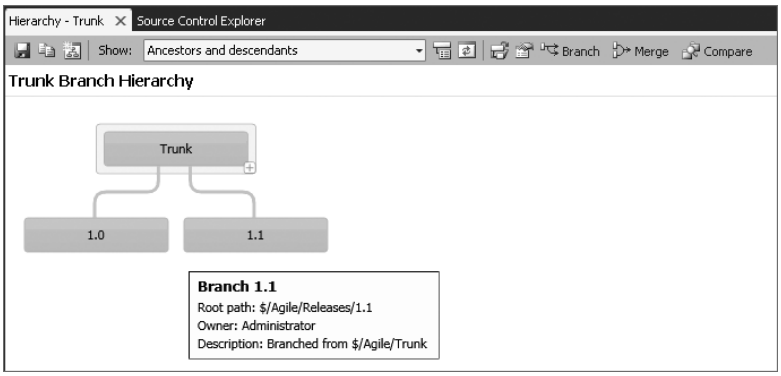


Рис. 57.22

Еще одним новым инструментом системы TFS 2010 является Changeset Tracking, позволяющий увидеть источник изменений в конкретном множестве изменений. В нем имеются два представления: Timeline Tracking (рис. 57.23) и Hierarchy Tracking, который показывает иерархию ветвей в более привлекательном и понятном виде. Как и в случае представления Branch Visualization, инициировать слияние можно путем перетаскивания множества изменений из одной ветви в другую.

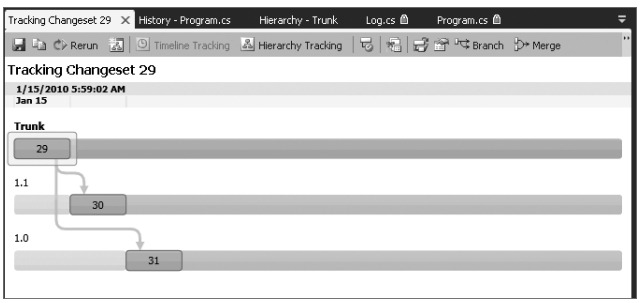


Рис. 57.23

Слияние

Если вы внесли усовершенствование или исправили ошибку в одной ветви, было бы хорошо иметь возможность применить эти изменения и к другим ветвям. Именно этим и занимается операция *слияния* (merge). Чтобы начать слияние, щелкните правой кнопкой мыши на папке или файле, для которого вы хотите выполнить слияние, и выберите команду Branching and Merging⇨Merge контекстного меню. После выбора источника и приемника слияния вам будет представлен список множеств изменений, которые могут быть использованы. Выберите интересующие вас множества изменений и щелкните на кнопке Finish. В случае каких-либо конфликтов вам будет предоставлена возможность их разрешения.

Инструмент Team Foundation Build

Программа Team Foundation Build представляет собой инструмент, являющийся частью системы TFS и позволяющий увидеть в рабочем пространстве последнюю версию от системы управления версиями, сборку проектов, запуск тестов, выполнение других задач, а в конечном итоге — отчет о результатах и вывод в общую папку. Каждая машина, способная собрать для вас проект, называется *агентом сборки* (build agent). TFS 2010 вводит также концепцию *контроллера сборки* (build controller), который отвечает за координацию действий нескольких агентов сборки. Информация, необходимая агентам сборки для выполнения их работы, называется *определением сборки* (build definition).

Для создания определения сборки щелкните правой кнопкой мыши на папке Builds в окне Team Explorer и выберите из контекстного меню команду New Build Definition (рис. 57.24). На вкладке General укажите имя сборки и (необязательно) описание.

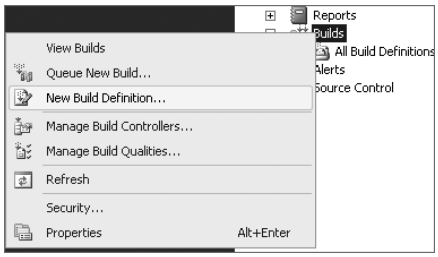


Рис. 57.24

По умолчанию сборки ставятся в очередь вручную, но на вкладке Trigger это поведение можно изменить. Существует пять вариантов, показанных на рис. 57.25. Новый вариант, появившийся в системе TFS 2010, — Gated Check-in. При наличии определений сборок, запускаемых таким образом, регистрации необязательно поступают непосредственно в систему управления версиями. Вместо этого они архивируются (вы получите соответствующее приглашение), после чего агент сборки выполняет свою работу. Если сборка выполнена успешно, множество архивирования вносится в репозиторий системы управления версиями. В противном случае вы уведомляетесь о происшедшем и можете запросить множество архивирования, внести необходимые изменения и вновь попытаться выполнить регистрацию. Использование определения сборки этого типа предупреждает ситуацию, когда содержимое репозитория не компилируется, что может крайне отрицательно сказаться на остальных членах команды.

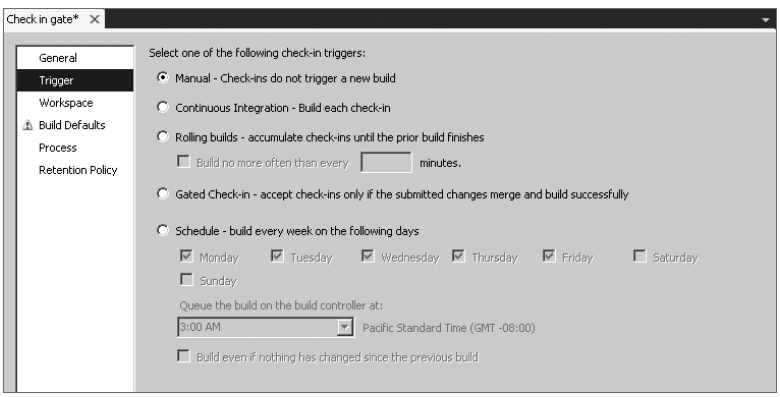


Рис. 57.25

В зависимости от размера вашего проекта, времени, необходимого на выполнение сборки и проведение тестов, и от того, насколько часто ваша команда регистрирует изменения, такое поведение может приводить к дополнительным накладным расходам. Третий вариант, Rolling Builds, позволяет снизить нагрузку, но с его применением лучше не спешить.

Настройка рабочего пространства используется в сложных сценариях с зависимостями между командными проектами, так что может вполне хватить настроек по умолчанию на этой вкладке. На странице Build Defaults можно выбрать контроллер сборки и общедоступную папку для вывода. Они используются для автоматических сборок, но при сборке вручную эти параметры могут быть переопределены.

Показанная на рис. 57.26 вкладка Process позволяет настраивать процесс сборки. Здесь необходимо выбрать как минимум один проект или решение и конфигурацию (наподобие `x86|Debug` или `AnyCPU|Release`). Остальные значения необязательны, но включают информацию, позволяющую указать местоположение сборок автоматизированных тестов, следует ли выполнять анализ кода, формат номеров сборок и многое другое.

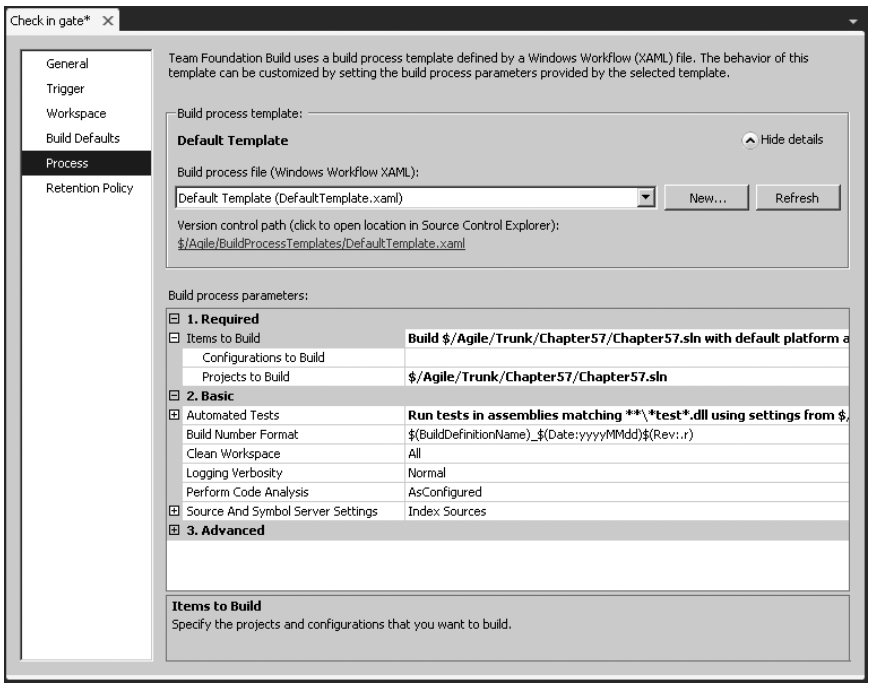


Рис. 57.26

Процесс сборки в системе TFS 2010 основан на шаблоне процесса, определенном с помощью технологии Windows Workflow 4.0. Вы можете создать собственный шаблон процесса, скопировав шаблон процесса по умолчанию и внеся в него изменения. Ряд пользовательских действий связан с инструментом Team Build, и всегда можно создать собственные действия. Если у вас есть определение сборки из предыдущей версии Team Build, используйте Upgrade Template, все требования которого сводятся к указанию пути к старому файлу `TfsBuild.proj`.



По умолчанию шаблоны процессов находятся в папке `$/TeamProjectName/BuildProcessTemplates` репозитория системы управления версиями.

Политика сохранения позволяет выбрать количество сборок, которое может храниться в общедоступной папке до того, как некоторые из них начнут удаляться. Рекомендуется использовать настройку Keep All как минимум для успешных сборок, если только вы не ограничены в количестве хранимых файлов.

Для начала сборки вручную можно щелкнуть правой кнопкой мыши на узле Builds в Team Explorer и выбрать из контекстного меню команду Queue New Build. После того как сборка будет поставлена в очередь, можете открыть ее, дважды щелкнув на ней в окне Build Explorer. После этого откроется отчет Build Report, существенно улучшенный по сравнению с предыдущей версией. Этот отчет, показанный на рис. 57.27, включает информацию о текущих действиях, состоянии предыдущих сборок и их продолжительности, а также предоставляет ссылки на ряд других элементов, связанных с данной сборкой.

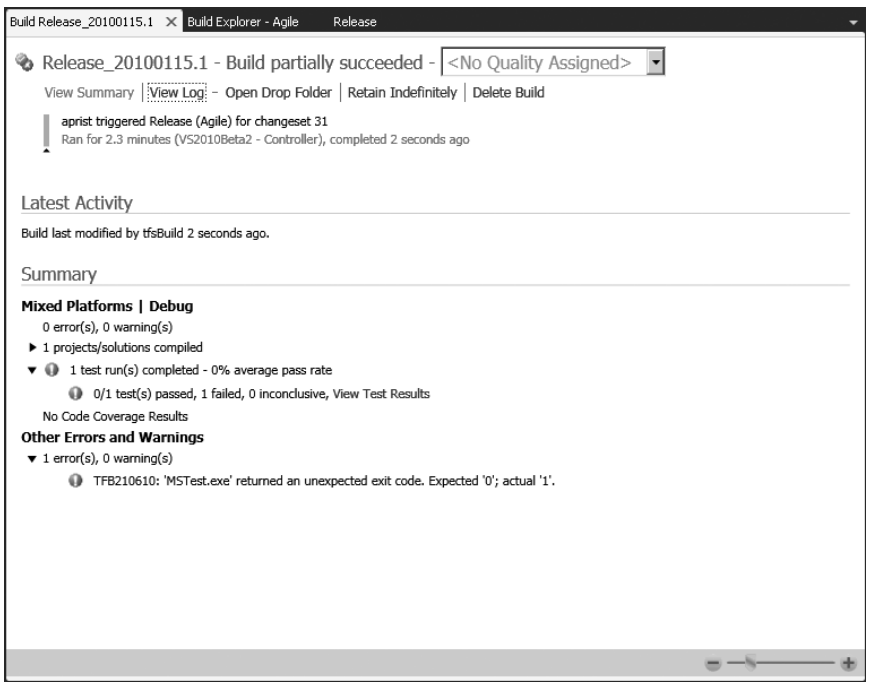


Рис. 57.27



Для того чтобы получать уведомления о событиях сборки, воспользуйтесь работающим в области уведомлений приложением Build Notifications system, которое устанавливается вместе с инструментом Team Explorer. Будучи настроенным, это приложение изменяет внешний вид пиктограммы в зависимости от состояния сборки.

Отчеты и интеллектуальные ресурсы

Для предоставления полезной информации о менеджерах, пользователях и разработчиках проекта система TFS использует инструмент Report Server, который представляет собой часть Microsoft SQL Server. К отчетам можно обратиться непосредственно из программы Team Explorer, через сайт отчетов (`http://mytfs/reports/`) и с помощью коллекции SharePoint или веб-интерфейса Team System Web Access. Кроме того, их можно настроить в качестве подписки на сайте отчетов для рассылки посредством электронной почты.

Большим преимуществом является то, что разработчики теперь могут сосредоточиться на своей работе, а не на составлении отчетности вручную, а менеджеры проектов и руководители организаций могут получать оперативную информацию о состоянии дел от провайдеров отчетов TFS.

Каждый шаблон процесса предоставляет собственное множество отчетов. Версия CMMI предоставляет три дополнительных отчета и шаблона наподобие Scrum for Team System from Conchango, и содержит отчеты, пригодные для методологии Scrum, такие как Delta Report и Burndown Charts. Здесь же мы опять сосредоточимся на версии MSF for Agile Development.

Имеющиеся отчеты включают Burndown and Burn Rate, Stories Progress, Build Success Over Time, Build Quality Indicators, Test Case Readiness, Test Plan Progress, Bug Status, Bug Trends и Reactivations. На рис. 57.28 продемонстрирован ход работ за несколько месяцев и объем оставшихся работ. Отчеты могут фильтроваться по датам, типу рабочих элементов, итерациям и т.д. Их можно экспортировать в форматах XML, CSV, TIFF, PDF, Web и Excel.

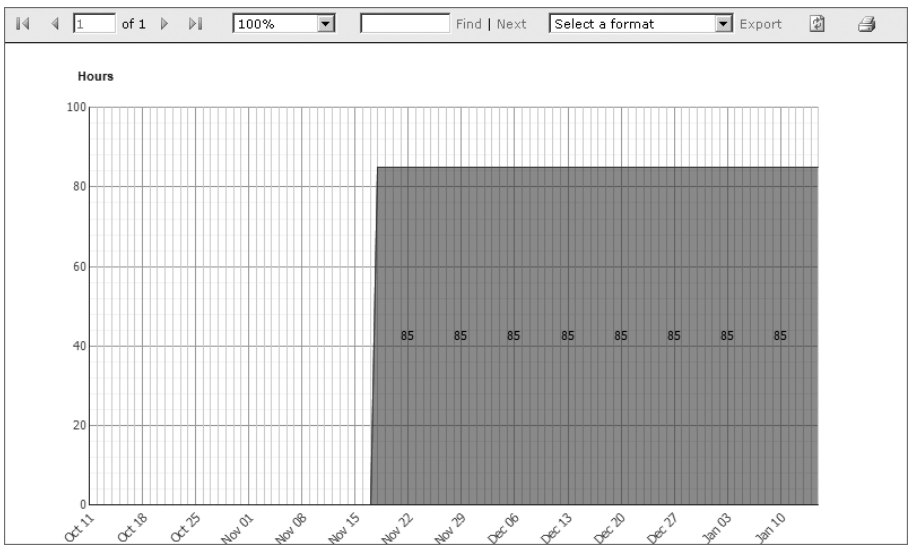


Рис. 57.28

Зачастую для получения пользы от отчета вовсе не нужна информация за целые годы. В зависимости от природы выводимых данных вы можете вообще ничего не увидеть на начальной стадии выполнения проекта. Например, отчет Test Failing (неудачные тесты) требует проведения по меньшей мере одного теста, а для отчета

Regressions необходимо успешное прохождение тестов, которые в настоящее время пройти не удается. Аналогично отчет Scenarios Details требует как минимум регистрации рабочего элемента Scenario Work Item. После пары итераций работы с системой TFS у вас наберется достаточное количество полезных метрик.



Компания Microsoft ясно указывает, что не нужно обращаться к данным из баз данных TFS непосредственно. Вместо этого следует использовать отчеты и предоставляемые инструменты. В системе TFS 2010 имеются новые представления Data Warehouse Views, добавленные в каждую базу данных TFS, которые гарантированно не будут изменяться в дальнейшем и разработаны специально для того, чтобы вы могли создавать собственные отчеты.

Командный портал

Система Team Foundation использует коллекцию SharePoint для создания портала для каждого командного проекта. Он применяет все преимущества коллекции SharePoint, но настраивается для каждого шаблона процесса отдельно. Начальная страница каждого командного портала включает наиболее важные отчеты, последние новости и полезные ссылки. Система TFS 2010 включает также возможность создания пользовательских инструментальных панелей, которые могут предназначаться для конкретных пользователей или для всех членов проекта. Для перехода к порталу щелкните правой кнопкой мыши на командном проекте в окне Team Explorer и выберите в контекстном меню команду Show Project Portal.

Документация

В зависимости от шаблона процесса некоторые документы могут быть включены как шаблоны. Например, проект MSF for Agile Software Development содержит документы Word для создания персоналий и сценариев. Эти документы доступны также из папки документов `Team Explorer`.

Управление процессом

В коллекции SharePoint имеются документы, определяющие процесс, которого придерживается ваш проект. Это руководство доступно всем разработчикам команды.

Списки коллекции SharePoint

У вас могут быть библиотеки изображений, дискуссионные группы, анкеты, объявления, ссылки и многое другое, способствующее сотрудничеству внутри команды.

Инструментальные панели

Инструментальные панели (dashboards) представляют собой страницы SharePoint Web Part, предназначенные для того, чтобы предоставлять обзор состояния проекта. Шаблон процесса MSF for Agile определяет две панели, включая показанную на рис. 57.29 панель My Dashboard, содержащую списки назначенных вам задач, ошибок и контрольных примеров, а также панель Project Dashboard, которая содержит метрики

и информацию о достижениях всей команды. Используя интерфейс SharePoint UI, можно копировать существующие панели и вносить изменения в выводимые части.

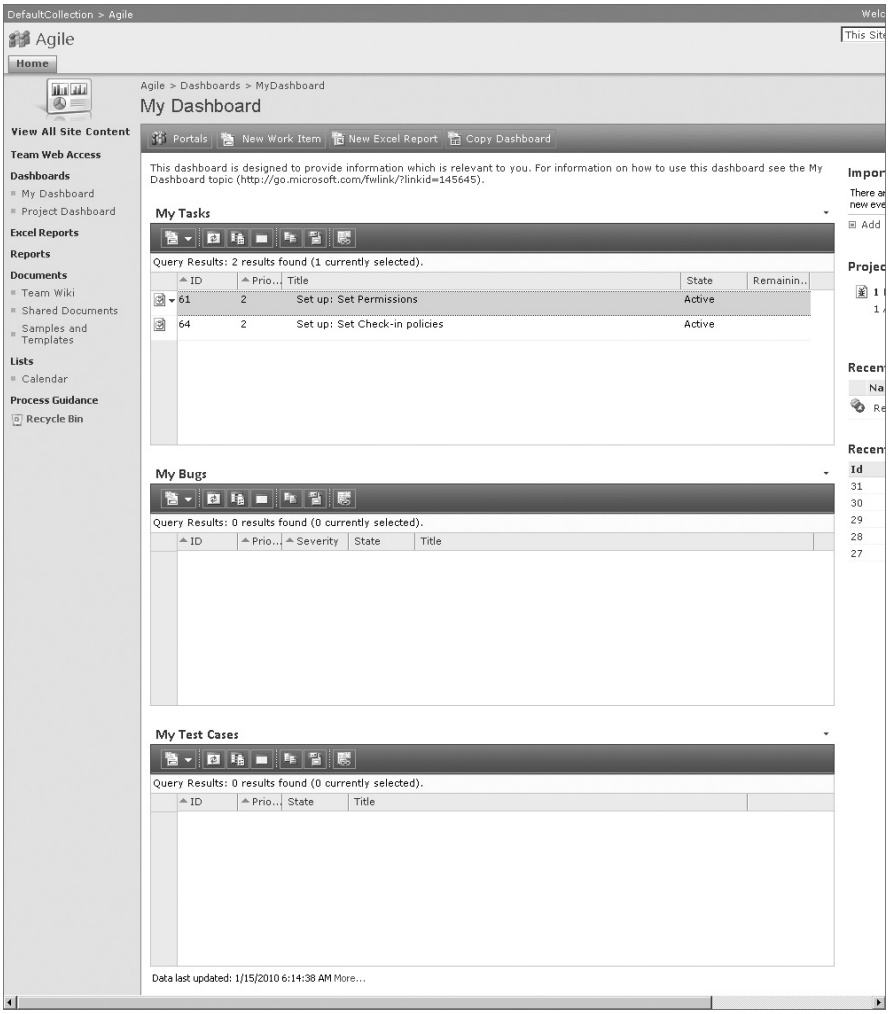


Рис. 57.29

Инструмент Team System Web Access

Это отдельный бесплатный инструмент от Microsoft, который интегрируется с системой TFS и позволяет делать почти то же самое, что можно делать из системы Visual Studio, но с применением веб-интерфейса (рис. 57.30). Вы можете создавать и модифицировать рабочие элементы и запросы, просматривать отчеты и документы, инициировать и отслеживать сборки. Единственной областью с ограниченной функциональностью является управление версиями. Вы можете просматривать историю или детальную информацию о множествах изменений, но не можете ни помечать документы как редактируемые, ни регистрировать изменения в силу природы веб-клиента.

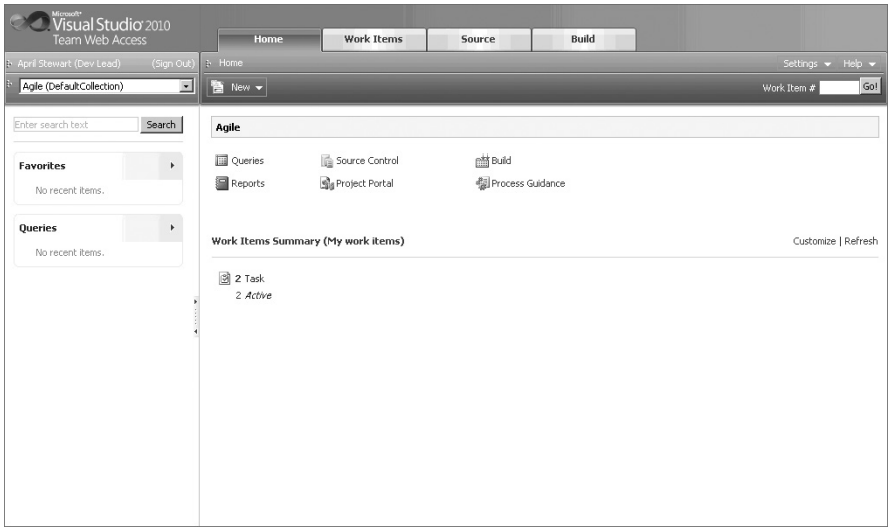


Рис. 57.30

Администрирование TFS

Отслеживание всех настроек, веб-сайтов, баз данных и учетных записей, требуемых TFS, может быть очень трудным делом. Инструмент Team Foundation Server Administration Console (рис. 57.31) агрегирует массу информации о вашей инсталляции системы TFS в одном месте. Он также предоставляет возможность решения многих задач, включая управление коллекциями проектов и настройку контроллеров и агентов сборок.

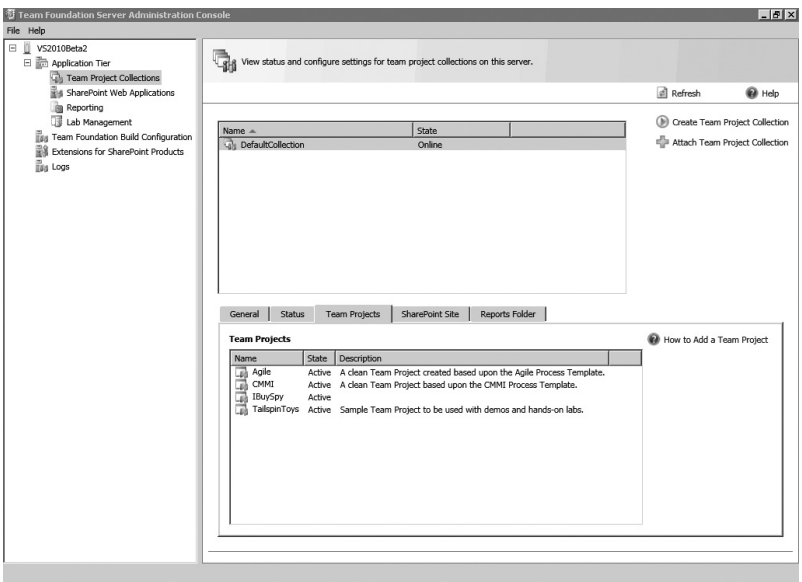


Рис. 57.31

Автоматизация TFS и настройка процессов

Из этой главы вы узнали, как шаблоны процессов определяют бо́льшую часть поведения системы TFS; например, они определяют типы рабочих элементов, руководство процессом и отчеты. Чтобы завершить данную главу, бегло рассмотрим настройку шаблона процесса и редактирование типов рабочих элементов.

Чтобы выполнить редактирование типов рабочих элементов или шаблонов процессов, можно модифицировать определяющий их XML-код, но проще воспользоваться редактором Process Editor, который представляет собой часть пакета Team Foundation Server Power Tools, загружаемую с сайта Microsoft.

Типы рабочих элементов

С помощью меню Tools⇨Process Editor⇨Work Item Types можно открыть тип рабочего элемента непосредственно из существующего командного проекта и приступить к его редактированию. При этом все внесенные вами изменения будут немедленно распространены на всех пользователей. Другой подход заключается в экспорте типа рабочего элемента в файл, после чего этот файл открывается для редактирования, а затем вновь импортируется в командный проект. Все эти варианты расположены в одном и том же подменю.



Рекомендуется иметь отдельный командный проект для всестороннего тестирования вносимых изменений перед тем, как импортировать их в активный командный проект (или редактировать в нем типы рабочих элементов).

После того как вы отредактировали тип рабочего элемента (непосредственно из командного проекта или из файла), у вас появились три вкладки, с которыми вы можете работать: Fields, Layout и Workflow (рис. 57.32).

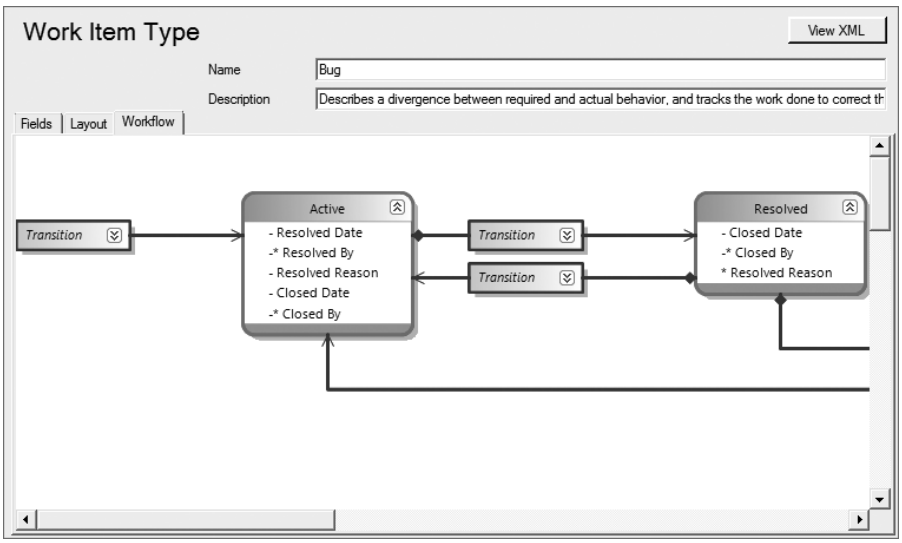


Рис. 57.32

На первой вкладке содержится список полей с типами данных и именами, использующимися для однозначной идентификации полей из различных типов рабочих элементов. Например, заголовок для ошибки и заголовок для задачи имеют одно и то же имя "`System.Title`"; другие поля также могут иметь одинаковые имена, но означать совершенно разные вещи.

Вторая вкладка позволяет модифицировать компоновку. Здесь у вас нет визуального проектировщика, но вы можете работать в проектировщике деревьев для создания вложенных групп, столбцов и управляющих элементов. Каждый управляющий элемент отображается на поле с использованием его имени. Имеется также кнопка Preview Form, которая позволяет увидеть конечный продукт.

На третьей вкладке можно модифицировать последовательность действий, а панель инструментов позволяет добавлять состояния и переходы. На рис. 57.32 показано, как может осуществляться переход ошибки из состояния активной в состояние исправленной. Дважды щелкнув на переходе, можно определить причины, действия и поля. Последние используются для установки значений по умолчанию для полей после перехода; например, после перехода ошибки из активной в исправленную поле `Resolved By` получает в качестве значения текущего пользователя, а поле `Resolved Date` устанавливается равным текущей дате на сервере.

В следующем разделе мы рассмотрим включение типов рабочих элементов в шаблон процесса.

Настройка шаблона процесса

В качестве примера настройки шаблона процесса модифицируем шаблон MSF for Agile Development. Для создания локальной копии откройте окно Process Template Manager с помощью пункта меню Team⇨Team Project Collection Settings и выберите команду Process Template Manager. Когда откроется диалоговое окно Process Template Manager, выберите шаблон и щелкните на кнопке Download. Теперь можете открыть его в редакторе Process Template Editor, воспользовавшись командой меню Tool⇨Process Editor⇨Process Template⇨Open Process Template и выбрав только что загруженный файл `processtemplate.XML`.

В окне редактора Process Template Editor (рис. 57.33) имеется представление в виде дерева, предназначенное для настройки разных областей командного проекта. В узле Work Item Tracking можно модифицировать Default Work Item List, создать запросы рабочего элемента по умолчанию и импортировать типы рабочих элементов. В узле Areas & Iteration можно настроить значения по умолчанию для этих списков и указать отображение полей и столбцов для инструмента Microsoft Project Integration. В узле Source Control можно указать соответствующие права и выполнить настройки. Узел Portal позволяет добавить документы в библиотеки документов из коллекции SharePoint, и наконец в узле Reports можно добавить файлы определения Report Definition.

Хотя этот инструмент может существенно упростить работу, для решения некоторых задач необходимо отредактировать XML-файлы. Например, описанный инструмент не позволяет указать шаблон SharePoint для портала.

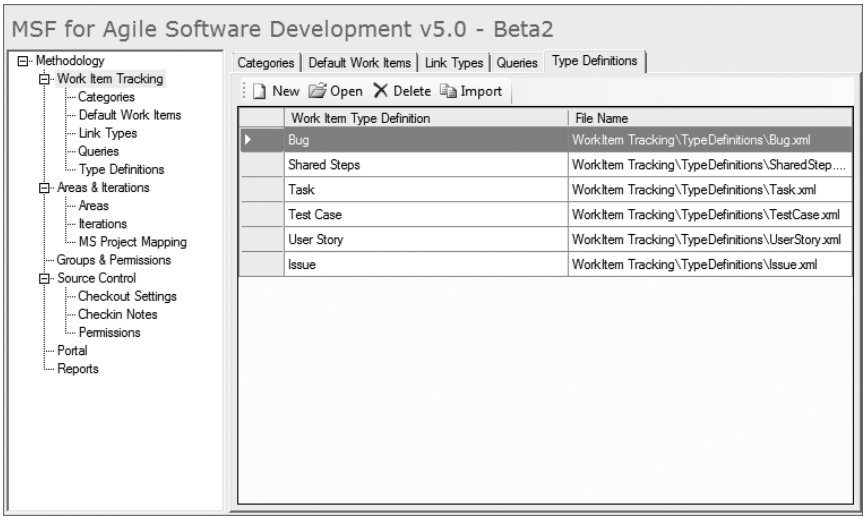


Рис. 57.33

Резюме

Из этой главы вы узнали, как система Team Foundation Server может помочь в работе, которая требует интеграции различных ролей. Менеджеры проекта могут отслеживать рабочие элементы с помощью программы Excel или Project, в то время как архитекторы, разработчики и тестировщики работают с проектами системы Visual Studio, используя возможности системы по управлению версиями и легко связывая изменения кода с назначенными рабочими элементами. Каждое изменение может при необходимости запускать сборку, что обеспечивает поддержание высокого качества проекта. TFS отслеживает все происходящее в проекте и генерирует метрики для отчетов, которые могут быть просмотрены с использованием самых разных интерфейсов, таких как Visual Studio, Team Portal и Team System Web Access. В конце главы речь шла о том, как настроить весь процесс в целом, модифицируя шаблоны процессов и типы рабочих элементов для большего соответствия требованиям вашей организации.

Предметный указатель

X

XML-комментарий, 264

А

Авторизация
  ролевая, 769
Агент сборки, 1157
Агрегатор
  классификаторов, 1072
Адрес, 728
Аннотация, 1153
Аргумент
  входящий, 751
  входящий и исходящий, 751
  исходящий, 751
  свойства, 751
Архивирование, 1154
Архитектура
  Model-View-Controller (MVC), 481
  ObjectSpaces, 664
Ассоциация, 225
  классов, 225
  коллекции, 225
Атрибут
  отладки, 893
Аутентификация, 461; 771
  на основе форм, 771

Б

Безопасность типов, 352
Библиотека
  ASP.NET AJAX Control Toolkit, 474
  Code Snippets Manager, 187
  Image Library, 342
  LINQ, 354
  Managed Extensibility Framework (MEF), 1060
  MSBuild, 960
  MSBuildTasks, 960
  Parallel LINQ, 874
  Task Parallel Library, 872
  WCF Service Library, 732
  документов, 561
Брокер, 1075

В

Веб-приложение, 905
Веб-часть, 562
Ветвление, 1155
Ветвь, 1155
Визуализатор, 899
Выделение блока, 102
Выражение, 745
  пустое, 645
  регулярное, 118

Г

Генератор кода
  Preprocessed Text Templates, 325
Генератор T4, 305
  блок, 309
    Class Feature, 312
  директива, 307
    assembly, 317
    import, 318
    include, 319
    output, 317

template, 316
ошибка
Compiling Transformation, 320
Executing Transformation, 321
Генерация кода, 305
Глобальный кэш сборок, 966
Граф
зависимостей, 1097

Д

Действие, 743
составное, 743
управления потоком, 744
Декомпилятор, 971
Делегат, 225
Дескриптор, 1073
XML, 267
XML-комментария, 267
<c>, 267
<code>, 267
<example>, 268
<exception>, 269
<include>, 270
<list>, 271
<para>, 272
<param>, 273
<paramref>, 273
<permission>, 274
<remarks>, 274
<returns>, 275
<see>, 275
<seealso>, 276
<summary>, 277
<typeparam>, 277
<typeparamref>, 278
<value>, 278
Диаграмма
Activity Diagram, 1089
Class Diagram, 1092
Component Diagram, 1091
Layer Diagram, 1093
Sequence Diagram, 1090
Use Case Diagram, 1089
базы данных, 599
Дизассемблер, 970
Директива
ссылок, 174
Документация, 264

З

Зависимость развертывания, 356
Закладка, 745
Закрепляемое, 83
Запись журнала, 579
Запрос рабочего элемента, 1141

И

Импорт, 1062
Имя
строгое, 965
Инсталлятор, 992
веб-проекта, 1021
службы, 1003
Инструмент
Add-in Manager, 1043
Code Coverage, 1129
Copy Web Site, 1015
Exception Assistant, 876
Health Monitoring, 917
IntelliTrace, 871; 1110
Microsoft Test and Lab Manager, 1110
MSDeploy, 1016
Team Foundation Build, 1157
Team System Web Access, 1162
Test and Lab Manager, 1133
Test Impact Analysis, 1132
Test Load Agent, 1126
Test and Lab Manager, 1121
Test Impact Analysis, 1130
Web Deployment Tool, 1015; 1016
Инструмент CSS, 455
Интеллектуальный дескриптор, 386; 423
Интерфейс, 225
ISignatureHelpSourceProvider, 1075
Development Tools Extensibility, 1053
IClassificationAggregatorService, 1072
IClassifierAggregatorService, 1076
ICompletionBroker, 1076
ICompletionSource, 1075
ICompletionSourceProvider, 1075

IContentTypeRegistryService, 1072; 1076
IDropHandler, 1074
IDropHandlerProvider, 1074
IDTCommandTarget, 1045
IDTExtensibility2, 1045
IIntellisenseController, 1075
IIntellisenseControllerProvider, 1076
IMouseProcessorProvider, 1074
IQuickInfoBroker, 1076
IQuickInfoSource, 1075
IQuickInfoSourceProvider, 1075
ISignatureHelpBroker, 1076
ISignatureHelpSource, 1075
ISmartTagBroker, 1076
ISmartTagSource, 1075
ISmartTagSourceProvider, 1075
ITagger, 1073
ITaggerProvider, 1073
ITextBuffer, 1068; 1075
ITextDocumentFactoryService, 1076
ITextEdit, 1068
ITextSearchService, 1076
ITextView, 1069
ITextViewLine, 1069
IWpfKeyboardTrackingService, 1076
IWpfTextViewCreationListener, 1074
IWpfTextViewHost, 1069
IWpfTextViewMargin, 1073
IWpfTextViewMarginProvider, 1073
REST, 580
Source Code Control, 290
Инфраструктура приложения, 768
История, 1153
Источник, 1075

К

Кадр стека, 868
Каталог, 1066
AggregateCatalog, 1067
AssemblyCatalog, 1066
DirectoryCatalog, 1066
TypeCatalog, 1066
Класс, 224
ITextDocument, 1068
ITextSnapshot, 1068
абстрактный, 225
доступа, 259
метаданных, 536
разделяемый, 330
Классификатор, 1069
Ковариантность, 358
Код
внешний, 896
Коллекция
Features, 561
SharePoint, 555
командных проектов, 1139
Команда, 104; 1036
Database File, 595
Find and Replace, 120
Find in Files, 120
Find Symbol, 123
Incremental Search, 125
Insert Snippet, 184
Navigate To, 124
Quick Find, 116
Quick Info, 172
Quick Replace, 117
Replace in Files, 122
Text Template, 306
View History, 297
Комментарий, 102
Комплект
Starter Kit, 349
Компонент
BindingNavigator, 614
BindingSource, 612
DataColumn, 605
DataRaw, 605
DataSet, 605
DataTable, 605
MEF, 1032
Web Service Data Source, 628
Компоновка, 227
Контейнер композиции, 1061
Контекст домена, 802
Контравариантность, 360
Контракт, 728; 1064
данных, 723
ожидаемый, 726
формальный, 726
службы, 723

сообщений, 723
Контроллер, 481; 1075
Контроллер сборки, 1157
Контрольная точка, 51
Концентратор, 1098
Культура
инвариантная, 850
конкретная, 849
нейтральная, 849

Л

Лексема
комментария, 285
Ленивое вычисление, 372
Лист рассылки, 561

М

Макрорегистратор, 1055
Макрос, 1032; 1053
Манипулятор поля, 453
Маркер свойства, 407
Маршрутизатор, 495
Мастер
Add-in Wizard, 1039
Performance Wizard, 1106
Setup Wizard, 992
Метка, 1152
Метод
анонимный, 361
действия, 484
лямбда, 361
Метрика
Cyclomatic Complexity, 1104
Depth of Inheritance, 1103
Lines of Code, 1103
Maintainability Index, 1104
Механизм
ASP.NET Dynamic Data, 529
List Comprehensions, 371
Navigation Framework, 522
Pattern Matching, 371
Text Template Transformation, 325
Visual State Manager, 523
генерации отчетов, 688
локальный, 689
серверный, 690
объектно-реляционного отображения, 663
Многофункциональное интернет-приложение, 515
Множество
архивирования, 1154
Множество изменений, 1149
Модель, 481; 1088
Copy-Modify-Merge, 293
DTE, 1033
Lock-Modify-Unlock, 293
автоматизации, 1032
концептуальная, 665
отражения, 979
сущностная, 665
хранения, 665

Н

Набор сущностей, 665
Навигатор
по дескрипторам, 451
Надстройка, 423; 1032; 1039
Class Designer, 221
Code Contracts, 302
CodeRush Xpress, 193
FxCop, 299
GhostDoc, 264; 281
Modeling Power Toys, 232
Sandcastle, 265
StyleCop, 302
Visual Basic PowerPacks, 365
Наследование, 225
Настройка
Before Building, 946
Compilation Constants, 951
DLL Base Address, 951
Enable Optimizations, 951
For New Solutions Use the Currently Selected Project as the Startup Project, 947
Generate Debug Info, 951
Maximum Number of Parallel Project Builds, 947
MSBuild Project Build Output Verbosity, 948

MSBuild Project Log File Verbosity, 948
On Run, When Build or Deployment Errors Occur, 947
On Run, When Projects Are Out of Date, 947
Only Build Startup Projects and Dependencies on Run, 947
Remove Integer Overflow Checks, 951
документа, 423
редактора, 1074
Нумерация строк, 99

O

Область, 501
просмотра, 1069
Область видимости, 743
Область проектирования, 223
Облачные вычисления, 576
Обобщенная вариантность, 357
Обработка
исключений, 909
ошибок, 909
Обработчик
перетаскивания, 1074
Обрамление, 1073
Объект
POCO, 797
Объектно-реляционное
отображение, 663
Окно
Add New Item, 306
Apply Styles, 456
Autos, 867
Breakpoints, 863
Call Hierarchy, 109
Call Stack, 867
Class Details, 225
Class View, 106
Code Definition, 108
Code Metrics, 1102
Command, 103; 864
CSS Properties, 457
Data Link Properties, 607
Data Sources, 600; 608
Disassembly, 870
Document Outline, 110; 395
Error List, 107; 320
Exception Assistant, 874
Exceptions, 875
History, 297
Immediate, 105; 864
IntelliTrace Events, 871
Lab Center, 1135
Locals, 867
Macro Explorer, 1054
Manage Styles, 456
Mapping Details, 671
Memory, 870
Modules, 868
New Project, 376
Object Browser, 107
Output, 863
Parallel Stacks, 873
Parallel Tasks, 874
Pending Checkins, 294
Plug-in Settings, 291
Processes, 869
Properties, 67; 226; 671
Publish Web, 1016
Query Builder, 610
Quick Find/Replace, 116
QuickWatch, 866
Registers, 870
Server Explorer, 204; 594
Solution Explorer, 56
Task List, 284
Testing Center, 1134
Threads, 868; 930
Toolbox, 63; 670
UML Model Explorer, 1092
Watch, 865
History, 1153
закрепленное, 48
открепленное, 48
плавающее, 83
развернутое, 48
свернутое, 48
Операции
CRUD, 529; 679
Операция
вставки, 800
вызова, 801

домена, 800
модификации, 800
специальная, 801
удаления, 800
Определение сборки, 1157
Отключение
жесткое, 436
мягкое, 436
Отладка, 861
в смешанном режиме, 933
многопоточных приложений, 930
посмертная, 933
распараллеленных приложений, 930
удаленная, 927
хранимых процедур SQL Server, 932
Отладчик, 1037
Отображение, 665; 666
Отражение, 259
Отслеживание, 988
Очередь
закрытая, 213
открытая, 213
системная, 213
Ошибка
на уровне приложения, 910
на уровне страницы, 909

П

Пакет
.vsix, 1070
Visual Studio Content Installer, 1052
Visual Studio Integration SDK, 1070
Visual Studio SDK, 1032
Visual Studio Content Installer, 1059
VSPackages, 1032
VSTO, 421
Панель, 47
Actions Pane, 424
Task Pane, 424
Toolbox, 224
Папка, 134
решения, 134
Переменная
группы, 704
отчета, 704
Переход к определению, 103
Перечисление, 225
Персистентность, 746
Планировщик, 745
Платформа
ASP.NET AJAX, 471
SharePoint 2010, 556
SharePoint Foundation, 556
SharePoint Server, 556
Windows Azure, 576
Windows SharePoint Services, 556
Windows Workflow Foundation, 741
WPF, 515
Подсветка ссылок, 99
Подсистема
классификации, 1069
модели текста, 1068
операционная, 1070
представления текста, 1069
Подсказка, 52
по данным, 862
закрепленная, 892
плавающая, 892
Поиск
всех ссылок, 103
Поле, 1073
Получатель события, 561
Представление, 481; 598
частичное, 506
Презентатор, 1075
Прецедент, 1089
Привязка, 500; 728
Привязка данных, 409
Приложение
XBAP, 515
без постоянного подключения, 784
без постоянного подключения, 765
хоста, 745
Программа
Expression Blend, 414
LINQPad, 661
Проект, 55; 139
Database, 140
Office, 139
Reporting, 140
SharePoint, 139
Silverlight, 140
Test, 140

WCF, 140
Web, 139
Windows, 139
Workflow, 140
командный, 1139
установки, 56
Проектировщик отчетов, 688
Прокси-класс метаданных, 504
Процедура
хранимая, 655
Процессор
мыши, 1074
Псевдоним команды, 104

P

Рабочее пространство, 1147
Рабочий поток, 741
Рабочий процесс, 562
конечного автомата, 570
последовательный, 570
Равенство возможностей, 355
Разархивирование, 1154
Раздел конфигурации
compiler, 816
configSections, 812
cryptographySettings, 815
remoting, 814
runtime, 813
sturtup, 813
system.diagnostics, 815
system.net, 814
system.web, 816
Разработка
через тестирование, 164
Расширение, 1031
Регион, 96
Редактирование
многострочное, 102
Редактор
CSS, 458
HTML, 458
HTML Designer, 450
Visual Studio 2010 Editor, 1068
Реестр, 997
Режим
End and Continue, 889
автономный, 128
дополнения, 168
заглушки, 169
интерактивный, 128
подсказок, 164; 168
полного доверия, 156
полноэкранный, 103
прерывания, 862
частичного доверия, 156
Разделение окна, 100
Ресурс, 407; 840
Рефакторинг, 182; 228
базы данных
переименование, 1116
перемещение схемы, 1116
полностью квалифицированные имена, 1116
операция
Encapsulate Field, 196
Extract Interface, 197
Extract Method, 194
Generate Method Stub, 201
Organize Usings, 202
Remove Parameters, 200
Rename, 200
Reorder Parameters, 199
Решение, 55; 134
Роль, 577
ASP.NET Web Role, 577
Worker Role, 577

C

Сборка, 280; 962
PIA, 356
Свойство, 363
зависимости, 412
присоединенное, 399
Сворачивание, 97
Сворачивание текста, 86; 99
Связь, 665
Сеанс, 1075
профилирования, 1107
Сервис-релиз, 44
Сериализация, 854; 951
Синтаксис "свойство-элемент", 400
Синхронизация

принудительная, 788
фоновая, 790
Скаффолдинг, 530
Слияние, 1156
Слой объектно-реляционного отображения, 649
Служба, 219
RIA Services, 795
Runtime Intelligence Service, 986
домена, 798
клиентская, 766
редактора, 1076
синхронизации, 783
Сниппет, 177; 182
Событие, 1037
генерируемое после сборки, 283; 348
сборки, 147; 952
Соединитель, 224
Сопоставление образцов, 371
Спецификация
CLS, 355
Список обследования, 561
Среда
Development Fabric, 578
Среда выполнения, 745
Средство
Access Control Service, 589
Service Bus, 588
просмотра отчета, 688
Ссылка
на внешнюю сборку, 151
на проект, 151
на службу, 62
на файл, 151
Стандарт кодирования, 298
Стиль, 413
Страница
контента, 564
мастера, 467
правил доступа, 476
Строка
подключения, 830; 1019
Структура, 225
Сущность, 224; 665

T

Тема, 413
приложения, 524
элементов управления, 524
Тест
Coded UI Test, 1128
автоматизированный, 1121
веб-тест, 1122
загрузки, 1125
модульный, 1127
обобщенный, 1129
Тестовый прогон, 244
Технология
Active Server Pages (ASP), 480
ADO.NET Entity Framework, 531
AppFabric, 588
ASP.NET, 464; 905
ClickOnce, 1006
Entity Framework, 664
IntelliSense, 126; 163; 1075
LINQ, 633
LINQ to Entities, 678
LINQ to SQL, 531; 649
LINQ to XML, 641
Runtime Intelligence, 983
Silvernight, 515
SQL AZURE, 587
Web Forms, 480
Windows Azure, 577
Windows Forms, 375
Windows Presentation Foundation, 392
Тип
классификации, 1072
контента, 561
модели представления, 797
содержимого, 1071
Точка
выполнения, 862
импорта, 1061
прерывания, 862; 878
по адресу, 880
простая, 879
условная, 863; 881
функции, 880
трассировки, 885

экспорта, 1061
Трассировка, 913
на уровне приложения, 914
на уровне страницы, 913

У

Узел
Data Connections, 220
Event Logs, 205
Message Queues, 213
Performance Counters, 216
Servers, 205
Services, 219
Укрупнение кода, 99
Умышленное запутывание, 972
интеллектуальное, 979
Унифицированный язык моделирования, 1088
Управление
версиями, 1147
исходными текстами, 1147
Упрощенный синтаксис, 368
Утверждение, 253
Утилита
Dotfuscator, 982
Microsoft Visual SourceSafe 2005, 290
Reflector, 348
Strong Name, 980
TamperTester, 986
WCF Test Client, 732

Ф

Файл
.AddIn, 1044
.config, 809
.cs, 317
.csproj, 140
.fsproj, 140
.mdf, 595
.resources, 843
.sln, 135
.suo, 135
.tdlog, 1111
.testsettings, 244
.tt, 307
.udl, 607
.vbproj, 140
.vsmacros, 1054
.vsmdi, 244
.vstemplate, 337
.xap, 518
app.config, 810
applicationHost.config, 811
machine.config, 810
resx, 842
security.config, 811
web.config, 810
для исключительного редактирования, 1149
для совместного редактирования, 1149
записанный, 1149
определения отчета, 688
поддержки, 245
редактируемый, 1149
текстовый, 841
Фильтр, 883
Форма сведений, 412
Функция
обратного вызова, 745

Х

Хранилище кода, 289
CVS, 290
Subversion, 290
Visual SourceSafe 2010, 292
выбор, 289
запись файла, 294
извлечение файла, 294
история изменений, 297
настройки, 291
объединение изменений, 296
откладывание изменений, 294
создание, 292
фиксация версии, 297
Хранимая процедура, 598

Ш

Шаблон
ASP.NET Web Application, 531

Cloud Service, 577
Database Project, 1113
Dynamic Data, 507
Dynamic Data Entities, 531
Dynamic Data Linq to SQL Web Application, 531
F# Tutorial, 368
Modeling Project, 1088
Process Template, 1094
Reports Application, 686
Shared Add-in, 1040
Silverlight Application, 519
Silverlight Navigation Application, 516
SQL-CLR Database, 1112
Syndication Service Library, 722
Visual Studio Add-in, 1040
VSIX Project, 1070
WCF Workflow Service Application, 722
интерактивный, 349
поля, 545
проекта, 336
ASP.NET AJAX Server Control, 448
ASP.NET AJAX Server Control Extender, 448
ASP.NET Crystal Reports Web Site, 448
ASP.NET MVC 2 Web Application, 447
ASP.NET Reports Web Site, 447
ASP.NET Server Control, 448
ASP.NET Web Service, 447
Dynamic Data Web Site and Web Application, 448
Empty Web Site, 444
Web Application, 442
Web Site, 442
процесса, 1139
редактирования, 510
страницы, 542
сущности, 549
фильтра, 551
элемента, 332
элемента управления, 413
Шаблонные символы, 118

Э

Эквивалентность типов, 356
Экспорт, 1062
Элемент, 958
Target, 959
Task, 959
рабочий, 1141
вопрос, 1143
задача, 1143
контрольный пример, 1143
общий этап, 1143
ошибка, 1143
пользовательский сценарий, 1143
AJAX Control Extenders, 473
Элемент управления
Chart, 699
Gauge, 700
Image, 698
ImplicitStyleManager, 523
Line, 693
List, 698
Matrix, 696
Rectangle, 693
Subreport, 699
Table, 693
Text Box, 691
графический, 380
Button, 381
Label, 382
TextBox, 382
компоновочный, 398
Canvas, 399
Grid, 400
контейнерный, 387
FlowLayoutPanel, 388
Panel, 387
SplitContainer, 387
StackPanel, 401
TableLayoutPanel, 388
контентом, 423
системы WPF, 397
служебный, 385
стыковкой и закреплением
Anchor, 389

Dock, 389
Этап сборки, 280

Я

Язык программирования
C#, 366
Язык
интегрированных запросов, 632
Язык программирования
Visual Basic, 361
DSL, 665
F#, 368
JavaScript, 469
декларативный, 352
динамический, 353
императивный, 352
машинно-ориентированный, 366
функциональный, 354
Язык разметки
XAML, 392; 515

C# 4 И ПЛАТФОРМА .NET 4 ДЛЯ ПРОФЕССИОНАЛОВ

Кристиан Нейгел
Билл Ивьен
Джей Глинн
Карли Уотсон
Морган Скиннер



www.dialektika.com

Книга известных специалистов в области разработки приложений с использованием .NET Framework посвящена программированию на языке C# 2010 в среде .NET Framework 4 и в предшествующих версиях. Книгу отличает простой и доступный стиль изложения, изобилие примеров и рекомендаций по написанию высококачественных программ. Подробно рассматриваются такие вопросы, как основы языка программирования C#, организация среды .NET, работа с данными, написание Windows- и веб-приложений, взаимодействие через сеть, создание веб-служб и многое другое. Немалое внимание уделено проблемам безопасности и сопровождения кода. Тщательно подобранный материал позволит без труда разобраться с тонкостями использования Windows Forms и построения веб-страниц. Читатели ознакомятся с работой в Visual Studio 2010, а также с применением различных технологий, встроенных в .NET. Книга рассчитана на программистов разной квалификации.

ISBN 978-5-8459-1656-3

в продаже

ЯЗЫК ПРОГРАММИРОВАНИЯ C# 2010 И ПЛАТФОРМА .NET 4

5-е издание

Эндрю Троелсен



www.williamspublishing.com

Версия .NET 4 привнесла множество новых API-интерфейсов в библиотеках базовых классов, а также новых синтаксических конструкций в языке C#.
В этой книге вы найдете полное описание всех нововведений в характерной для автора дружественной к читателю манере. Помимо общих вопросов, подробно рассматривается среда Dynamic Language Runtime (DLR); библиотека Task Parallel Library (TPL, включая PLINQ); технология ADO.NET Entity Framework (а также LINQ to EF); расширенное описание API-интерфейса Windows Presentation Foundation (WPF); улучшенная поддержка взаимодействия с COM.
В книге рассматриваются следующие темы

- Особенности платформы .NET 4 и языка Visual C# 2010
- Детали технологии .NET – лидера в производстве современного программного обеспечения
- Полезные советы по разработке от эксперта в .NET, который изучает эту платформу, начиная с ее первой версии
- Полное описание технологий WPF, WCF и WF, поддерживаемых ядром платформы .NET

ISBN 978-5-8459-1682-2

в продаже

C# 4.0
ПОЛНОЕ РУКОВОДСТВО

Герберт Шилдт



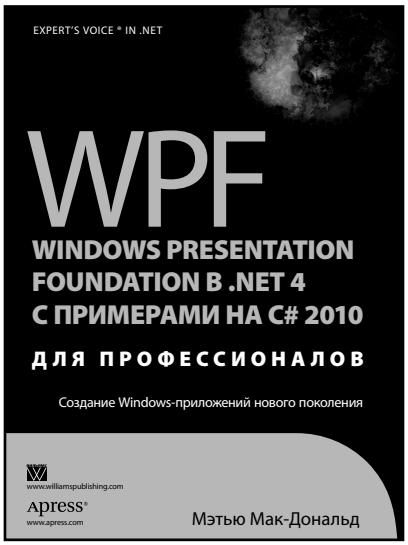
www.williamspublishing.com

В этом полном руководстве по C# 4.0 — языку программирования, разработанному специально для среды .NET, — детально рассмотрены все основные средства языка: типы данных, операторы, управляющие операторы, классы, интерфейсы, методы, делегаты, индексаторы, события, указатели, обобщения, коллекции, основные библиотеки классов, средства многопоточного программирования и директивы препроцессора. Подробно описаны новые возможности C#, в том числе PLINQ, библиотека TPL, динамический тип данных, а также именованные и необязательные аргументы. Это справочное пособие снабжено массой полезных советов авторитетного автора и сотнями примеров программ с комментариями, благодаря которым они становятся понятными любому читателю независимо от уровня его подготовки. Книга рассчитана на широкий круг читателей, интересующихся программированием на C#.

ISBN 978-5-8459-1684-6 **в продаже**

WPF
WINDOWS PRESENTATION FOUNDATION В .NET 4 С ПРИМЕРАМИ НА C# 2010 ДЛЯ ПРОФЕССИОНАЛОВ

Мэтью Мак-Дональд



www.williamspublishing.com

Книга ведущего специалиста в области технологий .NET представляет собой учебное и справочное пособие по технологии WPF, входящей в состав .NET 4, для разработчиков высококлассных приложений, которые ориентированы на семейство операционных систем семейства Windows. Данная технология позволяет сочетать в рамках приложения пользовательский интерфейс, документы и медиасодержимое, располагая при этом полной поддержкой со стороны операционной системы.

Функциональность WPF расширена поддержкой Tablet PC и других форм устройств ввода. Теперь WPF предоставляет более развитый конвейер рисования и печати, инфраструктуру доступности и автоматизации пользовательских интерфейсов, управляемые данными интерфейсы и виртуализацию, а также точки интеграции приложений с командной оболочкой Windows. В этой книге описано, как в действительности работает WPF.

ISBN 978-5-8459-1657-0 **в продаже**

WCF 4
WINDOWS COMMUNICATION FOUNDATION И .NET 4
ДЛЯ ПРОФЕССИОНАЛОВ

Пабло Сибраро и др.



www.dialektika.com

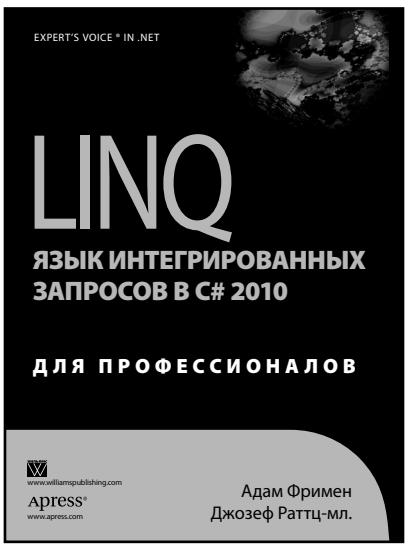
В этой книге рассказывается о платформе Windows Communication Foundation в .NET 4.0. Платформа WCF является технологией .NET, используемой при создании приложений, предназначенных для широкого круга запросов, обмена сообщениями в различных коммуникационных случаях и выполнения потоков задач, состоящих из действий служб. Эта новая книга поможет понять принцип ориентации на службы, изучить примеры коммуникации и научиться описывать бизнес-процессы декларативно. Вы также научитесь эффективно использовать среду разработки Visual Studio 2010 для создания решений, в которых максимально применяются новые возможности платформы WCF 4.0.

В данной книге описывается взаимодействие разработчика и архитектора при создании приложений, интегрирующихся в новую парадигму программирования платформы WCF 4.0. Вы также узнаете, как установить решение в этой новой архитектурной форме, основанной на платформе WCF 4.0, с помощью технологий и служб .NET.

ISBN 978-5-8459-1713-3 **в продаже**

LINQ
ЯЗЫК ИНТЕГРИРОВАННЫХ ЗАПРОСОВ В C# 2010 ДЛЯ ПРОФЕССИОНАЛОВ

Адам Фримен
Джозеф Раттц-мл.



www.williamspublishing.com

LINQ представляет собой часть платформы .NET Framework, которая обеспечивает обобщенный подход к запросам данных из различных источников. Знание LINQ быстро становится необходимым для всех разработчиков приложений .NET. Эта книга посвящена написанию кода с помощью LINQ. Многие книги предлагают простые примеры использования методов и зачастую ими ограничиваются. Настоящая книга не такая. За счет демонстрации особенностей применения широкого разнообразия операций и прототипов LINQ она становится неоценимым источником реальных примеров использования LINQ. Книга написана в дружественном к читателю стиле и позволит эффективно освоить применение LINQ в разработке приложений.

ISBN 978-5-8459-1701-0 **в продаже**

C# 2010 УСКОРЕННЫЙ КУРС ДЛЯ ПРОФЕССИОНАЛОВ

Трей Нэш



www.williamspublishing.com

Книга ведущего специалиста в области технологий .NET представляет собой интенсивный курс по новейшей версии языка C#, воплотившей в себе важные дополнения и предлагающей среду, в которой функциональное программирование может органично переплетаться с обычным стилем императивного программирования на C#. Эта книга может служить как кратким учебным пособием, так и постоянно используемым справочником. Вы быстро освоите синтаксис C#, изучая, как среда CLR упрощает решение многих задач программирования. Вы также получите большой набор рекомендаций по построению эффективного, многократно используемого и надежного кода. Зачем тратить месяцы или даже годы на поиск наилучших подходов к проектированию и написанию кода на C#, если в данной книге будет показано, как все делать правильно с самого начала? Вы узнаете, как C# работает и взаимодействует с CLR, как использовать массивы и многое другое.

ISBN 978-5-8459-1638-9

в продаже