Самоучитель

Джо Майо

Microsoft® Visual Studio 2010

Настройка среды разработки

Вводный курс и особенности использования языков C# и VB.Net

Разработка основных типов приложений

Современные технологии программирования: WPF, WCF, ASP.NET, Silverlight и др.

bhv

Microsoft® Visual Studio® 2010

A Beginner's Guide

Joe Mayo

New York Chicago San Francisco
Lisbon London Madrid Mexico City
Milan New Delhi San Juan
Seoul Singapore Sydney Toronto

Джо Майо

Самоучитель Microsoft® Visual Studio 2010

Санкт-Петербург
«БХВ-Петербург»
2011

УДК 681.3.06
ББК 32.973.26-018.2
М14

Майо Дж.

М14 Самоучитель Microsoft Visual Studio 2010. — СПб.: БХВ-Петербург, 2011. — 464 с.: ил.

ISBN 978-5-9775-0609-0

Показано создание различных типов приложений в интегрированной среде разработки Microsoft Visual Studio 2010. Рассмотрены основы программирования на языках C# и VB, работа с решениями, проектами, сборками и библиотеками классов. Описаны инструменты, предназначенные для анализа и отладки кода, поиска и исправления ошибок. Рассмотрена работа с базами данных с помощью языка интегрированных запросов LINQ. Приведена информация о языках XML и XAML. Описаны основные концепции работы с системой Windows Presentation Foundation, технология Silverlight, построение Web-приложений с помощью технологии ASP.NET MVC, создание Web-сервисов с помощью Windows Communications Foundation. Рассмотрено создание собственной программы-мастера для работы над проектами, шаблонов для автоматизации генерируемых фрагментов кода и рутинных задач, добавочных модулей и др.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Перевод с английского и редактирование	*Ольги Кокоревой*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 02.08.10.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 37,41.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

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

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

ISBN 978-0-07-166895-8 (англ.)
ISBN 978-5-9775-0609-0 (рус.)

Оглавление

Благодарности

Работа над любой книгой — это не только труд автора. Свой вклад в эту работу вносит весь коллектив высококвалифицированных профессионалов, каждый из которых выполнял свою часть работы, помогая автору. Я хочу поблагодарить лично всех тех членов группы подготовки этой книги, без которых ее появление было бы невозможным.

Джейн Браунлав (Jane Brownlow), ответственный редактор, оказала бесценную помощь в запуске проекта этой книги и все время помогала мне оставаться на правильном пути. Мег Морин (Megg Morin), рецензент издательства McGraw-Hill, приняла бразды правления от Джейн и помогла пройти остаток пути. Джойя Энтони (Joya Anthony), координатор проекта, следила за сдачей отдельных глав. Мэдху Бхардваж (Madhu Bhardwaj), менеджер проекта и Пэтти Мон (Patty Mon), заведующий редакцией, помогали в координации работы по редактированию и верстке. Всех этих, и многих других сотрудников McGraw-Hill, которые помогали выпуску книги, я приношу огромную благодарность за их вклад в общую работу, терпение и профессионализм.

Отдельной благодарности заслуживает Рой Огборн (Roy Ogborn), технический редактор этой книги. Я знаком с Роем уже много лет, и всегда радуюсь, когда он соглашается редактировать мои книги. Помимо того, что он очень тщательно и внимательно отлавливает все ошибки, в некоторых областях он дает и очень ценные советы касательно того, на каких технологиях следует поставить акцент, как сделать изложение более понятным для новичка или о том, какой язык выбрать для достижения той или иной цели. Огромное спасибо Рою за его выдающиеся редакторские качества и ценные советы.

Джо Майо (Joe Mayo)

Введение

Visual Studio уже в течение многих лет является ведущей интегрированной средой разработки (Integrated Development Environment, IDE) среди всех инструментальных средств разработчика, поставляемых Microsoft. Новая версия этого ПО, которой посвящается эта книга, представляет собой естественный результат эволюции, являющийся логическим продолжением всех предшествующих версий. В этой книге показано, как максимально использовать себе на благо мощный потенциал Visual Studio 2010, приобрести новые знания и навыки, а также повысить производительность труда при разработке приложений. Основное внимание в книге уделено написанию приложений для .NET (произносится как "Dot Net") — платформы Microsoft для разработки различных типов ПО, от обычных программ до Web-приложений.

Как следует из ее названия, книга представляет собой руководство для начинающих. Однако мнения о том, кто такой "новичок", могут сильно разниться, поэтому давайте сначала обсудим, кого следует считать новичком в контексте этой книги. Я, как ее автор, не считаю, что вы совсем ничего не знаете о программировании. Я предполагаю, что вы все же имеете некоторое представление о том, что такое программирование, хотя бы в общих чертах, и способны, как минимум, написать простой командный файл, макрос или командный сценарий, чтобы автоматизировать какую-нибудь рутинную задачу. Кроме того, в контексте этой книги новичками следует считать и всех тех, кто не знаком с Visual Studio, но имеет опыт разработки ПО с использованием каких-либо других технологий, например, Cobol, Dreamweaver или Java. Каким бы ни был ваш предварительной опыт и базовый уровень знаний, изложение в книге построено таким образом, чтобы идти от простого к более сложному и постепенно приобретать опыт создания приложений с помощью Visual Studio 2010.

Данная книга состоит из 13 глав, разбитых на 4 части, и содержит три приложения, в которых приведен справочный материал. Вот краткий обзор материала, представленного в этой книге:

- Часть I: *"Базовая информация о Visual Studio 2010"*. В состав этой части входят *главы 1 — 4*. *Глава 1* содержит краткое описание VS и рассказывает о предоставляемых этой средой преимуществах и типах приложений, которые можно создавать с ее помощью. Вкратце описывается процедура установки VS, после чего даются советы и рекомендации по выбору опций установки и каталогов, в которые производится установка. *Главы 2 — 4* представляют собой вводный курс по C# и VB — двум наиболее популярным у разработчиков языкам

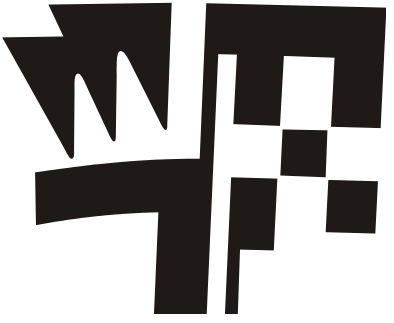
программирования, поддерживаемым в VS. Они содержат тот минимум базовой информации, который необходим для понимания дальнейшего материала, излагаемого в данной книге. По мере ее чтения вы будете знакомиться и с остальными функциональными возможностями этих языков и учиться использовать их в своих программах. Даже если вы уже имеете опыт программирования, я рекомендую прочесть эти главы хотя бы бегло, поскольку в них приведено множество полезных советов, которые позволят вам повысить индивидуальную производительность вашего труда за счет грамотного применения особенностей VS.

- Часть II: *"Изучаем среду разработки VS 2010"*. В этой части рассматриваются некоторые распространенные задачи, которые большинство программистов выполняют ежедневно, в том числе: работа с решениями и проектами, отладка кода и работа с данными. *Глава 5* описывает создание и построение проектов, а также прочие задачи, связанные с манипулированием проектами и решениями. Обратите особое внимание на указания по работе со сборками (assemblies) и библиотеками классов (class libraries), поскольку по мере того, как вы будете переходить от простых проектов к более сложным, их роль будет неуклонно возрастать. Вне зависимости от того, какой философии разработки вы придерживаетесь, вам, как и любому другому программисту, в любом случае придется искать и исправлять ошибки в коде, причем важность этого навыка со временем тоже будет только возрастать. Этому вопросу посвящена *глава 6*, в которой описываются инструменты VS, предназначенные для анализа кода, поиска и исправления ошибок. Наконец, еще одна важная задача, с которой сталкивается каждый программист, — это работа с данными. VS позволяет создавать базы данных, добавлять в них таблицы, устанавливать между ними взаимосвязи, и много другое. Когда база данных будет создана и готова к использованию, вы сможете написать программный код, который будет обращаться к этой базе данных. В данной книге рассматривается язык интегрированных запросов LINQ для SQL (LINQ to SQL) как одна из наиболее простых, но, в то же самое время, достаточно мощных технологий работы с базами данных, доступных разработчикам ПО.
- Часть III: *"Разработка приложений с помощью VS 2010"*. После того как вы приобретете базовые знания о поддерживаемых в VS языках программирования и в общих чертах ознакомитесь со средой разработки, можно приступать к реальному программированию с помощью VS. Платформа .NET поддерживает различные технологии, и подход данной книги заключается в изложении материала "с прицелом на будущее". Это значит, что при рассмотрении приоритет отдается новейшим технологиям, появившимся совсем недавно. Автор не ставит перед собой цели научить вас абсолютно всему, что связано с этими технологиями, потому что для описания каждой из них потребуется отдельная книга. Тем не менее, в главах из этой части показывается, как наиболее рационально пользоваться возможностями VS при разработке приложений. Здесь приводится базовая информация о том, как быстро начать разработку собственных приложений. *Главы 8* и *10* содержат вводную информацию о форме языка XML

(Extensible Markup Language), называемой XML Application Markup Language (XAML). Поскольку эта книга предназначена для начинающих, основные справочные сведения о языках XML и XAML приведены в *приложениях 1* и *2*. Если вы не знакомы с этими языками, то перед чтением *глав 8* и *10* я рекомендую прочесть эти приложения. Кроме того, *главу 8* следует читать перед *главой 10*, потому что многие из концепций работы с Windows Presentation Foundation (WPF), технологией разработки обычных приложений, применимы и к Silverlight, технологии создания Web-приложений. Остальные две главы, входящие в эту часть, демонстрируют построение Web-приложений с помощью технологии ASP.NET MVC и создание Web-сервисов с помощью Windows Communications Foundation.

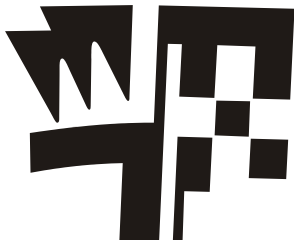
- Часть IV: *"Расширение возможностей VS 2010"*. В дополнение ко всем программам-мастерам (wizards), инструментальным средствам, утилитам и средствам редактирования, которые предлагает среда VS, вы можете дополнительно расширить и нарастить ее возможности. В *главе 12* показано, как создать собственные программы-мастера для работы над проектами, как создавать собственные шаблоны для автоматически генерируемых фрагментов кода и как создавать макросы для автоматизации рутинных задач в среде VS. Если возможность написания макросов, с которой вы познакомились в VS, не кажется вам достаточно мощной, прочтите *главу 13*, в которой рассказывается о создании добавочных модулей (Add-In), программ, которые вы можете написать и установить для расширения возможностей VS. Я надеюсь, что данная книга поможет вам в овладении возможностями VS и что после этого процесс программирования будет доставлять вам удовольствие.

Джо Майо (Joe Mayo)



ЧАСТЬ I

Базовая информация о Visual Studio 2010



Глава 1

Первое знакомство с Visual Studio 2010

В этой вводной главе мы рассмотрим следующие ключевые концепции, необходимые начинающим, чтобы приступить к работе с Microsoft Visual Studio 2010:

- Чем может вам помочь Visual Studio 2010;
- Установка Visual Studio 2010 и выбор инсталляционных опций;
- Какие типы приложений можно создавать с помощью Visual Studio 2010.

Чаще всего первое знакомство с Visual Studio (VS) 2010 начнется для вас с установки этого продукта. Как и большинство других программ, среда разработки VS очень проста в установке. В данной главе будет описан процесс инсталляции, а также даны рекомендации, помогающие разобраться в предлагаемых опциях. Как только процедура установки завершится, вы сможете в первый раз запустить среду VS. После этого вам потребуется разобраться с навигацией по различным окнам среды разработчика. Эта глава дает краткое описание организации VS и показывает, как найти нужные функции и как работать с окнами. Наконец, будет рассказано, как найти типы приложений, которые можно компилировать (compile) и строить (build) с помощью VS. К этому моменту вы наверняка уже знаете, что VS позволяет создавать приложения .NET, но обсуждение мы начнем с детального описания задач, которые вы можете решать с помощью Visual Studio.

Что позволяет делать Visual Studio 2010?

Visual Studio 2010 (VS) представляет собой интегрированную среду разработки (Integrated Development Environment, IDE). IDE — это набор инструментов разработчика ПО, собранный в составе единого приложения и облегчающий труд программиста при написании приложений. Давайте вспомним, что без IDE (в данном случае — без VS) для написания программы требуется текстовый редактор, с помощью которого программист вводит весь исходный код своей будущей программы. Затем, когда исходный код написан, необходимо запустить из командной строки компилятор, чтобы создать исполняемый файл приложения. Основная проблема, связанная с использованием текстового редактора и компилятора, запускаемого из командной строки, заключается в том, что вы выполняете большое количество ручной работы и теряете при этом много времени. К счастью, с помощью VS многие из этих рутинных и трудоемких задач, связанных с повседневной рабо-

той программиста, можно автоматизировать. Последующие несколько разделов данной главы поясняют, чем может вам помочь интегрированная среда разработки и почему во главу угла в VS поставлена продуктивность труда разработчика.

Автоматически генерируемый код

В состав VS входит целый набор типовых проектов, из которых каждый разработчик может подобрать именно то, что ему в данный момент требуется. Каждый раз, когда вы создаете новый проект, VS автоматически создаст "скелет" будущего приложения, причем этот код можно немедленно скомпилировать и запустить на исполнение.

В составе каждого типового проекта имеются элементы, которые по желанию добавлять в ваш проект. Любой проект, в любом случае, содержит автоматически сгенерированный код, который представляет собой основу будущей программы. В следующей главе будет показано, как нужно действовать, чтобы создавать новые проекты, добавлять в их состав новые элементы и просматривать автоматически сгенерированный код. VS предлагает множество готовых к использованию элементов управления, включая и код, необходимый для их создания. Это экономит время разработчиков, избавляя их от необходимости каждый раз заново создавать типовой программный код для решения часто встречающихся задач. Многие из более сложных элементов управления содержат так называемые "программы-мастера" (Wizards), которые помогают настроить поведение элементов управления, автоматически генерируя код в зависимости от выбранных вами опций.

Опыт быстрого кодирования (Rapid Coding Experience)

Редактор VS оптимизирует работу программиста по кодированию. Существенная часть синтаксических элементов новой программы выделяется при помощи системы цветовых обозначений. В вашем распоряжении окажутся такие технологии, как Intellisense, известная как автодополнение. В ходе того, как вы будете вводить новый код, на экране будут появляться всплывающие подсказки. Наконец, для ускорения выполнения многих задач вам будет предоставлено большое количество клавиатурных комбинаций (keyboard shortcuts). Существует и набор средств быстрой переработки или рефакторинга (refactoring), которые позволяют быстро усовершенствовать структуру кода, не отрываясь от процесса программирования. Например, функция переименования (Rename) позволяет изменить имя идентификатора там, где оно определено, и это повлечет за собой изменение имени данного идентификатора повсюду, где он встречается в коде программы. Кроме того, VS предоставляет и множество других, не менее удобных возможностей. Например, иерархия вызовов (call hierarchy) позволяет программисту проследить всю цепочку вызовов в коде приложения, с верхних уровней до самых нижних. Еще одна возможность, автоматически генерируемые фрагменты (snippets), позволяет вводить сокращения, которые разворачиваются в шаблоны кода (code template). Наконец, списки действий (action lists) предназначены для автоматической генерации нового кода.

Все необходимое — всегда под рукой

Вам действительно необходимо научиться ориентироваться в среде VS, чтобы разобраться во всем множестве новых возможностей, призванных упростить нелегкую задачу быстрой разработки высококачественных приложений. Так, окно **Toolbox** просто "до отказа" забито различными элементами управления, окно **Server Explorer** предназначено для работы с сервисами и базами данных операционной системы, а окно **Solution Explorer** дает возможность работать с вашими проектами, тестировать утилиты, и предоставляет средства визуального проектирования. Кроме того, предоставляются и средства работы с компиляторами.

Настраиваемость и расширяемость

Многие из элементов, образующих среду VS, являются настраиваемыми. Это значит, что вы можете менять цвета отображения элементов кода, опции редактора, а также общее оформление. Набор опций настолько обширен, что и в самом деле необходимо потратить некоторое время, чтобы ознакомиться с ними и привыкнуть к тому, где и какую из них следует искать. Если стандартные настройки среды VS, которые по умолчанию предлагаются сразу же после установки, вас не устраивают, или не предлагают нужных вам опций, вы можете написать собственные макросы, чтобы автоматизировать последовательности часто выполняемых шагов. Для более сложных вариантов настройки VS предлагает специальный интерфейс прикладного программирования (Application Programming Interface, API), предназначенный для создания собственных дополнительных модулей (add-ins) и расширений (extensions). Некоторые сторонние разработчики уже предлагают приложения, способные интегрироваться в среду VS. В качестве примера можно назвать среду разработки на Delphi от компании Embarcadero[1], которая встраивается в Visual Studio. Настраиваемая и гибкая среда VS позволит вам работать так, как удобно именно вам.

По мере того как вы будете читать эту книгу, имейте в виду все эти важные концепции и внимательно читайте все рекомендации, которые позволят вам продуктивнее работать с VS. Вашим первым шагом, с которого начнется работа с VS, должна стать установка этого продукта, которая будет обсуждаться в следующем разделе.

Установка Visual Studio 2010

Предположительно, все сказанное в предыдущих разделах должно было подогреть ваше желание начать работу с VS и воспользоваться всеми возможностями новой версии данного продукта. Если до сих пор вам никогда не приходилось устанавливать VS, внимательно прочтите этот раздел, потому что именно здесь приводятся пошаговые инструкции, которые позволят вам успешно выполнить процедуру установки. Даваемые по ходу описания указания и рекомендации позволят

[1] Дополнительную информацию см. здесь: **http://en.wikipedia.org/wiki/Embarcadero_Delphi**, скачать можно отсюда: **http://www.embarcadero.com/**. — *Прим. перев.*

вам правильно выбрать опции, нужные именно вам, и в итоге получить среду разработчика, настроенную в соответствии именно с вашими потребностями. В процессе установки VS нужно будет выполнить следующие шаги:

1. Уточнить системные требования и выяснить, удовлетворяет ли им ваша системная конфигурация.

> ***Системные требования***
>
> На момент написания данной книги Microsoft рекомендует следующую аппаратную конфигурацию компьютера: процессор — 32-разрядный (x86) или 64-разрядный (x64) CPU, не менее 1 Гбайт RAM, жесткий диск со скоростью вращения 5400 RPM, 3 Гбайт свободного дискового пространства, привод DVD-ROM, Видео — DirectX с разрешением 1280×1024, тактовая частота процессора — 1.6 Ггц. Рекомендуемая операционная система: Windows Vista (все варианты, кроме Starter), Windows XP SP2 или более новый (все варианты, кроме Starter), Windows 7 (на момент написания этой книги — только редакция Ultimate), Windows Server 2003 (SP1 или R2, или более новая версия), Windows Server 2008 (SP1, R2 или более новая версия). Не забудьте посетить сайт Microsoft Developer Network (MSDN) и уточнить системные требования, потому что со временем они могут измениться.

2. Когда вы впервые вставите дистрибутивный DVD с копией VS 2010 в свой привод, на экране появится окно-заставка Microsoft Visual Studio 2010, показанное на рис. 1.1. В этом окне доступны опции **Install Microsoft Visual Studio 2010** и **Check For Service Releases**. Чтобы начать процедуру установки, выберите опцию **Install Microsoft Visual Studio 2010**.

Рис. 1.1. Окно Microsoft Visual Studio 2010 Setup

3. Следующее окно, которое вы увидите на экране, отображает приветствие Microsoft Visual Studio 2010. В примере, показанном на рис. 1.2, изображено окно

приветствия, в котором пользователь выбрал для установки версию **Ultimate**. Здесь нужно отметить, что для других вариантов установки процедура будет аналогичной, изменяться будет лишь количество и состав доступных опций, который зависит от выбранного варианта установки.

4. Если на этой странице вы установите флажок **Help Improve Setup**, то программа-инсталлятор в ходе процесса установки будет собирать информацию из журналов (logs), создаваемых в ходе установки, и отправит их в Microsoft после завершения процесса, используя действующее интернет-соединение. Чтобы помочь вам сделать информированный выбор касательно того, хотите ли вы отправлять эту информацию разработчикам продукта через Интернет, они поместили в данное окно ссылку **Privacy Statement**. Щелкните мышью по этой ссылке и внимательно прочтите информацию о том, как и с какими целями Microsoft будет использовать полученную от вас информацию. Когда вы примете окончательное решение, щелкните по кнопке **Next**. После того как загрузятся компоненты, необходимые для установки, на экране появится окно с текстом лицензионного соглашения, показанное на рис. 1.3.

Рис. 1.2. Окно приветствия программы Setup

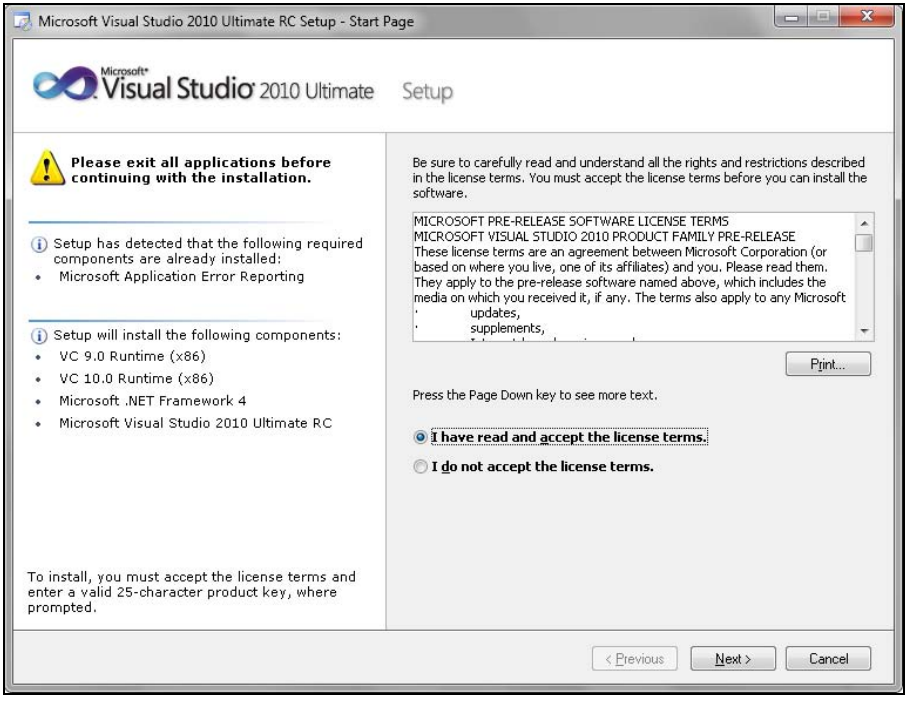


Рис. 1.3. Окно программы Setup с текстом лицензионного соглашения

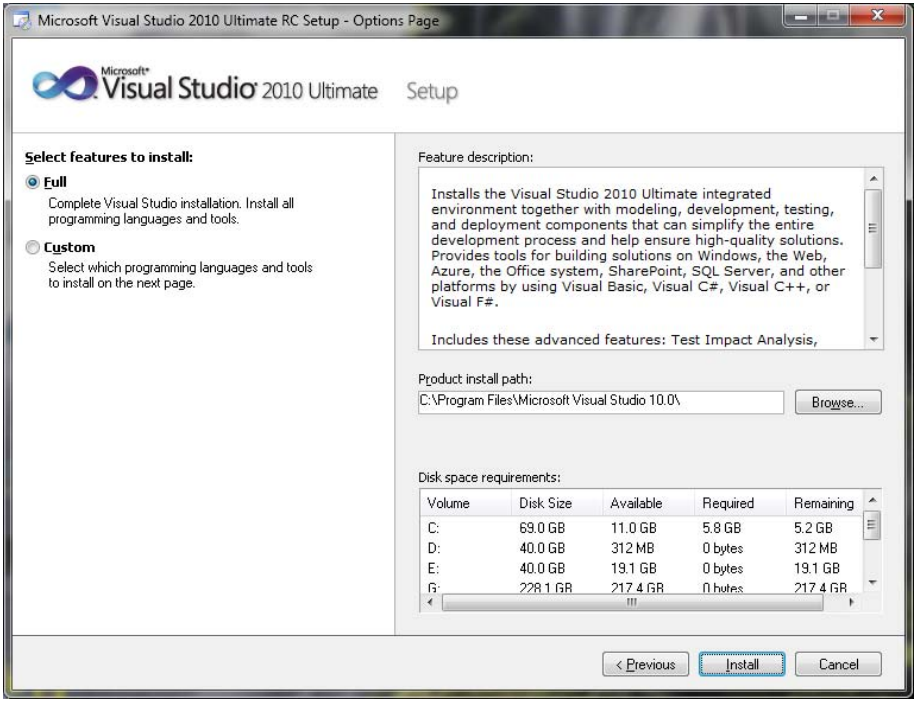


Рис. 1.4. Окно индивидуальной настройки программы Visual Studio Setup

5. В окне, показанном на рис. 1.3, вы увидите список компонентов, которые планируется установить. Кроме того, вам необходимо будет прочесть текст лицензионного соглашения с Microsoft. Текст соглашения нужно прочесть, потому что Microsoft необходимо удостовериться в том, что вы понимаете, на каких условиях вы можете пользоваться данным ПО. Условия лицензионного соглашения могут варьироваться, в зависимости от того, какой тип пакета вы приобрели, а также от конкретного региона или страны, в которой вы работаете. Прочитав условия лицензионного соглашения, вы должны будете установить флажок **I have read and accept the license terms** для продолжения установки. Далее вам потребуется ввести лицензионный ключ продукта (его можно найти на фирменной упаковке приобретенного ПО), а затем ввести свое имя. Программа-установщик автоматически введет код продукта в том случае, если вы загрузили его с сайта Microsoft Developer Network (MSDN). Нажмите кнопку **Next**, и в следующем окне вы получите возможность индивидуально настроить процесс установки продукта.
6. Окно, показанное на рис. 1.4, позволит вам выбрать между полной (Full) и выборочной (Custom) установкой. Если вы выберете опцию **Custom**, вы получите возможность индивидуально выбирать, какие компоненты VS должны быть установлены. Это — хорошая возможность отказаться от инсталляции компонентов, о которых вы точно знаете, что никогда не будете ими пользоваться.

Если вы устанавливаете VS впервые, и при этом не испытываете нехватки свободного дискового пространства, вы вполне можете пролистать список всех доступных для выбора опций и отметить их все — хотя бы для того, чтобы просто ознакомиться со всеми предлагаемыми возможностями. К процессу установки вы сможете вернуться в любое время после его завершения и вновь запустить программу Setup, чтобы внести изменения в установленную конфигурацию.

Конфигурационный экран, показанный на рис. 1.4, позволяет изменить каталог, в который будет производиться установка VS. Запомните путь к этому каталогу, потому что впоследствии именно там вы сможете найти примеры программного кода, общие сборки (common assemblies), а также множество другой информации, включая объекты, влияющие на поведение среды разработки. На данном этапе вам следует оценить объем имеющегося у вас дискового пространства и убедиться, что вам его хватит, чтобы установить все выбранные компоненты. На этом конфигурирование опций для установки будет завершено. Щелкните мышью по кнопке **Install**, и процесс установки начнется. На экране появится примерно такое окно, как на рис.1.5, отражающее степень завершенности процесса инсталляции. Небольшие галочки в этом окне будут появляться рядом с названиями успешно установленных компонентов.

1. В ходе процесса установки Visual Studio 2010 программе VS Installer потребуется перезагрузить компьютер. Когда настанет этот момент, программа установки сообщит об этом, выведя окно, показанное на рис. 1.6. Увидев это окно, закройте все приложения, с которыми вы в данный момент работаете, предварительно убедившись в том, что все важные данные сохранены, и нажмите в этом окне кнопку **Restart Now**.

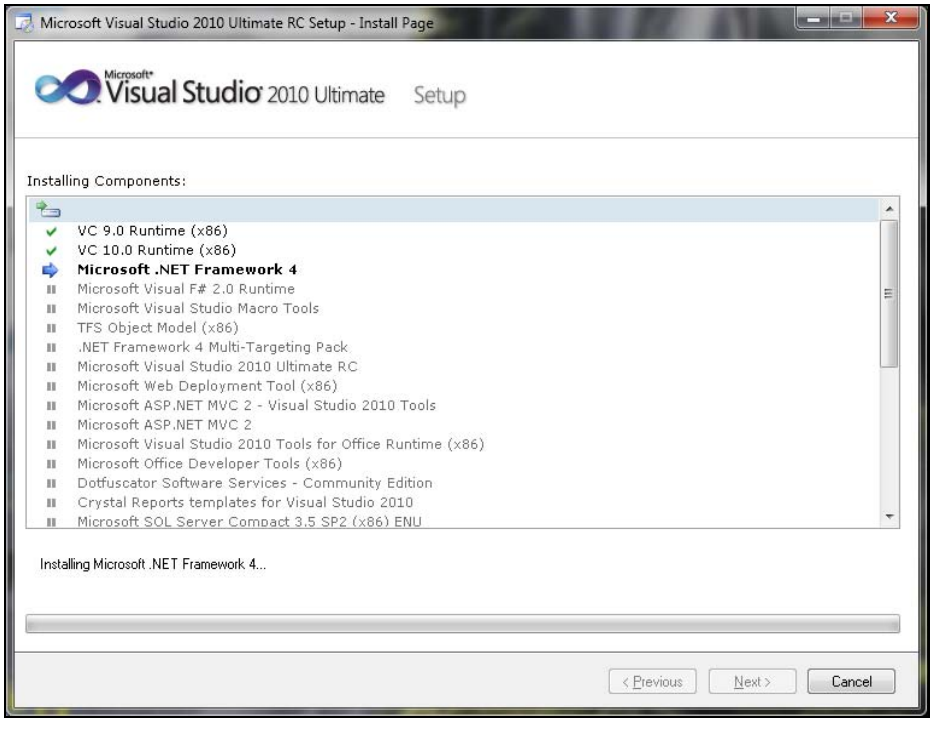


Рис. 1.5. Окно **Setup Progress**

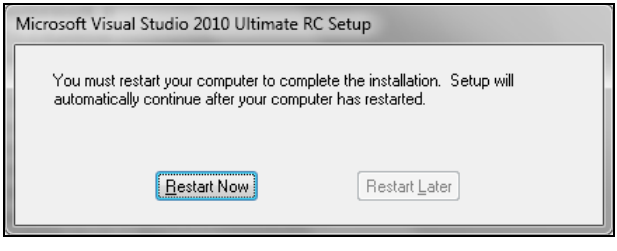


Рис. 1.6. Окно с сообщением, предлагающим перезагрузить компьютер

2. После того как процедура установки завершится успехом, на экране появится окно **Success**, показанное на рис. 1.7. Если в процессе установки произойдет какая-нибудь ошибка или сбой, то в данном окне будут даны указания, помогающие понять причину неудачи и устранить проблему.

Итак, процедура установки почти завершилась. Теперь, при желании, вы можете установить и электронную документацию, нажав в окне, показанном на рис. 1.7, кнопку **Install Documentation**. На экране вновь появится окно, которое вы уже видели в начале процесса установки (рис. 1.8). В этом окне установите и опции для инсталляции сервисных релизов (Service Releases). Это следует сделать не только потому, что сервисные релизы предлагают дополнительные функции VS, но и потому, что они включают и важные обновления безопасности.

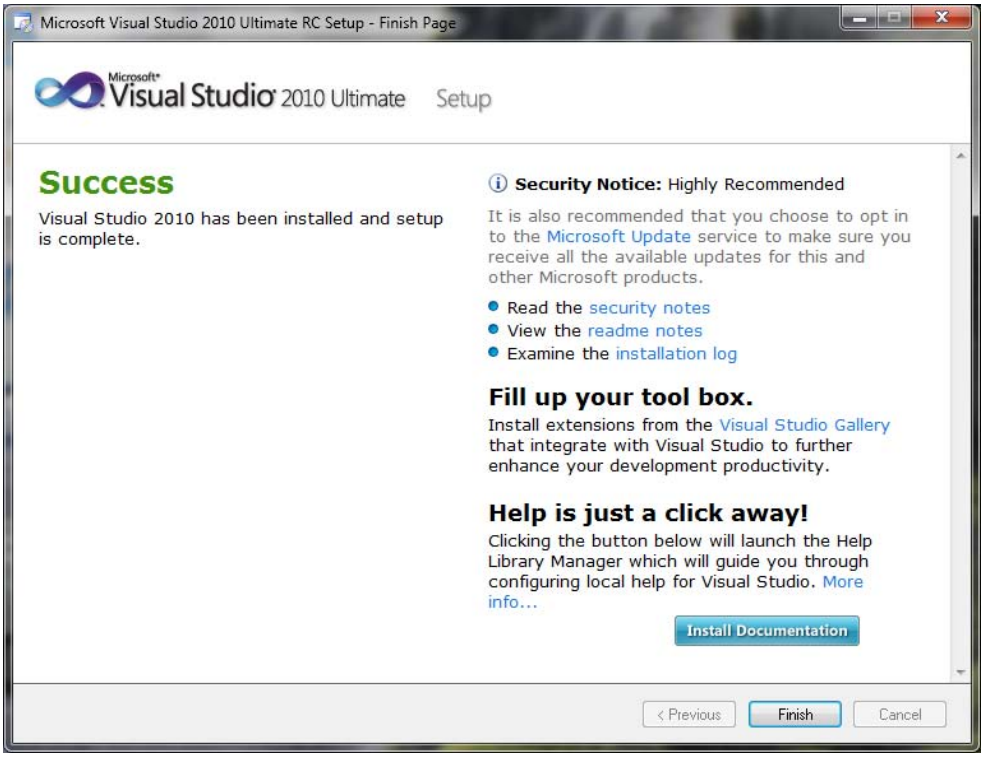


Рис. 1.7. Окно **Setup Success**

Рис. 1.8. Проверка наличия сервисных выпусков (service releases)

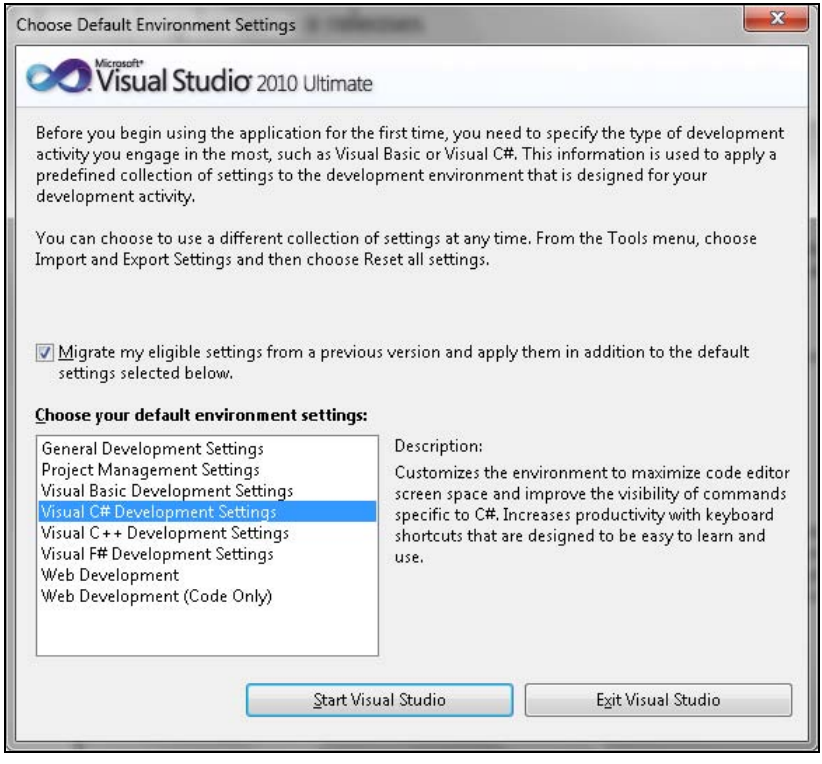


Рис. 1.9. Окно **Default Environment Settings**

Теперь вы готовы к тому, чтобы запустить VS в первый раз. При первом запуске вам потребуется выполнить еще один важный конфигурационный шаг — выбрать среду, которую вы хотите открывать по умолчанию, как показано на рис. 1.9.

Выбор настроек среды по умолчанию во многом зависит от того, на каком языке и в какой среде вы в основном будете писать свои программы. Впрочем, настройки среды не являются чем-то закостенелым и выбираемым раз и навсегда — если впоследствии вы захотите их поменять, вы всегда сможете это сделать. О том, как изменить настройки по умолчанию, будет рассказано в следующем разделе данной главы. В этой книге речь пойдет как о Visual Basic (VB), так и C#, поэтому вам, скорее всего, будет удобнее выбрать настройки, характерные для того языка, который предпочитаете лично вы. Примеры, приведенные в этой книге, будут использовать настройки для VB или для C# — в зависимости от обсуждаемой темы. Выбор настроек определяет расположение и набор окон, отображаемых по умолчанию, а также стандартные настройки, которые по умолчанию будут использоваться средой VS IDE.

Примечание

Что лучше выбрать — C# или VB? На самом деле, как C#, так и VB представляют собой первоклассные языки, поддерживающие платформу .NET. Языки как таковые ограничиваются одним лишь синтаксисом, а все дополнительные сервисы перемещены

в состав библиотеки .NET Framework Class Library, которая доступна всем поддерживаемым языкам. Между языками есть небольшое количество несущественных различий, но в реальности выбор языка зависит только от ваших личных предпочтений. На практике, знание обоих языков будет для вас преимуществом, потому что это поможет пониманию ценной информации, изложенной во множестве книг и статей, иллюстрирующих работу с платформой .NET. При этом излагаемая информация часто не зависит от используемого языка. Соответственно, совсем не рационально отказываться от чтения отличной книги или статьи о платформе .NET только потому, что приводимые там примеры написаны не на том языке, который предпочитаете лично вы.

К настоящему моменту предполагается, что вы уже успешно установили VS 2010 и настроили среду разработчика в соответствии со своим выбором и предпочтениями. Итак, откройте VS, и в следующем разделе мы приступим к рассмотрению высокоуровневых элементов интерфейса, доступных на странице **Start**.

Навигация по среде Visual Studio 2010

В этом разделе мы бегло рассмотрим высокоуровневый интерфейс VS и опишем все возможности, которые становятся вам доступными при первом запуске Visual Studio 2010. То, что вы увидите на экране, называется интегрированной средой разработчика Visual Studio (Integrated Development Environment, IDE).

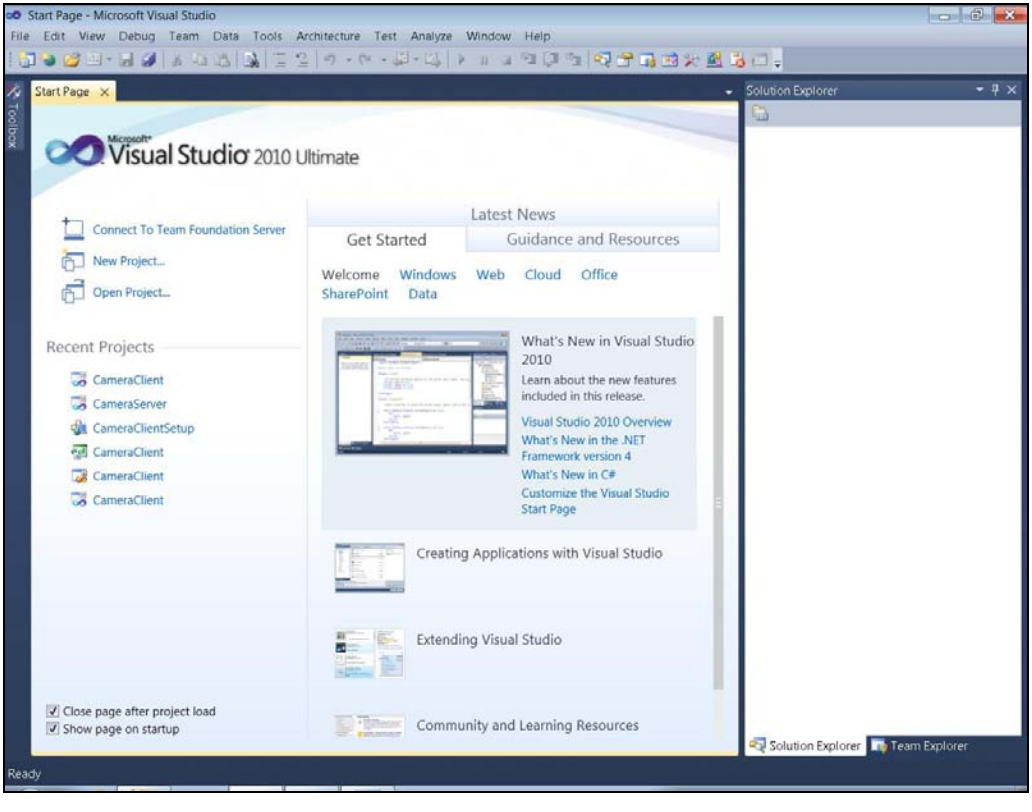


Рис. 1.10. Начальный экран Visual Studio 2010

Даже если вы — опытный программист, уже работавший с одной из предыдущих версий Visual Studio, просмотрите эту информацию хотя бы бегло, потому что благодаря ей вы быстрее научитесь ориентироваться в IDE и находить нужные вам функции. Кроме того, знание функций, доступных по умолчанию, позволит вам эффективнее различать между стандартными функциональными возможностями и контекстно-зависимыми функциями программных компонентов, над которыми вы работаете.

На рис. 1.10 показано, как выглядит среда VS сразу же после первого запуска. Это относится ко всем частям экрана, на примере которых мы и будем основываться, поясняя, как организована среда IDE. В следующем описании мы ассоциируем каждую функцию с именем, чтобы облегчить процесс поиска нужных функций, чтобы вы уже знали, где их искать, когда впоследствии будет упоминаться имя той или иной из них. В последующих нескольких разделах будут описаны все части экрана, показанного на рис. 1.10.

Меню

В верхнем левом углу экрана, показанного на рис. 1.10, вы увидите начало строки меню (menu bar), начинающейся с пунктов **File**, **Edit**, **View**, **Tools** и т. д. Строка меню представляет собой стандартный элемент интерфейса любого Windows-приложения. Помимо стандартного управления файлами, меню **File** представляет собой стандартную "начальную точку", откуда вы начинаете работать с любым проектом. Кроме того, меню **File** предоставляет возможности быстрого доступа к недавно открывавшимся файлам и проектам.

Меню **Edit** предоставляет стандартные возможности редактирования: функции вырезки в буфер (cut), копирования (copy) и вставки (paste). Кроме того, оно позволяет установить закладку (bookmark), чтобы обеспечить быстрый доступ к нужному фрагменту кода и быструю навигацию по большим файлам.

Не стоит жалеть времени на ознакомление с опциями, доступными через меню **View**, хотя бы для того, чтобы посмотреть, какие функции есть в вашем распоряжении. Но если вы только приступаете к изучению Visual Studio и являетесь начинающим программистом, то не стоит "зацикливаться" на этом сейчас — в дальнейшем мы обратим на эти функции особое внимание, обсудим предназначение каждой из них и объясним, в какой ситуации конкретное представление (view) будет особенно полезным.

Меню **View** позволяет быстро получить доступ ко всем инструментальным окнам, имеющимся в VS. Кроме того, меню **View** содержит пункт **Other Windows**, куда включены дополнительные окна приложений, которые могут оказаться удобными при написании новых программ.

Меню **Tools** предоставляет множество различных функциональных возможностей: например, с его помощью вы можете подключить отладчик, чтобы просмотреть работу ваших программ пошагово, проходя одну строку за другой, подключиться к базе данных, установить добавочные модули, макросы и выполнить множество других действий. Одна из важнейших опций меню **Tools** так и называется — **Options**, так как она предоставляет доступ к сотням и сотням настроек VS, позволяющих индивидуально настроить среду разработки.

Меню **Test** можно использовать, чтобы выполнить модульное тестирование вашей новой программы, по одному модулю единовременно. Именно в этом меню различные редакции VS предоставляют доступ к различным наборам инструментов тестирования.

Пункты меню **Analyze**, **Architecture** и **Team** предоставляют расширенные наборы инструментов, предназначенных для повышения производительности приложений, работы с их архитектурными компонентами, а также для их интеграции с Microsoft Team Foundation Server[2].

Меню **Windows** и **Help** имеют предназначение, стандартное для большинства приложений Windows. Так, меню **Windows** позволяет манипулировать различными окнами VS, а с помощью меню **Help** можно читать техническую документацию по VS.

РЕКОМЕНДАЦИЯ

Многие элементы меню ассоциированы с клавиатурными комбинациями, которые предоставляют более быстрый доступ к нужной функции, чем ее вызов через меню. Если вы заинтересованы в том, чтобы запомнить как можно больше клавиатурных комбинаций, то обратите внимание на то, что при открытии меню многие клавиатурные комбинации, ассоциированные с тем или иным пунктом меню, приводятся и в самом меню, справа от имени соответствующей команды. Например, чтобы открыть окно **Solution Explorer**, вы можете выбрать соответствующую команду через меню **View**, а можете нажать клавиатурную комбинацию <CTRL>+<W>, <S>.

Инструментальная панель (Toolbar)

Вернемся к рис. 1.10. Под строкой меню на этой иллюстрации находится инструментальная панель. На инструментальной панели расположены кнопки, предоставляющие быстрый доступ к наиболее часто используемым функциям. Набор кнопок инструментальной панели представляет собой подмножество команд, которые доступны через меню. Инструментальные панели контекстно-чувствительны, и набор доступных через них опций меняется в зависимости от той работы, которую вы выполняете через VS в данный момент. Любую инструментальную панель можно отобразить или скрыть. Делается это с помощью команд меню **View** | **Toolbars**.

Кроме того, инструментальные панели можно настраивать по вашему выбору. Для этого выполните щелчок правой кнопкой мыши по нужной панели, прокрутите появившееся контекстное меню и выберите из него опцию **Customize**. После этого на экране появится окно настройки инструментальной панели, где вы сможете добавить на нее кнопки быстрого вызова функций, которыми часто пользуетесь именно вы, но которые не были включены в набор функций предлагаемых для данной инструментальной панели по умолчанию.

---

[2] Team Foundation Server — это ПО Microsoft, предназначенное для совместной работы над проектами, доступное как в виде отдельного приложения, так и в виде серверной платформы для Visual Studio. Подробнее о Team Foundation Server см.: **http://ru.wikipedia.org/wiki/Team_Foundation_Server**, **http://cmcons.com/articles/microsoft/**. — *Прим. перев.*

Рабочая область (Work Area)

Еще раз вернемся к рис. 1.10. В центре окна, представленного на иллюстрации, находится стартовая страница (Start page). Это — та самая область, которой вы можете пользоваться для введения кода ваших программ и работы с визуальными редакторами. Стартовая страница разделена на две области: область управления проектами (project management) и информационную зону (information). Расположенная слева область управления проектами предоставляет возможность быстро начать работу над новым проектом или выбрать из списка один из проектов, с которыми вы работали недавно. Расположенная справа информационная область содержит ресурсы, помогающие новичку начать работу с VS, например, ссылки на Web-сайт Microsoft, пошаговые инструкции, помогающие разобраться с новыми возможностями, а также постоянно обновляемую вкладку, на которой можно прочесть последние новости сообщества разработчиков Microsoft.

Инструментальный набор (Toolbox)

В крайней правой части окна, показанного на рис. 1.10, располагается вертикальная вкладка, озаглавленная **Toolbox**, которая содержит контекстно-чувствительный список элементов управления (controls). Эти элементы управления можно перетаскивать мышью в текущую рабочую область, чтобы включить их в состав разрабатываемой программы.

Термин "контекстно-чувствительный" означает, что некоторые элементы списка будут отображены или скрыты, в зависимости от того, где в последний раз был выполнен щелчок мышью, или от контекста, в котором вы в данный момент работаете. Например, вы можете вводить программный код или создавать/редактировать новую Web-страницу. Если вы еще не начали ни над чем работать, то страница **Toolbox** будет пустой.

Окно Solution Explorer

Окно **Solution Explorer**, находящееся на правой границе страницы **Start** (рис. 1.10), отображает все ваши решения (solutions), проекты (projects) и элементы этих проектов. Именно здесь можно найти и требуемым образом организовать все файлы и настройки, принадлежащие проекту. На рис. 1.10 окно **Solution Explorer** пусто, потому что еще не открыто ни одного решения и не создано ни одного проекта. Если вы закроете это окно, а затем вам потребуется вновь его открыть, то это можно сделать через меню **View**, которое уже обсуждалось чуть ранее.

Строка состояния

В самой нижней части экрана, показанного на рис. 1.10, находится строка состояния (status bar), которая сообщает вам обо всем, что происходит в среде VS в настоящий момент. На рис. 1.10 в строке состояния выведено сообщение **Ready**, которое говорит о том, что среда VS готова к работе. В ходе вашей работы с VS строка состояния будет изменяться в зависимости от контекста, отображая информацию, относящуюся к задаче, выполняемой на текущий момент. Например, если

вы работаете с редактором кода, то в строке состояния будет отображаться текущая строка и позиция, в которой находится курсор, а также другая информация о статусе редактора.

Управление окнами VS

Внимательно взгляните на экран VS, показанный на рис. 1.10. Обратите внимание, что все окна в рабочей области — **Toolbox**, **Start** и **Solution Explorer** — снабжены строками заголовка (title bars). В строке заголовка окна присутствуют три значка: **Window Position** (треугольная стрелка, направленная вниз,), **Maximize/Restore Down** (окно,) и **Close** (косой крест,). На рис. 1.11 показано окно **Solution Explorer**, где в строке заголовка эти три кнопки находятся на правой границе. Значок **Window Position** позволяет изменить расположение и состояние окна, предоставляя следующие опции: **Dock**, **Float**, **Dock As Tabbed Document**, **Auto Hide** и **Hide**. Вы можете развернуть окно на всю рабочую область или изменить его размеры и позволить ему свободно перемещаться ("плавать") по всей рабочей области с помощью кнопки **Maximize/Restore Down**. Если окно пристыковано (docked), то кнопка **Maximize/Restore Down** изменяет свой вид на значок с изображением кнопки (), которая позволяет развернуть и свернуть окно. Кнопка **Close** позволяет закрыть окно. В последующих нескольких разделах будет описано, как пользоваться этими кнопками, чтобы открывать, закрывать, сворачивать, разворачивать, пристыковывать, отстыковывать окна и изменять вид окна на окно с вкладками.

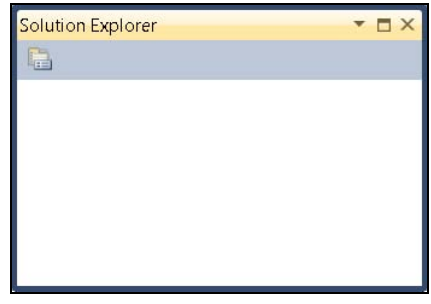


Рис. 1.11. Значки строки заголовка окна

Распахивание и сворачивание окон

Если задержать курсор мыши над вкладкой **Toolbox**, то она развернется и отобразит список из трех значков, доступных и на инструментальной панели окна **Toolbox**: **Window Position** (треугольная стрелка, направленная вниз), **Hide** (кнопка) и **Close** (косой крест). Вид развернутого окна **Toolbox** представлен на рис. 1.12. Изображение кнопки **Hide** находится чуть в стороне, и на левой границе по-прежнему присутствует вертикальная вкладка.

Если вы переместите курсор с поля окна **Toolbox**, оно снова свернется и примет вид вкладки, расположенной на левой границе экрана.

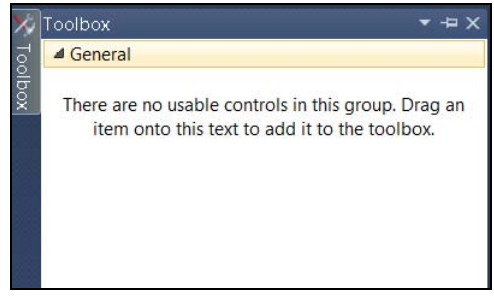


Рис. 1.12. Развернутое окно **Toolbox**

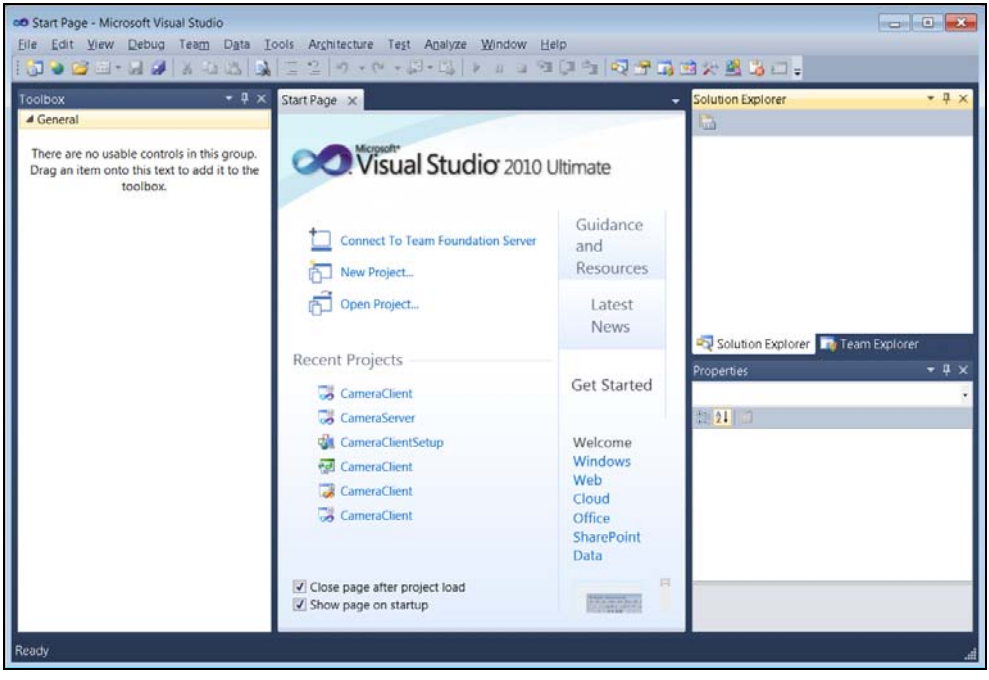


Рис. 1.13. Закрепленное окно **Toolbox**

Любое свернутое окно, например, окно **Toolbox**, можно развернуть, а затем щелкнуть мышью по значку с изображением канцелярской кнопки (**Hide**), в результате чего вид окна станет подобен виду окна **Solution Explorer**. На рис. 1.13 показано "закрепленное" окно; кнопка на значке **Hide** (над всплывающей подсказкой **Auto Hide**) примет вертикальное положение, а вкладка **Toolbox** на левой границе экрана исчезнет.

Щелчок мышью по значку **Hide** в любом развернутом окне свернет это окно и отобразит вкладку, аналогичную вкладке **Toolbox**. Еще один способ сворачивания окна состоит в том, чтобы выбрать опцию **Auto Hide** из меню **Window Position** (треугольная стрелка, направленная вниз).

Пристыковка окон

Опция **Dock** позволяет отображать окно в пристыкованном (docked) положении, аналогичном положению окна **Solution Explorer** на рис. 1.10. Пристыкованное окно можно переместить в любую позицию в пределах рабочей области. Чтобы переместить пристыкованное окно, выделите его строку заголовка и перетаскивайте его мышью в любое нужное вам положение. На рис. 1.14 показан экран VS во время перетаскивания окна.

Как показано на рис. 1.14, в рабочей области появится набор значков, указывающих, куда можно переместить окно. Тень окна будет показывать, где разместится окно, когда вы переместите его в зону пристыковки. Отпустите курсор мыши, и окно переместится из старой зоны пристыковки в новую.

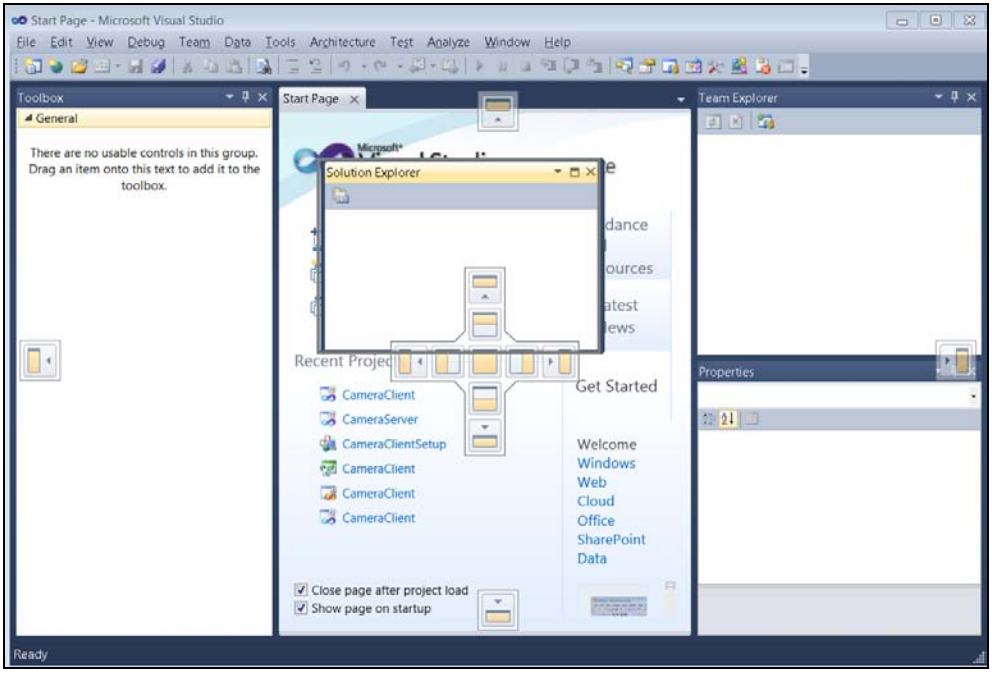


Рис. 1.14. Перемещение окна в новую область пристыковки

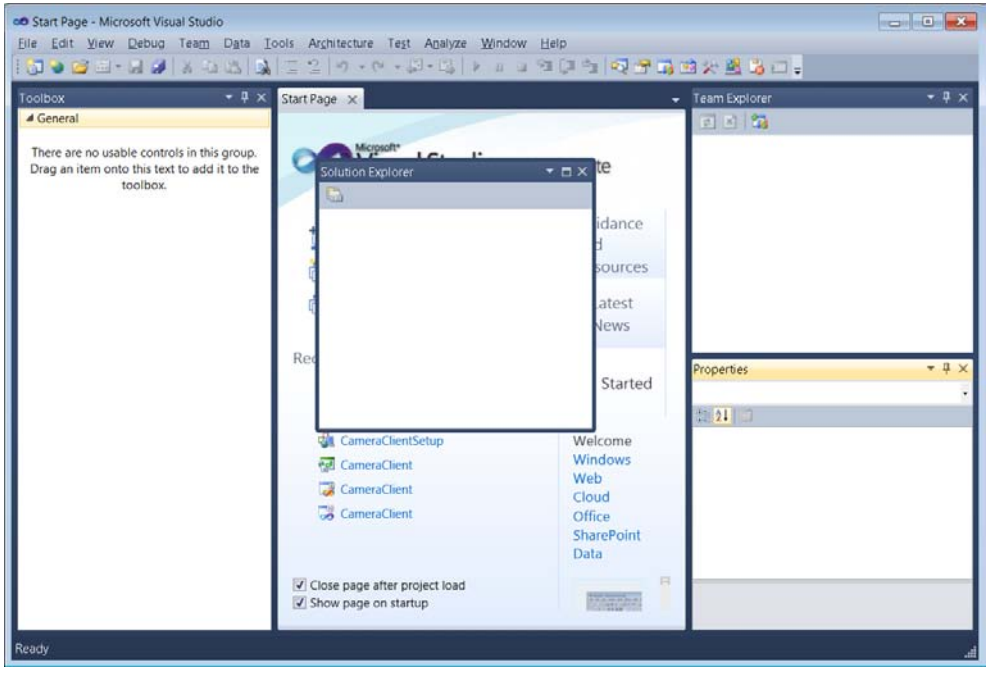


Рис. 1.15. Плавающее окно **Solution Explorer**

Плавающие окна

Опция **Float** позволяет окнам появляться в любой области VS IDE, и вы можете переместить туда их все. Чтобы переместить плавающее окно, выделите его строку заголовка и, действуя стандартным методом перетаскивания (drag and drop), перемещайте его в нужную позицию. Когда окно окажется там, где вы хотите его видеть, отпустите курсор мыши. Как вариант, вы можете выполнить двойной щелчок мышью на заголовке окна. На рис. 1.15 показано окно **Solution Explorer**, плавающее поверх остальных окон.

Окна с вкладками

Примером использования опции **Dock As Tabbed Document** является страница **Start**. Любое окно, для которого выбран режим **Dock As Tabbed Document**, будет отображаться как новая вкладка в рабочей области, как и все остальные окна, настроенные таким образом. Например, если вы выберете режим **Dock As Tabbed Document** для окна **Toolbox**, то оно станет отображаться в виде документа на вкладке в той же группе, что и окно **Start**, как показано на рис. 1.16.

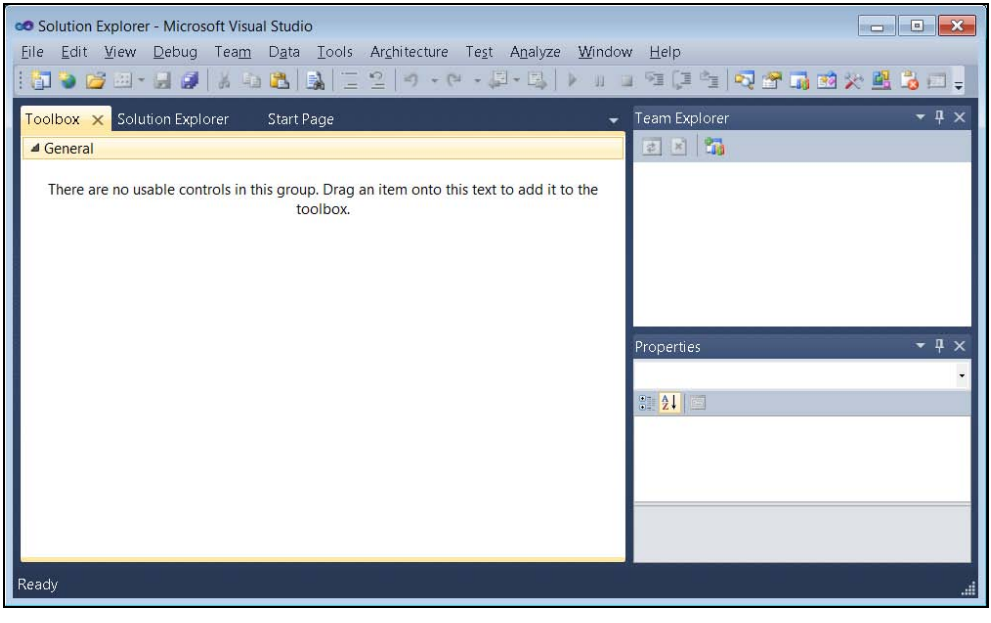


Рис. 1.16. Окно, представляющее документы на вкладках

Совет

Разместите окна таким образом, который является для вас наиболее удобным. На практике, вероятно, вам не захочется, чтобы окно отображалось как документ на вкладке. В последующих главах этой книги будут приведены примеры, в которых вам будет предлагаться перетаскивать мышью элементы из окна **Toolbox** на страницу визуального редактора, которая тоже представлена в виде документа на вкладке.

Перетаскивание элементов из одного документа на вкладке в другой — довольно неудобный процесс. Существуют различные способы работы с окнами в VS, и вы, немного поэкспериментировав, сможете сами выбрать то, что лучше всего подходит именно вам.

Чтобы изменить вид окна, представленного как документ на вкладке, выберите соответствующую вкладку и мышью перетащите ее за пределы окна со вкладками. После этого документ на вкладке превратится в плавающее окно.

Открытие и закрытие окон

Щелчок мышью по значку **Close** в строке заголовка окна приводит к закрытию этого окна. Еще один способ закрыть окно заключается в том, чтобы выбрать опцию **Hide** из меню, выводимого при нажатии кнопки **Window Position.**

Способ повторно открыть закрытое окно зависит от типа этого окна: VS или **Project Item**. Если это окно является окном VS, можно воспользоваться меню **View** и выбрать из него опцию, соответствующую нужному вам окну. Есть и альтернативный вариант — вы можете воспользоваться для открытия окон клавиатурными комбинациями. Эти клавиатурные комбинации отображаются и в меню **View** справа от названия нужного вам окна.

Окна для элементов проекта, отображающихся в окне **Solution Explorer**, представляют собой еще один тип окон. В большинстве случаев для открытия элемента проекта достаточно найти его название в составе соответствующего проекта, отображаемого в окне **Solution Explorer**, и выполнить на нем двойной щелчок мышью. В некоторых случаях вам необходимо сначала выполнить щелчок правой кнопкой мыши на имени нужного проекта в окне **Solution Explorer**, а затем выбрать из раскрывшегося контекстного меню нужную опцию, но эти ситуации мы рассмотрим чуть далее в последующих главах книги.

Итак, вы научились манипулировать окнами и модифицировать вид среды VS в соответствии с личными потребностями. Впрочем, иногда бывают и такие ситуации, когда вам потребуется вернуть настройки среды VS в их исходное состояние. О том, как это делается, будет рассказано в следующем разделе.

Модификация настройки среды после установки

Причины, по которым вам может потребоваться вернуть все настройки среды в тот первозданный вид, который они имеют сразу же после завершения установки, могут быть разными. Например, это может быть простое желание вернуть для всех настроек значения по умолчанию, импортировать общие настройки, созданные другим разработчиком, или же просто переключать настройки, переходя от проекта к проекту. В данном разделе будет показано, как добиться каждой из этих целей.

Запустите VS и выберите из меню команды **Tools** | **Import And Export Settings.** На экране появится окно мастера импорта и экспорта настроек (**Import and Export Settings Wizard**), показанное на рис. 1.17. В окне **Import and Export Settings Wizard** имеются опции **Export selected environment settings**, **Import selected**

environment settings и **Reset all settings**. В последующих нескольких разделах каждая из этих опций будет рассмотрена более подробно.

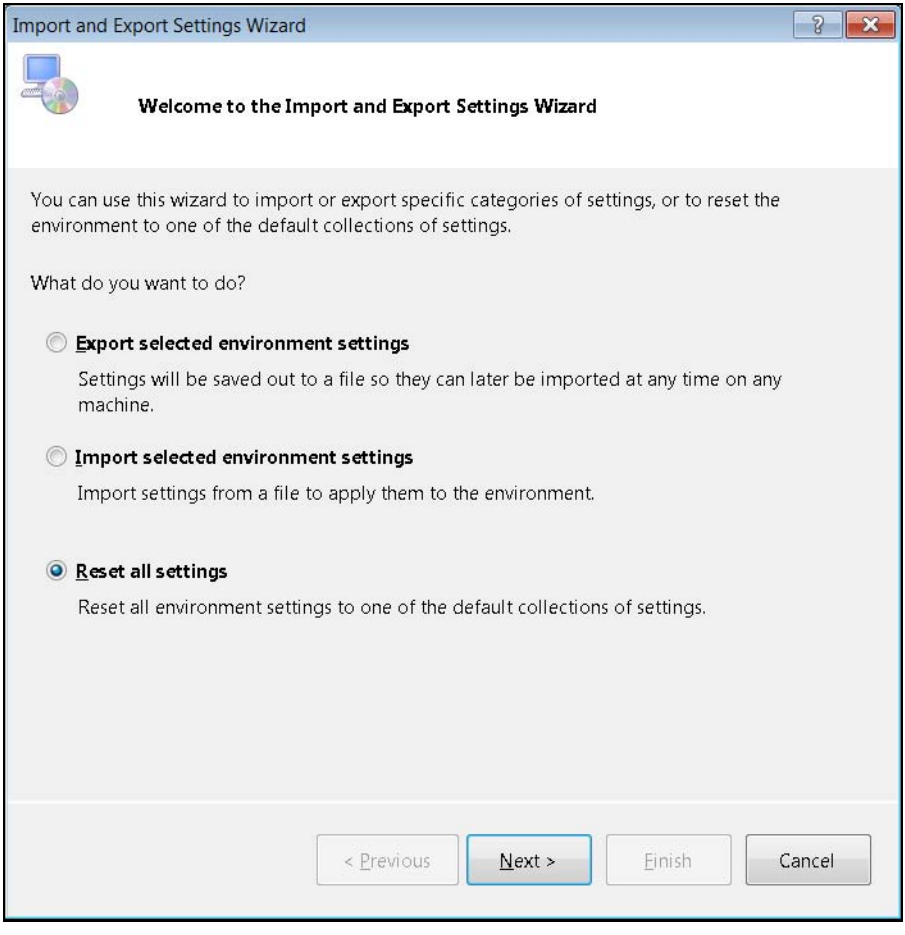


Рис. 1.17. Окно **Import and Export Settings Wizard**

Экспорт выбранных параметров настройки среды

Начнем обсуждение с операции экспорта, которая позволит вам предоставить ваши настройки среды в распоряжение других разработчиков. Это также может оказаться полезным, если вы планируете внести в свои настройки существенные изменения и хотите сохранить резервную копию своих настроек, чтобы впоследствии иметь возможность при необходимости быстро к ним вернуться. Чтобы выполнить операцию экспорта, выберите опцию **Export selected environment settings** (см. рис. 1.17), а затем нажмите кнопку **Next**. На экране появится окно **Choose Settings to Export**, показанное на рис. 1.18, в котором вы сможете выбрать экспортируемые настройки.

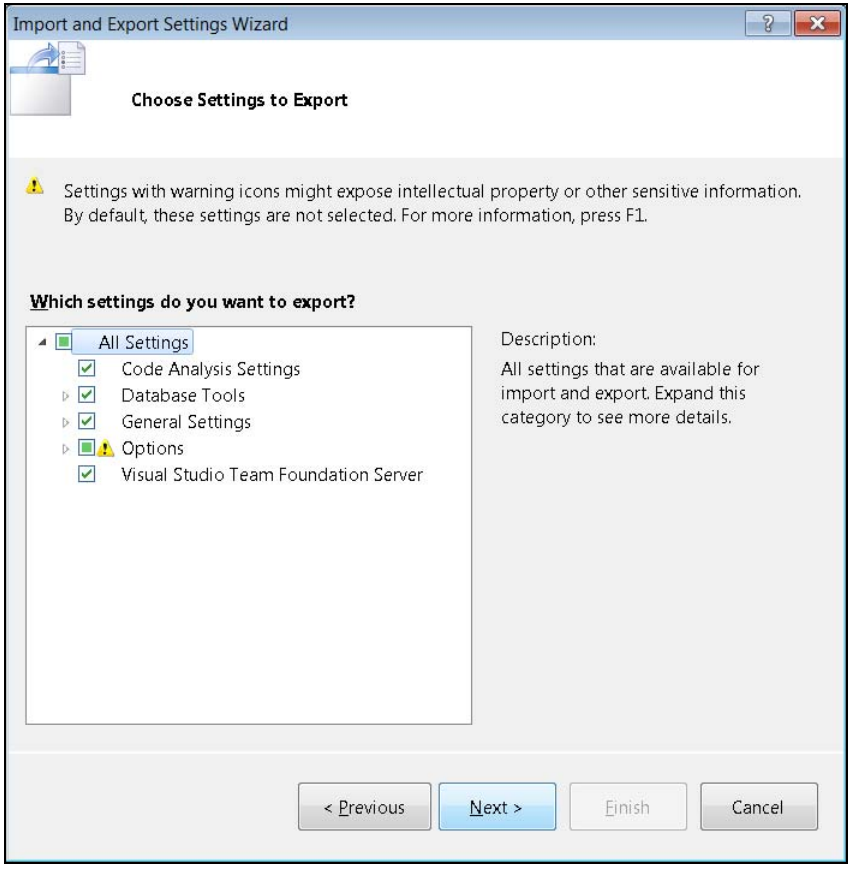


Рис. 1.18. Окно **Choose Settings to Export**

Вы увидите в этом окне целое "дерево" настроек, из которого можно выбрать именно те настройки, которые вам требуется экспортировать.

Значком предупреждения в виде восклицательного знака на фоне желтого треугольника ( ) будут помечены настройки, которые не рекомендуются к экспорту по личным мотивам и по соображениям безопасности. Как правило, настройки, помеченные таким значком, имеют отношение к полным путям к системным файлам или таким объектам за пределами среды VS, которые обычно не предоставляются в общий доступ.

После того как вы выберете все подлежащие экспорту настройки, нажмите кнопку **Next**, и на экране появится окно **Name Your Settings File**, показанное на рис. 1.19.

Два текстовых поля в окне, показанном на рис. 1.19, предназначены для ввода имени файла с экспортированными настройками и пути к каталогу, в котором этот файл должен быть сохранен. Обратите внимание, что стандартное имя файла, предлагаемое по умолчанию, включает текущую дату. Это может оказаться полезным, если вы часто экспортируете настройки — тогда при импорте вы сможете быстро найти нужный файл. Нажмите кнопку **Finish**, и операция экспорта будет завершена. Об этом вам сигнализирует окно **Export Complete**, показанное на рис. 1.20.

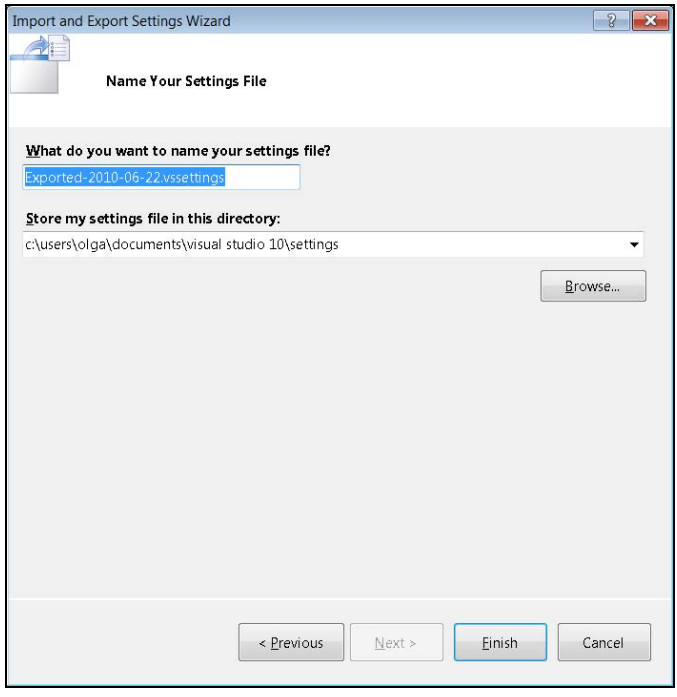


Рис. 1.19. Диалоговое окно **Name Your Settings File**

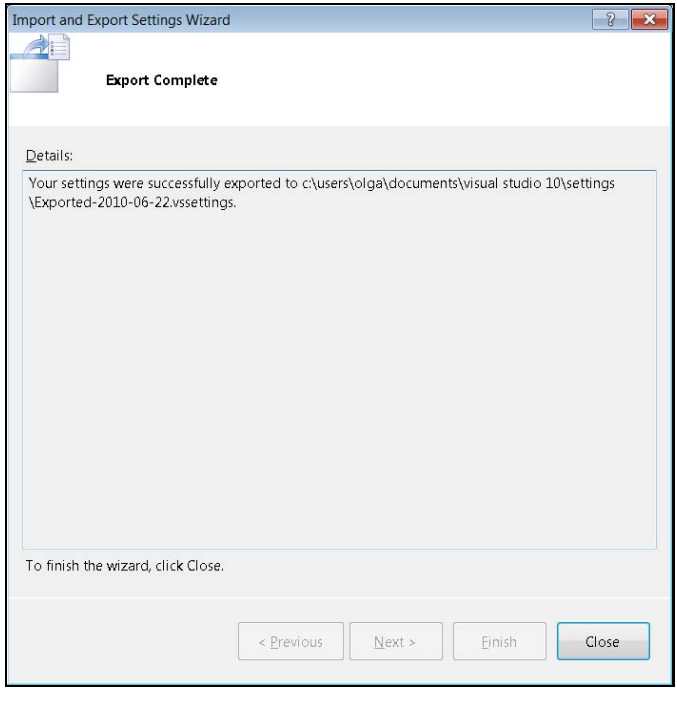


Рис. 1.20. Диалоговое окно **Export Complete**

Щелкните мышью по кнопке **Close**, чтобы закрыть это окно. Теперь, когда в вашем распоряжении есть экспортированный файл с настройками IDE, вы в любой момент сможете вернуться к сохраненным настройкам, просто импортировав этот файл. Кроме того, данный файл можно предоставить в распоряжение других разработчиков (это важно, если вы работаете в команде над крупным проектом). О том, как это делается, будет рассказано в следующем разделе.

Импорт сохраненных настроек среды разработчика

Импорт сохраненных параметров настройки выполняется, когда требуется быстро вернуться к конкретным ранее сохраненным настройкам среды разработчика, импортировать настройки, сохраненные другим разработчиком, или же изменить некоторые параметры для проекта, над которым вы работаете в данный момент.

Чтобы выполнить операцию импорта, откройте среду VS, затем выберите из меню команды **Tools | Import and Export Settings**. Когда на экране появится окно **Import and Export Settings Wizard**, показанное на рис. 1.17, выберите опцию **Import selected environment settings** и нажмите кнопку **Next**. На экране появится окно **Save Current Settings**, показанное на рис. 1.21.

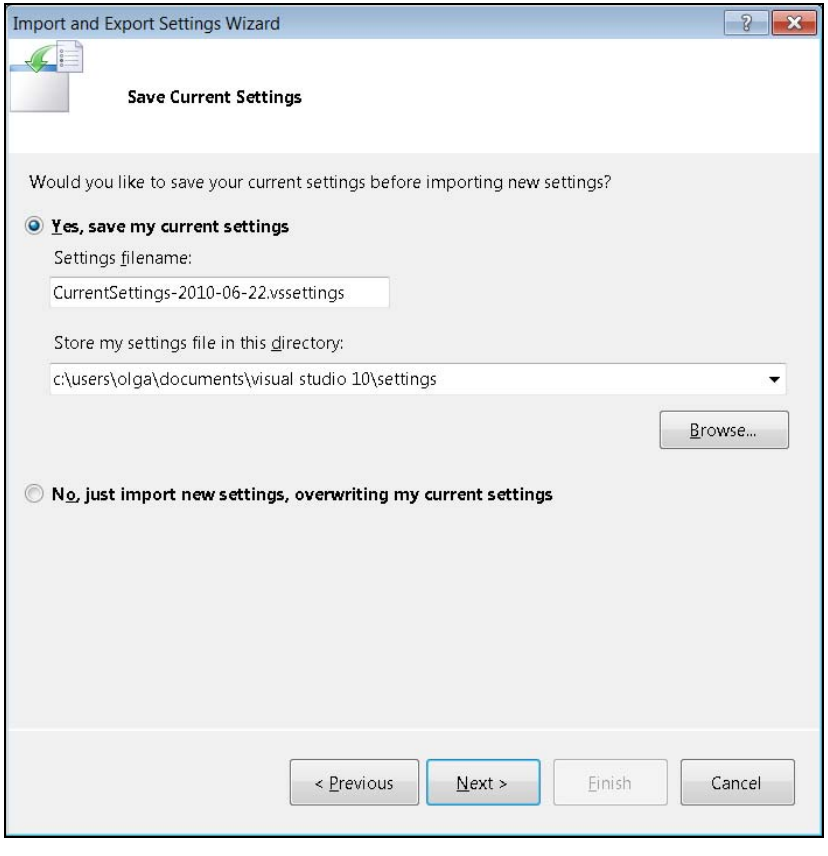


Рис. 1.21. Диалоговое окно **Save Current Settings**

РЕКОМЕНДАЦИЯ

В Интернете вы можете найти несколько доступных для загрузки цветовых схем, предназначенных для Visual Studio. На момент написания данной книги, цветовые схемы для Visual Studio можно было скачать с сайта **http://winterdom.com/2007/11/vs2008colorschemes**. Эти схемы разрабатывались в расчете на Visual Studio 2008, но пригодны и для применения с Visual Studio 2010.

Диалоговое окно **Save Current Settings** позволяет предварительно выполнить экспорт (что равносильно созданию резервной копии) ваших текущих настроек, прежде чем приступать и импорту новых. Если вы выполните резервное копирование, то впоследствии вы в любой момент сможете вернуться к заранее сохраненному набору настроек, если что-то в новом наборе вас не устроит. Впрочем, вы можете и отказаться от опции экспорта. Нажмите кнопку **Next**, и на экране появится окно **Choose a Collection of Settings to Import** (рис. 1.22).

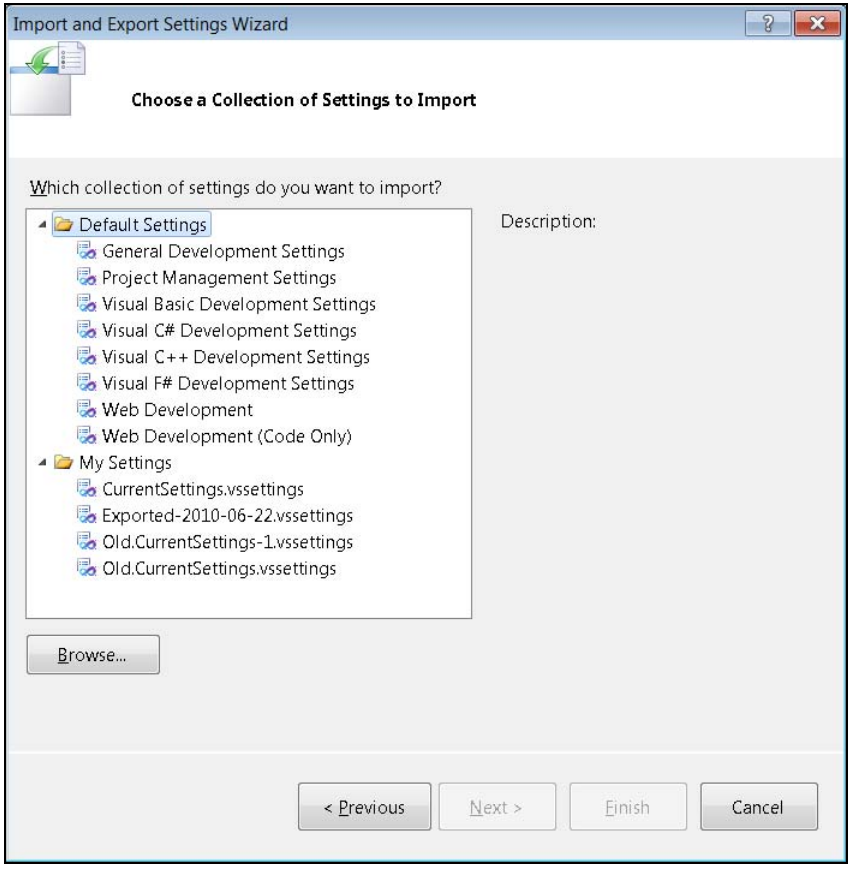


Рис. 1.22. Окно **Choose a Collection of Settings to Import**

Как показано на рис. 1.22, вы можете импортировать некоторые из предопределенных настроек, являющихся частью VS. Эти настройки можно найти в составе

ветви **Default Settings**. Кроме того, вы можете импортировать и индивидуальные настройки, которые находятся в составе ветви **My Settings**. Индивидуальные настройки включают в свой состав текущие настройки, плюс любые другие настройки, которые вы сохранили в каталоге по умолчанию, как было показано на рис. 1.19 и 1.21.

При желании, по вашему выбору вы можете нажать кнопку **Browse** и перейти в каталог, где расположен экспортированный файл с настройками. Выбрав файл с настройками, щелкните по кнопке **Next**, и на экране появится окно **Choose Settings to Import**, показанное на рис. 1.23.

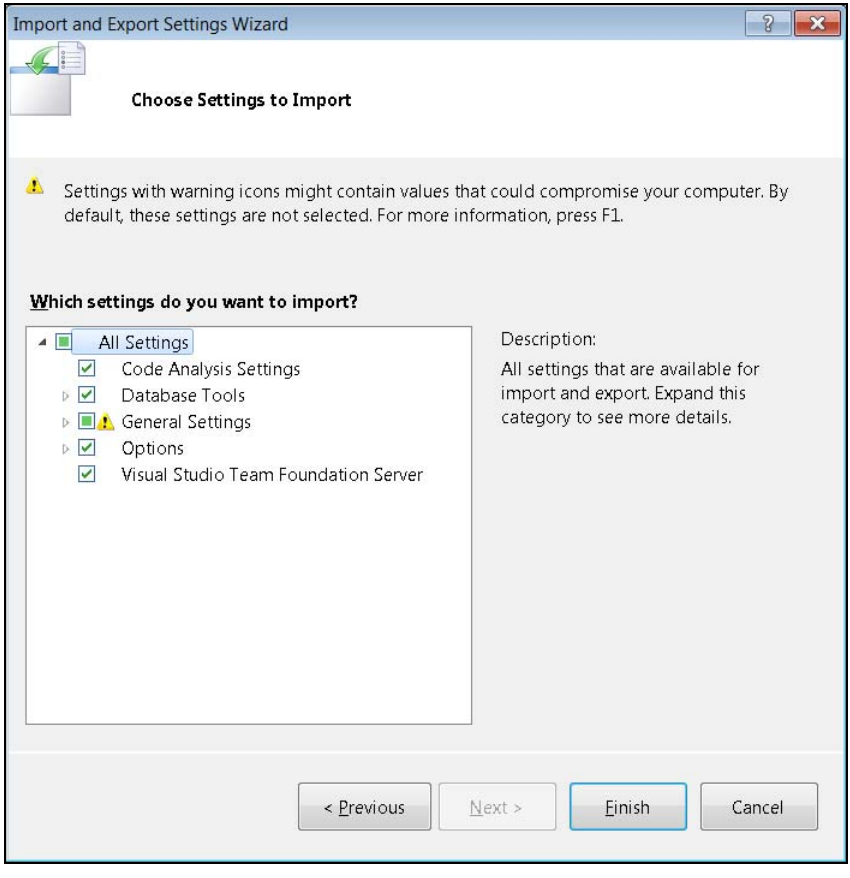


Рис. 1.23. Окно **Choose Settings to Import**

Окно **Choose Settings to Import** позволяет вам выбирать только те настройки, которые вам требуются, и импортировать их в вашу среду. Таким образом, операция импорта позволит обновить только те настройки, которые были выбраны (см. рис. 1.23).

Все остальные текущие настройки, которые не были выбраны (см. рис. 1.23), останутся неизменными. Чтобы начать операцию импорта, щелкните по кнопке

Finish. Когда операция импорта завершится, вы увидите окно **Import Complete**, показанное на рис. 1.24.

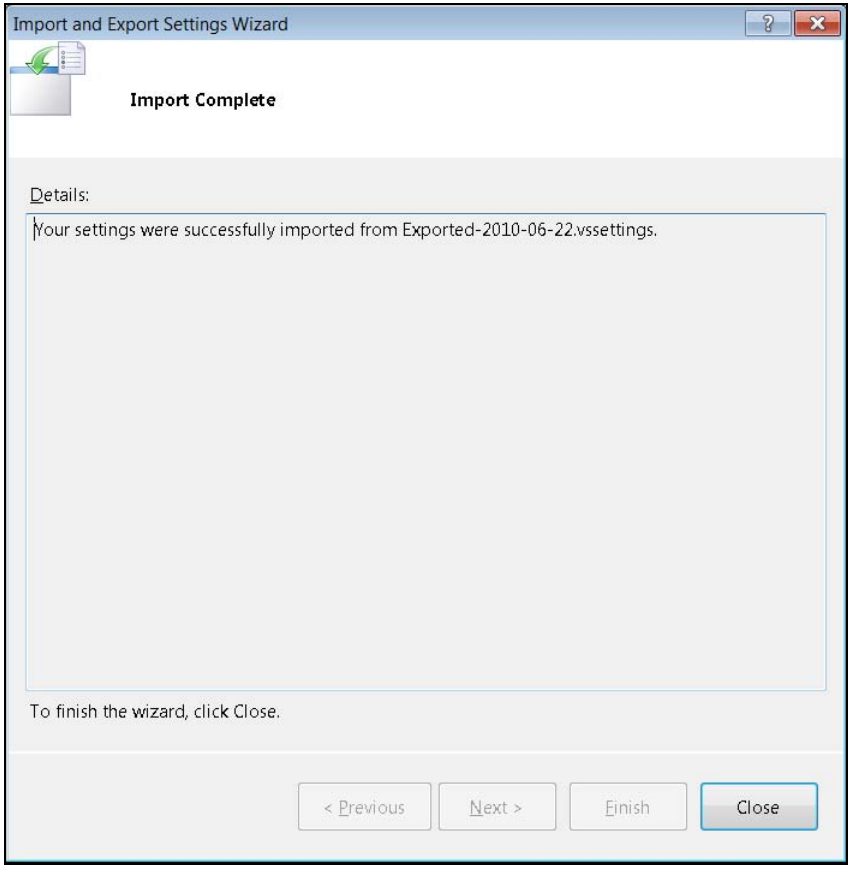


Рис. 1.24. Окно **Import Complete**

Так как операция импорта завершена, окно можно закрыть, нажав кнопку **Close**. Еще одна из возможных опций, **Reset**, позволяющая вернуть все настройки VS к значениям, предлагаемым по умолчанию, будет описана в следующем разделе.

Сброс настроек к стандартным значениям

Сброс значений параметров настройки VS к стандартным значениям, предлагаемым по умолчанию, может потребоваться, если вам нужно быстро вернуть среду VS в исходное состояние или если вам приходится переключаться между стандартными настройками VS. Например, в этой книге часто будет производиться переключение между наборами стандартных параметров по умолчанию для двух основных языков программирования, рассматриваемых в ней — VB и C#. Таким образом, будет гарантироваться нужный набор настроек для каждого из обсуждаемых языков. Чтобы выполнить переключение к набору стандартных параметров

по умолчанию, выберите из меню команды **Tools** | **Import And Export Settings**. В результате появится уже знакомое нам окно **Import and Export Settings Wizard**, показанное на рис. 1.17.

Чтобы выполнить переход к стандартным параметрам по умолчанию, выберите в этом окне опцию **Reset All Settings** и нажмите кнопку **Next**. На экране появится окно **Save Current Settings**, выглядящее в точности так, как было показано на рис. 1.21. Выберите нужную вам опцию сохранения и нажмите кнопку **Next**. На экране появится окно **Choose a Default Collection of Settings**, показанное на рис. 1.25.

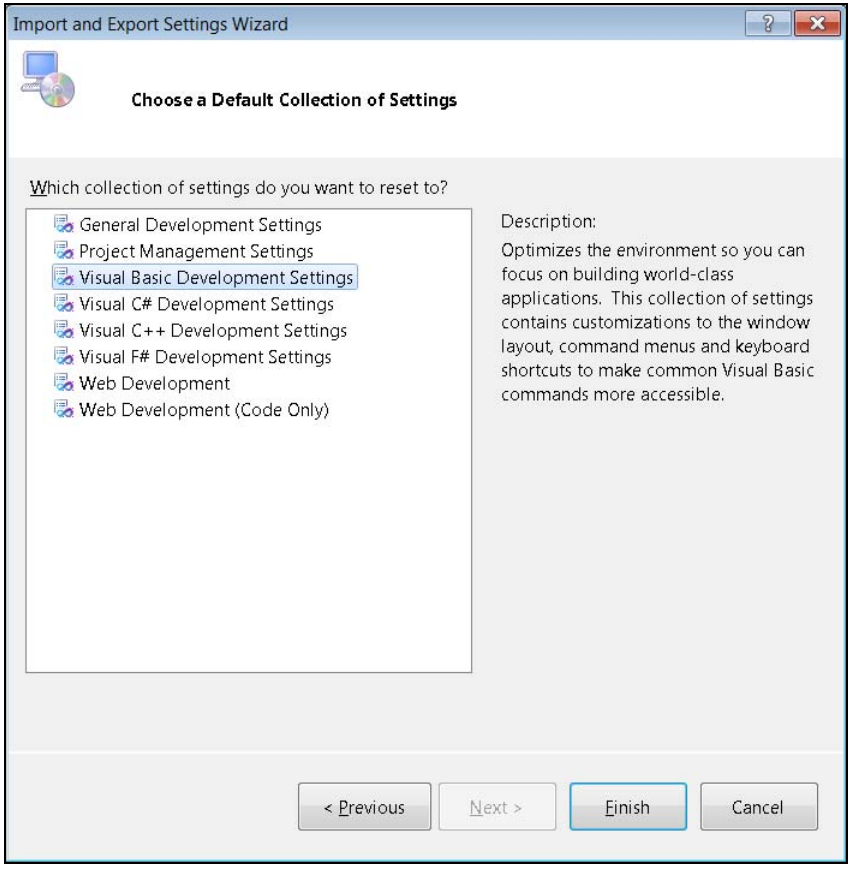


Рис. 1.25. Окно **Choose a Default Collection of Settings**

Из рис. 1.25 видно, что существует несколько стандартных наборов параметров настройки среды VS по умолчанию, из которых вы можете выбирать. Каждый из этих стандартных наборов представляет собой тот же самый набор, который вы выбирали во время инсталляции (см. рис. 1.9) или в ветви **Default Settings** (см. рис. 1.22). Выберите нужный вам набор и нажмите кнопку **Finish**. Начнется операция сброса и восстановления, а когда она завершится, на экране появится окно **Reset Complete**, показанное на рис. 1.26. Чтобы закрыть это окно, нажмите в нем кнопку **Close.**

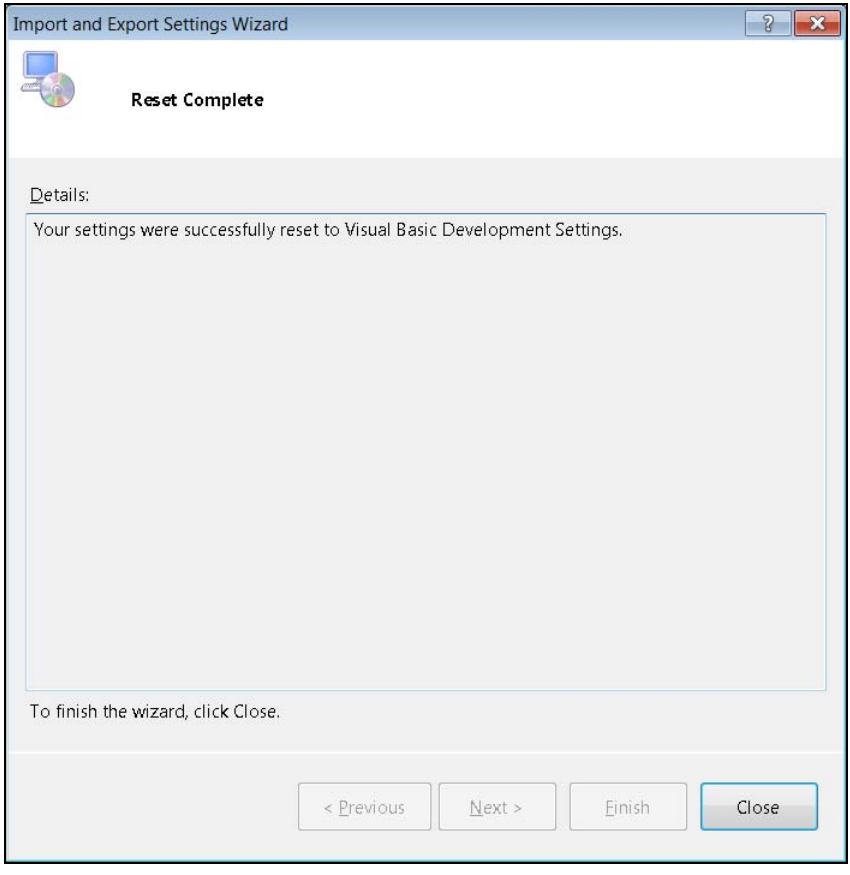


Рис. 1.26. Диалоговое окно **Reset Complete**

Чуть ранее в этой главе мы вкратце, на самом поверхностном уровне, уже коснулись концепции проектов. Далее, по ходу изложения, мы рассмотрим эту тему более подробно. Так, в следующем разделе мы обсудим типы проектов, с которыми можно работать в среде VS.

Знакомство с типами проектов Visual Studio

Visual Studio поддерживает множество типов проектов, что позволяет разработчикам создавать приложения на основе типовых шаблонов. В этом разделе будет показано, как определить, какие типы проектов доступны в среде VS, и даны краткие описания этих типов проектов.

Чтобы просмотреть, какие типы проектов вы можете создавать, выберите из меню команды **File** | **New** | **Project**, как показано на рис. 1.27.

Примечание

Если вы настроили среду для работы с VB, то вы сразу же заметите, что в меню **File** | **New Project** в подменю для выбора будут доступны только две опции, в отличие

от трех опций для C#. Хотя точные формулировки и расположение опций в меню не всегда будут совпадать, вы всегда можете рассчитывать на то, что функциональные возможности будут одинаковы, за исключением особо оговоренных случаев.

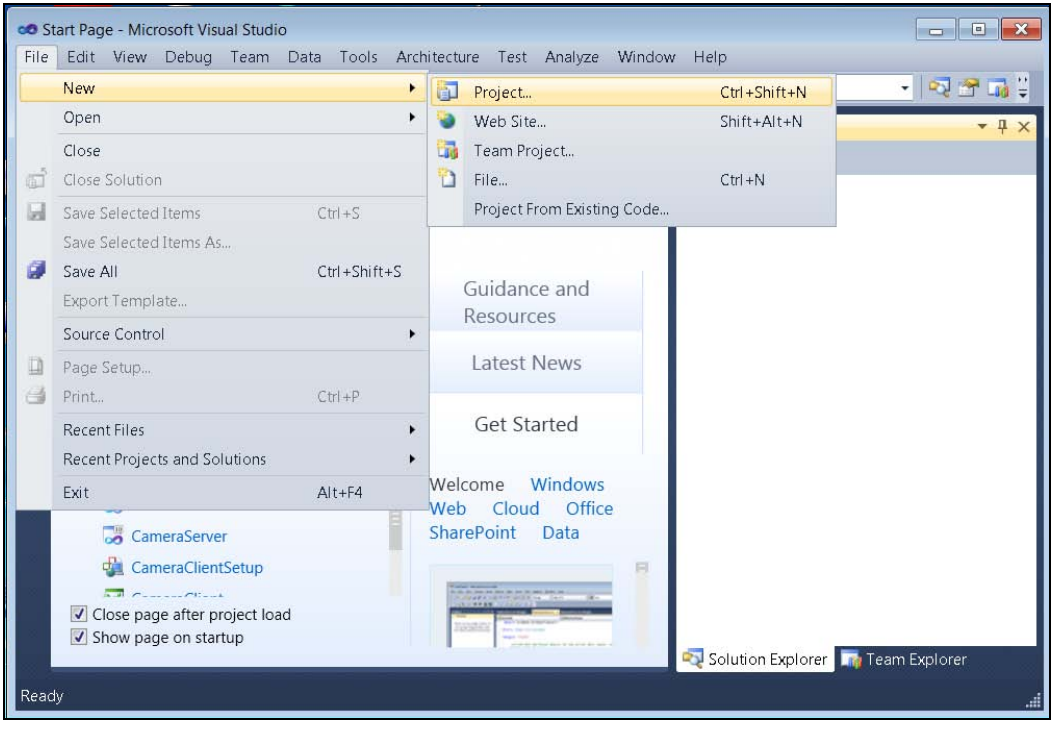


Рис. 1.27. Выбор опции **New Project** из меню **File**

Как видно из рис. 1.27, вы можете создать не только новый проект, но и новый Web-сайт, открыть файл для редактирования или запустить программу-мастер (wizard), которая создаст новый проект из уже имеющихся файлов. Многие из этих опций будут рассматриваться в деталях далее в этой книге, а пока давайте рассмотрим процесс создания нового проекта. Выберите из меню команды **File** | **New** | **Project**, и на экране появится окно **New Project**, показанное на рис. 1.28.

Окно **New Project** демонстрирует, что создать можно множество различных проектов, в том числе: **Windows**, **Web**, **Office**, **SharePoint**, **Cloud**, **Reporting**, **Silverlight**, **Test**, **WCF** и **Workflow**. Некоторые из этих типов проектов не перечислены в окне **New Project**, но если вы прокрутите список шаблонов (**Installed Templates**), то вы увидите, что в списке они присутствуют. На рис. 1.28 показан вид окна для проектов на C#, но для других языков программирования, входящих в состав VS, тоже имеются аналогичные опции. В число этих языков входят VB, C++ и F#.

Если в ходе установки вы выбрали настройки VB, то предлагаемый по умолчанию список типов проектов будет включать проекты VB и C# (они будут перечис-

лены в ветви **Other Languages**). В последующих нескольких разделах будут описаны типы доступных проектов, часть из которых будет подробнее рассмотрена в последующих главах этой книги.

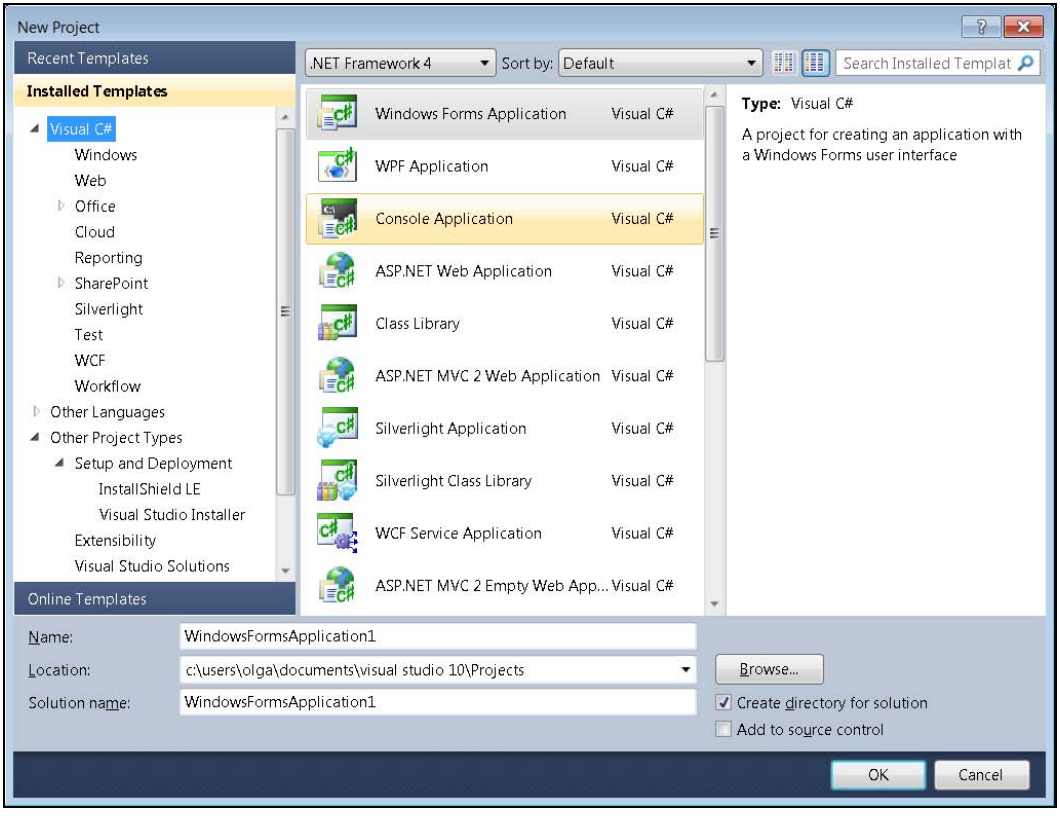


Рис. 1.28. Диалоговое окно **New Project**

Проекты Windows

Если вы выберете опцию **Windows Projects**, то вы увидите длинный список типов проектов приложений Windows, которые вы можете создавать с помощью среды VS. К числу их относятся: настольные приложения (Desktop Applications), включая **Windows Presentation Foundation (WPF)**, **Windows Forms**, и консольные приложения (**Console Applications**). Опция **Console Application** предназначена для построения так называемых консольных приложений, которые запускаются из командной строки и не имеют графического пользовательского интерфейса (Graphical User Interface, GUI). Как правило, эта опция должна выбираться, если вы хотите написать утилиту для администраторов, которую можно использовать при написании административных скриптов, запускаемых из командной строки. Кроме того, данная опция пригодна и для быстрого тестирования вашей программы.

Далее в этой книге мы напишем несколько консольных приложений на языках VB и C#. Кстати, написание консольных приложений — это очень удобный способ изучить тот или иной язык программирования, сконцентрировавшись на его особенностях и не отвлекаясь ни на что другое. Windows Forms представляет собой старую технологию разработки приложений с графическим пользовательским интерфейсом. Более современная технология разработки приложений с графическим пользовательским интерфейсом основана на платформе .NET и называется WPF (Windows Presentation Foundation). Более подробно она будет рассмотрена в одной из последующих глав этой книги.

К числу остальных проектов Windows относится проект Windows Services. Службы или сервисы Windows (Windows Services) — это программы, которые постоянно работают в фоновом режиме и не нуждаются в графическом пользовательском интерфейсе (GUI). Далее, в числе типов проектов можно обнаружить опцию **Class Libraries**. Библиотеки классов (Class Libraries) предназначены для хранения многократно используемого кода, который часто называют промежуточным (middleware). Наконец, библиотеки элементов управления (Control Libraries) представляют собой библиотеки, которые содержат графические элементы управления. Графические элементы управления можно перетаскивать мышью из окна **Toolbox** в окно визуального редактора графического интерфейса в VS.

Web-проекты

К числу Web-проектов относятся такие, как **ASP.NET**, **Server Controls**, **Web Services** и **Dynamic Data**. Проект ASP.NET позволяет создать приложение, хостинг для которого предоставляется Web-сервером, например, таким, как Internet Information Server (IIS), и которое работает в Web-браузере. Проекты типа **Server Controls** позволяют создавать библиотеки элементов управления GUI, которые можно перетаскивать в рабочую область визуального проектирования Web-страницы в VS. Компоненты типа **Web Services** можно использовать многократно, вызывая их через Интернет. Важной функцией компонентов типа Web Services является то, что они используют универсальные (ubiquitous) протоколы, которые могут вызываться кодом, работающим на любой платформе, что упрощает интеграцию между разнородными компьютерными системами. Проекты типа **Dynamic Data** предоставляют возможность быстрого создания работающих Web-сайтов, основываясь на существующей схеме базы данных.

Проекты Office

В течение многих лет разработчики пользовались языком Visual Basic for Applications (VBA) для написания программ, автоматизирующих работу с Microsoft Office. Проекты Office позволяют вам автоматизировать приложения Office с помощью платформы .NET, используя для этого такие языки, как VB и C#. К числу поддерживаемых приложений Microsoft Office принадлежат Excel, Word, Project, PowerPoint, Outlook, Visio и InfoPath.

Проекты SharePoint

SharePoint представляет собой технологию, предназначенную для разработки Web-приложений в стиле порталов. Она тесно ассоциируется с приложениями Office, а также координацией и управлением рабочими группами. Чтобы создавать и запускать проекты SharePoint, необходимо, чтобы компьютер, который вы используете для работы с VS, работал под управлением одной из серверных платформ Microsoft, например, Windows Server 2008. SharePoint не работает на таких платформах, как Windows 7, Windows Vista или Windows XP.

Проекты по работе с базами данных (Database Projects)

Проекты для работы с базами данных (Database projects) включают в свой состав проекты SQL Server. Проекты данного типа предлагают возможности тесной интеграции с SQL Server с целью построения такого кода .NET, который будет работать как часть SQL Server. Например, вы можете писать хранимые процедуры (stored procedures) и функции, используя для этого либо C#, либо VB, и использовать в своем коде преимущества инфраструктуры .NET. VS упрощает развертывание кода на SQL Server, фактически сводя его к единственному щелчку мышью.

Заключение

В этой главе вы познакомились с преимуществами среды VS и получили обзорную информацию о том, как эта интегрированная среда разработчика может помочь вам повысить производительность своей работы, ознакомились с такими ее возможностями, как автоматическая генерация кода, средства быстрой разработки приложений, средства визуального проектирования и расширяемость. После прочтения этой главы вы должны быть способны самостоятельно установить VS, выбрать опции настройки среды таким образом, который наиболее удобен именно для вас и для тех задач, которые вы собираетесь выполнять. Еще один набор навыков направлен на манипулирование настройкой среды разработчика, включая быстрое возвращение всех настроек среды в исходное состояние, предлагаемое по умолчанию, если вы внесли слишком большое количество изменений в настройки среды. Познакомившись со всеми основными функциями IDE, теперь вы можете запустить VS и закрепить полученные знания и навыки на практике. После этого вы сможете приступить к работе над своим первым проектом. Этой теме будет посвящена следующая глава данной книги.

Глава 2

Необходимый минимум знаний о C# и VB.NET: базовый синтаксис

В этой главе вы ознакомитесь с основополагающими концепциями и приобретете следующие ключевые навыки:

- Основные сведения о том, как начать работу над проектом;
- Использование редактора VS;
- Кодирование выражений и утверждений.

Платформа .NET поддерживает несколько различных языков программирования. Так как все эти языки работают на одной и той же платформе и используют одни и те же библиотеки классов, выбор языка, на котором будет вестись разработка, превращается в вопрос личных предпочтений программиста. Иными словами, вы можете выполнять одни и те же задачи, вне зависимости от того, на каком языке вы работаете. Платформа .NET позволяет вам выбирать язык, но при этом сохраняет все предоставляемые преимущества.

Visual Studio (VS) 2010 поставляется с поддержкой четырех языков программирования: C#, C++, F# и Visual Basic.NET (VB). Наиболее популярными языками платформы .NET являются C# и VB, и, соответственно, в VS для них и обеспечивается самый высокий уровень поддержки. По этой причине в данной книге все примеры приводятся как на C#, так и на VB. Хотя вы можете отдать предпочтение любому из этих языков, знание обоих будет серьезным преимуществом. Большая часть материалов, доступных в Интернете, в книгах и журнальных статьях, содержит материалы либо на C#, либо на VB, а иногда (но не всегда) — и на обоих этих языках. Вполне возможно, и даже наиболее вероятно, что вам не захочется отказываться от прочтения замечательной статьи или книги только потому, что вы не владеете языком, на котором написаны приведенные там примеры.

Все материалы, изложенные в *главе 1*, концентрировались на проектах и доступных разработчику возможностях. Эта вводная информация очень важна, но я просто уверен, что вы, как любой настоящий программист, желаете как можно скорее приступить к делу и начать анализировать примеры, а также писать собственный код. В этой главе мы начнем изложение с создания нового проекта, анализа автоматически сгенерированного кода, а затем научимся добавлять в состав проекта новый код. Это — первая глава, где будет обсуждаться синтаксис языка. По ходу изложения будут даваться и рекомендации относительно того, как оптимальным образом использовать функциональные возможности VS, позволяющие продуктивно работать с выбранным языком и облегчающие труд программиста. Начнем

мы с создания простого проекта, а затем обсудим типы данных и выражения каждого из рассматриваемых языков.

Создание простейшего проекта

В *главе 1* уже были вкратце описаны типы проектов, которые можно создать с помощью VS. В этой главе мы продвинемся на шаг дальше и действительно создадим новый проект. Поскольку основное внимание в этой главе уделяется изучению языков C# и VB, мы создадим самый простой проект, а именно — консольное приложение. Консольное приложение представляет собой очень простую программу, которая позволяет вводить, обрабатывать и читать текст из командной строки. В последующих главах мы рассмотрим более сложные приложения, основанные на WPF и ASP.NET.

Чтобы начать работу над проектом, запустите VS и выберите из меню команды **File | New | Project**. Вы увидите окно **New Project**, показанное на рис. 2.1. Ваша первая задача будет заключаться в том, чтобы выбрать тип создаваемого приложения. В данном примере необходимо выбрать опцию **Console Application**. После этого введите имя новой программы (в данном примере — **FirstProgram**) и укажите путь к папке, в которой будет создан новый проект. Остальные опции, доступные в окне **New Project**, позволяют указать номер версии .NET Framework, опции сортировки, размеры значка, представляющего приложение, и возможности поиска.

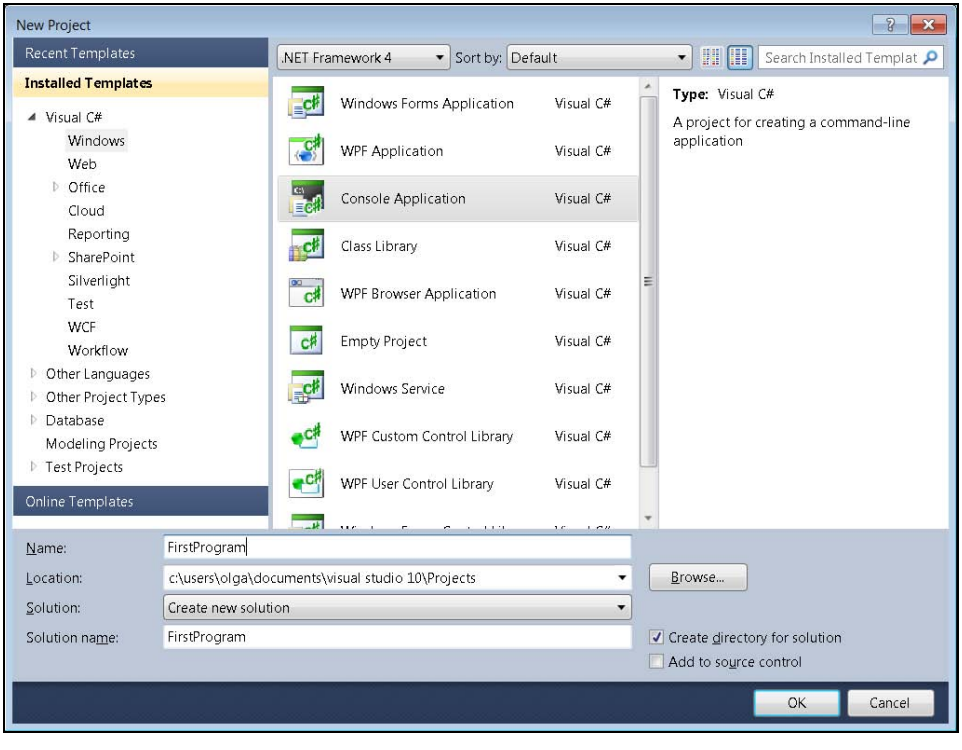


Рис. 2.1. Окно **New Project**

Примечание

Часто бывает полезно задать путь к проекту, отличный от предлагаемого по умолчанию. По умолчанию проекты предлагается сохранять в папке пользовательского профиля My Documents (Мои документы)[1], а этот путь очень длинен, и вводить его целиком — задача довольно затруднительная, причем всегда существует риск допустить ошибку или опечатку. Если вы выберете более короткий путь, то данные проблемы будут сняты. Если вы работаете в составе группы над большим проектом, очень удобно использовать для этой цели специально выделенную сетевую папку, предоставленную в общий доступ.

Примечание

В примерах кода, предлагаемых в данной книге, проекты называются FirstProgramCS (примеры на C#) и FirstProgramVB (примеры, содержащие код на VB). Это соглашение об именовании проектов с указанием языка разработки соблюдается по всей книге.

В верхней части по центру окна, показанного на рис. 2.1, указана версия .NET Framework. Инфраструктура .NET Framework представляет собой набор библиотек классов, модулей времени исполнения (runtime-модулей), а также языков программирования, поддерживаемых в VS. Помимо всего прочего, VS позволяет писать программы, рассчитанные на множество версий .NET Framework, включая версии 2.0, 3.0, 3.5 и 4.0. VS будет компилировать ваш код так, чтобы он работал с выбранной вами версией. Как правило, новые проекты по умолчанию компилируются в расчете на последнюю версию, .NET Framework 4.0, в предположении того, что вам захочется пользоваться новейшими и наиболее мощными возможностями .NET. Основная причина, по которой вам может потребоваться поддержка более ранних версий .NET, заключается в том, что программисту может потребоваться переработка кода, написанного для одной из более ранних версий. Функции сортировки и поиска, расположенные правее этого раздела, позволяют быстро находить нужные типы, применяя для этого различные способы, отдавая предпочтение способу, наиболее удобному лично для вас.

После того как вы нажмете в этом окне кнопку **OK**, на экране появится заготовка проекта консольного приложения на выбранном вами языке программирования. Эту заготовку можно просмотреть в окне **Solution Explorer**, показанном на рис. 2.2. Окно **Solution Explorer** (см. рис. 2.2) содержит готовое решение, представляющее собой контейнер, который, в свою очередь, может содержать множество различных проектов.

Впоследствии вы получите более четкое представление о роли решений (solutions) в организации проектов и возможностей по поддержке разнообразных программных проектов. Сразу же под текущим решением будет находиться ваш первый проект, FirstProgram.

В составе этого проекта находятся его элементы, представляющие собой файлы и настройки. В состав проекта может входить множество различных элементов, а состав индивидуальных элементов проекта зависит от его типа. Например, сущест-

[1] В Windows Vista и Windows 7 этот путь обычно выглядит так: C:\Users\<*UserName*>\My Documents, где <*UserName*> представляет собой имя пользователя, под которым он обычно регистрируется в системе. — *Прим. перев.*

вуют элементы проектов, которые являются частью приложения WPF, но не могут входить в состав консольного приложения. В составе проекта FirstProgram особый интерес представляет собой файл с именем Program.cs (или Module1.vb, если вы программируете на VB). Как будет показано в следующем разделе, этот файл представляет собой файл с программным кодом.

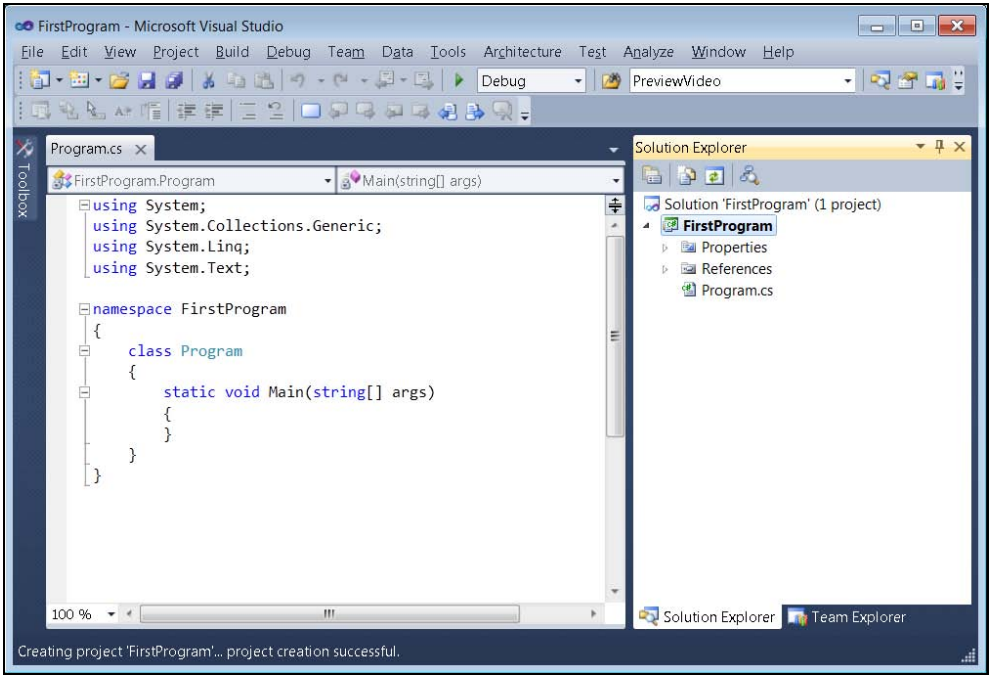


Рис. 2.2. Проект консольного приложения в окне **Solution Explorer**

Исследование "скелета" кода будущей программы

После того как вы запустите программу-мастер New Project Wizard и выберете опцию создания нового консольного приложения, в редакторе VS вы увидите файл с именем Program.cs (или Module.vb), который содержит "заготовку" кода будущей программы. VS создает этот "скелет" будущей программы, используя встроенные шаблоны для большинства поддерживаемых типов проектов. Этот код вы можете дополнять, удалять и модифицировать по собственному усмотрению. Рассмотрим пример такого кода, приведенный в листинге 2.1 (для C#) и в листинге 2.2 (для VB).

Листинг 2.1. Код "заготовки" будущего консольного приложения (код на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace FirstProgram
{
        class Program
        {
                static void Main(string[] args)
                {
                }
        }
}
```

Листинг 2.2. Код "заготовки" будущего консольного приложения (код на VB)

```
Module Module1
       Sub Main()
       End Sub
End Module
```

Код-заготовка, представленный в листингах 2.1 и 2.2, автоматически создается средой VS, когда разработчик начинает работу над новым проектом и выбирает опцию создания нового консольного приложения. Эта "заготовка" и представляет собой начальную точку, с которой вы начинаете работу над новой программой. То, что вы получаете в свое распоряжение, представляет собой готовый код компьютерной программы. Конечно, на данный момент эта программа не делает ничего полезного, но, тем не менее, это полноценная программа, которая может запуститься и способна корректно завершить работу. Если посмотреть на код программы внимательно, то можно заметить, что, например, код на C# содержит наборы вложенных фигурных скобок. Код на VB содержит идентификаторы `Module` и `Sub` с соответствующими идентификаторами `End`, которые, как и фигурные скобки в C#, обозначают границы блока. Фигурные скобки в коде на C# всегда должны иметь пару (т. е. каждой открывающей фигурной скобке должна соответствовать парная закрывающая), и каждая пара определяет блок. Последующие разделы дают детальное пояснение смысла и назначения приведенных образцов кода.

Метод Main

Самый глубоко вложенный блок кода на C# представляет собой определение `static void Main(string[] args)`, которое называется методом (method). В VB метод называется `Sub Main`, и его цель совершенно аналогична. Чуть далее будет показано, что методы (methods) представляют собой единственно возможный способ, позволяющий группировать код в логические последовательности, реализующие конкретные функциональные возможности. Проще всего представлять себе методы как действия, которые вы, как разработчик метода, предписываете компьютеру выполнить. Имя этого конкретного метода — `Main`, и это означает, что он представляет собой точку входа программы, место, где консольное приложение начинает выполнение. Кроме того, о методе `Main` можно думать, как о точке, куда передается управление при запуске вашей программы. Следовательно, код, опреде-

ляющий те действия, которые должна выполнять ваша программа, нужно поместить в метод `Main`.

На языке C#, имя `Main` должно писаться с заглавной буквы. Кроме того, необходимо помнить, что язык C# является чувствительным к регистру символов. Это значит, что идентификаторы `Main` (с заглавной буквы) — это не то же самое, что и `main` (со строчной буквы). Хотя VS автоматически исправляет написание, заменяя строчные буквы на заглавные, там, где это необходимо (если вы забыли это сделать), не забывайте, что язык VB не является чувствительным к регистру. Таким образом, расстановка прописных букв представляет собой общую проблему, особенно для программистов на VB, изучающих C#.

В C# методы могут возвращать значения, например, числа, текст или данные других типов, и тип значений, которые они могут возвращать, указывается непосредственно перед именем метода. В VB, `Sub` (это ключевое слово представляет собой сокращение от термина *subroutine*) не возвращает значений, в отличие от функций (*Function*), что скоро будет продемонстрировано на примерах. Поскольку `Main`, в примере на C#, не возвращает значения, тип возвращаемого значения заменен ключевым словом `void`. Методы могут указывать параметры для аргументов, которые вызывающие методы или функции могут им передавать. В случае с `Main`, параметр представляет собой массив строк с именем переменной `args`. Параметр `args` будет содержать все параметры, переданные программе из командной строки.

Еще одной частью метода C# `Main` является ключевое слово `static`, которое представляет собой модификатор, указывающий, что в течение всего времени исполнения программы должен существовать только один экземпляр метода. Чтобы понять, что представляют собой экземпляры, вспомним, что методы являются членами типов объекта, причем объекты могут быть чем угодно в составе разрабатываемого вами приложения — например, если вы пишете программу для работы с базами данных, они могут быть записями таблицы (скажем, `Customer`, `Account`, или `Vehicle`). Представьте себе, что программа предназначена для использования на предприятии, которое имеет множество заказчиков. Каждый заказчик — это отдельный экземпляр, а это значит, что каждый экземпляр объекта `Customer` содержит методы, которые принадлежат всем экземплярам. Если такой объект, как `Customer`, имеет методы, принадлежащие всем экземплярам, то эти методы не являются статическими (`static`). Однако если объект `Customer` имеет статический метод, то должна быть только одна копия этого метода, которую все остальные объекты `Customer` будут использовать совместно. Например, представьте себе, что вам требуется установить цену со скидкой для всех заказчиков, независимо от того, кем они являются (например, частными лицами или предприятиями). В этом случае можно объявить статический метод и назвать его, например, `GetCustomerDiscount`. Теперь представьте себе другую ситуацию, когда вам при установлении скидок нужно учитывать индивидуальность заказчика. Тогда вам потребуется информация об индивидуальных заказчиках, например, адрес. В этом случае логично будет создать метод с именем `GetAddress`, и при объявлении данного метода не определять его как статический.

VB использует для этой цели ключевое слово `shared`, которое имеет такой же смысл, как ключевое слово `static` в C#. Модули по своим внутренним свойствам

являются разделяемыми (inherently shared), и все методы модуля должны быть разделяемыми. Таким образом, метод VB `Main` является разделяемым.

В C# фигурные скобки определяют начало и конец метода `Main`. В VB метод `Main` начинается с ключевого слова `Sub`, и диапазон его действия простирается до `End Sub`. Далее, обратите внимание на то, что в C# метод `Main` заключен в фигурные скобки, которые принадлежат некоей сущности, называемой *классом* (class), который имеет имя `Program`. В VB метод `Main` заключен в сущность, именуемую модулем (*module*). О классах и модулях пойдет речь в следующем разделе.

Класс Program

Методы всегда находятся внутри декларации типа. Тип может быть классом (`class`) или структурой (`struct`) для C#, или же классом (`class`), модулем (`module`) или структурой (`struct`) — для VB. Термин тип (`type`) может показаться вам чем-то чужеродным, поэтому проще всего думать о нем как о чем-то, что способно содержать внутри себя другие объекты. Методы — это один из типов объектов, которые могут содержаться внутри типов (types). Фрагмент кода, который приведен в листингах 2.3 (для C#) и 2.4 (для VB), представляет собой тип, который содержит метод `Main`. В программах на C# он представляет собой класс, а в приведенном примере на VB он является методом:

Листинг 2.3. Фрагмент кода на C#, представляющий собой класс, который содержит метод Main

```
class Program
{
     // Метод Main для краткости пропущен
}
```

Листинг 2.4. Фрагмент кода на VB, представляющий собой метод Main

```
Module Module1
' Метод Main для краткости пропущен
End Module
```

Большинство типов объектов, которые вы создаете, будут представлять собой классы, как было показано в только что приведенном примере на C#.

Программируя на VB, вы можете заменить модуль (`Module`) классом (`Class`). Хотя VS по умолчанию использует модули (`Module`) в качестве типа объекта по умолчанию для всех новых проектов, такое поведение является наследием предшествующих версий VB. На практике вы совершенно не обязаны использовать модули, а наоборот, вам следует предпочесть именно классы. Класс `Program` содержит метод `Main`. В состав класса `Program` можно добавить и другие методы или модуль `Module1`, что вы неоднократно увидите на примерах, которые будут приводиться в последующих главах этой книги. Консольное приложение определяет заготовку кода класса, который получает имя `Program`. В действительности вы можете назвать

класс как угодно. Впрочем, какие бы имена вы ни выбрали, желательно, чтобы они несли смысловую нагрузку и указывали на предназначение класса.

Например, если создаваемый класс предназначен для работы с заказчиками, вполне резонно и назвать его соответственно — `Customer`. Кроме того, класс должен содержать только методы, предназначенные для работы с заказчиками. Не следует добавлять в этот класс методы, предназначенные для непосредственной работы со счетами, платежными поручениями, типами продукции или какой-либо другой информацией, кроме данных о заказчиках. Почему? Да просто потому, что это загромоздит код класса `Customer`, сделает его запутанным и сложным для понимания. Организация классов осуществляется с помощью пространств имен (namespaces), которые мы обсудим в следующем разделе.

Пространство имен FirstProgram

Пространство имен (namespace) помогает добиться того, чтобы иена создаваемых вами классов были уникальными и внутренне непротиворечивыми. Фактически, это аналогично добавлению фамилии и отчества к вашему имени, что делает, например, ваше имя если и не полностью уникальным, то хотя бы чуть более индивидуальным. Имя пространства имен, впрочем, предшествует имени класса, в то время как ваши отчество и фамилия следуют за вашим именем.

Пространство имен упрощает организацию кода и помогает быстро находить нужные фрагменты в чужом коде. Эта организация позволяет строить библиотеки кода, которые повышают для программистов вероятность быстро найти именно то, что им нужно. Платформа .NET имеет гигантскую библиотеку классов (class library), которая организована по пространствам имен (namespaces) и сборкам (assemblies). Не огорчайтесь, если сейчас все сказанное кажется вам чересчур абстрактным, все эти концепции станут вам лучше понятными по мере того, как вы все больше и больше будете программировать. Основным в .NET является пространство имен `System`, которое, в свою очередь, имеет множество подчиненных пространств имен, которые называются подпространствами имен (sub-namespaces). Например, предположим, что вам требуется найти классы .NET для работы с данными. Где их искать? В подпространстве имен `System.Data`! Еще один пример: если вам нужны классы .NET для работы с сетевыми протоколами, включая TCP/IP, FTP или HTTP, то искать их следует в подпространстве имен `System.Net`.

Еще одно преимущество пространств имен заключается в том, что они позволяют различать между собой классы, имеющие одинаковые имена, но находящиеся в разных библиотеках. Например, представьте себе, что вы приобрели некую библиотеку от сторонних разработчиков, и в этой библиотеке тоже есть класс `Customer`. Как отличить класс `Customer` из этой библиотеки от вашего собственного класса с таким же именем? Пространства имен предоставляют удобную возможность сделать это без особых усилий. Каждый из классов `Customer` имеет собственное пространство имен, поэтому вам достаточно написать код, указывающий пространство имен для каждого из классов `Customer`, где каждый из них будет определяться своим пространством имен. По этой причине наилучшей рекомендацией будет совет использовать пространства имен всегда и везде.

Класс `Program` в листинге 2.1 принадлежит к пространству имен `FirstProgram`. В листинге 2.5 приведен фрагмент этого кода на C#:

Листинг 2.5. Фрагмент кода заготовки программы на C#, определяющий пространство имен

```
namespace FirstProgram
{
     // Код класса пропущен для краткости
}
```

В пространство имен можно поместить множество классов. В данном случае под нахождением класса внутри пространства имен понимается то, что этот класс заключен между открывающей и закрывающей скобками пространства имен.

Директивы `using` в начале кода на C# в листинге 2.1 на самом деле представляют собой сокращения, упрощающие написание кода. Например, пространство имен `System` содержит класс `Console`. Если бы в коде не употреблялась директива `using System`, вам потребовалось бы писать `System.Console.WriteLine` вместо `Console.WriteLine`. Это всего лишь очень простой и краткий пример, но в реальности использование директив `using` действительно помогает писать более простой и удобочитаемый код.

Модуль VB должен быть объявлен (декларирован) на глобальном уровне. Это означает, что он не может быть добавлен в создаваемое вами пространство имен. В листинге 2.6 показано, как выглядит пространство имен в VB:

Листинг 2.6. Определение пространства имен в коде на VB

```
Namespace FirstProgram
        Public Class Customer


        End Class
End Namespace
```

В данном примере демонстрируется, что пространство имен `FirstProgram` содержит класс `Customer`. Следующая задача, за которую вы, как программист, хотели бы взяться, естественно, состоит в написании кода. Но, прежде чем приступать к этой задаче, давайте ознакомимся с некоторыми функциональными возможностями редактора кода VS (Code editor).

Обзорная информация о редакторе кода VS

Редактор кода VS — это инструмент, который вы будете использовать чаще всего при выполнении своей основной работы программиста. В этом разделе будут описаны его наиболее важные функции, а также будет рассказано о том, как выполнить его индивидуальную настройку. На рис. 2.3 показано окно редактора кода с загруженным в него исходным текстом заготовки будущего консольного приложения (см. код на C# из листинга 2.1).

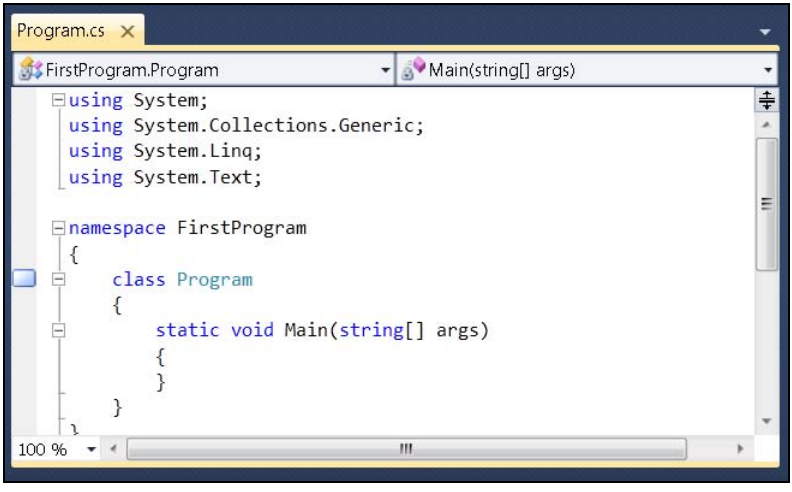


Рис. 2.3. Редактор кода VS

В последующих нескольких разделах обсуждаются различные элементы редактора кода, начиная с инструментов для обнаружения классов (class) и их членов (member).

Средства обнаружения классов и их членов

Два раскрывающихся списка в верхней части окна редактора представляют собой средства обнаружения классов (class locator) и их членов (member locator). Они предназначены для быстрого перемещения по коду. Если ваш файл содержит множество классов, то раскрывающимся списком **Class locator** можно пользоваться, чтобы быстро выбрать нужный класс. Когда вы выберете класс, редактор быстро переместит вас к первой строке этого класса, содержащей его декларацию. В своей повседневной работе я редко помещаю в файл более одного класса, поэтому, как правило, каждый из моих файлов содержит только один класс. Следовательно, я очень редко пользуюсь определителем классов и не уделяю ему излишнего внимания. Но, если вы будете пользоваться программами-мастерами VS (VS wizards), которые автоматически генерируют код, и поместите большое количество классов в один файл, тогда определитель классов будет очень полезен. Он действительно представляет собой средство быстрого обнаружения нужных классов, позволяющее легко их находить и разбираться с тем, что же делает автоматически сгенерированный код. Определитель членов (member locator) представляет собой раскрывающийся список в правом верхнем углу. Он содержит список методов и других членов класса, предварительно выбранного в определителе классов (class locator).

До сих пор мы обсуждали только один тип членов класса — метод, но на самом деле их гораздо больше, и с некоторыми из них вы познакомитесь в следующих главах этой книги. Если вы выберете из списка **Member locator** имя одного из членов класса, то редактор кода сразу же переместит вас к первой строке этого члена класса. Не забывайте о замечательных возможностях определителя членов (member

locator) и пользуйтесь ими всякий раз, когда вам приходится пролистывать большой файл. Вы быстро убедитесь в том, что определители классов и членов действительно существенно ускоряют поиск нужных фрагментов кода.

Вертикальная линия на левой границе экрана редактора называется границей индикатора (indicator margin). Здесь располагаются значки для различных функций, таких, как закладки (bookmarks) и отладочные точки останова (debug breakpoints). Закладки обсуждаются в следующем разделе.

Закладки

Закладка, установленная на первую строку класса `Program`, была показана на рис. 2.3. Закладки упрощают быструю навигацию по коду программы, исключая необходимость в ручных переходах при одновременной работе с несколькими документами или с несколькими фрагментами кода в одном и том же документе. В табл. 2.1 приведен список клавиатурных комбинаций для работы с закладками.

Таблица 2.1. *Клавиатурные комбинации для работы с закладками*

Клавиатурная комбинация	Описание
<CTRL>+<B>, <T>	Включение и выключение закладки
<CTRL>+<B>, <N>	Переход к следующей закладке
<CTRL>+<B>, <P>	Переход к предыдущей закладке
<CTRL>+<B>, <C>	Очистить список всех закладок
<CTRL>+<W>, <B>	Открыть окно **Bookmarks**

Одна из приведенных в табл. 2.1 комбинаций, <CTRL>+<W>, <B>, открывает окно **Bookmarks**, показанное на рис. 2.4, которое позволяет манипулировать всеми закладками, расставленными по всему коду вашего приложения.

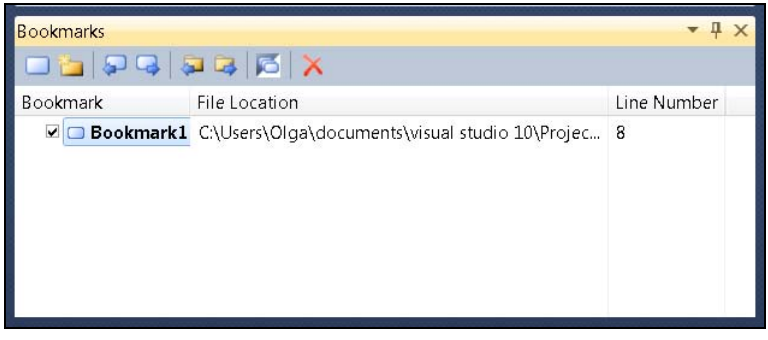


Рис. 2.4. Диалоговое окно **Bookmarks**

Закладка имеет панель инструментов, точно такую же, как и та, которую вы видите в активном окне редактора VS. Кнопки на этой инструментальной панели соответствуют командам и их клавиатурным комбинациям, перечисленным в табл. 2.1, плюс возможность перемещения между папками.

В списке закладок можно устанавливать или сбрасывать флажки, таким образом, включая или отключая соответствующие им закладки. Когда закладка неактивна, команды **Previous** и **Next** будут ее игнорировать. Если выполнить на закладке двойной щелчок мышью, то можно изменить ее имя. Поля **File Location** и **Line Number** указывают точное местоположение закладки.

Установка параметров настройки редактора

Редактор предлагает широкие возможности по конфигурированию, причем этих опций гораздо больше, чем может себе представить среднестатистический пользователь. Чтобы просмотреть список доступных опций, выберите из меню команды **Tools | Options**. На экране появится окно **Options**, показанное на рис. 2.5. Как видно из этой иллюстрации, если выбрать команды **Environment | Fonts and Colors**, то вы сможете изменить вид интерфейса редактора VS. В отношении нашего текущего обсуждения редактора, это именно те опции, с помощью которых вы можете изменить цветовые выделения ключевых слов и конструкций кода, выбрав именно ту систему цветовых обозначений, которая вам удобна и привычна.

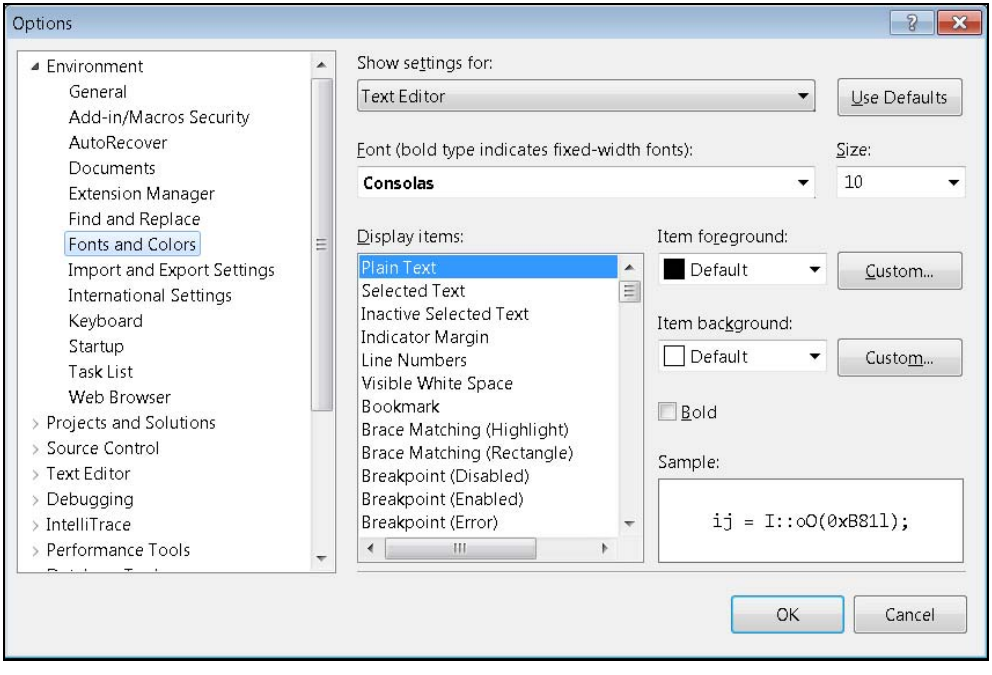


Рис. 2.5. Диалоговое окно **Options**

Совет

Если вы захотите предоставить в общий доступ свою систему цветовых обозначений синтаксических конструкций языка, вы можете воспользоваться программой-мастером **Import and Export Settings Wizard**, о которой уже говорилось в *главе 1*. Кроме того, в окне **Options** имеется ветвь **Import And Export Settings**, расположенная сразу же под опцией **Fonts and Colors**.

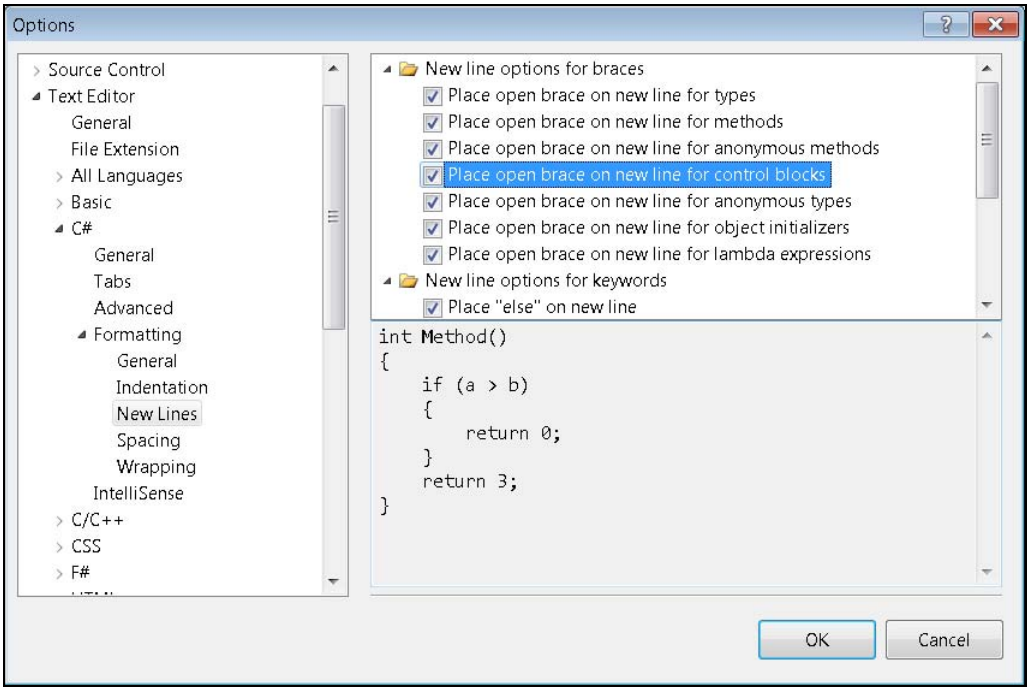


Рис. 2.6. Варианты настройки редактора кода для языка C#

Большинство опций настройки редактора в окне **Options** находятся в ветви опций, зависящих от конкретного языка программирования. На рис. 2.6 показаны варианты настройки для языка C#.

Диалоговое окно **Options**, показанное на рис. 2.6, показывает настройку опций **Text Editor | C# | Formatting New Lines**. Как видите, здесь имеется обширный набор опций, в том числе и для деталей автоматического форматирования новых строк и расположения фигурных скобок. Если код вашей программы отформатирован не так, как вы привыкли его воспринимать, загляните на эту страницу и настройте форматирование так, как вам привычнее.

Экономия времени при помощи фрагментов (Snippets)

Фрагменты (Snippets) заслуживают внимательного изучения хотя бы потому, что они помогают здорово экономить время. Фрагмент (snippet) представляет собой набор базовых конструкций и ключевых слов, из которых формируется шаблон будущего фрагмента кода. Код, из которого состоят эти фрагменты, обычно представляет собой типовые конструкции, часто встречающиеся в повседневной практике программирования. Там вы увидите большое количество общих утверждений и блоков кода. В данной главе будет приведено много таких фрагментов. В этом разделе мы обсудим технику использования автоматически созданных фрагментов и проиллюстрируем ее практическими примерами.

Чтобы воспользоваться фрагментами, просто начните вводить префикс типовой конструкции и дождитесь появления "списка продолжений", отображаемого функцией Intellisense. Затем дважды нажмите клавишу <TAB> и заполните все поля появившейся формы. Переход между полями тоже осуществляется с помощью клавиши <TAB>. Завершив редактирование, нажмите клавишу <ENTER>.

Поскольку в этой главе уже зашла речь о пространствах имен, давайте проиллюстрируем сказанное на примере автоматической генерации шаблона для пространства имен. Для начала откройте в окне редактора любой код и начните вводить с клавиатуры часть файла, внешнего по отношению ко всем блокам кода, например, непосредственно под утверждениями `using`, но над любыми из существующих утверждений, объявляющих пространства имен. Введите с клавиатуры букву **n** и дождитесь появления списка продолжений, который прокрутится до элемента `namespace`. Введите с клавиатуры символ **a**, и вы увидите, что элемент `namespace` останется в списке один, как показано на рис. 2.7.

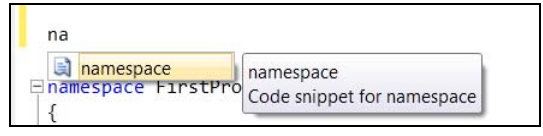


Рис. 2.7. Использование автоматически генерируемых фрагментов

Примечание

Если в окне, показанном на рис. 2.7, нажать клавиатурную комбинацию <CTRL>+ +<ALT>+<SPACE>, то вы сможете переключаться между режимами Intellisense, которые называются **Consume First** и **Standard**. В режиме **Standard**, который включается при нажатии клавиатурной комбинации <CTRL>+<ALT>+<SPACE>, ключевые слова выбираются автоматически по мере того, как вы вводите символы с клавиатуры. Однако бывают и такие ситуации, когда вы пытаетесь ввести слово, которого пока еще нет в базе данных, и функция Intellisense ведет себя слишком агрессивно, добавляя одно из уже существующих слов вместо того, что вы вводите, и не дожидаясь, пока вы закончите ввод. В таких случаях нажмите клавиатурную комбинацию <CTRL>+<ALT>+<SPACE>, переключитесь в режим **Consume First**, и выбрано будет именно то, что вы вводите. Помимо этого, в режиме **Consume First** можно пользоваться клавишей <↓>, чтобы выбирать подсвеченные элементы списка возможных завершений.

Идентифицировать фрагменты в списке завершений (completion list) можно по значкам с изображением разорванного листа бумаги. В этот момент вы можете нажать клавишу <TAB>, чтобы завершить ввод ключевого слова `namespace`. Затем следует нажать клавишу <TAB> еще раз, чтобы создать шаблон на основе информации, введенной вами в выделенные подсветкой поля. На рис. 2.8 показаны результаты создания шаблона пространства имен за счет нажатия клавиши **n** и двукратного нажатия клавиши <TAB>.

Как показано на рис. 2.8, в поле `Namespace` выведенной формы можно вести нужное вам имя вместо предлагаемого по умолчанию (`MyNamespace`), которое в данном случае представляет

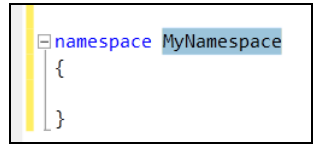


Рис. 2.8. Заполнение шаблона автоматически генерируемого фрагмента кода

собой просто "болванку" (placeholder). Если шаблон, с которым вы работаете, содержит более одного поля, для перемещения между полями используется клавиша <TAB>. В случае с шаблоном пространства имен, показанным на рис. 2.8, форма содержит только одно поле, и для завершения достаточно заполнить это поле.

VB предлагает несколько вариантов работы с автоматически генерируемыми фрагментами (snippets): за счет ввода префиксов и за счет использования списка выбора (pick list). Чтобы на практике ознакомиться с тем, как работают автоматически генерируемые фрагменты (snippets) в VB, переместите курсор за пределы модуля `Module1`, под строку `End Main` (иными словами, за пределы блока `Main`). Введите с клавиатуры символы `Su` и нажмите клавишу <TAB>. Обратите внимание, что VS создаст `Sub` (метод) наряду с шаблоном, который содержит поля для заполнения автоматически генерируемого фрагмента `Sub`.

Еще один способ добавления автоматически генерируемых фрагментов VB заключается в том, чтобы нажать клавишу <?>, а затем — клавишу <TAB>. На экране появится список выбора, показанный на рис. 2.9. Этот список можно прокручивать до тех пор, пока в одной из папок не будет найден нужный вам фрагмент. В комплекте с VB поставляется гораздо больше автоматически генерируемых фрагментов кода, чем в комплекте с C#.

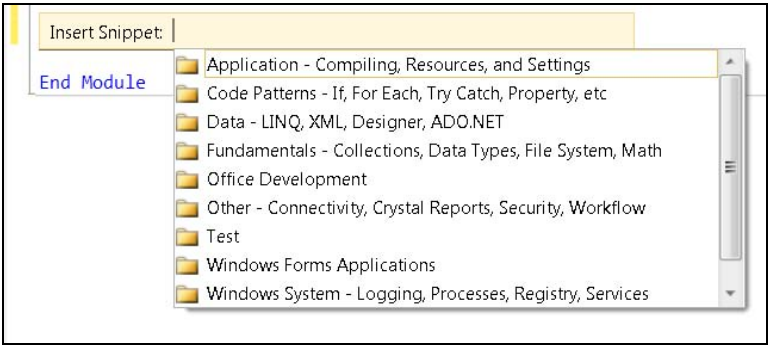


Рис. 2.9. Список выбора автоматически генерируемых фрагментов кода на VB

Теперь, когда вы научились работать с автоматически генерируемыми фрагментами кода, перейдем к обсуждению различных типов утверждений, используемых в C# и VB, а также деталей, иллюстрирующих работу этих утверждений.

Кодирование выражений и утверждений

Как на C#, так и на VB существуют различные типы утверждений, в том числе: присваивания (assignment), инициирования методов (method invocations), ветвления (branching) и циклы (loops). Начнем с рассмотрения элементарных типов данных, таких как целые (integers) и строки (strings), и после этого перейдем к построению выражений и установке значений путем выполнения операций присваивания. Далее мы обсудим ветвления, такие как `if` и `switch` в C# или `case` в VB. Наконец, мы обсудим разнообразные циклы, в частности, `for` и `while`. Все эти языковые особенно-

сти будут обсуждаться в общих чертах. Между конкретными реализациями в языках C# и VB, естественно, имеются различия, но здесь мы рассматриваем базовые концепции, которые принципиально не отличаются.

Прежде чем написать какой-либо код, следует разобраться в том, как работает технология Intellisense. Intellisense — это важный инструмент повышения индивидуальной производительности труда, освоив который вы сможете минимизировать общее количество нажатий на клавиши в общих сценариях работы.

Использование технологии Intellisense

Ранее мы уже ознакомились с тем, как работают автоматически генерируемые фрагменты кода. Они используют технологию Intellisense для отображения списка завершения. Технология Intellisense встроена в редактор кода VS, что позволяет автоматически завершать утверждения, используя минимум нажатий на клавиатуру. В следующем примере будет проиллюстрировано использование Intellisense на примере добавления строки кода к методу Main. Пока ничего не вводите, просто посмотрите, как работает Intellisense.

Представьте себе, что вы хотите ввести простую строку кода, которая выглядит так:

```
Console.WriteLine("Hello from Visual Studio 2010!");
```

Если вы будете следовать приведенным далее инструкциям, то вы увидите, как VS поможет ускорить работу по вводу текста программы, минимизировав количество нажатий на клавиши при клавиатурном вводе:

1. Внутри скобок метода `Main`, введите символ **c**. Обратите внимание — появится окно Intellisense со списком всех доступных идентификаторов, начинающихся на **c**. Этот список называется списком завершения.
2. Теперь введите символ **o**, и вы увидите, что элементы списка завершения будут отфильтрованы — теперь все идентификаторы в этом списке будут начинаться на **co.**
3. Теперь введите символ **n**, и вы увидите, что в списке завершения останется только один элемент — **Console**. Это — именно то, что нам и требуется. Обратите внимание, что желаемый результат был достигнут нажатием всего лишь трех клавиш.
4. Многие люди в этот момент нажимают клавиши <ENTER> или <TAB>, чтобы редактор VS завершил ввод слова **Console**, но на самом деле это будет уже лишней тратой времени на лишнее нажатие клавиши.

   Поскольку вы знаете, что между **Console** и **WriteLine** должен стоять оператор "точка" (.), продолжайте ввод текста программы, введя с клавиатуры символ "точка" (.), в результате чего VS отобразит в окне редактора строку **Console.** Далее появится новый список завершения, который будет перечислять члены класса `Console`, из которого вы теперь можете выбирать.

Примечание

Да, нужно признать, что пару разделов нам пришлось потратить на объяснение принципов, в соответствии с которыми вы можете сэкономить пару нажатий на клавиши.

Но это — не единственное, на что следует обратить внимание в приведенном объяснении. Основная ценность этих сведений заключается в том, что существует множество вариантов повышения производительности индивидуального труда. Каждый раз, когда вы пользуетесь той или иной опцией из числа новых возможностей VS, вы хоть немного, но повышаете индивидуальную производительность вашего труда.

5. Теперь введите с клавиатуры строку **write**, и вы увидите, что в списке выбора появятся элементы **Write** и **WriteLine**. Теперь введите букву **l**, и вы увидите, что элемент **WriteLine** останется единственной опцией в списке завершения.

Примечание

Если вы уже несколько раз вводили строку **WriteLine**, то вы заметите, что список завершения после того, как вы введете несколько символов, будет перемещаться прямо к элементу **WriteLine**, а не к элементу **Write**. Это происходит потому, что Intellisense запоминает наиболее часто используемые идентификаторы, и выбирает их из списка первыми. Если вы продолжите ввод, Intellisense выделит из списка идентификаторы, в которых имеется точное соответствие. Обратите внимание на установленную опцию на рис. 2.10. Intellisense предварительно выбирает наиболее часто используемые члены класса, показывая, что такое поведение включено по умолчанию.

6. Давайте сэкономим еще одно нажатие на клавишу. Для этого нажмите клавишу <(>, чтобы дать возможность VS закончить ввод имени метода `WriteLine`.
7. В этот момент вы можете закончить ввод утверждения, в результате чего метод `Main` будет выглядеть так, как показано в листингах 2.7 (для C#) и 2.8 (для VB):

Листинг 2.7. Автоматически сгенерированный фрагмент кода метода Main на C#

```
static void Main(string[] args)
{
        Console.WriteLine("Hello from Visual Studio 2010!");
}
```

Листинг 2.8. Автоматически сгенерированный фрагмент кода метода Main на VB

```
Sub Main()
        Console.WriteLine("Hello from Visual Studio 2010!")
End Sub
```

Если вы — разработчик программ на C# и хотите изменить опции настройки Intellisense, выберите из меню команды **Tools** | **Options**, затем выберите команды **Text Editor** | **C#** | **Intellisense**, и вы увидите опции настройки Intellisense, которые были показаны на рис. 2.10. Для VB такой возможности не предоставляется.

Обратите внимание, что в окне есть текстовое поле с меткой **Committed by typing the following characters**, которое содержит набор символов, ввод которых заставляет VS вводить оставшуюся часть идентификатора, выделенного в списке выбора, плюс введенный вами символ. Если вернуться к шагу 4, то здесь вы найдете и символ точки (.), который подтверждает выбор подсвеченного на данный момент идентификатора.

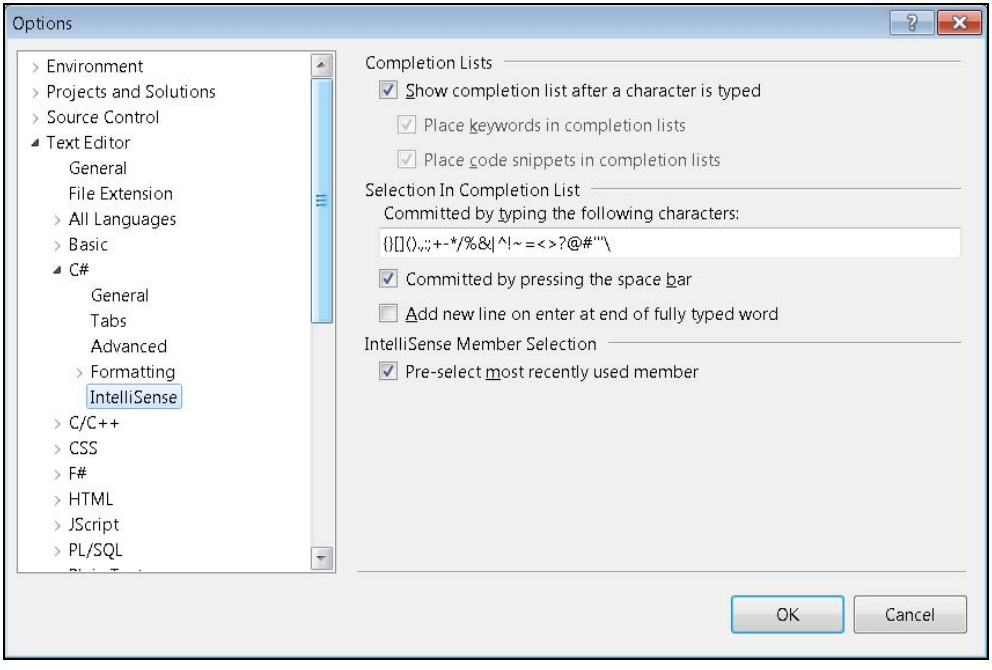


Рис. 2.10. Опции Intellisense

Итак, вы получили программу, которая уже выполняет какие-то действия, а именно вывод сообщения на консоль. В следующем разделе будет показано, как запустить эту программу на исполнение.

Запуск программ

В VS вы можете запустить программу на исполнение как без отладки, так и в отладочном режиме. Отладка (Debugging) представляет собой процесс поиска ошибок в коде программы. Если программу запустить на исполнение в режиме отладки, вы сможете задавать точки останова (breakpoints) и пошагово исполнять код программы. Подробнее об отладке будет рассказано в *главе 6*. Запуск программы без отладочных опций позволяет просто выполнить программу, пропуская точки останова, которые могут быть для нее установлены.

Чтобы запустить программу в обычном режиме (без отладки), выберите из меню команды **Debug | Start Without Debugging** или нажмите клавиатурную комбинацию <CTRL>+<F5>. В результате запустится окно работы с сеансом командной строки, в котором будет выведен текст следующего содержания: "Hello from Visual Studio 2010!" (или текст, который вы сами зададите). Окно сеанса работы с командной строкой останется открытым до тех пор, пока вы не нажмете клавишу <ENTER> или до тех пор, пока вы не закроете это окно.

Чтобы запустить программу с отладочными опциями, либо выберите из меню команды **Debug | Start Debugging**, либо нажмите клавишу <F5>. В данном случае, из-за способа кодирования, окно сеанса работы с командной строкой быстро закро-

ется после выполнения кода программы. Если вы будете недостаточно внимательны или просто мигнете в момент исполнения, вы можете даже не заметить, как это произойдет. Чтобы избежать такой ситуации, вы можете добавить в текст программы утверждение `Console.ReadKey`, расположенное сразу же под утверждением `Console.WriteLine`, благодаря чему окно останется открытым до тех пор, пока вы не нажмете еще какую-нибудь клавишу. Модифицированный метод `Main` приведен в листингах 2.9 (для C#) и 2.10 (для VB).

Листинг 2.9. Модифицированный метод Main (для C#)

```
static void Main(string[] args)
{
       Console.WriteLine("Hello from Visual Studio 2010!");
       Console.ReadKey();
}
```

Листинг 2.10. Модифицированный метод Main (для VB)

```
Sub Main()
        Console.WriteLine("Hello from Visual Studio 2010!")
        Console.ReadKey()
End Sub
```

Нажатие клавиши <F5> отобразит строку "Hello from Visual Studio 2010!" в окне сеанса работы с командной строкой, точно так же, как это было при запуске программы в обычном режиме (без отладки).

Чтобы разобраться с различием между этими двумя опциями, подумайте о разнице между простым запуском программы и ее отладкой. Если вы просто запускаете программу на исполнение, вам требуется, чтобы она оставалась открытой до тех пор, пока вы сами ее не закроете. Однако если вы отлаживаете программу, то в большинстве случаев вы устанавливаете точки останова (breakpoint) и выполняете код пошагово. Когда сеанс отладки завершается, вам требуется, чтобы программа прекратила исполнение с тем, чтобы вы могли вернуться к работе над ее кодом.

Теперь, когда вы научились добавлять код в состав метода `Main` и выполнять программу, можно приступать к изучению строительных блоков различных алгоритмов, о чем будет рассказано в следующем разделе.

Простейшие типы и выражения

Базовыми элементами любого кода, который вы разрабатываете, являются простейшие типы и выражения, к рассмотрению которых мы приступаем.

Простейшие типы

Для разрабатываемой программы обычно определяются переменные, тип которых является одним из базовых или простейших типов.

Переменные могут содержать значения, которые вы можете читать, выполнять над ними различные манипуляции и записывать. Существуют различные типы пе-

ременных, указывающие, что за данные могут в них содержаться. Платформа .NET имеет базовые, примитивные типы (известные как встроенные) и типы, индивидуально определяемые разработчиком (custom types). Индивидуально определяемые типы представляют собой такие типы, которые разработчик создает сам и задает для них свойства, необходимые для конкретной программы. Например, если вы пишете программу для управления данными о клиентах предприятия, то вам наверняка понадобится переменная, которая будет содержать тип заказчика. О создании индивидуально определяемых типов мы поговорим чуть позже, а пока обсудим простейшие типы, которые являются своего рода "строительными блоками" для более сложных конструкций. Простейшие (примитивные) типы являются составной частью языков программирования, и они встроены в платформу .NET. Примитивный тип представляет собой элементарный тип данных, с которыми может работать платформа .NET. Такой тип не может быть разложен на еще более элементарные типы. В противоположность этому, индивидуально определяемые типы как раз и создаются с использованием одного или нескольких элементарных — например, тип `Customer` может состоять из имени, адреса и, возможно, еще некоторых полей, содержащих данные одного из элементарных типов. Элементарные типы данных с краткими описаниями перечислены в табл. 2.2.

Таблица 2.2. *Элементарные типы данных (примитивы)*

VB	C#	.NET	Описание
`Byte`	`byte`	`Byte`	8-битное беззнаковое целое число
`SByte`	`sbyte`	`SByte`	8-битное знаковое целое число
`Short`	`short`	`Int16`	16-битное знаковое целое число
`UInt16`	`ushort`	`UInt16`	16-битное беззнаковое целое число
`Integer`	`int`	`Int32`	32-битное знаковое целое число
`UInt32`	`uint`	`UInt32`	32-беззнаковое целое число
`Long`	`long`	`Int64`	64-битное знаковое целое число
`UInt64`	`ulong`	`UInt64`	64-битное беззнаковое целое число
`Single`	`float`	`Single`	32-битное число с плавающей точкой
`Double`	`double`	`Double`	64-битное число с плавающей точкой
`Boolean`	`bool`	`Boolean`	`true` или `false`
`Char`	`char`	`Char`	16-битный символ Unicode
`Decimal`	`decimal`	`Decimal`	96-битное десятичное число (используется для указания денежных сумм)
`String`	`string`	`String`	Строка символов Unicode

При просмотре табл. 2.2 имейте в виду, что язык C# является чувствительным к регистру символов, и все элементарные типы данных пишутся строчными символами. Кроме того, в таблице есть и третий столбец — для встроенных типов дан-

ных .NET. Иногда вам может встретиться код, использующий типы данных .NET, который является псевдонимом для специфических типов данных, имеющихся в C# и VB. Пример, приведенный в листингах 2.11 и 2.12, показывает, каким образом можно объявить знаковую 32-разрядную целую переменную для C# и VB, вместе с встроенным типом данных .NET. В соответствии с табл. 2.2, в качестве встроенных типов данных C# использует `int`, а VB — `Integer`, как это и определено для упомянутых языков, когда речь идет о 32-разрядном целом числе со знаком (32-bit signed integer). Кроме того, в коде на C# и VB определена переменная `age` с использованием встроенного типа .NET, `Int32`. Обратите внимание, что тип .NET одинаков для обоих языков, например:

Листинг 2.11. Пример кода на C#, иллюстрирующий объявление переменных

```
int age1;
Int32 age2;
```

Листинг 2.12. Пример кода на VB, иллюстрирующий объявление переменных

```
Dim age1 as Integer
Dim age2 as Int32
```

Фактически, тип .NET всегда будет одинаков для всех языков, поддерживаемых платформой .NET. Каждый язык использует собственный синтаксис для типов .NET, и каждый из типов, специфичных для конкретного языка, является псевдонимом для типа .NET.

Выражения

Вычисления, выполняемые кодом программы, осуществляются выражениями (expressions), которые представляют собой комбинации переменных, операторов (например, сложения или умножения) или ссылок на другие члены класса. В следующем примере кода приведено выражение, которое осуществляет вычислительную операцию и присваивает результат целой переменной:

На C# такое выражение может выглядеть следующим образом:

```
int result = 3 + 5 * 7;
```

На VB это выражение будет выглядеть так:

```
Dim result As Int32 = 3 + 5 * 7
```

Переменная `result` в приведенном примере в C# определена с использованием типа `int`, а в VB — с помощью типа `Int32` (см. Табл. 2.2). Назвать переменную можно как угодно; в данном примере выбрано имя `result`. В коде на VB тип новой переменной — `Int32`, встроенный тип данных .NET. Вместо этого вы могли бы использовать и ключевое слово `Integer`, которое является псевдонимом для `Int32`. Выражение `3 + 5 * 7` представляет собой комбинацию операторов `+` (сложение) и `*` (умножение). В ходе выполнения программы значение этого выражения вычисляется, а результат присваивается переменной `result`. Это значение будет равно 38, потому что выражения используют стандартные правила алгебраических при-

оритетов. В рассматриваемом примере сначала вычисляется значение `5 * 7`, потому что приоритет имеет операция умножения. Результат умножения складывается с числом 3.

Чтобы изменить порядок операций при вычислении значения выражения, используются скобки. В следующем примере сначала выполняется сложение 3 и 5, а затем полученный результат умножается на 7:

На C# выражение со скобками будет выглядеть так:

```
int differentResult = (3 + 5) * 7;
```

На VB то же самое выражение будет выглядеть следующим образом:

```
Dim differentResult As Int32 = (3 + 5) * 7
```

Вследствие группировки операндов при помощи скобок, переменная `differentResult` после вычисления выражения получит значение 56.

Тернарные операторы C# и операторы Immediate If в VB

Тернарные операторы C# и операторы `immediate if` (`iif`)[2] в VB позволяют проверить истинность условия и, в зависимости от его соблюдения или несоблюдения, возвратить другое значение. В листингах 2.13 и 2.14 показано, как работают тернарные операторы и операторы `immediate if`.

Листинг 2.13. Пример тернарного оператора в C#

```
int bankAccount = 0;
string accountString = bankAccount = = 0 ? "checking" : "savings";
```

Листинг 2.14 Пример оператора immediate if в VB

```
Dim accountString As String =

        IIf(bankAccount = 0, "checking", "saving")
```

Условная часть этого оператора оценивает значение переменной `bankAccount` и проверяет, равно ли оно нулю. Проверка выполняется во время работы программы ("at runtime"). Если условие истинно (`true`), то возвращается значение первого выражения, которое следует сразу же за знаком вопроса (C#) или сразу же за запятой (VB). В данном примере этим значением будет `checking`. В противном случае, если условие не соблюдается, то его значение устанавливается как `false`, и возвращается значение второго выражения, следующего за двоеточием (C#) или за второй запятой (VB). Возвращаемое значение будет либо строкой `checking`, либо (как в данном примере) объявленной переменной `accountString` присваивается строковое значение `saving`.

[2] `Immediate IF` или "немедленный IF" представляет собой упрощенный вариант конструкции `IF...Then`, когда проверяется условие и возвращается одно из двух значений. — *Прим. перев.*

Примечание

В ранних версиях языка программирования VB требовалось завершать строку утверждения, продолжающегося со следующей строки, символом подчеркивания (_). В последней версии VB символы продолжения строки больше не являются обязательными. Если раньше вы уже программировали на VB, возможно, что отсутствие символов продолжения строки уже привлекло ваше внимание. Не беспокойтесь, это не ошибка, теперь такой синтаксис абсолютно правилен и легален.

Перечисления

Перечисление (enum) позволяет задать набор значений, которые легко читаются и распознаются в коде программы. Пример, который мы сейчас рассмотрим, создает перечисление (enum), содержащее список типов банковских счетов, например, `checking` (проверка), `saving` (сбережения) и `loan` (кредит). Чтобы создать перечисление, откройте новый файл, щелкнув мышью по имени проекта, из контекстного меню выберите команды **Add** | **New Item** | **Code File**, назовите его, например, `BankAccounts.cs` (или `BankAccounts.vb`), и вы увидите новый пустой файл. Введите код перечисления, показанный в листингах 2.15 и 2.16.

Листинг 2.15. Пример перечисления на C#

```
public enum BankAccount
{
        Checking,
        Saving,
        Loan
}
```

Листинг 2.16. Пример перечисления на VB

```
Enum BankAccount
        Checking
        Saving
        Loan
End Enum
```

В листингах 2.17 и 2.18 показано, как можно работать с новым перечислением, которое называется `BankAccount`.

Листинг 2.17. Использование перечислений в C#

```
BankAccount accountType = BankAccount.Checking;

string message =
        accountType = = BankAccount.Checking ?
                    "Bank Account is Checking" :
                    "Bank Account is Saving";
```

Листинг 2.18. Использование перечислений в C#

```
Dim accountType As BankAccount = BankAccount.Checking


Dim message =
       IIf(accountType = BankAccount.Checking,
             "Bank Account is Checking",
             "Bank Account is Saving")
```

Переменная `accountType` имеет тип `BankAccount` (перечисление). Она инициализируется значением `Checking`, которое является членом перечисления `BankAccount`. Следующее утверждение использует тернарный оператор для проверки статуса банковского счета `accountType` и выясняет, что переменная имеет значение `Checking`. Если это так, сообщение получает значение, указанное в первой строке. В противном случае сообщение инициализируется второй строкой. Естественно, что в данном случае будет использована первая строка, потому что приведенный пример очень прост — он изначально закодирован таким образом.

Ветвления

Утверждения, выполняющие ветвления (branching), позволяют выбрать путь, по которому программа будет выполняться дальше, в зависимости от конкретного условия.

Например, представьте себе ситуацию, когда вам требуется предоставить скидку постоянному клиенту. Условие определяет, принадлежит ли клиент к группе тех, кто имеет право на скидку, или нет. Возможные варианты дальнейших действий — либо клиенту предоставляется скидка, либо за сервис запрашивается полная цена. Для этой цели применяются два основных типа утверждений ветвления: `if` и `switch` (`Select Case` в VB). В последующих разделах будет показано, как реализовать логику ветвления программы с помощью утверждений `if` и `switch`.

Выражения

Утверждения `if` позволяют выполнить операцию только в том случае, когда логическое выражение, представляющее собой условие, в процессе выполнения программы получает значение `true`. Рассмотрим пример (листинги 2.19 и 2.20), в котором на консоль выводится утверждение, если переменная `result` получает значение большее, чем 48, в результате использования оператора сравнения `>` (greater than).

Листинг 2.19. Пример логического выражения для C#

```
if (result > 48)
{
        Console.WriteLine("result is > 48");
}
```

Листинг 2.20. Пример логического выражения для VB

```
If result > 48 Then
        Console.WriteLine("Result is > 48")
End If
```

В C# фигурные скобки не являются обязательными, если у вас есть только одно утверждение, которое должно быть выполнено после того, как условие станет истинным (получит логическое значение `true`). Однако использование фигурных скобок станет обязательным, если при соблюдении условия выполнены должны быть два или большее количество выражений. Условие при этом должно получить одно из двух логических (Boolean) значений — либо `true`, либо `false`. Кроме того, здесь можно использовать раздел `else`, который выполняется в случае несоблюдения условия. Раздел (clause) — это еще один метод указать, что конкретный элемент является частью другого, составного выражения. Ключевое слово `else` не используется в качестве выражения как такового, поэтому оно и называется разделом (clause). Пример использования ключевого слова `else` приведен в листингах 2.21 и 2.22.

Листинг 2.21. Пример использования ключевого слова else в C#

```
if (result > 48)
{
        Console.WriteLine("result is > 48");
}
else
{
        Console.WriteLine("result is <= 48");
}
```

Листинг 2.22. Пример использования ключевого слова else в VB

```
If result > 48 Then
        Console.WriteLine("Result is > 48")
Else
        Console.WriteLine("Result is <= 48")
End If
```

В только что приведенном примере показано, что если результат вычисления выражения представляет собой число, не превышающее 48, то оно должно быть либо меньшим, либо равным этому значению (48).

Фрагменты if и else

Фрагмент `if` создает шаблон, на основе которого вы сможете построить утверждение `if`. Чтобы воспользоваться этим фрагментом, введите с клавиатуры строку `if`, а затем нажмите клавиши <TAB>, <TAB>. На экране появится окно, показанное на рис. 2.11 (для C#) или на рис. 2.12 (для VB).

Как показано на рис. 2.11, курсор будет находиться в поле формы шаблона, выделенном подсветкой. В этом поле нужно указать условие для утверждения `if`. Для C# введите условие, значение которого требуется вычислить, и нажмите клавишу <ENTER>. Фрагмент завершит работу, поместив курсор в пределах блока утверждения `if`. Для VB просто поместите курсор туда, откуда вы хотите продолжить ввод.

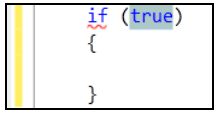


Рис. 2.11. Шаблон утверждения `if` (C#)

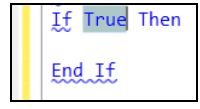


Рис. 2.12. Шаблон утверждения `if` (VB)

В языке C# шаблон утверждения `else` аналогичен шаблону для утверждения `if`. Чтобы воспользоваться этим шаблоном, введите `else` и нажмите два раза клавишу <TAB>. Появится форма шаблона `else` с курсором, расположенным между блоками `else`. В VB автоматически генерируемого фрагмента `else` нет. Просто введите ключевое слово `Else` между последними утверждениями `If` и `End If`.

Утверждения Switch/Select

Утверждение `switch` (утверждение `Select Case` для VB) сообщает программе о необходимости вычислить одно из множества условий и, в зависимости от результата, осуществить ветвление. Рассмотрим пример, который выполняет различные действия, в зависимости от того, какое значение получает переменная `name`. Примеры кода, как всегда, приведены в листингах 2.23 (для C#) и 2.24 (для VB).

Листинг 2.23. Пример кода, иллюстрирующего использование выражения switch для C#

```
var name = "Megan";

switch (name)
{
        case "Joe":
                Console.WriteLine("Name is Joe");
                break;
        case "Megan":
                Console.WriteLine("Name is Megan");
                break;
        default:
                Console.WriteLine("Unknown Name");
                break;
}
```

Листинг 2.24. Пример кода, иллюстрирующего использование выражения Select Case для VB

```
Dim name As String = "Megan"

Select Case name
        Case "Joe"
                Console.WriteLine("Name is Joe")
        Case "Megan"
                Console.WriteLine("Name is Megan")
        Case Else
                Console.WriteLine("Unknown name")
End Select
```

В примере на C# используется ключевое слово `switch`, а определение значения осуществляется в блоке, заключенном в фигурные скобки. Выполнение кода будет зависеть от того, какое из утверждений `case` дает результат, соответствующий значению `switch`. Случай, определяющийся ключевым словом `default`, выполняется в том случае, когда совпадений не найдено. Ключевое слово `break` является обязательным. Когда программа исполняет утверждение `break`, выполнение утверждения `switch` прекращается, и исполнение программы продолжается со следующего утверждения, идущего за последней закрывающей скобкой утверждения `switch`.

Что касается примера на VB, то утверждение `Select Case` использует переменную `name` как условие и исполняет код на основании соответствия, найденного для переменной `name`. Блок кода `Case Else` будет выполнен, если соответствие не было найдено.

Фрагменты кода для утверждения Switch

Для фрагментов утверждения `switch` есть два сценария: минимальное утверждение `switch` и расширенное утверждение `switch` с ключевыми словами `enum`. Сначала протестируем минимальное утверждение `switch`, два раза нажав клавишу <TAB>. В результате на экране появится шаблон утверждения `switch`, показанный на рис. 2.13.

В этом случае можно заместить значение `switch_on` на рис. 2.13 значением, которое вам требуется использовать в утверждении `switch`. После того как вы нажмете клавишу <ENTER>, вы увидите следующий фрагмент кода, приведенный в листинге 2.25, в который развернется утверждение `switch` с ключевым словом `default`.

Листинг 2.25. Фрагмент кода на C#, в который развернется утверждение `switch` с ключевым словом `default`

```
switch (name)
{
        default:
              break;
}
```

Утверждения VB `Select` работают аналогично оператору `switch` в C#. Введите с клавиатуры символы **Se** и нажмите клавиши <TAB>, <TAB>. На экране появится шаблон VB, как показано на рис. 2.14.

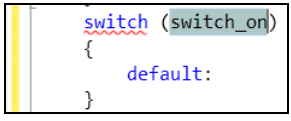


Рис. 2.13. Шаблон фрагмента кода `switch`

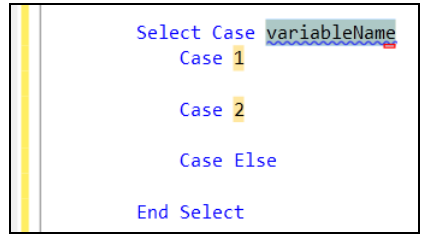


Рис. 2.14. Шаблон фрагмента кода `Select Case`

В C# нужные утверждения `case` обычно добавляются вручную. Однако фрагмент кода `switch` еще более упрощает использование перечислений (enums), автоматически создавая утверждения `case` для каждого из значений, указанных в `enum`. В следующем примере мы используем переменную `accountType` с типом данных `enum BankAccount` из листингов 2.15 и 2.16. Чтобы проверить, как утверждение `switch` работает с перечислениями (enums), введите с клавиатуры начальные символы этого утверждения, **sw**, и нажмите клавиши <TAB>, <TAB>. На экране появится шаблон утверждения `switch`, в котором будет подсвечено поле условия. Введите в это поле **accountType** и нажмите клавишу <ENTER>. Фрагмент `switch` автоматически сгенерирует случаи для членов перечисления `BankAccount`. Выглядеть этот код будет так, как показано в листинге 2.26.

Листинг 2.26. Фрагмент `switch` автоматически сгенерировал случаи для членов перечисления `BankAccount`

```
switch (accountType)
{
        case BankAccount.Checking:
              break;
        case BankAccount.Saving:
              break;
        case BankAccount.Loan:
              break;
        default:
              break;
}
```

Перечисление (enum) удобочитаемо, и, кроме того, автоматическая генерация кода минимизирует риск опечаток, который всегда существует при ручном вводе строк с клавиатуры. Теперь, познакомившись с ветвлениями, перейдем к изучению циклов (loops).

Циклы

Циклов существует множество: `for`, `for each`, `while` и `do`. В последующих нескольких разделах мы разберем, как работает каждый из них.

Циклы For

Циклы `for` позволяют указать условие, которое указывает, сколько раз должен выполняться заданный блок утверждений.

Рассмотрим пример, приведенный в листингах 2.27 и 2.28.

Листинг 2.27. Пример цикла for для C#

```
for (int i = 0; i < 3; i++)
{
        Console.WriteLine("i = " + i);
}
```

Листинг 2.28. Пример цикла for для VB

```
For i As Integer = 0 To 2
        Console.WriteLine("i = " & i)
Next
```

В только что приведенном примере на C# переменная `i` — это переменная цикла, имеющая тип данных `int`. Цикл будет выполняться до тех пор, пока значение i меньше 3, а само это значение увеличивается на единицу после каждого исполнения цикла. Проверка условия `i < 3` выполняется каждый раз перед очередной итерацией, и если выражение ложно, то цикл выполняться не будет.

> ***Примечание***
>
> Операторы `+` и `&` из приведенного в примере фрагмента кода выполняют операцию конкатенации строк. Хотя `i` — это целая переменная (`integer`), перед конкатенацией она будет преобразована к типу `string`.

В VB цикл `for` инициализирует переменную `i` целым значением (`integer`) и повторяет цикл три раза (переменная `i` пробегает значения с 0 до 2, включительно).

Фрагмент кода для цикла for

Чтобы использовать автоматическую генерацию кода для цикла `for` в C#, введите с клавиатуры начальные символы утверждения, **fo**, и нажмите клавиши <TAB>, <TAB>. На экране появится шаблон фрагмента, показанный на рис. 2.15.

Та же самая последовательность ввода с клавиатуры (**fo**, <TAB>, <TAB>) действует и для автоматически генерируемых фрагментов кода цикла `for` в VB, за тем исключением, что в данном случае вы увидите на экране шаблон, показанный на рис. 2.16.

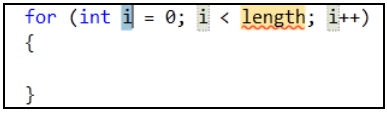


Рис. 2.15. Шаблон цикла `for` для C#

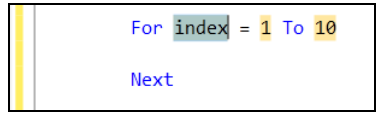


Рис. 2.16. Шаблон цикла `for` для VB

Шаблон цикла `for` в C# отличается от предыдущих шаблонов тем, что вам нужно заполнить два поля. Во-первых, вам необходимо дать имя переменной цикла (по умолчанию ей присваивается имя `i`). Затем, после нажатия клавиши <TAB>, фокус ввода передается полю, задающему размер цикла, в котором указывается `Length` как шаблон.

Если вам нравится имя переменной цикла `i`, которое является общепринятым, просто нажмите клавишу <TAB> и задайте продолжительность цикла. После завершения вы увидите готовый цикл `for`, и курсор будет находиться внутри него.

Циклы For Each

Циклы `For each` позволяют выполнить блок кода для всех элементов массива (array) или коллекции (collection). Массивы содержат объекты, которые хранятся в памяти в виде списков.

Коллекции представляют собой более сложные структуры. Коллекции хранятся в памяти в различных формах, которые имеют вид стеков (Stack), списков (List), очередей (Queue) и так далее. В листингах 2.29 и 2.30 приведены примеры цикла, который обрабатывает массив строк.

Листинг 2.29. Пример цикла, обрабатывающего массив строк (для C#)

```
string[] people = { "Megan", "Joe", "Silvia" };

foreach (var person in people)
{
        Console.WriteLine(person);
}
```

Листинг 2.30. Пример цикла, обрабатывающего массив строк (для VB)

```
Dim people = {"Megan", "Joe", "Silvia"}

For Each person As String In people
        Console.WriteLine(person)
Next
```

В этом примере `people` — это массив, состоящий из трех текстовых строк. Блок кода в этом цикле будет исполнен три раза, по одному для каждого элемента массива. Каждая из итераций цикла присваивает имя конкретному человеку.

Фрагмент кода для цикла For Each

Чтобы добавить автоматически генерируемый код, используя шаблон C#, введите с клавиатуры начальные символы утверждения, **fore**, и нажмите клавиши <TAB>, <TAB>. На экране появится шаблон, показанный на рис. 2.17.

Шаблон цикла `for each` дает вам возможность заполнить три поля. Здесь `var` — это неявный указатель типа, который позволяет избежать указания типа элемента. Компилятор определит его самостоятельно, зато вы сэкономите несколько клавиатурных нажатий. Элемент `field` будет представлять собой коллекцию типов элемента. Вы можете оставить переменную `var` "как есть" или же явно указать для нее тип (в нашем примере это будет тип `string`). Чтобы перемещаться между полями, пользуйтесь клавишей <TAB>. В частности, рекомендуется добавить несущие смысловую нагрузку идентификаторы, как для элемента, так и для коллекции, по которой будут выполняться итерации цикла.

Чтобы автоматически создать фрагмент кода для цикла `For Each` на VB, введите с клавиатуры последовательность <?>, <TAB>, <C>, <ENTER>, <C>, <ENTER>, <f>, <ENTER>, и вы увидите окно шаблона цикла `For Each`, показанное на рис. 2.18.

```
foreach (var item in collection)
{

}
```

Рис. 2.17. Шаблон автоматически генерируемого фрагмента кода для цикла `for each` на C#

```
For Each item As String In collectionObject

Next
```

Рис. 2.18. Шаблон автоматически генерируемого фрагмента кода цикла `For Each` для VB

Циклы While

Цикл `while` позволяет выполнять блок кода до тех пор, пока указанное условие продолжает оставаться истинным. Рассмотрим пример, в котором ведется обратный отсчет (листинги 2.31 и 2.32).

Листинг 2.31. Пример на C#, иллюстрирующий применение цикла while

```
int count = 3;

while (count > 0)
{
        Console.WriteLine("count: " + count);
        count--;
}
```

Листинг 2.32. Пример на VB, иллюстрирующий применение цикла while

```
Dim count As Integer = 3

While count > 0
        Console.WriteLine("count: " & count)
        count -= 1
End While
```

Цикл `while` исполняется до тех пор, пока значение счетчика цикла остается больше нуля. Поскольку значение счетчика изначально равно 3 и каждый раз после выполнения очередной итерации оно будет уменьшаться на единицу, то счетчик поочередно примет значения 3, 2, 1, и после этого цикл прекратит выполнение. Однако будьте внимательнее — позаботьтесь о том, чтобы не создать бесконечный цикл.

Автоматически генерируемый фрагмент кода для цикла while

Чтобы автоматически создать фрагмент кода для цикла `while`, введите с клавиатуры начальные символы утверждения, **wh**, и нажмите клавиши <TAB>, <TAB>. Вы увидите шаблон, показанный на рис. 2.19 (C#) или рис. 2.20 (VB).

Для C#, заполнение поля `condition` и нажатие клавиши <ENTER> завершает создание блока кода, и вы увидите вновь сгенерированный фрагмент с курсором, помещенным в его пределы.

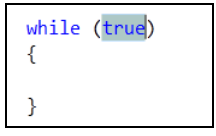


Рис. 2.19. Шаблон автоматически генерируемого фрагмента кода цикла `while` для C#

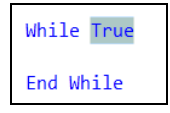


Рис. 2.20. Шаблон автоматически генерируемого кода на VB для цикла `while`

Циклы Do

Цикл `do` применяется в тех случаях, когда требуется выполнить код цикла хотя бы один раз. Рассмотрим пример, демонстрирующий создание простого меню, принимающего пользовательский ввод (листинги 2.33 и 2.34).

Листинг 2.33. Пример на C#, демонстрирующий создание простого меню

```
string response = "";

do
{
        Console.Write("Press 'Q' and Enter to break: ");
        response = Console.ReadLine();
} while (response != "Q");
```

Листинг 2.34. Пример на VB, демонстрирующий создание простого меню

```
Do
        Console.Write("Press Q and Enter to break: ")
        response = Console.ReadLine()
Loop While response <> "Q"
```

В этом примере вы всегда будете получать приглашение к вводу следующего вида: **Press 'Q' and Enter to break:**. Функция `Console.ReadLine` принимает пользовательский ввод, который имеет тип данных `string`. Если пользовательский ввод представляет собой строку, состоящую из одного заглавного символа **Q**, выполнение цикла прекращается.

В VB имеется еще одна вариация этого цикла, использующая ключевое слово `Until` (листинг 2.35).

Листинг 2.35. Код цикла Do на VB с использованием ключевого слова Until

```
Do
        Console.Write("Press Q and Enter to break: ")
        response = Console.ReadLine()
Loop Until response = "Q"
```

В этом коде условие `Until` управляет выполнением цикла, которое будет повторяться до тех пор, пока условие не будет выполнено. Этот вариант представляет собой противоположность цикла `Do While`.

Автоматически генерируемый фрагмент кода для цикла Do

Чтобы воспользоваться автоматической генерацией кода для цикла `do`, введите с клавиатуры символы **do**, а затем дважды нажмите клавишу <TAB>. На экране появится шаблон цикла `do`, показанный на рис. 2.21.

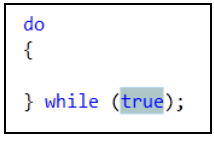


Рис. 2.21. Шаблон автоматически генерируемого кода на C# для цикла `do`

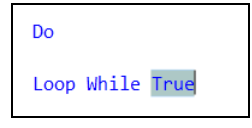


Рис. 2.22. Шаблон автоматически генерируемого фрагмента кода для цикла `Do While` в VB

Введите условие в поле `condition` для цикла `do`, а затем нажмите клавишу <ENTER>. Курсор будет помещен в пределы блока кода цикла `do`.

Чтобы воспользоваться возможностью автоматической генерации кода цикла `do` для VB, введите с клавиатуры <?>, <TAB>, <C>, <ENTER>, <C>, <ENTER>, а затем воспользуйтесь клавишами с изображением стрелок, чтобы выбрать нужный вам вариант цикла `Do`. На рис. 2.22 показан пример шаблона цикла `Do While`.

Заключение

Работа с поддерживаемыми языками программирования является базовым навыком, необходимым разработчику приложений .NET. Двумя наиболее популярными языками .NET являются C# и VB. Именно поэтому данная глава и рассматривала только их. В данной главе рассказывалось о типах, выражениях, утверждениях, блоках кода, условиях и ветвлениях. Кроме того, было рассказано об основных функциях VS, облегчающих труд программиста, например, таких, как редактор кода (code editor), закладки (bookmarks), Intellisense, а также автоматически генерируемых фрагментах кода (snippets).

В *главе 3* будет сделан следующий шаг в изучении языков программирования, в том числе — будет рассказано о классах и их членах.

Глава 3

Изучение основ C# и VB.NET: типы и члены

В этой главе вы ознакомитесь со следующими важными аспектами программирования на C# и VB:

- Создание классов
- Написание кода методов
- Поля кода и их свойства

Тип (type) — это общий термин для обозначения таких понятий, как классы (classes), модули (modules), перечисления (enums) и т. д. В этой главе будут подробно обсуждаться типы классов (class type), с помощью которых вы сможете создавать собственные, индивидуально настроенные типы. Кроме того, будет продемонстрирована ценность классов — после того, как вы узнаете больше о членах класса (class members). Вы научитесь использовать такие члены класса, как поле (field), метод (method) и свойство (property). Начнем обсуждение с создания и использования классов.

Создание классов

Ранее в этой книге уже было рассказано о том, как использовать примитивы (primitive types), встроенные в поддерживаемые платформой .NET языки программирования и являющиеся псевдонимами для еще более низкоуровневых типов .NET. Кроме использования встроенных типов, вы можете создавать и собственные. Это делается через классы, экземпляры которых вы можете создавать (instantiate) и на основе которых вы можете создавать новые объекты. В следующем разделе будет рассказано о том, как создать класс, а затем, с его помощью, создать экземпляр объекта.

Синтаксис класса

Чтобы создать новый, индивидуально определенный (custom) класс, щелкните правой кнопкой мыши по названию проекта, выберите из контекстного меню команды **Add | Class**, назовите класс именем по своему выбору (в нашем примере это будет класс `Employee`) и введите расширение имени файла: .cs для C# или .vb — для VB.

Затем щелкните по кнопке **Add** (VS добавит расширение автоматически, если вы не сделаете этого сами). Вы получите файл, код которого показан в листингах 3.1 и 3.2.

Листинг 3.1. Новый класс Employee (код на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram

{
        public class Employee
        {
                public string FirstName;
        }
}
```

Листинг 3.2. Новый класс Employee (код на VB)

```
Public Class Employee
    Public Dim FirstName As String
End Class
```

В C# класс `Employee` будет почти идентичен классу `Program`, который вы создавали в предыдущей главе, с тем исключением, что его имя будет другим — в нашем примере класс будет называться `Employee`. Что касается VB, то до сих пор вы создавали только модули, и `Employee` будет первым классом VB, обсуждаемым в данной книге. В состав класса можно добавлять новые члены, которыми могут быть события (events), поля (fields), методы (methods) и свойства (properties). В листингах 3.1 и 3.2 показан пример поля (`FirstName`), а о событиях, методах и свойствах будет рассказано в последующих разделах данной главы. Поле (field) — это переменная класса, которая содержит информацию, характерную для этого класса.

В листингах 3.3 и 3.4 показано, как создать экземпляр объекта типа `Employee`, используя новый, только что созданный вами тип, и как пользоваться этим экземпляром. Этот код можно поместить в состав `Main` или любого другого метода. В одном из последующих разделов этой главы методам будет уделено более пристальное внимание.

Листинг 3.3. Пример кода на C#, использующего класс

```
Employee emp = new Employee();
emp.FirstName = "Joe";
```

Листинг 3.4. Пример кода на VB, использующего класс

```
Dim emp As New Employee
emp.FirstName = "Joe"
```

В листингах 3.3 и 3.4 объявлена переменная, имеющая тип `Employee`. Новый экземпляр объекта в C# создается как `Employee()`, а в VB — как `New Employee`, и этот экземпляр присваивается переменной `emp`. С помощью этого нового экземпляра вы можете получать доступ к объекту `Employee`, включая и его члены. В примере из листингов 3.3 и 3.4 полю `FirstName` данного экземпляра объекта `Employee` присваивается строковое значение `"Joe"`. Таким образом, вы видите, что объект может содержать данные.

Теперь можно определить новый класс, создать экземпляр этого класса и пользоваться им. В следующем разделе мы обсудим такое свойство классов, как наследование (inheritance).

Наследование классов

Класс может использовать члены другого класса, используя такую возможность, как наследование (inheritance). В терминах программирования, потомком (child) называется класс, созданный на основе другого класса, который в этой терминологии будет называться родительским классом или просто родителем (parent). Класс-потомок при этом может наследовать от родительского класса такие его члены (например, поля или методы), наследование которых допускается родительским классом. В примере, который мы сейчас рассмотрим, будет создан новый класс `Cashier`, который является потомком класса `Employee`, созданного чуть ранее. Чтобы создать новый класс, щелкните правой кнопкой мыши по названию проекта, и из контекстного меню выберите команды **Add | Class**, Затем присвойте имя новому классу (в нашем примере — `Cashier`). В листингах 3.5 и 3.6 показан код нового класса и модификации, которые реализуют наследование.

Листинг 3.5. Пример на C#, иллюстрирующий наследование классов

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram
{
        public class Cashier : Employee
        {
        }
}
```

Листинг 3.6. Пример на VB, иллюстрирующий наследование классов

```
Public Class Cashier
        Inherits Employee


End Class
```

В C# отношение наследования между классами обозначается символом двоеточия. Обратите внимание на символ двоеточия, следующий за идентификатором `Cashier`, сразу же за которым указывается имя класса, от которого производится наследование, `Employee`. В VB для этой цели используется ключевое слово `Inherits`. Это ключевое слово вводится с новой строки, а за ним следует имя класса, от которого происходит наследование. В принципе, это означает, что класс `Cashier` имеет все те члены, которые принадлежат к классу `Employee`. Преимущества, предоставляемые наследованием, продемонстрированы примером в листингах 3.7 и 3.8.

Листинг 3.7. Код на C#, использующий наследование

```
Cashier cashr = new Cashier();
cashr.FirstName = "May";
```

Листинг 3.8. Код на VB, использующий наследование

```
Dim cashr As New Cashier
cashr.FirstName = "May"
```

В соответствии с кодом, приведенным в листингах 3.7 и 3.7, в `Cashier` нет поля с именем `FirstName`. Однако в классе `Employee` есть поле с именем `FirstName`, и класс `Cashier` является производным от `Employee`. Благодаря наследованию, `Cashier` автоматически унаследует поле `FirstName`, и код, приведенный в листингах 3.7 и 3.8, абсолютно законен. Наследование можно рассматривать как некую разновидность специализации — например, класс `Cashier` представляет собой специализированный вариант класса `Employee`. Чтобы воспользоваться преимуществом этой специализации, вы можете добавить к новому классу `Cashier` еще одно поле, `assignedCashRegister`. После этого класс `Cashier` не только будет иметь все поля и методы класса `Employee`, но и сможет хранить значение регистрационного имени или номера. Экземпляр класса `Employee` не сможет содержать этой информации. Инфраструктура .NET Framework широко использует наследование, благодаря чему и может предлагать разработчикам множество библиотек классов.

> ***Совет***
>
> При описании отношений наследования между классами часто можно сказать, что дочерний класс является представителем родительского. Допустим, в рассматриваемом примере мы можем сказать, что класс `Cashier` является представителем класса `Employee` ("Cashier is an Employee" — и в самом деле, кассир является служащим).

Если при программировании вы будете применять этот подход, причем построенные фразы будят выглядеть вполне логично (как в нашем случае — все кассиры являются служащими, но не каждый из служащих обязательно должен быть кассиром), тогда, по всей вероятности, вы правильно пользуетесь наследованием.

Автоматически генерируемый фрагмент кода класса

В C# есть автоматически генерируемый фрагмент кода класса, в то время как в VB его нет. Прежде чем воспользоваться возможностью автоматической генерации кода класса, создайте новый класс. Для этого щелкните правой кнопкой мыши по названию проекта, выберите из контекстного меню команды **Add** | **New Item** | **Code File** и присвойте имя новому файлу. В нашем примере назовем новый файл Manager. Теперь вы увидите пустой файл, с которым можно начинать работать. Чтобы воспользоваться автоматически генерируемым фрагментом кода класса, введите начальные символы ключевого слова, **cl** и нажмите клавиши <TAB>, <TAB>. На экране появится шаблон фрагмента кода класса, показанный на рис. 3.1.

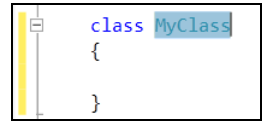


Рис. 3.1. Шаблон автоматически генерируемого фрагмента кода класса на C#

В этом шаблоне достаточно ввести в поле имени имя нового класса и нажать клавишу <ENTER>. Новый класс будет создан, и курсор будет находиться в пределах блока класса. Теперь, научившись создавать классы, пора начинать с ними работать — например, добавлять новые члены. Начнем обсуждение с написания методов.

Написание методов

Программные алгоритмы удобно разбивать на блоки кода, называемые методами. В разных языках программирования методы называются по-разному. Чаще всего встречаются такие названия, как функции (functions), процедуры (procedures) и подпрограммы (subroutines). В данной книге будет использоваться наиболее общий термин — метод (method), за исключением особо оговоренных случаев. С методами вы уже познакомились при написании `Console.WriteLine`, где `WriteLine` — это метод класса `Console`. Метод содержит одно или несколько утверждений. Причины, по которым программисты создают методы, включают необходимость разбиения кода программы на модули, чтобы изолировать и собрать все сложные операции в одном блоке или сгруппировать в одном модуле наиболее общие и часто используемые операции, чтобы упростить их повторное использование. В следующем разделе будет показано, как декларировать и использовать методы.

Декларирование и использование методов

Для начала, давайте рассмотрим очень простой метод. На его примере будут продемонстрированы синтаксис и логика исполнения программы. В листингах 3.9 и 3.10 утверждение `Console.Writeline` из метода `Main`, обсуждавшегося в *главе 2*,

будет перемещено в новый контейнерный метод, а в метод `Main` будет добавлено утверждение, вызывающее этот новый метод.

Листинг 3.9. Объявление и вызов нового метода на C#

```
// Program.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram
{
        class Program
        {
              static void Main(string[] args)
              {
                    MessagePrinter msgPrint = new MessagePrinter();
                    msgPrint.PrintMessageInstance();
              }
        }
}
// MessagePrinter.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram
{
        class MessagePrinter
        {
                public static void PrintMessageStatic()
                {
                        Console.WriteLine("Hello from a static method.");
                }
                public void PrintMessageInstance()
                {
                        Console.WriteLine("Hello from an instance method.");
                }
         }
}
```

Листинг 3.10. Объявление и вызов нового метода на VB

```
' VB (Module1.vb):

Module Module1
    Sub Main()
        MessagePrinter.PrintMessageShared()

        Dim msgPrint As New MessagePrinter()
        msgPrinter.PrintMessageInstance()
    End Sub
End Module

' VB (MessagePrinter.vb)

Public Class MessagePrinter
    Public Shared Sub PrintMessageShared()
         Console.WriteLine("Hello from a shared method.")
    End Sub

    Public Sub PrintMessageInstance()
         Console.WriteLine("Hello from an instance method.")
    End Sub
End Class
```

В листингах 3.9 и 3.10 показаны два типа методов — статический (static) и экземпляр (instance). В VB совместно используемые (shared) методы представляют собой то же самое, что и статические. Определить, к какому типу принадлежит метод, можно по ключевым словам. Статические методы имеют модификатор `modifier` (в VB — `shared`), а экземпляры модификатора не имеют. В первую очередь, давайте рассмотрим статические (совместно используемые) методы — сначала рассмотрим, как они объявляются, а затем — как они вызываются.

Статический (в VB — разделяемый) метод `PrintMessageStatic` (в VB — `PrintMessageShared`) имеет модификатор доступа `public`. Это значит, что любой другой код, использующий класс-контейнер, `MessagePrinter`, сможет использовать этот метод. Если вы не включите модификатор доступа `public`, то метод по умолчанию будет частным (`private`), и пользоваться им сможет только код, находящийся в составе класса `MessagePrinter`.

Метод `PrintMessageStatic` имеет ключевое слово `void`. Это означает, что данный метод не возвращает никакого значения. В VB если метод не возвращает значения, он объявляется как `Sub`, что и показано в листинге 3.10. Далее будет показано, как создавать методы, которые возвращают значения вызывающему коду, который инициирует данный метод. Пустой список параметров, добавленный к методу `PrintMessageStatic` (`PrintMessageShared` — VB) означает, что метод не имеет никаких параметров. Параметры позволяют вызывающему коду передавать методу информацию. Вскоре мы обсудим этот вопрос в подробностях.

В пределах блока метода имеется утверждение `Console.WriteLine`. Вы можете добавлять в состав метода столько утверждений, сколько необходимо, чтобы реализовать поставленную перед вами задачу. Далее мы рассмотрим, как следует вызывать `PrintMessageStatic` (`PrintMessageShared` — в VB). Эта задача выполняется кодом, приведенным в листингах 3.11 и 3.12.

Листинг 3.11. Код на C#, вызывающий `PrintMessageStatic`

```
Program.PrintMessageStatic();
```

Листинг 3.12. Код на VB, вызывающий `PrintMessageShared`

```
MessagePrinter.PrintMessageShared()
```

Только что приведенный пример представляет собой выражение, содержащееся в методе `Main` и вызывающее функцию `Program.PrintMessageStatic` (`PrintMessageShared` — в VB).

Обратите внимание на класс (или тип), который содержит все методы и называется `MessagePrinter`. В C# статический метод вызывается через содержащий его тип — именно поэтому метод `PrintMessageStatic` вызывается с использованием префикса `Program`. В VB совместно используемые (разделяемые) методы можно инициировать или через тип методы, или через экземпляр этого типа. Экземпляры методов мы сейчас и рассмотрим.

Следующий метод, `PrintMessageInstance`, представляет собой экземпляр метода. Обратите внимание, что он не имеет модификатора `static`. Остальная часть определения метода в точности такая же, как и у метода `PrintMessageStatic`.

Так как `PrintMethodInstance` представляет собой метод-экземпляр, его вызов осуществляется по-другому, через экземпляр содержащего его типа, как показано в следующих фрагментах из листингов 3.9 и 3.10.

Листинг 3.13. Вызов `PrintMethodInstance` на C# (фрагмент листинга 3.9)

```
MessagePrinter msgPrint = new MessagePrinter();
msgPrint.PrintMessageInstance();
```

Листинг 3.14.Вызов `PrintMethodInstance` на VB (фрагмент листинга 3.10)

```
Dim msgPrint As New MessagePrinter()
msgPrinter.PrintMessageInstance()
```

Как показано в этом примере, `msgPrint` имеет тип `MessagePrinter`. Используя утверждение `new`, `MessagePrinter` создает новый экземпляр `MessagePrinter` во время исполнения программы и присваивает его переменной `msgPrint`. Теперь, когда мы создали экземпляр `MessagePrinter` и сослались на него с помощью переменной `msgPrint`, можно вызывать метод `PrintMessageInstance` через переменную `msgPrint`. Теперь рассмотрим, как добавлять параметры к методу, и обсудим, почему это важно.

Объявление параметров и передача аргументов

Передача параметров методу представляет собой отличный способ написания многократно используемого кода. Например, представьте себе, что вам нужен метод, печатающий отчеты, в которых содержатся имена всех клиентов. Нет никакого смысла в написании отдельного метода для каждого клиента, особенно если список клиентов постоянно меняется. В листингах 3.15 и 3.16 приведен код метода, который принимает список клиентов и печатает отчет, содержащий их имена.

Листинг 3.15. Объявление метода, принимающего параметры (для C#)

```
// Program.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram
{
        class Program
        {
                 static void Main(string[] args)

                {
                     MessagePrinter msgPrint = new MessagePrinter();

                     string[] customerNames = { "Jones", "Smith", "Mayo" };
                     string reportTitle = "Important Customer Report";
                     msgPrint.PrintCustomerReport(customerNames, reportTitle);
                }
        }

// C# (MessagePrinter.cs):

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram
{
        public void PrintCustomerReport(
                   string[] customers, string title = "Customer Report")
        {
```

```
                Console.WriteLine(title);
                Console.WriteLine();
                foreach (var name in customers)
                {
                        Console.WriteLine(name);
                }
        }
    }
}
```

Листинг 3.16. Объявление метода, принимающего параметры (для VB)

```
' VB (Module1.vb):

Module Module1
    Sub Main()
        Dim msgPrint As New MessagePrinter()

        Dim customerNames = {"Jones", "Smith", "Mayo"}
        Dim reportTitle As String = "Important Customer Report"

        msgPrint.PrintCustomerReport(customerNames, reportTitle)
    End Sub
End Module

' VB (MessagePrinter.vb):

Public Class MessagePrinter
    Sub PrintCustomerReport(ByVal customers As String(), ByVal title
As String)
        Console.WriteLine(title)
        Console.WriteLine()
        For Each name In customers
            Console.WriteLine(name)
        Next
    End Sub
End Class
```

Параметры передаются методу в виде списка идентификаторов, где в качестве разделителя используется запятая и указывается тип каждого параметра. Эти квалификаторы явно указывают тип параметра ожидаемым методом. В листингах 3.15 и 3.16 метод `PrintCustomerReport` принимает два параметра: заголовок типа `string` `customers` типа `string array`. При запуске программы метод отображает заголовок в строке заголовка окна консоли, затем отображает пустую строку, после чего последовательно пробегает цикл, отображая в окне консоли имена всех клиентов.

Метод `Main` создает новый экземпляр `MessagePrinter`, на который указывает переменная `msgPrint`, а затем через `msgPrint` вызывает `PrintCustomerReport`. Передаваемые аргументы, `reportTitle` и `customerNames`, имеют позиции и типы, совпадающие с позициями и типами параметров `PrintCustomerReport`, указанными при объявлении метода `PrintCustomerReport`.

В приведенном примере вызывающий код должен предоставлять все данные, фактически передаваемые в качестве параметров. Однако указывать параметры не обязательно, и вы можете пропускать те из них, которые не являются обязательными. Рассмотрим модификацию метода `PrintCustomerReport`, в которой заголовок является необязательным параметром (листинги 3.17 и 3.18).

Листинг 3.17. Модификация метода `PrintCustomerReport`, в которой заголовок является необязательным параметром (код на C#)

```
public void PrintCustomerReport(
        string[] customers, string title = "Customer Report")
{
        Console.WriteLine(title);
        Console.WriteLine();

        foreach (var name in customers)
       {
             Console.WriteLine(name);
       }
}
```

Листинг 3.18. Модификация метода `PrintCustomerReport`, в которой заголовок является необязательным параметром (код на VB)

```
Sub PrintCustomerReport(
       ByVal customers As String(),
       Optional ByVal title As String = "Customer Report")

       Console.WriteLine(title)
       Console.WriteLine()

       For Each name In customers
           Console.WriteLine(name)
       Next
End Sub
```

В только что приведенном коде при вызове необходимо передать массив, содержащий имена клиентов, но заголовок передавать не обязательно. При написании методов необязательные параметры должны указываться последними. Вызов метода с необязательным параметром осуществляется так, как показано в листингах 3.19 и 3.20.

Листинг 3.19. Вызов метода с необязательным параметром (код на C#)

```
custProg.PrintCustomerReport(customerNames);
```

Листинг 3.20. Вызов метода с необязательным параметром (код на VB)

```
msgPrint.PrintCustomerReport(customerNames)
```

Поскольку вызывающий код не указывает параметра для заголовка, будет использоваться значение, указанное в `PrintCustomerReport`. Это значение становится значением по умолчанию и присваивается параметру `title`.

Кроме передачи методам аргументов, можно получать от них возвращаемые значения.

Возвращение данных и использование значений, возвращаемых методами

Методы могут возвращать значения. Синтаксис возврата значений продемонстрирован в методе, приведенном в листингах 3.21 и 3.22. Этот метод принимает значение типа `int`, возводит его в квадрат и возвращает это значение. Вызывающий код затем присваивает значение, возвращенное методом, переменной, и отображает его в окне консоли. Создайте новый класс, назовите его Calc.cs или Calc.vb, и заполните его кодом из листингов 3.21 и 3.22, соответственно.

Листинг 3.21. Возвращение значений из методов в C#

```
// Program.cs:

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram
{
    class Program
        {
                static void Main(string[] args)
                {
                        Calc mathProg = new Calc();

                        int squaredInt = mathProg.SquareInt(3);
                        Console.WriteLine("3 squared is " + squaredInt);

                        Console.ReadKey();
```

```
            }
        }
}

// Calc.cs:

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram
{
        public class Calc
       {
              public int SquareInt(int number)
             {
                return number * number;
             }
       }
}
```

Листинг 3.22. Возвращение значений из методов в C#

```
' VB (Module1.vb):

Module Module1
    Sub Main()
        Dim mathProg As New Calc()
        Dim squaredInt As Integer = mathProg.SquareInt(3)
        Console.WriteLine("3 squared is " & squaredInt)
    End Sub
End Module

VB (Calc.vb):

Public Class Calc
    Public Function SquareInt(ByVal number As Integer) As Integer
        Return number * number
    End Function
End Class
```

Обратите внимание, что в коде на C# (листинг 3.20) метод `SquareInt` объявлен как `int`, а не с использованием ключевого слова `void`, как это делалось ранее для методов, не возвращавших значений. Если при объявлении метода вы указываете тип данных, то это значит, что данный метод должен возвращать некоторое значение,

тип которого соответствует объявленному. В только что приведенном примере метод был объявлен как `int`, следовательно, данный метод гарантирует, что результат вычисления будет иметь тип `int`. В методе `Main` имеются утверждения, которые вызывают метод `SquareInt`, выводят на консоль возвращаемые им результаты.

В примере на VB (листинг 3.21) метод объявляется как `Function`. Методы, объявленные как `Sub`, значений не возвращают. Обратите внимание на сигнатуру функции — она объявляется как целая. Об этом говорят ключевые слова `As Integer`, добавленные после списка параметров. Именно эта конструкция и указывает тип возвращаемого значения функции — `Integer`.

Автоматически генерируемые фрагменты кода методов

Язык C# не предлагает автоматической генерации фрагментов кода для методов (хотя вы при желании можете создать собственные). Зато в VB эта возможность имеется. Чтобы воспользоваться ею, введите с клавиатуры строку **`Sub`**, а затем дважды нажмите клавишу <TAB>. В результате на экране появится шаблон, показанный на рис. 3.2. Если с клавиатуры ввести строку **Fun**, а затем дважды нажать клавишу <TAB>, то на экране появится шаблон, показанный на рис. 3.3.

```
Sub MySub()

    End Sub
```

Рис. 3.2. Шаблон для автоматической генерации кода процедуры (`sub`) в VB

```
Function MyFunc() As Integer

    Return 0
End Function
```

Рис. 3.3. Шаблон для автоматической генерации кода функции (`function`) в VB

Кодирование полей и свойств

Поле представляет собой переменную, которая является членом класса (типа), в противоположность переменным, которые объявляются внутри классов и представляют собой локальные переменные (local variables) или переменные с локальным диапазоном действия (locally scoped variables).

Свойства (Properties) представляют собой члены типов, которые предоставляют вам функциональные возможности, промежуточные между полями и методами. Свойства допускают чтение и запись, как и поля. Дополнительно можно определить код, который выполняется каждый раз, когда вы читаете или записываете свойство, по аналогии с методами. Поля и свойства будут подробно рассмотрены в последующих нескольких разделах.

Объявление и использование полей

Как уже говорилось, поле представляет собой переменную, которая является членом класса, или контейнера другого типа, например, структуры (`struct`), который, в принципе, очень близок к классу. За счет этого вы получаете то преимуще-

ство, что поле и данные, которые в нем содержатся, оказываются доступны всем остальным членам класса (а также, в зависимости от модификатора доступа, через наследование, и другим производным классам). Пример, приведенный в листингах 3.23 и 3.24, показывает, как осуществляется такое объявление. Этот код моделирует банковский счет, который имеет поле типа `decimal` с именем `currentBalance`. Это поле, как и следует из его названия, содержит текущий баланс счета. Класс имеет два метода: `Credit` и `Debit`. Метод `Credit` увеличивает значение поля `currentBalance`, а метод `Debit` его уменьшает.

Листинг 3.23. Использование полей и свойств в коде на C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram
{
        class Program
        {
                private decimal accountBalance = 100m;

                static void Main(string[] args)
                {
                        Program account = new Program();
                        account.Credit(100m);
                        account.Debit(50m);
                        Console.WriteLine("Balance: " + account.CurrentBalance);

                        Console.ReadKey();
                }

                public void Credit(decimal amount)
                {
                        accountBalance += amount;
                }
                public void Debit(decimal amount)
                {
                        accountBalance -= amount;
                }
                public decimal CurrentBalance
                {
                        get
                        {
                                return accountBalance;
```

```
                    }
                    set
                    {
                            if (value < 0)
                            {
                                // списание денег со счета
                            }
                            accountBalance = value;
                     }
                }
        }
}
```

Листинг 3.24. Использование полей и свойств в коде на VB

```
Module Module1
Private Dim accountBalance As Decimal = 100
        Sub Main() Credit(100)
            Debit(50)
            Console.WriteLine("Balance: " & CurrentBalance)

            Console.ReadKey()
        End Sub

        Sub Credit(ByVal amount As Decimal)
            accountBalance += amount
        End Sub

        Sub Debit(ByVal amount As Decimal)
            accountBalance -= amount
        End Sub

        Public Property CurrentBalance() As Decimal
               Get
                      Return accountBalance
               End Get
               Set(ByVal value As Decimal)

                   If value < 0 Then
                      ' списание денег со счета
                   End If

                   accountBalance = value
               End Set
        End Property
End Module
```

Посмотрите, где объявляется поле `accountBalance`: в начале класса `Program` (`Module1` — в VB). Это объявление находится в том же диапазоне, что и `Main` и остальные методы. Это значит, что поле является членом класса `Program` (`Module1` — в VB), как и `Main`, `Credit` и `Debit`. Когда переменные наподобие `accountBalance` объявляются как члены класса (в отличие от локальных переменных, объявляемых внутри блоков метода), они называются полями. Поле `accountBalance` имеет тип `decimal`, который является удачным выбором для финансовых вычислений.

Поле `accountBalance` имеет модификатор `private`. Это значит, что оно может использоваться только членами того же класса, к которому оно принадлежит. Реализации методов `Credit` и `Debit`, соответственно, увеличивают и уменьшают значение поля `accountBalance`. Поскольку методы `Credit` и `Debit` являются членами того же самого класса, что и `accountBalance`, они могут считывать значения `accountBalance` и изменять их.

Чтобы изменить значение поля `accountBalance`, `Main` вызывает методы `Credit` и `Debit`. Кроме того, `Main` отображает значение `accountBalance` в окне консоли, используя свойство `CurrentBalance`. В следующем разделе мы разберем, как работает свойство `CurrentBalance`.

Объявление и использование свойств

Свойства (Properties) являются членами класса, которые используются так же, как и поля, с той лишь разницей, что при чтении или записи свойства можно добавлять специализированную логику. В листингах 3.23 и 3.24 есть пример свойства, `CurrentBalance`. Для удобства чтения соответствующие фрагменты кода приведены в листингах 3.25 и 3.26.

Листинг 3.25. Пример свойства CurrentBalance в коде на C# (фрагмент листинга 3.23)

```
public decimal CurrentBalance
{
        get
        {
                return accountBalance;
        }

        set
        {
                if (value < 0)
                {
                    // списание денег со счета
                }
                accountBalance = value;
        }
}
```

Листинг 3.26. Пример свойства CurrentBalance в коде на VB (фрагмент листинга 3.24)

```
Public Property CurrentBalance() As Decimal
    Get
            Return accountBalance
    End Get
    Set(ByVal value As Decimal)

           If value < 0 Then
               ' списание денег со счета
           End If

           accountBalance = value
    End Set
End Property
```

Свойства имеют инструменты доступа (accessors), которые называются `get` и `set`. Эти инструменты позволяют добавлять специализированную логику при использовании свойств. Когда вы производите чтение свойства, исполняется только инструмент `get`, а инструмент `set` исполняется только тогда, когда свойству присваивается значение.

В только что рассмотренном примере инструмент `get` возвращает значение свойства `currentBalance`, не производя никаких модификаций. В тех случаях, когда требуется реализовать некую логику, например, в дополнение к текущему балансу подсчитать проценты, инструмент `get` может содержать логику, позволяющую выполнить эти вычисления перед возвратом значения. Инструмент `set` обладает логикой, позволяющей проверить, не является ли значение меньшим нуля. Такая ситуация может возникнуть, если клиент допустил овердрафт (снял сумму, превышающую установленный лимит). Если результат отрицателен, то вы можете реализовать логику, в соответствии с которой с клиента взимается плата за овердрафт. Переменная `keyword` содержит значение, присвоенное свойству, и предшествующий оператор `set` присваивает это значение полю `accountBalance`. Следующее утверждение метода `Main` в листингах 3.27 и 3.28 читает значение `CurrentBalance`, фактически исполняя инструмент `get`, который возвращает значение `currentBalance`.

Листинг 3.27. Чтение значения CurrentBalance (код на C#)

```
Console.WriteLine("Balance: " + account.CurrentBalance);
```

Листинг 3.28. Чтение значения CurrentBalance (код на VB)

```
Console.WriteLine("Balance: " & CurrentBalance)
```

Так как свойство `CurrentBalance` возвращает значение поля `accountBalance`, утверждение `Console.WriteLine` выведет значение, считанное из `CurrentBalance`, в командную строку.

Многие свойства, которые вам придется писать, будут представлять собой просто "обертки" (wrappers) для текущего состояния объекта и не будут реализовывать никакой другой логики. Соответствующие примеры для C# и VB приведены в листингах 3.29 и 3.30.

Листинг 3.29. Свойство, представляющее собой просто "обертку" для текущего состояния объекта и не реализующее никакой другой логики (код на C#)

```
private string m_firstName;

public string FirstName
{
        get
        {
                return m_firstName;
        }
        set
        {
               m_firstName = value;
        }
}
```

Листинг 3.30. Свойство, представляющее собой просто "обертку" для текущего состояния объекта и не реализующее никакой другой логики (код на VB)

```
Private m_firstName As String
Public Property FirstName() As String
        Get
               Return m_firstName
        End Get
        Set(ByVal value As String)
            m_firstName = value
        End Set
End Property
```

В листингах 3.29 и 3.30 поле `m_firstName`, обычно называемое поддерживающим полем, представляет собой частную переменную (private variable), которую возвращает свойство `FirstName` через инструмент доступа `get`, или присваивает ей значение через инструмент доступа `set`. Вы можете сэкономить на этом общем синтаксисе, воспользовавшись автоматическим свойством, как показано в листингах 3.31 и 3.32.

Листинг 3.31. Автоматически реализуемые свойства (код на C#)

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

Листинг 3.32. Автоматически реализуемые свойства (код на VB)

```
Public Property FirstName As String
```

Автоматическое свойство `FirstName` является логическим эквивалентом расширенного `FirstName` с инструментами доступа и поддерживающим полем. Компилятор сам создаст развернутую версию, в которой поддерживающее поле гарантированно будет иметь уникальное имя, что позволит избежать конфликтов.

Имейте в виду, что когда вы используете автоматические свойства, вы не можете добавлять собственный код к инструментам доступа `get` и `set`.

Автоматически генерируемый фрагмент кода для свойства

Чтобы создать автоматически генерируемый фрагмент кода для свойства, введите с клавиатуры начальные символы, **pro** и два раза нажмите клавишу <TAB>. На экране появится шаблон кода свойства, показанный на рис. 3.4 (для C#) или рис. 3.5 (для VB). Шаблон кода свойства для C# по умолчанию создает автоматическое свойство. Для VB шаблон создаст полный код свойства с инструментами доступа `get` и `set`.

```
public int MyProperty { get; set; }
```

Рис. 3.4. Шаблон автоматически генерируемого кода для свойства на C#

```
Private newPropertyValue As String
Public Property NewProperty() As String
    Get
        Return newPropertyValue
    End Get
    Set(ByVal value As String)
        newPropertyValue = value
    End Set
End Property
```

Рис. 3.5. Шаблон автоматически генерируемого кода для свойства на VB

Заключение

В этой главе мы обсудили создание классов для определения собственных типов. После этого вы научились создавать и использовать экземпляры классов, также называемые объектами, а также добавлять к определениям класса поля, методы и свойства. Далее было дано углубленное описание методов и подробно рассказывалось о том, как следует определять параметры и возвращаемые значения. Далее была затронута тема определения свойств (определяемых как автоматически, так и индивидуально). Наконец, обсуждался и такой вопрос, как наследование классов.

В следующей главе будет продемонстрировано, как создавать другие типы, называемые интерфейсами (interface). Кроме того, вы узнаете о том, как добавлять еще один тип членов класса — события (events).

Глава 4

Необходимый минимум знаний о языках C# и VB.NET: среднеуровневый синтаксис

В данной главе будут рассмотрены следующие ключевые концепции программирования на C# и VB:

- Использование делегатов (Delegates) и событий (Events)
- Реализация интерфейсов (Interfaces)
- Кодирование с использованием массивов (Arrays) и общих (универсальных, или родовых) типов (Generics)

В предшествующих главах рассказывалось о базовых синтаксических правилах и демонстрировалось создание индивидуально определяемых типов. В этой главе мы продвинемся еще на шаг дальше. Вы узнаете о делегатах (delegates), событиях (events), интерфейсах (interfaces) и приобретете базовые навыки, необходимые для работы с массивами (arrays) и общими типами (generics). Хотя этот материал и не претендует на то, чтобы считаться предназначенным для "продвинутых" программистов, он рассчитан на то, чтобы дать читателю достаточный объем информации, нужной для того, чтобы понимать основные языковые концепции. В данной главе на практических примерах будут показаны все языковые возможности, которые будут встречаться в книге и далее, по ходу изложения. Сейчас — самое время познакомиться с ними поближе и понять их смысл и значение. Начнем наше обсуждение с делегатов и событий.

Разбираемся с делегатами и событиями

Бывают ситуации, когда программисту требуется написать гибкий код, реагирующий на выполнение конкретных операций. Например, когда разработчики .NET Framework создавали пользовательские интерфейсы, они добавили многократно используемые элементы управления, такие, как кнопки, раскрывающиеся списки и сетки таблиц (grids). При написании этих элементов разработчики не знали, как именно мы будем их использовать. И в самом деле — как можно узнать, что за код должен запускаться и что он должен делать, когда пользователь выбирает один из интерфейсных элементов — например, щелкает по кнопке мышью? Таким образом, элементы управления имеют точки взаимодействия, встроенные в их код

таким образом, чтобы они могли поддерживать связь с вашей программой. Эти точки взаимодействия называются событиями (events). Эти события срабатывают всякий раз, когда пользователь выполняет определенное действие — например, нажимает кнопку или выбирает элемент из списка. Программисты пишут код, который связывает эти события с другим кодом в разрабатываемой программе и должен выполняться при наступлении этих событий, например, таких, как нажатие кнопки. Это — именно та задача, для выполнения которой используются делегаты (delegates).

Событие определяет тип уведомлений, которые могут предоставляться объектом, и делегат позволяет связать событие с тем кодом, который должен исполняться при его наступлении.

В этом разделе будет продемонстрирована вся "механика" взаимодействия делегатов и событий. Имейте в виду, что эта "механика" на самом деле очень проста, хотя, на первый взгляд, она и может показаться чем-то абстрактным. Делегаты и события чаще всего используются при работе с технологиями .NET Framework, которые их используют. Например, к их числу относятся Windows Presentation Foundation (WPF), Silverlight и ASP.NET.

Лучше всего познакомиться с этими технологиями уже сейчас, хотя бы бегло. На протяжении последующих глав мы вернемся к более подробному обсуждению делегатов и событий.

В следующем разделе мы добавим дополнительную логику в реализацию инструмента доступа `set` в `CurrentBalance` (см. *главу 3*) и возбудим событие для вызывающего кода.

События

Событие — это тип элемента класса, который позволяет вашему классу или экземпляру класса уведомлять другой код о том, что происходит в пределах этого класса. Чтобы помочь вам понять, как используются события, в данном разделе мы ассоциируем событие с кодом `accountBalance` для расчетного счета. Код, представленный в листингах 4.1 и 4.2, представляет собой модифицированную версию кода, приведенного в листингах 3.23 и 3.24 из *главы 3*. Этот код, в дополнение к тому, что он делал в *главе 3*, теперь имеет событие и реализует логику, которая его возбуждает.

Чтобы понять, чем могут быть полезны события, рассмотрим программу, использующую класс, который управляет событиями. Существуют различные типы событий, например, таких, как проверка накоплений. Если клиент допускает овердрафт, последствия могут быть различными, в зависимости от используемого типа счета. Однако, все, что вам может потребоваться — это обобщенный класс `account`, который может использоваться для банковских счетов различного типа и не имеет информации о правилах овердрафта. Это делает класс универсальным и дает возможность его повторного и многократного использования по различным сценариям. Таким образом, для класса `account` можно определить событие, которое выдаст уведомление в том случае, если будет допущен овердрафт. Затем,

в пределах проверки для специализированного экземпляра класса `account`, например, вы можете зарегистрировать обработчик события (event handler) таким образом, что экземпляр класса будет получать уведомление через обработчик события каждый раз, когда клиент допускает овердрафт.

В листингах 4.1 и 4.2 свойство `CurrentBalance` модифицируется таким образом, чтобы возбуждать событие `OverDraft` каждый раз, когда баланс счета отрицателен (значение, присвоенное свойству `CurrentBalance`, меньше 0). Метод `Main` вызывает (hooks up) другой метод, который запускается каждый раз, когда происходит событие. Сначала давайте обсудим события, а затем перейдем к рассмотрению того, как вызвать метод, ожидающий наступления события и получающий сообщение, отправляемое этим событием.

Листинг 4.1. Демонстрационный пример события (код на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram
{
        class Program
        {
                private decimal accountBalance = 100m;
                static void Main(string[] args)

                {
                        Program account = new Program();
                        account.OverDraft += new EventHandler(account_OverDraft);
                        account.CurrentBalance = -1;

                        Console.ReadKey();
                }

                public decimal CurrentBalance
                {
                        get
                        {
                                 return accountBalance;
                        }
                        set
                        {
                                if (value < 0)
                                {
                                        if (OverDraft != null)
```

```
                                        {
                                              OverDraft(this, EventArgs.Empty);
                                        }
                                  }
                                  ccountBalance = value;
                            }
                      }


                       static void account_OverDraft(object sender, EventArgs e)
                      {
                              Console.WriteLine("Overdraft Occurred");
                      }

                      public event EventHandler OverDraft;
              }
}
```

Листинг 4.2. Демонстрационный пример события (код на VB)

```
Module Module1
    Private Dim accountBalance As Decimal = 100

    Sub Main()
        AddHandler OverDraft, AddressOf AccountOverdraft
        CurrentBalance = -1

        Console.ReadKey()
    End Sub

    Public Event OverDraft As EventHandler

    Public Sub AccountOverdraft (ByVal sender As Object, ByVal e As
EventArgs)
           Console.WriteLine("Overdraft Occurred")
    End Sub
End Module
```

В листингах 4.1 и 4.2 определено событие, которое называется `OverDraft`. Событие `OverDraft` является открытым (public) и определяется с помощью ключевого слова `event`. `EventHandler` — это делегат, который мы скоро обсудим. В принципе, он позволяет определить тип метода, который может вызываться событием. Делегат определяет род коммуникационного соглашения, которого должен придерживаться любой код, который дожидается срабатывания события.

Рассмотрим чуть внимательнее инструмент доступа для свойства `CurrentBalance`. Внутри утверждения `if` определяется утверждение, которое проверяет, не является

ли значение `CurrentBalance` отрицательным. В примере на C# пример использует другое выражение `if`, которое проверяет, не является ли пустым (`null`) значение события `OverDraft`.

В языке C#, когда событие равно `null`, это означает, что никакой код не "подписан" на получение уведомления, то есть что никакой другой код не ожидает его появления. Однако когда событие в C# не является равным `null`, это говорит о том, что какой-то другой метод где-то "подписался" на этот метод и ожидает быть вызванным, когда событие сработает. Как правило, в таких случаях говорят, что метод "прослушивает" событие. Таким образом, предполагая, что вызывающий код связан с методом, событие `OverDraft` сработает. Эта проверка равенства события `null` имеет очень важное значение. Если никакой код не выполняет "прослушивание" события (а нашему коду известно, что этот случай соответствует ситуации, когда событие имеет значение `null`), и мы возбуждаем событие, вызывая `OverDraft (this, EventArgs.Empty)`, то во время исполнения произойдет ошибка (срабатывание исключения ссылки на `null`). Это будет случаться всякий раз, когда будет устанавливаться значение свойства `CurrentBalance`. Аргументы события C# означают, что текущий объект (который представляет собой экземпляр класса `Program`), указатель `this` и пустой массив `EventArgs` будут переданы в качестве сообщения от события любым другим методам, которые "подписались" на данное событие. Интересно заметить, что многие методы могут быть подписаны на ваше событие (или ни одного — такие ситуации тоже бывают, и каждый из таких "подписавшихся" методов будет уведомлен о событии при его срабатывании). К этому моменту вам должно стать уже понятно, что события на самом деле представляют собой просто спонтанную форму коммуникаций в пределах вашей программы.

В VB вам нет необходимости выполнять проверку `Nothing` (в C# эквивалентом `Nothing` является `null`).

Только что приведенное обсуждение показало, что метод, "подписавшийся" на событие, выполняется всякий раз при срабатывании этого события (получение сообщения). Следующая часть этого раздела объясняет, как используется делегат (delegate), чтобы указать, что данный метод собой представляет.

Делегаты

Делегаты позволяют вам "подписывать" методы на получение извещений от конкретных событий. Делегат указывает допустимую сигнатуру, количество аргументов, их типы для метода, который будет "подписан" на это событие в качестве "подписчика" (listener) или обработчика (handler). Делегат `EventHandler` для события `OverDraft` указывает событию, какой будет сигнатура метода. Это делается так, как показано в листингах 4.3 и 4.4 (для C# и VB, соответственно).

Листинг 4.3. Определение делегата (код на C#)

```
public event EventHandler OverDraft;
```

Листинг 4.4. Определение делегата (код на VB)

```
Public Event OverDraft As EventHandler
```

`EventHandler` представляет собой класс, который принадлежит к библиотеке классов .NET Framework, а эта библиотека классов, по умолчанию, указывает, что любые методы "подписанные" на событие `OverDraft`, должны определять два параметра: объект любого типа и класс `EventArgs`. `EventHandler` также указывает, что метод не возвращает значения явно. Следующий метод, `account_OverDraft` (`AccountOverdraft` — в VB), проверяет соответствие предопределенной сигнатуры `EventHandler`. Код этого метода приведен в листингах 4.5 и 4.6.

Листинг 4.5. Код метода account_OverDraft (код на C#)

```
static void account_OverDraft(object sender, EventArgs e)
{
        Console.WriteLine("Overdraft Occurred");
}
```

Листинг 4.6. Код метода `AccountOverdraft` (код на VB)

```
Public Sub AccountOverdraft(ByVal sender As Object, ByVal e As
EventArgs)
        Console.WriteLine("Overdraft Occurred")
End Sub
```

Обратите внимание, что в C# метод `account_OverDraft` (`AccountOverdraft` — в VB) не возвращает значения. У этого метода есть два параметра, которые имеют типы `object` и `EventArgs`, соответственно. Метод `account_OverDraft` (`AccountOverdraft` — в VB) "подписан" на событие `OverDraft` в методе `Main` листингов 4.1 и 4.2. В листингах 4.7 и 4.8 соответствующий фрагмент для вашего удобства будет повторен.

Листинг 4.7. Метод account_OverDraft "подписан" на событие OverDraft в методе Main (код на C#)

```
account.OverDraft += new EventHandler(account_OverDraft);
account.CurrentBalance = -1;
```

Листинг 4.8. Метод AccountOverdraft — в VB) "подписан" на событие OverDraft в методе Main

```
AddHandler OverDraft, AddressOf AccountOverdraft
CurrentBalance = -1
```

В примере на языке C#, синтаксис `+=` используется для назначения делегата событию (программисты часто говорят "привязять событие" — "wire up an event").

В примере на VB для той же цели используются `AddHandler` и `AddressOf` — таким образом, метод `AccountOverDraft` привязывается к событию `OverDraft`. В примере на C# делегат представляет собой новый экземпляр `EventHandler`, а событием является `OverDraft`. Как вы помните, типом делегата `OverDraft` является `EventHandler`, и этот тип точно определяет соглашения о сообщении.

Следующий фрагмент "головоломки" — метод, который должен быть уведомлен при наступлении события. Этот метод как параметр передается новому экземпляру делегата `EventHandler`. Ранее вы уже видели, где метод `account_OverDraft` (`AccountOverDraft` — в VB) получал сигнатуру, указанную классом `EventHandler`, давая возможность указать наш метод в качестве нового параметра `EventHandler`. Благодаря этой единственной строке кода (той самой, которая содержала утверждение `+=`), метод `account_OverDraft` (`AccountOverdraft` — в VB) связывался с событием `OverDraft`. Это означает, что когда для переменной `CurrentBalance` инструмент доступа `set` устанавливает отрицательное значение, срабатывает событие `OverDraft`, потому что осуществляется вызов `OverDraft(this, EventArgs.Empty)`, который инициирует метод `account_OverDraft` (`AccountOverdraft` — в VB), который мы привязали к событию. Этот метод, в свою очередь, исполняет свой код.

Еще одно замечание о событиях: код, реализующий графический интерфейс пользователя (GUI), пользуется ими активно. Рассмотрим, например, код GUI, в котором имеются многократно используемые элементы, например, кнопки и окна списков. Каждый раз, когда пользователь нажимает кнопку или выбирает один из элементов списка, ваш код должен выполнять ту или иную операцию, например, такую, как сохранение данных.

Все это осуществляется на базе событий: например, для кнопки это будет событие `Click`, а для поля раскрывающегося списка — `SelectedItemChanged`. Это — стандартный способ программирования графических пользовательских интерфейсов. Как правило, вы имеете некое событие, определяете метод, который связан с этим событием, так что в процессе работы вашей программы она будет выполнять определенную работу, реагируя на действия пользователя.

Завершение кода делегатов и обработчиков

Хотя данная функция, как таковая, и не представляет собой то же самое, что и автоматическая генерация фрагментов, но для создания событий и делегатов в C# имеется поддержка технологии Intellisense Code Completion, которая помогает связыванию событий и делегатов, а также генерирует метод обработчика. Процесс является двухступенчатым: создание делегата и создание обработчика. Для начала следует ввести ссылку на экземпляр, содержащий событие, имя события и утверждение `+=`. Как только вы введете с клавиатуры знак равенства (=), вы увидите всплывающую подсказку, выглядящую примерно так, как показано на рис. 4.1.

Как видите, редактор кода выводит всплывающую инструкцию, предлагающую нажать клавишу <TAB>, чтобы создать новый экземпляр делегата. Нажмите клавишу <TAB>, и функция Code Completion выведет еще одну всплывающую подсказку, в которой говорится о том, как создать метод обработчика, как показано на рис. 4.2.

Рис. 4.1. Автоматическое завершение кода для присвоения делегата

Рис. 4.2. Окно подсказки Code Completion, предлагающее вам имя метода

Вы можете просто нажать клавишу <TAB>, чтобы согласиться с именем, предложенным по умолчанию, или же изменить предложенное имя, и уже только затем нажать клавишу <TAB>. В любом случае в вашем распоряжении оказывается очень быстрый способ связывания метода обработчика и события посредством типа делегата.

Точно так же, как делегат предоставляет интерфейс для метода, в сущности, представляющий собой соглашение о коммуникации, вы можете определить и интерфейсы к классам, тоже описывающие способ коммуникаций с этими классами. Называются эти соглашения о коммуникациях достаточно интуитивно — интерфейсами.

Реализация интерфейсов

Интерфейсы представляют собой еще одну языковую функцию, предоставляющую программистам необходимую гибкость. Интерфейс может быть полезен, если вам нужна группа классов, которые можно в любое время менять между собой, но при этом вам требуется написать те же самые операторы для каждого из классов. В принципе, вам нужно написать код, который использует класс только один раз, но при этом может переключаться на фактический класс. Именно здесь и окажутся удобными интерфейсы. Интерфейс создает связь-соглашение (contract), которому должны соответствовать все взаимозаменяемые классы. Так, если интерфейс заявляет, что класс, реализующий его, имеет метод `A` и свойство `B`, тогда каждый класс, реализующий этот интерфейс, должен иметь метод `A` и свойство `B`. Компилятор устанавливает это соглашение принудительно, и оно не может быть нарушено. В последующих разделах будет показано, как написать интерфейс, а затем построить несколько классов, реализующих данный интерфейс. Наконец, будет написан и код, взаимодействующий с классами через этот интерфейс.

Работая с интерфейсами, важно иметь в виду, что они не содержат никакого кода, кроме определений членов. Определение членов — это и есть соглашение, устанавливаемое интерфейсом.

Вы же должны построить класс, который содержит члены, определяемые интерфейсом, и написать код, предоставляющий реализацию членов интерфейсов. Общее место, создающее больше всего путаницы — это то, что интерфейс не содержит никакого исполняемого кода, в отличие от классов, которые реализуют этот интерфейс.

В последующих нескольких разделах будет показано, как создать интерфейс, как написать класс, который содержит код, написанный вами и реализующий соглашение, описанное интерфейсом, а также — как написать код, оперирующий классами, которые реализуют интерфейс (гарантируя выполнение соглашения).

Создание интерфейса

Чтобы создать интерфейс, щелкните правой кнопкой мыши по имени проекта в окне **Solution Explorer**, выберите из контекстного меню команды **Add | New Item**. Затем выберите опцию **Code** в языковой ветви в группе **Installed Templates** и выберите опцию **Interface**. Присвойте интерфейсу имя (в нашем примере это будет имя **IAccount**) и нажмите кнопку **Add**. В соответствии со стандартным соглашением, классы интерфейсов всегда получают имена, начинающиеся с заглавной буквы **I**. Вы увидите, что в ваш проект добавлен интерфейс, исходный текст которого показан в листингах 4.9 и 4.10.

Листинг 4.9. Пример интерфейса на C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram
{
        public interface IAccount
        {
                void Credit(decimal amount);
                void Debit(decimal amount);
                decimal CurrentBalance { get; set; }
        }
}
```

Листинг 4.10. Пример интерфейса на VB

```
Public Interface IAccount
        Sub Credit(ByVal amount As Decimal)
        Sub Debit(ByVal amount As Decimal)
        Property CurrentBalance As Decimal
End Interface
```

Добавив интерфейс, вы должны будете внести в него изменения, так, чтобы код соответствовал тому, что вы видите в листингах 4.9 и 4.10. Обратите внимание, что члены `IAccount` не имеют реализации и производят впечатление незаконченных, потому что они не содержат ни одной строчки кода. Кроме того, ни один из членов не имеет модификатора `public`, потому что неявно подразумевается, что все члены интерфейса имеют тип `public`. В последующих нескольких разделах будет показано, как следует строить классы, реализующие интерфейс `IAccount`. Начиная с этого момента, вы сможете на практике прочувствовать преимущества, предоставляемые интерфейсами.

Написание классов, реализующих интерфейсы

Чтобы создать класс, щелкните правой кнопкой мыши по имени проекта в окне **Solution Explorer**, выберите из контекстного меню команды **Add | New Item**, затем выберите опцию **Code** в языковой ветви **Installed Templates**, а затем выберите опцию **Class**. Дайте имя новому классу (в нашем примере это будет имя **Checking**) и нажмите кнопку **Add**. Действуя точно так же, создайте еще один класс и назовите его **Saving**. Новые классы приведены в листингах 4.11—4.14.

Листинг 4.11. Класс Checking, реализующий интерфейс IAccount (код на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram
{
        class Checking : IAccount
        {
               public void Credit(decimal amount)
               {
                     // реализация логики проверки
                     CurrentBalance += amount;
                     Console.Writeline("Added " + amount.ToString() +
                                       " to Checking Account");
               }
               public void Debit(decimal amount)
              {
                     // реализация логики проверки
                     CurrentBalance -= amount;
                     Console.Writeline("Debited " + amount.ToString() +
                                        " from Checking Account");
              }
              public decimal CurrentBalance { get; set; }
        }
}
```

Листинг 4.12. Класс Checking, реализующий интерфейс IAccount (код на VB)

```
Public Class Checking
       Implements IAccount

       Public Sub Credit(ByVal amount As Decimal) Implements IAccount.
Credit



       ' Реализация кода проверки
       CurrentBalance += amount
                Console.Writeline("Added " & amount.ToString() &
                                  " to Checking Account")
       End Sub

       Public Sub Debit(ByVal amount As Decimal) Implements IAccount.Debit
       ' Реализация кода проверки
       CurrentBalance -= amount
               Console.Writeline("Debited " + amount.ToString() +
                                 " from Checking Account")
       End Sub

       Public Property CurrentBalance As Decimal Implements IAccount.
CurrentBalance
End Class
```

Листинг 4.13. Класс Saving, реализующий интерфейс IAccount (код на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FirstProgram
{
        class Saving : IAccount
        {
                public void Credit(decimal amount)
                {
                        // реализация логики работы с накоплениями
                        CurrentBalance += amount;
                        Console.Writeline("Added " + amount.ToString() +
                                          " to Saving Account");
                }

                public void Debit(decimal amount)
```

```
        {
                // реализация логики работы с накоплениями
                CurrentBalance -= amount;
                Console.Writeline("Debited " + amount.ToString() +
                                " from Saving Account");
        }

        public decimal CurrentBalance { get; set; }
    }
}
```

Листинг 4.14. Класс Saving, реализующий интерфейс IAccount (код на VB)

```
Public Class Saving
Implements IAccount
Public Sub Credit(ByVal amount As Decimal) Implements IAccount.
Credit
' реализация логики работы с накоплениями
CurrentBalance += amount
Console.Writeline("Added " & amount.ToString() &
" to Saving Account")
End Sub
Public Sub Debit(ByVal amount As Decimal) Implements IAccount.Debit
' реализация логики работы с накоплениями
CurrentBalance -= amount
Console.Writeline("Debited " + amount.ToString() +
" from Saving Account")
End Sub
Public Property CurrentBalance As Decimal
Implements IAccount.CurrentBalance
End Class
```

Обратите внимание, что в обоих листингах, как в классе `Checking` (листинги 4.11 и 4.12), так и в классе `Saving` (листинги 4.13 и 4.14) интерфейс `IAccount` реализован следующим образом:

- В C#: `class Checking : IAccount` и `class Saving : IAccount`
- В VB:

```
Public Class Checking
        Implements IAccount
```

и

```
Public Class Saving
        Implements IAccount
```

В коде на C# за именем класса следует символ двоеточия, а за ним — имя интерфейса. Это означает, что класс должен реализовать указанный интерфейс. В коде

на VB для той же цели используется ключевое слово `Implements`. Таким образом, обозначается, что оба класса, как `Checking`, так и `Saving`, реализуют интерфейс `IAccount`.

Внимательно изучив код классов `Checking` и `Saving`, вы обнаружите, что в их состав входят члены `Credit`, `Debit`, и `CurrentBalance`, которые указаны в интерфейсе `IAccount`. Однако принципиальная разница заключается в том, что интерфейс `IAccount` не содержит их реализации, но для классов `Checking` и `Saving` эту реализацию написали вы. Листинги 4.11—4.14 содержат упрощенную реализацию интерфейса, потому что их цель — всего лишь проиллюстрировать, как класс реализует интерфейс, не заставляя читателя разбираться с большим количеством кода. На практике, код методов классов `Checking` и `Saving` должен быть различным, потому что это — разные типы счетов, и для каждого из них действуют свои бизнес-правила.

Итак, вы уже создали интерфейс и написали классы, реализующие этот интерфейс. В следующем разделе будет приведено несколько примеров, призванных проиллюстрировать практическое использование интерфейсов.

Написание кода, использующего интерфейсы

Наилучшим способом оценить значимость интерфейсов является рассмотрение конкретных примеров, демонстрирующих, как интерфейсы помогают решать поставленные перед программистом задачи. В этом разделе будет продемонстрировано, как пишется код, получающий индивидуальный доступ к классам `Checking` и `Saving`, в особенности это актуально для повторяющегося кода. Затем я продемонстрирую написание такого кода за один прием. Код конкретного примера составляет платежную ведомость, при этом индивидуально получая доступ к классам `Checking` и `Saving` (фактически, это выполняется по аналогии с составлением ведомости на зарплату сотрудникам предприятия). Начнем с плохого примера (листинги 4.15 и 4.16), единственная цель которого состоит в том, чтобы продемонстрировать, как все это должно работать.

Листинг 4.15. Обработка платежной ведомости с явными экземплярами классов Checking и Saving (код на C#)

```
public void ProcessPayrollForCheckingAndSavingAccounts()
{
        Checking[] checkAccounts = GetCheckingAccounts();
        foreach (var checkAcct in checkAccounts)
        {
                checkAcct.Credit(500);
        }

        Saving[] savingAccounts = GetSavingAccounts();
        foreach (var savingAcct in savingAccounts)
        {
```

```
                savingAcct.Credit(500);
        }
}
public Checking[] GetCheckingAccounts()
{
         Checking[] chkAccts = new Checking[2];

         chkAccts[0] = new Checking();
         chkAccts[1] = new Checking();
         return chkAccts;
}
public Saving[] GetSavingAccounts()
{
        int numberOfAccounts = 5;
        Saving[] savAccts = new Saving[numberOfAccounts];
        for (int i = 0; i < numberOfAccounts; i++)
        {
                savAccts[i] = new Saving();
        }
return savAccts;
}
```

Листинг 4.16. Обработка платежной ведомости с явными экземплярами классов Checking и Saving (код на VB)

```
Sub ProcessPayrollForCheckingAndSavingAccounts()
        Dim checkAccounts As Checking() = GetCheckingAccounts()
        For Each checkAcct In checkAccounts
            checkAcct.Credit(500)
        Next

        Dim savingAccounts As Saving() = GetSavingsAccounts()

        For Each savingAcct In savingAccounts
            savingAcct.Credit(500)
        Next
End Sub

Function GetCheckingAccounts() As Checking()
         Dim chkAccts(1) As Checking

         chkAccts(0) = New Checking()
         chkAccts(1) = New Checking()

         Return chkAccts
```

```
End Function

Function GetSavingsAccounts() As Saving()
        Dim numberOfAccounts As Integer = 5
        Dim savAccts(numberOfAccounts) As Saving

        For i As Integer = 0 To numberOfAccounts
                savAccts(i) = New Saving()
        Next

        Return savAccts
End Function
```

В целях экономии пространства, в листингах 4.15 и 4.16 приведен сокращенный вариант, в котором содержатся только фрагменты кода, принципиальные для понимания общей концепции. Полный код работающего приложения можно найти на сайие **http://www. mcgrw-hill.co.uk**. Чтобы понять, как все это работает, представьте себе, что вы написали для метода `Main` код, приведенный в листингах 4.17 и 4.18.

Листинг 4.17. Код обработки платежной ведомости на C#

```
Program bank = new Program();
bank.ProcessPayrollForCheckingAndSavingAccounts();
```

Листинг 4.18. Код обработки платежной ведомости на VB

```
ProcessPayrollForCheckingAndSavingAccounts()
```

Давайте сосредоточимся на методе `ProcessPayrollForCheckingAndSavingAccounts`. Как видите, этот алгоритм сначала вызывает `GetCheckingAccounts`, чтобы получить массив объектов `Checking`. Как вы помните, массив представляет собой список элементов указанного типа, и в нашем примере — это тип `Checking`. Затем алгоритм последовательно просматривает объекты `Checking` и инициирует метод `Credit`, чтобы добавить к каждому счету по 500. Некоторые сотрудники хотят получать оплату наличными (Checking), но некоторые хотят, чтобы средства зачислялись на их накопительные счета (Saving), или даже на какой-то другой счет. Поэтому алгоритм вызывает метод `GetSavingsAccounts`, чтобы получить список счетов, принадлежащих сотрудникам, которые хотят, чтобы выписанные им чеки были перечислены на накопительные счета. Вы не можете не заметить, что алгоритм в методе `GetSavingsAccounts` отличается от алгоритма в `GetCheckingAccounts`. Я сделал это умышленно, чтобы продемонстрировать различные варианты использования циклов. Тем не менее, это не влияет на вызывающий код, потому что эти алгоритмы инкапсулированы в свои методы. Аспект, на который здесь следует обратить внимание, заключается в том, что метод `GetCheckingAccounts` вернет только экземпляры класса `Checking`, а метод `GetSavingsAccounts`, соответст-

венно — только экземпляры класса `Saving`. Остальная часть алгоритма метода `ProcessPayrollForCheckingAndSavingAccounts` совпадает с обработкой для класса `Checking`.

Ваше внимание непременно должно привлечь дублирование кода в методе `ProcessPayrollForCheckingAndSavingAccounts`. Хотя методы `Credit` классов `Checking` и `Saving` должны иметь разные реализации, код, вызывающий `Credit`, может быть одним и тем же, чтобы исключить дублирование. В листингах 4.19 и 4.20 показано, как можно воспользоваться тем преимуществом, что и `Checking`, и `Saving` реализуют тот же самый интерфейс, `IAccount`. Посмотрите, как следует вызывать `Credit` для любого из типов, реализующих интерфейс `IAccount`, используя один и тот же алгоритм. Это позволит исключить дублирование кода, которое имело место в листингах 4.15 и 4.16.

Листинг 4.19. Обработка платежной ведомости с помощью интерфейса IAccount (код на C#)

```
public void ProcessPayrollForAllAccounts()
{
        IAccount[] accounts = GetAllAccounts();

        foreach (var account in accounts)
        {
                account.Credit(1000);
        }
}
public IAccount[] GetAllAccounts()
{
        IAccount[] allAccounts = new IAccount[4];

        allAccounts[0] = new Checking();
        allAccounts[1] = new Saving();
        allAccounts[2] = new Checking();
        allAccounts[3] = new Saving();

        return allAccounts;
}
```

Листинг 4.20. Обработка платежной ведомости с помощью интерфейса IAccount (код на VB)

```
Sub ProcessPayrollForAllAccounts()
       Dim accounts As IAccount() = GetAllAccounts()

       For Each account In accounts
              account.Credit(1000)
       Next
```

```
End Sub

Function GetAllAccounts() As IAccount()
        Dim allAccounts(3) As IAccount

        allAccounts(0) = New Checking()
        allAccounts(1) = New Saving()
        allAccounts(2) = New Checking()
        allAccounts(3) = New Saving()

        Return allAccounts
End Function
```

Код, приведенный в листингах 4.19 и 4.20, может вызываться из метода `Main` так, как показано в листингах 4.21 и 4.22.

Листинг 4.21. Вызов кода, приведенного в листинге 4.19, из метода Main (код на C#)

```
Program bank = new Program();
bank.ProcessPayrollForAllAccounts();
```

Листинг 4.22. Вызов кода, приведенного в листинге 4.20, из метода Main (код на VB)

```
ProcessPayrollForAllAccounts()
```

Изучая код, приведенный в листингах 4.19 и 4.20, вы увидите, что список счетов `accounts` представляет собой массив элементов типа `IAccount`. Хотя вы не можете создать экземпляр интерфейса как такового, вы можете назначить экземпляр класса, который реализует этот интерфейс, используя переменную, объявленную как имеющую тип `interface`. В этом случае метод `GetAllAccounts` вернет список всех объектов, которые реализуют интерфейс `IAccount`.

Рассмотрев внимательнее метод `GetAllAccounts`, вы обнаружите, как массив строится из объектов `Checking` и `Saving`. Поскольку и `Checking`, и `Saving` реализуют интерфейс `IAccount`, как вы видели в листингах 4.11—4.14, экземпляры как класса `Checking`, так и класса `Saving` могут непосредственно присваиваться элементам массива `IAccount`.

Вернемся к рассмотрению метода `ProcessPayrollForAllAccounts`. В этой реализации вы увидите, как цикл последовательно, выполняя одну итерацию за другой, обрабатывает каждый элемент массива `IAccount`, вызывая `Credit`. Причина того, что вы можете таким образом вызывать `Credit`, заключается в том, что `IAccount` определяет соглашение для метода `Credit`. Вызов `Credit` для каждого экземпляра действительно инициирует метод `Credit` во время исполнения экземпляра `Checking` или `Saving`. Код, который вы написали для `Checking.Credit` и `Saving.Credit`, будет исполняться по мере того, как вы непосредственно вызываете их (см. листинги 4.19 и 4.20). Кроме того, обратите внимание на то, что

теперь удалось исключить дублирование кода, поскольку для объектов `Checking` и `Saving` реализуется один и тот же алгоритм, а именно `IAccount.Credit()`.

Теперь после того, как было продемонстрировано, каким образом интерфейсы помогают обрабатывать объекты различного типа, точно так же, как если бы они были однотипными, и как это помогает упростить код, который необходимо написать для взаимодействия с этими объектами за счет исключения дублирования. Представьте себе, как обстояли бы дела, если бы вам было предложено добавить дополнительные типы банковских счетов, чтобы они работали с данным алгоритмом, не пользуясь при этом интерфейсами. В этом случае вам бы потребовалось писать отдельный код алгоритма для каждого типа счета. Теперь же вы можете создавать новые типы счетов, получая их на основе `IAccount`, и эти новые типы автоматически будут работать с одним и тем же алгоритмом.

Автоматически генерируемый фрагмент для интерфейсов

Прежде чем воспользоваться автоматической генерацией кода интерфейса, откройте новый файл. Для этого щелкните правой кнопкой мыши по имени вашего проекта в окне **Solution Explorer**, выберите команды **Add** | **New Item** | **Code File** и дайте имя новому файлу. В нашем примере, для кода на C# это будет имя IInvestment.cs, а для кода на VB — имя IInvestment.vb. Раскроется новый пустой файл, и вы можете начать с ним работать. Чтобы воспользоваться автоматической генерацией кода интерфейса, введите с клавиатуры начальные символы, **int**, и нажмите клавиши <TAB>, <TAB>. На экране появится окно шаблона интерфейса, похожее на окна, представленные на рис. 4.3 (C#) и рис. 4.4 (VB).

Поскольку подразумевается, что имена интерфейсов начинаются с заглавной буквы **I**, в шаблоне будет подсвечен идентификатор, следующий за данным префиксом.

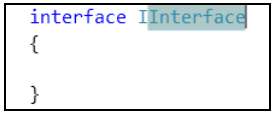


Рис. 4.3. Шаблон для автоматически генерируемого кода интерфейса на C#

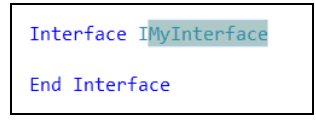


Рис. 4.4. Шаблон для автоматически генерируемого кода интерфейса на VB

Применение массивов (Arrays) и общих типов (Generics)

Какой бы код вы ни разрабатывали, вам, как правило, потребуется группировать объекты в коллекции, которые содержат объекты одного и того же типа. Для этой цели вы можете использовать массив, представляющий собой контейнер, который может быть пустым или состоять из некоторого количества элементов, каждый

из которых содержит экземпляр конкретного типа. Чуть далее в этой главе будет показано, как пользоваться массивом, чтобы найти нужные вам элементы. Кроме того, в .NET Framework имеются общие классы коллекций (generic collection classes), предоставляющие еще более мощные возможности, чем массивы. В этом разделе будет показано, как пользоваться массивами и общими (родовыми, generic) коллекциями.

Программирование с использованием массивов

Ранее в этой главе уже было приведено несколько примеров программирования с использованием массивов. Чтобы воспользоваться этой возможностью, вам требуется объявить переменную, как имеющую тип `array`, указать размерность, а затем начать пользоваться массивом, проиндексировав его элементы. В листингах 4.23 и 4.24 приведен пример, который демонстрирует создание и использование массива.

Листинг 4.23. Создание и использование массива (код на C#)

```
private void ArrayDemo()
{
        double[] stats = new double[3];

        stats[0] = 1.1;
        stats[1] = 2.2;
        stats[2] = 3.3;

        double sum = 0;

        for (int i = 0; i < stats.Length; i++)
        {
                sum += stats[i];
        }

        Console.WriteLine(
               stats[0] + " + " +
               stats[1] + " + " +
               stats[2] + " = " +
               sum);
}
```

Листинг 4.24. Создание и использование массива (код на VB)

```
Sub ArrayDemo()
        Dim stats(2) As Double

        stats(0) = 1.1
```

```
        stats(1) = 2.2
        stats(2) = 3.3

        Dim sum As Double = 0

        For i As Integer = 0 To 2
                sum += stats(i)
        Next

        Console.WriteLine(
               stats(0) & " + " &
               stats(1) & " + " &
               stats(2) & " = " &
               sum)
End Sub
```

В примере на C# (см. листинг 4.23), переменная `stats` объявлена как `double[]`, т. е. как массив элементов, имеющих тип `double`. Вам необходимо создать экземпляр массива, и это делается присвоением переменной `stats` значения `new double[3]`, где 3 — это количество элементов массива. В C# при доступе к массивам нумерация элементов массива начинается с 0, а это значит, что элементы массива имеют номера 0, 1 и 2, соответственно.

В примере на VB (см. листинг 4.24) переменная `stats` объявлена как массив элементов типа `double`. Обратите внимание на ранг массива (2), который указывает, что 2 — это последний индекс в массиве. Так как индексация элементов массива начинается с 0, переменная `stats` содержит индексы 0, 1 и 2. Таким образом, общее количество элементов массива равно 3.

Присваивание значений массиву означает, что необходимо указать имя массива и индекс элемента, которому должно быть присвоено значение. Например, запись формата `stats[0]` (в VB — `stats(0)`) означает, что значение присваивается первому элементу массива (иначе говоря, элементу с индексом 0). Из примера, приведенного в листинге, видно, что элементам массива `stats` присваиваются значения 1.1, 2.2 и 3.3, соответственно. На каждой итерации цикла `for` производится суммирование элементов массива, и результирующее значение этой суммы присваивается переменной `sum`. Наконец, в данном примере показано, как осуществляется считывание значений элементов массива путем разбора аргумента, переданного утверждению `Console.WriteLine`. В данном примере продемонстрировано, как прочитать значение конкретного элемента массива, используя синтаксис поэлементного доступа.

Массив представляет собой коллекцию однотипных элементов, имеющую фиксированный размер, вследствие чего массивы имеют несколько ограниченные функциональные возможности.

На практике, вероятно, вам потребуются более сложные коллекции объектов, например, такие, как класс `List`, который представляет собой родовую (generic) коллекцию. Не все классы коллекций в .NET Framework представляют собой родовые (generic) коллекции, но на сегодняшний день родовые коллекции являются предпочтительными для применения в большинстве классов.

Кодирование родовых коллекций (Generics)

Родовыми типами (Generics) называются языковые особенности, которые позволяют писать код, эффективно работающий со многими типами данных. Определение родового класса содержит поле-заполнитель для типа данных, с которым вы хотите оперировать, и его можно использовать для объявления типа, с которым вы собираетесь работать. В .NET есть целая библиотека родовых коллекций и родовых типов. Данный раздел призван послужить лишь кратким введением в родовые типы, поскольку рассмотреть их полностью в рамках одной главы не представляется возможным. Кроме того, будет приведен краткий пример использования родовых классов и типов, с которыми вы, вероятнее всего, встретитесь в будущем. В листингах 4.25 и 4.26 показано, как следует объявлять родовой тип `List` (список). Этот код указывает, что он работает со списком счетов типа `Checking`, а затем начинает заполнение списка элементами и выполняет операции над элементами типа `Checking`. Не забывайте включать в состав вашего кода директиву `using` (для VB — директиву `imports`) для пространства имен `System.Collections.Generic` при его объявлении ближе к началу вашего файла.

Листинг 4.25. Пример работы с родовой коллекцией типа "список" (код на C#)

```
private void ListDemo()
{
        List<Checking> checkAccts = new List<Checking>();

        checkAccts.Add(new Checking());
        checkAccts.Add(new Checking());

        for (int i = 0; i < checkAccts.Count; i++)
        {
                Console.WriteLine(checkAccts[i].CurrentBalance);
        }
}
```

Листинг 4.26. Пример работы с родовой коллекцией типа "список" (код на VB)

```
Sub ListDemo()
        Dim checkAccts As New List(Of Checking)

        checkAccts.Add(New Checking())
        checkAccts.Add(New Checking())

        For i As Integer = 0 To checkAccts.Count - 1
                Console.WriteLine(checkAccts(i).CurrentBalance)
        Next
End Sub
```

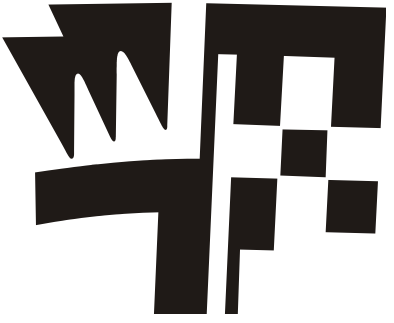
В .NET родовой тип `List` объявляется как `List<T>` (для C#) или `List(Of T)` — для VB. Здесь `T` — это "заполнитель", зарезервированный для указания типа, где вы можете указать любой тип, с которым вам требуется работать. Например, для списка целых чисел вы можете написать `List<int>`, а для списка строк — `List<string>`. Для кода на VB эти определения, соответственно, будут выглядеть так: `List(Of Integer)` и `List(Of String)`. В листинге 4.25, для кода на C#, можно видеть, что `checkAccts` объявляется как `List<Checking>`, а в листинге 4.26 (для кода на VB) это объявление выглядит так: `List(Of Checking)`. Так как список динамически растет, чтобы иметь возможность вмещать любое количество элементов, вы можете использовать метод `Add` для добавления в список новых элементов. Когда элементы будут добавлены в состав списка `List`, для доступа к ним можно использовать синтаксис доступа к элементам, как показано в приведенном цикле, где за один прием обрабатывается один элемент. Такие коллекции, как `List`, удобны, поскольку предоставляют множество облегчающих программирование методов, например, `Clear`, `Contains`, `Remove` и т. д.

В дополнение к `List`, пространство имен `System.Collections.Generic` имеет множество других родовых коллекций, например, `Dictionary`, `Queue` и `Stack`. Каждый родовой тип инициализируется путем замены параметров `type` на один из типов, с которыми вы собираетесь работать, После этого с ним можно будет работать, используя специализированные методы соответствующей коллекции. Каждый раз, когда вы видите параметрический синтаксис, вы можете догадаться об использовании родовых классов, и, таким образом, догадаться о предназначении этого кода, и понять, где искать нужную информацию в документации.

Заключение

В этой главе было рассказано о важнейших навыках, которые пригодятся вам для понимания материалов, изложенных в последующих главах. Владея информацией о делегатах и событиях, вы сможете быстро разобраться с тем, как работают управляемые событиями приложения с графическим пользовательским интерфейсом (GUI). Понимание принципов работы интерфейсов имеет непосредственное отношение к умению строить Web-сервисы, и, кроме того, интерфейсы имеют еще и другие, весьма обширные области применения. Наконец, далее вы начнете активно применять массивы и другие родовые типы, базовые знания о которых были получены вами в ходе чтения данной главы.

Имейте в виду, что изложенная здесь информация представляет собой только краткий вводный курс в программирование на C# и VB, и что вам придется проделать еще очень значительный объем самостоятельной работы по изучению обоих этих языков. В следующей главе мы будем обсуждать такую важную тему, как построение проектов VS.



ЧАСТЬ II

Изучаем среду разработки VS 2010

Глава 5

Создание и построение проектов

В данной главе будут рассмотрены следующие ключевые концепции:

- Работа с проектами и решениями.
- Установка свойств проекта в окне **Properties**.
- Справочная информация об использовании библиотек классов.
- Компиляция и запуск проектов.
- Использование редактора классов (Class Designer).

Проекты (projects) и решения (solutions) представляют собой способ VS, предназначенный в помощь программистам для организации кода в ходе разработки и развертывания приложений. В ходе разработки вы получаете гибкую иерархическую структуру, позволяющую организовать код таким образом, чтобы над ним было удобно работать как программистам-одиночкам, так и целым коллективам разработчиков. С точки зрения развертывания готовых приложений, вы можете строить проекты различных типов, в результате чего вы сможете получать как исполняемые файлы, так и библиотеки (часто называемые сборками), обеспечивающие возможность работы вашей программы во время ее исполнения.

В процессе чтения этой главы вы научитесь использованию проектов и решений. Вы узнаете, как следует находить настройки и опции для проектов и их индивидуальной настройки, как ссылаться на сборки и как использовать различные опции при компиляции кода. В качестве дополнительного бонуса, в этой главе будет рассказано о том, как конструктор классов (Class Designer) помогает добиться высокого уровня визуализации вашего кода и выполнять некоторые задачи по конструированию. Начнем с рассмотрения таких основополагающих понятий, как решения (solutions) и проекты (projects).

Конструирование решений и проектов

В VS можно строить приложения, которые варьируются в широком диапазоне как по размерам, так и по уровню сложности. На простейшем уровне вы можете создать консольный проект, состоящий из одного или нескольких файлов, содержащих программный код. На высших уровнях вы можете строить приложения, применяемые в масштабах крупных предприятий, и состоящие из множества проектов

различного типа, организованных таким образом, чтобы поддерживать работу больших коллективов, работающих параллельно.

VS использует иерархическую модель, которая помогает организовать ваш код и обеспечивает необходимую гибкость в управлении настройками проекта. Некоторые функции, например, такие, как решения и проекты, четко определены, но при этом предоставляют необходимую свободу, например, в добавлении файлов и папок, что помогает в индивидуальной настройке файловой структуры в соответствии с вашими потребностями.

В силе всегда остаются два принципа организации проектов и решений: в каждый конкретный момент времени вы работаете только с одним решением, и каждое решение может содержать в своем составе один или несколько проектов. Чтобы избежать избыточной сложности, я буду использовать термин "проект", но при этом вы должны иметь в виду, что в действительности речь идет о решении, состоящем из одного проекта, и вы имеете дело с проектом, который входит в состав решения. Проекты различных типов имеют уникальные настройки и опции, но мы начнем с создания консольного приложения. Это позволит нам сосредоточиться на основных свойствах проектов всех типов, избежав ненужных в данном случае частностей.

Создание новых проектов

Чтобы создать новый проект, нажмите клавиатурную комбинацию <CTRL>+ +<SHIFT>+<N>. Откроется окно **New Project**, показанное на рис. 5.1. Клавиатурная комбинация <CTRL>+<N> откроет один-единственный файл, который вы не сможете компилировать, так что не забудьте о клавише <SHIFT>. Естественно, что для создания нового проекта можно пользоваться и меню, но клавиатурные комбинации — это просто боле быстрый способ выполнения той же самой задачи.

Функции, доступные в окне **New Project**, были описаны в *главе 2*. Процесс установки опций в этом окне будет одинаков и должен выполняться каждый раз при создании нового проекта. VS запоминает тип последнего созданного вами проекта, что может быть полезным, когда вы создаете большое количество однотипных проектов. Для целей нашего примера выберите в качестве типа проекта опцию **Console Application**.

Способы, которыми вы создаете новые проекты на VB и C#, отличаются тем, что все решения, которые необходимо принять для проекта на C#, принимаются сразу же и одновременно, а для проектов на VB вы сначала создаете проект, а затем сохраняете дополнительную информацию о нем — это делается при первом сохранении проекта.

При создании проектов на C# можете редактировать такие поля, как **Name**, **Location**, **Solution** и **Solution Name** (см. рис. 5.1). В C# поле **Name** содержит имя создаваемого проекта, а поле **Solution Name** указывает имя решения, в состав которого входит данный проект.

При вводе имени проекта VS автоматически обновит поле **Solution Name** именем, совпадающим с именем проекта. Если вы разрабатываете решение, в состав

которого входит больше одного проекта, такое поведение может оказаться нежелательным. Поэтому, если вы ставите перед собой цель разработки именно решения, включающего в свой состав несколько проектов, не забудьте о том, что после присвоения имени новому проекту вы можете указать и имя решения, которое вы считаете лучше подходящим для ваших целей. Например, на рис. 5.1 проекту присвоено имя **ProjectDemo**, а решению, в состав которого входит этот проект — **SolutionDemo**. VS допускает использование в именах символов пробел. В результате этого пространство имен по умолчанию для проекта будет использовать для обозначения пробелов символы подчеркивания. В некоторых случаях вам необходимо иметь в виду это соглашение об именовании, рекомендуется избегать использования символов подчеркивания в именах идентификаторов.

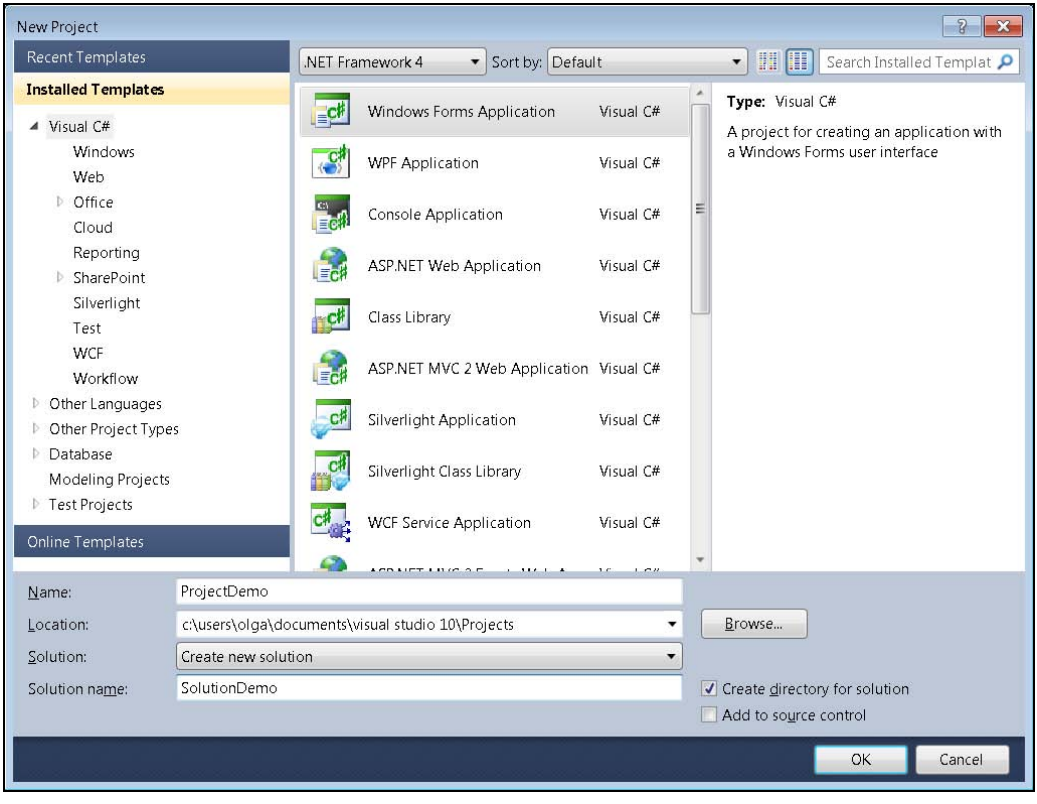


Рис. 5.1. Окно **New Project**

Если вы работаете над очень простым проектом и хотите, чтобы все файлы вашего проекта хранились в одной папке, сбросьте флажок **Create Directory For Solution**. Тем не менее, большинство приложений, которые вам предстоит разрабатывать, будут состоять из нескольких проектов, поэтому сбрасывать этот флажок в общем случае не рекомендуется — эта опция поможет вам систематизировать структуру папок и избежать путаницы и противоречий. В любом случае, когда вы

будете добавлять новый проект в состав решения, VS всегда будет создавать для него отдельную вложенную папку в составе решения.

Если вы установите флажок **Add To Source Control**, VS откроет новое окно, в котором вы сможете сконфигурировать репозиторий исходного кода (source control). Репозиторий исходного кода представляет собой хранилище для регистрации кода). Эта возможность особенно полезна для коллективной работы, когда каждый разработчик сможет регистрировать свой код в общем хранилище исходных кодов при работе над масштабными решениями. Затем нажмите кнопку **OK**, и решение будет создано.

> ***Совет***
>
> Если вы по ошибке создали проект не того типа, который вам действительно требуется, выберите из меню команды **File | Close Solution** и удалите из файловой системы папку решения со всем ее содержимым. VS часто предлагает установить на файлы блокировки ОС, поэтому решение нужно закрыть, чтобы получить возможность удаления файлов. VS поддерживает список недавно открывавшихся проектов (**Recent Projects**), и в этом списке будет присутствовать имя только что удаленного решения. Если вы выберете это имя из списка, то VS определит, что папки решения больше не существует, и предложит вам удалить соответствующий элемент из списка **Recent Projects**. После этого вы можете начать свою работу заново и использовать те же имена решения/проекта для создания проекта нужного вам типа.

Чтобы создать новый проект консольного приложения на VB, вам необходимо задать только имя проекта (параметр **Name**). Как только проект будет создан, при первом сохранении вашего проекта вам будет предложено заполнить поля **Name**, **Solution Name**, **Location**, **Create Directory** и **Add To Source Control**, которые имеют тот же смысл, что и аналогичные параметры проекта на C#, который был рассмотрен в предыдущем разделе. Итак, вы научились выполнять задачу создания нового проекта, которая для каждого из языков выполняется чуть по-разному.

Ориентируемся в окне Solution Explorer

Новый проект, только что созданный VS, отобразится в окне **Solution Explorer** (рис. 5.2). В то время как другие окна VS отображают специализированные представления информации о приложении, окно **Solution Explorer** — это то самое окно, где вы сможете получить наиболее общие сведения о вашем приложении и всех его "артефактах".

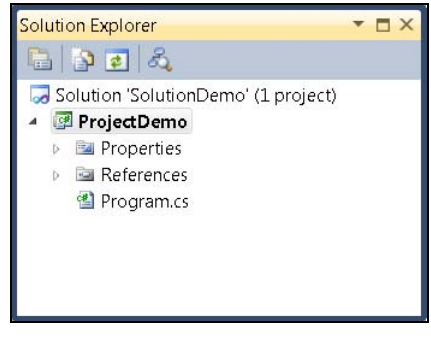


Рис. 5.2. Окно **Solution Explorer**

Одной из главных функций проекта, показанных на рис. 5.2, являются иерархические отношения. Решение у вас будет только одно. В VB по умолчанию файл решения не отображается, но вы можете изменить это положение вещей, выбрав из меню команды **Tools | Options | Projects And Solutions** и установив флажок **Always Show Solution**.

В состав решения можно добавлять новые проекты, а также папки для организации проектов. Для этого щелкните правой кнопкой мыши по имени решения в окне **Solution Explorer** и выберите из контекстного меню команды **Add | New Project**. После этого вы сможете создать новый проект и включить его в состав вашего решения. Команды **Add | Existing Project** позволяют добавить в состав открытого на данный момент решения один из уже существующих проектов. Причина, по которой эта опция предоставляется, заключается в том, что хотя решения VS ассоциируют один или несколько проектов с конкретным решением, существующий проект может быть ассоциирован и с другими решениями. Иначе говоря, несколько решений могут совместно использовать один и тот же проект.

Чтобы добавить в состав решения новую папку, выберите из контекстного меню команды **Add | New Solution Folder**. В состав решения можно добавить целую иерархию папок, если это требуется для организации проектов. Когда речь идет об иерархии папок в составе решения, нужно иметь в виду следующий важный факт: в отличие от опции создания папок в составе проекта (они представляют собой физически существующие папки файловой системы), папки решения являются логическими структурами — иными словами, опция **New Solution Folder** не создает физических папок в файловой системе. Если вы хотите, чтобы физическая структура ваших папок на жестком диске соответствовала структуре решения в окне **Solution Explorer**, то эти папки вы должны будете создать сами. Чтобы избежать путаницы, не забывайте о том, что физическое размещение проектов может и не совпадать с тем, что вы видите в окне **Solution Explorer.**

Помимо организации проектов, папки решений помогают и ассоциировать с вашими проектами некоторые специфические особенности ("артефакты"). Хотя папки решений не связаны с физическими папками файловой системы, они включаются в репозиторий таких инструментов по работе с исходным кодом, как, например, **Visual Source Safe** и **Team System.** Например, папку решения можно использовать для того, чтобы включить копию построенной вами внешней библиотеки классов в состав проекта и с ее помощью выполнить построение проекта. Таким образом, когда другие разработчики, принимающие участие в проекте, захотят получить доступ к репозиторию, все они будут работать с теми же файлами и теми же версиями. Папки решений можно использовать для файлов любых типов, включая сопроводительную документацию и вообще все, что требуется хранить организованно, и с соблюдением контроля версий.

В зависимости от типа проекта, VS может скрывать файлы некоторых типов, ассоциированные с проектом. Инструментальная панель **Solution Explorer** содержит командную кнопку **Show All Files**, которую следует нажать, чтобы увидеть все такие скрытые файлы.

Если вы выделите решение, вы увидите только кнопку **Add A New Solution Folder**, поэтому, чтобы увидеть кнопку **Show All Files**, вам необходимо выбрать проект. Примером скрытого файла является, например, папка bin со всем ее содержимым, представляющим собой вывод вашего проекта, получаемый в процессе его компиляции.

Исследование настройки свойств

Каждый проект имеет набор ассоциированных с ним свойств, которые можно конфигурировать. Когда вы только создали проект, эти параметры получают стандартные значения, наиболее распространенные для проектов данного типа. Однако вы можете индивидуально сконфигурировать эти настройки, так, чтобы они соответствовали именно вашим требованиям. Для каждого проекта существует логическая папка с именем **Properties** (см. рис. 5.2), и выбор этой папки раскроет окно **Properties** (в VB — **My Project**), как показано на рис. 5.3.

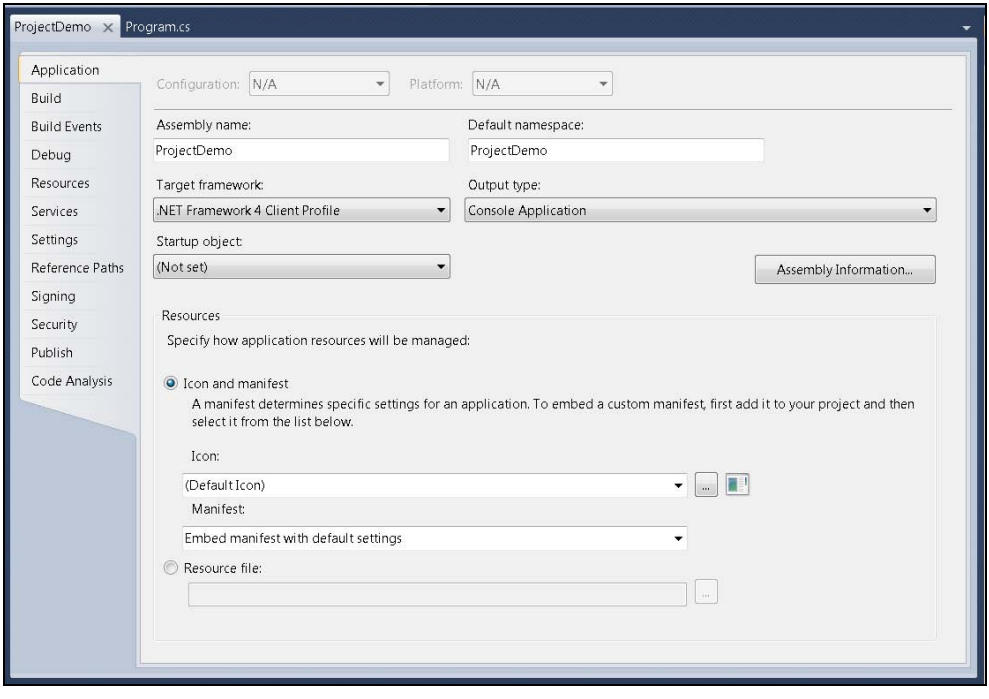


Рис. 5.3. Окно **Project Properties**

Это окно имеет несколько вкладок, на каждой из которых доступны для конфигурирования различные свойства, сгруппированные по темам, на которые указывают заголовки вкладок. В зависимости от типа проекта, количество вкладок и доступных для конфигурирования групп параметров меняется, но некоторые из вкладок содержат общие настройки, характерные для проектов любого типа. В последующих нескольких разделах будут описаны все основные опции, доступные на вкладке **Application**.

Опция Assembly Name

Файлы сборок, создаваемые VS, могут представлять собой динамически загружаемые библиотеки (*.dll) или исполняемые файлы (*.exe). Поле **Assembly name** содержит имя файла сборки для текущего проекта, причем по умолчанию в качестве

имени файла используется имя, которое вы присвоили своему проекту. Например, на рис. 5.3 поле **Assembly Name** содержит строку **ProjectDemo**. Так как в данном случае мы имеем дело с консольным приложением, выходной файл будет иметь расширение *.exe. Следовательно, файл будет называться ProjectDemo.exe. Если бы нашей задачей было построение библиотеки классов, то название файла выглядело бы так: ProjectDemo.dll.

Пространство имен по умолчанию

Пространство имен по умолчанию (**Default** в C# и **Root** — в VB) указывает, какое пространство имен будет использоваться автоматически при добавлении нового файла с кодом в состав вашего проекта. Изначально этот параметр установлен на значение, совпадающее с именем вашего проекта. Если вы хотите, чтобы для вновь добавляемых файлов использовалось другое пространство имен, установите соответствующее значение для этого параметра.

Целевая инфраструктура (Target Framework)

VS обеспечивает поддержку множества различных целевых инфраструктур .NET Framework, давая вам возможность работать с версиями .NET от v2.0 до v4.0[1]. Нужную вам версию целевой инфраструктуры следует выбирать из раскрывающегося списка **Target Framework**. Для VB вы найдете эту опцию на вкладке **Compile** в окне, которое раскрывается после нажатия кнопки **Advanced Compile Options**. Не забудьте установить опцию **VB project** из группы **.NET Framework 4.0 Client Profile** на значение .NET Framework 4.0, потому что впоследствии мы будем ссылаться на библиотеку классов .NET Framework 4.0, а версии целевой инфраструктуры должны быть совместимы с одной сборкой, чтобы ссылаться друг на друга.

Так как на вашем компьютере может быть установлено множество различных версий .NET, поддерживаемых VS 2010, вы можете свободно переключаться между проектами, которые используют различные версии .NET. Это особенно полезно, если вы работаете с несколькими различными проектами, предназначенными для различных версий .NET или если вы оказываете поддержку пользователям старых версий продукта и, наряду с этим, активно занимаетесь разработкой других проектов, используя .NET 4.0.

Тип вывода (Output Type)

Поле **Output type** (**Application type** — в VB) указывает тип сборки, который вы должны получить при компиляции и построении проекта. Поддерживаются три

[1] .NET Framework — программная технология от компании Microsoft, предназначенная для создания обычных программ и Web-приложений. Одной из основных идей Microsoft .NET является совместимость различных служб, написанных на разных языках. Например, служба, написанная на C++ для Microsoft .NET, может обратиться к методу класса из библиотеки, написанной на Delphi, на C# можно написать класс, наследованный от класса, написанного на Visual Basic .NET, и т. д. Каждая библиотека (сборка) в .NET имеет сведения о своей версии, что позволяет устранить возможные конфликты между разными версиями сборок. Подробнее см. **http://en.wikipedia.org/wiki/.NET_Framework**, **http://msdn.microsoft.com/en-us/netframework/default.aspx**. — *Прим. перев.*

типа выводных файлов — приложение Windows (Windows Application), консольное приложение (Console Application) и библиотека классов (Class Library). К настоящему моменту вы уже научились создавать консольные приложения, в результате компиляции и сборки которых на выходе получается исполняемый файл (*.exe). Чуть далее в этой главе будет рассказано о том, как создаются библиотеки классов (Class Library), в результате компиляции и построения которых на выходе получаются *.dll-файлы. В *главе 8* мы вплотную займемся проектами приложений Windows, в результате компиляции и сборки которых, как и в случае с консольными приложениями, получаются исполняемые файлы (*.exe).

Совет

Если вы работаете над проектом WPF, то для него тип выходного файла (Output Type) устанавливается на **Windows Application** (приложение Windows). Если вы установите поле **Output Type** для проекта WPF на **Console Application**, то на экране появится окно **Console**. Такой вариант будет полезен для целей промежуточной отладки, где вы можете дополнительно выводить и сообщения `Console.WriteLine`. Разумеется, в состав VS входят и замечательные отладочные средства (в том числе — окно **Output**), о которых будет более подробно рассказано в *главе 6*. Однако предложенный здесь вариант тоже представляет собой опцию, которую многие находят удобной.

Начальный объект (Startup Object)

В состав консольного приложения, как и в состав приложения WPF, можно добавить несколько методов `Main`, но в каждый конкретный момент времени активным может быть только один из этих методов `Main`. Поле **Startup** позволяет указать, какой из классов содержит тот метод `Main`, который должен служить точкой входа для вашего приложения. Одной из причин, по которым это может потребоваться, является возможность запуска вашего приложения в различных конфигурациях. Это позволит упростить тестирование за счет того, что вы сможете переходить непосредственно к программной части и будете освобождены от выполнения большого количества рутинной работы по навигации.

Значок (Icon) и манифест (Manifest)

Щелкнув мышью по кнопке с изображением многоточия, расположенной справа от раскрывающегося списка **Icon**, вы сможете выбрать файл с изображением значка, который будет символизировать ваше приложение (*.ico).

Совет

VS содержит набор готовых значков, которые вы можете использовать со своими приложениями. В каталоге C:\Program Files\Microsoft Visual Studio 10.0\Common7\VS2010ImageLibrary\1033 лежит Zip-файл с именем VS2010ImageLibrary, который и содержит эти стандартные значки. В вашей системе путь к этому файлу может быть другим, если для установки VS 2010 вы выбрали каталог, отличный от предложенного по умолчанию. Распакуйте этот файл, и вы увидите обширную библиотеку разнообразных ресурсов, в том числе — значки, звуковые файлы, анимации и системные значки, характерные для операционной системы Microsoft Windows и ее приложений.

Файл манифеста (manifest) позволяет задать настройки Microsoft Windows User Access Control (UAC) и обеспечить поддержку функции, известной под названием Click-Once. С помощью этой функции вы можете осуществлять развертывание приложений WPF с Web-страницы на локальные компьютеры конечных пользователей. Манифест описывает функции приложения и его возможности по развертыванию с помощью функции Click-Once. Так как эти манифесты генерируются автоматически при публикации приложений Click-Once, вам обычно не требуется создавать их самостоятельно. Самостоятельное написание манифестов практикуется опытными программистами и выходит за рамки знаний, необходимых новичку.

В VB имеется кнопка **UAC Settings**, которая позволяет непосредственно модифицировать файл манифеста приложения с расширением .manifest. Чтобы грамотно пользоваться этой возможностью, вам необходимо ознакомиться с настройками UAC и параметрами операционной системы, которые их задают.

Если вы выберете опцию **Resources**, вы сможете включить в проект файл ресурсов Win32, и тогда вы впоследствии сможете получать к ним доступ из кода вашего приложения. Этот прием тоже выходит за рамки стандартных знаний, необходимых новичкам.

Информация о сборке (Assembly Information)

Нажав кнопку **Assembly Information**, вы увидите окно, показанное на рис. 5.4. Оно содержит информацию о метаданных, необходимую для сборки вашего проекта. Названия большинства полей, имеющихся в этом окне, самоочевидны. Так как сборки могут включать в свой состав множество файлов, вы можете варьировать ее состав и выбирать номер версии для каждого файла.

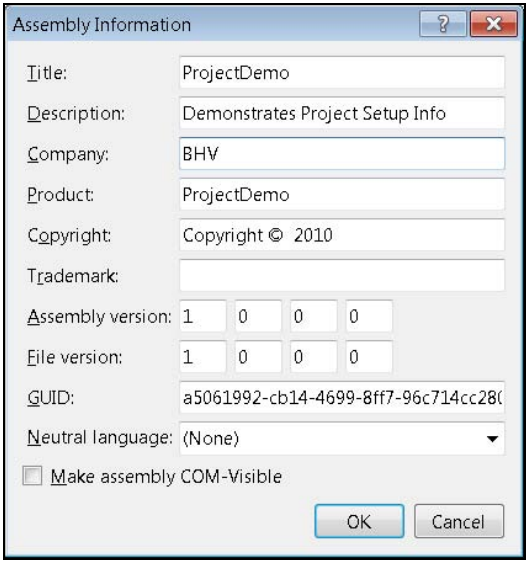


Рис. 5.4. Окно **Assembly Information**

Работая с платформой .NET, вы можете поддерживать двустороннее взаимодействие с приложениями Component Object Model (COM[2]). Эту возможность вы можете обеспечить, если присвоите сборке глобально уникальный идентификатор (Globally Unique Identifier[3], GUID), чтобы COM-приложения могли его найти, для чего следует установить флажок **COM visible**. Значение опции **Neutral Language**, предложенное по умолчанию **(None)**, следует оставить без изменений, если только вы не хотите использовать какой-нибудь другой вариант локальных настроек, отличный от en-US (который используется для приложений, работающих в США — US English).

Чтобы просмотреть, как эти настройки работают, нажмите клавишу <F6>, чтобы построить приложение, а затем перейдите в каталог, который система должна по умолчанию использовать для вывода готовой сборки вашего проекта. В моем случае путь к этому каталогу выглядел так: C:\VS2010\Chapter05\SolutionDemo\ProjectDemo\bin\Debug. В вашем случае расположение этого каталога может отличаться (это обязательно будет так, если при установке VS вы указали другой установочный каталог, отличный от предложенного по умолчанию, или если вы создали ваш проект в другой папке — тоже отличной от предложенной по умолчанию). В любом случае, найдите папку, в которой должен быть сохранен результат компиляции и сборки вашего проекта, и в этой папке вы найдете файл ProjectDemo.exe (он будет находиться в папке bin\Debug, вложенной в папку вашего проекта). Щелкните правой кнопкой мыши по файлу ProjectDemo.exe, из контекстного меню выберите команду **Properties**, а затем перейдите на вкладку **Details** раскрывшегося окна **ProjectDemo Properties**, как показано на рис. 5.5.

Как видно на иллюстрации, приведенной на рис. 5.5, в состав метаданных файла включена информация о сборке (**Assembly Information**). Это удобно не только для вас, но и для конечных пользователей, потому что благодаря этому существует возможность открыть файл и прочесть важную информацию о нем, в том числе — убедиться, что вы работаете именно с той сборкой, которая требуется. Это помогает при отладке, да и в любом случае всегда полезно знать, какие файлы установлены в вашей системе.

---

[2] COM (Component Object Model) — Объектная модель компонентов. Представляет собой технологический стандарт Microsoft, предназначенный для создания ПО на основе взаимодействующих распределенных компонентов, каждый из которых может использоваться во многих программах одновременно. Подробнее см. **http://www.microsoft.com/com/default.mspx**, **http://www.developing.ru/com/**. — *Прим. перев.*

[3] GUID (Globally Unique Identifier) — это статистически уникальный 128-битный идентификатор. Основная его особенность состоит в уникальности, которая позволяет создавать расширяемые сервисы и приложения без опасения конфликтов, вызванных совпадением идентификаторов. Хотя уникальность каждого отдельного GUID не гарантируется, общее количество уникальных ключей настолько велико (2^{128} или $3{,}4028\times10^{38}$), что вероятность того, что в мире будут независимо сгенерированы два совпадающих ключа, достаточно мала. Более подробную информацию см. по следующим адресам: **http://en.wikipedia.org/wiki/Globally_Unique_Identifier**, **http://www.rsdn.ru/?article/mag/200802/UuidCrypto.xml**. — *Прим. перев.*

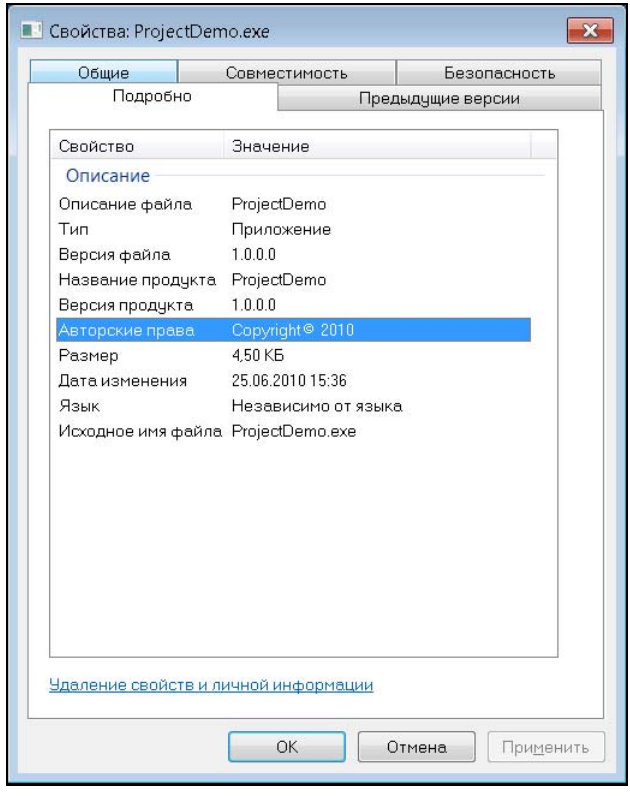


Рис. 5.5. Окно свойств файла

Ссылки на внешние сборки (Referencing Assemblies)

Как правило, все проекты ссылаются на внешние сборки. Например, файл System.dll — это сборка .NET Framework, которая содержит все примитивы (встроенные типы .NET), поэтому она обычно включается во все проекты. Для каждого типа проектов существует свой набор сборок, специфичный именно для проектов данного типа, и все они перечислены в списке **References**. Сборки, указанные в этом списке, либо необходимы для проекта выбранного вами типа, либо необязательны, но добавлены как опция, потому что содержат широко используемые библиотеки, которые обычно применяются такими проектами. Вы свободно можете удалить из списка любую из ссылочных сборок, но имейте в виду, что удаление одной из обязательных сборок, скорее всего, приведет к тому, что вы не сможете выполнить компиляцию своего проекта.

Ссылки на внешние сборки добавляются в состав проекта, чтобы сообщить компилятору о том, где следует искать типы, которые используются в вашем приложении. Когда вы запустите компилятор, он определит, какие типы используются в коде вашей программы, и будет искать их в наборе сборок, на которые ссылается ваш проект. Добавление ссылки на сборку не добавляет весь ее код в состав вашего

проекта, а просто сообщает компилятору, где искать ту информацию о типах, которой ему не хватает.

Примечание

Начинающие программисты часто путают ссылки на сборки и пространства имен. Утверждение `using` (`Imports` в VB) позволяет вам написать код без полного указания ссылок на типы в вашей сборке. С другой стороны, ссылка на сборку — это просто способ сообщить компилятору, в какой конкретно из внешних сборок следует искать эти типы. Обратите внимание, что это — две различных цели. В эту путаницу вносит свой вклад еще и тот факт, что вы получите от компилятора абсолютно одинаковые сообщения об ошибках — и в том случае, когда вы не даете ссылки на нужную компилятору внешнюю сборку, и в том случае, когда вы не включите в код директивы `using` (для VB — `Imports`) для указания пространства имен, к которому принадлежит тип, нужный компилятору для успешного завершения работы. Просто не забывайте о том, что сначала вам необходимо добавить ссылку на внешнюю сборку, а после этого — воспользоваться директивой `using` (`Imports`) в начале файла с кодом вашей программы.

Добавление ссылки на внешнюю сборку .NET

Чтобы добавить ссылку на внешнюю сборку, щелкните правой кнопкой мыши по имени проекта и выберите из появившегося контекстного меню команду **Add Reference**. На экране появится окно **Add Reference**, показанное на рис. 5.6. Перейдите в этом окне на вкладку **.NET**, и вы увидите список внешних сборок в поле **Global Assembly Cache**.

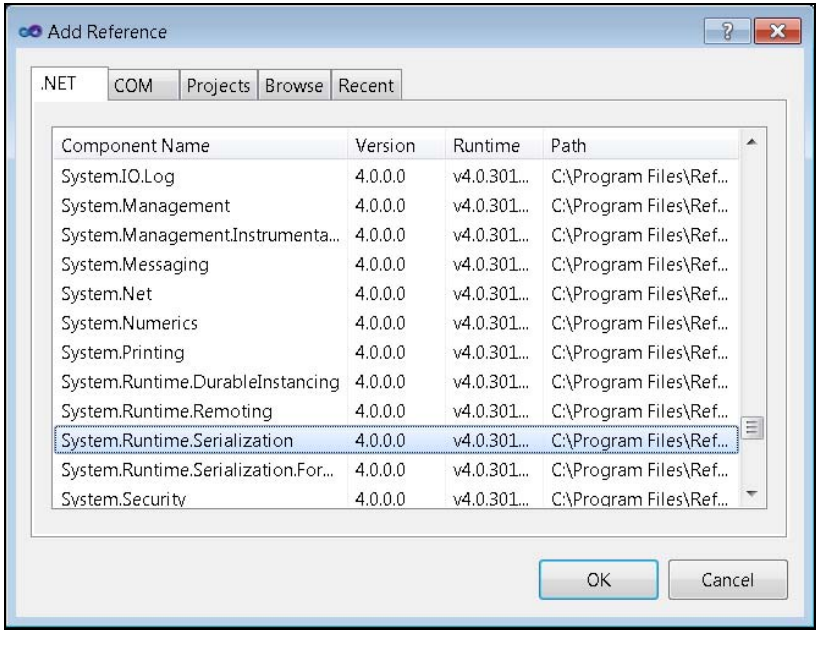


Рис. 5.6. Окно **Add Reference**

Глобальный кэш сборок (Global Assembly Cache, GAC[4]) представляет собой общий репозиторий (хранилище) сборок. Microsoft и сторонние разработчики ПО помещают в GAC все сборки, чтобы упростить их совместное использование любыми программами, которые в них нуждаются.

На вкладке COM представлены все COM-приложения, которые на данный момент зарегистрированы в вашей системе. Например, если разрабатываемое вами приложение должно взаимодействовать с Excel, вам следует перейти на вкладку COM и добавить ссылку на ту версию Microsoft Office Excel, с которой вы в настоящий момент работаете. Добавление ссылки на COM-объект приведет к тому, что VS автоматически сгенерирует новую сборку и назовет ее **Interop**. Эта сборка будет содержать "заглушки" для методов, которые упростят взаимодействие с указанным COM-объектом. Чтобы определить, какие операции возможны, вам потребуется ознакомиться с документацией на этот COM-объект (или приложение), но, в любом случае, этот способ очень удобен для работы со старыми приложениями и приложениями Microsoft Office, предоставляющими COM-интерфейс.

Осторожно!

Если вы добавляете ссылку на внешнюю сборку к проекту VB, не забудьте выбрать опцию **My Projects** для **ProjectDemo**, перейдите на вкладку **Compile** и щелкните мышью по кнопке **Advanced Compile Options**. Убедитесь в том, что значение в поле **Target Framework** установлено на **.NET Framework 4.0** (а не на **.NET Framework 4.0 Client Profile**). Причина, по которой дается этот совет, заключается в том, что проект библиотеки классов автоматически устанавливается на .NET Framework 4.0, а целевая инфраструктура должна совпадать как для ссылающейся сборки, так и для сборки, на которую дается ссылка.

На вкладке **Recent** есть список сборок, недавно добавленных к проекту. Этот список предоставляет дополнительное удобство на тот случай, если вы, добавив ссылку к одному из проектов, захотите быстро добавить эту же ссылку и к другим проектам. Вкладка **Browse** окна **Add Reference** позволяет выполнять поиск файлов *.dll, на которые вам требуется добавить ссылку. При этом вам необходимо помнить, что если вы делаете ссылку на файл *.dll для проекта, находящегося в составе

[4] GAC (Global Assembly Cache) — глобальный кэш сборок .NET для платформы Microsoft CLR (Common Language Runtime, "общеязыковая исполняющая среда" — компонент пакета Microsoft .NET Framework, исполняющая программа, написанная на .NET-совместимых языках программирования, см. **http://en.wikipedia.org/wiki/Common_Language_Runtime**). Глобальный кэш сборок представляет собой централизованное хранилище для совместно используемых библиотек. Любое приложение, которое использует определенную сборку, сначала будет искать ее в GAC, и только затем — в других местах. Обычно GAC — это каталог: C:\Windows\Assembly\GAC. Каталог GAC обладает особой структурой и содержит множество вложенных каталогов, имена которых генерируются по определенному алгоритму. Ни в коем случае не следует копировать файлы сборок в GAC вручную — вместо этого надо использовать инструменты, созданные специально для этой задачи. Эти инструменты "знают" внутреннюю структуру GAC и умеют генерировать надлежащие имена подкаталогов. Более подробную информацию можно найти по следующим адресам:
http://msdn.microsoft.com/en-us/library/yf1d93sz.aspx,
http://en.wikipedia.org/wiki/Global_Assembly_Cache,
http://www.codeproject.com/KB/dotnet/demystifygac.aspx. — *Прим. перев.*

одного и того же решения, то лучше использовать вкладку **Project**, которая управляет зависимостями (dependencies) и гарантирует, что проект будет обновлен, если изменится проект, на который он ссылается. Файловые ссылки не имеют возможности "узнать" об изменении внешнего файла *.dll, потому что он находится за пределами вашего решения.

В большинстве случаев, если вы ссылаетесь на внешний файл *.dll, вы не имеете в распоряжении кода, поэтому ссылка на проект окажется невозможной. В следующем разделе о ссылках на проекты будет рассказано более подробно.

Примечание

Окно **New Projects**, открывающееся при нажатии клавиатурной комбинации <CTRL>+<N>, содержит типы проектов Office, которые помогут вам в построении проектов, взаимодействующих с приложениями Microsoft Office.

Управление ссылками на сборки

Время от времени вам придется удалять ссылки на сборки, которые либо стали ненужными, либо были добавлены по ошибке. В C# для этого нужно открыть папку **References**, выбрать ссылку, подлежащую удалению, и нажать кнопку <DELETE>. В VB для выполнения той же самой задачи следует раскрыть окно **Properties**, выполнив двойной щелчок мышью по папке **My Project**, перейти на вкладку **References**, выбрать подлежащую удалению ссылку и щелкнуть по кнопке **Remove**. На рис. 5.7 показана вкладка **References** для проекта на VB.

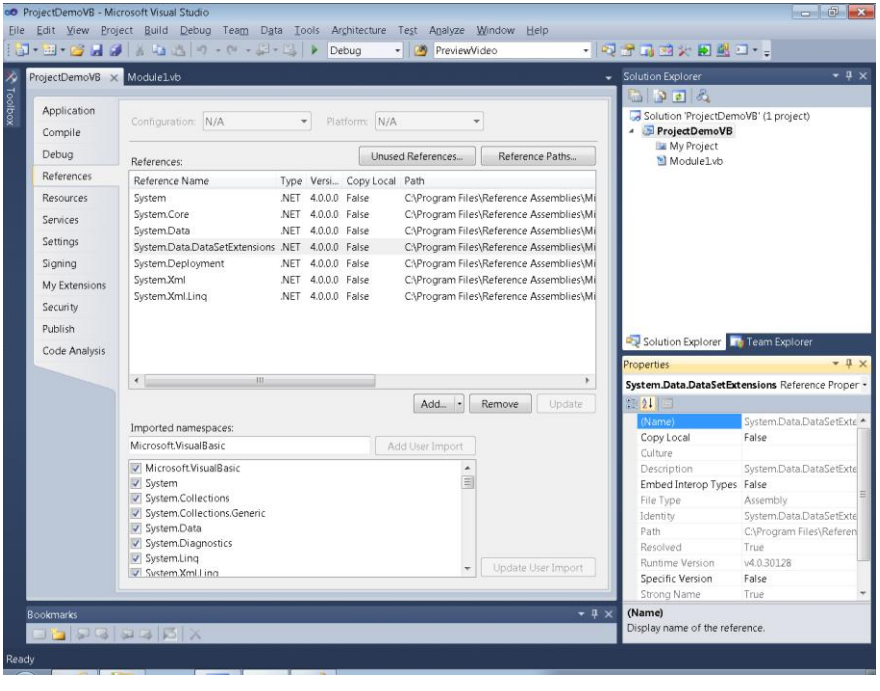


Рис. 5.7. Вкладка **References** проекта VB **My Project**

В VB на вкладке **References** предлагаются дополнительные функции. Например, вы можете воспользоваться кнопкой **Add**, чтобы добавить новую функцию. Кроме того, нажатием кнопки **Unused References** вы можете обнаружить и удалить ссылки на сборки, которые не используются в вашем коде. Щелчок мышью по кнопке **Reference Paths** позволяет указать папку, в которой VS будет искать сборки, на которые вам требуется дать ссылки.

В C# для управления путями к ссылочным сборкам в окне **Properties** есть отдельная вкладка — **Reference Paths**. Когда VS ищет сборки, на которые даны ссылки, то поиск будет проводиться сначала по каталогу текущего проекта, а затем — по папкам, указанных в списке **Reference Paths**, а затем — по папкам, перечисленным в списке сборок в окне **Add References**.

Ссылки на собственные библиотеки классов

Существуют различные причины, по которым вам может потребоваться создавать собственные библиотеки классов. Например, вы можете написать многократно используемый код или хотите систематизировать свой код и организовать его хранение в различных сборках. Чтобы сделать это, вам потребуется создавать проекты типа **Class Library**, а затем ссылаться на эти библиотеки классов из другого кода. Для начала создайте новый проект типа **Class Library**, а затем создайте ссылку на этот проект библиотеки классов из консольного приложения.

Давайте создадим проект библиотеки классов в составе нашего решения **SolutionDemo**.

Щелкните правой кнопкой мыши по решению **SolutionDemo** и выберите из контекстного меню команды **Add** | **New Project**. На этот раз, выберите в качестве типа проекта опцию **Class Library**, вместо опции **Console Application**, а затем присвойте новому проекту имя **ClassLibraryDemo**. Щелкните по кнопке **OK**, и в состав вашего решения **SolutionDemo** будет добавлен новый проект библиотеки классов. Теперь в составе вашего решения существуют два проекта.

Чтобы использовать код в составе проекта **ClassLibrary**, щелкните правой кнопкой мыши по проекту **ProjectDemo** и выберите из контекстного меню команду **Add Reference**. На этот раз выберите вкладку **Project**, на которой будут перечислены все проекты, входящие в состав одного и того же решения. Выберите проект **ClassLibraryDemo** и нажмите кнопку **OK**. После этого ссылка на проект **ClassLibraryDemo** появится в папке **References** проекта **ProjectDemo**.

Совет

Иногда вам может потребоваться сбросить все свойства переименованного проекта. Вы можете переименовать любой проект, выполнив по нему щелчок правой кнопкой мыши и выбрав из появившегося меню команду **Rename**. Однако это не изменит имени папки проекта, физически существующей в файловой системе. Если вы хотите физически изменить имя физической папки, закройте решение (выберите из меню команды **File** | **Close Solution**), а затем переименуйте нужную папку. Когда вы вновь откроете решение, **Solution Explorer** не сможет загрузить проект. Это происходит потому, что имя папки проекта в составе файла решения изменено не было. Чтобы ис-

править эту ситуацию, выберите ваш проект в окне **Solution Explorer** и откройте окно **Properties**. В окне **Properties** выберите свойство, указывающее путь к файлу (**Path**), и введите имя пути к только что переименованному файлу проекта. Как вариант, можно нажать кнопку с изображением многоточия, расположенную справа от поля имени пути, и, перемещаясь по файловой системе, найдите ваш файл проекта (*.csproj). Затем переместитесь обратно в окно **Solution Explorer**, щелкните правой кнопкой мыши по имени проекта, который не был загружен, и нажмите кнопку **Reload Project**.

Теперь, когда у вас уже есть ссылка на библиотеку классов, можно писать код, который будет использовать объекты из данной библиотеки. О том, как это делается, будет рассказано в следующем разделе.

Использование кода из библиотеки классов

Чтобы иметь возможность использования кода библиотеки классов, необходимо убедиться в том, что на эту библиотеку классов дана ссылка.

Если вы пользуетесь C#, вы можете воспользоваться директивой `using`, а если вы работаете в VB, пользуйтесь директивой `Imports`. Это позволит вам использовать типы, определенные в библиотеке классов, не указывая полного пути к ним.

После того как вы убедились в том, что имеете ссылку на сборку библиотеки классов, а также в том, что управление пространствами имен осуществляется правильно, можно начинать пользоваться этой библиотекой классов и создавать экземпляры объектов, на которые дана внешняя ссылка. Кроме того, вы сможете инициировать эти объекты и получать доступ к ним так, как если бы они представляли собой объекты из вашей собственной сборки.

.NET CLR позаботится о том, чтобы гарантировать, что ваши вызовы к библиотеке классов будут работать корректно и совершенно незаметно для вас. В предыдущем разделе было показано, как создать ссылку из одного объекта на другой, чтобы дать компилятору возможность найти нужную сборку. В этом разделе будет рассказано о том, как писать код, который указывает, какие объекты из библиотеки классов следует использовать.

Предположим, что вы пишете приложение, предназначенное для ведения учета успеваемости студентов. Для этого вам потребуется библиотека классов, которая упростит ведение учета студентов. Для этого можно переименовать файлы Class1.cs или Class1.vb в составе проекта **ClassLibraryDemo** в Student.cs или Student.vb.

Если вы работаете на C#, VS запросит у вас, не хотите ли вы переименовать имя файла класса из Class1 в Student. VB переименует файл класса автоматически, не выводя никаких запросов. Это — удобный способ синхронизировать имена ваших классов и файлов. Обычной практикой является создание в составе файла только одного класса. В листингах 5.1 и 5.2 приведен код нового класса `student` после переименования и внесения необходимых изменений, реализующих функции учета студентов.

Листинг 5.1. Код библиотеки классов на C#

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;

namespace ClassLibraryDemo
{
       public class Student
       {
              public List<int> GetStudentGrades(string studentName)
              {
                     return new List<int> { 80, 100, 95 };
              }
       }
}
```

Листинг 5.2. Код библиотеки классов на VB

```
Public Class Student
    Public Function GetStudentGrades(ByVal studenName As String) As
List(Of Integer)
              Dim intList As New List(Of Integer)
              intList.Add(80)
              intList.Add(100)
              intList.Add(95)
              Return intList
    End Function
End Class
```

Для нашего обсуждения важной частью листингов 5.1 и 5.2 является класс `Student`, входящий в пространство имен `ClassLibraryDemo`. Как вы помните, это пространство имен необходимо вам, чтобы получить ссылку на объект `Student` из вызывающего кода. Как это делается, продемонстрировано в листингах 5.3 и 5.4. Не забывайте, что в VB пространство имен неявно устанавливается так, чтобы совпадать с определением пространства имен на странице **My Project**, а это определение по умолчанию совпадает с именем проекта.

Листинг 5.3. Код приложения на C#, вызывающий код библиотеки классов

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using ClassLibraryDemo;

namespace ProjectDemo
{
        class Program
```

```
        {
                static void Main(string[] args)
                {
                        string studentName = "Joe";
                        Student myStudent = new Student();
                        List<int> grades = myStudent.GetStudentGrades(studentName);

                        Console.WriteLine("Grades for {0}:", studentName);
                        foreach (int grade in grades)
                        {
                                Console.WriteLine(" - " + grade);
                        }
                        Console.ReadKey();
                }
        }
}
```

Листинг 5.4. Код приложения на VB, вызывающий код библиотеки классов

```
Imports ClassLibraryDemoVB

Module Module1

    Sub Main()
            Dim grades As List(Of Integer)
            Dim studentName As String = "Joe"
            Dim myStudent As New Student

            grades = myStudent.GetStudentGrades(studentName)

            Console.WriteLine("Grades for {0}:", studentName)

            For Each grade In grades
                    Console.WriteLine(" - " & grade)
            Next

            Console.ReadKey()
    End Sub
End Module
```

В листингах 5.3 и 5.4 обратите особое внимание на директиву `using` (в VB — `Imports`), указывающую на то, что вы можете пользоваться типами из пространства имен `ClassLibraryDemo`, указывая их в сокращенной, а не в полной форме. Далее, обратите внимание, что код в листингах 5.3 и 5.4 создает экземпляры `Student` и `myStudent`, а затем вызывает метод `GetStudentGrades`.

Совет

Вызов к `Console.ReadKey` в листингах 5.3 и 5.4 приводит к тому, что выполнение программы останавливается, и программа ждет, когда пользователь нажмет клавишу на клавиатуре. Если бы метода `Console.ReadKey` не было, то программа завершала бы выполнение метода `Main`, что приводило бы к закрытию приложения еще до того, как пользователь получал бы возможность просмотра вывода программы.

Далее, вам потребуется выполнить компиляцию кода с тем, чтобы убедиться, что код не содержит синтаксических ошибок, а затем запустить новую программу с тем, чтобы убедиться в ее корректной работе. В следующем разделе будет рассказано о том, как выполняются компиляция программы и ее запуск в среде VS.

Компиляция приложений

В меню **Build** присутствуют несколько различных опций компиляции. По этой причине новичкам не всегда бывает сразу же ясно, какую опцию нужно использовать в каждом конкретном случае. Опции компиляции могут иметь различные диапазоны действия — либо в пределах только данного проекта, либо в масштабах всего решения. Опции в верхней части меню действуют в масштабах всего решения, а опции из нижней части контекстно-зависимы и действуют лишь применительно к текущему проекту. В последующих нескольких разделах оба набора опций будут описаны подробно. После прочтения этих разделов вы научитесь строить, перестраивать и очищать как проекты, так и целые решения.

Построение решений и проектов

Построение (Building) обычно означает, что сначала вы запускаете компилятор, чтобы осуществить компиляцию всех исходных файлов. Иногда "построение" означает больше, чем просто компиляцию. Например, если вы пишете приложения ASP.NET, то VS будет генерировать код на основании элементов управления Web, имеющихся на странице, а затем сгенерированный код будет скомпилирован с обычным. Таким образом, термин "построение" (build) является более точным, чем "компиляция" (compile).

Во время стандартного построения, VS строит только элементы проекта или решения, которые не являются "устаревшими" (out of date). Если говорить более конкретно, то будут перестроены только те проекты, в которые внесены изменения с момента последней компиляции, а проекты, оставшиеся без изменений, перестраиваться не будут. Опция **Build** обычно является самой быстрой, потому что при ее использовании в ходе нормального процесса разработки перестраиваются только те проекты, которые были изменены. Но следует иметь в виду, что в некоторых случаях вам потребуется перестраивать все (опция **Build All**), чтобы быть полностью уверенным в том, что вы не работаете со старым кодом.

Перестройка решений и проектов

Опция **Rebuild** выполняет те же действия, что и опция построения (build) — она выполняет построение объектов, входящих в состав проекта или решения. Причинами, по которым вам может потребоваться перестройка кода, являются:

- добавление нового кода (в данном случае необходимо гарантировать, что он работает с уже существующим);
- создание новой сборки;
- ситуации, когда в процессе стандартного построения возникают ошибки.

Многие разработчики (и я в том числе) предпочитают "стягивать" последние изменения с репозитория **Source Control** в свое решение каждое утро, прежде чем начинать работу. Это гарантирует, что текущий код всегда будет компилироваться и строиться с учетом всего, что имеется в репозитории **Source Control**. Это позволяет предотвратить возникновение ситуаций, когда ваше локальное решение будет сильно отличаться от того, что присутствует в репозитории **Source Control.**

Прежде чем вы приступите к развертыванию вашего приложения, вам потребуется перестроить его, чтобы убедиться в том, что сборка происходит без ошибок. В зависимости от вашего процесса, вам может потребоваться выполнить тестирование только что перестроенного кода, прежде чем вы начнете распространять свое приложение. Изготовление новой сборки путем перестройки кода позволит вам удостовериться в том, что вы начали выполнять развертывание именно последней (самой актуальной) версии сборки.

Иногда нормальный процесс построения вашего приложения может не сработать или сработать некорректно. Например, возможна такая ситуация, когда вы обнаружите ошибки, ассоциированные с кодом, написанным вами ранее. Хотя VS — это действительно великолепный инструмент, который способен управлять очень сложными зависимостями и связями между проектами, сложные ситуации, когда не все полученные сборки корректно строятся и потом корректно работают. Если у вас возникнет такая ситуация, вы всегда можете воспользоваться опцией **Rebuild All**, и это поможет вам принудительно перестроить все элементы проекта или решения.

Процесс перестройки (rebuild) требует для завершения больше времени именно потому, что теперь будут перестраиваться все элементы проекта. Если вы работаете над совсем небольшим проектом, то эта разница может быть совсем не заметна. Но если вы создаете масштабное решение, в состав которого входят десятки проектов, то последовательная цепочка перекомпиляций и перестроек может отнять у вас целый рабочий день, существенно снизив производительность вашего труда. Часто перестройка проекта не требует больших трудозатрат, чем обычное построение, но бывают и другие случаи, когда разница во времени может оказаться существенной. Скорее всего, в таких ситуациях вам может помочь именно полная перестройка всего решения. Но, сказав это, необходимо одновременно заметить, что каждая последующая версия VS повышает производительность процесса перестройки по сравнению с предыдущей. Таким образом, вы можете интерпретировать производительность как соотношение между скоростью выполнения операций **Build** и **Rebuild**, вместо того, чтобы сравнивать VS с любым другим инструментом.

Очистка решений и проектов

Операция очистки (Clean) удалит все выходные файлы, включая файлы *.dll, *.exe, или любые другие, созданные в результате процесса построения (Build). Вам довольно часто может потребоваться эта операция, если вам хочется иметь полную уверенность в том, что все выходные сборки перестроены заново, или просто получить более компактную копию проекта.

Обычно полная перестройка гарантирует, что все выходные файлы являются новейшими. Кроме того, вы можете выполнить и операцию очистки, чтобы убедиться в том, что все ранее полученные сборки удалены, а затем выполнить построение и просмотреть, какие файлы были созданы в результате. Помимо прочего, это поможет вам разобраться, включает ли сборка решения все проекты. При нормальных обстоятельствах VS автоматически управляет всеми зависимостями, о чем будет рассказано в следующем разделе. Однако в более сложных сценариях некоторые разработчики могут менять эти зависимости вручную. Опция очистки представляет собой инструмент, позволяющий определить, действительно ли проект компилируется и собирается в процессе построения решения. С практической точки зрения вам время от времени потребуется просматривать даты сборок (с той же целью). Просто опция очистки — это дополнительный инструмент, которым проще пользоваться.

Наиболее распространенное применение опции очистки — получение более компактной версии проекта. Вам может потребоваться сжать проект или решение одним из архиваторов и, например, отправить его коллегам по электронной почте. Естественно, в этом случае ваша цель заключается в том, чтобы минимизировать размер файла, прикладываемого к сообщению. При этом если исходный код сжимается очень хорошо, то файлы *.dll и *.exe потребляют больший объем свободного пространства, даже когда они добавляются в архив. Таким образом, если перед архивацией проекта или решения вы выполните команду **Clean**, то вы получите намного более компактный архив.

Управление зависимостями и порядком построения

Зависимость описывает для VS, от каких других проектов зависит данный проект, чтобы его работа была корректной. Скажем, что касается примера, рассматриваемого в данной главе, то проект **ProjectDemo** ссылается на **ClassLibraryDemo** и использует код, содержащийся в **ClassLibraryDemo**. Это значит, что **ProjectDemo** зависит от **ClassLibraryDemo**. VS добавит эту зависимость автоматически, и для вас это очень хорошо, потому что когда VS будет создавать сборку вашего решения, то будет учтена актуальность всех ваших проектов. VS будет управлять деревом зависимостей. Каждый раз, когда вы будете перестраивать вашу сборку, VS будет искать в этом дереве нужные зависимости и строить все проекты, которые зависимостей не имеют. Затем VS построит все проекты, которые зависят от последнего набора проектов и которые были перестроены. Этот процесс будет продолжаться до тех пор, пока не будет перестроено все решение. Пока все проекты,

которые находятся на вершине дерева зависимостей, не будут ссылаться на последние версии обновленных проектов, на которые они ссылаются.

Вы можете управлять зависимостями вручную. Для этого нужно щелкнуть правой кнопкой мыши по имени проекта или решения в окне **Solution Explorer**, а затем выбрать из меню команду **Project Dependencies**. Откроется окно **Project Dependencies**, показанное на рис. 5.8.

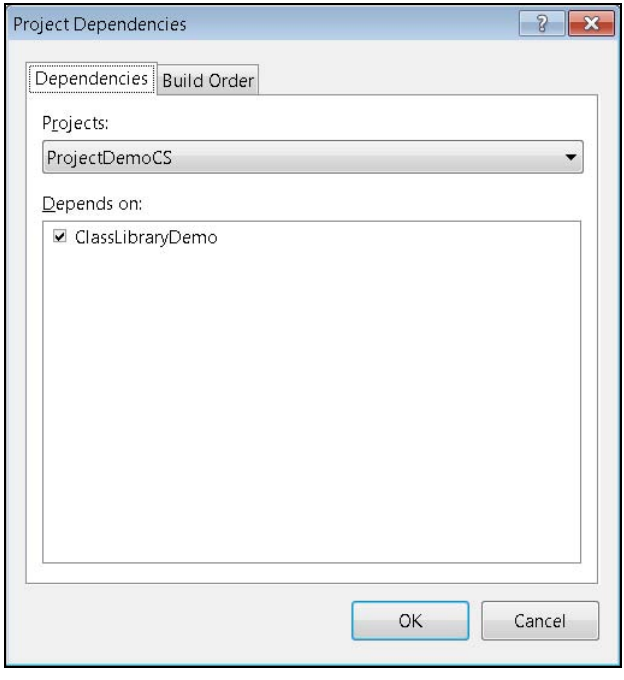


Рис. 5.8. Окно **Project Dependencies**

В окне **Project Dependencies** вы можете выбрать из раскрывающегося списка тот проект, для которого вы хотите установить зависимость. Существует и список проектов, зависимость от которых вам требуется установить.

Как показано на рис. 5.8, проект **ProjectDemo** имеет зависимость от проекта **ClassLibraryDemo.** VS создает эту зависимость автоматически.

Зависимости проекта непосредственно влияют на порядок построения проекта. Как уже говорилось ранее, проекты, от которых зависят другие проекты, строятся в первую очередь, до построения зависимых проектов. В окне **Project Dependencies**, показанном на рис. 5.8, вы можете перейти на вкладку **Build Order**, где сможете управлять порядком выполнения процесса сборки. Кроме того, попасть на вкладку **Build Order** можно, выполнив щелчок правой кнопкой мыши по имени нужного вам проекта в окне **Solution Explorer**, а затем выбрав из контекстного меню команду **Project Build Order**. Вкладка **Build Order** показана на рис. 5.9.

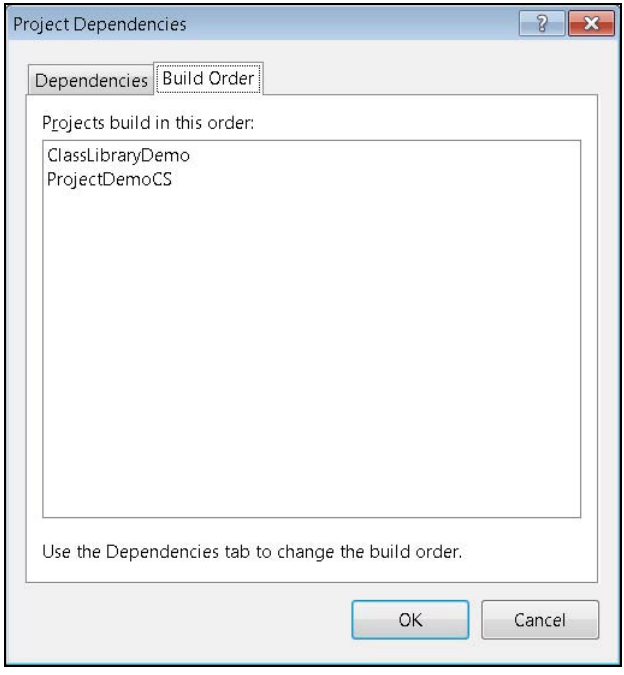


Рис. 5.9. Вкладка **Build Order**

Внимание!

Не следует изменять порядок зависимостей проекта, за исключением тех случаев, когда вы очень хорошо понимаете, что вы делаете. Результаты ошибки могут быть очень серьезными, и восстановление правильных зависимостей для сложного проекта может потребовать длительного времени. Автоматическое управление зависимостями в VS очень надежно, и за редким исключением вы можете на него положиться.

Управление параметрами компиляции

Окно свойств проекта содержит вкладку для параметров компиляции. Для каждого проекта вы можете индивидуально устанавливать параметры компиляции. На рис. 5.10 показана вкладка C#. Открыть ее можно, выполнив двойной щелчок мышью по папке **Properties** для выбранного проекта. Некоторые из параметров, представленных на этой вкладке, относятся к сложным аспектам программирования и выходят за рамки этой книги. Например, в данной книге не обсуждаются аспекты COM Interop, генерация небезопасного кода (unsafe code generation) или сборки сериализации (serialization assemblies). Поэтому я просто кратко упомяну эти настройки и, не вдаваясь в подробности, дам краткое пояснение их смысла и предназначения, чтобы вы знали, что делать, если в будущем столкнетесь с одной из них.

Константы компиляции `DEBUG` и `TRACE` позволяют вам использовать классы `Debug` и `Trace`, соответственно. Эти классы являются членами пространства имен `System.Diagnostics` инфраструктуры .NET Framework. Кроме того, вы можете

выполнять построение кода, который зависит от ваших собственных, индивидуально заданных констант, добавляя эти константы в поле **Conditional Compilation Symbols** в виде списка строк, в котором в качестве разделителя используется запятая.

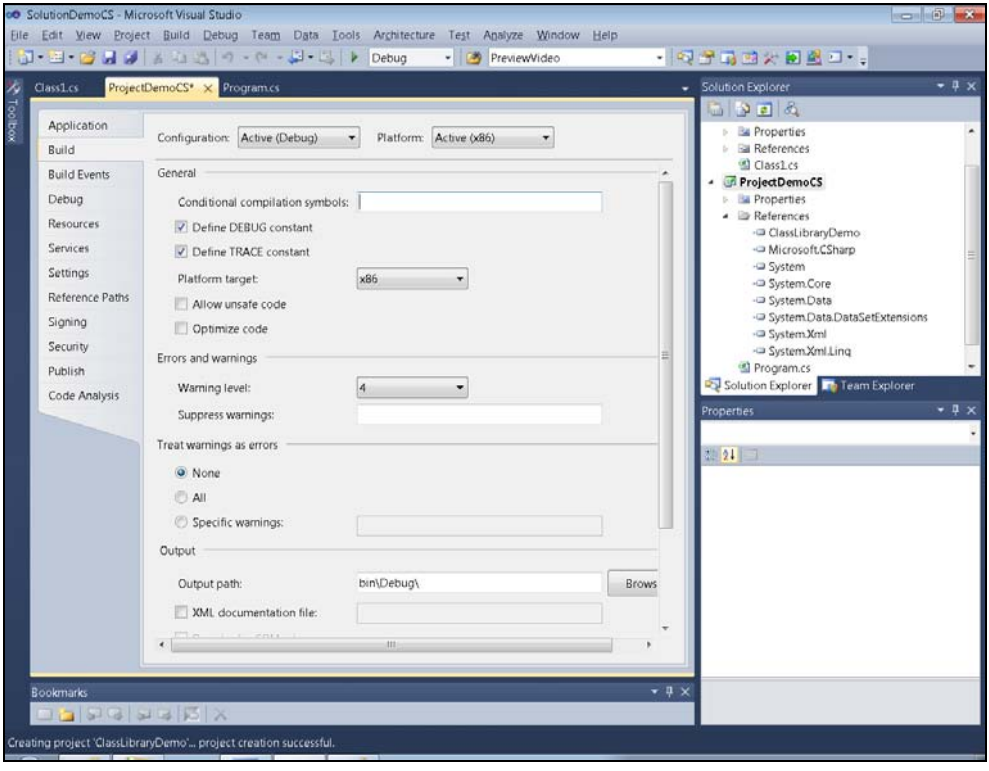


Рис. 5.10. Вкладка, задающая опции компиляции в C#

C# дает возможность писать код, который классифицируется как "небезопасный" (unsafe), подразумевая, что вы можете использовать указатели и другие функциональные возможности в небезопасном контексте. Небезопасный (Unsafe) код по-прежнему является управляемым кодом (управление осуществляется CLR). Однако в данной ситуации под "небезопасностью" понимается то, что CLR не может гарантировать, что код является безопасным, потому что данный код содержит указатели. Эта возможность предназначается для опытных программистов, поэтому флажок и не взведен. Это сделано в страховочных целях, поэтому устанавливайте данную опцию, когда вы точно знаете, чего вы хотите.

Все предупреждающие сообщения ассоциируются с определенными уровнями предупреждений. По умолчанию, всего таких уровней четыре, и в это число входят все предупреждения от компилятора. Если установить уровень предупреждений на один из нижних уровней, то выводиться будут все предупреждения этого уровня и более высоких. Кроме того, вы можете подавлять конкретные предупреждения, добавляя их в виде списка, где в качестве разделителя используется запятая, в поле

Suppress Warnings. На практике, подавлять вывод предупреждений не рекомендуется, потому что при этом можно проглядеть ошибку, которую потом будет очень трудно локализовать.

Когда вы построите приложение, ваша программа запустится, даже если в ходе построения имели место ошибки. Исключением из этого правила будет только та ситуация, когда ошибки появляются еще на стадии компиляции. Иногда предупреждения оказываются настолько серьезными, что их лучше трактовать как ошибки. Для этого предназначаются параметры из раздела **Treat Warnings As Errors**, которые предоставляют вам необходимую гибкость для настройки различных сценариев реагирования на предупреждения.

Путь к каталогу для вывода результатов построения программы по умолчанию задается как вложенный каталог bin\Debug в составе папки вашего проекта (для отладочного вывода) и bin\Release (для сборок без отладочной информации). При желании эти пути можно изменить.

Установка опции **XML documentation file** приведет к тому, что из вашего кода будут извлекаться документирующие комментарии XML и сохраняться в указанном вами файле XML. Установка этой опции увеличивает время, необходимое для процесса построения, поэтому устанавливать ее при выполнении отладочной сборки не обязательно. Файл с документацией XML можно использовать с инструментарием сторонних разработчиков, предназначенным для автоматического построения технической документации.

Флажок **Register For COM Interop** следует устанавливать только в том случае, если вы создаете сборку .NET, которая вызывается из COM-приложения. Если вы выполняете сериализацию XML для типов в вашей сборке, то для ускорения процесса сериализации можно установить опцию **Generate Serialization Assembly**.

C# имеет еще одну группу настроек на вкладке **Build Events**. Запускать код можно до или после построения каждого проекта. Вы можете задать условия построения, при которых оно осуществляется всегда (**always**), после успешной сборки одного из проектов (**on a successful build**), или только когда осуществляется обновление (**only when an update occurs**). События при построении (**build events**) имеют набор макросов, которые позволяют вам получать информацию о текущем процессе сборки.

VB имеет набор опций, специфичных для компилятора VB. Эти опции показаны на странице **Compile** (рис. 5.11). Большинство опций компиляции для VB и C# подобны друг другу, за исключением **Option Explicit**, **Option Strict**, **Option Compare** и **Option Infer**. В VB можно отключить объявление переменных перед их использованием. Когда установлена опция **Option Explicit**, вы обязаны объявлять каждую переменную до ее использования. Кроме того, любой тип можно присвоить другому по умолчанию, но если установлен флажок **Option Strict**, то вы обязаны будете использовать код, который осуществляет преобразование типов из более длинных в более короткие (эту опцию часто называют "сужающим преобразованием" — narrowing conversion).

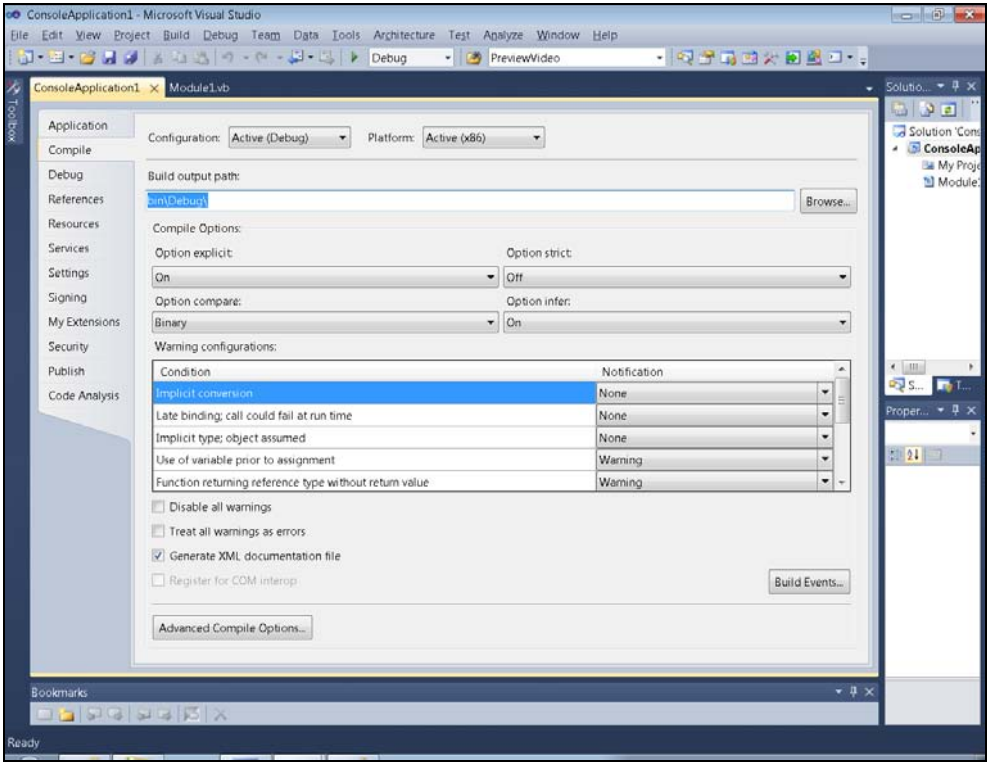


Рис. 5.11. Страница **Compile** для опций компиляции VB

Установка опции **Option Compare** принудительно задает сравнение строк двоичным образом). Однако при работе с разными языками, вы можете предпочесть замену **Option Compare** на **text**, чтобы иметь возможность сравнить, как это повлияет на аспекты, специфичные для различных языковых настроек, при сравнении строк. Опция **Option Infer** позволяет работать с переменными, в предположении того, что их типы основываются на значениях, присваиваемых этим переменным, а не на явном объявлении типов переменных. Вот пример такого присваивания:

```
Dim studentName = "Joe"
```

В этом примере совершенно очевидно, что `"Joe"` — это строка (`String`). Поскольку опция **Option Infer** установлена, этот синтаксис законен, и переменная `studentName` получает тип `String`, потому что ей присваивается строковое значение.

Перемещение по проекту в режиме просмотра классов

Режим просмотра классов представляет собой альтернативный метод работы с классами, который позволяет вам просматривать решения и артефакты проектов, используя логическую организацию кода. Если вы работаете на C#, вы можете открыть режим просмотра классов, нажав клавиатурную комбинацию <CTRL>+<W>,

<C> или выбрав из меню **View** команду **Class View**. Работая на VB, вы можете переключиться в режим просмотра классов, нажав клавиатурную комбинацию <CTRL>+<SHIFT>, <C> или выбрав из меню следующие команды: **View | Other Windows | Class View**. На рис. 5.12 показано окно **Class View**.

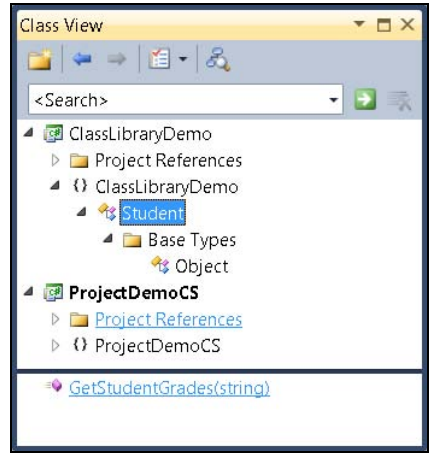


Рис. 5.12. Окно **Class View**

В окне **Class View** вы можете просматривать иерархию узлов, началом которой является проект, включать в нее ссылки и пространства имен и помещать классы внутрь этих пространств имен. В составе каждого класса могут содержаться базовые типы (**Base Types**), которые представляют собой списки базовых классов, производных классов и интерфейсов, реализованных для конкретных классов. Обратите внимание, что на рис. 5.12 выбран класс `Student`, а в нижней панели показан список членов этого класса.

Как показано на инструментальной панели **Class View**, вы можете создавать новые папки, использовать клавиши с изображением стрелок для перемещения вверх и вниз по иерархической лестнице, а также управлять опциями, которые задают, что отображать в иерархии. Кроме того, обратите внимание на кнопку с изображением объектов, которая показывает, как создавать диаграмму классов. Эту тему мы подробнее рассмотрим в следующем разделе.

Использование конструктора классов (Class Designer)

Во время работы над проектом иногда бывает полезно иметь возможность высокоуровневого просмотра содержимого проекта, особенно если этот проект был создан кем-то другим, и ранее вы с ним не работали. Именно здесь и будет особенно полезен конструктор классов (Class Designer). В дополнение к визуализации кода, **Class Designer** предоставляет вам базовый инструментарий для самостоятельного конструирования классов. В первую очередь, давайте рассмотрим средства визуализации существующих классов.

Визуализация существующих классов с помощью Class Designer

Каждый раз, когда вы выбираете проект в решении **Solution Explorer**, на инструментальной панели окна **Solution Explorer** появляется кнопка **Class Designer** (). Кнопка **Class Designer** появится и в окне **Class View**. Щелчком мыши по значку **View Class Diagram** вы можете вывести диаграмму классов для вашего решения (рис. 5.13).

Как видно из рис. 5.13, VS создаст новый файл, с именем ClassDiagram1.cd, который будет содержать визуальное представление для вашего кода. Как видите, окно

Properties остается открытым, давая вам возможность просматривать информацию о выбранном классе **Program**.

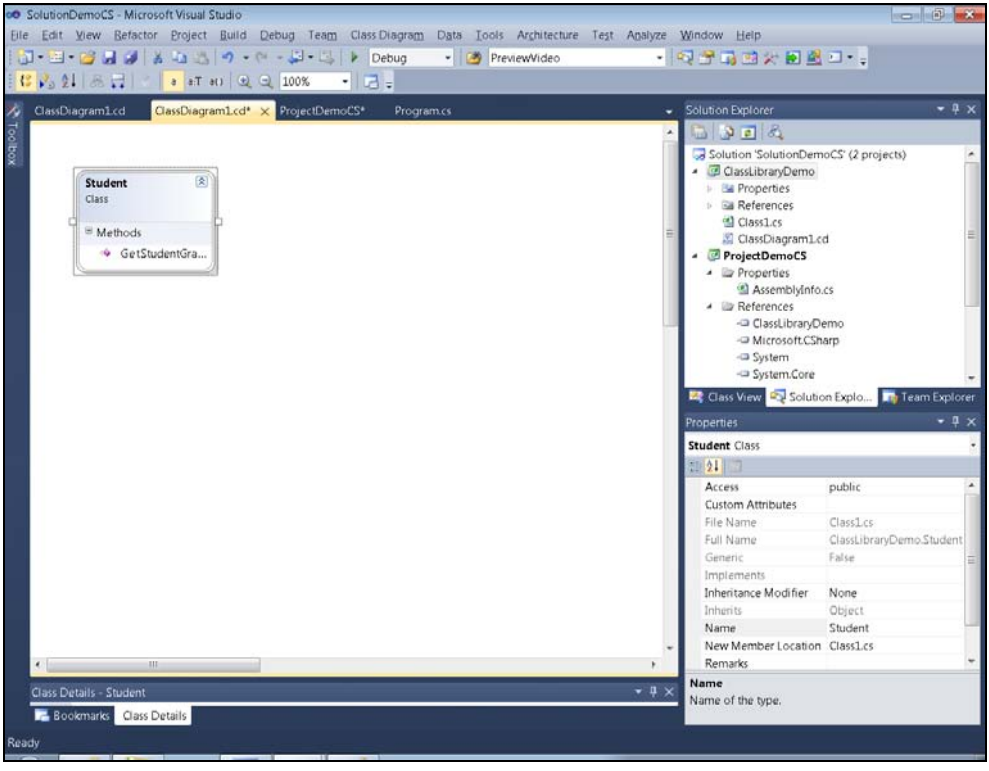


Рис. 5.13. Визуализация кода с помощью **Class Designer**

Наконец, окно **Class Details** дает возможность просмотра дополнительной информации о членах класса Program.cs. На рис. 5.13 представлено минимальное представление для одного класса, который содержит только один метод, `Main`, но, если бы ваш текущий проект содержал больше классов, вы увидели бы их все. Это может быть очень удобно, если вам требуется подробно изучить имеющуюся кодовую базу.

В дополнение к визуализации кода, в **Class Designer** вы получаете возможность вносить в код мелкие изменения, о чем будет подробнее рассказано в следующем разделе.

Генерация кода с помощью Class Designer

Class Designer предоставляет вам графические инструменты для автоматической генерации кода. В левой панели окна, показанного на рис. 5.13, имеется вкладка, которая называется **Toolbox**. Если вы наведете на нее курсор мыши и чуть-чуть задержите его над данной вкладкой, вы увидите группу значков, соответствующих таким группам опций, как **Class**, **Enum**, **Inheritance** и т. д. На рис. 5.14 показаны результаты использования инструментов **Toolbox**.

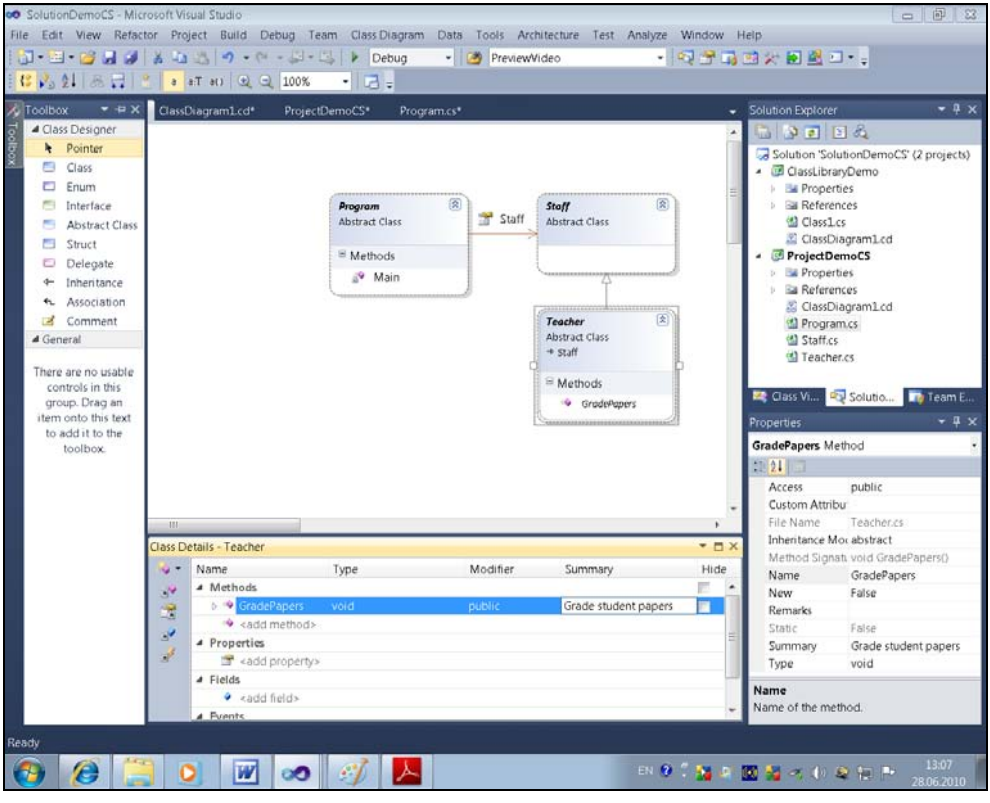


Рис. 5.14. Генерация кода с помощью **Class Designer**

Как видите (см. рис. 5.14), окно **Toolbox** имеет опции для тех типов объектов, которые вы можете добавить к своей диаграмме классов. Каждый из элементов **Toolbox** совпадает с типом кода, который вы обычно пишете. Сама диаграмма классов имеет дополнительные элементы, включая абстрактный класс **Staff**, обычный класс **Teacher**, отношение наследования, связывающее эти классы, где класс **Teacher** является производным от **Staff**, и ассоциацию от **Program** к **Staff**.

Чтобы создать новый объект, перетащите объект из окна **Toolbox** на поверхность панели **Class Designer**. После этого появится окно ввода, показанное на рис. 5.15.

Окно **New Abstract Class**, показанное на рис. 5.15, типично для большинства объектов **Class Designer**, которые вы добавляете к диаграмме классов. В этом окне нужно

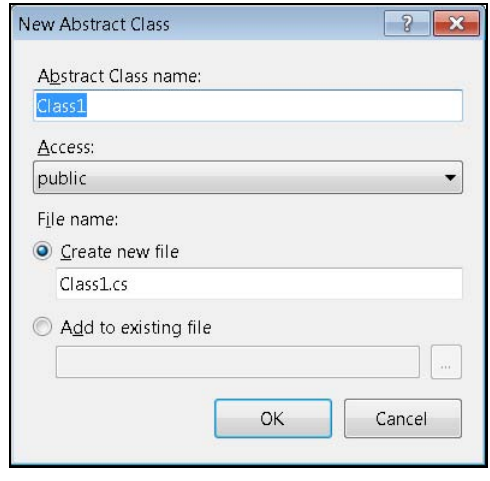


Рис. 5.15. Добавление нового объекта в **Class Designer**

ввести начальную информацию об имени класса и указать файл, в состав которого должен быть добавлен код. Впрочем, не все объекты **Toolbox** работают таким образом. Чтобы создавать ассоциации и отношения наследования, требуется выбрать нужный элемент в окне **Toolbox**, выбрать объект, с которого начинается линия в окне **Class Designer**, и протянуть ее к объекту в окне **Class Designer**, на который дается ссылка.

Еще два окна, в которых вы можете модифицировать данные — это окна **Class Details** и **Properties**. Например, в окно **Class Details** я добавил метод **GradePapers**. Вы можете сами добавлять члены в состав объектов. Для этого надо щелкнуть мышью по объекту в окне **Class Designer**, а затем добавить член в **Class Details**. Метод **GradePapers** имеет комментарий **Summary** для документации и параметр с именем **papers** и типом **List<*string*>**.

Окно **Properties** является контекстно-чувствительным. В нем показаны опции, доступные для объекта, который выбран в **Class Designer**. На рис. 5.14 в окне **Class Designer** выбран класс **Teacher**, и свойство **Summary** в окне **Properties** заполнено комментарием. В листинге 5.5 показан код на C#, содержащийся в классе Teacher.cs, а в листинге 5.6 — код на VB, содержащийся в классе Teacher.vb. Этот код создается после того, как все описанные операции будут выполнены с помощью графического редактора кода.

Листинг 5.5. Код на C#, сгенерированный с помощью Class Designer

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ProjectDemo
{
        /// <summary>
        /// Teaches Classes
        /// </summary>
        public class Teacher : Staff
        {
                /// <summary>
                /// Grade student papers
                /// </summary>
                /// <param name="papers">Papers to grade</param>
                public void GradePapers(List<string> papers)
                {
                        throw new System.NotImplementedException();
                }
        }
}
```

Листинг 5.6. Код на VB, сгенерированный с помощью Class Designer

```
''' <summary>
''' Teaches Classes
''' </summary>
Public Class Teacher
       Inherits Staff

       ''' <summary>
       ''' Grade student papers
       ''' </summary>
       Public Sub GradePapers(ByVal papers As List(Of String))

       End Sub
End Class
```

Как показано в листингах 5.5 и 5.6, код, сгенерированный с помощью **Class Designer**, по умолчанию включает директивы `using` и пространства имен в том виде, как они указаны в свойствах проекта. Имя класса, `Teacher`, совпадает с именем объекта на диаграмме классов, а метод `GradePapers` — это тот же метод, который указан в окне **Class Details**. Кроме того, в коде присутствует комментарий, как он был указан в окне **Property**. Все, что вам осталось — заменить вызов к `throw new System.NotImplementedException` вашим собственным кодом на C# или добавить ваш код для `GradePapers` на VB.

Заключение

После прочтения этой главы вы приобрели навыки создания решений и проектов. В данной главе было рассказано о том, как задавать свойства проектов и добавлять к ним новые члены. Кроме того, в данной главе продемонстрировано, как добавлять к проекту библиотеки классов и как следует ссылаться на эти библиотеки из других проектов. Если вы предпочитаете более формальный вариант процесса разработки, то можете воспользоваться предлагаемой VS возможностью использования **Class Designer** как для визуализации кода, так и для его автоматической генерации. В следующей главе книги будут подробно описаны процессы написания кода и его отладки.

Глава 6

Отладка с помощью Visual Studio

В данной главе будут рассмотрены следующие ключевые концепции:

- Исследование имеющегося отладочного инструментария;
- Установка точек останова;
- Исследование статуса программы;
- Решение проблем с помощью отладочных средств VS.

Наш код часто содержит ошибки — чаще, чем нам хотелось бы. К счастью, на случай возникновения необходимости в устранении ошибок VS предлагает богатый набор отладочных инструментов. В этой главе будет показано, как использовать отладчик VS для устранения проблем в работе вашего кода. Здесь будет показано, как решать проблемы путем установки точек останова, путем пошагового исполнения программы и исследования статуса программы. Кроме того, в эту главу включен отдельный раздел, посвященный инструментам, предназначенным для исследования структуры кода программы во время разработки. Помимо установки точек останова вы узнаете, как выполнять их индивидуальную настройку и как управлять списком точек останова. Далее будут представлены опции, доступные при пошаговом исполнении кода. Кроме того, в данной главе будут продемонстрированы способы, применяя которые вы можете определять значения различных переменных, а также описаны различные инструменты для изучения кода программы. Во-первых, начнем с обсуждения фрагмента кода, на примере которого будут демонстрироваться концепции отладки, излагаемые в этой главе.

Код, на примере которого в данной главе будут демонстрироваться приемы отладки

Чтобы продемонстрировать отладку полнофункционального приложения, использующегося в реальной жизни, потребуется написать множество страниц кода. Для целей данной главы это — избыточно, и, кроме того, такой подход серьезно затруднит понимание основ. Поэтому в данной главе мы рассмотрим пример, который имитирует полномасштабное приложение. Хотя это — всего лишь модель, для целей данной главы этого примера будет вполне достаточно. При отладке вам потребуется проходить по иерархическим деревьям в составе вашего кода, где один

метод вызывает другой, который, в зависимости от задач, выполняемых вашей программой, может иметь несколько уровней вложенности. Код, в примерах из данной главы, будет иметь множество уровней вызовов, так что это наглядно проиллюстрирует технику отладки программ в VS.

Листинг 6.1 показывает пример кода, который будет использоваться в данной главе. Программа представляет собой консольное приложение, подобное всем тем программам, которые уже обсуждались в предшествующих главах этой книги. Создайте новый проект консольного приложения, выбрав из меню команды **File | New | Project**, затем в качестве типа приложения выберите опцию **Console Application**, дайте имя новому проекту и нажмите кнопку **OK**. Приложение, код которого приведен в листингах 6.1 и 6.2, вычисляет скидку для клиента, основываясь на процентной ставке, установленной для этого клиента, и на том товаре или услуге, которые были этим клиентом заказаны.

Листинг 6.1. Код на C#, предназначенный для иллюстрации примеров из этой главы

```
// C#: Program.cs

using System;

namespace DebugAndTestDemo
{
        class Program
        {
                static void Main()
                {
                        Customer cust = new Customer();
                        cust.Discount = .1m;

                        Order ord = new Order();
                        ord.AddItem(5.00m);
                        ord.AddItem(2.50m);

                        cust.Order = ord;
                        decimal discount = cust.GetOrderDiscount();

                        Console.WriteLine("Customer Discount: {0}", discount);
                        Console.ReadKey();
                }
        }
}

// C#: Customer.cs

namespace DebugAndTestDemo
```

```
{
        class Customer
        {
                public decimal Discount { get; set; }
                public Order Order { get; set; }

                public decimal GetOrderDiscount()
                {
                        return Order.Total * Discount;
                }
        }
}

// C#: Order.cs

using System.Collections.Generic;

namespace DebugAndTestDemo
{
        class Order
        {
                private List<decimal> orderItems = new List<decimal>();

                public decimal Total
                {
                        get
                        {
                                decimal amount = 0;

                                foreach (var item in orderItems)
                                {
                                        amount = amount + item;
                                }

                                return amount;
                        }
                }

                public void AddItem(decimal amount)
                {
                        orderItems.Add(amount);
                }
        }
}
```

Листинг 6.2. Код на VB, предназначенный для иллюстрации примеров из этой главы

```
' VB: Module1.vb

Module Module1

    Sub Main()
        Dim cust As Customer = New Customer()
        cust.Discount = 0.1D

        Dim ord As Order = New Order()
        ord.AddItem(5D)
        ord.AddItem(2.5D)

        cust.Order = ord

        Dim discount As Decimal = cust.GetOrderDiscount()

        Console.WriteLine("Customer Discount: {0}", discount)
        Console.ReadKey()
    End Sub
End Module

' VB: Customer.vb

Class Customer
        Property Discount As Decimal
        Property Order As Order
        Function GetOrderDiscount() As Decimal
                Return Order.Total * Discount
        End Function
End Class

' VB: Order.vb

Class Order
        Private orderItems As New List(Of Decimal)

        Public ReadOnly Property Total() As Decimal
                Get
                        Dim amount As Decimal = 0

                        For Each item In orderItems
                                amount = amount + item
                        Next

                        Return amount
```

```
            End Get
        End Property

        Sub AddItem(ByVal item As Decimal)
            orderItems.Add(item)
        End Sub
End Class
```

Даже беглый взгляд на код, приведенный в листингах 6.1 и 6.2, показывает, что эта программа более сложна, чем примеры, рассматривавшиеся в предыдущих главах. Чтобы понять, что происходит при запуске программы, начнем ее рассмотрение с метода `Main`, который представляет собой точку входа приложения. В методе `Main` создаются два экземпляра объектов — `Customer` и `Order`.

После того как будет создан экземпляр объекта `Customer`, можно просмотреть его свойство `Discount`, которое получает значение .1 (10%). Это значит, что каждый клиент может получить индивидуальную скидку, что может быть полезным, если вы захотите, например, поощрять постоянных клиентов, приобретающих у вас те или иные товары.

Далее, создается экземпляр объекта `Order`, после чего происходит вызов к методу `AddItem`, который добавляет элемент к ссылке на заказ `ord`. Этот код только добавляет объем заказа, а в реальной ситуации, скорее всего, в программе должна быть еще ссылка на детальную информацию о том, что именно было заказано. Класс `Customer` имеет свойство `Order`, которое код передает и экземпляру `ord` класса `Order`. После этого вы получаете конкретного клиента (`Customer`) с объемом скидки, и этот экземпляр ссылается на конкретный заказ (`Order`), который, в свою очередь, имеет элементы (в данном случае они представлены денежной суммой — только по соображениям краткости).

Данная программа вычисляет общую скидку в денежном выражении для данного клиента, вызывая метод `GetOrderDiscount` для экземпляра `Customer`, который возвращает вычисленную сумму скидки. Затем это возвращенное значение выводится на консоль. В принципе, в коде этой программы была создана парочка экземпляров объектов, `cust` и `ord`, которым были переданы необходимые им данные, а также инструкции по их обработке. В результате работы, проделанной кодом этой программы, на консоль выводится скидка в денежном выражении, вычисленная для указанного клиента на основании сделанного им заказа.

Весь код, содержащийся в методе `Main`, находится на первом уровне иерархии вызовов. Методы и свойства, находящиеся в классах `Customer` и `Order`, находятся на втором уровне иерархии.

Внимательно изучив код класса `Order`, вы непременно обратите внимание на то, что там имеются свойство `Total` и метод `AddItem`.

Метод `AddItem` добавляет параметр `item` в свою коллекцию `orderItems`. Далее по коллекции `orderItems` производятся итерации с использованием свойства `Total` в качестве переменной цикла, при этом сначала вычисляется сумма возврата денег по всем заказанным элементам. Обратите внимание, что класс `Customer` имеет свойство `Discount`, которое содержит значение типа `decimal`, которое будет использоваться как процент скидки. Метод `GetOrderDiscount` из класса `Customer` ум-

ножает значение `Discount` на значение `Total` из класса `Order`, и возвращает сумму скидки, предоставляемой на этот заказ.

На данном этапе для вас очень важно тщательно изучить приведенный код и понять взаимоотношения и коммуникации между различными объектами. Обратите внимание, что каждый класс имеет четко определенную цель, причем его цель, как правило, четко взаимосвязана с его названием. Эта связь (смысловая нагрузка) помогает решить, какие данные и какие методы должен иметь класс. Например, класс `Order` имеет свойство `Total` и метод `AddItem`, а класс `Customer` имеет свойство `Discount` и метод `GetOrderDiscount`. Каждый объект поддерживает коммуникации с другими объектами, и их совместная работа служит для выполнения общей задачи. Например, класс `Customer` должен вычислять скидку, потому что именно этот класс имеет сведения о том, какой должна быть эта скидка (а информация об этом передана ему из метода `Main`). При этом класс `Customer` должен взаимодействовать с классом `Order`, потому что класс `Order` — это единственный объект, который имеет информацию о том, что именно заказано, в каком количестве, и как вычислять общую сумму заказа в денежном выражении.

Хотя я привел пример кода и продемонстрировал, как он работает, вам следует запустить программу самостоятельно и исследовать внутреннюю логику ее работы. В VS есть много различных средств, позволяющих визуализировать логику программы и выполнить ее отладку. Все эти средства будут подробнее рассмотрены в последующих разделах данной главы.

Инструменты кодирования, упрощающие разработку

Одним из новых инструментов VS 2010 является **Call Hierarchy** — это средство позволяет определить, какой код вызывает тот или иной метод и какие именно методы вызывает ваш код. Для начала, давайте обсудим, почему этот инструмент так важен, а затем проиллюстрируем его пользу на конкретном примере. Окно **Call Hierarchy** показано на рис. 6.1.

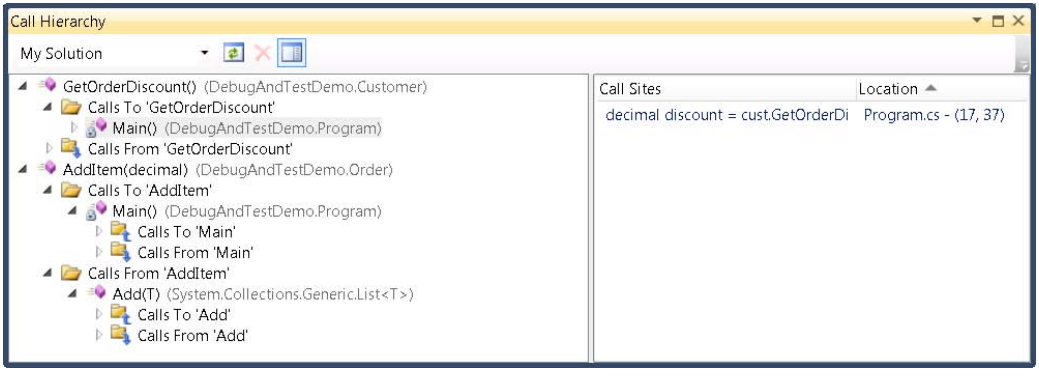


Рис. 6.1. Окно **Call Hierarchy**

Иерархия вызовов сообщает о вашем коде большое количество ценной информации, в том числе, степень повторного использования (degree of reuse), влияние изменений, а также потенциальную важность процедуры. Вызывающая точка (call site) представляет собой код, который инициирует другие члены класса. Например, в листинге 6.1, метод `Main` представляет собой точку вызова для метода `GetOrderDiscount`, который представляет собой вызываемый код.

С точки зрения повторяемости использования, большое количество точек вызова у метода может указывать на его относительную универсальность и возможности многократного использования. С другой стороны, небольшое количество вызовов на возможности повторного использования кода не указывает, а нулевое количество вызовов говорит о том, что код практически не используется и, вероятно, может быть даже удален.

Большое количество точек вызова может также говорить о том, что внесение изменений в данный метод окажет существенное влияние на работу программы. Просмотр точек вызова, из которых поступают запросы к методу, также может дать много полезной информации о том, куда и откуда передаются те или иные значения, и какие изменения могут потребоваться в вызывающих и вызываемых методах (иными словами, если некоторое изменение в вызывающем методе является необходимым, то какую цепочку изменений в вызываемых методах оно за собой повлечет).

Все сказанное демонстрирует важность и актуальность исследования иерархии вызовов. Теперь же давайте сосредоточимся на том, как она работает. Во-первых, важно понять, что иерархия вызовов является контекстно-чувствительной. Это значит, код, имеющий фокус ввода в окне редактора, определяет и вашу точку зрения. В рассматриваемом нами примере фокус имеет метод `GetOrderDiscount` из класса `Customer`, и нам требуется просмотреть, откуда вызывается метод `GetOrderDiscount` и какие фрагменты кода в его составе являются точками вызова (call sites). Чтобы исследовать иерархию вызовов, либо щелкните правой кнопкой мыши по методу `GetOrderDiscount` в окне редактора и из раскрывшегося контекстного меню выберите команду **View Call Hierarchy**, либо выделите метод `GetOrderDiscount` в окне редактора и нажмите клавиатурную комбинацию <CTRL>+<K>, <T>. VS отобразит окно **Call Hierarchy**, показанное на рис. 6.1.

Окно **Call Hierarchy**, показанное на рис. 6.1, показывает вызовы, поступающие к методу `GetOrderDiscount` (ветвь **Calls To**), и вызовы, исходящие из этого метода (ветвь **Calls From**). Ветвь **Calls To** представляет собой список точек вызова к методу `GetOrderDiscount`. Список **Calls From** — это список утверждений, входящих в состав метода `GetOrderDiscount`, из которых осуществляются вызовы к другим членам класса.

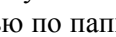
Раскрывающийся список в верхней части окна, показанного на рис. 6.1, в котором выбран элемент **My Solution**, показывает, как иерархия вызовов будет выполнять поиск, чтобы найти точки **Calls To** и **Calls From**. Для выбора доступны следующие опции: **My Solution**, **Current Project** и **Current Document** — их смысл самоочевиден.

Если в процессе работы над вашим кодом вам захочется обновить окно **Call Hierarchy**, нажмите кнопку **Refresh** (). Каждый раз, когда вы будете просматривать иерархию вызовов, выбранный элемент будет добавляться в список. Вы можете воспользоваться кнопкой **Remove Root** (), чтобы удалить элемент из списка. Кнопка **Toggle Details Pane** () позволяет отображать или, наоборот, скрывать панель **Details**, на которой показано расположение точки вызова и ее код. На рис. 6.1 выбран метод `Main`, в результате чего отображается вызов к методу `GetOrderDiscounts` из экземпляра `cust` класса `Customer` из листинга 6.1. Показана и строка кода, которая фактически осуществляет этот вызов. Вы можете выполнить двойной щелчок мышью, чтобы перейти в окно редактора кода как раз к этому утверждению. На практике, двойной щелчок мышью по любой из точек вызова в окне **Call Hierarchy** перемещает вас в окне редактора как раз к той строке кода, которая содержит этот вызов.

Окно **Call Hierarchy** показывает все возможные пути, которыми можно выйти к конкретной строке кода. Хотя эта возможность очень полезна, она ограничена статическим представлением вашего кода и не предоставляет возможности подробного просмотра кода работающей программы, которая может потребоваться для целей отладки. При отладке вам обычно требуется видеть рабочее состояние программы в конкретный момент ее исполнения. В последующих нескольких разделах вы познакомитесь с различными функциями отладчика, которые помогают исследовать поведение кода программы во время ее исполнения.

Конфигурирование отладочного режима

По умолчанию, VS создает проекты с активизированным отладочным режимом, в котором указаны настройки проекта, благодаря которым и становится возможной отладка вашего приложения. Панель инструментов VS показывает текущие конфигурационные настройки, которые вы используете в данный момент. В раскрывающемся списке присутствуют отладочная (**Debug**) и обычная (**Release**) конфигурации. Обычная конфигурация (Release) определяет настройки, которые ваша программа будет иметь, когда вы начнете ее распространение для фактического использования потребителями в производственных условиях. Кроме того, вы можете создать и индивидуально настроенную конфигурацию, в которой можно задать именно те свойства проекта, которые вам нужны. В данной главе мы будем использовать отладочную конфигурацию (Debug).

Чтобы изучить возможности, предоставляемые отладочной конфигурацией (Debug), убедитесь в том, что она выбрана в инструментальной панели (

). Чтобы сделать это, необходимо, чтобы у вас был открыт какой-нибудь проект. Затем выполните двойной щелчок мышью по папке **Properties** вашего проекта в окне **Solution Explorer** и перейдите на вкладку **Build**, как показано на рис. 6.2.

Как показано на рис. 6.2, оптимизация для вашего проекта отключена, и определены обе переменные компилятора — `TRACE` и `DEBUG`. Окно, показанное на рис. 6.2, отображает настройки для проекта на C#, а для VB показанная на этом рисунке вкладка будет называться **Compile**. Если включить оптимизацию, то компилятор

будет выполнять дополнительную обработку кода, в том числе, его структурные изменения. В результате такой оптимизации скомпилированный код будет компактнее по размерам и выполняться будет быстрее. При отладке оптимизация вам не нужна, потому что вы будете выполнять код пошагово и следить за выводом компилятора. Константы компилятора (также известные под названием директив компилятора — compiler directives), такие, как `TRACE` и `DEBUG`, используются компилятором для активизации или блокировки отдельных блоков кода. Например, пространство имен `System.Diagnostics` имеет класс `Debug`, который будет применяться только в том случае, если определена константа компилятора `DEBUG`.

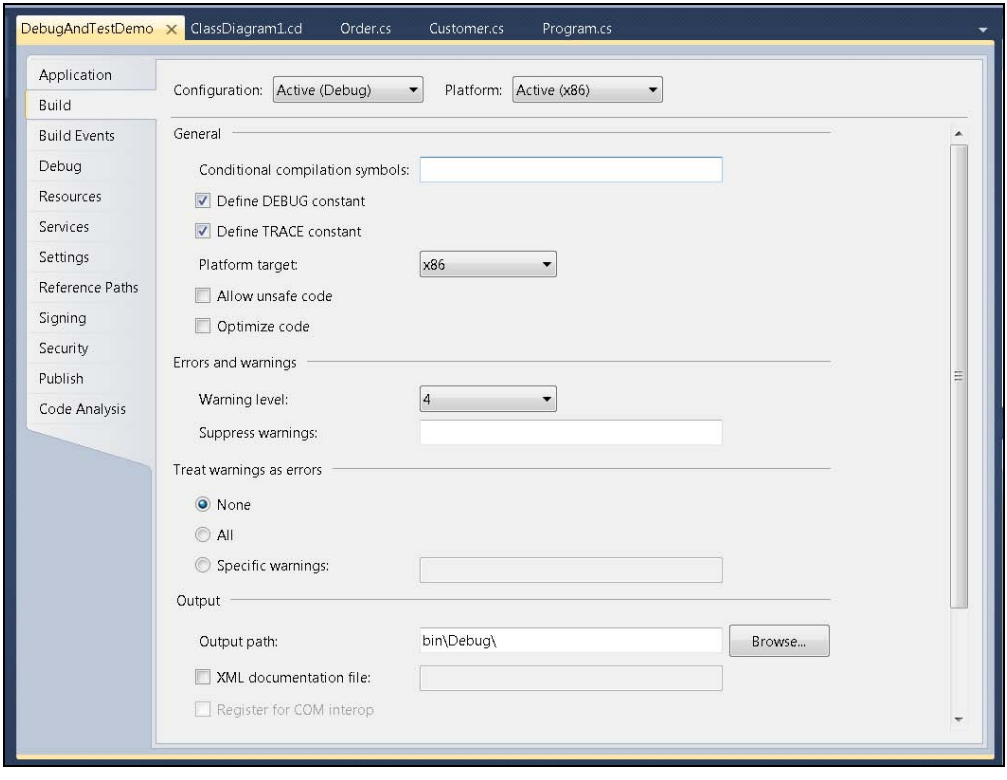


Рис. 6.2. Вкладка **Build** окна свойств проекта C# (для проектов VB она называется **Compile**)

Выполните построение вашего приложения, в результате чего будет получен ряд файлов, подходящих для использования в отладочных целях. Чтобы просмотреть эти файлы, выполните щелчок правой кнопкой мыши по имени решения, проекта или папки в окне **Solution Explorer** и выберите из контекстного меню команду **Open Folder in Windows Explorer**. Затем перейдите во вложенную папку bin\Debug, которая будет выглядеть примерно так, как показано на рис. 6.3.

На рис. 6.3 вы увидите четыре файла, два из которых представляют собой файлы приложения, а два предназначены для поддержки работы приложения под отладчиком. Файлы DebugAndTestDemo.exe представляет собой исполняемый файл

консольного приложения, появление которого вполне ожидаемо (в случае успешного построения). Файл *.pdb представляет собой файл с отладочной информацией (symbol file), который помогает синхронизировать идентификаторы в вашем коде с исполняемым файлом, упрощая тем самым пошаговое исполнение кода отладчиком VS.

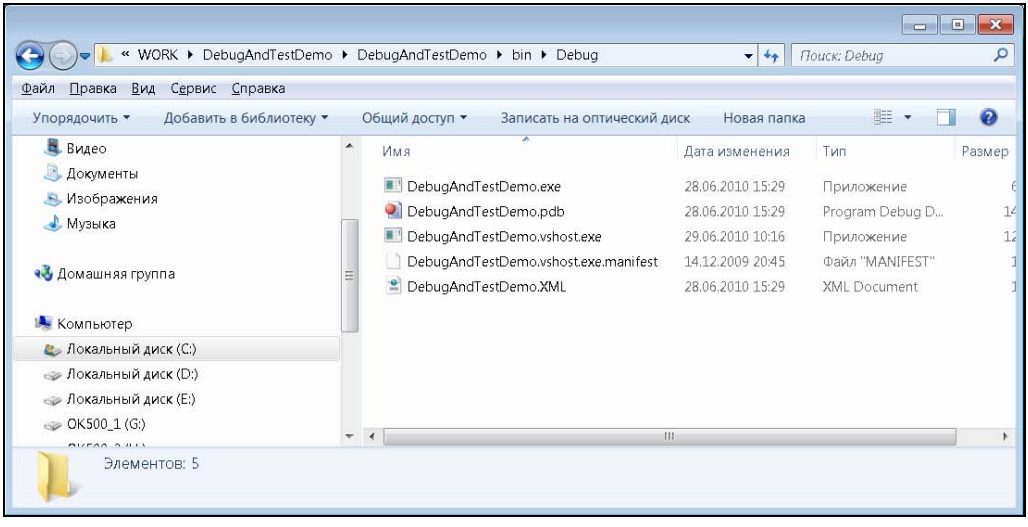


Рис. 6.3. Содержимое папки bin\Debug после построения проекта

Файлы, с составе имен которых присутствует строка vshost, представляют собой файлы, предназначенные для обеспечения процесса отладки. Файл *.vshost обеспечивает более быструю загрузку вашего приложения во время отладки, дает вам возможность тестирования вашего приложения с использованием различных конфигураций безопасности, а также позволяет оценивать выражения в ходе отладки. Файлы, имена которых содержат строку vshost, предназначены исключительно для отладочных целей, поэтому вы не должны распространять их с вашим приложением. Конечному пользователю они ни к чему и будут только занимать дисковое пространство. Обычно файлы со строкой vshost в составе имен нужны только для целей отладки с помощью VS. Существуют различные параметры настройки отладочного режима, которые вы можете конфигурировать в VS. Они влияют на ваш отладочный сеанс и модифицируют настройки конфигурационных файлов vshost. Чтобы задать настройку отладочного режима, откройте окно свойств проекта и перейдите на вкладку **Debug**, показанную на рис. 6.4.

В окне, показанном на рис. 6.4, опция **Configuration** установлена на **Active (Debug)**, а из списка **Platform** выбрана платформа **x86**. Из раскрывающегося списка Platform можно выбрать следующие опции: **Any CPU**, **x86**, **x64** или **Itanium** — в зависимости от типа процессорной платформы, для которой выполняется построение приложения. Компилятор будет выполнять оптимизацию кода для выбранного вами

типа CPU. Если вы запускаете VS под управлением 64-разрядной операционной системы, то в этом поле может отображаться опция **Active (Any CPU)**.

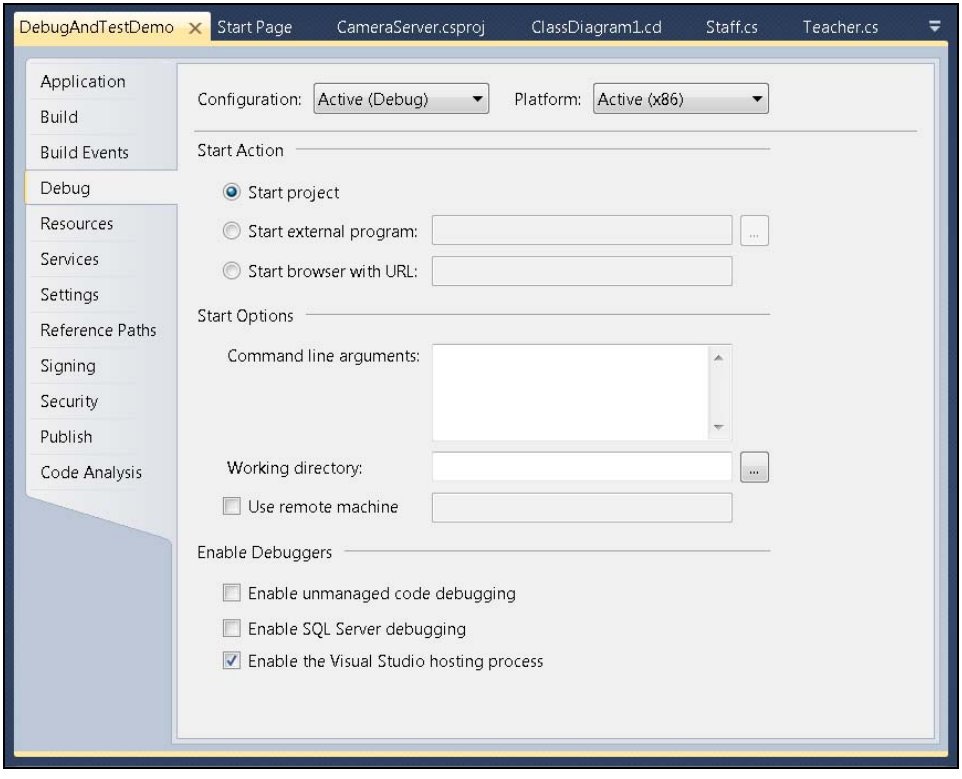


Рис. 6.4. Вкладка **Debug** окна свойств проекта

Группа опций **Start Action** на вкладке **Debug** указывает, каким образом начинается сеанс отладки. По умолчанию предлагается опция **Start Project**. Опция **Start External Program** позволяет вам подключать отладочный сеанс VS к уже работающему приложению, а опция **Start Browser With URL** дает возможность отладки Web-приложений. В общем случае, для отладки приложений, предназначенных для работы на настольных компьютерах, вам будет нужна только предлагаемая по умолчанию опция **Start Project**. Для Web-приложений, которые автоматически запускаются в браузере, страница свойств меняется.

В группе **Start Options** вы можете добавить в поле **Command line arguments** список аргументов командной строки, разделенных пробелами. Если вы пишете приложение, которое должно запускаться в сеансе командной строки или из командного сценария (скрипта), то эта возможность будет очень полезна для тестирования и отладки различных комбинаций аргументов командной строки. В этом случае вы сможете прочитать значения аргументов командной строки, которые вы ввели в текстовое поле **Command line arguments**, прочитав их из массива аргументов, переданных методу `Main`.

Рабочий каталог — это корневой каталог, из которого ваша программа читает файлы и куда программа выводит файлы, полученные в результате ее работы. По умолчанию, для отладочных конфигураций это будет каталог bin\Debug, а для рабочих конфигураций — каталог bin\Release. Вы можете изменить рабочий каталог, введя путь к нужному вам каталогу в поле **Working Directory**.

Флажок **Use Remote Machine** предназначен для более сложных сценариев отладки, при которых вы сможете отлаживать приложение, работающее на удаленном компьютере. Чтобы пользоваться этой возможностью, вам потребуется установить на удаленном компьютере специальное ПО для удаленной отладки. Затем вам необходимо будет убедиться в том, что в поле **Output path** вкладки **Build** окна свойств указан правильный путь к исполняемому файлу приложения, которое требуется отлаживать, что этот выводной каталог предоставлен в общий доступ, и что ваше приложение обладает всеми правами, необходимыми для доступа к этому разделяемому каталогу.

В этой книге основное внимание уделяется управляемому коду (managed code), который работает на .NET CLR. VS обладает возможностью отладки и неуправляемого кода (unmanaged code), например, кода, написанного на C++ и непосредственно взаимодействующего с операционной системой. В общем случае, вам следует оставлять флажок **Enable unmanaged code debugging** в сброшенном состоянии, за исключением тех ситуаций, когда вы пишете управляемый код, который взаимодействует с неуправляемым кодом, скажем, библиотекой COM DLL, и вам требуется возможность отладки как управляемого кода, так и неуправляемого. Кроме того, VS предоставляет вам возможность открывать хранимые процедуры SQL Server, устанавливать точки останова и пошагово выполнять код хранимой процедуры в отладочных целях. Если вы планируете отладку хранимых процедур, установите этот флажок.

Примечание

Под управляемым кодом (managed code)[1] понимается код, который работает на .NET Common Language Runtime (CLR). CLR — это виртуальная машина, которая предоставляет различные сервисы, в том числе — управление памятью (memory management), исполнение кода (code execution), сборку мусора (garbage collection)[2], безопасность (security) и т. п. Противоположность управляемому коду составляет так

[1] Управляемый код (managed code) — это код программы, исполняемый виртуальной машиной, например, Java или .NET, либо какой-либо другой. При этом обычный машинный код называется неуправляемым кодом (unmanaged code). Более подробно на эту тему см. следующие источники: **http://blogs.msdn.com/brada/archive/2004/01/09/48925.aspx**, **http://www.developer.com/net/cplus/article.php/2197621/Managed-Unmanaged-Native-What-Kind-of-Code-Is-This.htm**, **http://www2.research.att.com/~bs/bs_faq.html#CppCLI**, **http://www.rsdn.ru/forum/dotnet/3283182.flat.aspx**. — *Прим. перев.*

[2] В программировании "сборка мусора" (garbage collection) — это одна из форм автоматического управления памятью. Специальный код, называемый *сборщиком мусора* (*garbage collector*), периодически освобождает память, удаляя объекты, которые уже не будут востребованы приложением — то есть производит *сборку мусора*. Очень важная для программистов тема, см. следующие источники: **http://en.wikipedia.org/wiki/Garbage_collection_(computer_science)**, **http://www.hpl.hp.com/personal/Hans_Boehm/gc/**, **http://tinyurl.com/24revug**, **http://www.osnews.com/story/6864**, **http://www.cs.kent.ac.uk/people/staff/rej/gc.html**, **http://msdn.microsoft.com/ru-ru/library/w908yt2c(VS.90).aspx**. — *Прим. перев.*

называемый неуправляемый код (unmanaged code). Неуправляемый код не использует .NET CLR; вместо этого такой код работает на компьютере непосредственно и напрямую взаимодействует с операционной системой. Если вы работаете с неуправляемым кодом, вы должны сами заботиться об управлении памяти и писать низкоуровневый код, реализующий все те сервисы, которые CLR обычно предоставляет вам автоматически. VS дает возможность написания неуправляемого кода на C++, но, поскольку в этой книге основное внимание уделяется программированию на C# и VB, которые создают исполняемые файлы, запускающие управляемый код на CLR, мы сосредоточимся именно на управляемом коде.

Флажок **Enable the Visual Studio hosting process** представляет собой ту самую настройку, установка которой приводит к созданию файлов vshost в выводном каталоге. При обычных условиях этот флажок рекомендуется оставить во взведенном состоянии, из-за описанных чуть ранее преимуществ, предоставляемых файлами vshost. Единственным исключением может быть ситуация, в которой сервисы, предоставляемые файлами vshost, конфликтуют с кодом, который вы исполняете. Но это — сценарий очень сложный и крайне редкий.

Совет

В ранних версиях VS иногда возникали ситуации, в которых выводилось сообщение об ошибках прав доступа к файлам vshost. Эти ошибки вызывались установкой блокировок на файл. Это может случиться, если вы подключились к работающему процессу из другого экземпляра VS или если процесс завершился ненадлежащим образом, причем последовательность останова была такова, что блокировки с файлов vshost не были сняты. Одним из вариантов обхода такой ситуации является сброс флажка **Enable the Visual Studio hosting process**. После этого требуется перестроить проект, вновь установить флажок **Enable the Visual Studio hosting process**, а затем снова выполнить построение. Как вариант, можно перезагрузить операционную систему. Выбор в данном случае зависит исключительно от вас — что для вас проще. Эта ситуация не говорит о дефектах VS или вашей операционной системы, потому что когда приложение работает, файловые блокировки необходимы. Скорее, этот сценарий указывает на наличие ошибок в вашем коде или на некорректное закрытие приложения.

В дополнение к настройкам отладочного режима в окне свойств, большое количество опций отладки будет вам доступно и в окне **Options**. Чтобы просмотреть эти опции, выберите из меню команды **Tools | Options** и в левой панели раскрывшегося окна выберите узел **Debugging** (рис. 6.5).

Как видно из рис. 6.5, существует большое количество опций, позволяющих вам конфигурировать процесс отладки. Основное отличие этих опций от опций, специфичных для проекта, заключается в том, что опции проекта действуют только применительно к этому проекту, в то время как опции отладки, задаваемые в окне **Options**, позволяют вам задать опции для всех проектов. В тех случаях, когда это возможно, опции отладки, задаваемые в окне **Options**, будут предлагаться по умолчанию и для всех вновь создаваемых проектов. Следовательно, если вы хотите, чтобы некоторые опции по умолчанию предлагались для всех ваших проектов, в первую очередь установите их в окне **Options**. Опции отладочного режима, предоставляемые в окне **Options** (см. рис. 6.5), слишком многочисленны, многие из них относятся к сложным сценариям и выходят за рамки вопросов, обсуждаемых в этой книге. Поэтому перечислить их все не представляется возможным. Если когда-

нибудь у вас возникнет вопрос о том, доступна ли в VS та или иная возможность, рекомендуется просто раскрыть окно **Options** и попробовать ее найти.

Теперь, когда ваша система сконфигурирована для отладки, можно приступать к установке точек останова и собственно отладке приложений.

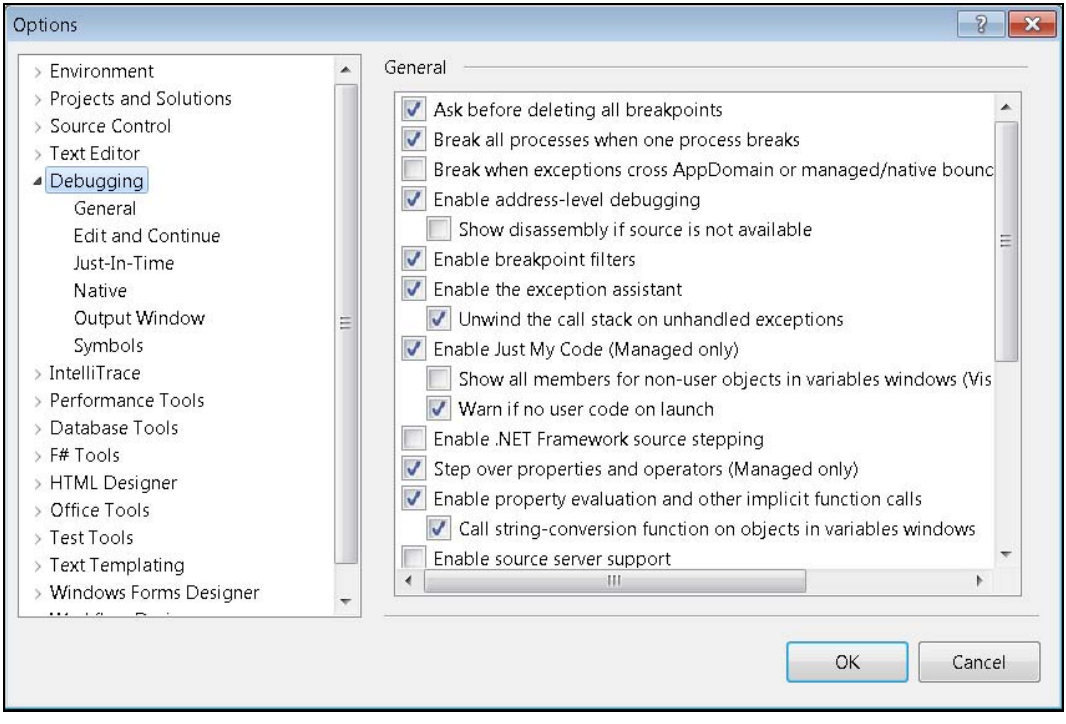


Рис. 6.5. Отладочные опции в окне **Options**

Установка точек останова

Точки останова (breakpoints) представляют собой такие утверждения в составе вашего кода, на которых программа должна автоматически делать паузу, по аналогии с тем, как останавливается воспроизведение музыки или фильма, когда вы нажимаете кнопку PAUSE на вашем медиаплеере или аппаратном проигрывателе. Когда программа достигает установленной вами точки останова, ее исполнение приостанавливается, и вы получаете возможность выполнения отладочных задач. К числу отладочных задач относятся просмотр значений переменных в данном "замороженном" состоянии программы, оценка выражений или редактирование кода с последующим продолжением исполнения. В последующих нескольких разделах будет показано, как следует создавать точки останова в вашем приложении и как нужно ими управлять.

Создание точки останова

Чтобы создать точку останова, необходимо открыть один из проектов, а затем открыть в окне редактора кода один из файлов, содержащих исходный код программы. Как уже говорилось ранее, в примерах, которые будут приводиться в этой главе, мы будем использовать код тестового приложения, приведенный в листингах 6.1 и 6.2. В окне редактора кода VS на левой границе окна редактора кода имеется широкая полоса, чуть подсвеченная серым цветом. Если выполнить щелчок мышью на этой границе, VS установит точку останова на соответствующем утверждении. Можно также щелкнуть мышью по нужному вам утверждению, что передаст ему фокус ввода, а затем нажать клавишу <F9>. В результате этих действий тоже будет создана точка останова. При этом на границе появится большая красная точка, а само утверждение, на которое установлена точка останова, будет выделено красной подсветкой, как показано на рис. 6.6. Обратите внимание, что точку останова можно установить только на код, который действительно исполняется в ходе работы программы. Если вы выберете строку, которая не исполняется (примером такой строки является, например, определение пространства имен), то точка останова установлена не будет, а в нижней части экрана появится сообщение о том, что в данной позиции точку останова создать невозможно ("A breakpoint could not be inserted at this location").

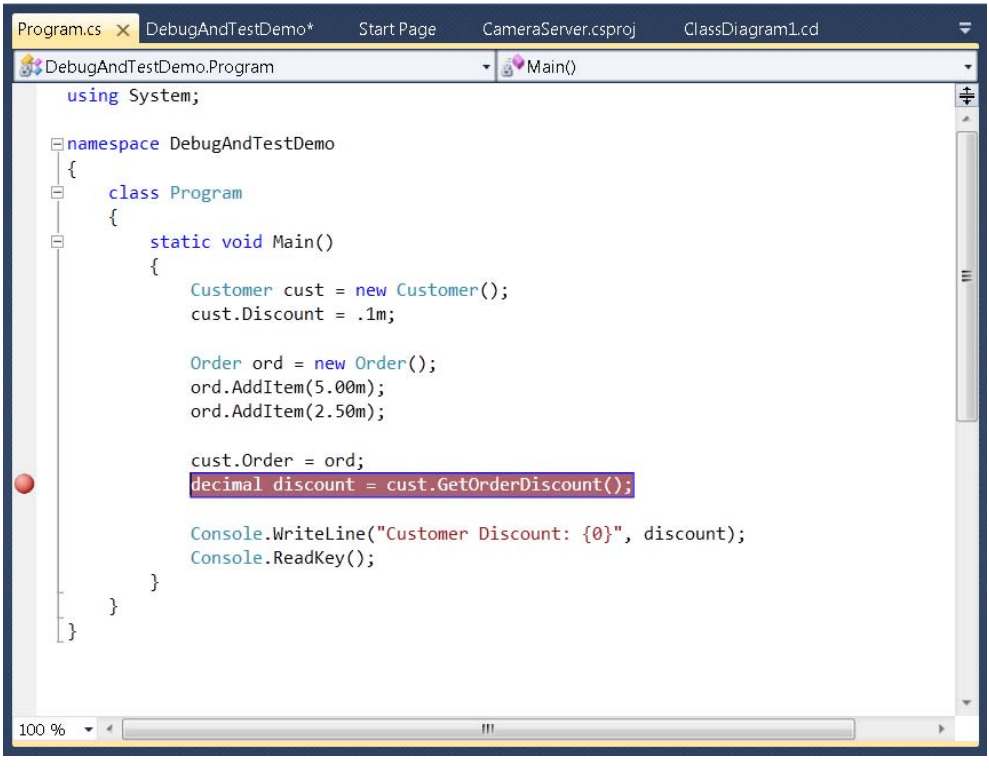


Рис. 6.6. Точка останова

Чтобы убедиться в том, что VS сделает паузу, встретив точку останова, запустите приложение в режиме отладки. Чтобы начать исполнение программы в отладочном режиме, можно выбрать из меню команды **Debug | Start Debugging**. Как вариант, можно нажать клавишу <F5> или же щелкнуть кнопку инструментальной панели **Start With Debugging** с изображением небольшой треугольной зеленой стрелки, направленной вправо (). Точка останова на рис. 6.6 установлена на вызов метода `GetOrderDiscount` в методе `Main`. Когда программа встречает точку останова, соответствующая строка кода подсвечивается желтым цветом, а в центре красной точки, обозначающей точку останова, появляется желтая стрелка. Щелчок мышью по кнопке **Continue** (это та же самая кнопка, которая использовалась для запуска программы в режиме отладки) или нажатие клавиши <F5> приведут к тому, что VS продолжит исполнение кода программы. Остановить процесс отладки тоже можно несколькими различными способами. Во-первых, можно выбрать из меню команды **Debug | Stop Debugging**, во-вторых, можно нажать клавиатурную комбинацию <SHIFT>+<F5> и, наконец, можно щелкнуть мышью по кнопке инструментальной панели **Stop Debugging** с изображением небольшого синего квадрата ().

Совет

Если вы пишете программу, которая выполняет большой объем работы или, наоборот, выполняет очень маленький объем работы, но зависает в ненамеренно вами созданном бесконечном цикле, вы можете прервать исполнение программы нажатием кнопки **Break All** (это кнопка с двумя вертикальными синими прямоугольниками, расположенная левее кнопки **Stop Debugging**). Как вариант, можно нажать клавиатурную комбинацию <CTRL>+<ALT>+<BREAK>. Когда вы сделаете это, ваша программа прекратит выполнение на той строке кода, которую она выполняла в момент, когда вы нажали кнопку паузы. Впоследствии вы сможете возобновить исполнение программы с этой точки. Данная кнопка работает по аналогии с кнопкой PAUSE на персональных проигрывателях (программных или аппаратных).

Индивидуальная настройка точки останова

В предыдущем разделе были изложены базовые сведения о точках останова. Кроме того, было показано, как установить точку останова на нужной строке кода. Однако это далеко не все, что необходимо знать о точках останова. Например, их можно индивидуально настроить таким образом, чтобы программа останавливалась на ней в зависимости от определенных критериев — например, от количества раз, когда данная точка останова была встречена (hit count), от выполнения или невыполнения конкретного условия и т. д. Чтобы просмотреть, какие опции вам доступны, наведите курсор на одну из точек останова, щелкните правой кнопкой мыши, и вы увидите полный набор доступных опций в появившемся контекстном меню. Опции этого контекстного меню вкратце описаны в табл. 6.1.

Таблица 6.1. Опции, доступные в контекстном меню точки останова

Опция	Описание
Delete Breakpoint	Удаляет точку останова
Disable/Enable Breakpoint	Если вы не хотите удалять точку останова, потому что впоследствии вы планируете использовать ее повторно, вы можете блокировать ее, а затем вновь активизировать, когда она вам снова понадобится
Location	Это — тот вид точек останова, когда выполняется щелчок на границе окна редактора кода. Вы можете изменить параметры этой точки останова, выбрав из контекстного меню команду Location и отредактировав параметры в появившемся диалоговом окне
Condition	Позволяет ввести выражение, в зависимости от значения которого программа будет останавливаться, встретив данную точку останова. Программа будет останавливаться, либо если выражение получает значение `TRUE`, либо если изменяется значение переменной. Выражение должно быть основано на переменных, встречающихся в вашем коде
Hit Count	Приводит к тому, что начнет останавливаться на этой строке после того, как она будет исполнена заданное количество раз, либо когда счетчик представляет собой кратное некоторого числа (т. е. каждый n-й раз), либо когда счетчик достигнет или превысит некоторое значение
Filter	Точка останова будет срабатывать (т. е. исполнение программы будет останавливаться) только в том случае, когда встретится указанная вами комбинация машины (machine), процесса (process) или нити исполнения (thread)
When Hit	Задает точку трассировки (tracepoint), которая выводит сообщение в окно вывода. Сообщение можно сконфигурировать таким образом, чтобы в его состав были включены различные системные значения, например, функции (function), нити управления (thread) и т. д. Вы можете просматривать сообщения в окне **Output**. Чтобы воспользоваться этой возможностью, выберите из меню команды **View** | **Output Window** или нажмите клавиатурную комбинацию <CTRL>+<ALT>+<O>. В момент срабатывания точки останова вы можете запускать макрос
Edit Labels	Вы можете ассоциировать точки останова с метками, чтобы организовать их по группам
Export	Эта опция позволяет экспортировать точки останова во внешний файл формата XML

Вы можете установить функциональную точку останова, щелкнув по методу, чтобы его выделить, а затем выбрав из меню команды **Debug** | **New Breakpoint** | **Break At Function** или нажав клавиатурную комбинацию <CTRL>+<D>, <N>.

Управление точками останова

По мере работы над кодом вашей программы, в нем появляется все больше и больше точек останова, разбросанных по всему вашему проекту. Вы можете управлять всеми точками останова централизованно из единого окна. Для этого

выберите из меню команды **Debug** | **Windows** | **Breakpoints**, в результате чего появится окно, показанное на рис. 6.7.

Рис. 6.7. Окно **Breakpoints**

Большая часть функций, доступных в окне **Breakpoints**, уже пояснена, за исключением того факта, что опции инструментальной панели действуют применительно ко всем точкам останова, которые установлены и активизированы на текущий момент. Щелчок мышью по заголовку столбца приводит к сортировке его содержимого. Поле поиска **Search** помогает фильтровать точки останова, а список **Columns** позволяет расставить акценты на тех полях, по которым должен производиться поиск. Кроме того, на инструментальной панели окна **Breakpoints** имеются кнопки **Export** ( ) и **Import** ( ), которые позволяют вам экспортировать точки останова во внешний XML-файл и, соответственно импортировать их из внешнего XML-файла. Двойной щелчок мышью по любой точке останова из числа имеющихся в окне **Breakpoints** перемещает вас к строке кода в окне редактора кода, на которую установлена данная точка останова. Щелчок правой кнопкой мыши по точке останова отображает контекстное меню, содержащее опции, уже обсуждавшиеся ранее в этом разделе.

После того как вы установите точки останова, вы можете пошагово выполнять код программы, изучая последовательность ее выполнения. Обсуждению этой темы посвящен следующий раздел.

Пошаговое выполнение кода

Пошаговое выполнение кода представляет собой процесс исполнения одной или нескольких строк кода, протекающий под контролем программиста, выполняющего отладку. На самом мелко структурированном уровне, вы можете выполнять этот процесс не просто пошагово, а строку за строкой. Хотя такой режим часто бывает необходим для выяснения того, где же коренится ошибка, не менее часто он бывает слишком трудоемок и неуклюж. Поэтому и были предусмотрены методы "прошагивания" через несколько строк кода, за счет их выполнения за один прием.

Чтобы начать пошаговое выполнение кода, откройте проект, установите точку останова и запустите программу на исполнение в отладочном режиме. Программа

запустится, и ее выполнение продолжится до тех пор, пока не встретится точка останова. В этот момент времени исполнение программы будет прервано, и вы сможете выполнить различные отладочные операции: **Step**, **Step over** и **Step out**. Доступные отладочные операции вкратце перечислены в табл. 6.2. Объяснения, предложенные в этой таблице, даны в предположении того, что в ходе исполнения программы встретилась точка останова, в данный момент ее исполнение прервано, и дальнейший процесс будет зависеть от того, какую из операций вы предпримете.

Таблица 6.2. *Шаговые операции*

Операция	Описание
Step Over	Выполняет код текущей строки и перемещается к следующей строке кода, после чего исполнение программы снова останавливается, и отладчик ожидает ваших дальнейших инструкций. Чтобы выполнить эту операцию, выберите из меню команды **Debug** | **Step Over**. Альтернативные варианты выполнения этой операции — нажатие клавиши <F10> или кнопки **Step Over** () на инструментальной панели. Кроме того, можно выполнить щелчок правой кнопкой мыши и выбрать соответствующую опцию из контекстного меню. Большинство разработчиков Visual Studio быстро запоминают "горячую клавишу" <F10> и отдают предпочтение именно ей
Step Into Specific	Если текущая строка выполняет вызов метода, то команда **Step Into** передаст управление первой строке вызываемого метода, и исполнение программы приостановится там. Чтобы выполнить эту операцию, выберите из меню команды **Debug** | **Step Into**. Альтернативные варианты заключаются в нажатии клавиши <F11> или в щелчке мышью по кнопке **Step Into** на инструментальной панели (). Практика показывает, что клавиша <F11> предоставляет быстрейший вариант выполнения этой операции
Step Out	Если вы находитесь внутри метода, вы можете вернуться обратно к вызывающему коду, выполнив операцию **Step Out**. Чтобы выполнить операцию **Step Out**, либо выберите из меню команды **Debug** | **Step Out**, либо нажмите клавиатурную комбинацию <SHIFT>+<F11>, либо щелкните мышью по кнопке **Step Out** на инструментальной панели (). Обратите внимание, что внутри функции не пропускается ни одной строки кода; они выполняются в точном соответствии с логикой вашей программы. Исполнение вашей программы автоматически приостановится на строке кода, следующей за возвратом из этой функции
Run to Cursor	Иногда вам может потребоваться выполнить блок кода и остановиться на определенной строке. На этой строке можно установить другую точку останова и продолжать выполнение программы до тех пор, пока не будет достигнута новая точка останова. Однако если вы не хотите сохранять новую точку останова на той строке, где хотите сделать паузу в выполнении программы, можно щелкнуть правой кнопкой мыши по этой строке и выбрать из появившегося контекстного меню команду **Run To Cursor**. Опять же, здесь не пропускается ни одна из строк кода, программа только сделает паузу, когда ее исполнение достигнет строки, на которую вы поместили курсор. Как вариант, вы можете выделить строку, на которой хотите сделать паузу, и нажать клавиатурную комбинацию <CTRL>+<F10>. Это особенно полезно, если у вас возникло подозрение, что выполняются все итерации цикла

Таблица 6.2 *(окончание)*

Операция	Описание
Set Next Statement	Вы можете пропускать несколько строк кода, "перепрыгивая" через них как вперед, так и назад, без выполнения пропущенного кода. Например, может оказаться полезным пропустить метод, чтобы выяснить, что в действительности вы хотели войти в него. Это позволяет вернуться к строкам кода, начиная с которых вам хочется прослеживать логику работы приложения, ведь если перезапуск приложения в действительности не требуется, то вам и не захочется его выполнять. Чтобы вернуться к строке кода, откуда вы хотите перейти к вызову метода, выделите желтую стрелку на границе и переместите ее назад к вызову метода, после чего вы сможете выполнить команду **Step Into**. Как вариант, если у вас есть одно или несколько утверждений, которые вы не хотите исполнять, перетащите желтую стрелку на границе к утверждению, следующему сразу же за пропускаемым кодом, и продолжайте пошаговый отладочный сеанс. Этот подход особенно удобен, если вы используете функцию **Edit and Continue**, которая позволяет редактировать код программы "на лету", экспериментировать с различными идеями по программированию и мгновенно перевыполнять эти строки кода. Обратите внимание, что VS не сбрасывает значения переменных к исходному состоянию, поэтому, чтобы получить ожидаемые результаты, вам необходимо сбросить эти значения вручную

Операция **Step Over** выполняет код текущей строки и переходит к следующей. Выполнить операцию **Step Over** можно, выбрав из меню команды **Debug** | **Step Over**, нажав на клавиатуре клавишу <F10> или щелкнув мышью по кнопке Step Over () на инструментальной панели.

Итак, вы научились выполнять пошаговую отладку кода, что чрезвычайно полезно. Однако не менее важно иметь возможность просмотра значений переменных и наблюдать за их изменением. Именно этот навык вы приобретете после прочтения следующего раздела.

Исследование состояния приложения

Состояние или статус приложения (application state) — это значения всех переменных в вашем коде, текущий путь исполнения, а также любая информация, сообщающая вам о том, что делает ваша программа. Возможность просмотра информации о состоянии программы во время отладки очень важна, потому что она позволяет выявить различия между тем, что программа делает в действительности, и тем, что вы от нее ожидали. В VS есть множество различных окон, позволяющих просматривать информацию о состоянии приложения, в данном разделе мы рассмотрим их все.

Примечание

При исследовании статуса вашего приложения, вам потребуется учитывать концепцию диапазона (scope). Когда переменная находится в пределах диапазона, вы сможете просмотреть ее значение. Диапазон определяется в пределах блока. В C# блоки определяются фигурными скобками, а в VB — утверждениями `begin` и `end`. Приме-

рами диапазона являются поля класса и локальные переменные. Поле частного класса будет находиться в пределах диапазона для всех методов этого класса, но не в другом классе. Локальная переменная будет находиться в пределах диапазона для всех методов, в которых она определена, но будет находиться вне диапазона для других методов. Еще один пример диапазона — переменная цикла, которая будет находиться в пределах диапазона в теле цикла, но за его пределами — вне поля цикла.

Окна *Locals* и *Autos*

Окна **Locals** и **Autos** показывают значения переменных в вашей системе на момент текущей точки останова. Окно **Locals** содержит список всех переменных, к которым может получать доступ текущее утверждение (эти переменные считаются находящимися в пределах диапазона). Окно **Autos** показывает переменные из текущей и предыдущей строк кода. Вы можете открыть окна **Locals** и **Autos**, выбрав из меню команды **Debug** | **Windows**, при активном сеансе отладки VS в тот момент, когда исполнение программы приостанавливается при достижении точки останова. Эти окна могут уже присутствовать на экране в нижнем левом углу следом за окном Output, если вы не выполняли перенастройки среды разработчика Visual Studio.

Как показано на рис. 6.8, окно **Locals** показывает все переменные в диапазоне видимости метода `Main` из листинга 6.1.

Окно **Locals** отображает переменные в крупномодульном представлении, и список может быть довольно длинным, в зависимости от того, какое количество переменных находится в диапазоне видимости. Окно **Locals** удобно использовать для того, чтобы выяснить, какие переменные находятся под влиянием текущего алгоритма. На рис. 6.9 показано окно **Autos.**

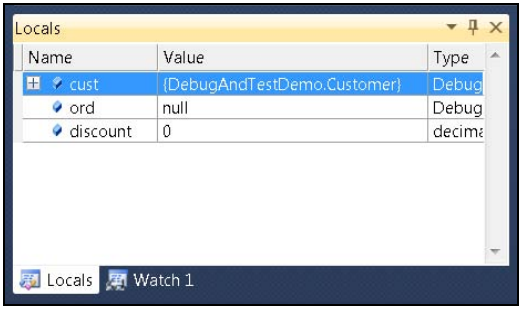


Рис. 6.8. Окно `Locals`

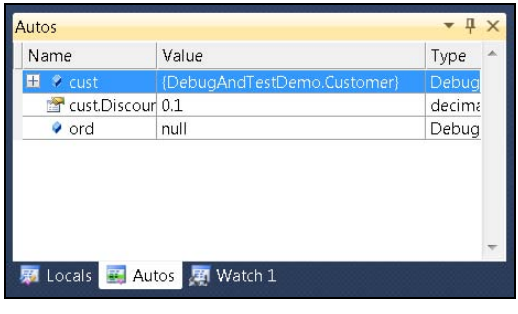


Рис. 6.9. Окно **Autos**

Обратите внимание на то, что окно **Autos** отображает данные в более мелкомодульном представлении, и это касается как переменных, так и свойств объектов из текущей и предшествующей строк кода. Окно **Autos** удобно использовать для более целенаправленного выяснения событий, на данный момент происходящих в вашем коде.

Окно *Watch*

Окно **Watch** позволяет создать индивидуальный список переменных, за которыми вам требуется вести наблюдение. Вы можете перетаскивать переменные мышью из окна редактора кода или вручную вводить имена переменных в окно **Watch**. Выберите из меню команды **Debug** | **Windows** | **Watch**, и вы увидите список из четырех окон **Watch**, в которых вы можете создать четыре различных набора данных для просмотра по одному набору за раз. На рис. 6.10 показано окно **Watch** с одной переменной.

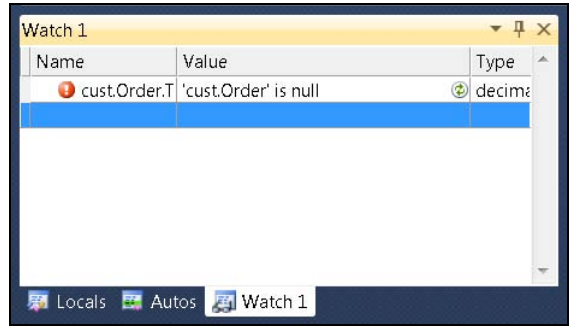


Рис. 6.10. Окно **Watch**

Окна **Locals** и **Autos** могут оказаться загроможденными слишком большим количеством переменных и будут замедлять вашу работу по мере усложнения вашего кода, особенно когда интересующие вас переменные находятся в самом конце списка или так далеко друг от друга, что для их просмотра вам придется прокручивать список. Вот тут вам и придет на помощью окно **Watch** — с его помощью вы сможете быстро добраться до объекта без необходимости постоянного разворачивания иерархического дерева. Например, вы можете ввести в окно строку `cust.Order.Total`, как показано на рис. 6.10, чтобы просмотреть результирующее значение свойства `Totals` свойства `Order` экземпляра `cust`. Кроме того, в этом окне можно редактировать значения ваших переменных и свойств. Для этого в данном окне следует либо выполнить двойной щелчок мышью по текущему значению в столбце **Value** или выполнить щелчок правой кнопкой мыши по имени переменной и выбрать из контекстного меню команду **Edit**. Когда значение будет изменено, цвет, которым оно отображается, сменится с черного на красный, чтобы обратить ваше внимание на изменение. Данный способ редактирования значений "на лету" очень удобен, особенно если вы хотите повторно выполнить предыдущие строки кода с новым значением, не прерывая сеанса отладки и не перезапуская программу. Данные методы работы позволяют добиться существенной экономии времени.

Окно *Immediate*

При отладке часто бывает полезно оценивать значение выражений на текущий момент. Окно **Immediate** позволяет вводить имена переменных и большого количества выражений различных типов. Чтобы раскрыть окно **Immediate**, выберите из меню команды **Debug** | **Windows**. Кроме того, в зависимости от опций настройки VS, это окно уже может быть открыто в правом нижнем углу экрана отладки. Окно **Immediate** показано на рис. 6.11.

В примере, представленном на рис. 6.11, в окне **Immediate** показаны три выражения, демонстрирующих доступные вам возможности по чтению свойств, исполнению методов и оценке выражений. В данном примере эти выражения были введены вручную, и вы можете делать то же самое, вводя практически любой код, который вам требуется.

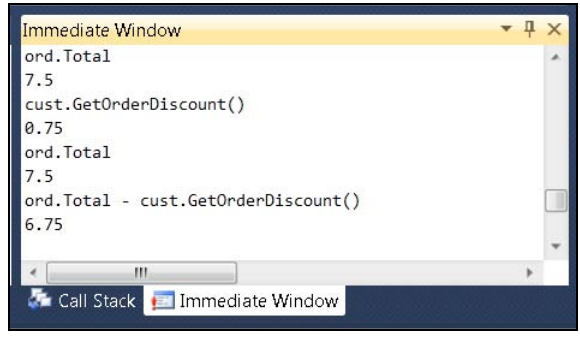


Рис. 6.11. Окно **Immediate**

Примечание

Пример, показанный на рис. 6.11, относится к C#. При оценке выражений на VB, предваряйте утверждения символом вопросительного знака (?).

Окно *Call Stack*

Как уже говорилось ранее в этой главе при обсуждении инструментария, применяющегося во время разработки, в окне **Call Hierarchy** вам предоставляется возможность просмотра кода, вызывающего ваши методы вызовов во время разработки. Соответственно, вы имеете аналогичную возможность просмотра пути исполнения кода во время исполнения программы — для этого служит окно **Call Stack**. В ходе отладочного сеанса вы найдете окно **Call Stack** в правом нижнем углу VS на вкладке, расположенной рядом с окном **Immediate**, если только вы не изменили изначальную настройку среды VS. В противном случае, вы можете открыть это окно, выбрав из меню команды **Debug** | **Windows** | **Call Stack**. В окне **Call Stack** вы можете просмотреть текущий путь выполнения вашего приложения от метода `Main` до текущей строки кода. На рис. 6.12 показано окно **Call Stack**. Чтобы понять, почему оно называется **Call Stack**, обратите внимание на то, что каждый следующий метод как бы "помещается в стек" поверх предыдущего. При этом текущий метод находится на вершине стека, а точка входа — в самом низу, а последующие вызовы — между ними. Этот "стек вызовов" можно сравнить, например, со стопкой тарелок, где последняя тарелка находится наверху.

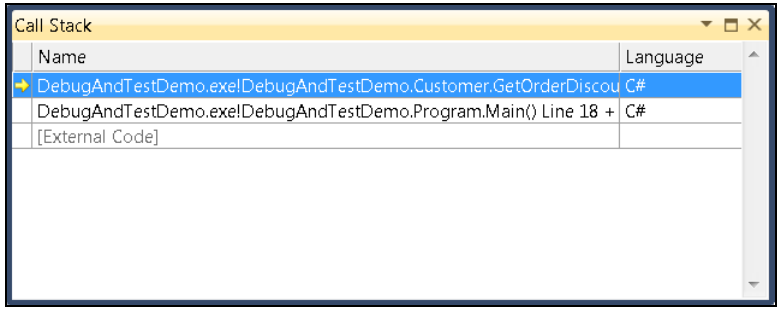


Рис. 6.12. Окно **Call Stack**

В окне **Call Stack**, показанном на рис. 6.12, видно, что в процессе отладки произведен вход в метод `GetOrderDiscount`. Двойной щелчок мышью по другому методу в окне **Call Stack** перемещает вас к точке вызова, из которой вызывался данных метод. Это — очень важный и очень мощный инструмент, потому что он позволяет вам перейти к вызывающему коду и исследовать состояние приложения в точке вызова, предоставляя ценную визуальную информацию о том, какие вычисления производились перед вызовом текущего метода и как они были сформулированы.

Окно *QuickWatch*

Окно **Quick Watch** дает возможность быстрого просмотра выражения. При вводе выражения оно предлагает поддержку Intellisense, позволяя повторно выполнить оценку выражения и добавить выражение в окно **Watch**. Чтобы открыть окно **Quick Watch**, выберите из меню команды **Debug** | **Quick Watch** или нажмите клавиатурную комбинацию <CTRL>+<D>, <Q>. Если вы выберете выражение в редакторе кода, то окно **QuickWatch** отобразит это выражение. На рис. 6.13 показано типичное окно **QuickWatch** и его использование для оценки выражения.

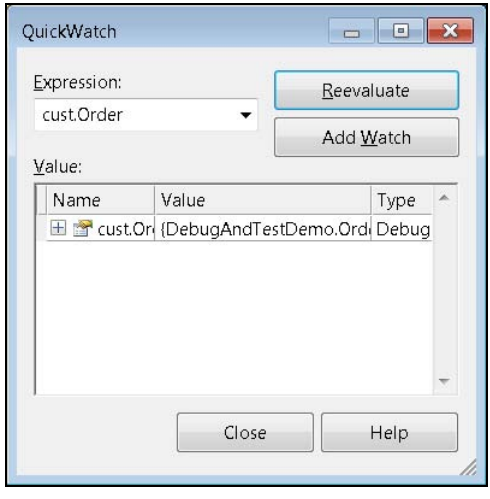


Рис. 6.13. Окно **QuickWatch**

Если щелкнуть мышью по кнопке **Reevaluate**, показанной на рис. 6.13, то результаты вычисления появятся в столбце **Value**. Столбец **Value** будет содержать только текущее выражение. Если вы хотите сохранить выражение, нажмите кнопку **Add Watch**, в результате чего выражение будет загружено в окно **Watch**. Необходимо иметь в виду, что если вы закроете окно **Watch**, ваше выражение будет удалено, однако оно будет присутствовать в списке истории, из которого вы впоследствии сможете снова его выбрать.

Наблюдение переменных с привязкой к источнику

В процессе отладки, вы можете задержать курсор над любой переменной, чтобы увидеть ее значение. Но, как только вы отведете курсор в сторону, исчезнет и всплывающее окно подсказки, отображающее значение переменной. Функция **Pin To Source** идет на шаг дальше, позволяя выводить значение переменной на постоянной основе. Чтобы использовать функцию **Pin To Source**, щелкните правой кнопкой мыши по переменной и выберите опцию **Pin To Source**. В качестве альтернативного варианта, вы можете задержать курсор над переменной в отладчике

и щелкнуть мышью по значку с изображением кнопки, который появится вместе со значением переменной в окне всплывающей подсказки. Результат привязки к источнику с помощью функции **Pin To Source** показан на рис. 6.14.

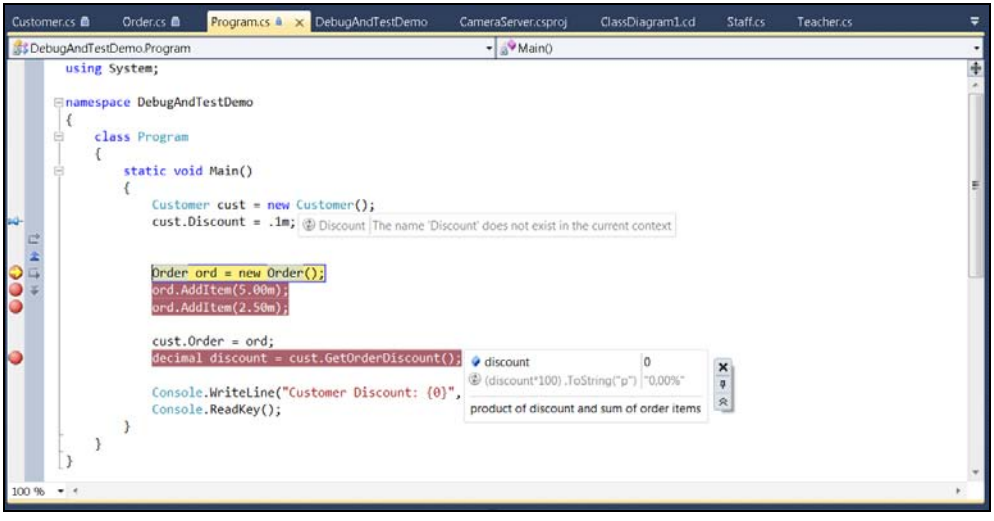


Рис. 6.14. Результат привязки значения переменной к источнику

После того как вы выполните привязку значения переменной к источнику, вы сможете продолжать отладку и выполнить прокрутку назад к переменной, чтобы прочесть ее текущее значение. В дополнение к просмотру значения, вы можете добавить комментарий. Для этого щелкните мышью по значку "шеврона", который появится, когда вы задержите курсор над "привязанным" значением. В данном примере "привязанное" значение имеет комментарий "product of discount and sum of order items".

VS обнаружит "привязанное" значение после строки, но вы можете не увидеть значения, если оно встречается в длинной строке, которая превышает ширину вашего экрана. К счастью, вы можете щелкнуть мышью по "привязанному" значению и перетащить его в то место экрана, где вы всегда сможете его видеть. Чтобы избежать путаницы, не забывайте все-таки размещать такие "привязанные" значения достаточно близко к той переменной, для которой они отображаются.

Щелкните правой кнопкой мыши по "привязанному" значению, чтобы отобразить контекстно-чувствительное меню с опциями **Edit Value** | **Hexadecimal Display** | **Add/Remove Expression**. На рис. 6.14 показано добавленное выражение `(discount * 100) .ToString("p")`, чтобы отобразить значение, выраженное через проценты. Добавление выражений позволяет сделать выражение более удобным для чтения и восприятия. Кроме того, можно добавлять другие выражения, связанные с данным, чтобы иметь возможность мгновенного просмотра и понять, как данное значение влияет на другие расчетные результаты.

Закрыть "привязанное" значение можно, задержав над ним курсор и щелкнув мышью по кнопке **Close** (X).

Работа с IntelliTrace

Окно **IntelliTrace** отображает список всех изменений, которые претерпело приложение в ходе отладочного сеанса. По мере того как вы пошагово исполняете код, в окне **IntelliTrace** отображается каждый шаг вашего отладочного сеанса. С помощью инструментальной панели окна **IntelliTrace** вы можете установить режим представления информации — **Diagnostic Events** (диагностические события) или **Call View** (просмотр вызовов). Режим просмотра диагностических событий позволяет фильтровать события по категориям (**Category**) и нитям управления (**Thread**). Щелчок мышью по любому из элементов, отображаемых в окне **IntelliTrace**, позволяет просматривать статус приложения на тот момент времени, когда было зафиксировано конкретное событие. Окно **IntelliTrace** показано на рис. 6.15.

Окно **IntelliTrace** может быть очень полезно, если вы перешли через утверждение, которое изменило значение переменной, а затем вам потребовалось вернуться назад и посмотреть, каким было значение переменной до того, как оно было изменено. На рис. 6.15 показана как раз такая ситуация, когда подсвеченное событие, **Breakpoint hit: Main**, позволяет просмотреть окна **Locals** и **Call Stack**. Важное отличие в данном случае заключается в том, что показанные значения относятся к моменту времени, когда случилось конкретное событие, а не к текущему моменту времени, что дает вам очень важную и ценную информацию о том, что происходило в тот момент с вашей программой. Еще одна область применения функции IntelliTrace заключается в том, что с ее помощью можно изучать файлы журналов, созданные другим разработчиком или тестировщиком ПО с помощью нового инструмента Microsoft Test and Lab, которое может записывать весь сеанс тестирования.

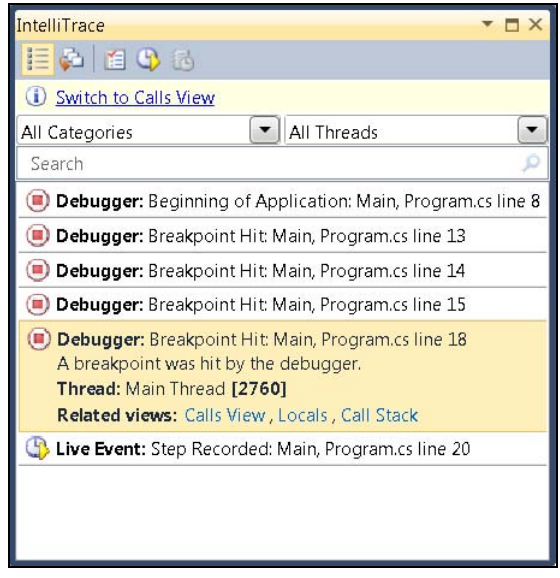


Рис. 6.15. Окно **IntelliTrace** отображает историю отладочного сеанса

Чтобы сконфигурировать опции IntelliTrace, следует выбрать из меню опции **Tools | Options | IntelliTrace**. IntelliTrace создает файлы журналов, которые существуют до тех пор, пока запущена и работает среда VS. При закрытии VS, файлы журналов удаляются, поэтому не забудьте скопировать их перед тем, как закрыть VS. Расположение файлов журнала задается в ветви **Tools | Options | IntelliTrace | Advanced.**

Если вы получите файл журнала от другого разработчика, вы можете загрузить его, выбрав из меню команды **File | Open | Open New**. После этого вы сможете просмотреть историю отладки и статус приложения для каждого события, которое имело место в ходе отладочного сеанса.

Решение проблем с помощью отладчика VS

В начальных разделах этой главы вы ознакомились с основными инструментами отладки VS и приобрели базовые навыки работы с отладчиком. В этом разделе мы попробуем решить несколько практических проблем, которые демонстрируют, как следует применять отладчик VS для их решения. Мы рассмотрим две наиболее типичных ситуации, с которыми может столкнуться программист в процессе повседневной работы: выявление и обработку некорректных данных и решение проблемы нулевых указателей (null references). Сама по себе рассматриваемая программа не особо сложна, но ее логика развита уже достаточно, чтобы дать вам представление о том, каким образом можно заблудиться в лабиринте и какие подходы следует применять, чтобы оттуда выбраться. Сначала давайте рассмотрим программу как таковую, а затем выполним два упражнения на поиск и устранение ошибок.

Программа с ошибками

Код программы, которая рассматривается в этом разделе, содержит ошибки, и для выполнения упражнений важно, чтобы вы ввели этот код именно так, как он приведен в листингах, ничего не изменяя и не исправляя, даже если вы уже видите эти ошибки. В противном случае у вас не получится проделать описываемые упражнения. Я подробно опишу каждый фрагмент кода, но поначалу постараюсь "не выдавать секретов". Чуть позднее мы пошагово пройдем всю процедуру отладки, с помощью которой в реальной жизни программисты выполняют поиск и исправление ошибок в своем коде. Рассматриваемая программа представляет собой средство поиска, которое по введенной фамилии ищет этого человека в списке клиентов. Если программа находит имя в списке клиентов, она выводит фамилию и имя этого клиента. В противном случае, программа выведет сообщение, информирующее пользователя о том, что указанная фамилия в списке клиентов не обнаружена.

Программа состоит из трех основных частей: класса, который содержит информацию о клиенте, класса, который возвращает список клиентов, и класса, содержащего метод `Main`, запускающий программу. В последующих нескольких разделах каждый из классов будет описан подробно.

Класс *Customer*

Каждый раз, когда вам требуется работать с данными, необходимо создать класс, который будет содержать эти данные. Так как приложение предназначено для работы с клиентами, вполне естественно будет назвать этот класс `Customer`. Код этого класса приведен в листингах 6.3 (для C#) и 6.4 (для VB).

Листинг 6.3. Код класса Customer на C#

```
public class Customer
{
        public string FirstName { get; set; }
        public string LastName { get; set; }
}
```

Листинг 6.4. Код класса Customer на VB

```
Public Class Customer
        Property FirstName As String
        Property LastName As String
End Class
```

В этом примере приведен лишь необходимый минимум информации, предназначенный для демонстрационных целей. Любое приложение, имеющее практическую ценность, должно иметь класс, в котором содержится гораздо большее количество свойств. Обратите внимание, что оба свойства имеют тип `string`.

Класс *CustomerRepository*

В нашей программе мы создадим класс, единственное предназначение которого заключается в работе с данными. Это — типовой и широко распространенный шаблон, который называется репозиторием (Repository). В листингах 6.5 и 6.6 приведен код класса `CustomerRepository`, который имеет метод, возвращающий список объектов `Customer`.

Листинг 6.5. Код класса CustomerRepository на C#

```
using System.Collections.Generic;

public class CustomerRepository
{
        public List<Customer> GetCustomers()
        {
                var customers = new List<Customer>
                {
                        new Customer
                        {
                                FirstName = "Franz",
                                LastName = "Smith"
                        },
                        new Customer
                        {
                                FirstName = "Jean "
                        },
                        new Customer
                        {
                                FirstName = "Wim",
                                LastName = "Meister"
                        }
                };

                return customers;
        }
}
```

Листинг 6.6. Код класса CustomerRepository на VB

```
Public Class CustomerRepository
        Public Function GetCustomers() As List(Of Customer)
                Dim customers As New List(Of Customer) From
                     {
                            New Customer With
                            {
                                    .FirstName = "Franz",
                                    .LastName = "Smith"
                            },
                            New Customer With
                            {
                                    .FirstName = "Jean "
                            },
                            New Customer With
                            {
                                    .FirstName = "Wim",
                                    .LastName = "Meister"
                            }
                }

                Return customers
        End Function
End Class
```

Метод `GetCustomers` возвращает значение типа `List<Customer>` (`List(Of Customer)` — для VB). Для целей нашего дальнейшего обсуждения, то, как работает метод `GetCustomers`, большого значения не имеет. Такой метод обычно получает список клиентов из базы данных, от Web-сервиса или другого объекта. Для простоты, `GetCustomers` инициализирует список `List` объектами `Customer`. Особо важное значение имеет та часть метода, в которой свойство `FirstName` получает значение `"Jean "`. Обратите внимание на символ пробела, добавленный в конец имени. Этот пробел необходим, чтобы получить именно такой сценарий поведения программы, который задуман (иначе говоря, здесь умышленно внесена ошибка, которую мы будем устранять в ходе сеанса отладки). То, что для объекта `Customer` со свойством `FirstName`, установленным на `"Jean "`, отсутствует свойство `LastName` — это тоже "так и задумано", чтобы затем продемонстрировать устранение данной ошибки.

Программа поиска

В листингах 6.7 и 6.8 приведен код поисковой программы, которая использует класс `CustomerRepository`, чтобы получить список объектов `Customer`. Программа кружится в цикле, обрабатывая список результатов, проверяя каждый член списка на соответствие заданному критерию поиска. Если найдено совпадение, то про-

грамма выводит полное имя клиента. Если совпадений не найдено, программа сообщает о том, что поиск закончился неудачей (клиент не найден).

Листинг 6.7. Поисковая программа на C#, возвращающая полное имя клиента

```
using System;

class Program
{
        static void Main()
        {
                var custRep = new CustomerRepository();

                var customers = custRep.GetCustomers();

                var searchName = "Jean";
                bool customerFound = false;

                foreach (var cust in customers)
                {
                        // 1. Первая ошибка
                        if (searchName == cust.FirstName)
                        {
                                Console.WriteLine(
                                        "Found: {0} {1}",
                                        cust.FirstName,
                                        cust.LastName);
                                        customerFound = true;
                        }
                }

                if (!customerFound)
                {
                       Console.WriteLine("Didn't find customer.");
                }

                Console.ReadKey();
        }
}
```

Листинг 6.8. Поисковая программа на VB, возвращающая полное имя клиента

```
Module Module1

        Sub Main()
            Dim custRep As New CustomerRepository

            Dim customers As List(Of Customer)
```

```
            customers = custRep.GetCustomers()

            Dim searchName As String = "Jean"
            Dim customerFound As Boolean = False

            For Each cust As Customer In customers
                ' 1. Первая ошибка
                If (searchName = cust.FirstName) Then

                        Console.WriteLine(
                                "Found: {0} {1}",
                                cust.FirstName,
                                cust.LastName)
                                customerFound = True

                End If
            Next

            If (customerFound = False) Then
                Console.WriteLine("Didn't find customer.")
            End If

            Console.ReadKey()

        End Sub

End Module
```

Обратите внимание, что переменная `searchName` получила значение `"Jean"`. В цикле переменная `searchName` сравнивается со свойством `FirstName` каждого из экземпляров класса `Customer`. Когда программа будет запущена, она выведет следующую строку:

```
Didn't find customer.
```

В принципе, мы ожидаем, что программа должна найти совпадающую строку и вывести ее на экран, но этого не происходит. Это и есть первая ошибка, и сейчас мы обсудим, каким образом можно ее обнаружить, использовав отладчик VS.

Поиск ошибки

На данном этапе мы знаем, что в нашей программе есть ошибка, и эта ошибка постоянно воспроизводится. Это означает, что для ее устранения необходимо использовать отладчик VS, чтобы обнаружить источник возникшей проблемы. В данной ситуации программа сообщает, что запись о клиенте не найдена, иными словами — что у нас нет клиента с именем (`FirstName`) `Jean`. Но при этом мы твердо знаем, что у нас есть такой клиент! Поэтому нам необходимо выяснить, почему же

программа не может найти нужную информацию. Давайте рассмотрим, какие шаги следует выполнить с помощью отладчика VS, чтобы выявить источник ошибки.

1. Начнем с установки точки останова на цикл `foreach` в методе `Main`. Это решение принято не случайно. Учитывая природу нашей проблемы, я выбрал именно ту часть программы, которая с высокой вероятностью прольет свет на то, что же явилось источником проблемы. Посмотрев на программу, несложно прийти к выводу, что одной из причин, по которой программа не может найти `searchName`, является то, что она не выполняет цикл `foreach`.
2. Нажмите клавишу <F5>, чтобы запустить программу в отладочном режиме. Благодаря этому программа не будет выполнена полностью, а дойдет до точки останова, сделает паузу, и мы сможем изучить ее состояние.
3. После того как VS встретит точку останова, наведите курсор на переменную `customers`, чтобы просмотреть, получено ли ею какое-нибудь значение. Вы увидите, что в списке `customers` имеются три значения. Тот факт, что клиенты имеются, указывает на то, что цикл `foreach` все же выполняется, поэтому предположение о его невыполнении отбрасывается как неверное.
4. Далее, установим точку останова на утверждение `if`, выполним щелчок правой кнопкой мыши на точке останова и установим следующее условие:
   - Для C#: `cust.FirstName == "Jean"`
   - Для VB: `cust.FirstName = "Jean"`

   Здесь наша цель заключается в том, чтобы посмотреть, что происходит, когда утверждение if находит запись, совпадающую с `searchName`. В этот момент мы предполагаем, что среди записей о клиентах есть клиент с именем `Jean`. При работе с небольшой программой, вы можете использовать для поиска этой записи окна **Autos**, **Locals** или **Watch**. Однако большинство реальных практических приложений содержат списки, в которых содержатся не две и не три записи, а намного большее их количество. Поэтому вместо того, чтобы тратить время на прокрутку длинных списков, насчитывающих десятки и сотни записей, воспользуйтесь возможностями отладчика VS по быстрому поиску нужной записи. Имейте в виду, что даже планы, которые на первый взгляд кажутся рациональными и хорошо продуманными, работают не всегда, как вы вскоре убедитесь. Но наша основная цель в данном случае заключается в том, чтобы сделать наиболее продуктивный первый шаг. Установка условной точки останова демонстрирует, как установка условных точек останова помогает предотвратить непродуктивный расход времени на пошаговую прокрутку циклов.
5. Нажмите клавишу <F5>, чтобы вновь запустить программу в отладочном режиме. Вы ожидаете встретить точку останова — однако этого не происходит. Почему? Ведь вы уже знаете, что с логикой программы все в порядке, потому что утверждение `if` представляет собой обычный оператор проверки на равенство. Возможно, вы просмотрели базу данных или любой другой источник, из которого были получены данные, но в данном сценарии вы совершенно определенно

знаете, что клиент с именем `Jean` присутствует в списке. Эта ситуация иллюстрирует типичную проблему, когда качество данных, с которыми вы работаете, не соответствует требуемому уровню.

6. Тогда давайте изменим условие точки останова следующим образом:
   - Для C#: `cust.FirstName.Contains("Jean")`
   - Для VB: `cust.FirstName.Contains("Jean")`

   После этого перезапустим программу. Обратите внимание — теперь мы подозреваем, что наша программа работает с некорректными данными, поэтому мы и дополняем вызовом `Contains` наши строки, задающие условия. Мы рассчитываем найти "лишние" символы (например, пробелы) в данных, с которыми выполняется сравнение. Задержите курсор над `cust.FirstName` или посмотрите на значение переменной `cust` в одном из окон отладчика, чтобы убедиться, та ли это строка, которую мы ищем. Теперь точка останова будет срабатывать на всех записях, которые содержат последовательность символов `"Jean"`, включая такие, как, например, `Jean-Claude`. Таким образом, вы получите множество совпадений, включая и те записи, которых вы не искали. Преимущество заключается в том, что в данном случае вам потребуется просматривать гораздо меньшее количество записей, и благодаря этому вы сможете сэкономить время. Если вы получите несколько записей, вы можете нажать клавишу <F5>, и точка останова будет срабатывать на каждой записи, позволяя вам исследовать значение. В данном случае, поскольку набор записей совсем мал, мы найдем нужную запись сразу же.

7. Нажмите клавишу <F10>, чтобы перешагнуть через условие `if`. В результате мы должны получить ответ на вопрос о том, правильно ли было оценено условие. В данном случае, VS не вступает в утверждение `if`, а вместо этого перемещается к концу утверждения, и это означает, что `searchName` и `cust.FirstName` не совпадают. В свою очередь, это говорит о том, что вам следует пристальнее присмотреться к `cust.FirstName`, чтобы выяснить, что за проблема возникает с вашими данными.

8. Далее, воспользуемся парочкой инструментов отладчика VS, чтобы исследовать значение `cust.FirstName` и обнаружить, почему не срабатывает условие проверки на равенство. Откройте окно **Immediate** (либо выбрав из меню команды **Debug** | **Windows** | **Immediate**, либо нажав клавиатурную комбинацию <CTRL> <ALT>+<I>) и выполните оценку следующего выражения:

   ```
   cust.FirstName
   ```

   В результате вы получите следующую оценку (рис. 6.16):

   ```
   "Jean "
   ```

   Теперь вам становится очевидно, что ошибка возникает в результате ошибки в данных — на конце имени имеется пробел. Очевидно, что `"Jean"` не совпадет с `"Jean "` как раз из-за наличия лишнего символа в данных. Помимо пробела, лишний символ может оказаться и каким-либо еще непечатаемым (т. е. управ-

ляющим) символом. Хорошо бы выяснить, точно ли это пробел или какой-то другой символ. VS может помочь и в этом вопросе тоже.

9. Откройте окно **Memory**[3] и введите в поле **Address** строку `cust.FirstName`. Нажмите клавишу <ENTER>. В результате вы сможете просмотреть данные в шестнадцатеричном представлении по тому адресу памяти, где они располагаются (рис. 6.17).

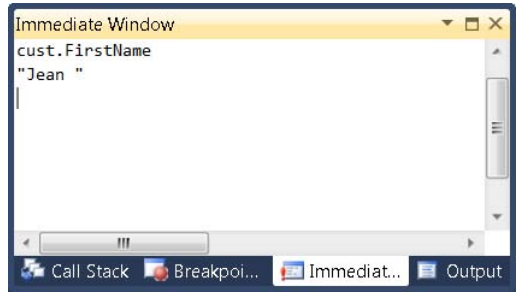


Рис. 6.16. Результат оценки выражения `cust.FirstName`

На левой границе окна **Memory** указываются адреса памяти, по которым можно выполнять их прокрутку. В данном случае шестнадцатеричный дамп будет прокручен до адреса, по которому впервые встречается в памяти переменная `cust.FirstName`. В середине окна находится шестнадцатеричное представление данных, а на правой границе — представление данных в виде читаемых символов, где все символы, не имеющие печатного представления, представлены в виде точек. Как видите, в первой строке третьего столбца присутствует строка `".J.e.a.n."`. В среде .NET символы используют 16-битную кодировку Unicode, а данные символа заполняют только первый байт, а второй представлен как `00`, в результате чего символы и разделены точками. Если бы данные использовали другой набор символов (например, японские иероглифы — Kanji, то оба байта были бы заполнены не точками, а другими символами. Шестнадцатеричное представление данных `".J.e.a.n."` выглядит так: `"00 4a 00 65 00 61 00 6e 00 20"`. Сверившись с кодовой таблицей Unicode, которую можно найти по адресу **http://unicode.org/**, вы увидите, что визуальное и шестнадцатеричное представление данных совпадают.

Как видите, на конце первой строки находятся байты `00 20` (см. рис. 6.16), а это говорит о том, что на конце строки `Jean` находится символ пробела. Получение этой информации поможет вам уведомить об ошибке того человека, который отвечает за данные, с тем, чтобы он позаботился об их качестве и удалил лишние пробелы. Некоторые аппаратные и программные платформы могут использовать в качестве разделителей другие, нестандартные символы, и такие символы могут по ошибке попасть в данные, что и вызывает ошибки, подобные только что рассмотренной.

---

[3] В отладочном режиме можно открыть до четырех окон **Memory** либо через меню (**Debug** | **Windows** | **Memory**, либо с помощью клавиатурных комбинаций <ALT>+<6>, <ALT>+<M>+<2>, <ALT>+<M>+<3> и <ALT>+<M>+<4>, соответственно). Кроме того, эти окна окажутся доступными только в том случае, если в диалоговом окне **Options**, описанном ранее в этой главе, в составе ветви **Debugging** активизирована опция **Address-Level Debugging**. Подробнее об окнах **Memory** см. здесь: **http://msdn.microsoft.com/en-us/library/s3aw423e.aspx**. — *Прим. перев.*

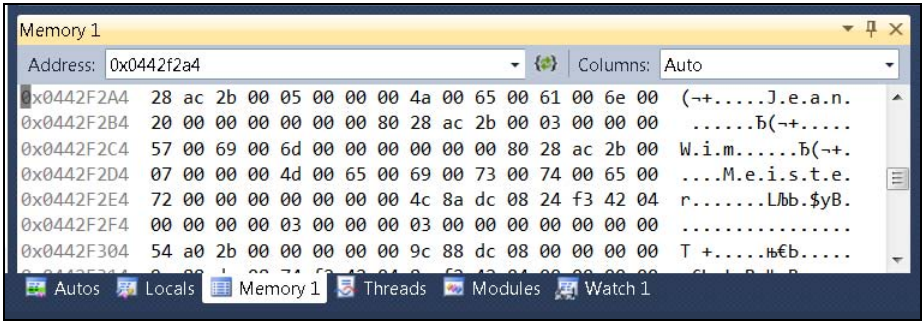


Рис. 6.17. Содержимое окна **Memory** с отображением проблемных данных

Исправление первой ошибки

Даже несмотря на то, что ошибка вызвана некачественными данными, а это совершенно не обязательно может быть вашей виной, перспективы исправления проблемы на уровне источника данных зачастую бывают весьма иллюзорны. Это значит, что вы как программист должны задуматься над тем, чтобы внести в ваш код такие исправления, которые не позволят этой ошибке возникать впредь. В этом разделе мы разработаем и применим исправление для данной ошибки. При этом мы преследуем двоякую цель: во-первых, нам требуется проиллюстрировать часто возникающие ситуации и подходы к исправлению ошибок, а во-вторых, показать, что на практике проблемы обычно бывают многогранными. Мы продемонстрируем еще один, совершенно иной пример ошибки, которая тоже часто встречается в мире программирования. Так что процедура, описанная в данном разделе, сначала исправит первую ошибку, а затем поможет обнаружить еще одну, не столь очевидную. Итак, чтобы исправить первую ошибку, пошагово проделайте следующие операции.

1. Нажмите клавиатурную комбинацию <SHIFT>+<F5>, чтобы остановить текущий сеанс отладки.
2. Теперь реализуем код, исправляющий только что обнаруженную нами ошибку. Закомментируйте содержимое цикла `foreach` и замените его кодом, приведенным в листингах 6.9 (для C#) и 6.10 (для VB). Этот код призван защитить программу от ошибок, вызванных лишними пробелами в данных.

Листинг 6.9. Код на C#, защищающий программу от лишних пробелов в данных

```
var firstName = cust.FirstName.Trim();
var lastName = cust.LastName.Trim();

if (searchName == cust.FirstName)

{
```

```
        Console.WriteLine(
                "Found: {0} {1}",
                firstName,
                lastName);
        customerFound = true;
}
```

Листинг 6.10. Код на VB, защищающий программу от лишних пробелов в данных

```
Dim firstName As String = cust.FirstName.Trim()
Dim lastName As String = cust.LastName.Trim()

        If (searchName = cust.FirstName) Then

            Console.WriteLine(
                    "Found: {0} {1}",
                    cust.FirstName,
                    cust.LastName)
            customerFound = True
        End If
Next
```

Обратите внимание, что наш новый код использует метод `string.Trim` для удаление лишних пробелов на конце строки, и результаты такой очистки присваиваются локальным переменным. `Trim` по умолчанию использует символ пробела, но перегружается, если вы укажете другой символ (на тот случай, если в конце строки окажется другой символ, а не пробел, см. рис. 6.17). Далее программа будет использовать очищенные данные.

Нажмите клавишу <F5>, чтобы запустить программу и проверить, как она будет работать. Вот тут, к сожалению, вас подстерегает еще одна ошибка — `NullReferenceException`. В отличие от ошибок времени исполнения, которые являются следствием некорректных данных, в данном случае VS здорово помогает вам в выявлении таких исключений, когда они имеют место в вашем коде. В следующем разделе ошибка `NullReferenceException` будет описана с большим количеством подробностей, что должно будет вам помочь, если вам придется иметь дело с устранением подобных ошибок в ваших программах.

Отладка и решение проблем NullReferenceException

Ошибки типа `NullReferenceException` встречаются достаточно часто и заслуживают отдельного рассмотрения, чтобы помочь вам эффективно бороться с такими проблемами. Как было описано в предыдущем разделе (см. шаг 3), VS сделает паузу, встретив ошибку `NullReferenceException`. В данном конкретном примере, VS приостановит исполнение программы в строке, которая очищает

свойство `LastName` (см. листинги 6.9 и 6.10), повторим эти строки здесь для вашего удобства:

- Для C#:

```
var firstName = cust.FirstName.Trim();
var lastName = cust.LastName.Trim();
```

- Для VB:

```
Dim firstName As String = cust.FirstName.Trim()
Dim lastName As String = cust.LastName.Trim()
```

Насколько вы помните, причина вызова метода `Trim` для свойств `FirstName` и `LastName` заключалась в том, чтобы выполнить очистку данных от лишних символов перед тем, как выполнять над этими данными дальнейшие операции. Хотя основное внимание мы уделяли свойству `FirstName`, мы точно так же вызвали метод `Trim` и для свойства `LastName`, чтобы помочь защитить программу от некорректных данных (хотя бы уже просто — на всякий случай). Следующие шаги призваны продемонстрировать, как следует использовать VS для анализа текущей ситуации и принимать решения об исправлении ошибки.

1. Если среда разработки VS еще не запущена, запустите ее, откройте проект и запустите программу. Дождитесь, когда VS остановит исполнение, выбросив ошибку `NullReferenceException`.
2. Задержите курсор над свойством `cust.LastName`, чтобы просмотреть значение. В качестве альтернативного варианта, вы можете просмотреть значение в одном из окон, доступных в отладочном режиме. Обратите внимание, что значение `LastName` установлено на `null`. Это и есть критическая точка нашего анализа — мы обнаружили значение, установленное на `null`. Совершенно очевидно, что значение `cust` не равняется `null`, потому что предыдущее утверждение, очищающее данные `FirstName`, было выполнено без ошибки, что и подтверждается путем изучения переменной `firstName`. В данном примере найти значение `null` было совсем не сложно, потому что оно появляется в той строке, на которой VS делает паузу. Бывают и более сложные ситуации, в которых вы передаете объект методу из библиотеки от сторонних разработчиков, и при этом у вас нет исходного кода этой библиотеки, причем VS останавливается на строке, в которой производится вызов метода. В этом случае, вам потребуется исследовать значения, передаваемые методу, чтобы выяснить, какое же из них равно `null`.

   Как только вы найдете значение `null`, вы поймете, почему возникла ошибка `NullReferenceException`. Значение `null` фактически означает отсутствие значения — а именно, что переменной вообще не было присвоено никакого значения. Если вы попытаетесь сослаться на переменную, которой присвоено значение `null`, вы получите сообщение об ошибке `NullReferenceException`. Это имеет смысл, поскольку вы пытаетесь выполнить операцию над переменной, которая не имеет определения. В данном конкретном примере, значение `LastName` равно `null`, но мы все равно ссылаемся на `LastName`, вызывая метод `Trim`. Это нело-

гично, потому что нет самой строки, из которой можно было бы удалять лишние символы — сама строковая переменная установлена на `null`.

Ошибка `NullReferenceException` возникает, чтобы защитить программу от выполнения некорректной операции. После того как вы найдете значение `null` и выясните причину появления сообщения об ошибке, необходимо выяснить, почему же переменная получает значение `null`, чтобы принять информированное решение о том, как же исправить эту проблему.

3. В окне **Immediate** введите следующую команду:

   - Для C#: `customers.IndexOf(cust)`
   - Для VB: `?customers.IndexOf(cust)`

   Эта команда возвратит значение `1`, которое представляет собой индекс текущей записи типа `Customer`, `cust`, в списке клиентов. Данная информация позволит добиться существенной экономии времени при попытке найти этот объект среди остальных данных программы.

4. На данный момент отладчик сделал паузу на строке, которая выполняет очистку `LastName`, в которой и произошла ошибка `NullReferenceException`, и на левой границе имеется желтая стрелка, символизирующая точку останова. С помощью мыши перетащите ее к строке, из которой осуществляется вызов `GetCustomers`. В настоящий момент наша задача состоит в том, чтобы выяснить, где и в какой момент переменная получает значение `null`. При удаче, мы можем сделать остановку в этой точке и, возможно, выяснить причину ошибки, по которой переменная получает значение `null`.

5. Нажмите клавишу <F11>, чтобы войти в метод `GetCustomers`. VS перейдет к первой строке метода `GetCustomer`.

6. Дважды нажмите клавишу <F10>, чтобы увидеть, какие значения возвращены. Данный пример настолько прост, что вы можете обнаружить данные визуально. Однако, в реальных сценариях, вам, возможно, придется запускать такой код, который выполняет запросы к базам данных или получает их из других источников, и подготавливать эти данные в такой форме, чтобы их можно было передавать любому вызывающему коду. Работа с базами данных будет подробно рассмотрена в *главе 7*, где вы узнаете больше о технике осуществления запросов. В данном же примере основная цель — рассказать о технике отладки, стараясь удержать изложение на максимально простом уровне, не отвлекая внимание читателя на вещи, не имеющие прямого отношения к делу. Поэтому нам необходимо обследовать данные и посмотреть, не являются ли они источником значений типа `null`. Для этой цели введем в окно Immediate следующую команду:

   - Для C#: `customers[1].LastName`
   - Для VB: `?customers(1).LastName`

   Дополнительно вам необходимо просмотреть список клиентов в одном из отладочных окон: **Autos**, **Locals** или **Watch**, обратив особое внимание на объект коллекции `Customer` с индексом, равным 1. Вспомним, что ранее на шаге 3

мы уже выяснили, что интересующий нас объект типа `Customer` имеет индекс 1. На основании полученного результата нам удалось выяснить, что свойство `LastName` для этого объекта типа `Customer` было установлено на `null` в источнике данных, и мы ничего не можем сделать, чтобы оно получало другие значения, а не `null`. Здесь мы сталкиваемся с еще одной разновидностью некорректных данных. Если вы можете проследить четкую тенденцию, вы будете вполне правы, если станете с недоверием относиться к данным, поставляемым конкретным пользователем или получаемым из той базы данных, откуда вы часто получаете такую некорректную информацию. На данном этапе, мы получили всю информацию, которая нам необходима для исправления ошибки. Это исправление гарантирует, что мы никогда, ни по какой случайности, не будем передавать методам данные типа `null`. Нажмите клавиатурную комбинацию <SHIFT>+<F5>, чтобы завершить отладочный сеанс.

7. В данном примере мы исправим проблему за счет того, что реализуем проверку значений `null` перед тем, как передавать эти данные вызываемым методам. В случае если какая-нибудь переменная окажется равной `null`, мы будем заменять ее стандартным значением по умолчанию. Закомментируйте содержимое цикла `foreach` и замените его кодом, приведенным в листингах 6.11 и 6.12.

Листинг 6.11. Код на C#, выполняющий проверку данных на равенство null

```
string firstName = string.Empty;
if (cust.FirstName != null)
{
        firstName = cust.FirstName.Trim();
}

string lastName =
        cust.LastName == null ?
        "" : cust.LastName.Trim();

        if (searchName == firstName)
        {
                Console.WriteLine(
                        "Found: {0} {1}",
                        firstName,
                        lastName);
                        customerFound = true;
        }
```

Листинг 6.12. Код на VB, выполняющий проверку данных на равенство null

```
Dim firstName As String = String.Empty

If (cust.FirstName IsNot Nothing) Then
```

```
        firstName = cust.FirstName.Trim()
End If

Dim lastName As String

If cust.LastName Is Nothing Then
        lastName = ""
Else
        lastName = cust.LastName.Trim()
End If

If (searchName = firstName) Then
        Console.WriteLine(
                "Found: {0} {1}",
                cust.FirstName,
                cust.LastName)
        customerFound = True
End If
```

Код, приведенный в листингах 6.11 и 6.12, исправляет проблему двумя различными способами, давая вам возможность воспользоваться различными подходами, в зависимости от того стиля, который вы предпочитаете. Сначала выполняется проверка свойств `FirstName` и `LastName`, чтобы выяснить, не получает ли одно из них значения `null` (в случае VB — `Nothing`). Если эти свойства не равны `null`, то это значит, что они содержат приемлемые строковые значения, и с ними вполне можно работать. В противном случае, мы возвращаем пустую строку.

В VB для работы с `Nothing` используются операторы `Is` и `IsNot`, в отличие от C#, где для работы с `null` применяются операторы `==` and `!=` , соответственно. Кроме того, оператор VB `Iif`, который является эквивалентом тернарному оператору C#, оценивает выражения `true` и `false`, а это приводит к тому, что ошибка `NullReferenceException` возникает даже в тех случаях, когда условие `false` исполняется. Поэтому в листинге 6.12 вместо оператора Iif применяется более "многословная" конструкция `If Then Else`.

Выдача пустой строки в качестве значения по умолчанию характерна только для этого примера. На практике вам придется ориентироваться по ситуации и решать, имеет ли смысл присваивать значение по умолчанию. Например, присутствие значения `null` может соответствовать неправильному условию, и вы можете предпочесть зарегистрировать такое событие в журнале и не давать пользователю возможности продолжать выполнение текущей операции. Еще один вариант действий заключается в том, чтобы пропустить текущую запись, обработать все остальные, а затем вывести список записей, которые не были обработаны. Вы можете выбрать любой из предложенных здесь подходов или отвергнуть их все. Однако моя точка зрения заключается в том, что вам следует

задуматься о том, что в каждой конкретной ситуации означает работа со значением `null`, а не просто устранить проблему нулевых указателей.

8. Нажмите клавишу <F5>, чтобы запустить программу. Вы получите следующий вывод:

```
Found: Jean
```

Вот теперь можно сказать, что вы победили!

Заключение

После прочтения данной главы вы должны приобрести базовые навыки отладки кода. *Раздел "Инструменты кодирования, упрощающие разработку"* подробно описывает, как изучать структуру вашего кода в ходе разработки. Далее в этой главе было рассказано о том, как следует устанавливать точки останова и выполнять их индивидуальную настройку. Затем была рассмотрена методика пошагового исполнения кода в отладочном режиме, навигация по приложению, вход в методы и выход из них, а также изменение каталога, в котором располагается исполняемый файл вашего приложения. Наконец, были рассмотрены отладочные окна, позволяющие изучить статус приложения на каждый конкретный момент времени. В частности, было описано использование окна истории отладки и его применение для изучения состояния приложения на каждом из этапов отладочного сеанса. Наконец, в заключительном разделе этой главы применение всех этих методик было проиллюстрировано на практическом примере, показывающем, как устранять ошибки в коде ваших приложений с помощью отладчика VS.

В следующей главе мы перейдем к более подробному изучению технологий .NET и рассмотрим, какие средства VS предоставляет для работы с данными.

Глава 7

Работа с данными

Прочитав данную главу, вы ознакомитесь со следующими ключевыми концепциями:

- Работа с базами данных SQL с помощью Visual Studio 2010;
- Запросы к данным с помощью языка LINQ (Language Integrated Query);
- Использование LINQ к SQL для осуществления запросов к данным SQL Server и манипулирования этими данными.

Большинство нашей повседневной работы состоит из операций, в ходе которых производятся те или иные манипуляции над данными, а большинство этих данных берется из баз данных (databases). Так как данные очень важны для работы наших приложений, выделим отдельную главу, в которой рассказывается о том, как осуществляется работа с данными в VS. Концепции, излагаемые в этой главе, исключительно важны для правильного понимания, потому что это повлияет практически на все аспекты вашей ежедневной работы программиста. Кроме того, в данной главе будет приведено большое количество примеров, иллюстрирующих, как следует работать с данными. Эти примеры сыграют важную роль и для понимания остального материала, излагаемого в данной книге. В любом случае, важность работы с данными никогда не следует недооценивать.

Хотя вы, безусловно, обладаете полной свободой выбора любых источников данных, примеры из этой главы рассматривают один из наиболее популярных и распространенных — Microsoft SQL Server. Microsoft поддерживает целый диапазон версий SQL Server — от бесплатных версий Express до продуктов уровня Enterprise, предназначенных для крупных корпоративных заказчиков. Так как SQL Server Express входит в комплект поставки VS, именно этой версией мы и воспользуемся для иллюстрации всех примеров из данной главы, а также для всех остальных примеров, которые будут рассматриваться в последующих главах их этой книги. Особого беспокойства это вызывать не должно, так как общие принципы работы с Express аналогичны принципам, применяемым при работе со всеми остальными версиями. Поэтому все, чему вы научитесь после прочтения этой главы, будет действовать и применительно ко всем остальным версиям SQL Server.

Операции над данными настолько важны, что в языках программирования реализована специальная поддержка для них — Language Integrated Query (LINQ). LINQ можно использовать для запросов к источникам данных разнообразных

типов — это могут быть объекты, XML или реляционные данные. В этой главе будет продемонстрировано, как следует использовать LINQ для запроса данных из SQL Server.

Работа с базами данных

Для непосредственной работы с базами данных VS предоставляет разнообразные средства. Бесплатные версии VS Express, например, такие, как Visual C# Express и Visual Basic Express, не имеют встроенной поддержки для такого инструментария. Однако вы можете заглянуть на сайт MSDN и загрузить бесплатную версию SQL Server Express, благодаря которой работа с базами данных станет возможной и с помощью Express-версий Visual Studio. Все функциональные возможности, демонстрируемые в этой главе, доступны в VS Professional или старше, в том числе и поддержка работы с SQL Server непосредственно из интегрированной среды разработчика (IDE).

Вводная информация о Server Explorer

Чтобы начать работу с базами данных, вам даже не обязательно открывать какой-либо проект. Вам достаточно просто запустить VS, а затем открыть окно **Server Explorer**, выбрав из меню команды **View** | **Other Windows** | **Server Explorer** или нажав клавиатурную комбинацию <CTRL>+<ALT>+<S>. Окно **Server Explorer**, показанное на рис. 7.1, позволяет работать с базами данных, серверами и SharePoint. Серверы предоставляют доступ к различным типам сервисов, которые дают вам возможность управления операционной системой, включая такие ее аспекты, как журналы событий (Event Logs), счетчики производительности (Performance Counters) и сервисы или службы (Services). Получать доступ к этим сервисам во время разработки очень удобно. Например, если вам необходимо перезагрузить один из сервисов операционной системы, вы можете сделать это очень быстро. SharePoint выходит за рамки тем, обсуждаемых в этой книге. В данном случае нас интересует раздел **Data Connections** в верхней части окна (см. рис. 7.1).

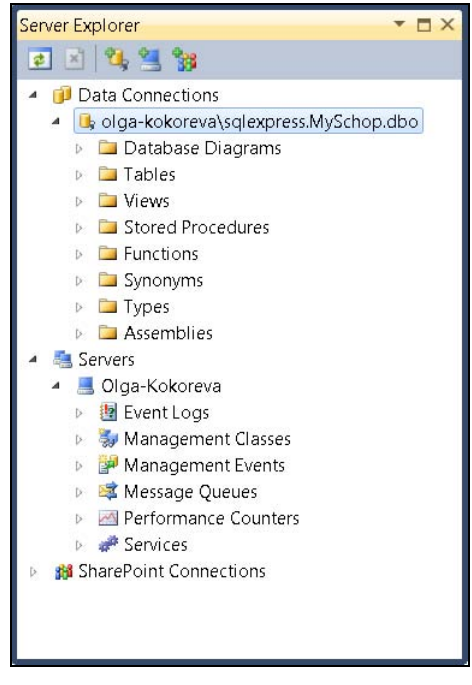


Рис. 7.1. Окно **Server Explorer**

Раздел **Data Connections** содержит список баз данных, из которого можно выбрать ту, с которой предполагается вести дальнейшую работу. Изначально этот список пуст, и вы должны будете сами добавлять в него соеди-

нения с базами данных. Для этого необходимо выполнить щелчок правой кнопкой мыши по области **Data Connections** и сконфигурировать параметры соединения с базой данных.

Так как процесс соединения с уже существующей базой данных аналогичен задаче создания новой базы данных, давайте начнем с процесса создания абсолютно новой базы данных. Эта процедура будет рассмотрена в следующем разделе.

Создание базы данных

Во всех примерах, которые мы рассмотрим далее в этой главе, будет использоваться база данных, которую мы создадим в данном разделе. Итак, давайте приступим к ее созданию, потому что нам в любом случае нужна база данных, с которой можно работать. Если в вашем распоряжении имеется VS Standard или более мощная версия, вам не потребуется никаких внешних инструментов для создания простой, потому что все необходимое для этой цели уже поставляется в составе VS. Впрочем, надо отметить, что бывают и более "продвинутые" сценарии, когда администратор базы данных может предпочесть другие подходы — например, использовать инструментарий SQL Server для создания базы данных, предоставив прикладным программистам возможности соединения с базами, которые администраторы создадут сами. Однако, в большинстве случаев, вам, чтобы начать работу, потребуется создавать базы данных именно самим.

Чтобы создать новую базу данных, щелкните правой кнопкой мыши по области **Data Connections** в окне **Server Explorer** и выберите из контекстного меню команду **Create New SQL Server Database**. В результате появится окно **Create New SQL Server Database**, показанное на рис. 7.2.

Рис. 7.2. Создание новой базы данных SQL Server

Имя сервера на рис. 7.2 выглядит так: **.\sqlexpress**. Точка перед символом обратной наклонной косой черты означает использование имени локального компьютера, а строка **sqlexpress** представляет собой имя сервера базы данных SQL Server Express. Имена серверов могут различаться, в зависимости от местоположения сервера и имени, присвоенного конкретному экземпляру сервера базы данных. Например, если бы вы выполняли развертывание приложения на предоставленный в общий доступ сервер Web-хостинга, то имя сервера могло бы выглядеть, например, так: **sql02.somedomain.com**. В этом случае имя сервера определяется хостинг-провайдером, услугами которого вы пользуетесь.

Вы имеете возможность выбирать опции аутентификации. Выбор таков: механизм аутентификации Windows и механизм аутентификации SQL Server. В данном случае я использую аутентификацию Windows, просто потому что это — простейший вариант. Кроме того, в примере используется локальная база данных, но вы можете пользоваться базой данных, уже существующей на одном из серверов в вашей сети или где-либо еще. Для базы данных, созданной на другом сервере, может потребоваться учетная запись SQL — входное имя (login) и пароль (password), впрочем, это уже другой метод аутентификации.

Добавив имя базы данных, нажмите кнопку **OK**, и база данных будет создана. Как показано на рис. 7.2, эта база получила имя **MyShop**. Это имя вполне соответствует приложению, предназначенному для обслуживания клиентов, заказывающих товары в магазине. Новая база данных появится в области **Data Connections** в окне **Server Explorer** (см. рис. 7.1). Теперь можно начинать добавлять в эту базу данных таблицы.

Добавление таблиц

Наша база данных должна содержать информацию о заказчиках, сделанных ими заказах, а также детальные сведения о каждом заказе. Данные должны храниться в таблицах, созданию которых посвящен данный раздел. В последующих разделах будет продемонстрировано выполнение таких операций над данными, как **Create**, **Read**, **Update** и **Delete** (CRUD). А пока давайте приступим к созданию таблиц.

Чтобы создать таблицу, выполните щелчок правой кнопкой мыши по ветви **Tables** вашей базы данных в окне **Server Explorer** и выберите из контекстного меню команду **Add New Table**. Вы увидите окно **Table Designer**, выглядящее примерно так, как показано на рис. 7.3. В окне, которое появится на вашем экране, изначально не будет столбцов **CustomerID** и **Name**, но их мы создадим на следующем шаге.

В окне **Table Designer** можно добавлять столбцы в состав таблиц и конфигурировать для них типы данных (например, назначать столбцу такие типы, как `integer`, `date`, `float` или `character`) и задавать другие свойства данных, которые должны храниться в этом столбце или в конкретной таблице. Например, таблица, показанная на рис. 7.3, имеет два столбца, **CustomerID** с типом данных `int` и **Name** с типом данных `nvarchar(50)`. Убедитесь в том, что опция `Null` для каждого из столбцов этой таблицы отключена — это необходимо для того, чтобы избежать ошибок в коде, который не выполняет проверок на `null`, как будет показано далее в этой главе.

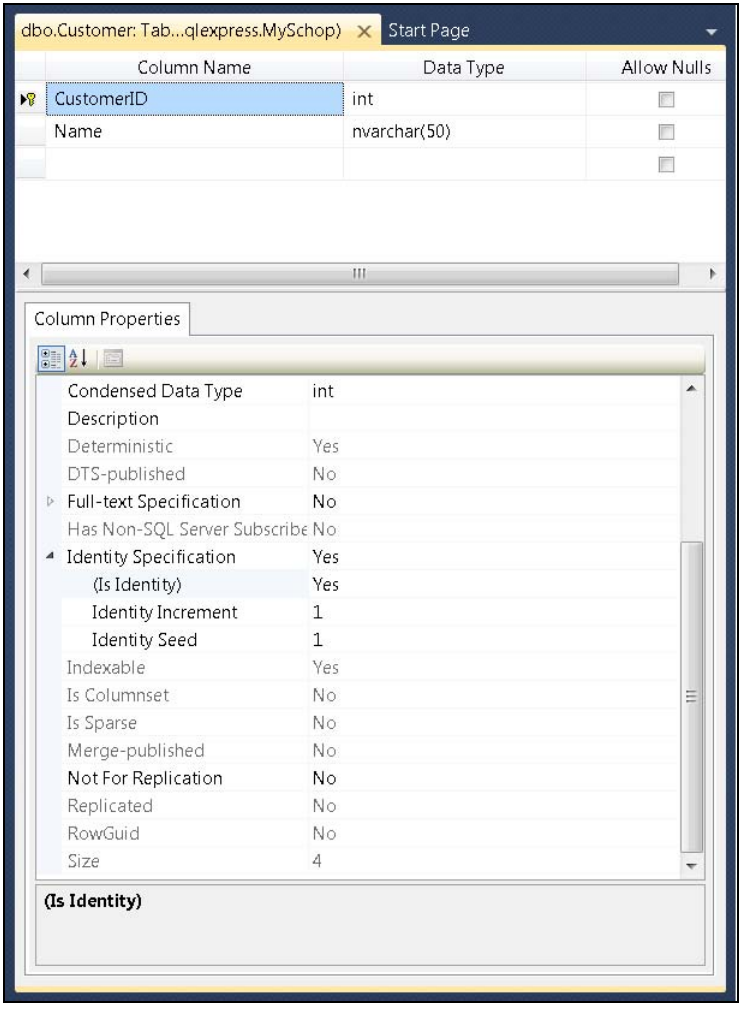


Рис. 7.3. Таблица **Customer**

Примечание

Каждая база данных, в том числе и база данных SQL Server, имеет свою систему типов данных, и эти системы не всегда полностью соответствуют системе типов данных, принятой для .NET. Впрочем, несмотря на сказанное, есть и типы, для которых находятся практически идеальные соответствия. Например, тип данных `int` в SQL — это то же самое, что и типы данных `int` в C# или `Integer` в VB. Такому типу данных SQL, как `nvarchar(50)`, соответствуют типы данных `string` (C#) или `String` (VB). Однако тип данных `nvarchar` ограничивается только 50 символами (или той длиной, которая указана в круглых скобках). С другой стороны, строковые типы данных (C# `string` и VB `String`) не имеют точно указанного размера. Подробное обсуждение типов данных SQL выходит за рамки круга тем, обсуждаемых в данной книге, но, тем не менее, вы должны быть осведомлены о том, что между типами данных в SQL и .NET существуют различия.

Столбец **CustomerID** является первичным ключом (primary key) и помечен соответствующим символом. Чтобы сделать столбец таблицы первичным ключом, выделите нужный столбец, щелкните правой кнопкой мыши и выберите из контекстного меню команду **Set Primary Key**. Если вам требуется составной первичный ключ (случай, когда первичный ключ определяется несколькими столбцами, не применимый в этом простейшем примере), вам следует нажать клавишу <CTRL> и последовательно выделить все столбцы, которые необходимо включить в состав первичного ключа. После того как все необходимые столбцы будут выделены, выполните щелчок правой кнопкой мыши и выберите из контекстного меню команду **Set Primary Key**.

Примечание

Если вы работаете с LINQ, о чем будет рассказано далее в этой главе, абсолютно необходимо присваивать первичный ключ каждой таблице.

Кроме установки первичного ключа, полезно установить для столбца свойство автоматического приращения (auto-increment) с тем, чтобы каждая вновь вводимая запись получала для первичного ключа уникальное значение. Как показано на рис. 7.3, столбец **CustomerID** выделен и список **Column Properties** прокручен до свойства **Identity Specification**. По умолчанию, список **Identity Specification** свернут и для него установлено значение **No**. Вам потребуется развернуть список **Identity Specification**, щелкнув по кнопке с изображением стрелки, и изменить значение на **Yes**, нажав на кнопку с изображением направленной вниз стрелки. Выберите значение **(Is Identity)**, которое по умолчанию установлено на **No**. Это автоматически активизирует поле **Identity Increment**, в котором должно быть указано число, добавляемое к каждой новой записи, и поле **Identity Seed**, которое указывает, каким должен быть первый номер. В результате установки поля **Identity Increment** первая запись, добавленная в таблицу, получит в качестве значения **CustomerID** число 1 (Identity Seed), а все последующие записи будут получать значения **CustomerID**, равные 2, 3, 4, и так далее (**Identity Increment**). Значение **CustomerID** для каждой вновь создаваемой записи будет представлять собой номер, уникальным образом идентифицирующий запись и упрощающий создание приложений для работы с данными.

Закончив создание таблицы, нажмите кнопку **Save** на инструментальной панели и, когда вам будет предложено сохранить таблицу, дайте ей имя **Customer.**

Теперь можно начать ввод данных в таблицу **Customer**. Для этого откройте новую базу данных в окне **Server Explorer**, перейдите в папку **Tables** в составе базы данных **MyShop**, щелкните правой кнопкой мыши по таблице **Customer** и выберите команду **Show Table Data.** Вы увидите сетку таблицы, примерно такую, как показано на рис. 7.4, куда можно начинать ввод данных о заказчиках.

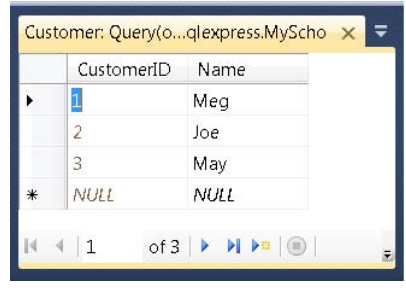


Рис. 7.4. Заполнение таблицы записями

Обратите внимание на то, что вам требуется всего лишь ввести имя в столбец **Name** (замещая

введенную туда по умолчанию строку `NULL`), но не требуется заполнять поле **CustomerID**, поскольку мы уже заблаговременно установили для него свойство Auto-increment.

В любой чуть более сложной базе данных обязательно будет присутствовать не одна таблица, а несколько. В базе данных **MyShop** любой заказчик (таблица **Customer**) делает заказ (таблица **Order**). Поэтому создайте еще одну таблицу и назовите ее **Order**. Таблица **Order** должна выглядеть так, как показано на рис. 7.5. Как видите, эта таблица имеет первичный ключ (Primary Key) с именем **OrderID**, поле с типом данных `datetime` и именем **OrderDate** и, наконец, поле **CustomerID**.

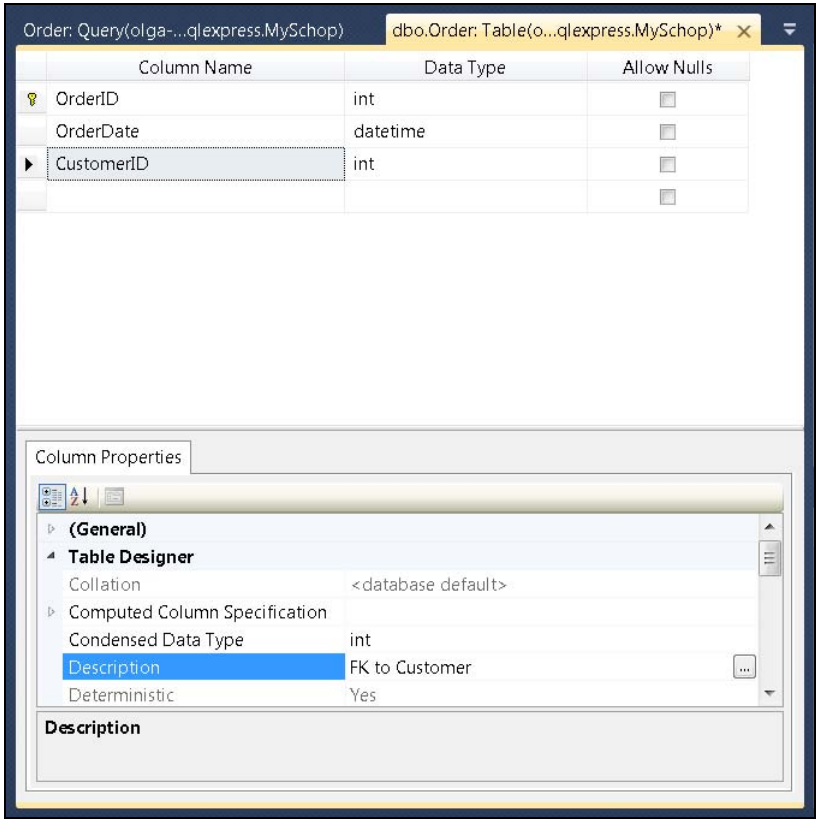


Рис. 7.5. Таблица **Order**

Поле **Description** в столбце **Column Properties** для поля **CustomerID** содержит символы **FK to Customer**. **FK** — это аббревиатура от Foreign Key (вторичный ключ). То, что поле помечено как вторичный ключ, означает, что между родительской таблицей (parent table) и дочерней таблицей (child table) существует связь (relationship). О связях между таблицами и вторичных ключах будет подробнее рассказано в следующем разделе, и там же будет продемонстрировано создание связей и вторичных ключей.

Установка связей между таблицами по вторичным ключам

Вторичные ключи (Foreign keys) позволяют устанавливать между двумя таблицами реляционные связи. Такие связи следует трактовать как отношения родитель/потомок (parent/child), главное/подробности (master/detail) или просто отношение "один-ко-многим" (one-to-many). При этом каждый из приведенных терминов является равнозначным эквивалентом для других. Я при дальнейшем изложении буду использовать термин "родитель/потомок", как наиболее понятный на интуитивном уровне. В предыдущем разделе было показано, как создать таблицы **Customer** и **Order**. Отношение, связывающее эти две таблицы, заключается в том, что таблица **Customer** является "родителем", а таблица **Order** — "потомком". Одной записи, принадлежащей к таблице **Customer**, может соответствовать множество записей в таблице **Order**. И действительно — клиент может зарегистрироваться, но не сделать ни одного заказа (или сделать множество заказов). По аналогии с этим, взрослый человек может либо не иметь детей, либо иметь столько детей, сколько захочет и сможет. Внешний ключ (foreign key) помогает управлять связями между таблицами в базе данных (в нашем примере это будут таблицы **Customer** и **Order**).

Техника создания отношений через внешний ключ заключается в том, что вы помещаете внешний ключ в дочернюю таблицу, **Order**, и ссылаетесь через этот внешний ключ на первичный ключ в родительской таблице, **Customer**. В данном случае ссылка реализуется за счет того, что целые значения в соответствующих столбцах двух таблиц просто совпадают. Если значение ID совпадает в обеих таблицах, то записи считаются связанными. Как показано на рис. 7.5, в таблице **Order** есть столбец **CustomerID**, с типом данных `int`, и помеченный как внешний ключ. Этот внешний ключ ссылается на являющееся первичным ключом поле **CustomerID** в таблице **Customer** (см. рис. 7.3).

Чтобы создать в VS отношение по внешнему ключу, щелкните правой кнопкой мыши по столбцу **CustomerID** в таблице **Order** и из контекстного меню выберите команду **Relationships**. Теперь можно создать отношение по внешнему ключу, как показано на рис. 7.6.

Выберите команду **Add**, а затем выберите свойство **Tables And Columns Specific**. Теперь щелкните мышью по кнопке с изображением многоточия, которая находится на правой границе выбранного вами поля. Обратите внимание, что символ многоточия не появится до тех пор, пока вы не щелкнете мышью по полю **Tables And Columns Specific** в группе **(General)** диалогового окна **Foreign Key Relationships** (см. рис. 7.6). После этого появится окно **Tables and Columns**, показанное на рис. 7.7.

В таблице первичного ключа (поле **Primary key table**) выберите таблицу **Customer**, в результате чего автоматически будет выбран первичный ключ, **CustomerID** (см. рис. 7.7).

> ***Внимание!***
>
> Если этого не происходит, внимательно проверьте, установили ли вы столбцы для первичного ключа и сохранили ли вы эти изменения, действуя так, как было описано ранее в этой главе.

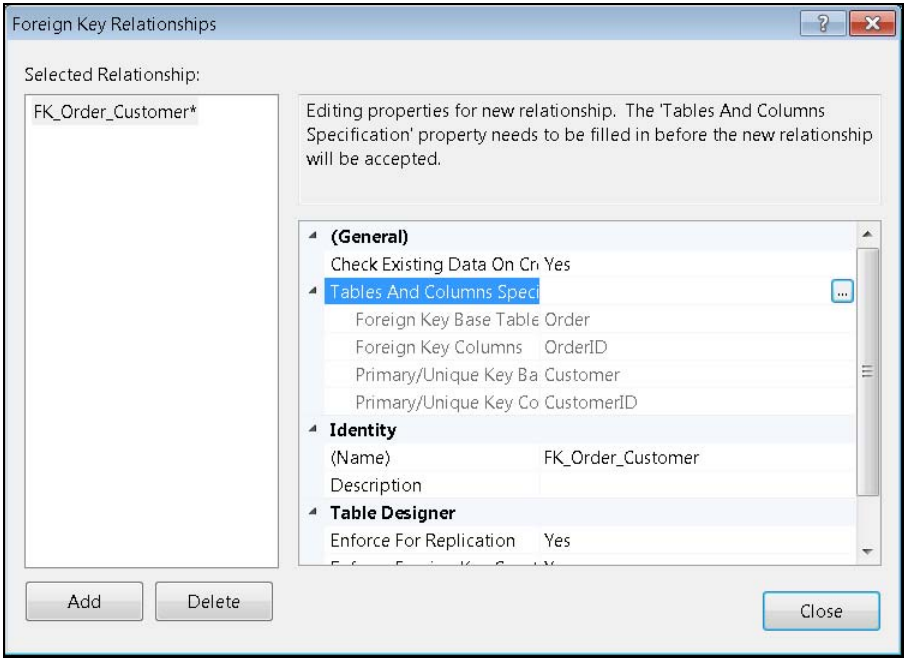


Рис. 7.6. Управление отношениями по внешнему ключу

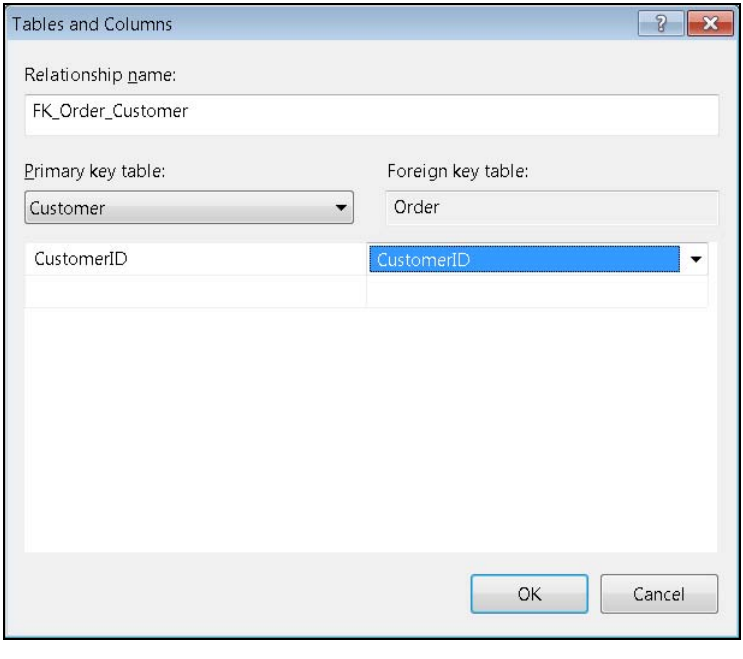


Рис. 7.7. Конфигурирование отношения между двумя таблицами по внешнему ключу

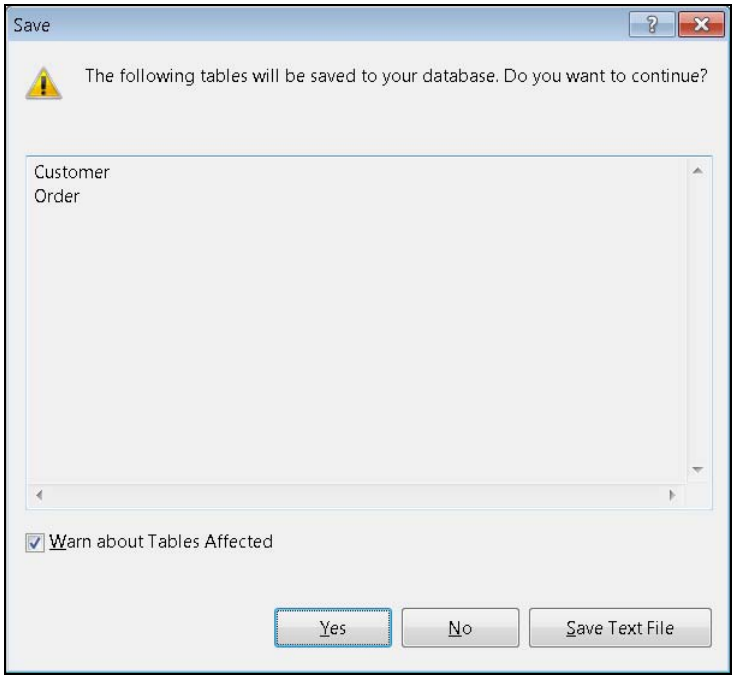


Рис. 7.8. Предупреждающее сообщение о сохранении отношения по внешнему ключу

В списке **Foreign key table** (см. рис. 7.7) вы сразу же увидите поле **OrderID**, которое является первичным ключом таблицы **Order**. Выделите поле **OrderID** и измените его на **CustomerID**, которое должно служить столбцом внешнего ключа. Нажмите для выхода кнопку **OK**, затем нажмите кнопку **Close**, чтобы завершить создание отношения по внешнему ключу. Нажмите кнопку **Save**, чтобы сохранить новое отношение по внешнему ключу. На экране появится предупреждающее сообщение, показанное на рис. 7.8, в котором будут перечислены таблицы, связанные новым отношением и затронутые внесенным изменением. Чтобы подтвердить внесение изменения в таблицы SQL Server, нажмите кнопку **Yes.**

Вы можете сбросить флажок **Warn about Tables Affected**, если не хотите видеть этих предупреждений, однако они служат важным предостережением, позволяющим избежать случайных ошибочных модификаций в SQL Server, который является внешним продуктом по отношению к VS 2010.

Как только внешний ключ будет задан, вы сможете начать добавление записей в таблицу **Order**, во многом так же, как вы делали это для таблицы **Customer**, но имейте в виду, что поле **CustomerID** должно совпадать с полем **CustomerID** таблицы **Customer** вследствие того, что теперь между таблицами существует отношение по внешнему ключу. Принудительная установка внешнего ключа дочерней таблицы таким образом, чтобы он ссылался на родительскую таблицу, позволяет обеспечить так называемую ссылочную целостность — внутреннюю непротиворечивость базы данных.

Совет

На рис. 7.7 показано редактируемое поле **Relationship Name**. Во многих случаях вам можно не беспокоиться об этом имени, потому что оно составляется в соответствии со стандартным соглашением об именовании и выглядит так: **FK_*Child_Parent***. Однако бывают ситуации, когда между одними и теми же таблицами вам потребуется иметь несколько отношений. Это значит, что VS будет добавлять в конец имени инкрементно наращиваемые номера. Таким образом, например, следующее отношение между этими таблицами получит следующее имя: **FK_*Child_Parent1***. В этих случаях неплохо будет заранее продумать собственную систему именования отношений, так, чтобы эти имена несли смысловую нагрузку. Впоследствии вам будет проще отличать отношения друг от друга и понимать, какие ограничения налагаются каждым из них. Чтобы понять, что имеется в виду под наложением ограничений (установкой правил), ведите новую запись в таблицу **Order**, но введите в поле **CustomerID** целое число, которого еще нет в таблице **Customer**, например, 9999. Попробуйте сохранить эту запись, а затем прочтите сообщение об ошибке, которое появится на экране.

Работая с множеством таблиц, вы, скорее всего, захотите лучше разобраться со структурой базы данных и отношениях между таблицами. В этом вам помогут диаграммы базы данных. Чтобы создать диаграмму базы данных, щелкните правой кнопкой мыши по папке **Database Diagrams** в составе вашей базы данных в окне **Server Explorer** и выберите из контекстного меню команду **Add New Diagram**. Получив информационное сообщение, запрашивающее создание объекта диаграммы базы данных, нажмите кнопку **Yes**. В окне **Add Table** нажмите клавишу <CTRL>, чтобы отобразить множество строк, щелкните мышью по каждой таблице, чтобы выделить ее, и нажмите кнопку **Add**. Появится новая диаграмма базы данных, выглядящая примерно так, как показано на рис. 7.9 (вы можете увидеть, что таблица **Order** располагается над таблицей **Customer**, и это нормально; значение в данном случае имеет расположение символов — ключа и символа бесконечности на концах стрелки, соединяющей две таблицы).

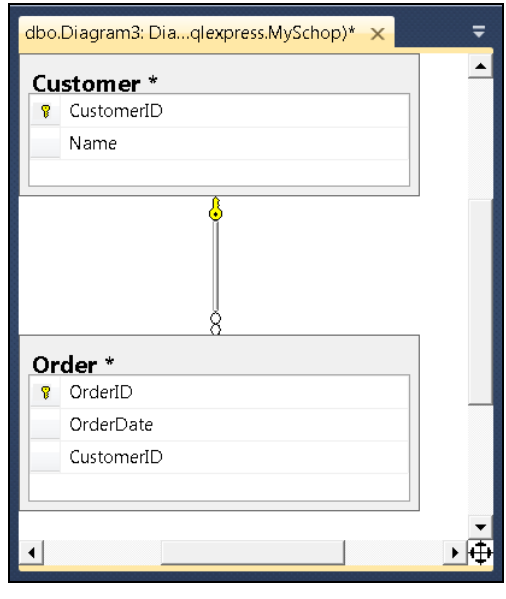


Рис. 7.9. Диаграмма базы данных

Как показано на рис. 7.9, диаграмма базы данных показывает таблицы, столбцы и отношения. Это окно можно использовать для добавления новых таблиц и отношений. Если вы хотите создать новую таблицу, щелкните правой кнопкой мыши в области окна, выберите из контекстного меню команду **Add Table**, а затем воспользуйтесь визуальным редактором для конфигурирования таблицы, так же, как это делалось в предыдущих примерах. Средство визуального проектирования полезно тем, что с его помощью отношения по внешнему ключу создаются намного проще, чем с помощью методов, которые мы применяли в примерах, приведенных

ранее. Чтобы создать связь по внешнему ключу, щелкните мышью по столбцу, который должен стать внешним ключом дочерней таблицы, затем перетаскиванием курсора проведите стрелку к столбцу, являющемуся главным ключом родительской таблицы, и отпустите кнопку мыши. Завершив работу по созданию диаграммы базы данных, VS предложит создать имя для этой диаграммы; вам следует ввести имя диаграммы по собственному выбору и нажать кнопку **OK**, чтобы сохранить диаграмму.

Остальные функции диаграммы базы данных включают навигацию, печать и просмотр множества диаграмм. Если размер вашей диаграммы базы данных превышает размер экрана, щелкните мышью по кнопке с изображением четырех стрелок, направленных в разные стороны (кнопка находится в правом верхнем углу диаграммы базы данных), и быстро перемещайте курсор мыши, чтобы перемещаться по документу.

Если вам требуется получить постоянную копию диаграммы, щелкните правой кнопкой мыши и скопируйте ее в буфер или выберите из меню команды **File | Print**. Кроме того, вы можете добавлять несколько диаграмм в папку **Database Diagrams**, благодаря чему вы получите множество различных представлений, что повышает удобство работы.

Кроме таблиц и диаграмм, можно использовать различные представления (views), хранимые процедуры (stored procedures), функции (functions), синонимы (synonyms), типы (types) и сборки (assemblies). Большинство из этих компонентов базы данных предназначены для работы над более сложными сценариями обработки данных, но, тем не менее, для дальнейшей успешной работы вам необходимо ознакомиться с хранимыми процедурами, которые будут рассматриваться в следующем разделе.

Добавление хранимых процедур

Хранимая процедура (stored procedure) представляет собой код, написанный на языке SQL и сохраненный таким образом, чтобы этот код являлся частью базы данных. Это — метод, сохраненный в самой базе данных, а не в коде вашей программы — отсюда и термин "хранимая процедура". В данном разделе будет показано, как создаются хранимые процедуры и как они исполняются. В последующих разделах этой главы будет продемонстрировано, как выполняется хранимая процедура из запроса к данным через LINQ.

Чтобы создать хранимую процедуру, щелкните правой кнопкой мыши по папке **Stored Procedures** нужной вам базы данных в окне **Server Explorer**, а затем выберите из меню команду **Add New Stored Procedure**. На экране появится окно редактора кода, в котором уже будет присутствовать "скелет" кода хранимой процедуры. Модифицируйте код таким образом, чтобы он извлекал все данные из таблицы **Customer**, как показано в листинге 7.1. После модификации кода шаблона, щелкните мышью по кнопке **Save**, и ваша новая хранимая процедура появится в папке **Stored Procedures** под именем вашей базы данных в окне **Server Explorer**.

Листинг 7.1. Пример хранимой процедуры

```
CREATE PROCEDURE GetCustomers
AS
declare @cust_count int
select @cust_count = count(*) from Customer
if @cust_count > 0
begin
        select [Name] from Customer
end
return
```

В листинге 7.1 объявлена переменная `@cust_count` и выполняется утверждение `select`, чтобы присвоить значение количества клиентов, `count(*)`, переменной `@cust_count`. Если значение `@cust_count` больше нуля, это означает, что в базе данных клиентов есть записи, и хранимая процедура запрашивает имена клиентов. Изучение TSQL (диалект SQL, принятый Microsoft) выходит за рамки данной книги, но вы можете, во-первых, бесплатно загрузить справочное руководство SQL Server Books Online и, во-вторых, приобрести следующую прекрасную книгу Дьюсана Петковича (Dusan Petkovic): "Microsoft SQL Server 2008: A Beginner's Guide", Fourth Edition, McGraw-Hill, 2008.

Чтобы выполнить эту хранимую процедуру, выполните щелчок правой кнопкой мыши по ее имени в базе данных в окне **Server Explorer** и из раскрывшегося контекстного меню выберите команду **Execute**. Если вы уже внесли в базу данных информацию о клиентах, то вывод, который вы получите, окажется примерно таким, как показано в листинге 7.2.

Листинг 7.2. Вывод хранимой процедуры

```
Running [dbo].[GetCustomers].
Name
--------------------------------------------------
Meg
Joe
May
No rows affected.
(3 row(s) returned)
@RETURN_VALUE = 0
Finished running [dbo].[GetCustomers].
```

Кроме исполнения хранимых процедур, VS позволяет их отлаживать. Чтобы выполнить отладку хранимой процедуры, установите точку останова на любую из строк кода хранимой процедуры. После этого выполните щелчок правой кнопкой мыши по имени хранимой процедуры в **Server Explorer**, а затем выберите из контекстного меню команду **Step Into Stored Procedure** или нажмите клавиатурную комбинацию <ALT>+<F5>. Если вам требуется больше информации по отладке, перечитайте *главу 6*, полностью посвященную этой теме.

Конфигурирование опций базы данных

VS имеет множество конфигурационных настроек для работы с базами данных. Чтобы выполнить их настройку, выберите из меню команды **Tools** | **Options** и разверните узел **Database Tools**, как показано на рис. 7.10.

Например, если вы развернете узел **Table and Database Designers** | **Table and Diagram Options**, то на раскрывшейся странице вы, в числе прочих опций, увидите и флажок **Prevent saving changes that require table re-creation**. По умолчанию, VS не дает возможности сохранять изменения внешних ключей, внесенные в существующие таблицы. Но если этот флажок сбросить, то вы получите возможность сохранять изменения внешних ключей, внесенные в существующую таблицу.

Как и в случае с остальными настройками VS, существуют десятки разнообразных параметров и установок, многие из которых должны быть вам понятны, если вы уже знакомы с SQL Server. Некоторые из настроек могут отличаться, в зависимости от той версии VS, которая имеется в вашем распоряжении. Поэтому тот экран **Options**, который вы увидите, может и отличаться от того, который показан на рис. 7.10.

Теперь, научившись создавать базы данных, таблицы и хранимые процедуры, вам необходимо разобраться с тем, как обращаться к базам данных из кода вашей программы. Остальная часть этой главы будет посвящена использованию LINQ для работы с данными. Сначала изучим базовый синтаксис LINQ на примере LINQ to Objects, а затем приступим к работе с SQL Server при помощи LINQ to SQL.

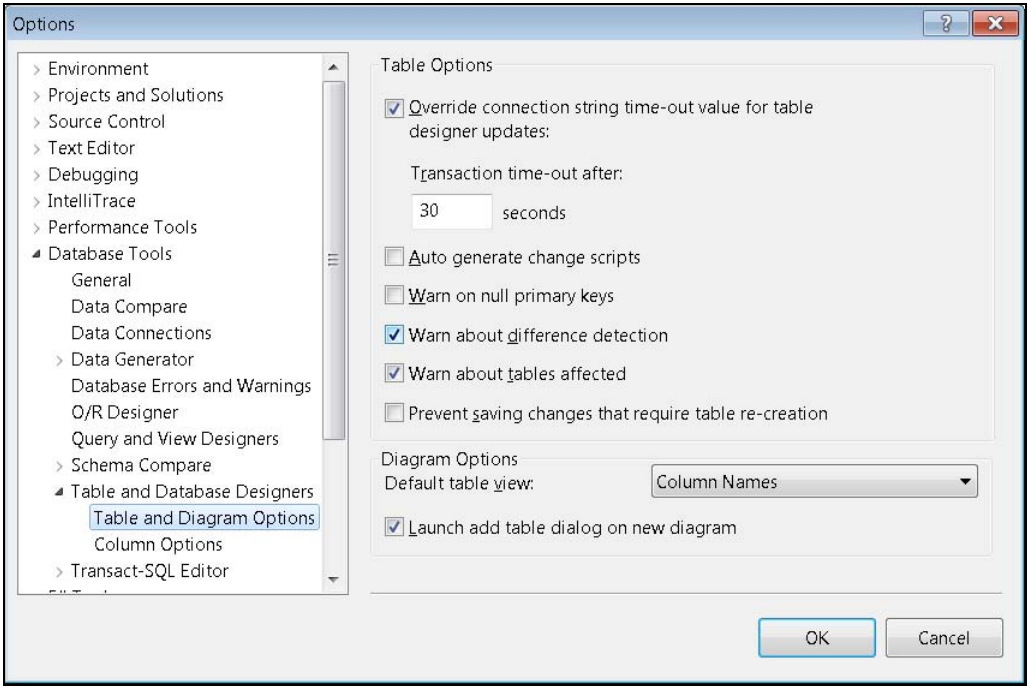


Рис. 7.10. Конфигурационные настройки VS для работы с базами данных (для Visual Studio Ultimate)

Изучаем Language Integrated Query (LINQ)

LINQ представляет собой набор функций, встроенных в языки программирования, такие как C# и VB, и предназначенных для работы с данными. LINQ — это аббревиатура от Language Integrated Query (синтаксис языка запросов, интегрированный с языками программирования платформы .NET Framework). Синтаксис LINQ представляет собой часть языка, в противоположность отдельной библиотеке. В этом разделе будут рассматриваться основы LINQ, включая LINQ to Objects, LINQ-провайдер для запросов коллекций объектов в памяти. Хорошая новость заключается в том, что синтаксис, который вы будете изучать, применим не только к LINQ to Objects, но и к остальным провайдерам LINQ, включая LINQ to SQL и др.

Для простоты изложения, примеры, приведенные в этой главе, будут использовать проект консольного приложения. Отображения данных в настольных приложениях с графическим пользовательским интерфейсом и Web-приложениях будут рассмотрены в дальнейших главах. Чтобы протестировать код, приведенный в этой главе, создайте консольное приложение и введите код примеров в метод `Main`, так, как было описано в предшествующих главах.

Выполнение запросов к коллекциям объектов с помощью LINQ

Одним из способов работы с LINQ является использование LINQ to Objects, благодаря чему вы сможете выполнять запросы к коллекциям объектов. LINQ можно использовать для запросов к любой коллекции, которая реализует интерфейс `IEnumerable`. Как вы, наверняка, помните, мы обсуждали интерфейсы в *главе 4*. В этой главе вы лишний раз убедитесь в том, насколько важны интерфейсы для разработки на платформе .NET. Программа, приведенная в листингах 7.3 (C#) и 7.4 (VB), использует LINQ для запросов к коллекции. Тип объекта представляет собой индивидуальный класс с именем `Customer`. Метод `Main` создает универсальный список объектов типа `Customer` и использует запрос LINQ, чтобы извлечь объекты `Customer`, для которых имя начинается на букву "M".

Листинг 7.3. Программа на C#, демонстрирующая использование LINQ

```
using System;
using System.Collections.Generic;
using System.Linq;

class Customer
{
        public string FirstName { get; set; }
        public string LastName { get; set; }
}

class Program
```

```
{
        static void Main(string[] args)
        {
                List<Customer> custList = new List<Customer>
                {
                        new Customer
                        {
                                FirstName = "Joe",
                                LastName = "Zev"
                        },
                        new Customer
                        {
                                FirstName = "May",
                                LastName = "Lee"
                        },
                        new Customer
                        {
                                FirstName = "Meg",
                                LastName = "Han"
                        }
                };

                var customers =
                    from cust in custList
                    where cust.FirstName.StartsWith("M")
                    select cust;

                foreach (var cust in customers)
                {
                        Console.WriteLine(cust.FirstName);
                }

                Console.ReadKey();
        }
}
```

Листинг 7.4. Программа на VB, демонстрирующая использование LINQ

```
Class Customer
       Property FirstName As String
       Property LastName As String
End Class

Module Module1

       Sub Main()
              Dim custList As New List(Of Customer) From
              {
                     New Customer With
```

```
                    {
                              .FirstName = "Joe",
                              .LastName = "Zev"
                    },
                    New Customer With
                    {
                              .FirstName = "May",
                              .LastName = "Lee"
                    },
                    New Customer With
                    {
                             .FirstName = "Meg",
                             .LastName = "Han"
                    }
          }

          Dim customers =
              From cust In custList
              Where cust.FirstName.StartsWith("M")
              Select cust

          For Each cust In customers
              Console.WriteLine(cust.FirstName)
          Next

          Console.ReadKey()
    End Sub
End Module
```

Примеры на C# и VB, приведенные в листингах 7.3 и 7.4, содержат примеры аналогичных запросов LINQ. Для уточнения, фрагменты кода, содержащие эти запросы, приведены в листингах 7.5 и 7.6.

Листинг 7.5. Запрос LINQ на C#

```
var customers =
        from cust in custList
        where cust.FirstName.StartsWith("M")
        select cust;
```

Листинг 7.6. Запрос LINQ на VB

```
Dim customers =
        From cust In custList
        Where cust.FirstName.StartsWith("M")
        Select cust
```

Переменная customers в запросах LINQ ссылается на новую коллекцию, где содержится результат выполнения запроса LINQ, который содержит список всех клиентов, для которых первым символом свойства FirstName является буква "M". Оператор from указывает диапазон переменных, откуда делается выборка, cust — это выбранное имя, а коллекция объектов, к которой выполняется запрос, custList, была создана и заполнена информацией предшествующим запросу кодом программы. Переменная диапазона используется для указания параметров запроса LINQ. В приведенном примере мы используем оператор where для фильтрации результатов запроса. Этот оператор where вызывает метод StartsWith, передавая ему свойство FirstName каждой переменной из диапазона cust.

Оператор select указывает, что каждый отдельный объект customer возвращается в новую коллекцию customers, которую мы объявили как var (в VB — Dim), имея в виду, что наша переменная customers становится коллекцией такого типа, который возвращается запросом LINQ. Кроме того, это означает, что результирующая коллекция customers будет содержать от нуля и более экземпляров типа Customer, в зависимости от фильтра, который мы укажем, и от того, содержит ли коллекция custList объекты типа Customer, возвращенные в результате оператора Select в нашем запросе LINQ. Оператор select является обязательным для запросов C#, а для VB эта часть запроса не обязательна, и, если она пропущена, то будет возвращен экземпляр переменной диапазона.

Проще говоря, наш запрос LINQ создает новый объект-коллекцию и присваивает его переменной customers (на самом деле, нам не важно, объектом какого типа в итоге окажется переменная customers), затем просматривает каждый объект в нашей ранее определенной и загруженной коллекции custList, выбирает из нее только объекты, у которых свойство FirstName начинается на букву "M", игнорируя все остальные, и, наконец, помещает отобранные объекты в новую коллекцию, созданную вами ранее и присвоенную переменной customers.

Создание проекции LINQ с анонимными типами

Вы можете осуществить индивидуальную настройку оператора select за счет использования так называемого анонимного типа (anonymous type). Эта индивидуальная настройка возвращаемых значений называется проекцией (projection). Анонимные типы упрощают создание индивидуальных проекций, позволяя возвращать результаты запросов LINQ в указанной вами форме, без необходимости преждевременно объявлять новый тип. Рассмотрим пример создания запроса, который объявляет новый анонимный тип для комбинирования свойств FirstName и LastName объекта Customer в новую переменную, FullName, которая будет создана как свойство со строковым типом данных, ассоциированное с объектом, возвращенным в переменной cust в составе утверждения foreach. Код данного примера приведен в листингах 7.7 и 7.8.

Листинг 7.7. Код на C#, иллюстрирующий создание проекции LINQ с анонимными типами

```
var customers =
    from cust in custList
    where cust.FirstName.StartsWith("M")
```

```
    select new
    {
        FullName =
        cust.FirstName + " " +
        cust.LastName
    };
foreach (var cust in customers)
{
    Console.WriteLine(cust.FullName);
}
```

Листинг 7.8. Код на VB, иллюстрирующий создание проекции LINQ с анонимными типами

```
Dim customers =
    From cust In custList
    Where cust.FirstName.StartsWith("M")
    Select New With
    {
        .FullName =
             cust.FirstName & " " &
             cust.LastName
    }

For Each cust In customers
    Console.WriteLine(cust.FullName)
Next
```

И в C#, и в VB в операторах `select` есть утверждение `new` (в VB — `New With`), которое определяет анонимный тип. Новый анонимный тип имеет единственное свойство, `FullName`, которое представляет собой комбинацию свойств `FirstName` и `LastName` в классе `Customer`, но новый тип будет иметь только свойство `FullName`. Обратите внимание, что цикл `foreach` использует свойство `FullName` вместо свойства `FirstName` из листинга 7.2. Преимущество и удобство этого анонимного типа заключается в том, что нам на самом деле не важно, что за объект генерируется для нас запросом LINQ, до тех пор, пока этот объект имеет новое свойство, ассоциированное с ним (в данном случае — `FullName`), а в нашей ситуации это условие соблюдено.

Переменная, `cust`, использовавшаяся в листингах 7.7 и 7.8, используется в двух различных диапазонах: в запросе LINQ и в утверждении `foreach`. Хотя идентификатор `cust` остается прежним, в данной ситуации используются два различных экземпляра. Хотя в своем собственном коде вы можете и не использовать этот подход, но я хотел продемонстрировать вам его, чтобы показать, что переменные диапазона, такие, как `cust`, ограничиваются диапазоном, в котором они определены.

Еще один нюанс кода, приведенного в листингах 7.7 и 7.8, заключается в том, что переменная `cust`, в цикле `foreach`, не принадлежит к типу `Customer`. В противоположность этому, она является экземпляром анонимного типа, созданного проекцией (оператор `select`) запроса LINQ. Следовательно, `FullName` является единственным свойством, которое имеет каждый экземпляр, `cust`, анонимного типа.

Использование LINQ для сортировки результирующей коллекции

Еще одна общая задача, которую вы, возможно, захотите выполнить над данными, заключается в сортировке результирующих данных таким образом, чтобы объекты были расположены в определенном порядке. Код, приведенный в листингах 7.9 и 7.10, выполняет сортировку списка клиентов в нисходящем порядке.

Листинг 7.9. Код на C#, выполняющий сортировку результирующего списка клиентов в нисходящем порядке

```
var customers =
    from cust in custList
    orderby cust.FirstName descending
    select cust;
```

Листинг 7.10. Код на VB, выполняющий сортировку результирующего списка клиентов в нисходящем порядке

```
Dim customers =
    From cust In custList
    Order By cust.FirstName Descending
    Select cust
```

Оператор `orderby` (`Order By` — в VB) указывает свойства, по которым должна выполняться сортировка. В этом примере сортировка списка производится по свойству `FirstName` в нисходящем порядке.

Это было лишь краткое описание всего, что вы можете делать с помощью LINQ, а на самом деле эти возможности намного шире. Я написал целую книгу о программировании с помощью LINQ ("LINQ Programming", McGraw-Hill/Professional, 2008). Оставшаяся часть этой главы проиллюстрирует все сказанное практическими примерами, причем продемонстрировано будет большое количество запросов LINQ. В данном случае вы будете работать с данными SQL Server, а не с объектами, хранящимися в памяти, как это было до сих пор.

Обработка данных с помощью LINQ to SQL

Технология LINQ to SQL позволяет поддерживать коммуникации с базами данных SQL Server. Существует множество типов провайдеров, например, LINQ to Entities для универсальных (родовых) баз данных (включая SQL Server), LINQ to XML для источников данных XML, и LINQ to Oracle для баз данных Oracle. В предшествующем разделе было показано, как использовать провайдера доступа к объектам в памяти, LINQ to Objects. Однако LINQ to SQL представляет собой простейшее в изучении средство, предоставляющее все необходимые возможности и поставляющееся вместе с VS. Когда вы изучите LINQ to SQL, переход к другим провайдерам окажется проще. В последующих нескольких разделах будет показано, как настраивать LINQ to SQL, как выполнять запросы и модифицировать данные.

Настройка LINQ to SQL

Настройка LINQ to SQL включает в себя запуск программы-мастера LINQ to SQL Wizard и добавление классов и методов. Действуя незаметно для пользователя, LINQ to SQL генерирует код, позволяя вам сэкономить много рабочего времени. В результате настройки LINQ to SQL вы получаете модель данных, представляющую собой среду с классами, которую вы можете использовать для запросов и модификации данных, хранящихся в базе данных, а также вызывать методы для инициации хранимых процедур.

Перед тем как настраивать LINQ to SQL, вам понадобится создать проект (для целей данной главы, как и в предыдущих главах, создадим консольное приложение). О том, как создаются проекты консольных приложений, подробно рассказывалось в *главе 5*, поэтому, если вы испытываете затруднения, вернитесь к этой главе. Выберите из меню команды **Add | New Item**, выберите опцию **LINQ to SQL Classes**, назовите файл MyShop.dbml и нажмите кнопку **Add**. В результате этого на экране появится окно **LINQ to SQL Designer**, с двумя панелями — одна для классов, а вторая — для методов. На рис. 7.11 показано окно **LINQ to SQL Designer**, в составе которого есть два класса и один метод.

Чтобы добавлять новые объекты в окно **LINQ to SQL Designer**, откройте окно **Server Explorer**, выберите базу данных и откройте папку **Tables**. Затем перетащите мышью таблицы **Customer** и **Order** из окна **Server Explorer** в левую панель окна **LINQ to SQL Designer**. В окне **LINQ to SQL Designer** (см. рис. 7.11) появятся классы **Customer** и **Order**, вместе со свойствами, каждое из которых соответствует полю в базе данных с таким же именем.

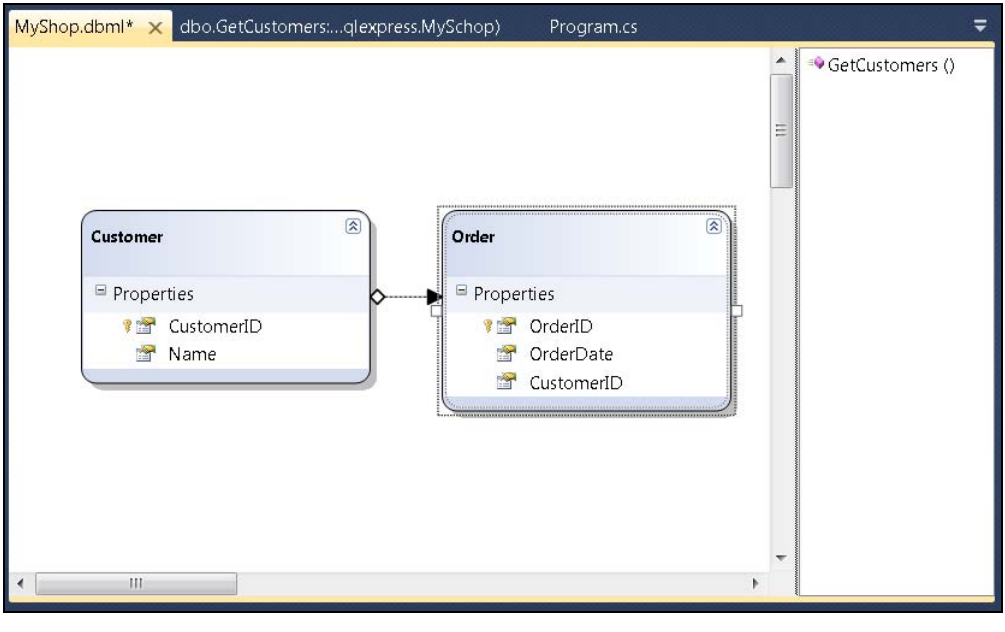


Рис. 7.11. Окно **LINQ to SQL Designer**

Линия, связывающая классы **Customer** и **Order**, называется ассоциацией (association). Как вы можете предположить с учетом ранее приводившегося описания межклассовых отношений, эта ассоциация определяет отношение между двумя классами. Хотя отношение между таблицами ограничено по внешнему ключу в дочерней таблице, который ссылается на первичный ключ родительской таблицы, ассоциация имеет противоположное направление; именно свойство родительского класса ссылается на всех потомков этого класса. При кодировании, вы можете использовать эту ассоциацию для навигации между родительскими и дочерними объектами.

Примечание

Такие особенности, как разница между отношениями по внешнему ключу в реляционных базах данных и ассоциациями в объектно-ориентированном коде, часто называются "рассогласованием интерфейсов" (impedance mismatch). Термин "рассогласование интерфейсов" (или "рассогласование нагрузки" — в таком виде термин "impedance mistmatch" был заимствован из электротехники, где под этим рассогласованием понимается неспособность входа одной системы принять выход другой), только в программировании он приобрел несколько иной смысл — рассогласование интерфейсов между объектами в объектно-ориентированном программировании и таблицами в базах данных. Интегрированный язык запросов LINQ был разработан как раз с той целью, чтобы устранить это рассогласование интерфейсов и работать с данными, рассматривая их с позиций объектно-ориентированного программирования. LINQ устраняет необходимость ручного выполнения низкоуровневой работы, такой, как копирование записей в объекты передачи данных (Data Transfer Objects, DTOs), которые вы разрабатываете и создаете.

В правой панели окна, показанного на рис. 7.11, показан метод `GetCustomers`, который позволяет вызывать хранимую процедуру `GetCustomers`. Вы можете помещать хранимые процедуры, такие, как процедура `GetCustomers`, в область методов окна **LINQ to SQL Designer**, открыв папку **Stored Procedures** под именем нужной вам базы данных в окне **Server Explorer** и перетащив мышью хранимую процедуру в правую панель окна **LINQ to SQL Designer**.

Если в вашей базе данных имеются представления (views) и функции (functions), то и их вы можете добавить в окно **LINQ to SQL Designer**, действуя точно так же, как вы только что действовали по отношению к классам и функциям. Прежде чем демонстрировать вам использования этих новых классов и представлений, я продемонстрирую вам еще некоторые возможности, предоставляемые вам в окне **LINQ to SQL Designer**.

Работа с LINQ to SQL Designer

Хотя наиболее важной из возможностей визуального редактора LINQ to SQL Designer является добавление классов и методов, неплохо будет рассмотреть и другие не менее полезные возможности этого инструмента, включая свертывание панели **Methods**, ее масштабирование и автоматическое макетирование (auto-layout). Доступ к этим возможностям осуществляется через контекстное меню, выводимое по щелчку правой кнопкой мыши в области окна **LINQ to SQL Designer**.

Большую часть времени, проводимого в окне **LINQ to SQL Designer**, вы будете работать с классами, и вам захочется получить как можно больше экранного пространства. Этой цели можно добиться, скрыв панель Methods. Просто выполните щелчок правой кнопкой мыши в области окна конструктора LINQ to SQL и выберите опцию **Hide Methods Pane**. Аналогично, выберите команду **Show Methods Pane**, если панель методов вам понадобилась, и вы хотите снова сделать ее видимой.

По умолчанию, для визуального конструктора LINQ to SQL установлен коэффициент масштабирования 100%, но вы можете изменить эту ситуацию, выполнив в области этой панели щелчок правой кнопкой мыши, выбрав из контекстного меню команду **Zoom**, а затем установив уровень масштабирования (Zoom level) в процентном выражении. Эта возможность полезна, если вы хотите получить более высокоуровневое представление, где в одном окне можно разместить большее количество объектов одновременно.

Если вы щелкнете правой кнопкой мыши и выберете из появившегося меню команду **Layout Diagram**, VS автоматически создаст макет вашей диаграммы таким образом, чтобы классы с отношениями могли физически располагаться на одной стороне, обеспечивая при этом минимальное перекрывание линий, символизирующих ассоциации — я называю эту функцию автоматическим макетированием (auto-layout). После того как вы выполните автоматическое макетирование, вы сможете вручную изменить расположение классов, выбрав и перетащив каждый класс на новое место — эту функцию я называю ручным макетированием (manual layout).

Совет

Будьте осторожны с функцией автоматического макетирования, и не стоит прибегать к ней, если вы уже имеете такой макет, который вас устраивает. Я обычно выполняю автоматическое макетирование после первого сеанса работы с базой данных в окне **LINQ to SQL Designer**. Затем я выполняю ручное макетирование, после чего работать с классами становится еще проще и удобнее. Использование автоматического макетирования после этого может привести к тому, что вы потеряете многие часы своей работы.

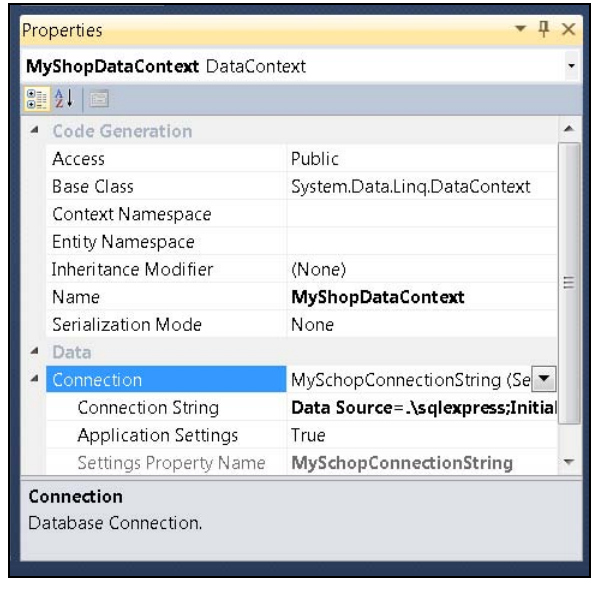


Рис. 7.12. Окно **Properties** в визуальном конструкторе LINQ to SQL

Разработчики в ходе работы на базой данных часто добавляют туда новые таблицы, которые, естественно, хотят видеть и в окне **LINQ to SQL Designer**. Добиться этого несложно — просто перетащите эти новые таблицы мышью из окна **Server Explorer**, как вы делали это ранее для таблиц **Customer** и **Order**. Если таблица претерпевает

изменения, вы можете выделить соответствующий класс в окне **LINQ to SQL Designer** и удалить этот класс, а затем перетащить в область визуального конструктора новую таблицу. Любые отношения по внешним ключам приведут к созданию ассоциаций с классами в окне **LINQ to SQL Designer**, естественно, если в окне присутствуют обе эти таблицы.

Важная часть работы с визуальным конструктором Linq to SQL составляют свойства (properties). Выполните щелчок правой кнопкой мыши в окне визуального конструктора, выберите из раскрывшегося меню команду **Properties**, и вы увидите окно **Properties**, выглядящее примерно так, как показано на рис. 7.12.

LINQ to SQL генерирует большой объем кода, и окно **Properties** позволяет вам модифицировать некоторые фрагменты этого кода, редактируя настройки в разделе **Code Generation**. Чтобы просмотреть задаваемые в этом разделе параметры, убедитесь в том, что нажата кнопка **Categorized** (), расположенная на крайней левой границе инструментальной панели окна **Properties**, а не кнопка **Alphabetical**, помеченная символами "AZ" (). Кроме того, вы можете просмотреть строку, задающую соединение с базой данных (database connection), которая была создана, когда вы перетягивали мышью объекты из окна **Server Explorer** в окно визуального конструктора LINQ to SQL и сохраняли их.

Кроме свойств общего содержимого окна визуального конструктора, вы можете просматривать свойства различных объектов, например, классов, ассоциаций и методов. Выберите объект, с которым хотите работать, щелкните по нему правой кнопкой мыши и выберите из появившегося меню команду **Properties**, после чего вы сможете просмотреть окно свойств для выделенного объекта.

Теперь у вас имеется модель данных, с которой вы можете работать. В последующих разделах будет показано, как следует работать с этой моделью, чтобы выполнять запросы к данным, добавлять новые данные, модифицировать и удалять существующие.

Введение в запросы с помощью LINQ to SQL

Чуть ранее, в этой главе рассказывалось о том, как пользоваться LINQ через провайдера LINQ to Objects. Все, что говорилось о LINQ to Objects, справедливо и в отношении других провайдеров LINQ, в том числе — LINQ to SQL. В этом разделе мы учтем все нюансы использования LINQ to SQL, из числа тех, что были уже описаны, и применим их для запросов к базам данных. В листингах 7.11 и 7.12 показан запрос LINQ с помощью LINQ to SQL, который возвращает значения из таблицы **Customer** базы данных **MyShop**, которая содержит таблицы, добавленные ранее в этой главе.

Листинг 7.11. Код на C#, выполняющий запрос данных с помощью LINQ to SQL

```
using System;
using System.Linq;


namespace LinqToSqlDemoCS
```

```
{
        class Program
        {
                static void Main()
                {
                        var myShop = new MyShopDataContext();
                        var customers =
                            from cust in myShop.Customers
                            where cust.Name != "Joe"
                            select cust;

                       foreach (var cust in customers)
                       {
                                Console.WriteLine("Name: " + cust.Name);
                       }
                       Console.ReadKey();
                }
        }
}
```

Листинг 7.12. Код на VB, выполняющий запрос данных с помощью LINQ to SQL

```
Module Module1

    Sub Main()
        Dim myShop As New MyShopDataContext

        Dim customers =
            From cust In myShop.Customers
            Where cust.Name IsNot "Joe"
            Select cust

        For Each cust In customers
            Console.WriteLine("Name: " & cust.Name)
        Next

        Console.ReadKey()
    End Sub
End Module
```

Если использовать данные, ранее введенные в эту базу данных (см. листинги 7.3 и 7.4), то вывод нашего кода будет таким:

```
Name: Meg
Name: May
```

За исключением того очевидного факта, что теперь мы получаем данные из реальной базы данных, разница между примерами, приведенными в листингах 7.11 и 7.12,

и ранее приведенными примерами, использовавшими LINQ to Objects, заключается в том, что мы должны использовать пространство имен `System.Linq` (только для C#), объявить контекст данных `MyShopDataContext` и сделать запрос в таблицу `Customers` из контекста данных. В C# использование директивы `using System.Linq` обязательно. Если ее пропустить, то компилятор выведет следующее сообщение об ошибке:

```
"Could not find an implementation of the query pattern for source type
'System.Data.Linq.Table<LinqToSqlDemoCS.Customer>'. 'Where' not found.
Are you missing a reference to 'System.Core.dll' or a using directive for
'System.Linq'?"
```

Запомните это сообщение, потому что каждый раз, когда вы будете добавлять новый файл к проекту C#, в котором используются запросы LINQ, появление такого сообщения будет сигналить вам о том, что вы забыли указать директиву `using` для пространства имен `System.Linq`.

Контекст данных представляет собой код, который генерируется VS, когда вы запускаете программу-мастер LINQ to SQL classes. Метод `Main` создает экземпляр класса `MyShopDataContext`, который представляет собой контекст данных. Имя этого экземпляра сгенерировано в тот момент, когда программа-мастер LINQ to SQL отработала, и вы присвоили имя своему файлу *.dbml.

Запросы LINQ to SQL выполняются из контекста данных, содержащего свойство, которое, в свою очередь, содержит коллекцию объектов класса, по имени которого названо и свойство, в данном случае — `myShop.Customers` и `myShop.Orders`. Запрос LINQ в методе `Main` использует экземпляр контекста данных `myShop` для доступа к коллекции `Customers` в части запроса `from`.

> ***Примечание***
>
> Провайдер LINQ to SQL использует плюрализованные свойства контекста данных. Однако результаты не являются идеальными — например, `Deer` превратится в `Deers`, что с точки зрения английского языка является неправильным. Кроме того, такой метод переименования, как плюрализация (pluralization)[1], был создан специально для английского языка, а для языков, отличных от английского, он может производить странные результаты. Если плюрализация, создаваемая в LINQ для класса, некорректна или просто вам не нравится, вы можете или выполнить двойной щелчок мышью по имени класса в окне визуального конструктора или изменить имя класса через окно **Properties**.

В данном разделе было продемонстрировано, как создается запрос LINQ to SQL, но, скорее всего, ваши запросы будут работать со многими таблицами, а не с одной, как в этом простейшем примере. Работа с множеством таблиц обсуждается в следующем разделе.

Выполнение запросов к множеству таблиц

До сих пор во всех запросах использовался единственный источник данных или таблица. Например, в листингах 7.11 и 7.12 это была таблица `Customers`. Однако

[1] Чуть подробнее на эту тему см. **http://tinyurl.com/4qwcjn**, и **http://msdn.microsoft.com/ru-ru/library/bb384507(VS.90).aspx**. — *Прим. перев.*

чаще всего вам требуется комбинировать результаты из множества таблиц, и это — именно та область, в которой вам могут оказаться полезными операторы `select`, осуществляющие выборку из многих таблиц и выполняющие объединение результатов. Чтобы продемонстрировать, как все это работает, определим сценарий, при котором вам требуется узнать даты всех заказов и имя клиента, который сделал заказ.

Этот множественный оператор `select` позволяет вам объединять таблицы на основании ассоциаций, созданных в визуальном конструкторе LINQ to SQL Designer. От родительского объекта можно перейти к дочернему объекту и получить доступ к обоим. Код, приведенный в листингах 7.13 и 7.14, демонстрирует выполнение множественного запроса, который получает данные из таблиц `Customers` и `Orders` и преобразует их в коллекцию объектов передачи данных.

Листинг 7.13. Код на C#, демонстрирующий выполнение множественного запроса

```
var myShop = new MyShopDataContext();
var customers =
from cust in myShop.Customers
from ord in cust.Orders
select new
{
Name = cust.Name,
Date = ord.OrderDate
};



foreach (var custOrd in customers)
{
Console.WriteLine(
" Name: " + custOrd.Name +
" Date: " + custOrd.Date);
}
```

Листинг 7.14. Код на VB, демонстрирующий выполнение множественного запроса

```
Dim myShop As New MyShopDataContext

Dim customers =
    From cust In myShop.Customers
    From ord In cust.Orders
    Select New With
    {
           .Name = cust.Name,
           .Date = ord.OrderDate
    }

For Each custOrd In customers
```

```
    Console.WriteLine(
        " Name: " & custOrd.Name &
        " Date: " & custOrd.Date)
Next
```

Вывод этой программы будет таким:

```
Name: Joe Date: 1/5/2010 12:00:00 AM
Name: May Date: 10/5/2010 12:00:00 AM
Name: May Date: 10/23/2010 12:00:00 AM
```

Теперь представьте себе, что код, представленный в листингах 7.13 и 7.14, находится в методе `Main` (см. листинги 7.11 и 7.12). Часть запроса, в результате которой выборка производится из нескольких таблиц, находится во втором утверждении `from`. Рассмотрим отношение "родитель/потомок" между таблицами `Customer` и `Order`, которое в данном запросе представлено переменными `cust` и `ord`. Второе утверждение `from` использует экземпляр `cust`, чтобы указать заказы для запроса, в данном случае — заказы, принадлежащие каждому клиенту. Экземпляр `ord` будет содержать каждый заказ, принадлежащий ассоциированному с ним экземпляру `cust`. Чтобы данные были полезны, проекция формируется по анонимному типу, который извлекает имя клиента и дату заказа.

В рассматриваемой базе данных было создано два заказа для `May`, один заказ для `Joe`, и ни одного заказа для `Meg`. Поскольку клиент с именем `Meg` ничего не заказывал, никаких данных для `Meg` наш запрос не выводит. Чуть далее будет показано, как добавлять в вывод запроса родительскую запись, даже если эта запись не имеет ни одной дочерней.

Запрос `select` удобен для простых запросов, но в более сложных запросах его использование усложняется. В данном случае лучшим выбором будет запрос `join`. Как и в случае с запросом `select` по множеству таблиц, запрос `join` комбинирует данные из двух таблиц, имеющих совпадающие ключи. В листингах 7.15 и 7.16 приведен пример запроса `join`, который выполняет ту же задачу, что и запрос, приведенный в листингах 7.13 и 7.14.

Листинг 7.15. Код на C#, иллюстрирующий выполнение запроса join

```
var myShop = new MyShopDataContext();

var customers =
    from cust in myShop.Customers
    join ord in myShop.Orders
        on cust.CustomerID equals ord.CustomerID
    select new
    {
          Name = cust.Name,
          Date = ord.OrderDate
    };

foreach (var custOrd in customers)
```

```
{
        Console.WriteLine(
        " Name: " + custOrd.Name +
        " Date: " + custOrd.Date);
}
```

Листинг 7.16. Код на VB, иллюстрирующий выполнение запроса join

```
Dim myShop As New MyShopDataContext

Dim customers =
    From cust In myShop.Customers
    Join ord In myShop.Orders
         On cust.CustomerID Equals ord.CustomerID
    Select New With
    {
            .Name = cust.Name,
            .Date = ord.OrderDate
    }

For Each custOrd In customers
    Console.WriteLine(
        " Name: " & custOrd.Name &
        " Date: " & custOrd.Date)
Next
```

Разница между этим запросом и запросом `select` по нескольким таблицам заключается в том, что вместо второго утверждения `from` используется утверждение `join`. Утверждение `join` идентифицирует переменную диапазона, `ord`, и оперирует на свойством `Orders` в контексте данных. Вам необходимо указать, по каким ключам выполняется объединение таблиц, указав сначала ключ в родительской таблице, `cust.CustomerID`, а затем внешний ключ в дочерней таблице, `ord.CustomerID`. Не забудьте использовать ключевое слово `equals`, потому что оператор равенства работать не будет.

Утверждения `select` по многим таблицам и `join` в SQL являются синонимами для внутренних объединений, потому что в дочерней таблице должен быть внешний ключ, который совпадает с первичным ключом в родительской таблице, для того, чтобы были возвращены какие-либо записи из родительской таблицы. Чтобы решить проблему с выводом записей из родительской таблицы, для которых нет записей в дочерней таблице, необходимо использовать внешнее объединение (outer join). Чтобы осуществить эквивалент внешнего левого объединения SQL в LINQ, вам нужно использовать стандартный оператор, который называется `DefaultIfEmpty`. Запрос, приведенный в листингах 7.17 и 7.18, выводит записи для всех клиентов, вне зависимости от того, делали они заказы или нет.

Листинг 7.17. Код на C#, выполняющий запрос, который выводит записи для всех клиентов

```
var myShop = new MyShopDataContext();

var customers =
    from cust in myShop.Customers
    join ord in myShop.Orders
             on cust.CustomerID equals ord.CustomerID
             into customerOrders
    from custOrd in customerOrders.DefaultIfEmpty()
    select new
    {
         Name = cust.Name,
        Date = custOrd == null ?
               new DateTime(1800, 1, 1) :
               custOrd.OrderDate
    };

foreach (var custOrd in customers)
{
    Console.WriteLine(
        " Name: " + custOrd.Name +
        " Date: " + custOrd.Date);
}
```

Листинг 7.18. Код на VB, выполняющий запрос, который выводит записи для всех клиентов

```
Dim myShop As New MyShopDataContext

Dim customers =
    From cust In myShop.Customers
    Group Join ord In myShop.Orders
          On cust.CustomerID Equals ord.CustomerID
          Into customersOrders = Group
    From custOrd In customersOrders.DefaultIfEmpty()
    Select New With
    {
             .Name = cust.Name,
             .Date = IIf(custOrd Is Nothing,
                     New DateTime(1800, 1, 1),
                     custOrd.OrderDate)
    }

For Each custOrd In customers
```

```
    Console.WriteLine(
        " Name: " & custOrd.Name &
        " Date: " & custOrd.Date)
Next
```

Вывод кода, приведенного в листингах 7.17 и 7.18, будет выглядеть следующим образом:

```
Name: Meg Date: 1/1/1800 12:00:00 AM
Name: Joe Date: 1/5/2010 12:00:00 AM
Name: May Date: 10/5/2010 12:00:00 AM
Name: May Date: 10/23/2010 12:00:00 AM
```

На C# — левое внешнее объединение (left outer join) выполняется точно так же, как и обычное объединение (join), за исключением двух дополнительных строк: утверждения `into` и второго утверждения `from`. Для VB левое внешнее объединение работает точно так же, как и простое объединение, за исключением трех строк: утверждения `Into`, второго утверждения `From` и ключевого слова `Group`. Утверждение `into` указывает идентификатор, который будет использоваться утверждением `from`. В утверждении `from`, использование оператора `DefaultIfEmpty` приведет к тому, что будет возвращено значение параметра типа по умолчанию, если последовательность пуста. В приведенном примере переменная, задающая последовательность — это переменная `customerOrders`, имеющая тип `Order`. Поскольку типы LINQ to SQL представляют собой классы, и `Order` является классом из коллекции `Orders`, значением по умолчанию является `null` (`Nothing` — в VB). Обратите внимание, что в приведенном примере проекция усовершенствована тернарным оператором (в VB — `immediate if`), который управляет тем, какое значение будет возвращено, если родительская запись не имеет дочерних. При выполнении левого внешнего объединения нужно убедиться в том, что вы сравниваете значение со значением по умолчанию, чтобы убедиться в том, что родительская запись не имеет дочерних, и гарантировать, что установлены допустимые значения. Код, приведенный в листингах 7.17 и 7.18, не только демонстрирует проверку значения по умолчанию, но и показывает, как следует использовать выражения в проекциях.

В дополнение к запросам LINQ, вы можете вызывать хранимые процедуры. Как уже говорилось ранее, при обсуждении работы с визуальным редактором LINQ to SQL Designer, хранимые процедуры можно перетаскивать из окна **Server Explorer** в окно редактора. Добавление хранимой процедуры добавляет метод в контекст данных. Примеры использования этого метода приведены в листингах 7.19 и 7.20.

Листинг 7.19. Пример на C#, иллюстрирующий применение метода, добавляемого в контекст данных при добавлении хранимой процедуры

```
var myShop = new MyShopDataContext();

var customers = myShop.GetCustomers();

foreach (var cust in customers)
```

```
{
        Console.WriteLine("Name: " + cust.Name);
}
```

Листинг 7.20. Пример на VB, иллюстрирующий применение метода, добавляемого в контекст данных при добавлении хранимой процедуры

```
Dim myShop As New MyShopDataContext

Dim customers As IEnumerable =
    myShop.GetCustomers()

For Each custOrd In customers
    Console.WriteLine("Name: " & custOrd.Name)
Next
```

Эти примеры выведут следующую информацию:

```
Name: Meg
Name: Joe
Name: May
```

Достаточно вызвать `myShop.GetCustomers`, и вы получите коллекцию объектов `Customer`.

Существует еще много более сложных сценариев работы с данными при помощи LINQ, но здесь они не рассматриваются, поскольку данная книга представляет собой всего лишь руководство для начинающих. Однако изложенного материала достаточно, чтобы овладеть техникой запросов. Кроме того, помимо запросов к базам данных вам потребуется выполнять и другие операции — например, ввод данных, который будет обсуждаться в следующем разделе.

> ***Совет***
>
> LINQ to SQL генерирует утверждения SQL (Structured Query Language), которые направляются в базу данных в качестве запросов. Если вы хотите просмотреть сгенерированные утверждения SQL, установите точку останова (breakpoint) на строку, следующую сразу же за запросом, и запустите программу в отладочном режиме. Когда точка прерывания будет достигнута, задержите курсор над переменной, которая содержит результаты запроса, и вы увидите утверждение SQL.

Ввод данных с помощью LINQ to SQL

Чтобы вставить в таблицу новую запись, вам необходимо создать экземпляр класса LINQ to SQL для этой таблицы, вызвать метод, осуществляющий вставку данных в таблицу, а затем — вызвать еще один метод, который фиксирует (commit) внесенные изменения. Пример, приведенный в листингах 7.21 и 7.22, демонстрирует добавление новой записи в таблицу `Customer`.

Листинг 7.21. Код на C#, иллюстрирующий добавление новой записи в таблицу

```
private static int InsertCustomer()
{
        var cust = new Customer { Name = "Jim" };
        var myShop = new MyShopDataContext();
        myShop.Customers.InsertOnSubmit(cust);
        myShop.SubmitChanges();
        return cust.CustomerID;
}
```

Листинг 7.22. Код на VB, иллюстрирующий добавление новой записи в таблицу

```
Function InsertCustomer() As Integer
    Dim cust = New Customer With
    {
            .Name = "Jim"
    }
    Dim myShop As New MyShopDataContext
    myShop.Customers.InsertOnSubmit(cust)
    myShop.SubmitChanges()
    Return cust.CustomerID
End Function
```

Как показано в листингах 7.21 и 7.22, каждое свойство коллекции, например, такое как `Customers`, имеет метод `InsertOnSubmit`, который принимает объект, тип которого совпадает с типом коллекции (в рассматриваемом примере — `Customer`). Не забывайте вызывать метод `SubmitChanges`, иначе там метод `SubmitChanges` тоже будет использоваться для обновления нового объекта, `cust`, после модификации значения `CustomerID`, которое вы считываете и возвращаете вызывающему коду.

Обновление данных с помощью LINQ to SQL

Чтобы обновить данные, вам необходимо получить объект для записи, которую требуется обновить, изменить полученный объект, а затем сохранить изменения в базе данных. Код, представленный в листингах 7.23 и 7.24, демонстрирует обновление записи в базе данных.

Листинг 7.23. Код на C#, модифицирующий запись в базе данных

```
private static void UpdateCustomer(int custID)
{
       var myShop = new MyShopDataContext();

       var customers =
           from cust in myShop.Customers
           where cust.CustomerID == custID
```

```
            select cust;

        Customer firstCust = customers.SingleOrDefault();

        if (firstCust != null)
        {
                firstCust.Name = "James";
        }
        myShop.SubmitChanges();
}
```

Листинг 7.24. Код на VB, модифицирующий запись в базе данных

```
Sub UpdateCustomer(ByVal custID As Integer)
    Dim myShop As New MyShopDataContext

    Dim customers =
        From cust In myShop.Customers
        Where cust.CustomerID = custID
        Select cust

    Dim firstCust As Customer =
        customers.SingleOrDefault()

    If (firstCust IsNot Nothing) Then
        firstCust.Name = "James"
    End If

    myShop.SubmitChanges()
End Sub
```

В запросах из листингов 7.23 и 7.24 из базы данных получается запись для клиента с именем `Jim`, объект переименовывается в `James`, и внесенные изменения сохраняются. Вызов метода `SingleOrDefault` обязателен, потому что результатом запроса LINQ to SQL является коллекция, но нам нужна только первая или только единственная запись. Существует еще операторный метод, который называется `Single`, но использование `SingleOrDefault` предпочтительнее, потому что этот метод возвращает значение по умолчанию в случае, когда по запросу не найдено ни одной записи, в то время как метод `Single` в таком случае приведет к выбрасыванию исключения (throw an exception). В коде используется утверждение `if`, чтобы защитить от возможности выбрасывания исключение, в противном случае код может выбросить исключение `NullReferenceException`, если переменная `firstCust` получит значение `null` (в VB — `Nothing`), а потому будет сделана попытка получить доступ к свойству `Name` объекта `null`. Не забывайте вызывать метод `SubmitChanges`; в противном случае изменения внесены не будут.

Итак, вы уже научились делать запросы, вводить и модифицировать данные. Теперь осталось только научиться удалять данные из базы.

Удаление данных с помощью LINQ to SQL

Чтобы удалить запись из базы данных, вам необходимо получить ссылку на объект для этой записи, вызвать метод для удаления этого объекта, а затем сохранить изменения. Примеры кода, выполняющего эти задачи, приведены в листингах 7.25 и 7.26.

Листинг 7.25. Код на C#, выполняющий удаление записей из базы данных

```
private static void DeleteCustomer(int custID)
{
        var myShop = new MyShopDataContext();

        var customers =
            from cust in myShop.Customers
            where cust.CustomerID == custID
            select cust;

       Customer firstCust = customers.SingleOrDefault();
       if (firstCust != null)
       {
                 myShop.Customers.DeleteOnSubmit(firstCust);
       }
       myShop.SubmitChanges();
}
```

Листинг 7.26. Код на VB, выполняющий удаление записей из базы данных

```
Sub DeleteCustomer(ByVal custID As Integer)
    Dim myShop As New MyShopDataContext

    Dim customers =
        From cust In myShop.Customers
        Where cust.CustomerID = custID
        Select cust

        Dim firstCust As Customer =
                customers.SingleOrDefault()

        If (firstCust IsNot Nothing) Then
             myShop.Customers.DeleteOnSubmit(firstCust)
        End If

    myShop.SubmitChanges()
End Sub
```

Этот пример аналогичен примеру с обновлением, в котором делался запрос, а затем для получения ссылки на запрошенный объект вызывался метод `SingleOrDefault`. Затем вы можете воспользоваться свойством коллекции (в данном примере это будет

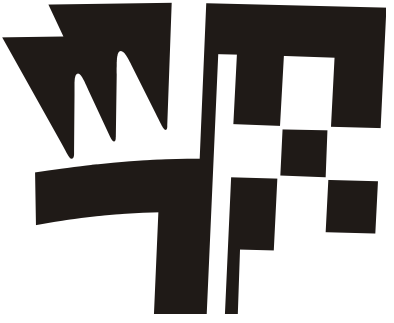
свойство `Customers`) для вызова метода `DeleteOnSubmit`. Вам необходимо будет выполнить проверку на `null` (`Nothing` — в VB), чтобы избежать ошибки `ArgumentNullException` при выполнении метода `DeleteOnSubmit` в тех случаях, когда переменная `firstCust` получает значение `null` (`Nothing` — в VB). Не забудьте вызвать метод `SubmitChanges`; в противном случае запись удалена не будет.

Наконец, сделаем заключительное замечание о трех последних разделах этой главы. Обсуждавшийся в них код осуществляет добавление в базу данных новых записей, модификацию существующих записей и удаление записей из базы данных. Обратите внимание, что методы вставки возвращают значение типа `int`, которое представляет собой идентификатор клиента (`CustomerID`). Каждый раз, когда вы осуществляете запрос к базе данных, вам обычно одновременно требуется получить поле идентификатора записи. Причина этого заключается в то, что идентификатор (ID) является уникальным для этой записи, и последующие действия можно выполнять с его использованием. Как операции модификации, так и методы удаления в приведенных примерах принимали параметр типа `int`, который использовался для поиска в базе данных нужной записи. Опять же, использование ID гарантирует, что возвращена будет только одна запись, и именно поэтому я был настолько уверен, вызывая метод `SingleOrDefault`. Поскольку эта глава посвящена работе с данными, я пока умышленно не показываю, как программа обрабатывает этот идентификатор. Однако в последующих главах эти идентификаторы будут активно использоваться, и вам будет продемонстрировано, как следует строить пользовательские интерфейсы. Обратите внимание, как код пользовательского интерфейса обращается с идентификаторами и как он их использует при взаимодействии с базой данных. В последующих главах будет приведено множество различных примеров, которые вы сможете впоследствии использовать в своих программах. Впрочем, часть из представленных примеров будет вам уже знакома.

Заключение

В этой главе было продемонстрировано, как работать с базами данных средствами VS, в том числе — создавать таблицы, отношения и хранимые процедуры. Далее был описан базовый синтаксис запросов LINQ. После прочтения этой главы вы должны приобрести навыки использования LINQ to SQL и научиться работать с классами и методами с помощью визуального конструктора. Наконец, в заключение было рассказано о том, как следует создавать, читать, обновлять и удалять данные с помощью LINQ to SQL.

В данной главе для демонстрации описываемых концепций работы с данными использовалось консольное приложение. Благодаря этому можно было полностью сосредоточиться на работе с данными и по минимуму отвлекаться на все остальное. Однако в реальной жизни приложения используют графические пользовательские интерфейсы. В остальных главах этой книги будет показано, как создаются графические пользовательские интерфейсы, причем будет приведено много примеров использования LINQ to SQL. В следующей главе будет рассмотрен процесс разработки приложения WPF.



ЧАСТЬ III

Разработка приложений с помощью VS 2010

Глава 8

Построение настольных приложений с помощью WPF

В этой главе будут описаны следующие ключевые концепции:

- Выполнение настройки экрана;
- Использование элементов управления;
- Работа с данными через пользовательский интерфейс.

Windows Presentation Foundation (WPF) представляет собой технологию .NET, предназначенную для построения настольных приложений. В результате построения приложения WPF вы получите исполняемый *.exe-файл, который можно запускать непосредственно на вашем компьютере или распространять на все компьютеры, где установлена платформа .NET, и запускать их там. С помощью WPF можно снабдить ваши приложения графическим пользовательским интерфейсом (Graphical User Interface, GUI), который упрощает для пользователей работу с вашими программами. В данной главе будет показано, как создать GUI, и будет дано описание элементов управления (controls), таких, как кнопки (**Button**) и текстовые поля (**TextBox**), которые вы можете размещать на экране. Кроме того, будет продемонстрирован захват событий от элементов управления, который позволяет добавлять в состав приложения код, поведение которого зависит от пользовательского ввода. Так как большинство приложений работают с данными, эта глава является логическим продолжением *главы 7* и показывает, каким образом следует связывать данные с элементами управления GUI.

В данной главе мы рассмотрим, как создать GUI на базе WPF с помощью VS Designer. При этом иногда вам потребуется работать на более низком уровне и манипулировать кодом XAML (произносится как "Zammel"), который определяет GUI. XAML — это формат XML, который используется технологиями WPF и Silverlight для определения GUI. Чтобы подробнее ознакомиться с форматом XAML, в состав этой книги были включены два приложения: *приложение 1, "Введение в XML"*, и *приложение 2, "Введение в XAML"*. Если вы еще не знакомы с форматом XML, начните с Приложения 1. Однако, если вы уже хорошо знакомы с базовым синтаксисом XML, можно сразу же начинать с приложения 2. Я постараюсь объяснить WPF таким образом, чтобы любая информация в формате XAML была понятна в контексте изложения. Однако если вы чувствуете себя неуверенно, путаетесь и испытываете сложности с пониманием, обратитесь к упомянутым приложениям, и это поможет вам разобраться в сути проблемы. Разобравшись с XAML, вы сможете

вернуться к чтению этой главы и продолжить его, начиная со следующего раздела, в котором рассказывается о том, как следует начинать работу над проектом WPF.

Начало работы над проектом WPF

В *главе 5* рассказывалось о создании и построении проектов. Пример, приведенный там, демонстрировал создание, разработку и сборку консольного приложения. Вся информация, приводившаяся в той главе, в общих чертах применима и к большинству других типов приложений. Материал, излагаемый в этом разделе, базируется на том, что вы уже знаете после прочтения *главы 5*, и дополняет этот материал сведениями, характерными для приложений WPF. Чтобы начать работать, откройте окно **New Project**, в качестве типа проекта и выберите опцию **WPF Application**. Затем дайте имя новому проекту, укажите его местоположение и имя решения (solution name). В примерах из этой главы я называю все примеры именем **MyShop**, поскольку приведенное приложение будет представлять собой усовершенствованный вариант разработки, которая рассматривалась в *главе 7* — приложение для обработки транзакций клиентов, делающих покупки в вашем магазине. На рис. 8.1 показано новое приложение WPF, открытое в VS. На данном экране вы можете видеть окна **Toolbox**, **Designer** и **Solution Explorer**. Окно **Toolbox** содержит элементы управления (controls), которые представляют собой готовые к применению элементы пользовательского интерфейса (UI), например, такие как командные кнопки (**Button**) или текстовые поля (**Textbox**). Эти элементы можно просто перетаскивать мышью в окно визуального редактора.

> ***Примечание***
>
> Существует еще одна технология .NET, предназначенная для создания графических приложений, Windows Forms. Это — более старая технология, поэтому в данной книге она не обсуждается. Основной технологией для разработки GUI-приложений, которой принадлежит будущее, является WPF, и основная цель данной книги — помочь вам освоить именно ее.

Визуальный редактор (Designer) позволяет сконструировать пользовательский интерфейс приложения. Его окно разделено на две области — панель визуального редактора **Design** в верхней части окна и панель **XAML** — в нижней. Панель **Design** предназначена для визуального проектирования пользовательского интерфейса — элементов управления (controls) и их расположения на экране. Редактор XAML позволяет работать с представлением элементов управления в формате XML. Панели **Design** и **XAML** взаимосвязаны, потому что изменение одной влечет за собой изменение другой. Например, если вы добавите кнопку (**Button**) в область **Design**, то вы увидите XML-представление этой кнопки в панели XAML.

Аналогичным образом, если вы добавите в область **XAML** элемент **TextBox**, то в области **Design** вы увидите визуальное представление текстового поля. Для манипулирования окнами в вашем распоряжении имеются различные элементы управления. Как панель **Design**, так и панель **XAML** имеют инструменты для масштабирования. Инструментом масштабирования для области **Design** является ползунок, расположенный в верхнем левом углу. Для области **XAML** элементом масштабирования

является выпадающий список в левом нижнем углу панели. Кроме того, масштабировать панели **Design** и **XAML** можно, вращая колесико мыши. В верхнем правом углу редактора XAML (и, соответственно, в нижнем правом углу панели **Design**), есть элементы управления, позволяющие переключаться между горизонтальным и вертикальным разбиениями окна (). Нажатие кнопки с изображением "шеврона" () позволяет "сворачивать" панель кода XML. Значок "разбиения" () под значком "шеврона" позволяет разбить окно редактора XAML на две панели, если вы перетянете его мышью вниз. Значок в виде двунаправленной стрелки между вкладками **Design** и **XAML** () позволяет переключать эти панели так, чтобы они менялись местами. Символ курсора на границе разделителя между панелями **Design** и **XAML** позволяет менять размеры каждого из окон.

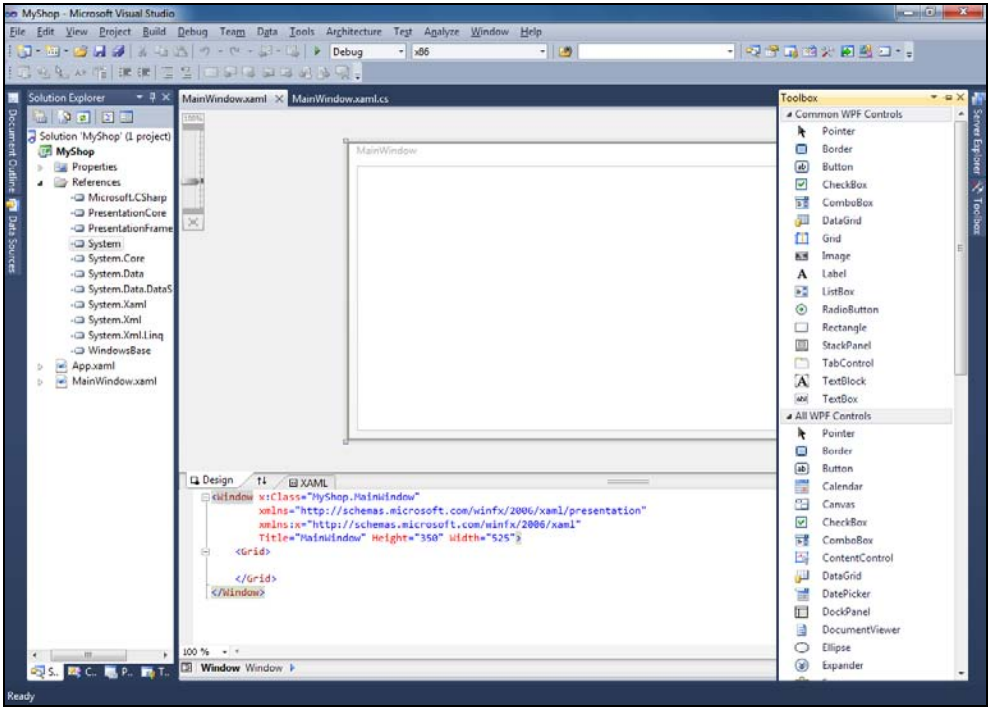


Рис. 8.1. Новый проект приложения WPF

Изучение расположения элементов (Layout)

Расположение элементов (layout) определяет, каким образом вы можете размещать в окне элементы управления и менять их размеры. Окна WPF и элементы управления, расположенные в них, имеют свойство **Content** (которое иногда может называться иначе). В некоторых ситуациях, например, в случае с элементом управления **Button**, содержимое может представлять собой текст. Однако в преобладающем большинстве случаев необходимо разместить в окне большое количество

элементов управления. В данном разделе мы сосредоточимся на размещении элементов управления в окнах. Элемент **Window** имеет свойство **Content**, которое принимает только один элемент управления. Это должен быть элемент, задающий макет (layout), на котором мы и сосредоточим внимание в данном разделе.

В состав WPF входят несколько вариантов элемента управления, задающего макет (layout), в том числе: **Grid**, **StackPanel**, **DockPanel**, **WrapPanel** и **Canvas**. По умолчанию VS генерирует окно с элементом **Grid**. Однако вы можете заменить элемент **Grid** любым другим элементом управления, который соответствует вашим потребностям. В данном разделе будет показано, как следует применять каждый из доступных элементов управления.

Макет *Grid*

Каждый раз, когда вы создаете новый проект WPF, VS добавляет элемент **Grid**. Элемент формата **Grid** представляет собой сетку, дающую вам возможность сгенерировать набор строк и столбцов, в которых могут содержаться другие элементы управления. В состав сетки (**Grid**) с помощью визуального конструктора можно добавлять элементы. Это делается щелчком мыши в середине окна. На рис. 8.2 показан столбец, добавленный к элементу **Grid.**

Тонкая вертикальная линия в середине окна представляет собой новую границу между двумя столбцами.

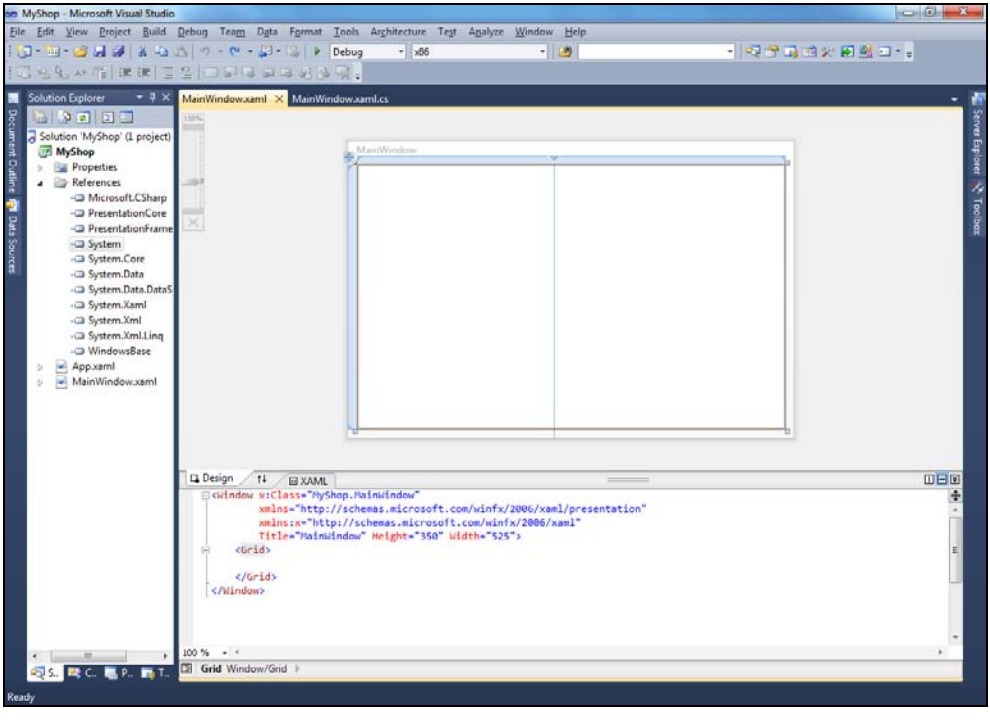


Рис. 8.2. Добавление столбцов и строк в состав сетки (Grid)

После щелчка мышью в окне, вы увидите две жирных линии в левой верхней части окна. Если вы задержите курсор над верхней границей, VS отобразит вертикальную линию, которая перемещается влево и вправо по мере того, как вы перемещаете курсор мыши вдоль верхней границы. Если сделать то же самое, наведя курсор на левую границу, вы сможете добавлять в состав сетки новые строки. Это — очень быстрый способ добавления строк и столбцов в состав сетки.

Стрелка на границе сетки позволяет перемещать границы строк и столбцов. Вы можете удалить строку или столбец, выбрав стрелку на границе элемента **Grid** и перетащив ее мышью за пределы окна.

> ***Внимание!***
>
> Не следует нажимать клавишу <DELETE> при выделенной границе, потому что таким образом вы удалите ваш элемент сетки (**Grid**), на конфигурирование которого вы уже успели затратить время. Если вы хотите удалить столбец или строку, подхватите мышью границу, которую требуется удалить, и перетащите ее за пределы окна.

Как только вы создадите столбцы и строки, вы сможете продолжить работу по индивидуальной настройке и определить, какое пространство должно быть отведено данной строке или столбцу. Есть три варианта настройки по изменению размеров: фиксированный (**Fixed**), взвешенный (**Weighted**) и автоматический (**Auto**). Чтобы установить любую из трех опций, задержите курсор над границей столбца или строки, и VS отобразит панель регулирования размеров, как показано на рис. 8.3.

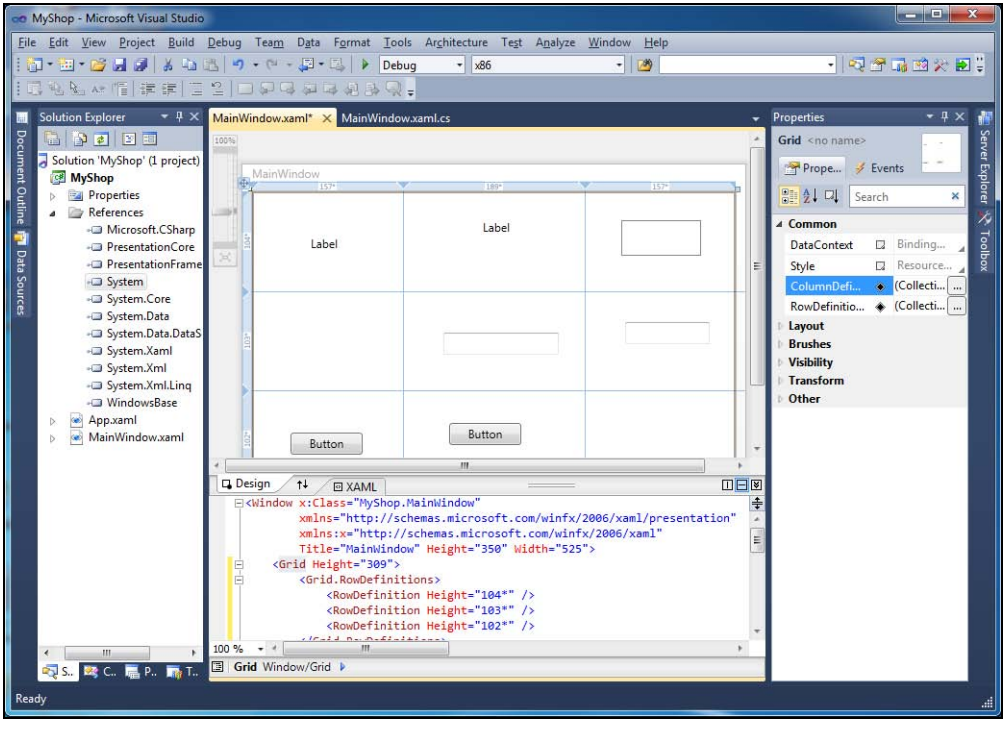


Рис. 8.3. Опции изменения размеров строк и столбцов

Значок "ромба" (diamond) слева соответствует опции **Fixed**, при которой размер останется неизменным. Значок "звездочки" в центре означает установку взвешенных пропорций (weighted proportion), при выборе которой размер останется прежним по отношению к размерам остальных столбцов. Справа находится значок **Auto**, при выборе которого размеры столбца определяются его содержимым, и ему отводится то пространство, которое остается после задания размеров других столбцов. После того как к элементу **Grid** будет добавлено информационное наполнение, вы сможете поэкспериментировать с этими настройками, чтобы задать желаемую компоновку.

На рис. 8.3 надо обратить внимание на следующий аспект: на границах каждого элемента **Grid** для каждой строки и каждого столбца указаны числа, которые задают для строк и столбцов точный размер в пикселах.

На рис. 8.3 в правой части показано окно **Properties**, где вы можете выбрать коллекции столбцов (**Column**) и строк (**Row**) и выполнить их индивидуальную настройку. Кроме того, на рис. 8.3 показаны элементы управления, которые были добавлены в состав элемента **Grid** и помещены в каждую ячейку сетки. Еще один популярный вариант размещения — это компоновка **StackPanel**, рассматриваемая в следующем разделе.

Макет *StackPanel*

Вариант размещения **StackPanel** идеален для случая, когда вы хотите разместить элементы управления "стопкой", один поверх другого.

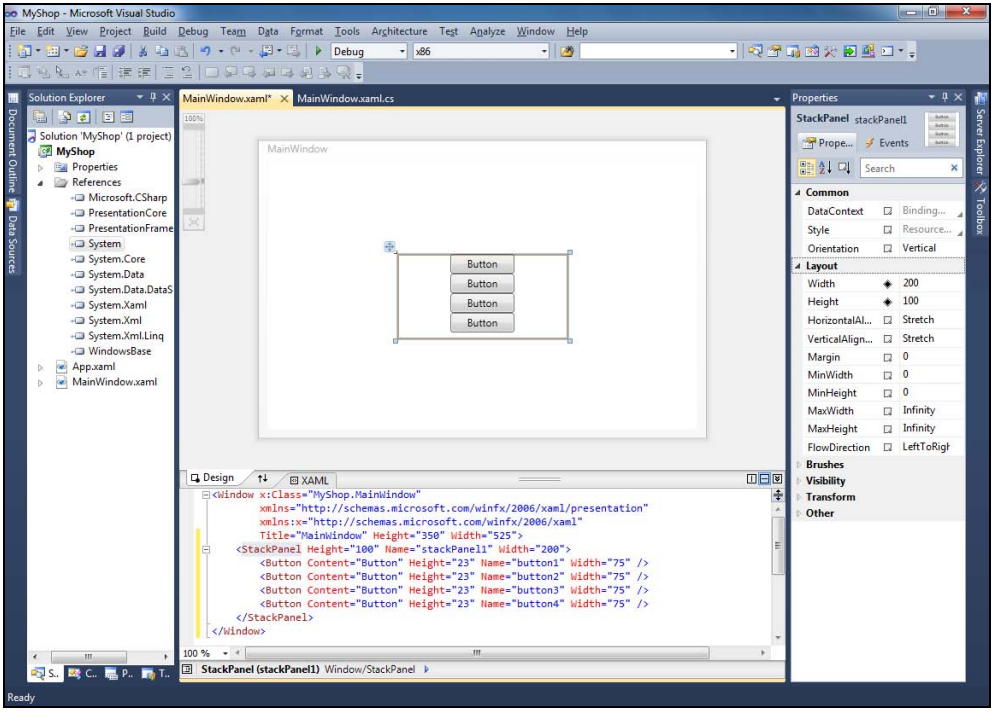


Рис. 8.4. Использование макета **StackPanel**

Вы можете использовать размещения **StackPanel**, перетащив мышью элемент **StackPanel** из панели **Toolbox** в область визуального конструктора. Если вы хотите использовать **StackPanel** в качестве основного макета, вы можете выбрать сетку, которая предлагается по умолчанию для всех новых проектов, и удалить элемент **Grid**. На рис. 8.4 показан элемент типа **StackPanel**, который содержит несколько кнопок.

На рис. 8.4, показано, что то, куда вы пытаетесь поместить кнопки, не имеет значения, потому что **StackPanel** всегда будет помещать их одну за другой. В дополнение к вертикальному расположению, **StackPanel** может размещать элементы управления и горизонтально. Для этого достаточно изменить значение свойства **Orientation** на **Horizontal** (рис. 8.4). В следующем разделе будет показано, как пристыковать элементы управления к краям контейнера.

Макет *DockPanel*

Вы уже знакомы с тем, как VS позволяет пристыковывать и отстыковывать окна в пределах рабочей зоны приложения. Благодаря этому вы можете организовать экран таким образом, чтобы иметь возможность одновременного доступа к большому количеству инструментальных средств. С помощью элемента управления **DockPanel** вы можете добиться, чтобы разработанное вами приложение вело себя точно так же.

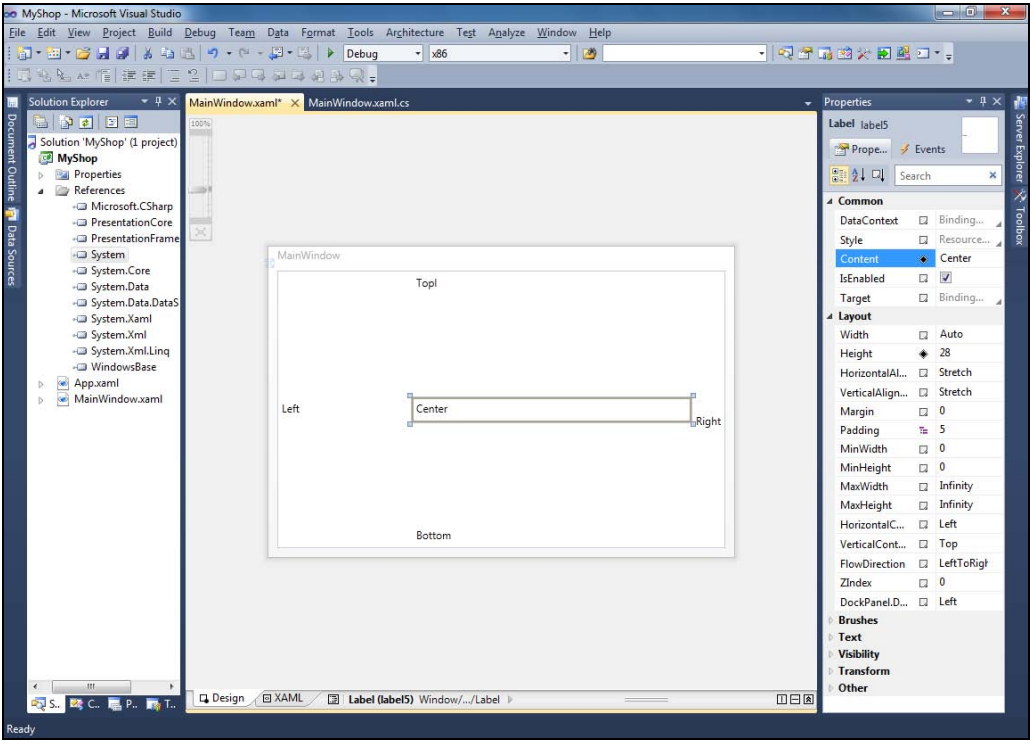


Рис. 8.5. Макет окна **DockPanel**

Для начала удалите предлагаемый по умолчанию макет **Grid**, затем перейдите в окно **Toolbox** и перетащите оттуда мышью элемент **DockPanel**, поместив его в рабочую область визуального редактора. Элемент **DockPanel** инициализируется значениями высоты (**Height**) и ширины (**Width**). Чтобы изменить значения, предлагаемые по умолчанию, выберите элемент **DockPanel**, откройте окно **Properties** и очистите поля свойств **Height** и **Width**. Для этого щелкните по соответствующему свойству правой кнопкой мыши и из раскрывшегося контекстного меню выберите команду **Reset Value**. В результате удаления свойств **Height** и **Width**, элемент **DockPanel** развернется на все окно. На рис. 8.5 показан элемент **DockPanel** с элементами управления **Label** в каждой из стыковочных позиций.

Каждый раз, когда вы перетаскиваете мышью элемент управления и помещаете его в область макета **DockPanel**, этот элемент по умолчанию будет оказываться в центре окна. Чтобы указать, к какой из стыковочных позиций должен быть прикреплен новый элемент управления, откройте окно **Properties** и установите свойство **DockPanel.Dock**. При добавлении нового элемента управления, этот вновь добавленный элемент становится центральным, а предыдущий будет пристыкован к той стороне элемента **DockPanel**, которую вы указали в свойстве **DockPanel.Dock**. Теперь рассмотрим следующий элемент управления, **WrapPanel**.

Макет *WrapPanel*

Когда элементы управления должны следовать один за другим и последующие должны продолжаться с новой строки, выберите макет **WrapPanel**.

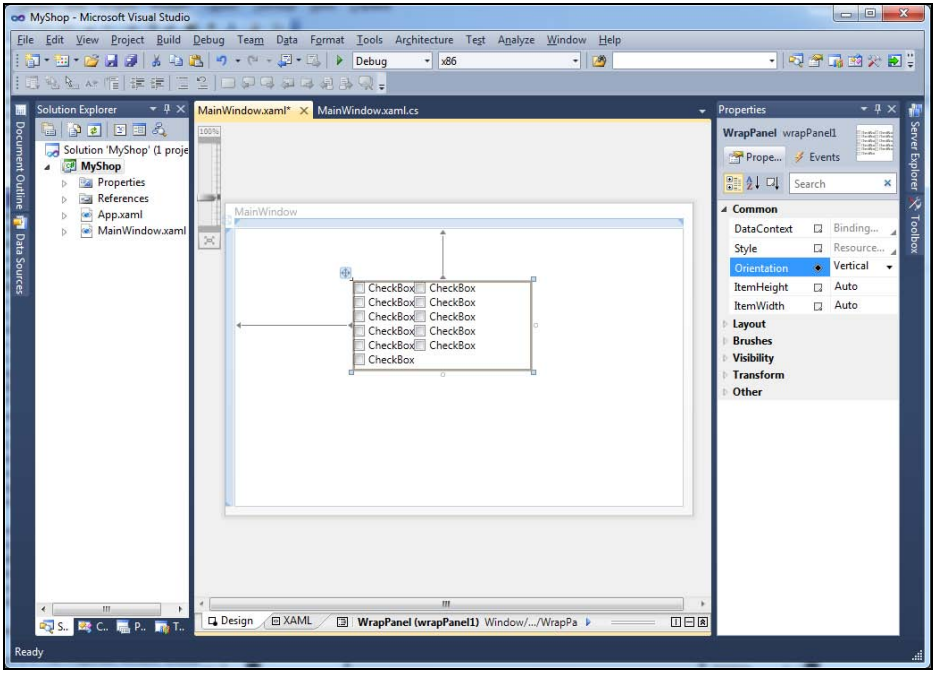


Рис. 8.6. Макет окна **WrapPanel**

Примеры ситуаций, когда этот вариант оформления будет полезен и удобен, включают добавление элементов управления, которые должны содержать текст, а также те случаи, когда вам необходимо просматривать элементы управления последовательно, один за другим. На рис. 8.6 показан вариант оформления **WrapPanel** с несколькими добавленными элементами управления типа **CheckBox**.

На рис. 8.6 продемонстрирован пример того, как надо располагать группу элементов управления так, чтобы оптимально заполнить имеющееся свободное пространство проектируемого окна. В случае с элементами управления типа **CheckBox**, свойство **Orientation** для **WrapPanel** установлено на **Vertical** (по умолчанию предлагается значение **Horizontal**). Когда элементы управления типа **CheckBox** заполнят весь вертикальный столбец, то остальные элементы этого типа перейдут в следующий. Поскольку размеры элементов управления **CheckBox** одинаковы, вы имеете единообразный макет с равномерным их распределением, причем с помощью макета **WrapPanel** этого добиться проще, чем с помощью макета **Grid** или какого-либо еще варианта макета. Наконец, последний вариант макета, который мы рассмотрим, будет вариант **Canvas**.

Макет *Canvas*

Существуют ситуации, когда вам может потребоваться явно задать расположение элементов управления. Если вы создаете приложение, которое должно строить диаграммы или если вы просто желаете точно указать расположение элементов управления, то макет **Canvas** — это именно то, что вам и требуется.

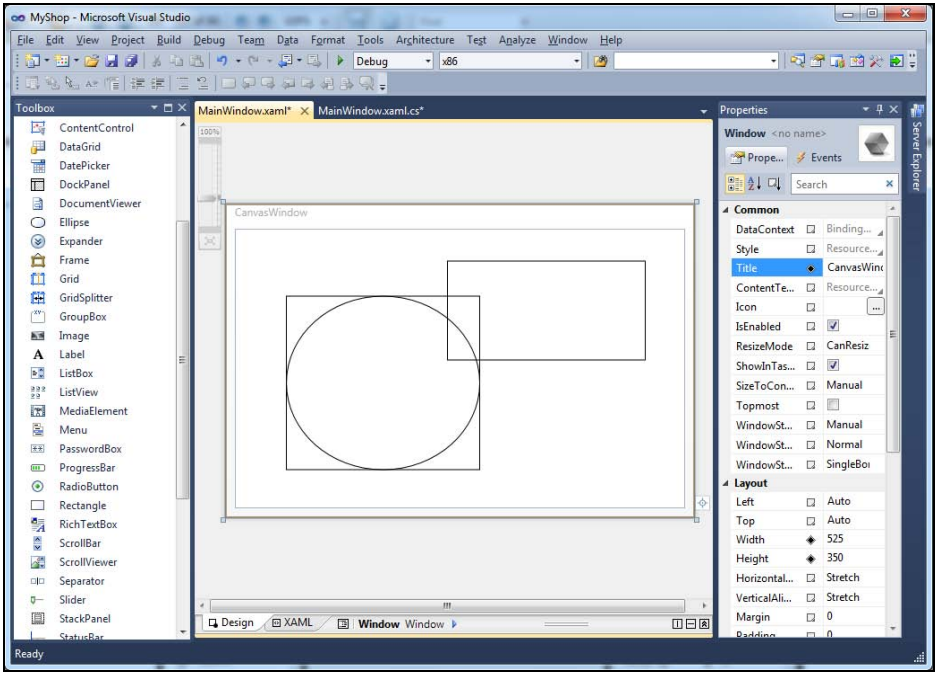


Рис. 8.7. Макет **Canvas**

В примере на рис. 8.7 на поверхность макета **Canvas** из окна **Toolbox** были перетащены мышью элементы управления **Rectangle** и **Ellipse**. Обратите внимание на свойства **Canvas.Left**, **Canvas.Top**, **Width** и **Height** в окне **Properties**, демонстрирующие абсолютное расположение выбранного элемента управления.

Теперь, научившись использовать элементы управления, задающие макет окна, перейдем к более подробному обсуждению элементов управления WPF в целом и рассмотрим их применение в ваших приложениях.

Использование элементов управления WPF

WPF предлагает множество типовых элементов управления, из которых вы, как из кирпичиков, можете строить пользовательский интерфейс создаваемого приложения. В данном разделе эти элементы управления разбиты по категориям, в том числе, таким, как текстовые элементы управления (text), элементы управления для выбора опций (selection), контейнеры (containers), информационные элементы управления (information), геометрические фигуры (shapes) и украшения (decorators). Элементы управления для работы с данными исключены из этого списка умышленно, так как они будут рассматриваться отдельно, в специально отведенном для этой цели разделе "Работа с данными в WPF". Но прежде чем начинать подробное обсуждение каждого из элементов управления, необходимо привести обзорную информацию о том, как работает среда VS с элементами управления.

Основные окна для работы с элементами управления

Работая с элементами управления, вы будете выполнять свои задачи, пользуясь четырьмя различными окнами: **Toolbox**, **Solution Explorer**, **Designer** и **Properties**. В предшествующих главах этой книги уже было рассказано о том, как открывать эти окна и получать доступ к их содержимому; но для удобства, краткая сводка этих окон, а также соответствующих им команд меню и клавиатурных комбинаций приведена в табл. 8.1.

Таблица 8.1. *Основные окна для работы с элементами управления*

Окно	Команды меню	Клавиатурная комбинация
Toolbox	**View** | **Toolbox**	<CTRL>+<W>, <X>
Solution Explorer	**View** | **Solution Explorer**	<CTRL>+<W>, <L>
Designer	Выполните двойной щелчок мышью по файлу *.xaml в окне **Solution Explorer**	<SHIFT>+<F7>
Properties	**View** | **Other Windows** | **Properties window**	<CTRL>+<W>, <P>

В окне **Toolbox** вы найдете множество элементов управления, разделенных на две группы, из которых первая содержит наиболее распространенные элементы управления WPF (**Common WPF Controls**), благодаря чему для вас упрощается

процедура поиска элементов управления, которые вы используете наиболее часто. Что касается второй группы, **All WPF Controls**, то она содержит полный список всех элементов управления WPF.

Использование окна **Design**, в котором осуществляется визуальное проектирование, уже было проиллюстрировано в предыдущем примере, когда обсуждалось использование элементов управления, задающих общий вид макетов окна. Открыть окно визуального редактора можно, выполнив двойной щелчок мышью по файлу *.xaml в окне **Solution Explorer**. Чтобы добавить элемент управления в окно визуального конструктора, выберите нужный вам элемент управления в окне **Toolbox** и перетащите его мышью в область редактирования. На рис. 8.8 показана командная кнопка (**Button**), которая была добавлена в область визуального редактирования.

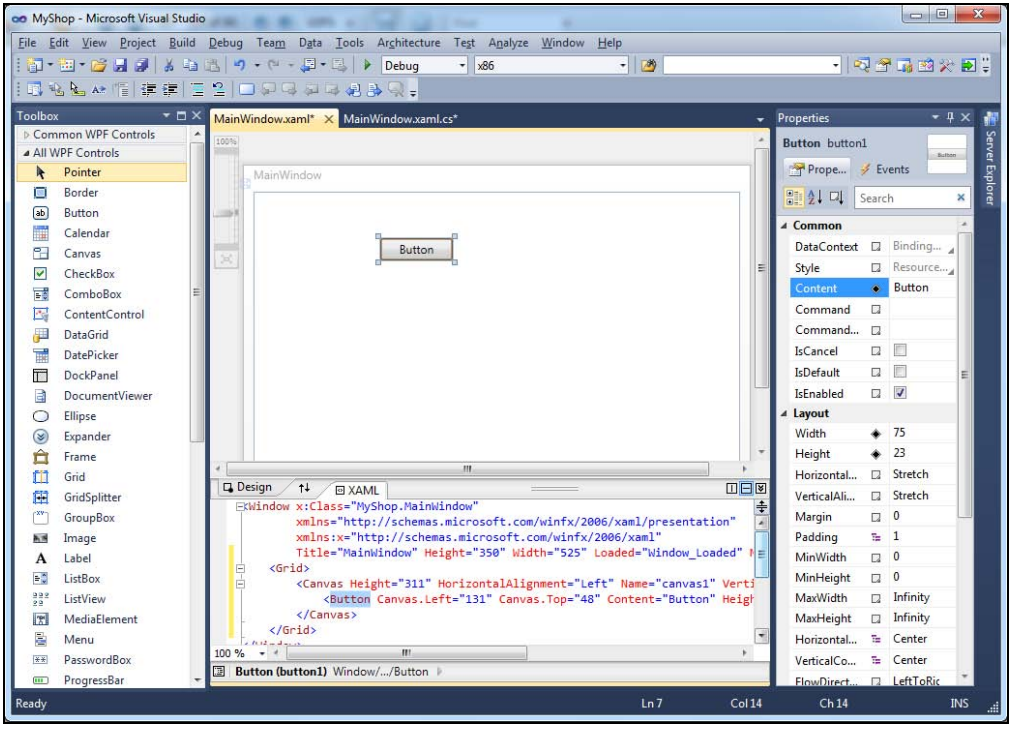


Рис. 8.8. Добавление элемента управления в область визуального конструирования (**Design**)

На рис. 8.8 показано окно **Toolbox**, в котором выбран элемент управления **Button**. В области визуального конструирования (окно **Design**) показан элемент управления **Button**, который был перетащен в него мышью. На практике, вы будете добавлять этот элемент управления на различные макеты таким образом, чтобы нужным образом расположить его на экране.

Под областью визуального проектирования описание элемента управления **Button** появляется в окне кода XAML для данного окна. Если вы испытываете затруднения с пониманием кода XAML, ознакомьтесь с *приложением 2*. Атрибуты

элемента управления **Button** в коде XAML совпадают со свойствами, заданными для этого элемента в окне **Properties**.

Совет

Научиться быстрому конструированию пользовательского интерфейса с помощью визуального конструктора очень важно, потому что это повышает производительность труда. Но не менее важно и научиться читать код XAML, ассоциированный с этим окном. Когда вы с уровня начинающего перейдете на уровень опытного программиста, вы обнаружите, что бывают ситуации, когда одними только средствами визуального конструктора нельзя полностью контролировать все нюансы визуального представления пользовательского интерфейса вашей программы. Хорошим подходом в данном случае будет самостоятельное экспериментальное исследование — попробуйте задавать и модифицировать свойства элементов управления, добавляемых в окно визуального редактора из окна **Toolbox**, а затем исследовать сгенерированный XAML-код.

Установка свойств

Окно **Properties** отображает все варианты, которые можно использовать для конфигурирования элемента управления. Для элементов управления, представляющих собой командные кнопки (**Button**), вам наверняка захочется в первую очередь модифицировать свойство **Content**, чтобы текстовая метка на этой кнопке поясняла ее предназначение. В рассматриваемом примере мы будем считать, что новая кнопка предназначается для создания записи о новом заказе клиента. Следовательно, зададим для свойства **Content** значение **New Order**.

Обработка событий

В дополнение к свойствам (properties), вы можете контролировать события, связанные с элементом управления. На рис. 8.9 показано содержимое вкладки **Events** окна **Properties**.

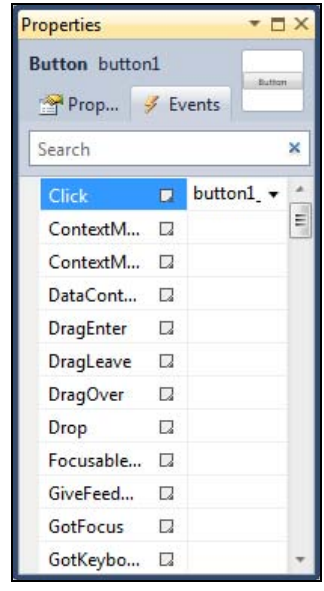


Рис. 8.9. Содержимое вкладки **Events** окна **Properties**

Элементы управления могут иметь десятки различных событий, которые помогают управлять их поведением в приложении. Некоторые события, например, **Click**, используются часто и широко, в то время как другие, скажем такие, как **Drag Over**, поддерживают только уникальные ситуации — например, такие как перетаскивание мышью (технология drag and drop), о которой вы, возможно, даже и не заботитесь. Чтобы обработать событие, вы можете выполнить двойной щелчок мышью по любому из событий в окне **Properties**, и VS "свяжет" это событие с методом-обработчиком, имеющим имя по умолчанию.

Поскольку событие **Click** распространено очень широко, я проиллюстрирую работу с событиями именно на его примере. Вы можете реализовать обработчик для

события **Click**, выполнив двойной щелчок мышью по событию **Click** в окне **Properties** на вкладке **Events**. Когда вы выполните двойной щелчок, VS откроет файл с именем MainWindow.xaml.cs (в предположении того, что окно, с которым вы работаете, называется MainWindow.xaml). Файл MainWindow.xaml.cs называется файлом, обеспечивающим поддержку программной логики кода (code-behind file)[1] и представляет собой именно тот файл, куда следует добавлять обработчики событий. Кроме того, VS также создаст "скелет" метода в файле MainWindow.xaml.cs, который обрабатывает событие **Button Click**, как показано в листингах 8.1 (код на C#) и 8.2 (код на VB).

> ***СОВЕТ***
>
> Элементы управления имеют события по умолчанию. Значимость событий по умолчанию заключается в том, что если вы выполните двойной щелчок мышью на элементе управления в окне визуального конструктора, VS сгенерирует обработчик события для события по умолчанию. Говоря более конкретно, представьте себе элемент управления типа "кнопка" (Button), для которой событием по умолчанию должно быть событие **Click**. Если вы выполните двойной щелчок мышью по элементу управления **Button** в окне визуального конструктора, то VS сгенерирует обработчик события для события **Click**.

Листинг 8.1. Код на C#, содержащийся в файле поддержки кода (code-behind file)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace ControlsCS
{
        /// <summary>
        /// Interaction logic for MainWindow.xaml
        /// </summary>
        public partial class MainWindow : Window
```

[1] В официальной документации Microsoft этот код называется выделенным (code-behind). Модель страницы с выделенным кодом (в отличие от однофайловой модели) позволяет поместить разметку в один файл (ASPX), а программный код — в другой. Имя файла с кодом зависит от используемого языка программирования. Подробнее см. **http://msdn.microsoft.com/en-us/library/015103yb.aspx** и **http://msdn.microsoft.com/en-us/library/015103yb.aspx**. — *Прим. перев.*

```
        {
                public MainWindow()
                {
                        InitializeComponent();
                }

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

Листинг 8.2. Код на VB, содержащийся в файле поддержки кода (code-behind file)

```
Class MainWindow

    Private Sub Button1_Click(
            ByVal sender As System.Object,
            ByVal e As System.Windows.RoutedEventArgs)
        Handles Button1.Click

    End Sub

End Class
```

Только что созданный обработчик события **Click** — это метод `button1_Click` (в VB — `Button1_Click`), который в листингах 8.1 и 8.2 выделен полужирным шрифтом. В *главе 4* мы уже рассматривали делегатов и события — для лучшего понимания излагаемого материал ее рекомендуется перечитать. Обратите внимание, как код на VB демонстрирует другой способ обработки событий в VB, явно указывая `Handles Button1.Click`. В принципе, когда пользователь щелкает мышью по кнопке, которая называется `button1`, будет вызываться этот обработчик. Это иллюстрирует концепцию программирования, управляемого событиями (event-driven programming), при котором вы пишете обработчики наподобие `button1_Click`, которые ведут себя в зависимости от действий пользователя. В дополнение к созданию обработчика событий для файла, осуществляющего поддержку кода (файла с выделенным кодом, code-behind), VS добавляет имя метода к событию **Click** на вкладке **Events** окна **Properties** (см. рис. 8.9).

В дополнение к созданию метода обработчика и присвоению этому методу имени на вкладке **Events** в окне **Properties**, VS добавляет этот метод в качестве атрибута кнопки (**Button**) в окне XAML, как показано в листинге 8.3. XAML не зависит от языка программирования и ведет себя одинаково, вне зависимости от того, программируете ли вы на C# или на VB.

Листинг 8.3. Код XAML, сгенерированный VS для кнопки

```
<Button Content="Button" Height="23"
        HorizontalAlignment="Left" Margin="76,43,0,0"
        Name="button1" VerticalAlignment="Top" Width="75"
        Click="button1_Click" />
```

Обратите внимание на то, что имя метода соответствует специальному соглашению об именовании: *controlName_Event*. Часть имени, представляющая собой строку *controlName*, является именем элемента управления (в нашем примере — `button1`), а *event* представляет собой имя обрабатываемого события. Проблема в данном случае заключается в том, что имя `button1` не несет никакой смысловой нагрузки. Впоследствии, когда вы вернетесь к работе над этим кодом, вы запутаетесь, пытаясь выяснить, что делают методы с именами `button1_Click`, `button2_Click` и т. п. Чтобы решить эту проблему с именованием, необходимо давать своим элементам управления осмысленные имена, причем это надо делать до того, как вы решите сделать с ними что-то еще.

Чтобы решить проблему, вернитесь на вкладку **Events** окна **Properties**. Не забудьте выделить элемент управления (**Button**) в окне визуального конструктора (**Designer**). В верхнем левом углу окна **Properties** есть поле, которое содержит идентификатор элемента управления, который вам нужно изменить, присвоив ему вместо `button1` какое-нибудь имя, несущее смысловую нагрузку. Например, если кнопка предназначена для того, чтобы ввести информацию о новом заказе для клиента, то кнопку можно назвать `NewOrderButton`. Затем удалите обработчик событий, который был присвоен событию `Click` для кнопки. Эти изменения, внесенные в окне **Properties**, отражены на рис. 8.10. Обратите внимание, что теперь как идентификатор, так и обработчик события имеют осмысленные имена.

После того как существующий обработчик события будет удален, а элементу управления присвоен новый идентификатор, выполните снова двойной щелчок мышью по событию. Код обработчика событий, которые сгенерирует для вас VS, показан в листингах 8.4 и 8.5.

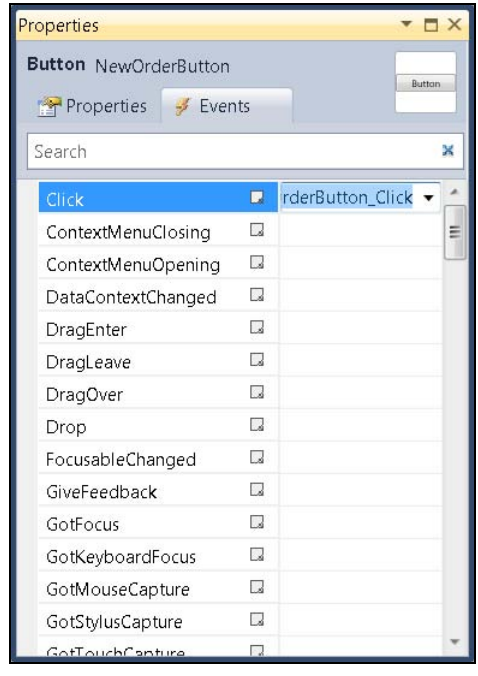


Рис. 8.10. После внесения изменений как идентификатор нового элемента управления, так и обработчик события получили осмысленные имена

Листинг 8.4. Код нового обработчика событий на C#, сгенерированный для кнопки NewOrderButton

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

Листинг 8.5. Код нового обработчика событий на VB, сгенерированный для кнопки NewOrderButton

```
Class MainWindow

    Private Sub Button1_Click(
            ByVal sender As System.Object,
            ByVal e As System.Windows.RoutedEventArgs)
    End Sub

    Private Sub NewOrderButton_Click(
            ByVal sender As System.Object,
            ByVal e As System.Windows.RoutedEventArgs) Handles
NewOrderButton.Click

    End Sub

End Class
```

В коде, который приведен в листингах 8.4 и 8.5, представлены оба обработчика события — как старый, `button1_Click` (`Button1_Click` — для VB), так и новый — `NewOrderButton_Click`. Возможно, вы удивитесь, почему обработчик события `button1_Click` не был удален, когда вы удалили его из события `Click` в окне **Properties**. Однако на то есть веская причина. Представьте себе, что вы уже написали код для этого обработчика. VS не удалит этот код именно из-за того, чтобы вы могли подстраховаться. Теперь у вас есть оба обработчика, а это значит, что вы с легкостью можете скопировать код обработчика `button1_Click` в обработчик `NewOrderButton_Click`, и только после этого удалить старый обработчик `button1_Click`. До сих пор мы еще не написали ни строчки кода, но этим мы займемся в следующем разделе.

Написание кода обработчиков событий

Одна из основных задач, которые требуется выполнить, когда пользователь нажимает кнопку, является открытие нового окна. Первое, что вам требуется для этого сделать — это добавить новое окно. Чтобы выполнить эту работу, вам потребу-

ется открыть окно **Solution Explorer**, выполнить щелчок правой кнопкой мыши по проекту, с которым вы работаете, и из контекстного меню выбрать команды **Add | New Item**. Затем в раскрывшемся новом окне **Add New Item** выберите опцию **Window (WPF)**, назовите это окно NewOrder.xaml и щелкните мышью по кнопке **Add**.

В результате вновь созданное окно будет открыто в визуальном редакторе (Designer).

> ***Совет***
>
> В контекстном меню проекта, выводимом при выборе команд **Add | New Item**, есть опция **Window**, и это позволяет сэкономить парочку щелчков мышью при создании нового окна.

Когда откроется новое окно **Designer**, вы можете быстро открыть окно выделенного кода (code-behind), нажав клавишу <F7>. В этом окне вы увидите код, приведенный в листингах 8.6 и 8.7.

Листинг 8.6. Выделенный код (code-behind) для нового окна (для проекта C#)

```
public partial class NewOrder : Window
{
        public NewOrder()
        {
                InitializeComponent();
        }
}
```

Листинг 8.7. Выделенный код (code-behind) для нового окна (для проекта VB)

```
Public Class NewOrder


End Class
```

Обратите внимание, что класс в этом коде называется `NewOrder`, потому что окно представляет собой просто еще один класс. Как вы знаете, вы можете создавать экземпляры классов и вызывать их методы, и именно этот подход вы и будете применять при использовании этого окна из обработчика событий `NewOrder_Click` в файле выделенного кода (code-behind) из окна `MainWindow`.

На практике вы должны будете заполнить окно `NewOrder` такими элементами управления, которые вам требуются для того, чтобы сформировать новый заказ. Это делается точно так же, как и в ранее приведенном в этой главе примере с кнопкой, а именно — перетаскиванием этих элементов управления в окно из окна **Toolbox**. Однако на данный момент мы этим заниматься пока не будем, потому что в данном случае наша основная задача — написать код для обработчика события `NewOrderButton_Click`, чтобы научиться добавлять код обработчика событий и открывать новые окна. Поэтому вернемся к обработчику события `NewOrderButton_Click` в окне `MainWindow.xaml.cs` и добавим к нему код, приведенный в листингах 8.8 (для C#) и 8.9 (для VB).

Листинг 8.8. Код обработчика события `NewOrderButton_Click` на C#

```
private void NewOrderButton_Click(object sender, RoutedEventArgs e)
{
        NewOrder newOrd = new NewOrder();
        newOrd.Show();
}
```

Листинг 8.9. Код обработчика события `NewOrderButton_Click` на VB

```
Private Sub NewOrderButton_Click(
        ByVal sender As System.Object,
        ByVal e As System.Windows.RoutedEventArgs)
Handles NewOrderButton.Click

        Dim newOrd As New NewOrder
        newOrd.Show()

End Sub
```

Так как окно `NewOrder` представляет собой класс, вы можете создавать его экземпляры, как было показано в только что приведенных листингах 8.8 и 8.9. Чтобы отобразить окно, необходимо вызвать метод `Show`.

Итак, вы получили программу WPF, которая обрабатывает события и открывает новые окна. Чтобы запустить эту программу, нажмите клавишу <F5>. Нажмите кнопку **New Order**, и вы увидите, как раскроется новое окно **New Order**. На данный момент это окно **New Order** нельзя назвать особенно полезным, потому что в нем нет ни элементов управления, ни инструментов для работы с данными. Поэтому в следующем разделе мы займемся тем, что заполним это окно элементами управления, предоставляющими средства для работы с данными.

Работа с данными в WPF

Материал, излагаемый в данном разделе, строится на основе кода, написанного в *главе 7*. В данном примере будет продемонстрировано, как связать данные с элементами управления WPF. Связыванием (Binding) называется процесс заполнения данными элементов управления и извлечения данных из этих элементов управления. В данном разделе будет продемонстрировано, как отображать данные с помощью созданного вами пользовательского интерфейса. Примеры, приведенные в последующих нескольких разделах, продемонстрируют выполнение операций создания (create), чтения (read), обновления (update) и удаления (delete), известных под собирательным названием (Create, Read, Update, Delete — CRUD) в WPF. Сначала мы рассмотрим, как осуществлять ввод данных, с использованием средств VS для конструирования экрана, предназначенного для ввода данных. Затем будут обсуждаться операции чтения, модификации и удаления данных с помощью элемента управ-

ления **DataGrid**. Начнем с простого связывания значений. Чтобы сделать примеры более наглядными, добавим в созданные в *главе 7* таблицы дополнительные поля. Основные сведения о том, как добавляются поля в базу данных и создаются модели сущностей LINQ to SQL, были приведены в *главе 7*, и я рекомендую вам освежить эту главу в памяти.

Настройка источника данных

Прежде чем вы сможете выполнить привязку данных к пользовательскому интерфейсу, вам необходимо создать источник данных. Для этой цели вполне подойдут таблицы, которые вы создавали при прочтении *главы 7*. Для начала обновите таблицу **Order** так, чтобы она содержала следующие поля:

- `OrderID`, тип данных `int`, первичный ключ (primary key), установлено свойство **auto-increment**
- `CustomerID`, тип данных `int`
- `OrderDate`, тип данных `datetime`
- `Location`, тип данных `varchar(50)`
- `Amount`, тип данных `money`

Затем модифицируйте таблицу **Customer**, в ней должны быть следующие поля:

- `CustomerID`, тип данных `int`, первичный ключ (primary key), установлено свойство **auto-increment**
- `Name`, тип данных `nvarchar(50)`
- `Age`, тип данных `int`
- `Birthday`, тип данных `datetime`
- `Income`, тип данных `money`

После того как база данных будет обновлена, вы сможете добавить к проекту новую модель сущностей LINQ to SQL, действуя точно так же, как было описано в *главе 7*.

Чтобы добавить источник данных для связывания, откройте в окне визуального конструктора окно `NewOrder` и выберите из меню команды **Data** | **Add New Data Source**. На экране появится окно **Choose a Data Source Type**, показанное на рис. 8.11.

Существуют различные пути соединения с источником данных, включая установление соединения непосредственно с базой данных, через Web-сервис, через объект или через SharePoint. В данной книге будет показано, как использовать LINQ to SQL. Для этого выберите в окне, показанном на рис. 8.11, опцию **Object** и нажмите кнопку **Next**, после чего появится окно **Select The Data Objects**, показанное на рис. 8.12.

В окне **Select The Data Objects** взведите флажок напротив каждого из объектов, который вы хотели бы связать с вашим приложением. В примере, который рассматривается в этой главе, используются объекты **Customer** и **Order**, поэтому флажки рядом с именами этих объектов взведены (рис. 8.12). Чтобы завершить конфигурирование источника данных для нашего приложения, нажмите кнопку **Finish**. Чтобы просмотреть источники данных, выберите из меню команды **Data** | **Show Data Sources**. На экране появится окно **Data Sources**, показанное на рис. 8.13.

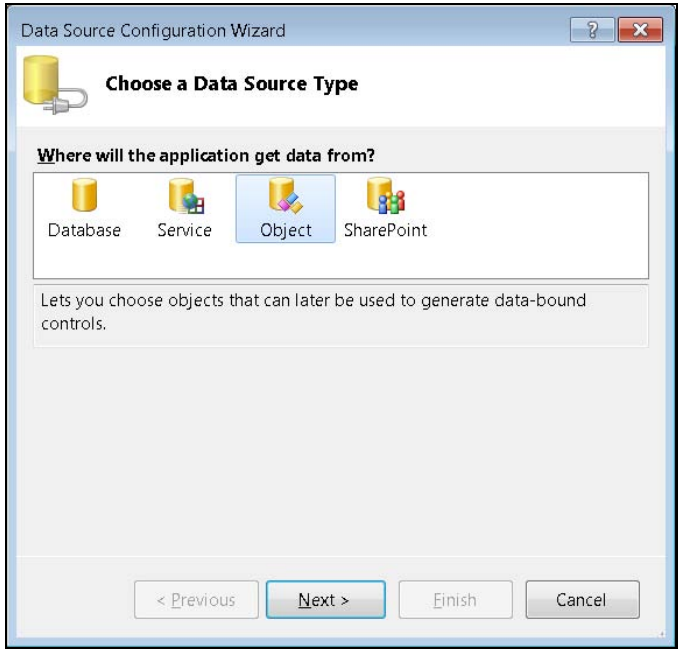


Рис. 8.11. Выбор нового источника данных

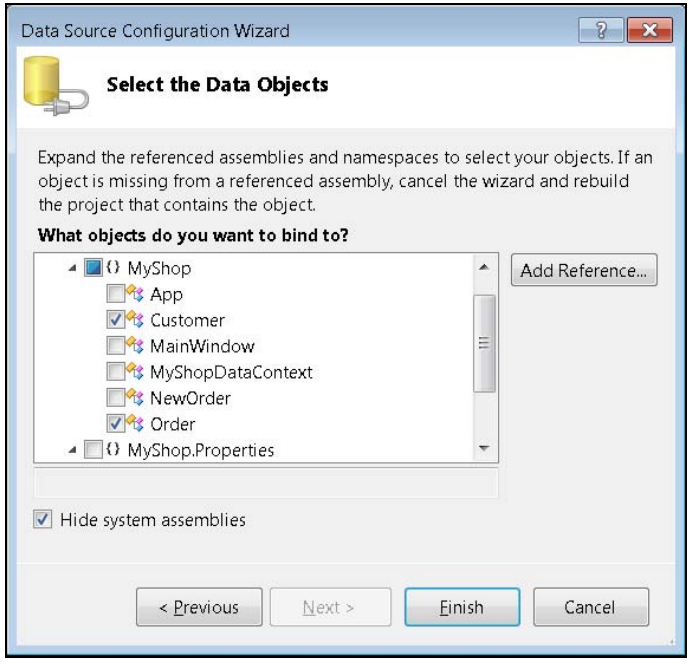


Рис. 8.12. Выбор объектов данных

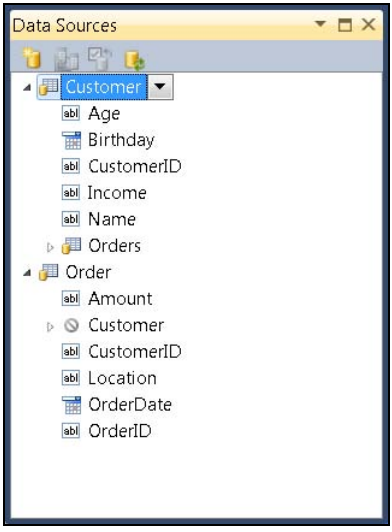


Рис. 8.13. Диалоговое окно **Data Sources**

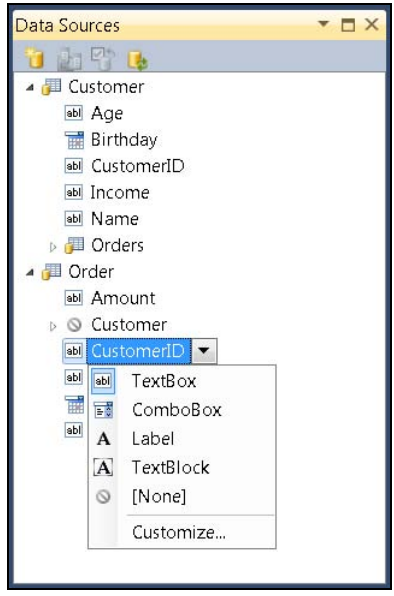


Рис. 8.14. Изменение типа элемента управления для поля базы данных

Окно **Data Sources** позволяет создавать в окне приложения элементы управления, связанные с каждым из полей источника данных. В окне **Data Sources**, показанном на рис. 8.13, видно, что таблицы **Customer** и **Object** перечислены со всеми своими полями. Далее, обратите внимание на то, что с каждым из полей ассоциирован значок. Значки описывают типы элементов управления, ассоциированных с каждым полем. Например, полю **Name** в таблице **Customer** соответствует элемент управления "текстовое поле" (**TextBox**), потому что данное поле, как вы помните, имеет тип данных `nvarchar(50)`, а вот полю **Birthday** соответствует значок с изображением календарика, потому что данное поле имеет тип данных `datetime`. Если вам не нравится тип элемента управления, предложенный для конкретного поля по умолчанию, вы можете его изменить, выделив это поле и выбрав из раскрывающегося списка другой подходящий тип элемента управления, как показано на рис. 8.14.

На рис. 8.14 показано, как изменить тип поля **CustomerID**. В нашем примере для этого поля лучше всего выбрать тип **ComboBox**, поскольку гораздо благоразумнее давать пользователю при оформлении нового заказа возможность выбирать идентификатор заказчика из списка, чем вводить этот идентификатор вручную. Кроме того, для объекта по умолчанию используется макет "сетка" (**Grid**), но в данном примере мы хотим применить новый порядок, а это значит, что тип элемента управления нужно изменить на **Details**. Чтобы создать новую форму заказа с помощью элементов управления, связанных с данными, выделите объект **Order** в окне **Data Sources** и перетащите мышью в окно **NewOrder** таблицу **Order**. Примерный вид окна после этой операции показан на рис. 8.15.

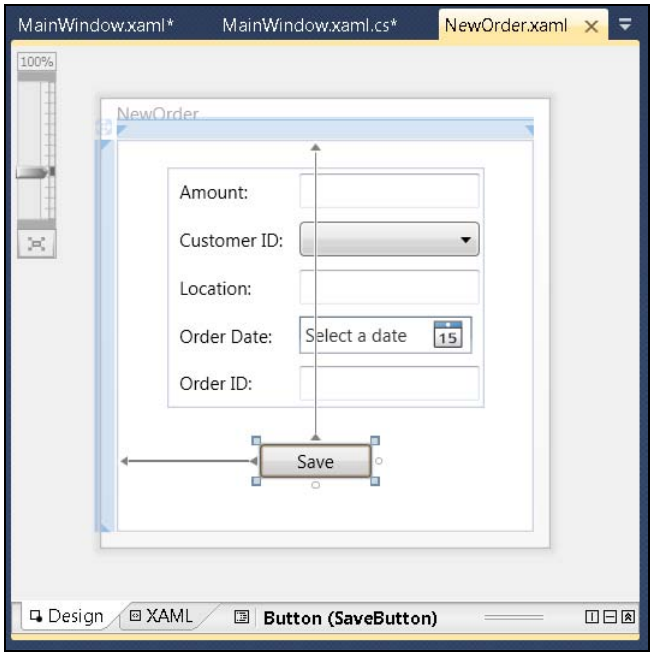


Рис. 8.15. Привязка элементов управления к источникам данных

На рис. 8.15 показано, как VS добавляет элемент, управляющий макетом **Grid** с двумя столбцами и одной строкой для каждого поля таблицы **Order**. Как уже объяснялось ранее, поле **CustomerID** представляет собой раскрывающийся список (элемент управления **ComboBox**), а поле **OrderDate** представляет собой "календарик", позволяющий выбрать конкретную дату. При этом поведение VS было достаточно интеллектуальным — например, в метках между словами были вставлены пробелы. Впрочем, в окно не было добавлено кнопки **Save**, которая нужна для сохранения данных, и которую вам придется добавить самостоятельно с помощью визуального конструктора, Зато VS автоматически добавляет элемент управления `CollectionViewSource` на панель XAML окна `NewOrder`, код которого приведен в листинге 8.10.

Листинг 8.10. Код элемента управления `CollectionViewSource`, автоматически добавленный VS на панель XAML окна `NewOrder`

```
<Window.Resources>
        <CollectionViewSource x:Key="orderViewSource"
        d:DesignSource="{d:DesignInstance my:Order, CreateList=True}" />
</Window.Resources>
```

Это — еще одна причина, по которой важно научиться понимать код XAML, генерируемый для окна, чтобы иметь возможность разобраться с тем, как в это окно добавляются объекты наподобие только что рассмотренного, и, при необходимости,

конфигурировать их программно. В рассматриваемом примере нам необходимо узнать имя элемента управления `CollectionViewSource`, которое в данном случае выглядит так: `orderViewSource`. Нам необходимо добавить объект `Order` в состав `CollectionViewSource`, чтобы связанные с ним элементы управления сохраняли данные, вводимые пользователем. Нажмите клавишу <F7>, чтобы увидеть код, который VS добавляет к обработчику события `Window Loaded`. Этот код представлен в листингах 8.11 и 8.12.

Листинг 8.11. Код на C#, добавляемый VS к обработчику события Window_Loaded

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
        System.Windows.Data.CollectionViewSource
        orderViewSource =
        ((System.Windows.Data.CollectionViewSource)
        (this.FindResource("orderViewSource")));
        // Load data by setting the
//CollectionViewSource.Source property:
        // orderViewSource.Source = [generic data source]
}
```

Листинг 8.12. Код на VB, добавляемый VS к обработчику события Window_Loaded

```
Private Sub Window_Loaded(
    ByVal sender As System.Object,
    ByVal e As System.Windows.RoutedEventArgs) Handles MyBase.Loaded
    Dim OrderViewSource As System.Windows.Data.CollectionViewSource
        = CType(Me.FindResource("OrderViewSource"),
        System.Windows.Data.CollectionViewSource)
    'Load data by setting the CollectionViewSource.Source property:
    'OrderViewSource.Source = [generic data source]
End Sub
```

Приведенный в листингах 8.11 и 8.12 код "скелета" обработчика получает ссылку на `OrderViewSource`, но это и все. Закомментированные строки кода представляют собой предположение того, каким образом следует заполнять элемент управления. Однако мы не заинтересованы в заполнении `OrderViewSource` данными из базы данных, потому что целью данного экрана является создание новой записи. Поэтому в данном случае правильным подходом будет не тот, что предложен комментариями, а наоборот, привязка пустого объекта. Впоследствии будет показано, как извлечь данные из этого объекта после того, как пользователь заполнит форму и щелкнет мышью по кнопке **Save**. В дополнение к присваиванию `OrderViewSource` пустого объекта `Order`, нам необходимо заполнить элемент управления `ComboBox`, который содержит список клиентов и их идентификаторов. Код, приведенный в листингах 8.13 и 8.14, представляет собой откорректированную версию обра-

ботчика события `Window_Loaded`, который присваивает `OrderViewSource` пустой объект `Order` и связывает список заказчиков с элементом управления `ComboBox`.

Листинг 8.13. Откорректированная версия обработчика события Window_Loaded на C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
        var orderViewSource =
            FindResource("orderViewSource") as CollectionViewSource;
        orderViewSource.Source =
        new List<Order>
        {
                new Order
                {
                      OrderDate = DateTime.Now
                }
        };
        customerIDComboBox.ItemsSource =
            from cust in new MyShopDataContext().Customers
            select cust;
}
```

Листинг 8.14. Откорректированная версия обработчика события Window_Loaded на VB

```
Private Sub Window_Loaded(
    ByVal sender As System.Object,
    ByVal e As System.Windows.RoutedEventArgs) Handles MyBase.Loaded

    Dim OrderViewSource As CollectionViewSource =
        CType(FindResource("OrderViewSource"), CollectionViewSource)
    OrderViewSource.Source =
        New List(Of Order) From
        {
              New Order With
              {
                       .OrderDate = DateTime.Now
              }
        }

    CustomerIDComboBox.ItemsSource =
        From cust In New MyShopDataContext().Customers
End Sub
```

Откорректированный вариант обработчика события `Window_Loaded` осуществляет две задачи: присваивание `orderViewSource` пустого заказа и заполнение поля раскрывающегося списка `customerIDComboBox` списком заказчиков. Объект `Order`,

присвоенный свойству `Source` элемента управления `orderViewSource`, пуст, за исключением того, что для поля `OrderDate` установлена текущая дата, что демонстрирует присвоение значений по умолчанию. Когда пользователь заполняет этот бланк заказа, WPF заполнит этот объект `Order` данными, потому что существует привязка по данным через `orderViewSource` к элементам управления, отображаемым в окне. В данном разделе было показано, каким образом данные назначаются элементам управления. Однако некоторые элементы управления требуют еще больше внимания, поскольку необходим контроль над корректным отображением данных. В следующем разделе будет показано, что необходимо сделать, чтобы обеспечить корректную работу поля раскрывающегося списка.

Конфигурирование поля раскрывающегося списка

Особенно сложными для конфигурирования являются такие элементы управления, как `ComboBox` и `ListBox`. Причина этого заключается в том, что они имеют несколько различных свойств, которые должны быть установлены для того, чтобы гарантировать, что выбранный элемент списка, каким бы он ни был, может быть прочитан и на него может быть дана корректная ссылка вплоть до оригинального источника данных. В данном разделе не делается попытки обучить вас связыванию WPF, потому что на эту тему написаны целые книги, в которых данному вопросу уделена не одна глава. Вместо этого я попробую помочь вам приобрести базовый навык, благодаря которому можно разобраться, как установить правильные свойства для элемента управления `ComboBox`. Выбрав такой подход, вы сможете лучше прочувствовать те особенности VS, которые помогают выполнять задачу настройки элементов управления.

В только что рассмотренном примере результаты запроса LINQ к объектам `Customer` присваиваются элементу управления `customerIDComboBox`, но это — всего лишь первый шаг, который необходимо выполнить, чтобы раскрывающийся список начал работать корректно. Вам необходимо указать, какое свойство объекта `Customer` должно отображаться, какое свойство `Customer` соответствует `Order` и какое свойство `Order` должно быть связано с этим элементом. Чтобы осуществить это связывание, откройте файл NewOrder.xaml в окне визуального редактора (Designer), выделите комбинированный список и задайте для него свойства так, как указано в табл. 8.2.

Код XAML, представленный в листинге 8.15, показывает результаты настройки, выполненной в окне **Properties** на основании информации, приведенной в табл. 8.2.

Листинг 8.15. Код XAML, показывающий результаты настройки поля раскрывающегося списка, выполненной в окне Properties на основании информации, приведенной в табл. 8.2

```
<ComboBox DisplayMemberPath="Name"
          SelectedValue="{Binding Path=CustomerID}"
          SelectedValuePath="CustomerID"
          Grid.Column="1" Grid.Row="1"
```

```
        Height="23" HorizontalAlignment="Left"
        Margin="3" Name="customerIDComboBox"
        VerticalAlignment="Center" Width="120">
</ComboBox>
```

Таблица 8.2. *Свойства элемента управления "комбинированный список" (ComboBox), настроенные для правильной привязки данных*

Свойство	Пояснение
`ItemsSource`	Это свойство устанавливается через код обработчика события `Window_Loaded`. Оно содержит коллекцию объектов, которые будут появляться в поле раскрывающегося списка. Вам потребуются два свойства, одно — для отображения и одно — для ключа отображаемого объекта. Ключ будет использоваться для установления соответствия объекта записи в базе данных или для того, чтобы ассоциировать объект с отношением к другому объекту. В случае со списком `Customer`, нас интересуют такие свойства, как `Name` — для отображения имени клиента и `CustomerID` — для ключа. Так как мы создаем новый заказ (объект `Order`), в раскрывающемся списке выбирается значение `CustomerID` для выбранного значения `Name`, и это значение будет присвоено полю `CustomerID` для объекта `Order`. Таким образом, при сохранении нового заказа в базе данных, его запись в базе данных получит значение `CustomerID` для клиента, на имя которого формируется заказ
`DisplayMemberPath`	Это — свойство `Name` для каждого из объектов `Customer`, связанных с полем раскрывающегося списка
`SelectedValuePath`	Как уже говорилось ранее, когда речь шла о свойстве `ItemsSource`, вам необходимо ассоциировать выбранный объект `Customer` с создаваемым объектом `Order`. `SelectedValuePath` — это имя ключа объекта `Customer`, в нашем примере это будет поле `CustomerID`
`SelectedValue`	При сохранении объекта `Order` вам необходимо ассоциировать ключ с выбранным клиентом. Свойство `SelectedValue` создает привязку к свойству объекта `Order`, которое получит значение ключа выбранного объекта `Customer`

`DisplayMemberPath` и `SelectedValuePath` представляют собой свойства объектов `Customer`, привязанных к элементу управления "комбинированный список" (`ComboBox`). Однако синтаксис `SelectedValue` использует связывающее выражение, в котором *`Path`* идентифицирует свойство объекта `Order`, которое получит значение `SelectedValuePath`. Связывание для `SelectedValue` основывается на объекте `Order`, который был присвоен свойству `Source` объекта `orderViewSource` в обработчике события `Window_Loaded`. Проходя этот полный цикл, объект `orderViewSource` становится предлагаемой по умолчанию привязкой к макету **Grid**; эта привязка была создана при перетаскивании источника данных **Order** в окно визуального конструирования (Design).

Теперь вы имеете бланк-форму для ввода, которая отображает данные и позволяет пользователю вводить информацию о новом заказе. Когда пользователь введет необходимые данные в поля формы, вам потребуется сохранить данные. Эта операция будет обсуждаться в следующем разделе.

Чтение и сохранение данных

После того как необходимая информация будет введена в поля бланка-формы заказа, пользователь должен сохранить данные, нажав кнопку **Save**. Чтобы сделать это, необходимо добавить к форме командную кнопку (элемент управления **Button**), установить для ее свойства **Content** значение **Save** и установить для свойства **Name** имя **SaveButton**. О том, как это делается, было рассказано ранее в этой главе. Затем выполните двойной щелчок мышью на кнопке **Save**, чтобы создать обработчик события **Click**. Код этого обработчика представлен в листингах 8.16 и 8.17.

Листинг 8.16. Код обработчика события Click для кнопки Save (для проекта C#)

```
private void SaveButton_Click(object sender, RoutedEventArgs e)
{
        CollectionViewSource orderViewSource =
            FindResource("orderViewSource") as CollectionViewSource;

        List<Order> ordList =
            orderViewSource.Source as List<Order>;
        Order ord = ordList.FirstOrDefault();

        var ctx = new MyShopDataContext();

        ctx.Orders.InsertOnSubmit(ord);

        ctx.SubmitChanges();

        MessageBox.Show("Order Saved!");
}
```

Листинг 8.17. Код обработчика события Click для кнопки Save (для проекта VB)

```
Private Sub SaveButton_Click(
    ByVal sender As System.Object,
    ByVal e As System.Windows.RoutedEventArgs)
    Handles SaveButton.Click

    Dim OrderViewSource As CollectionViewSource =
        CType(FindResource("OrderViewSource"), CollectionViewSource)

    Dim ordList As List(Of Order)
```

```
    ordList = CType(OrderViewSource.Source, List(Of Order))
    Dim ord As Order
    ord = ordList.FirstOrDefault()

    Dim ctx As New MyShopDataContext

    ctx.Orders.InsertOnSubmit(ord)
    ctx.SubmitChanges()

    MessageBox.Show("Order Saved!")
End Sub
```

Перед завершением работы обработчика событий `SaveButton_Click`, он отображает окно с сообщением пользователю о статусе заказа: **Order Saved**. Класс `MessageBox` имеет несколько перегрузок метода `Show`, который позволяет указать кнопки, значки и т. п.

Итак, теперь вы умеете создавать формы для добавления новых записей в базу данных. В следующем разделе мы, основываясь на этой информации, рассмотрим процедуры просмотра, модификации и удаление записей с помощью макета **DataGrid.**

Использование макета *DataGrid*

Макет **DataGrid** предоставляет наиболее удобные возможности для работы с данными, когда необходимо отобразить их в виде таблицы, содержащей множество строк и столбцов. В данном разделе будет показано, как отображать, модифицировать и удалять элементы данных с помощью макета сетки. Для начала нам необходимо просмотреть список заказов.

Мы воспользуемся источником данных, созданным в предыдущем примере, чтобы проиллюстрировать отображение данных в сетке таблицы (**Grid**). Сначала откройте окно **Data Source**, выбрав из меню команды **Data** | **Open Data Sources**. В предшествующем разделе мы указывали для поля `CustomerID` тип **ComboBox.** Если вы последовательно выполняете все шаги, измените тип поля `CustomerID` на **TextBox**, щелкнув мышью по полю **CustomerID** объекта **Order** в окне **Data Sources** и выбрав из раскрывшегося списка опцию **TextBox**. Измените тип элемента управления для объекта **Order** на сетку, щелкнув по объекту **Order** в окне **Data Sources** и выбрав из раскрывающегося списка опцию **DataGrid**. Затем раскройте файл MainWindow.xaml в визуальном редакторе и мышью перетащите объект **Order** из окна **Data Sources** в область визуального конструирования (**Designer**). Не забудьте перетащить уже имеющуюся в этом окне кнопку **New Order** в нижнюю часть формы. Кроме того, добавьте еще одну кнопку, установите для нее свойство **Name** на **UpdateButton** и установите для нее свойство **Content** на **Update**. Расположите кнопки **New Order** и **Update** в нижней части формы. Полученный результат должен выглядеть примерно так, как показано на рис. 8.16.

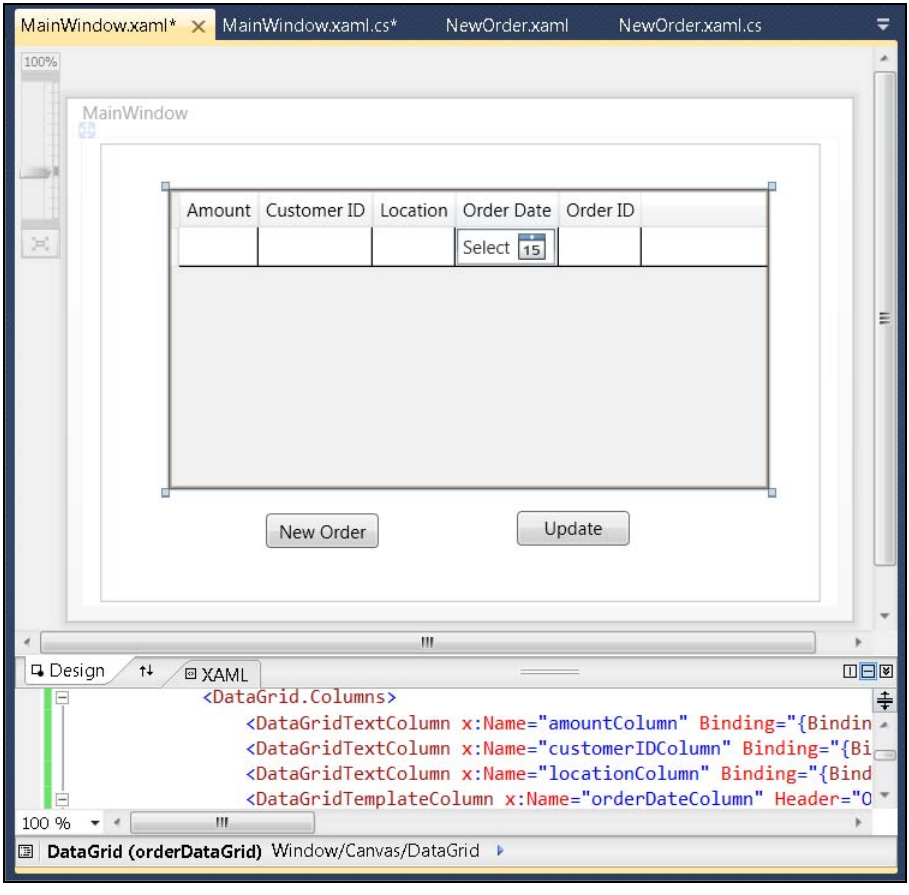


Рис. 8.16. Отображение информации в сетке таблицы

Как и в случае с примером из предыдущего раздела, VS добавит объект `CollectionViewSource`, точно так же, как это было в примерах, рассматривавшихся ранее в этой главе. В листингах 8.17 и 8.18 приведен код обработчика `Window_Loaded`, который позволяет отобразить данные о заказах в сетке таблицы.

Листинг 8.17. Код обработчика Window_Loaded, который позволяет отобразить данные о заказах в сетке таблицы (для проекта на C#)

```
private MyShopDataContext m_ctx = new MyShopDataContext();

private void Window_Loaded(object sender, RoutedEventArgs e)
{
        CollectionViewSource orderViewSource =
                FindResource("orderViewSource") as CollectionViewSource;

        orderViewSource.Source =
```

```
            from ord in m_ctx.Orders
            select ord;
}
```

Листинг 8.18. Код обработчика Window_Loaded, который позволяет отобразить данные о заказах в сетке таблицы (для проекта на VB)

```
Dim m_ctx As New MyShopDataContext

Private Sub Window_Loaded(
    ByVal sender As System.Object,
    ByVal e As System.Windows.RoutedEventArgs)
    Handles MyBase.Loaded

    Dim OrderViewSource As CollectionViewSource =
        CType(FindResource("OrderViewSource"), CollectionViewSource)

    OrderViewSource.Source =
        From ord In m_ctx.Orders
        Select ord
End Sub
```

Этот код загружает данные о заказах в сетку таблицы. Обратите внимание, что объект `MyShopDataContext`, `m_ctx` представляет собой поле, находящееся за пределами обработчика события `Window_Loaded`. Этот объект поднят до уровня поля с тем, чтобы один и тот же экземпляр мог использоваться множеством методов. Как видите (см. рис. 8.16), в нашей новой форме есть кнопка **Update**. Выполните двойной щелчок мышью по кнопке **Update**, чтобы запустить обработчик события `UpdateButton_Click`, который сохраняет изменения, внесенные в сетку данных. Код этого обработчика приведен в листингах 8.19 и 8.20.

Листинг 8.19. Код обработчика события UpdateButton_Click на C#

```
private void UpdateButton_Click(object sender, RoutedEventArgs e)
{
        m_ctx.SubmitChanges();

        MessageBox.Show("Updates and Deletes Saved!");
}
```

Листинг 8.20. Код обработчика события UpdateButton_Click на VB

```
Private Sub UpdateButton_Click(
    ByVal sender As System.Object,
    ByVal e As System.Windows.RoutedEventArgs)
```

```
    Handles UpdateButton.Click

    m_ctx.SubmitChanges()

    MessageBox.Show("Updates and Deletes Saved!")
End Sub
```

Когда вы запустите эту программу, вы сможете добавлять в таблицу новые строки, модифицировать поля существующих строк, а также удалять строки. Чтобы удалить строку, выделите ее и нажмите на клавиатуре клавишу <DELETE>. Закончив внесение изменений в сетку таблицы, щелкните мышью по кнопке **Update**, чтобы вызвать обработчик события `UpdateButton_Click`.

Чтобы понять, как все это работает, вспомните, что обработчик события `Window_Loaded` назначает коллекцию объектов `Order` объекту типа `CollectionViewSource`, `orderViewSource`, который осуществляет привязку данных к сетке таблицы. Каждая строка сетки связана с экземпляром объекта `Order`. Каждый объект `Order` является частью контекста данных LINQ to SQL `MyShopDataContext`. Так как мы используем поле `m_ctx`, то как метод `Window_Loaded`, так и метод `UpdateButton_Click` используют один и тот же экземпляр объекта. При внесении изменений в сетку таблицы, эти изменения сохраняются в объектах `Order`, поля которых были модифицированы. Объекты `Order` уведомляют `MyShopDataContext` о внесенных изменениях — этот сервис предоставляется LINQ to SQL. Метод `UpdateButton_Click` использует `m_ctx`, который представляет собой контекст данных `MyShopDataContext`, и знает об изменениях, внесенных в объекты `Order`. Вызов метода `SubmitChanges` в составе `m_ctx` сохраняет все внесенные изменения в базе данных.

Возможно, что для того, чтобы добиться полного понимания, вам придется несколько раз перечитать предыдущий раздел. Если вы все же испытываете затруднения с пониманием, будет полезно пересмотреть главы этой книги, посвященные особенностям языков программирования, а также *главу 7* — тогда вы получите полную картину того, как производятся манипуляции над данными.

Заключение

В этой главе была сделана попытка дать популярное введение в технологию WPF, проиллюстрировав излагаемый материал простейшими примерами. Тем не менее, хотя это всего лишь краткое введение, и вам предстоит узнать еще очень большое количество материалов о WPF, это краткое "руководство для начинающих" должно помочь вам выбрать правильное направление для самостоятельной работы. После прочтения этой главы вы должны уметь самостоятельно создавать новые проекты WPF. Далее в данной главе рассказывалось о макетах, и как они помогают организовать расположение элементов управления в ваших формах. Следующий раздел был посвящен работе с элементами управления. Хотя сущест-

вует множество элементов управления, поставляющихся в комплекте с WPF, равно как и множество продуктов от сторонних поставщиков, не следует забывать и о том, что практически работа с элементами управления сводится к их перетаскиванию мышью в область визуального конструирования и последующему редактированию их свойств. Большинство приложений так или иначе работают с данными, поэтому в этой главе было уделено достаточно внимания LINQ to SQL и показано, как создаются пользовательские интерфейсы для работы с данными.

Данная глава представляет собой краткое введение в разработку настольных приложений, которые по-прежнему крайне необходимы и пользуются популярностью. Однако многие из современных приложений предназначаются для работы в Интернете. В следующей главе будет показано, как создавать Web-приложения с помощью ASP.NET.

Глава 9

Разработка Web-приложений с помощью ASP.NET MVC

В этой главе будут описаны следующие ключевые концепции:

- Что означает MVC;
- Создание моделей;
- Создание контроллеров;
- Создание представлений;
- Работа с данными в ASP.NET MVC.

ASP.NET — это технология .NET, предназначенная для построения Web-приложений. VS обеспечивает поддержку разработки Web-приложений, предоставляя для этой цели такие окна, как **Toolbox**, визуальный конструктор (**Designer**), **Properties** и **Solution Explorer**. В данной главе рассказывается об использовании ASP.NET MVC. MVC — это сокращение от Model View Controller ("модель — представление — контроллер"). MVC представляет собой хорошо известную архитектуру программного обеспечения. В этой главе будет рассказано о том, как работает MVC, как эта архитектура реализована в ASP.NET MVC. Начнем обсуждение с того, что же представляет собой архитектура MVC.

Разбираемся с ASP.NET MVC

Основная концепция, которую необходимо усвоить, чтобы успешно работать с ASP.NET MVC — это архитектура Model View Controller. В MVC модель (Model), представление (View) и контроллер (Controller) являются тремя отдельными объектами. В табл. 9.1 приведены описания каждого из объектов MVC.

Таблица 9.1. *Предназначение объектов MVC*

Объект MVC	Предназначение
`Model`	Объект `Model` состоит из бизнес-объектов и данных
`View`	Каждое приложение MVC обычно имеет пользовательский интерфейс, который отображает для пользователя информацию и позволяет вводить данные. Данные, отображаемые объектом `View`, считываются из объекта Model, а данные, вводимые пользователем, добавляются в объект `View`, который присваивает их объекту `Model`

Таблица 9.1 *(окончание)*

Объект MVC	Предназначение
`Controller`	Объект `Controller` руководит деятельностью приложения. Когда пользователь делает запрос к вашему приложению, ASP.NET MVC инициирует объект `Controller`. `Controller` будет поддерживать коммуникации с `Model` и `View`, чтобы гарантировать корректную работу программы

С помощью MVC можно четко разделить области ответственности между объектами `Model`, `View` и `Controller`. Это сильно упрощает написание программ, к которым вы впоследствии легко можете вернуться для отладки, исправления ошибок и добавления новых функциональных возможностей. Помимо предназначения каждого из этих трех объектов, необходимо понимать и связывающие их отношения. Схема, представленная на рис. 9.1, иллюстрирует объекты `Model`, `View` и `Controller`, включая связи между ними. Между объектами `Model`, `View` и `Controller` могут существовать разнообразные взаимосвязи, поэтому схема, показанная на рис. 9.1, не дает полного теоретического представления обо всех возможных сценариях, а, скорее, представляет упрощенное обобщение, которое, тем не менее, позволяет быстро войти в курс дела.

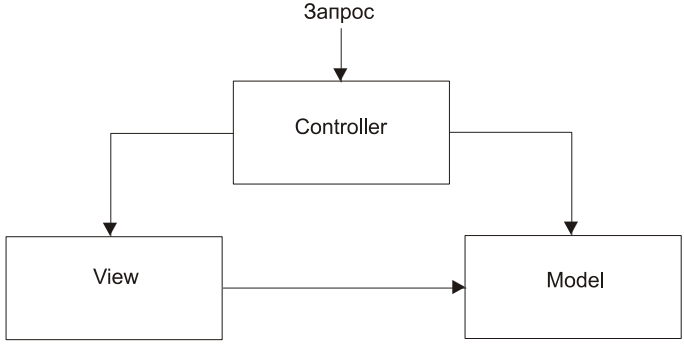


Рис. 9.1. Архитектура "модель — представление — контроллер" (MVC)

На рис. 9.1. показано, что объект `Controller` ссылается как на объект `View`, так и на объект `Model`. Это разумно в тех случаях, когда объект `Controller` управляет работой приложения. `Controller` исполняется в ответ на пользовательский запрос. Поскольку `Controller` также отвечает за координацию действий между объектами `Model` и `View`, на схеме показано, что `Controller` ссылается на `Model` и `View`. Объект `View` ссылается на `Model`, потому что `View` требуется осуществить привязку данных к пользовательскому интерфейсу и, следовательно, необходимо иметь информацию о том, какие данные доступны. Объект `Model` не ссылается ни на `View`, ни на `Controller`. `Model` представляет собой объект, содержащий данные и все остальные члены класса, которые помогают управлять этими данными — например, методы, необходимые для валидизации данных (validation).

Типичная последовательность операций, осуществляемых в ASP.NET MVC, начинается с запроса к объекту `Controller`. Объект `Controller` выполнит запрошенные действия, работая с объектом `Model`. Затем `Controller` передаст объект `Model` объекту `View` и запустит `View`. Объект `View` отобразит данные `Model` и будет интерактивно взаимодействовать с пользователем, осуществляющим любые операции через пользовательский интерфейс. На основании взаимодействия с пользователем с помощью `View`, могут быть сформированы новые запросы к объекту `Controller`, и вся процедура повторится. Остальные материалы данной главы демонстрируют написание кода, необходимого для работы этого процесса. Как всегда, начнем с создания нового проекта ASP.NET MVC.

Создание нового проекта ASP.NET MVC

Как и в случае с любым другим проектом VS, для создания нового проекта ASP.NET MVC вам необходимо открыть окно **New Project**, выбрав из меню команды **File** | **New** | **Project**. Затем создайте новые проекты типа **ASP.NET MVC 2 Web Application** и назовите его **MyShopCS** (**MyShopVB** — для проекта на VB). VS запросит, хотите ли вы создать проект модульных тестов (unit test project), и `ModelViewControllerRequest` даст вам возможность ответить утвердительно или отрицательно, нажав кнопки **Yes** или **No**, соответственно. Нажмите кнопку **Yes**, и тогда в состав решения будет включен проект модульных тестов. В принципе, вы можете выбрать любую опцию, поскольку на начальном этапе модульное тестирование практического значения не имеет, и данная тема в книге не рассматривается. Тем не менее, я вам настоятельно рекомендую поэкспериментировать и самостоятельно поисследовать все возможные опции. На рис. 9.2 показан новый проект в окне **Solution Explorer**.

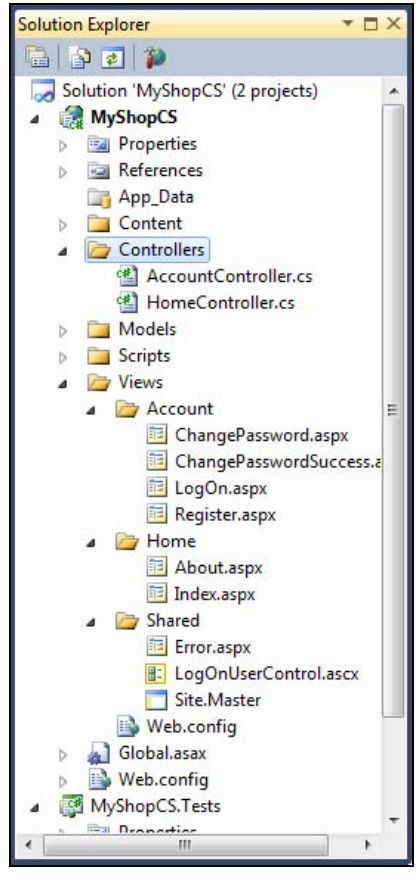


Рис. 9.2. Новый проект ASP.NET MVC в окне **Solution Explorer**

При создании нового проекта VS создаст следующие папки с работоспособным кодом:

- Папки Model, View и Controller содержат код для таких объектов MVC, как модели, представления и контроллеры, соответственно.
- Предназначение таких папок, как Properties и References, уже объяснялось в предшествующих главах.
- Папка App_Data создана с тем, чтобы дать вам возможность поставлять с вашим приложением локальную базу данных. Этот вариант идеален для небольших программ, при напи-

сании которых можно пользоваться бесплатным вариантом SQL Server — SQL Express. Далее по ходу изложения материала будет показано, каким образом можно добавить базу данных в папку App_Data.

- Папка Content предназначена для добавления любых ваших файлов, содержащих каскадные таблицы стилей (Cascading Style Sheets, CSS). CSS — это стандартизованный язык, предназначенный для определения компоновки и общего вида вашего Web-сайта.
- Папка Scripts содержит файлы, написанные на JavaScript, включая клиентские библиотеки jQuery и ASP.NET AJAX. JavaScript позволяет придать объектам `View` дополнительные интерактивные возможности и сделать работу с пользовательским интерфейсом более эффективной.
- Файл Global.asax содержит код, который запускается в различные периоды в течение жизненного цикла приложения. Этот файл мы исследуем далее в этой главе, при обсуждении маршрутизации (routing).
- Файл Web.config содержит конфигурационную информацию, например, такую, как строки, описывающие соединения с базой данных, а также другую информацию, которую вы не хотите жестко "зашивать" в приложение.

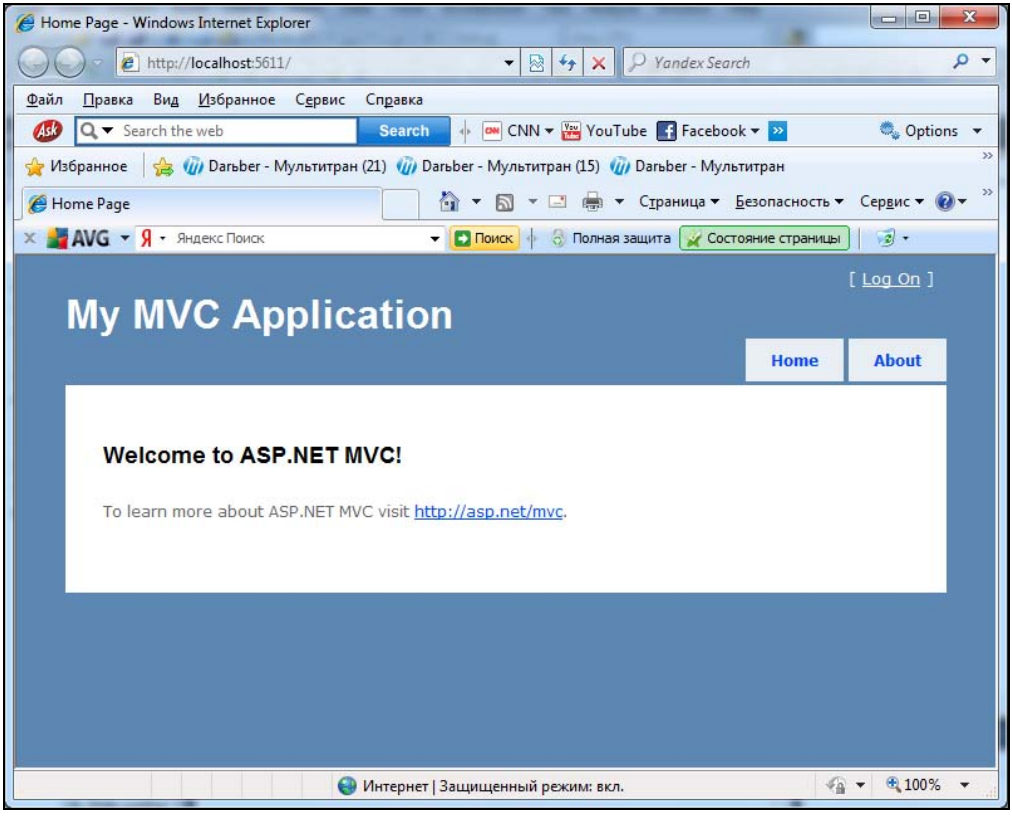


Рис. 9.3. Запуск стандартного кода, автоматически созданного для первого проекта ASP.NET MVC

Примечание

Если вы планируете поставлять вместе с вашим приложением локальную базу данных, наведите курсор мыши на папку App_Data в составе вашего проекта, щелкните правой кнопкой мыши и выберите из контекстного меню команды **Add | New Item**. В левой панели раскрывшегося окна выберите опцию **Data**, а в центральной панели — опцию **SQL Server Database**. После этого в состав папки App_Data будет добавлен пустой файл базы данных *.mdf. Работать с этой базой данных вы можете через окно **Server Explorer**, используя приемы, описанные в *главе 7*. Для начала вам потребуется добавить в состав пустой базы данных таблицы и другие объекты. Не забывайте о том, что на том сервере, куда вы будете развертывать свое приложение, должен быть установлен продукт SQL Server Express, в противном случае ваши операции по работе с базой данных работать не будут.

Код, сгенерированный мастером New Project Wizard, будет вполне работоспособен. Нажмите клавишу <F5>, чтобы запустить его, и через некоторое время вы увидите экран, выглядящий примерно так, как показано на рис. 9.3. Нажмите кнопку **OK**, когда вы увидите экран, где вам будет предложено запустить программу в отладочном режиме. После этого файл web.config будет модифицирован таким образом, чтобы допускать отладку — это именно то, что вам и требуется при разработке приложений.

Код "скелета" приложения, сгенерированный VS, уже предоставляет вам несколько работоспособных образцов, на основе которых вы можете выполнить сборку приложения и двигаться дальше. Однако VS не создает никаких моделей. Модели вам необходимо создавать самостоятельно, и эта тема обсуждается в следующем разделе.

Создание моделей

Как уже говорилось ранее, объект `Model` представляет данные для вашего приложения. Пример, приведенный в этом разделе для создания объекта `Model`, использует LINQ to SQL. Чтобы создать объект `Model`, щелкните правой кнопкой мыши по папке Models, выберите из контекстного меню команды **Add | New Item**, а затем выберите опцию **LINQ to SQL**. В результате этого будет создан пустой файл *.dbml, в который вам следует добавить сущности `Customer` и `Order`, используя те же самые приемы, которые описывались в *главе 7*.

В более сложных сценариях вам потребуется добавлять дополнительные объекты, реализующие бизнес-логику, или другие данные, не ассоциированные с LINQ to SQL. В данной книге мы ограничимся максимально простыми задачами, исключительно базового уровня, чтобы не отвлекаться на детали, а продемонстрировать, как следует пользоваться VS. Объекты `Model` можно помещать в папку Models или в отдельную библиотеку классов. В данной главе мы будем использовать папку Models.

Построение контроллеров

Запросы поступают непосредственно к объекту `Controller`, который уже обсуждался ранее в этой главе. Как было показано на рис. 9.2, в составе проекта MVC имеется папка Controllers. Как правило, классы объектов `Controller` располагаются

в папке Controllers. На рис. 9.2 в составе папки Controllers показаны два файла — AccountController.cs и HomeController. Содержимое файлов HomeControllerCS и HomeControllerVB показано в листингах 9.1 и 9.2, соответственно.

Листинг 9.1. Содержимое класса HomeController (код на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MyShopCS.Controllers
{
        [HandleError]
        public class HomeController : Controller
        {
                public ActionResult Index()
                {
                        ViewData["Message"] = "Welcome to ASP.NET MVC!";

                        return View();
                }

                public ActionResult About()
                {
                        return View();
                }
        }
}
```

Листинг 9.2. Содержимое класса HomeController (код на VB)

```
<HandleError()> _
Public Class HomeController
    Inherits System.Web.Mvc.Controller

    Function Index() As ActionResult
        ViewData("Message") = "Welcome to ASP.NET MVC!"

        Return View()
    End Function

    Function About() As ActionResult
        Return View()
    End Function

End Class
```

Код, приведенный в листингах 9.1 и 9.2, показывает, насколько тесно архитектура ASP.NET MVC связана с соглашениями. Обратите внимание, что класс называется `HomeController`. Строка `Controller` добавляется к имени класса в соответствии с соглашением об именовании, по которому ASP.NET MVC идентифицирует классы, являющиеся контроллерами. Кроме того, методы в составе класса идентифицируются как действия (Actions). Использование папки Controllers для хранения объектов `Controller`, дополнение имени класса строкой `Controller`, а также идентификация всех методов в составе класса как действий (actions) — все это соглашения, которые вы обязаны соблюдать. Адрес, который появляется в строке браузера при запуске приложения, демонстрирует, как эти соглашения поддерживают маршрутизацию для поиска нужного контроллера и инициации нужного вам действия (в приведенном примере это будет действие `About`). Запустите приложение и щелкните мышью по вкладке **About** (см. рис. 9.3). Вы увидите примерно следующий URL:

```
http://localhost:5611/Home/About
```

Подстрока `http://localhost:5611` в составе универсального определителя ресурса (URL) представляет собой адрес Web-сервера, встроенного в состав VS и позволяющего запускать Web-приложения без необходимости установки такого Web-сервера, как, например, Internet Information Server (IIS). Число (в нашем примере — 5611) представляет собой случайно выбранный номер порта, сгенерированный Web-сервером, поэтому, скорее всего, у вашего URL номер порта будет другим.

> ***Совет***
>
> При желании вы можете изменить номер порта для встроенного Web-сервера VS. Для этого откройте страницу свойств проекта, перейдя в окно **Solution Explorer** и щелкнув правой кнопкой мыши по имени проекта. Когда появится контекстное меню, выберите команду **Properties**, затем перейдите на вкладку **Web** в левой панели раскрывшегося окна. Затем перейдите в правую панель окна. Там, в группе опций **Servers**, вы сможете назначить Web-серверу нужный вам номер порта и выполнить другие настройки, касающиеся работы Web-сервера.

Для ASP.NET MVC важной частью URL является строка `/Home/About`. `Home` — это имя контроллера, и ASP.NET MVC дополняет его подстрокой `Controller`, чтобы указать, что искать следует класс `HomeController` (см. листинги 9.1 и 9.2), который физически расположен в папке Controllers. Вот почему очень важно создавать файлы в специально отведенных для этого папках файловой системы. `About` — это действие, которое соответствует методу `About` (см. листинги 9.1 и 9.2). По аналогии с методом `About`, метод `Index` будет инициирован через следующий URL:

```
http://localhost: 5611/Home/Index
```

Далее в этой главе будет рассказано о том, как ASP.NET MVC выполняет маршрутизацию, в результате которой устанавливается соответствие между URL и контроллерами. Методы `Index` и `About`, показанные в листингах 9.1 и 9.2, инициируют метод, который называется `View`. Это делается в соответствии с соглашением об инициации `View` с тем же именем, что и у метода, выполняющего действие. Например, вызов `View` в действии `Index` отобразит представление (View) с именем

Index, а вызов View в методе About, соответственно, отобразит представление с именем About.

Еще один аспект, на который нужно обратить внимание, заключается в том, как действие Index присваивает строку коллекции с именем ViewData. Коллекция ViewData предоставляет для контроллера односторонний способ передать данные модели (Model) представлению (View). Чуть далее в этой главе мы еще вернемся к контроллерам и рассмотрим их подробнее. В том числе, будет показано, каким образом вы можете создавать собственные контроллеры. Однако сейчас перейдем к обсуждению представлений (Views), чтобы ознакомиться с тем, что происходит, когда они инициируются контроллером.

Отображение представлений

Представление (View) — это такой объект, который отображается в браузере и позволяет приложению реагировать на действия пользователя. Представление (View) может отображать любую информацию, которую ему передает контроллер (Controller). Например, обратите внимание на действие Index в листингах 9.1 и 9.2. Это действие присваивает строку "Welcome to ASP.NET MVC!" с ключом "Message" коллекции ViewData.

Из чего состоят представления

На рис. 9.3 уже было показано представление (View), отображенное в браузере. Это представление всего лишь выводит сообщение. HTML-код этого представления показан в листинге 9.3. На практике представление (View) является комбинацией кода на языке разметки гипертекста (HTML) и на языке разметки ASP.NET (ASPX), но далее по тексту главы я, избегая излишнего усложнения, буду называть этот код просто HTML-кодом.

Листинг 9.3. HTML-код объекта представления (View)

```
<%@ Page Language="C#"
         MasterPageFile="~/Views/Shared/Site.Master"
         Inherits="System.Web.Mvc.ViewPage" %>

<asp:Content ID="indexTitle"
             ContentPlaceHolderID="TitleContent"
             runat="server">
    Home Page
</asp:Content>

<asp:Content ID="indexContent"
             ContentPlaceHolderID="MainContent"
             runat="server">

    <h2><%= Html.Encode(ViewData["Message"]) %></h2>
```

```
    <p>
        To learn more about ASP.NET MVC visit
<a href="http://asp.net/mvc"
    title="ASP.NET MVC Website">
    http://asp.net/mvc
</a>.
    </p>
</asp:Content>
```

Беглый взгляд на содержимое листинга 9.3 показывает, что в коде имеется директива `Page` и парочка контейнеров `Content`. Директива `Page` указывает атрибуты `MasterPage` и `Inherits`. `MasterPage` представляет собой отдельный файл, который содержит общий HTML-код, который может отображаться на всех страницах сайта. Вскоре вы увидите, как работает `MasterPage`, а пока сосредоточимся на изучаемом файле, который был представлен в листинге 9.3. ASP.NET MVC скомпилирует этот код в фоновом режиме, и сгенерированный при этом код будет наследовать от класса, указанного атрибутом `Inherits`.

Первый контейнер `Content` может содержать метаданные, которые передаются в заголовок HTML. Второй контейнер `Content` содержит информацию, которая будет отображена на экране. Обратите внимание на директиву `Html.Encode(ViewData["Message"])`, заключенную между связующими тэгами `<%=` и `%>`. Каждый раз, когда вам потребуется добавлять код или получить доступ к коллекции `ViewData`, переданной контроллеру, вам необходимо будет использовать связующие тэги. `Encode` представляет собой один из вспомогательных методов класса `Html`, о котором тоже пойдет речь чуть позже. Цель метода `Encode` заключается в трансляции тэгов HTML в их закодированные представления, что делается по соображениям безопасности. Это гарантирует, что вы не отобразите пользователю никакого потенциально вредоносного кода на JavaScript или другого кода разметки, который имеет возможности исполнения. Строка `ViewData["Message"]` должна быть вам уже знакома, так как она уже присутствовала в действии `Index` (см. листинги 9.1 и 9.2). В листинге 9.3 эта строка считывается и отображается данным представлением на экране.

Организация файлов представления

Структура файловой системы (рис. 9.2) показывает, что представления (Views) располагаются в папке Views и имеют расширение имени файла *.aspx. Каждая вложенная папка в составе папки Views соответствует своему объекту `Controller`, а представления, находящиеся в составе этой папки, соответствуют действиям, выполняемым данным контроллером. Когда контроллер передает управление представлению, вызывая метод `View`, ASP.NET MVC ищет это представление в папке Views по вложенным папкам, которые называются по именам соответствующих им контроллеров, причем имя файла должно тоже совпадать с именем действия, вызывающего это представление.

Обратите внимание, что в составе папки Views имеется вложенная папка Shared. В некоторых случаях вам могут потребоваться такие представления, которые могут совместно использоваться двумя или большим количеством действий контроллера. Эти совместно используемые представления располагаются в подпапке Shared. Каждый раз, когда ASP.NET MVC не находит соответствующего представления во вложенной папке, названной по имени вызывающего контроллера, поиск продолжится в папке Shared. Один из файлов, расположенных во вложенной папке Shared, играет особо важную роль. Это — файл MasterPage, который мы обсудим в следующем разделе.

Назначение файлов MasterPage

Большинство Web-сайтов имеют множество страниц, каждая из которых содержит общие элементы. Все они имеют один и тот же заголовок, меню, боковые панели (sidebars) и нижние колонтитулы формы (footers). Когда вы только приступаете к разработке сайта, вы можете без проблем дублировать это информационное содержимое, но такое дублирование по методу копирования и вставки в будущем может стать источником головной боли. Как только вам потребуется изменить общие элементы, вам потребуется посетить все страницы одну за другой. Это не будет проблемой, если страниц на вашем сайте немного. Однако в реальности большинство более или менее успешных сайтов разрастаются до десятков или даже сотен страниц. Это совершенно непрактично — вручную обновлять каждую из них для того, чтобы изменить один или несколько общих элементов.

Здесь и придут вам на помощь мастер-страницы (`MasterPages`), поскольку они указывают общее хранилище для всех таких элементов. В то же самое время, ваши информационные страницы будут использовать содержимое файла MasterPage. Каждый раз, когда вам потребуется внести изменения в такие информационные элементы, которые должны отображаться на каждой из страниц вашего сайта, достаточно модифицировать файл MasterPage, и это изменение коснется каждой страницы сайта, которая использует мастер-страницу, причем обновление произойдет автоматически. В листинге 9.4 показано содержимое файла MasterPage, автоматически сгенерированного ASP.NET MVC и используемого информационной страницей, показанной в листинге 9.3.

Листинг 9.4. Содержимое автоматически сгенерированного файла MasterPage

```
<%@ Master Language="C#" Inherits="System.Web.Mvc.ViewMasterPage" %>


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title><asp:ContentPlaceHolder ID="TitleContent" runat="server" /></title>
    <link href="../../Content/Site.css" rel="stylesheet" type="text/css" />
</head>
```

```
<body>
    <div class="page">

        <div id="header">
            <div id="title">
                <h1>My MVC Application</h1>
            </div>

            <div id="logindisplay">
                <% Html.RenderPartial("LogOnUserControl"); %>
            </div>

            <div id="menucontainer">

                <ul id="menu">
                    <li><%: Html.ActionLink("Home", "Index", "Home")%></li>
                    <li><%: Html.ActionLink("About", "About", "Home")%></li>
                </ul>

            </div>
        </div>

        <div id="main">
            <asp:ContentPlaceHolder ID="MainContent" runat="server" />

            <div id="footer">
            </div>
        </div>
    </div>
</body>
</html>
```

Последовательно просматривая содержимое листинга 9.4 от начала и до конца, вы обнаружите в верхней части этой страницы директиву `MasterPage`, которая утверждает, что данная страница представляет собой мастер-страницу, и архитектура ASP.NET MVC должна будет использовать ее соответствующим образом. `DTD` представляет собой тэг, который указывает, какие стандарты Web поддерживаются данной страницей. Браузеры читают эту информацию, которая помогает им определять наилучший способ отображения данной страницы.

Остальное содержимое страницы заключено в тэгах HTML и языка разметки ASP.NET MVC. Тэг `html` сообщает о том, что данный документ представляет собой документ HTML. Документы HTML состоят из двух частей, заголовка (head) и тела документа (body). Заголовок предназначен для метаданных, описывающих страницу, а тело документа используется для отображения содержимого документа.

В HTML тэг `div` выделят цепочки кода HTML, что полезно для макетирования и организации страницы. Тэги `Hx`, где *x* представляет собой число из диапазона от 1 до 6, описывают заголовки, причем `h1` — это самый крупный заголовок, а `h6` — самый мелкий.

Элементы управления `ContentPlaceHolder` являются инструментарием, необходимым для успешного использования мастер-страницы. Если вы внимательно посмотрите на тэги `Content` в листинге 9.3, то вы заметите, что они имеют идентификатор `ContentPlaceHolderID`, который совпадает с атрибутами `ID` элементов управления `ContentPlaceHolder` в листинге 9.4. Это значит, что когда происходит визуализация представления (View), то отображается мастер-страница (файл MasterPage), и ASP.NET MVC вставляет содержимое участков `Content` информационной страницы в соответствующие контейнеры `ContentPlaceHolders` мастер-страницы. ASP.NET MVC известно, какую мастер-страницу следует использовать, потому что директива `Page`, как показано в листинге 9.2, указывает атрибут `MasterPage`.

Как вы, наверняка, помните из предыдущих разделов, код в листинге 9.2 имеет связующее выражение для вспомогательного метода `Html.Encode`. Мастер-страница в листинге 9.4 вводит еще два вспомогательных метода `Html`, а именно `RenderPartial` и `ActionLink`.

Метод `ActionLink` имеет три параметра: `id`, `controller` и `action`. Когда метод `ActionLink` визуализируется в браузере, он трансформируется в якорный тэг (anchor tag), `a`, который представляет собой идентификатор `id`, указанный первым параметром метода `ActionLink`. Когда пользователь выполняет щелчок мышью по ссылке в браузере, приложение переходит к контроллеру, указанному третьим параметром метода `ActionLink`, и инициирует действие, заданное вторым параметром метода `ActionLink`. Таким образом, если пользователь щелкнет мышью по ссылке, полученной из вызова `ActionLink("About", "About", "Home")`, ASP.NET MVC инициирует действие `About` в `Home Controller`. В следующем разделе мы подробно рассмотрим вспомогательный метод `RenderPartial`.

Частные представления (они же пользовательские элементы управления)

Довольно часто возникает такая ситуация, когда, написав содержимое представления на одной странице, вы обнаруживаете, что точно такое же самое, идентичное содержимое требуется и на других страницах (двух или большем их количестве). Как уже говорилось в разделе, посвященном мастер-страницам, вам, естественно, захочется избежать рутинной работы по поддержке сайта, когда вам придется вручную обновлять одинаковое содержимое на большом количестве страниц. С одной стороны, мастер-страницы (MasterPages) хорошо подходят для управления информационным содержимым, которое украшает страницы по всему сайту. С другой, существуют еще частные представления (Partial View) — они идеальны для ограниченного повторного использования содержимого представления на различных страницах сайта.

Хорошим примером, демонстрирующим пользу частных представлений, как раз является код, автоматически сгенерированный программой-мастером ASP.NET MVC Project Wizard. В числе прочих файлов, эта программа-мастер создает файл LogonUserControl.ascx. Термины "частное представление" (Partial View) и "пользовательский элемент управления" (User Control) являются синонимами, причем термин "пользовательский элемент управления (User Control) хорошо знаком тем разработчикам, которые имеют опыт работы с предшествующими версиями ASP.NET Web Forms. Термин "частное представление" (Partial View) хорошо согласуется с принятой в ASP.NET MVC концепцией представлений (Views), в рамках которой частное представление является не полным представлением, а лишь частью содержимого представления, которая может использоваться повторно со многими другими представлениями. Такой элемент управления находится в папке Shared не случайно, особенно учитывая то, что его можно повторно использовать на множестве страниц. Не забывайте, если ASP.NET MVC не может найти в папке Views файл, названный по имени соответствующего контроллера, то поиск, как уже говорилось, будет продолжен в папке Shared. В листинге 9.5 показано содержимое файла LogonUserControl.ascx.

Листинг 9.5. Содержимое частного представления

```
<%@ Control Language="C#" Inherits="System.Web.Mvc.ViewUserControl" %>
<%
    if (Request.IsAuthenticated) {
%>
        Welcome <b><%: Page.User.Identity.Name %></b>!
        [ <%: Html.ActionLink("Log Off", "LogOff", "Account") %> ]
<%
    }
    else {
%>
        [ <%: Html.ActionLink("Log On", "LogOn", "Account") %> ]
<%
    }
%>
```

Директива `Control` в первой строке листинга 9.5 указывает на то, что данный файл является частным представлением (Partial View). Внутри элемента управления вы видите утверждение `if`, в котором синтаксические конструкции языка окружены связующими символами `<%` и `%>`. К дополнительным синтаксическим конструкциям, используемым для того, чтобы отделить код от языка разметки, возможно, надо привыкнуть, но для приложений MVC такой подход к управлению отображением разметки типичен. Свойство `IsAuthenticated` объекта `Request` сообщает, зарегистрировался ли пользователь на сервере, а программная логика гарантирует отображение соответствующих сообщений. Вспомогательные методы `Html ActionLink` генерируют тэги действия с URL для действий на контроллере

Account. Чуть ранее мы лишь коснулись вопросов маршрутизации и установления соответствий контроллеров и действий, но в следующем разделе эта тема будет обсуждаться более подробно.

Управление маршрутизацией

ASP.NET MVC имеет систему маршрутизации, которая устанавливает соответствия между URL и контроллерами с помощью действий и параметров, переданных этим действиям. Когда вы создаете новый проект ASP.NET MVC, маршрутизация по умолчанию устанавливается через файл, который называется Global.asax. В этот файл помещается информация о многих событиях, которые влияют на приложение. Когда вы запускаете приложение ASP.NET MVC, оно использует URL следующего вида:

http://domain/controller/action/param1/param2/…/paramN?optionalArg=optionalVal, например: **http://localhost:5611/Home/About**.

В приведенном примере строка **localhost:5611** представляет домен, строка, **Home** указывает контроллер, а строка **About** задает действие. Когда ASP.NET MVC принимает этот URL, создается экземпляр класса HomeController, и вызывается метод About.

Файл Global.asax имеет событие Application_Start, которое вызывается первым при запуске приложения. Именно в этом файле устанавливается маршрутизация таким образом, чтобы она всегда была доступна для всех запросов в течение всего времени, пока приложение работает. В листингах 9.6 и 9.7 показана стандартная, установленная по умолчанию маршрутизация для проекта ASP.NET MVC.

Листинг 9.6. Стандартная маршрутизация для проекта ASP.NET MVC (код на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace MyShopCS
{
    // Note: For instructions on enabling IIS6 or IIS7 classic mode,
    // visit http://go.microsoft.com/?LinkId=9394801

    public class MvcApplication : System.Web.HttpApplication
    {
            public static void RegisterRoutes(RouteCollection routes)
            {
                    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

                    routes.MapRoute(
```

```
                "Default", // Route name
                "{controller}/{action}/{id}", // URL with parameters
                new {
                        controller = "Home",
                        action = "Index",
                        id = "" } // Parameter defaults
            );
        }

        protected void Application_Start()
        {
            RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

Листинг 9.7. Стандартная маршрутизация для проекта ASP.NET MVC (код на VB)

```
' Note: For instructions on enabling IIS6 or IIS7 classic mode,
' visit http://go.microsoft.com/?LinkId=9394802

Public Class MvcApplication
    Inherits System.Web.HttpApplication

    Shared Sub RegisterRoutes(ByVal routes As RouteCollection)
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}")

        ' MapRoute takes the following parameters, in order:
        ' (1) Route name
        ' (2) URL with parameters
        ' (3) Parameter defaults
        routes.MapRoute( _
            "Default", _
            "{controller}/{action}/{id}", _
            New With
            {
                .controller = "Home", .action = "Index", .id = ""
            }
        )
    End Sub

    Sub Application_Start()
        AreaRegistration.RegisterAllAreas()

        RegisterRoutes(RouteTable.Routes)
    End Sub
End Class
```

В листингах 9.6 и 9.7 показано, чтоб событие `Application_Start` инициирует метод с именем `RegisterRoutes`, передавая ему свойство `Routes` класса `RouteTable`. Свойство `Routes` представляет собой статическую коллекцию `RouteCollection`, что говорит о том, что существует только одна копия для всего приложения, и она будет содержать множество маршрутов. Когда приложение запускается, эта коллекция будет пуста, и метод `RegisterRoutes` заполнит ее маршрутами для данного приложения.

Маршрутизация работает за счет сопоставления с образцом (pattern matching), на что указывают два утверждения в методе `RegisterRoutes`: `IgnoreRoute` и `MapRoute`. Метод `IgnoreRoute` полезен в ситуациях, когда вам требуется позволить IIS запросить точный URL. В этом случае, это будет любой файл с расширением *.axd, вне зависимости от параметров.

Метод `MapRoute` показывает общий шаблон для сопоставления URL с контроллерами (controllers), действиями (actions) и параметрами (parameters). Первый параметр представляет собой имя маршрута. Второй параметр описывает шаблон, причем каждая последовательность для сличения выделяется фигурными скобками. На основании такого URL, как, например, **http://localhost:1042/Home/About**, шаблон *{controller}/{action}/{id}* сличает **Home** с {*controller*}, а **About** — с {*action*}; для {*id*} совпадения нет. Следовательно, ASP.NET MVC добавит "*Controller*" к тому сегменту URL, для которого найдено совпадение с шаблоном {*controller*}, а это значит, что будет создан экземпляр контроллера с именем `HomeController`. `About` представляет собой метод внутри класса `HomeController`, который должен быть инициирован. Поскольку метод `About` не имеет параметров, предоставлять их (шаблон {*id*}) нет необходимости.

Третий параметр метода `MapRoute` указывает значения по умолчанию, где ключ совпадает с параметром шаблона, а значение, присваиваемое ключу, представляет собой то значение, которое ASP.NET MVC использует, если не может найти совпадения с URL. Вот два примера, иллюстрирующих, как это работает:

- **http://localhost:1042** инициирует метод `Index` класса `HomeController`, потому что не найдено совпадений ни для контроллера, ни для действия, а значениями по умолчанию являются `Home` для {*controller*} и `Index` — для {*action*}.
- **http://localhost:1042/Home** инициирует метод `Index` класса `HomeController`, потому что не указано никаких действий, а значением по умолчанию для {*action*} является `Index`.

Теперь вы можете создавать собственные индивидуальные маршруты, используя метод `MapRoute` и указывая другие значения по умолчанию для его параметров.

Сборка приложения, предназначенного для работы с базой данных клиентов

Теперь, ознакомившись с базовыми концепциями ASP.NET MVC, приступим к созданию очень простого приложения, предназначенного для работы с клиентской базой данных. С помощью этого приложения можно добавлять в базу данных

информацию о новых клиентах, отображать ее или модифицировать и, наконец, удалять из базы данных записи с информацией о клиентах. При реализации этого проекта вам потребуется создать модель, которая поддерживает базу данных с информацией о клиентах, создать индивидуальный контроллер с действиями по управлению записями о клиентах, а также создать разнообразные представления для управления взаимодействием с пользователями, когда они работают с этой базой данных.

Создание репозитория

Общим образцом, типичным для работы с данными, является построение хранилища (репозитория, repository), которое отвечает за операции, связанные с обработкой данных. Это — еще один способ осуществить разделение задач таким образом, чтобы изолировать логику работы программы в отдельных частях приложения, что позволяет упростить код и работу с ним. Репозиторий представляет собой класс, который позволяет создавать (create), читать (read), изменять (update) и удалять (delete) записи (весь этот набор носит собирательное название операций CRUD) конкретного типа данных. Листинги 9.8 и 9.9 содержат код на C# и VB, соответственно, создающий репозиторий для работы с объектами, содержащими информацию о клиентах. Чтобы создать этот класс, щелкните правой кнопкой мыши по папке Models и из появившегося контекстного меню выберите команды **Add | Class**. Присвойте новому классу имя `CustomerRepository`. В данном коде также подразумевается, что вы предварительно уже создали файл *.dbml для LINQ to SQL, назвали его MyShop и создали сущность `Customer` для таблицы `Customers` в MyShop так, как это уже было описано в *главе 7*.

Листинг 9.8. Создание репозитория для работы с информацией о клиентах (код на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace MyShopCS.Models
{
    public class CustomerRepository
    {
            private MyShopDataContext m_ctx
                = new MyShopDataContext();

            public int InsertCustomer(Customer cust)
            {
                    m_ctx.Customers.InsertOnSubmit(cust);
                    m_ctx.SubmitChanges();
                    return cust.CustomerID;
```

```
}

public void UpdateCustomer(Customer cust)
{
        var currentCust =
            (from currCust in m_ctx.Customers
            where currCust.CustomerID == cust.CustomerID
            select currCust)
            .SingleOrDefault();

        if (currentCust != null)

        {
              currentCust.Age = cust.Age;
              currentCust.Birthday = cust.Birthday;
              currentCust.Income = cust.Income;
              currentCust.Name = cust.Name;
        }

        m_ctx.SubmitChanges();

}

public Customer GetCustomer(int custID)
{
      return
          (from cust in m_ctx.Customers
           where cust.CustomerID == custID
           select cust)
           .SingleOrDefault();
}

public List<Customer> GetCustomers()
{
      return
          (from cust in m_ctx.Customers
           select cust)
           .ToList();
}

public void DeleteCustomer(int custID)
{
      var customer =
          (from cust in m_ctx.Customers
           where cust.CustomerID == custID
```

```
                    select cust)
                    .SingleOrDefault();

                m_ctx.Customers.DeleteOnSubmit(customer);
                m_ctx.SubmitChanges();

            }
        }
}
```

Листинг 9.9. Создание репозитория для работы с информацией о клиентах (код на VB)

```
Public Class CustomerRepository

    Private m_ctx As New MyShopDataContext

        Public Function InsertCustomer(
            ByVal cust As Customer) As Integer

            m_ctx.Customers.InsertOnSubmit(cust)
            m_ctx.SubmitChanges()
            Return cust.CustomerID

        End Function

    Public Sub UpdateCustomer(ByVal cust As Customer)

        Dim currentCust =
           (From currCust In m_ctx.Customers
           Where currCust.CustomerID = cust.CustomerID
           Select currCust).SingleOrDefault()

        If Not currentCust Is Nothing Then

            With currentCust
                .Age = cust.Age
                .Birthday = cust.Birthday
                .Income = cust.Income
                .Name = cust.Name
            End With

            m_ctx.SubmitChanges()

        End If
    End Sub
```

```
    Public Function GetCustomer(ByVal custID As Integer) As Customer

        Dim customer =
            (From cust In m_ctx.Customers
            Where cust.CustomerID = custID
            Select cust).SingleOrDefault()

        Return customer

    End Function

    Public Function GetCustomers() As List(Of Customer)

        Dim customers =
            (From cust In m_ctx.Customers
            Select cust).ToList()

        Return customers

    End Function

    Public Sub DeleteCustomer(ByVal custID As Integer)

        Dim customer =
            (From cust In m_ctx.Customers
            Where cust.CustomerID = custID
            Select cust).SingleOrDefault()

        m_ctx.Customers.DeleteOnSubmit(customer)
        m_ctx.SubmitChanges()

    End Sub

End Class
```

В создаваемом вами репозитории может быть больше методов для выполнения над данными самых различных операций, которые предусматривает ваше приложение. Впрочем, примеры, приведенные в листингах 9.8 и 9.9, можно считать типичными. Операции LINQ to SQL соответствуют материалу, излагавшемуся в *главе 7*, так что нет необходимости повторять здесь тот же самый материал. Цель нашего репозитория заключается в том, чтобы передать контроллеру объект, над которым он сможет работать с данными, не заполняя при этом метода контроллера логикой кода для работы с данными. В следующем разделе мы рассмотрим, как контроллер работает с репозиторием.

Создание контроллера Customer

Щелкните правой кнопкой мыши по папке Controllers и из появившегося контекстного меню выберите команды **Add | Controller**. Альтернативный вариант добиться той же цели заключается в том, чтобы нажать сначала клавиатурную комбинацию <CTRL>+<M>, затем — клавиатурную комбинацию <CTRL>+<C>, а затем присвоить новому файлу CustomerController. Установите флажок **Add action methods for Create, Update, Delete and Details scenarios**, как показано на рис. 9.4.

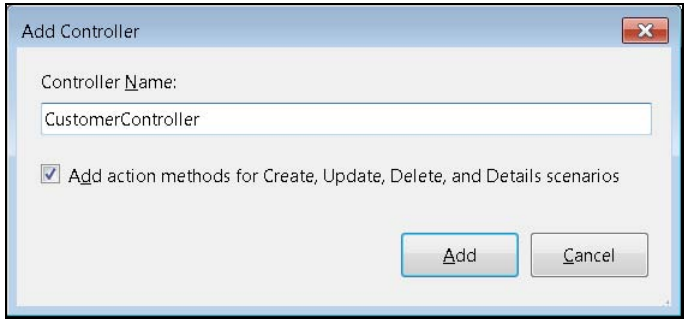


Рис. 9.4. Создание нового контроллера

В результате будет создан новый контроллер, содержащий различные методы для работы с данными репозитория `Customer`. В листингах 9.1 и 9.2 уже было примерно показано, как должен выглядеть контроллер. В данном случае наш контроллер будет выглядеть почти так же, за исключением того, что в нем будет больше методов для различных действий. В последующих нескольких разделах будет пояснено, каким образом следует выполнять различные операции над репозиторием данных о клиентах.

Отображение списка клиентов

Первая операция по работе с информацией о клиентах — это, разумеется, возможность отобразить список клиентов, который послужит отправной точкой для всех остальных операций. В листингах 9.10 и 9.11 показана реализация метода `Index` для контроллера `CustomerController`, которая формирует список клиентов для отображения. Приведенный в этих листингах код использует репозиторий `CustomerRepository`, созданный в предыдущем разделе. Обратите внимание, что для проекта на C# необходимо добавить в начало файла директиву `using` для пространства имен `MyShopCS.Models`.

Листинг 9.10. Контроллер для отображения списка клиентов (код на C#)

```
public ActionResult Index()
{
        var customers =
```

```
            new CustomerRepository()
            .GetCustomers();

        return View(customers);
}
```

Листинг 9.11. Контроллер для отображения списка клиентов (код на VB)

```
Function Index() As ActionResult

    Dim custRep As New CustomerRepository
    Dim customers As List(Of Customer)

    customers = custRep.GetCustomers()
    Return View(customers)
End Function
```

В листингах 9.10 и 9.11 показано, как метод `Index` использует репозиторий информации о клиентах `CustomerRepository`, чтобы получить список заказчиков. Чтобы отобразить этот список, его необходимо передать представлению (View).

Чтобы создать новое представление (View), выполните щелчок правой кнопкой мыши в любой строке метода `Index` и выберите из контекстного меню команду **Add View**. В результате раскроется окно **Add View**, показанное на рис. 9.5.

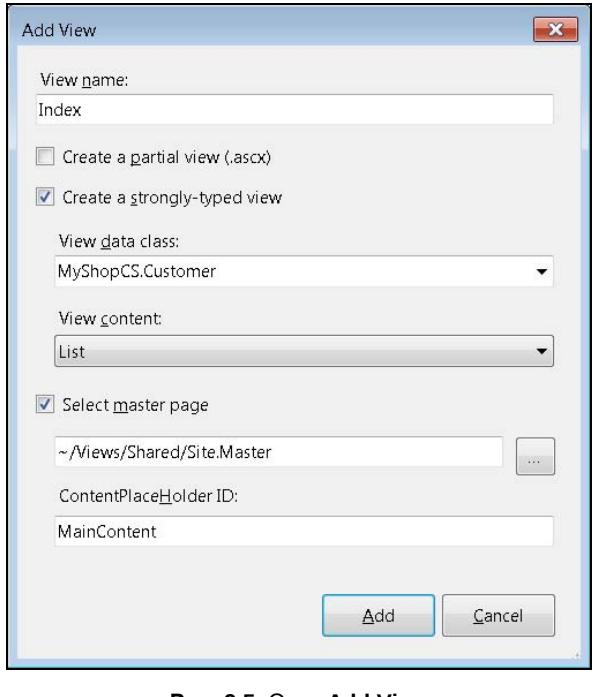


Рис. 9.5. Окно **Add View**

Новое представление имеет имя `Index`, которое соответствует имени метода, инициирующего данное представление. Присвоение представлению имени, соответствующего имени метода, реализующего конкретное действие, является стандартной практикой, принятой по умолчанию, но вы можете назвать новое представление так, как вам будет угодно. Если представление, которое вы хотите отобразить, называется иначе, чем метод, реализующий действие, вы можете воспользоваться следующим вариантом перегрузки метода представления:

```
View("SomeOtherViewName", customers);
```

Нам требуется использовать представление со строгим контролем типов (strongly typed View), а это значит, что вы будете иметь поддержку IDE для ссылок на свойства вашего собственного объекта при работе в представлении. Выбранный объект — это объект `Customer`, который уже определен в качестве сущности LINQ to SQL, который имеет тот же самый тип, что и возвращаемый в результате вызова метода `GetCustomers` в репозитории `CustomerRepository`.

Цель этого представления заключается в отображении списка клиентов, поэтому мы выберем список (**List**) в качестве содержимого представления. В результате представление будет предварительно заполнено шаблоном, предназначенным для отображения списка клиентов. Вы сможете модифицировать экран так, как вам покажется удобным. В дополнение, если вы предпочитаете писать собственный код для заполнения экрана, вы можете выбрать для представления опцию **Empty**, а затем вводить код представления самостоятельно и вручную. Впрочем, если вы выберете опцию **List**, то это ускорит вашу работу, что для начинающих немаловажно.

Чуть ранее в этой главе уже говорилось о мастер-страницах (`MasterPages`), и вы имеете возможность выбрать проедпочтительную для вас мастер-страницу и указать, в каком из элементов управления `ContentPlaceHolder` ваш код будет визуализировать информацию.

Нажмите кнопку **Add**, и в результате будет сгенерирован код, показанный в листинге 9.12.

Листинг 9.12. Представление для отображения списка клиентов

```
<%@ Page Title="" Language="C#"
         MasterPageFile="~/Views/Shared/Site.Master"
         Inherits="System.Web.Mvc
.ViewPage<IEnumerable<MyShopCS.Models.Customer>>" %>

<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>
            CustomerID
        </th>
        <th>
            Name
        </th>
        <th>
            Age
        </th>
        <th>
            Birthday
        </th>
        <th>
            Income
        </th>
    </tr>

<% foreach (var item in Model) { %>

    <tr>
        <td>
            <%= Html.ActionLink("Edit", "Edit",
                new { id=item.CustomerID }) %> |
            <%= Html.ActionLink("Details", "Details",
                new { id=item.CustomerID })%>
        </td>
        <td>
            <%= Html.Encode(item.CustomerID) %>
        </td>
        <td>
            <%= Html.Encode(item.Name) %>
        </td>
        <td>
            <%= Html.Encode(item.Age) %>
        </td>
        <td>
            <%= Html.Encode(String.Format("{0:g}",
                item.Birthday)) %>
        </td>
        <td>
```

```
            </td>
                  <%= Html.Encode(String.Format("{0:F}", item.Income)) %>
            </td>
        /tr>

    <% } %>

    </table>
     <p>
         </asp:Content><%= Html.ActionLink("Create New", "Create") %>
    </p>

</asp:Content>
```

Код, представленный в листинге 9.12, организует список клиентов (`Customers`) в виде таблицы. Тэги `tr` формируют строки, тэги `th` создают заголовки ячеек, а тэги `td` соответствуют самим ячейкам. После указания строки заголовка, цикл `foreach` последовательно пробегает итерациями всю модель и выводит содержимое каждой строки. Вспомним, что в листингах 9.10 и 9.11 действие `Index` вызывало представление (View) с параметром `List<Customer>` (`List(Of Customer)` — в VB). При создании представления мы указали тип объекта как `Customer`, а это означает, что ссылка на `Model` в цикле `foreach` выглядит как `List<Customer>`, и каждый элемент списка будет содержать объект `Customer`.

Для каждой отображаемой ячейки элемент представляет собой текущий объект `Customer`, и свойство объекта `Customer`, который должен быть отображен, ссылается на свойство этой ячейки. Что особенно важно при отображении данных, так это то, что в каждой из ячеек вместо непосредственного отображения данных используется вспомогательный метод `Html.Encode`. Такая практика является наилучшим подходом с точки зрения безопасности, так как это позволяет гарантировать, что отображаемые данные не трактуются как код HTML и никогда не запустят потенциально вредоносный код JavaScript. При чем здесь это? Дело в том, что человек, имеющий враждебные намерения, может добавить код на JavaScript во время ввода данных, и если вы решите отобразить это поле, то браузер может запустить этот код JavaScript, что приведет к весьма нежелательным для вас последствиям. Использование вспомогательного метода `Html.Encode` не позволит этому произойти. Другие вспомогательные методы `Html`, например, `ActionLink`, уже кодируют вывод, поэтому вам следует применять `Html.Encode` в тех случаях, когда не используются никакие другие вспомогательные методы. Обратите внимание, что код для цикла `foreach` заключен между символами `<%` и `%>`, так что он трактуется именно как код, а не как разметка.

Далее, вам, разумеется, потребуется перейти на страницу со списком пользователей через главное меню. Поэтому перейдите на мастер-страницу (MasterPage), которая в данном случае называется `Site.Master`, и добавьте для объекта `Customers` метод `ActionLink`. Фрагмент кода, демонстрирующий, как это делается, приведен в листинге 9.13.

Листинг 9.13. Добавление вспомогательного метода ActionLink для отображения списка клиентов

```
<ul id="menu">
<li><%= Html.ActionLink("Customers", "Index", "Customer")%></li>
<li><%= Html.ActionLink("Home", "Index", "Home")%></li>
<li><%= Html.ActionLink("About", "About", "Home")%></li>
</ul>
```

Параметры, передаваемые новому методу `ActionLink`, в порядке следования слева направо, указывают, что текст привязан к списку клиентов `Customers`, и ASP.NET инициирует метод `Index` по отношению к классу `CustomerController`, если пользователь щелкнет по ссылке. Результат отображения списка клиентов показан на рис. 9.6.

Как показано на рис. 9.6, вкладка **Customer** является в списке первой. Щелчок мышью по этой вкладке отображает список объектов списка `Customers`. Кроме того, как вы можете видеть, в этом списке имеются также ссылки, такие, как **Edit** и **Create New**. Операцию создания (Create) мы рассмотрим в следующем разделе.

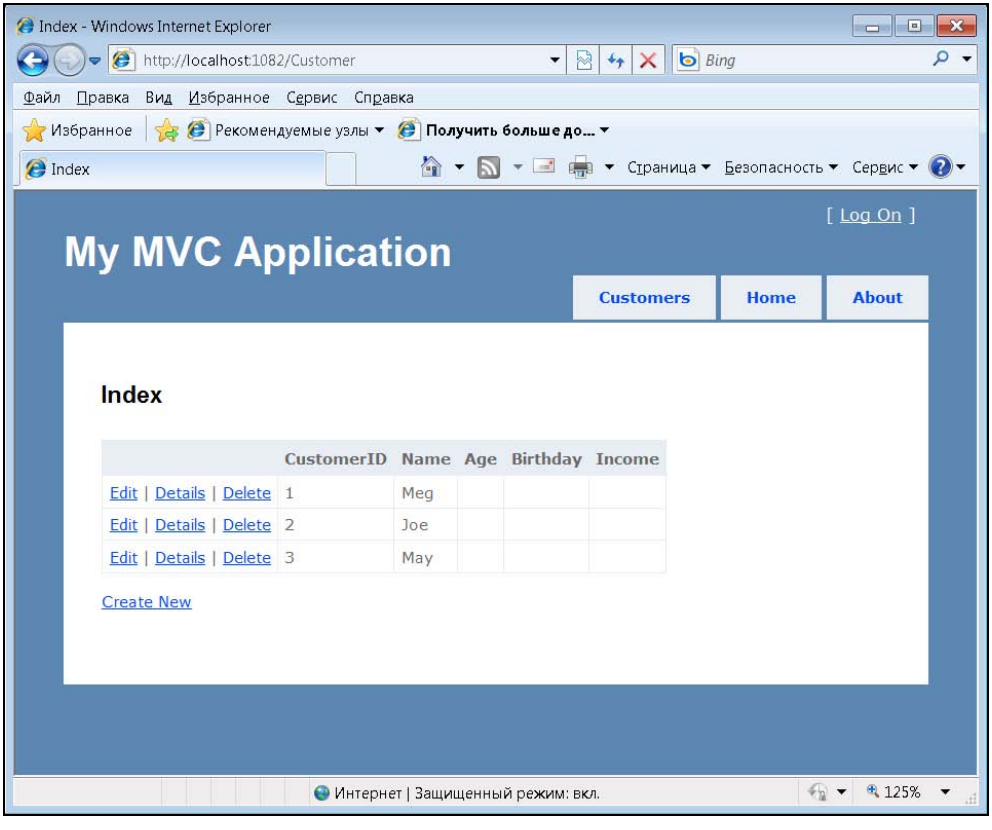


Рис. 9.6. Отображение списка объектов при первом запуске программы

Добавление нового клиента

Операция добавления нового клиента в список подразумевает отображение специального экрана для ввода данных и сохранения их в базе данных после завершения редактирования. При создании нового объекта ваш контроллер (Controller) нуждается в двух методах: в методе `get` для инициализации пустой записи, созданной для нового клиента, и методе `post`, задача которого заключается в сохранении информации о новом заказчике. В листингах 9.14 и 9.15 представлены реализации методов `get` и `post` для класса `CustomerController`.

Листинг 9.14. Создание нового объекта Customer (код на C#)

```
//
// GET: /Customer/Create
public ActionResult Create()
{
        Customer cust = new Customer
        {
                Birthday = new DateTime(1980, 1, 1)
        };

        return View(cust);
}

//
// POST: /Customer/Create
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Create(Customer cust)
{
        try
        {
                if (string.IsNullOrEmpty(cust.Name))
                {
                       ModelState.AddModelError(
                            "Name", "Name is required.");
                            return View();
                }

                new CustomerRepository()
                    .InsertCustomer(cust);

                return RedirectToAction("Index");
        }
        catch
        {
                return View();
        }
}
```

Листинг 9.15. Создание нового объекта Customer (код на VB)

```
' GET: /Customer/Create
Function Create() As ActionResult
    Dim cust As New Customer With
    {
          .Birthday = New DateTime(1980, 1, 1)
    }
    Return View(cust)
End Function

'
' POST: /Customer/Create
<HttpPost()> _
Function Create(ByVal cust As Customer) As ActionResult
    Try
            If String.IsNullOrEmpty(cust.Name) Then
                ModelState.AddModelError(
                     "Name", "Name is required.")
            End If

            Dim custRep As New CustomerRepository
            custRep.InsertCustomer(cust)

            Return RedirectToAction("Index")

    Catch
           Return View()
    End Try

End Function
```

В протоколе HTTP существуют различные типы методов для проведения различных операций. В листингах 9.14 и 9.15 продемонстрировано использование двух из них: `get` и `post`. Метод `get`, как правило, связан со считыванием данных, а метод **post** — с их записью. В приведенном примере используются оба метода, перегруженные в составе метода `Create`. В ASP.NET MVC, методы, определяющие действия, по умолчанию сводятся к запросам `get`, поэтому для применения метода `post` необходимо указывать атрибут `HttpVerbs`.

Действие `get` в составе метода `Create` инициирует новый объект `Customer` и передает его представлению (`View`). Когда пользователь заполнит форму и даст команду на фиксацию изменений (submit), то в составе метода `Create` будет выполнено метод-действие `post`, которое вставит новую запись в базу данных.

Обратите внимание, что в этом коде я изменил параметр метода `Create` с `FormsCollection` на `Customer`. ASP.NET MVC автоматически прочтет значения, введенные в форму, и сверит эти значения с совпадающими свойствами объекта,

переданного методу. Кроме того, метод проверяет, заполнено ли поле имени, и добавляет ошибку для `ModelState`. Когда возникает такая ошибка, необходимо вернуться назад к представлению, с тем, чтобы пользователь мог обнаружить свою ошибку ввода данных, исправить ее, и затем повторно попытаться сохранить внесенные изменения. ASP.NET MVC использует эту ошибку для вывода сообщений об ошибках в представлении.

Чтобы создать представление для ввода данных о новом клиенте, щелкните правой кнопкой мыши в любой строке метода `Create`, выберите из контекстного меню команду **Add View** и заполните поля раскрывшегося окна так, как показано на рис. 9.7.

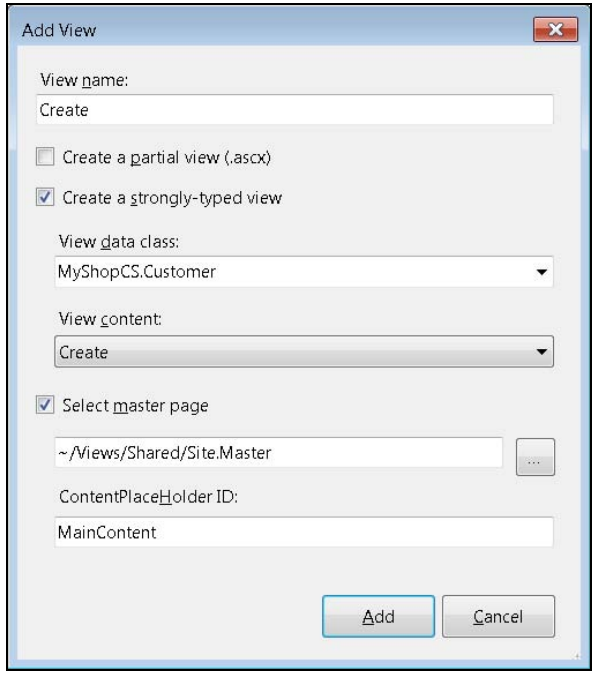


Рис. 9.7. Создание представления для ввода информации о новом клиенте

Обратите внимание, что в окне **Add View**, показанном на рис. 9.7, требуется создать представление со строгим контролем типов (установлен флажок **Create a strongly-typed view**) для класса `Customer`, но на этот раз в поле раскрывающегося списка **View content** выбрана опция **Create**. Нажмите кнопку **Add**, и в результате VS создаст представление, код которого приведен в листинге 9.16.

Листинг 9.16. Код представления, предназначенного для ввода информации о новом клиенте

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<MyShopCS.Customer>" %>


<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
```

```
     runat="server">
     Create
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

    <h2>Create</h2>

    <% using (Html.BeginForm()) {%>
        <%: Html.ValidationSummary(true) %>

        <fieldset>
            <legend>Fields</legend>

            <div class="editor-label">
                <%: Html.LabelFor(model => model.CustomerID) %>
            </div>
            <div class="editor-field">
                <%: Html.TextBoxFor(model => model.CustomerID) %>
                <%: Html.ValidationMessageFor(model => model.CustomerID) %>
            </div>

            <div class="editor-label">
                <%: Html.LabelFor(model => model.Name) %>
            </div>
            <div class="editor-field">
                <%: Html.TextBoxFor(model => model.Name) %>
                <%: Html.ValidationMessageFor(model => model.Name) %>
            </div>

            <div class="editor-label">
                <%: Html.LabelFor(model => model.Age) %>
            </div>
            <div class="editor-field">
                <%: Html.TextBoxFor(model => model.Age) %>
                <%: Html.ValidationMessageFor(model => model.Age) %>
            </div>

            <div class="editor-label">
                <%: Html.LabelFor(model => model.Birthday) %>
            </div>
            <div class="editor-field">
                <%: Html.TextBoxFor(model => model.Birthday) %>
                <%: Html.ValidationMessageFor(model => model.Birthday) %>
            </div>

            <div class="editor-label">
                <%: Html.LabelFor(model => model.Income) %>
            </div>
```

```
            <div class="editor-field">
                <%: Html.TextBoxFor(model => model.Income) %>
                <%: Html.ValidationMessageFor(model => model.Income) %>
            </div>

            <p>
                <input type="submit" value="Create" />
            </p>
        </fieldset>

    <% } %>

    <div>
        <%: Html.ActionLink("Back to List", "Index") %>
    </div>

</asp:Content>
```

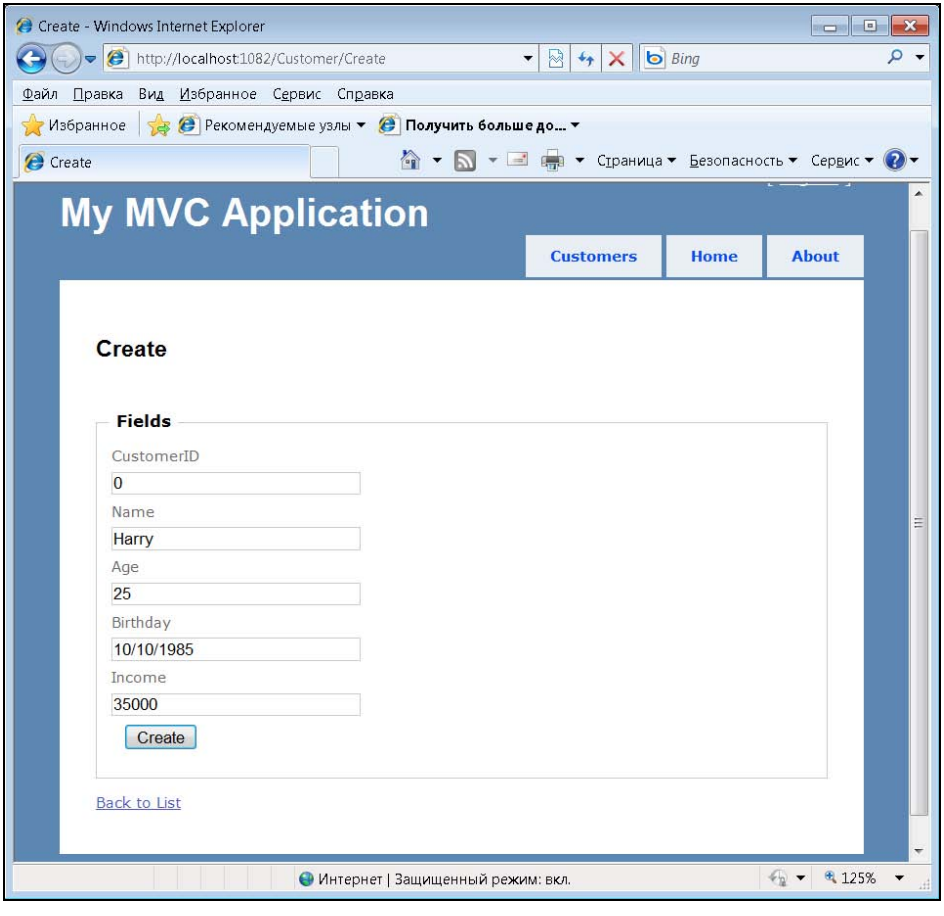


Рис. 9.8. Экран приложения во время работы метода `Create`

Вспомогательный метод `Html.ValidationMessageFor` отображает сообщения об ошибках, которые происходят во время обработки этой страницы. Сообщения об ошибках отображаются всякий раз, когда метод-действие контроллера добавляет ошибку к `ModelState`. Когда пользователь нажимает кнопку **Submit**, эта страница возвращается обратно методу `Create` вместе с атрибутом `AcceptVerbs`. Экран приложения в момент работы метода `Create` показан на рис. 9.8.

Кроме создания записей для новых клиентов, вы имеете возможность редактирования записей для уже существующих клиентов. О том, как это делается, рассказывается в следующем разделе.

Редактирование записей для существующих клиентов

По аналогии с тем, как мы создаем новые учетные записи для вновь регистрируемых клиентов, нам необходимо время от времени редактировать записи для имеющихся клиентов. Для этого нам тоже требуются два метода. Метод `get` заполняет отображенную форму редактирования уже имеющимися данными, а метод `post` служит для сохранения внесенных изменений. Использование этих двух методов для редактирования информации о клиентах продемонстрировано в листингах 9.17 и 9.18.

Листинг 9.17. Методы для редактирования записей о клиентах (код на C#)

```
//
// GET: /Customer/Edit/5

public ActionResult Edit(int id)
{
        Customer cust =
            new CustomerRepository()
                .GetCustomer(id);

        return View(cust);
}

//
// POST: /Customer/Edit/5

[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Edit(Customer cust)
{
        try
        {
                new CustomerRepository()
                    .UpdateCustomer(cust);

                return RedirectToAction("Index");
        }
```

```
        catch
        {
                return View();
        }
}
```

Листинг 9.18. Методы для редактирования записей о клиентах (код на VB)

```
'
' GET: /Customer/Edit/5

Function Edit(ByVal id As Integer) As ActionResult
         Dim custRep As New CustomerRepository
         Dim cust As Customer

         cust = custRep.GetCustomer(id)

         Return View(cust)
End Function
'
' POST: /Customer/Edit/5

<HttpPost()> _
Function Edit(ByVal id As Integer, ByVal cust As Customer)
    As ActionResult

    Try
          Dim custRep As New CustomerRepository
          custRep.UpdateCustomer(cust)

          Return RedirectToAction("Index")
    Catch
          Return View()
    End Try
End Function
```

В методе-действии `get` метода `Edit` вам необходимо получить ссылку на текущую запись, на которую указывает передаваемый `id`, и передать эту ссылку представлению `View` для отображения. Метод-действие `post` метода `Edit` принимает модифицированную информацию о клиенте и передает ее в репозиторий для обновления базы данных. В данном случае вам тоже необходимо создать представление, щелкнув правой кнопкой мыши в любой из строк метода `Edit` и выбрав из контекстного меню команду `Add View`. Создайте представление со строгим контролем типов, установите класс на `Customer`, а в поле раскрывающегося списка **Content type** выберите опцию **Edit**.

Наконец, нам осталось рассмотреть только одну операцию — удаление записи о клиенте из базы данных. Эту операцию мы рассмотрим в следующем разделе.

Удаление записи о клиенте из базы данных

Стандартный шаблон по умолчанию, применяемый для создания списка клиентов, добавляет и метод для отображения детальной информации о клиентах (`Details`), который работает по аналогии с методом для редактирования, рассмотренным в предыдущем разделе. Если вас интересует, вы можете создать нередактируемые страницы с детальной информацией о каждом клиенте, но для наших целей опция **Details** не особо нужна. Поэтому перейдем сразу же к рассмотрению метода, предназначенного для удаления записи о клиенте из базы данных. Соответствующие методы для C# и VB представлены в листингах 9.19 и 9.20.

Листинг 9.19. Метод для удаления записей о клиентах из базы данных (код на C#)

```
//
// GET: /Customer/Delete/5

public ActionResult Delete(int id)
{
        new CustomerRepository()
            .DeleteCustomer(id);

        TempData["Result"] = "Customer Deleted.";

        return RedirectToAction("Index");
}
```

Листинг 9.20. Метод для удаления записей о клиентах из базы данных (код на VB)

```
'
' GET: /Customer/Delete/5

Function Delete(ByVal id As Integer) As ActionResult
    Dim custRep As New CustomerRepository
    custRep.DeleteCustomer(id)

    TempData("Result") = "Customer Deleted."

    Return RedirectToAction("Index")
End Function
```

Помимо демонстрации использования репозитория для выполнения операции удаления, в листингах 9.19 и 9.20 имеются парочка новых элементов, о которых вам необходимо знать: `TempData` и метод создания представления (View). `TempData` представляет собой специальный объект для хранения данных в виде единого отображения представления. Поэтому, когда представление отобразится, оно сможет прочесть текущее значение `TempData`, но то же самое значение уже не будет дос-

тупно в следующем представлении, за исключением случаев, когда контроллер явно загрузит его снова.

Во всех остальных вызовах к представлению предполагалось, что будет возвращено представление, названное по имени метода контроллера, поэтому не было необходимо явно указывать имя представления. Однако представления для метода `Delete` не существует, поэтому нам необходимо явно указать представление `Index`.

Чтобы решить эту проблему с операцией удаления, в представление Index.aspx (расположенное в составе \Views\Customer) были внесены изменения, показанные в листингах 9.21 и 9.22.

Листинг 9.21. Удаление клиента из базы данных (код на C#)

```
... content removed
<h2>Index</h2>

<p>
    <% if (TempData["Result"] != null)
       { %>
            <label><%= Html.Encode(TempData["Result"].ToString() )%>
</label>
    <% } %>

</p>
<table>

... content removed

<% foreach (var item in Model) { %>
    <tr>
        <td>
            <%= Html.ActionLink("Edit", "Edit",
                new { id=item.CustomerID }) %> |
            <%= Html.ActionLink("Delete", "Delete",
                new { id=item.CustomerID })%>
        </td>

... content removed
```

Листинг 9.22. Удаление клиента из базы данных (код на VB)

```
... content removed

    <h2>Index</h2>

    <p>
        <% If Not TempData("Result") Is Nothing Then %>
```

```
            <label>
                <%= Html.Encode(TempData("Result").ToString())%>
            </label>
        <% End If%>
    </p>

    <p>
        <%= Html.ActionLink("Create New", "Create")%>
    </p>

<table>

... content removed

    <% For Each item In Model%>

        <tr>
             <td>
                <%=Html.ActionLink("Edit", "Edit",
                   New With {.id = item.CustomerID})%> |
                <%=Html.ActionLink("Delete", "Delete",
                   New With {.id = item.CustomerID})%>
            </td>

... content removed
```

Часть кода из листингов 9.21 и 9.22 была удалена во избежание повторов. Ближе к началу листингов вы видите утверждение `if`, которое проверяет, присутствует ли значение в `TempData["Result"]` (`TempData("Result")` — в VB) и отображает это значение в метке, если оно присутствует. Далее, вспомогательные методы `ActionLink` для редактирования заменены на методы для удаления и предают `id` записи текущего клиента контроллеру для удаления.

Заключение

Итак, в данной главе вы познакомились с основными компонентами MVC: моделями (Models), представлениями (Views) и контроллерами (Controllers). Далее, было продемонстрировано создание репозитория для доступа к данным, который упрощает кодирование. Кроме того, в этой главе было показано, как создавать контроллеры и представления. Обсуждение было продолжено темой маршрутизации и объяснения механизма, используемого для установления соответствий между URL, контроллерами, действиями и параметрами. Наконец, в заключение был приведен простой пример выполнения операций CRUD с помощью ASP.NET MVC.

Еще одна популярная Web-технология — это Silverlight. С помощью этой технологии вы получаете средства упростить работу для конечных пользователей. Обсуждению Silverlight будет посвящена следующая глава.

Глава 10

Разработка приложений Silverlight

В данной главе будут рассмотрены следующие ключевые концепции:

- Запуск нового проекта Silverlight;
- Работа в визуальном редакторе Silverlight Designer;
- Добавление элементов управления в состав приложения;
- Воспроизведение видео в Silverlight;
- Развертывание приложений Silverlight.

Silverlight — это Web-технология, позволяющая добавить в состав Web-приложения новые, богатые возможности. Эта технология, как и WPF, использует XAML, но приложения Silverlight работают на Web-страницах, поддерживаемых ASP.NET.

Предшествующие главы были призваны подготовить вас к работе с Silverlight. Поскольку Silverlight использует XAML, перед прочтением этой главы рекомендуется просмотреть *приложения 1* и *2*, так как это поможет освежить в памяти основы XAML (или ознакомиться с ними, если вы до сих пор этого не сделали). Silverlight имеет много общего с WPF. Поэтому перед чтением этой главы полезно будет также вернуться к *главе 8* и бегло ее просмотреть. В данной главе будет рассказано, как создать новый проект Silverlight, как добавить к нему новые элементы управления и как распространять приложения Silverlight.

Запуск проекта Silverlight

Как и при запуске других проектов, чтобы создать новый проект Silverlight, выберите из меню команды **File | New | Project** или нажмите клавиатурную комбинацию <CTRL>+<SHIFT>+<N>. Затем в раскрывшемся окне **New Project** выберите опцию **Silverlight application**. Далее вам, как обычно, потребуется дать имя новому проекту и указать путь к его папке. После этого VS отобразит окно для конфигурирования нового приложения Silverlight, показанное на рис. 10.1.

Silverlight дает вам возможности создать Web-сайт одновременно с приложением Silverlight. Вы можете и не создавать сайт, но ваше приложение Silverlight в обязательном порядке нуждается в хостинге на Web-странице. В рассматриваемом примере мы создадим и Web-сайт, поэтому установите флажок **Host the Silverlight application in a new Web site**. Далее вам потребуется выбрать тип проекта **Web.**

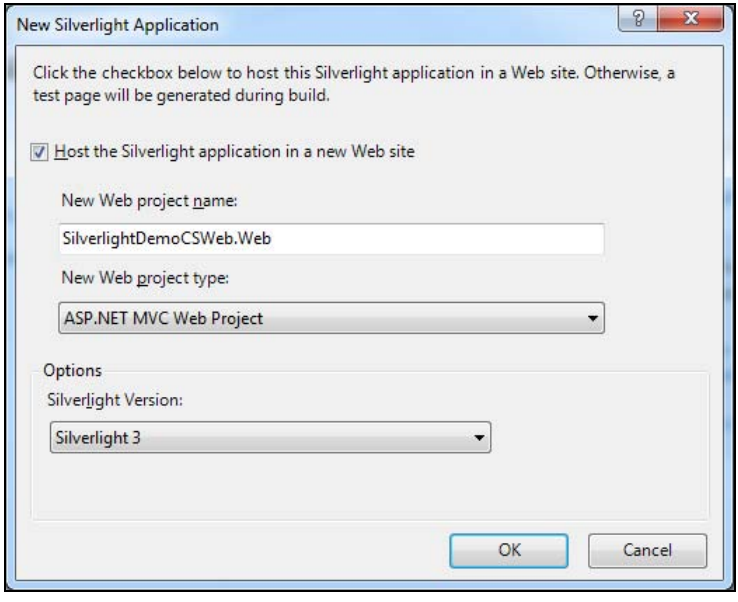


Рис. 10.1. Создание нового приложения Silverlight

Visual Studio 2010 позволяет создавать проекты трех типов — **ASP.NET Web application project** (Проект Web-приложения ASP.NET), **ASP.NET Web Site** (Web-сайт ASP.NET) и **ASP.NET MVC Web Project** (Web-проект ASP.NET MVC). Первые две опции используют Web-технологию ASP.NET, основанную на Web-формах. Поскольку данная книга уделяет основное внимание модели Web-разработки ASP.NET MVC, обсуждавшейся в *главе 9*, вам следует выбрать опцию **ASP.NET MVC Web Project**. Нажмите кнопку **OK**, чтобы создать приложение Silverlight, показанное на рис. 10.2.

На экране появится окно с сообщением, запрашивающим, хотите ли вы создать проект модульных тестов (unit test project), которое выглядит так, как показано на рис. 10.3. Как уже говорилось в *главе 9*, проекты модульных тестов в данной книге не рассматриваются, поэтому выбор опций в данном окне большого значения не имеет. Впрочем, самостоятельным и любознательным новичкам я рекомендую создать проект модульных тестов и поэкспериментировать с ним самостоятельно. Для продолжения нажмите кнопку **OK**.

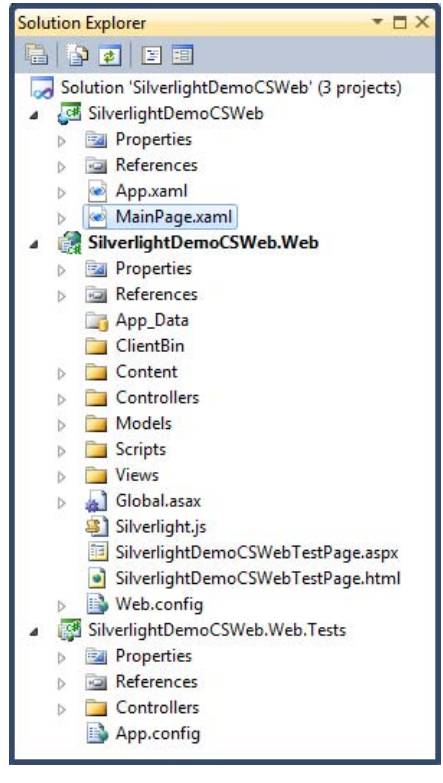


Рис. 10.2. Новый проект Silverlight

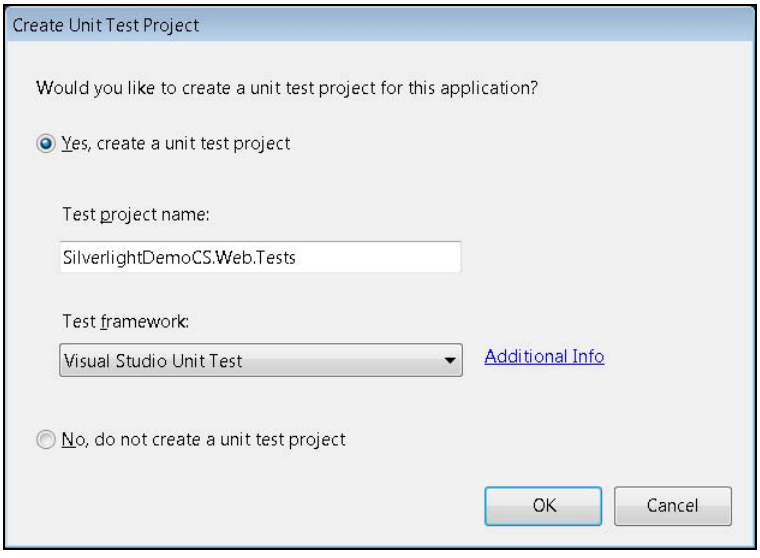


Рис. 10.3. Окно **Create Unit Test Project**

Аналогично приложениям WPF, приложения Silverlight начинаются с файла MainPage.xaml и файла App.xaml, где App.xaml запускается, чтобы инициализировать приложение, а файл MainPage.xaml содержит отображаемую страницу. Web-сайт представляет собой типичное приложение ASP.NET MVC, с тем исключением, что оно имеет тестовую страницу, которая обеспечивает хостинг приложению Silverlight, SilverlightDemoCSTestPage.aspx (SilverlightDemoVBTestPage.aspx — для VB). Кроме того, имеется еще файл SilverlightDemoCSTestPage.html (SilverlightDemoVBTestPage.html — для VB), который выполняет ту же самую функцию, что и SilverlightDemoCSTestPage.aspx (SilverlightDemoVBTestPage.aspx — для VB), а именно обеспечивает хостинг приложению Silverlight, за исключением того, что версия *.html использует JavaScript и тэг объекта HTML для хостинга Silverlight. В листинге 10.1 показано содержимое тестовой страницы и продемонстрировано, как она обеспечивает хостинг для приложения Silverlight. Этот листинг содержит код XAML, который нейтрален по отношению к языку и работает совершенно одинаково, вне зависимости от того, программируете ли вы на C# или на VB.

Листинг 10.1. Предоставление хостинга приложению Silverlight на Web-странице

```
<%@ Page Language="C#" AutoEventWireup="true" %>

<!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>SilverlightDemoCSWeb</title>
```

```
<style type="text/css">
html, body {
        height: 100%;
        overflow: auto;
}
body {
        padding: 0;
        margin: 0;
}
#silverlightControlHost {
        height: 100%;
        text-align:center;
}
</style>
<script type="text/javascript" src="Silverlight.js"></script>
<script type="text/javascript">
    function onSilverlightError(sender, args) {
        var appSource = "";
        if (sender != null && sender != 0) {
          appSource = sender.getHost().Source;
        }

        var errorType = args.ErrorType;
        var iErrorCode = args.ErrorCode;

        if (errorType == "ImageError" || errorType == "MediaError") {
          return;
        }

        var errMsg = "Unhandled Error in Silverlight Application "
                     + appSource + "\n" ;

        errMsg += "Code: "+ iErrorCode + "    \n";
        errMsg += "Category: " + errorType + "       \n";
        errMsg += "Message: " + args.ErrorMessage + "     \n";

        if (errorType == "ParserError") {
            errMsg += "File: " + args.xamlFile + "     \n";
            errMsg += "Line: " + args.lineNumber + "     \n";
            errMsg += "Position: " + args.charPosition + "     \n";
        }
        else if (errorType == "RuntimeError") {
            if (args.lineNumber != 0) {
                errMsg += "Line: " + args.lineNumber + "     \n";
                errMsg += "Position: " +  args.charPosition + "     \n";
            }
```

```
                errMsg += "MethodName: " + args.methodName + "     \n";
            }

            throw new Error(errMsg);
        }
    </script>
</head>
<body>
    <form id="form1" runat="server" style="height:100%">
    <div id="silverlightControlHost">
        <object data="data:application/x-silverlight-2, "
                type="application/x-silverlight-2"
                width="100%" height="100%">
                  <param name="source" value="ClientBin/SilverlightDemoCSWeb.xap"/>
                  <param name="onError" value="onSilverlightError" />
                  <param name="background" value="white" />
                  <param name="minRuntimeVersion" value="3.0.40818.0" />
                  <param name="autoUpgrade" value="true" />
                  <a href="http://go.microsoft.com/fwlink/?LinkID=149156&v=3.0.40818.0"
style="text-decoration:none">
          <img src="http://go.microsoft.com/fwlink/?LinkId=161376"
               alt="Get Microsoft Silverlight" style="border-style:none"/>
                  </a>
            </object>
            <iframe id="_sl_historyFrame"
                    style="visibility:hidden;height:0px;width:0px;border:0px">
            </iframe>
</div>
</form>
</body>
</html>
```

В листинге 10.1 содержится тэг объекта, который обеспечивает хостинг приложению Silverlight. Этот тег объекта имеет различные параметры, краткое описание которых дано в табл. 10.1.

Таблица 10.1. *Параметры тэгов объекта (Object Tag) для Silverlight*

Параметр	Описание
source	На рис. 10.2 показана папка ClientBin в составе проекта Web-приложения ASP.NET MVC. При выполнении сборки проекта Silverlight, VS поместит вывод проекта в папку ClientBin. Вывод скомпилированного проекта Silverlight представляет собой *.xap-файл. Этот файл фактически, представляет собой то же самое, что и сжатый *.zip-файл, но только с другим именем. Silverlight загружает *.xap-файл в браузер во время исполнения и запускает приложение

Таблица 10.1 *(окончание)*

Параметр	Описание
`onerror`	В листинге 10.1 имеется функция JavaScript `onSilverlightError`, которая вызывается каждый раз, когда в Silverlight встречается ошибка
`background`	Устанавливает фон элемента управления
`minRuntimeVersion`	Указывает, что пользователь должен установить на своем компьютере версию v3.0.40818.0 или более позднюю для подключаемого модуля Silverlight, чтобы приложение могло работать. Если требования к минимально необходимой версии не соблюдены, пользователь получит сообщение об ошибке
`autoUpgrade`	Если пользователь не имеет в своем распоряжении минимально необходимой версии, указанной настройки `minRuntimeVersion`, то установка для данного параметра значения true приведет к тому, что пользователю будет предложено начать процесс обновления версий

Вы можете запустить приложение и просмотреть Web-страницу, но увидеть пока сможете не так уж много. В следующем разделе мы приступим собственно к разработке приложения и добьемся, чтобы оно действительно делало что-то полезное. И начнем мы с рассмотрения визуального редактора (Designer).

Навигация в окне Silverlight Designer

Технология, на которой основывается отображение пользовательского интерфейса — это язык XML Application Markup Language (XAML), произносится как "Zamel". В *приложении 1* приведена вводная информация, необходимая для понимания кода, написанного на языке XML, а в *приложении 2* — вводные сведения по XAML. Если вы чувствуете, что понимать излагаемый материал вам трудно, прочтите эти два приложения — они помогут вам быстро войти в курс дела. Кроме того, очень полезно будет вернуться и к *главе 8*, потому что там вы найдете немало информации об элементах управления, а также о макетах окон, которые применяются как в Silverlight, так и в WPF.

Окно визуального конструктора Silverlight (Silverlight Designer) очень похоже на уже знакомое вам окно визуального конструктора WPF (WPF Designer), особенно это касается работы с элементами управления. Перетаскивание элементов управления из окна **Toolbox** в область визуального проектирования, конфигурирование сеток (Grids), взаимодействие с XAML и установка свойств (properties) в Silverlight осуществляются точно так же, как и в случае работы с WPF. Поскольку общих черт так много, я не буду повторять еще раз здесь все то, что уже было описано в *главе 8*. Дальнейшее изложение строится на основе уже обсуждавшихся материалов, а здесь будут показаны только важные отличия, существующие между WPF и Silverlight.

Использование элементов управления Silverlight

Silverlight обеспечивает мультимедийную поддержку — в том числе, потоковых видео и аудио. В окне есть элементы управления, которые упрощают обеспечение хостинга для ваших видео и управляют возможности просмотра ваших видеофайлов пользователями. Чтобы сконструировать окно для воспроизведения видео, показанное на рис. 10.4, проделайте следующее.

- Стартовой страницей вашего проекта является страница MainPage.xaml, которую вам нужно открыть таким образом, чтобы она отображалась в области визуального конструирования. Если на экран редактора выводится код XAML, щелкните мышью по вкладке **Design**, которая расположена в нижней части окна.
- Перетащите в окно стандартный макет **Grid**, предлагаемый по умолчанию, с которым вы можете работать совершенно аналогично тому, как это делается при разработке приложений WPF, обсуждавшихся в *главе 8*. Вам необходимо убедиться в том, что сетка таблицы имеет две строки, причем верхняя строка должна иметь размер, достаточный для того, чтобы в ней можно было расположить элемент управления **MediaElement**. Нижняя строка тоже должна быть достаточно большой, чтобы в ней можно было поместить командную кнопку (элемент управления **Button**). Задержите курсор над левой границей окна до тех пор, пока вы не увидите, что в окне появилась линия сетки. Перемещайте эту линию до тех пор, пока вы не увидите, что таблица разбита на две строки нужного вам размера, как показано на рис. 10.3. Когда таблица будет разбита нужным образом, щелкните мышью по границе окна.
- Перейдите в окно **Toolbox** и найдите в нем элемент управления **MediaElement**. Перетащите его в верхний ряд сетки таблицы, пользуясь стандартной технологией drag-and-drop. Если вы видите, что высота верхней строки недостаточна для того, чтобы в ней можно было разместить **MediaElement**, подхватите мышью значок стрелки на левой границе сетки и оттяните его еще немного вниз.
- Установите свойство **Name** для нового элемента управления **MediaElement** — в нашем примере это будет имя **VideoPlayer**.
- Элемент управления **MediaElement** имеет свойство **Source**, которое вы можете задать таким образом, чтобы оно содержало URL, указывающий на расположение видеоролика. В нашем примере я советую установить для этого свойства следующий URL: **http://mschnlnine.vo.llnwd.net/d1/ch9/8/3/7/0/7/4/OfficeVS10SC1_2MB_ch9.wmv**, который ссылается на ролик с презентацией VS 2010.
- Перетащите кнопку (элемент управления **Button**) из окна **Toolbox** в нижнюю часть окна визуального конструктора.
- Установите свойство **Name** для новой кнопки на **StartStopButton**, а свойство **Content** — на **Start**.

На рис. 10.4 показан макет сетки (**Grid**) с двумя строками. Верхняя строка содержит элемент управления **MediaElement**, а нижняя — командную кнопку. Имя элемента управления видео — **VideoPlayer**, при этом имя кнопки, нажатие на которую начинает показ видео — **StartStopButton**.

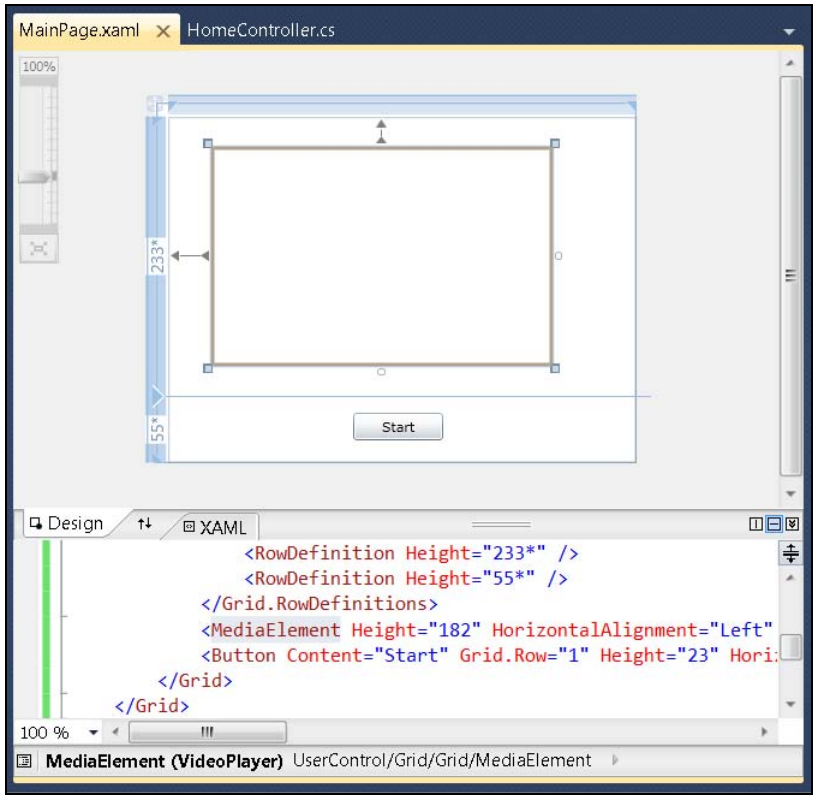


Рис. 10.4. Проект приложения Silverlight для воспроизведения видео

Двойной щелчок мышью по кнопке **StartStopButton** генерирует обработчик события `Click` в файле поддержки кода (code-behind) MainPage.xaml.cs, показанный в листингах 10.2 и 10.3.

Листинг 10.2. Начало и остановка воспроизведения видео в Silverlight (код на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace SilverlightDemoCS
```

```
{
        public partial class MainPage : UserControl
        {
                public MainPage()
                {
                        InitializeComponent();
                        VideoPlayer.AutoPlay = false;
                }

                private bool m_isPlaying = false;

                private void StartStopButton_Click(
                    object sender, RoutedEventArgs e)
                {
                        if (m_isPlaying)
                        {
                                VideoPlayer.Stop();
                                StartStopButton.Content = "Start";
                                m_isPlaying = false;
                        }
                        else
                        {
                                VideoPlayer.Play();
                                StartStopButton.Content = "Stop";
                                m_isPlaying = true;
                        }
                }
        }
}
```

Листинг 10.3. Начало и остановка воспроизведения видео в Silverlight (код на VB)

```
Partial Public Class MainPage
        Inherits UserControl

        Public Sub New()
                InitializeComponent()
                VideoPlayer.AutoPlay = False
        End Sub

        Dim m_isPlaying As Boolean = False

        Private Sub StartStopButton_Click(
                ByVal sender As System.Object,
```

```
                ByVal e As System.Windows.RoutedEventArgs)

                If (m_isPlaying) Then
                    VideoPlayer.Stop()
                    StartStopButton.Content = "Start"
                    m_isPlaying = False
                Else
                    VideoPlayer.Play()
                    StartStopButton.Content = "Stop"
                    m_isPlaying = True
                End If
        End Sub
End Class
```

По умолчанию, элемент управления **MediaElement** начинает воспроизводить видео из источника, на который ссылается свойство **Source**, сразу же после загрузки приложения. Поэтому в конструкторе поддерживающего кода я установил свойство **AutoPlay** на значение `false`. Поле `m_isPlaying` отслеживает статус элемента **MediaElement** — производится воспроизведение или нет. Обработчик события `Click` использует поле `m_isPlaying` для переключения между состояниями воспроизведения (`playing`) и паузы (`stopped`).

Это была всего лишь краткая демонстрация, показывающая, как следует работать с элементом управления **MediaElement**, но на самом деле вы можете выполнять намного большее количество операций, например, приостанавливать воспроизведение, буферизовать трекинг (tracking buffering), проверять позицию видео (checking video position) и многое другое.

Все, что вам требуется делать — это либо захватывать события от элемента управления **MediaElement**, либо применять элементы управления наподобие кнопок (buttons) и ползунков (sliders), чтобы взаимодействовать с элементом управления **MediaElement**, как показано в примере, приведенном в листингах 10.2 и 10.3. Попробуйте проделать полезное упражнение — возьмите за основу приведенный здесь код и добавьте дополнительные функциональные возможности к элементу управления **MediaElement**.

Запуск Silverlight "вне браузера" (Out-of-Browser, OOB)

Новой возможностью Silverlight 3 является запуск приложений "вне браузера" (out-of-browser, OOB). Под этим понимается то, что пользователь может загружать ваше приложение на свой рабочий стол без необходимости посещать сайт, предоставляющий хостинг. Чтобы реализовать OOB, откройте окно свойств приложения Silverlight, выполнив двойной щелчок мышью по папке **Properties** в окне **Solution Explorer**. Вы увидите окно, примерно похожее на изображенное на рис. 10.5.

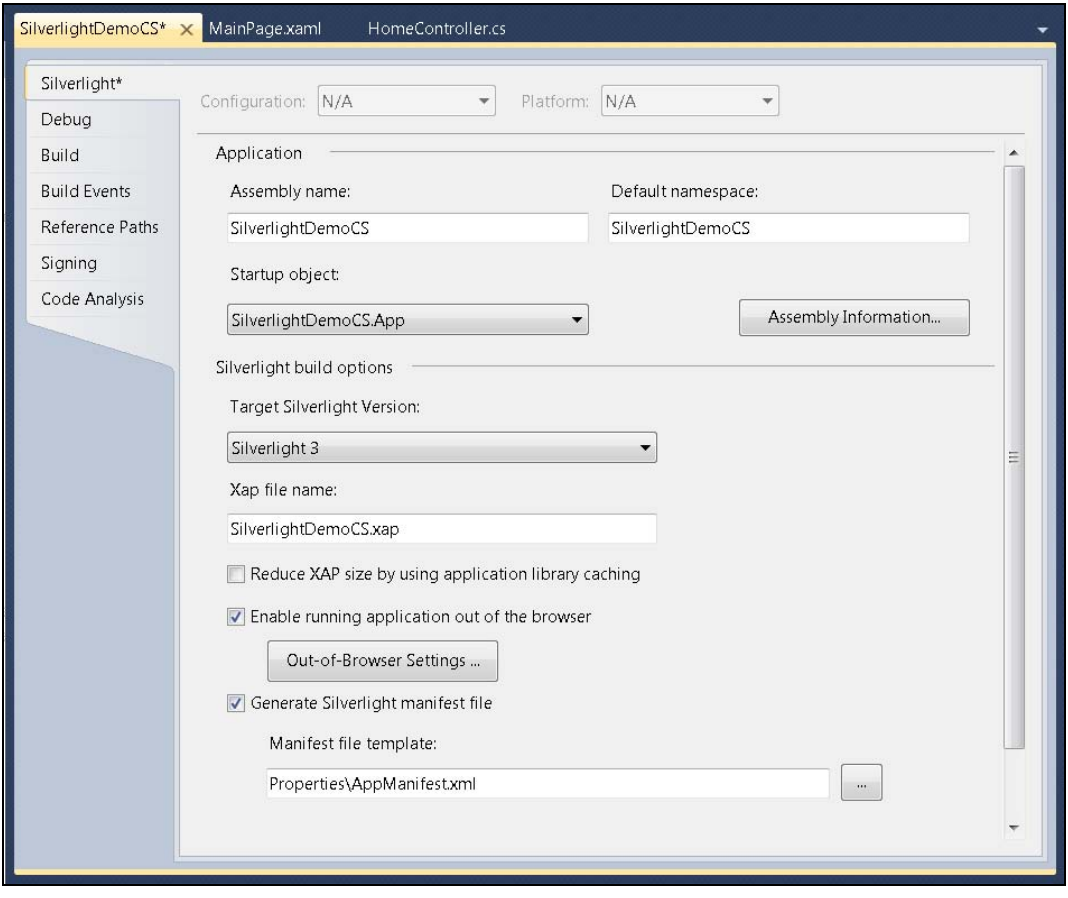


Рис. 10.5. Окно свойств Silverlight

Большинство свойств, отображаемых в окне, показанном на рис. 10.5, уже рассматривались в предыдущих главах. Основное отличие заключается в опциях построения — раздел **Silverlight build options** позволяет указывать номер версии и установить флажок, дающий возможность уменьшить размер *.xap-файла за счет кэширования. Однако если вам требуется использовать OOB, оставьте этот флажок в невзведенном состоянии, потому что данная опция несовместима с OOB. Файл манифеста (Manifest) описывает содержимое *.xap-файла. Чтобы активизировать OOB, установите опцию **Enable running application out of the browser**. Затем нажмите кнопку **Out-Of-Browser Settings**, чтобы отобразить на экране окно, показанное на рис. 10.6.

Настройки OOB в окне, показанном на рис. 10.6, позволяют задать различные свойства приложения, размер, который оно будет иметь во время исполнения, а также набор значков различных размеров, предназначенных для отображения в ОС Windows. Установка опции **GPU acceleration** позволяет приложению воспользоваться преимуществами аппаратной конфигурации локального компьютера, чтобы помочь оптимизировать графику.

После сохранения настроек OOB можно запустить приложение. Пользователь может выполнить щелчок правой кнопкой мыши по приложению, работающему в браузере, и выбрать опцию **Install SilverlightDemoCSApplication onto this computer**, как показано на рис. 10.7.

В следующем окне вам будут доступны опции для добавления приложения в меню **Start**, а также добавления значка, представляющего ваше приложение, на рабочий стол. На рис. 10.8 выбраны обе эти опции.

Когда вы нажмете кнопку **OK**, Silverlight создаст соответствующий пункт меню **Start** и добавит значок приложения на рабочий стол, как показано на рис. 10.9. Когда вы запустите приложение, оно будет работать в окне, а не в браузере.

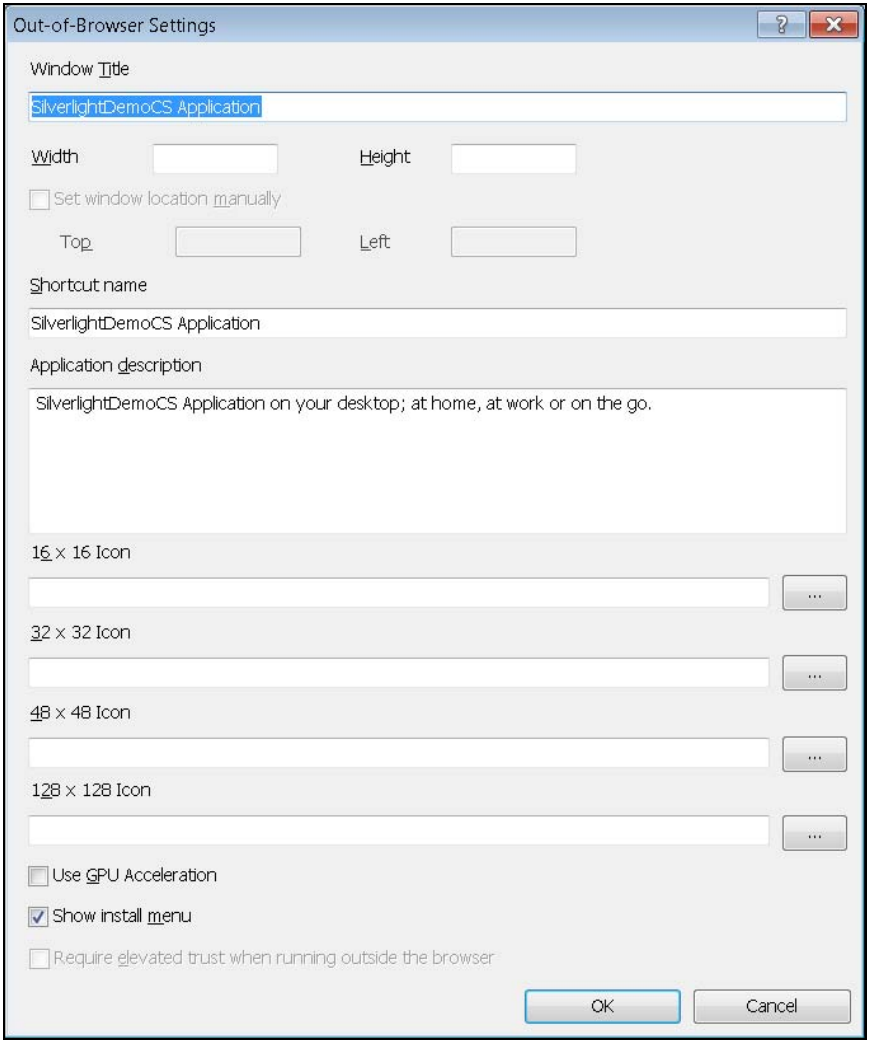


Рис. 10.6. Окно **Out-of-browser settings**

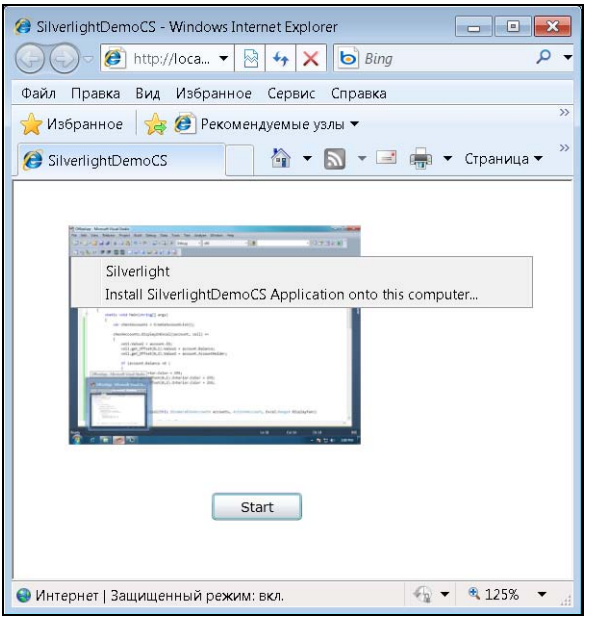


Рис. 10.7. Выбор OOB

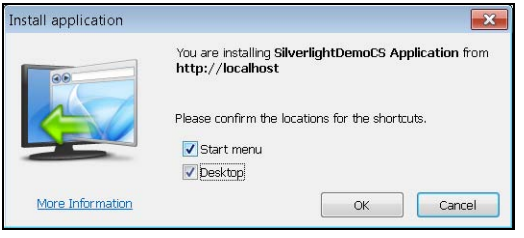


Рис. 10.8. Выбор опций развертывания OOB

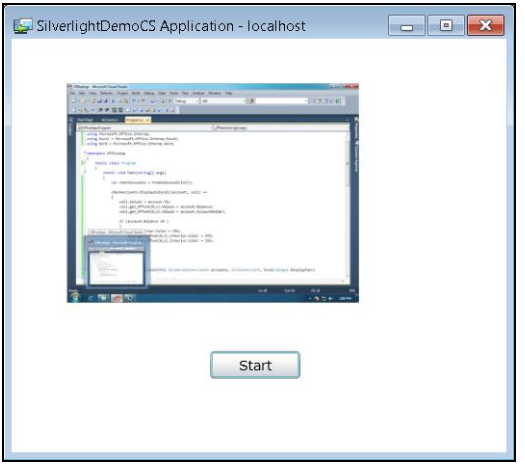


Рис. 10.9. Исполнение приложения OOB

Развертывание приложений Silverlight

Вы можете распространить приложение Silverlight на Web-сайт точно так же, как это делалось для приложений ASP.NET MVC. Однако вам потребуется убедиться в том, что установлены тип MIME и политика, гарантирующие работоспособность вашего приложения вне вашей среды разработчика.

Если вы работаете с IIS 7, Silverlight в вашей системе уже присутствует. Однако если вы выполняете развертывание на IIS 6 Server, вам потребуется установить тип MIME для *.xap-файлов на значение **application/x-silverlight-app**. Чтобы сделать это, выполните следующие операции:

Раскройте **Administrative Tools | Internet Information Services (IIS) Manager**.

1. В узле **Web Sites** разверните узел **IIS**, щелкните правой кнопкой мыши, наведя курсор на Web-сайт, где вы хотите развернуть ваше приложение Silverlight, и из контекстного меню выберите команду **Properties**.
2. Перейдите на вкладку **HTTP Headers**, выберите опцию **MIME Types**, затем выберите опцию **New**.
3. Введите аббревиатуру **.xap** в поле **Extension**, а в качестве типа MIME выберите опцию **application/x-silverlight-app**.
4. Три раза нажмите кнопку **OK**, чтобы закрыть все окна и выйти из IIS.
5. В качестве дополнения вы можете создать в корневой папке вашего Web-сайта файл политики (policy file). Вы можете использовать два типа файлов политики: crossdomain.xml или clientaccesspolicy.xml.

Файл политики crossdomain.xml создается для приложений Adobe Flash и может использоваться, в том числе, и с приложениями Silverlight. Рассмотрим простой пример, приведенный в листинге 10.4.

Листинг 10.4. Пример использования файла политики crossdomain.xml с приложениями Silverlight

```
<!DOCTYPE cross-domain-policy
  SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <allow-access-from domain="*" />
  <allow-http-request-headers-from domain="*" headers="*" />
</cross-domain-policy>
```

Разработчики в Microsoft хорошо понимают, что для разработки приложений Silverlight файл crossdomain.xml недостаточно гибок, поэтому они добавили еще один файл, обеспечивающий дополнительную поддержку: clientaccesspolicy.xml. Пример использования этого файла с приложениями Silverlight приведен в листинге 10.5.

Листинг 10.5. Использование файла clientaccesspolicy.xml с приложениями Silverlight

```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-methods="*">"
        <domain uri="*"/>
      </allow-from>
      <grant-to>
        <resource path="/" include-subpaths="true"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

Эта выдержка из файла clientaccesspolicy.xml показывает, что все домены имеют право доступа ко всему содержимому сайта, которое не защищено другими средствами. Вы можете ограничить доступ, заменив `*` в универсальном идентификаторе ресурса (`uri`) домена именем допустимого домена. Далее, вы можете изменить пути к ресурсам именем пути к сайту, чтобы ограничить доступ к отдельным папкам. Чтобы добавлять новые домены и пути, добавляйте дополнительные элементы политики к этому файлу.

Заключение

В этой главе объяснялось, как запустить приложение Silverlight. После ее прочтения вы должны уметь пользоваться элементом управления **MediaElement** и создавать пользовательские интерфейсы, пользуясь теми же техническими приемами, что и при создании приложений WPF. Функциональные возможности OOB позволяют запускать приложения Silverlight с рабочего стола. Наконец, последний раздел описывает развертывание приложений Silverlight на Web-сервер.

Итак, мы рассмотрели две технологии Web — ASP.NET MVC в *главе 9* и Silverlight — в данной главе. В следующей главе будет описана еще одна Web-технология: WCF Web Services.

Глава 11

Развертывание Web-сервисов с помощью WCF

В настоящей главе будут рассмотрены следующие ключевые концепции:

- Создание Web-сервисов;
- Развертывание Web-сервисов;
- Написание клиентских приложений, пользующихся Web-сервисами.

Windows Communication Foundation (WCF) представляет собой технологию .NET, предназначенную для создания сервисов Web. Web-сервис — это программное обеспечение, предоставляющее функциональные возможности, которыми может пользоваться любое другое приложение, написанное на любом языке программирования, работающее на любой аппаратной платформе, под управлением любой операционной системы. Единственное, что для этого требуется — это возможность взаимодействия через сеть.

Функциональные возможности Web-сервисов могут быть либо публичными (public), т. е. общедоступными, либо частными (private), т. е. доступными ограниченному кругу лиц. В качестве примеров публичных Web-сервисов можно привести те сервисы, которые по запросу предоставляют пользователям информацию о погоде. Вы указываете географическое местоположение и получаете прогноз погоды, выведенный на экран. Еще один пример такого сервиса — приложение, осуществляющее верификацию адреса. В этом случае вы указываете почтовый адрес, а приложение проверяет его на существование и, возможно, предлагает альтернативы. Примерами частных Web-сервисов могут быть приложения, предоставляющие возможность множеству приложений сотрудников крупной корпорации обращаться к Web-серверу клиентского отдела и по идентификатору клиента (customer ID), переданному в запросе, получать информацию об этом клиенте. Еще один вариант — обращение к Web-сервису отдела заказов, которому вы можете передать информацию о новом заказе клиента, чтобы Web-сервис в фоновом режиме провел для вас обработку этого заказа.

Общее между примерами, которые были только что описаны, заключается в том, что вне зависимости от того, является ли Web-сервис публичным или частным, он может обслуживать множества приложений или систем.

Каждому, кто обращается к Web-сервису, нужны одни и те же услуги, поэтому зачем пытаться изобретать велосипед при написании каждого нового приложения? Достаточно просто один раз создать и настроить сервис, с которым смогут работать все.

Вероятно, вас удивит, как же возможно добиться того, чтобы одна и та же технология была доступна для любой системы, независимо от платформы, языка или программного обеспечения. Web-сервис абстрагируется от вызывающей системы с помощью открытых стандартов и четко определенного интерфейса. Существуют универсальные коммуникационные протоколы, такие как HTTP, и стандарты на форматы данных, таких как XML, которые Web-сервисы могут использовать. Таким образом, если обе системы могут поддерживать коммуникации с помощью протокола передачи гипертекста (Hypertext Transfer Protocol, HTTP) и работать с форматом XML, тогда Web-сервис может быть полезным. Например, если приложение построено для рабочей станции Sun, работающей под управлением Solaris, сохраняет информацию в базе данных Oracle и написано на Java, оно все равно сможет взаимодействовать с вашим WCF-сервисом, даже если он работает на аппаратной платформе Intel под управлением Windows Server 2008, хранит информацию в базе данных SQL Server и написан на VB. Эта разница в аппаратных платформах, операционных системах не имеет значения, потому что Java переведет свой запрос в формат XML и перешлет данные XML сервису WCF по протоколу HTTP. Сервис WCF может интерпретировать формат XML, потому что именно WCF-сервис сообщил приложению, написанному на Java, в какой формат следует преобразовать код XML. Для интересующихся отмечу, что этот формат называется языком описания Web-сервиса (Web Service Description Language, WSDL) и представляет собой соглашение (или интерфейс), сообщающее вызывающему коду, как следует паковать XML и какие операции (например, GetWeather) можно вызывать через Web-сервис. WSDL незаметно для пользователей выполняет еще множество других операций, но основная его цель заключается в том, чтобы гарантировать, что клиенты, например, такие, как приложения на Java, использовали WSDL для упаковки своего кода XML и отправляли его сервису WCF. Сервис транслирует код XML в вызов метода, исполняет метод, затем снова пакует результаты в XML (так, как было указано правилами WSDL), а затем отсылает результаты обратно приложению Java. Сервис WCF использует открытые стандарты так, что любые другие системы тоже могут использовать те же самые стандарты для поддержания коммуникаций.

В данной главе будет показано, как создавать сервисы WCF с помощью VS, как создать клиентское приложение для взаимодействия с WCF-сервисом и как выполнять развертывание WCF-сервисов. Информация о развертывании, приведенная в этой главе, будет полезна и тем, кто собирается выполнять развертывание не только WCF-сервисов, но и других типов Web-приложений, в том числе ASP.NET MVC и Silverlight. Начнем, как всегда, с создания нового проекта WCF.

Создание нового проекта WCF

Чтобы создать новый проект WCF, нажмите клавиатурную комбинацию <CTRL>+<SHIFT>+<N> (запуск нового проекта) и в раскрывшемся окне выберите

опцию **WCF Service Library**. Присвойте новому проекту имя **WcfDemo** и укажите путь к новому проекту. В результате новый проект будет выглядеть так, как показано на рис. 11.1.

Проект **WCF Service Library** начинается двумя файлами со стандартными именами: IService1.cs (IService1.vb — для VB) и Service1.cs (Service1.vb — для VB), которые содержат интерфейс и класс, который реализует этот интерфейс. Если вам требуется освежить свои знания об интерфейсах, перечитайте *главу 4* и не забывайте, что интерфейс является неотъемлемой частью разработки под WCF.

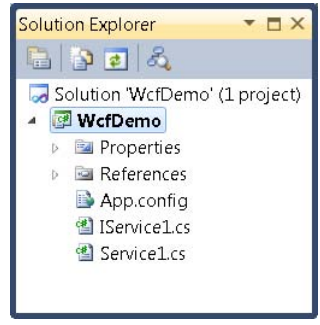


Рис. 11.1. Проект **WCF Service Library**

Указание соглашения (Contract) с WCF-интерфейсами

Класс IService1.cs (IService1.vb — для VB), показанный на рис. 11.1, содержит интерфейс.

Как вы уже знаете из *главы 4*, интерфейсы определяют наборы членов, которые не имеют реализаций. Фактическая реализация обеспечивается классами, которые реализуют указанные интерфейсы. Вы можете рассматривать интерфейсы как соглашения, которые гарантируют для сервиса определенные наборы операций. В дополнение к интерфейсам, частью соглашения для сервиса являются типы, ассоциированные с ним. Соглашение (contract) играет важную роль, потому что при написании кода, который использует Web-сервис, вашему коду будет видимо именно соглашение, а все, что выходит за его рамки, будет для вашего кода недоступно и "невидимо". Любое приложение, которому требуется обращаться к Web-сервису, будет осуществлять к нему вызовы, основываясь на том, что присутствует в соглашении (contract). В данном разделе будет показано, как определить соглашение WCF с помощью интерфейса. Кроме того, речь пойдет о встроенных типах и индивидуально заданных типах. В последующих разделах будет показано, как определить соглашение, реализовать его и как им пользоваться. Причем важность соглашения (contract) будет подчеркиваться на протяжении всего жизненного цикла приложения, чтобы продемонстрировать вам, как соглашение определяется, реализуется и используется.

Изучение соглашения, сгенерированного VS

Вряд ли вам захочется работать с интерфейсом, который называется `IService1`; такое название и в самом деле ровным счетом ничего не значит. Таким образом, лучше переименовать IService1.cs в ICustomerService.cs (IService1.vb в ICustomerService.vb — для проектов на VB), потому что конфигурировать мы этот сервис будем таким образом, чтобы он управлял записями о клиентах. Вы увидите сообщение о переимено-

вании кода, и в ответ на запрос, отображенный в этом окне, вам надо ответить утвердительно. Когда вы откроете файл ICustomerService.cs, вы увидите тот же самый код, что и в листингах 11.1 и 11.2. Этот код содержит интерфейс и атрибуты для соглашения `ICustomerService`.

Листинг 11.1. Интерфейс сервиса WCF (код на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfDemo
{
    // NOTE: You can use the "Rename" command on the
    // "Refactor" menu to change the interface name
    // "IService1" in both code and config file together.
    [ServiceContract]
    public interface ICustomerService
    {
        [OperationContract]
        string GetData(int value);

        [OperationContract]
        CompositeType GetDataUsingDataContract
             (CompositeType composite);

        // TODO: Add your service operations here
    }

    // Use a data contract as illustrated in the sample
    // below to add composite types to service operations
    [DataContract]
    public class CompositeType
    {
        bool boolValue = true;
        string stringValue = "Hello ";

        [DataMember]
        public bool BoolValue
        {
            get { return boolValue; }
```

```
            set { boolValue = value; }
        }

        [DataMember]
        public string StringValue
        {
            get { return stringValue; }
            set { stringValue = value; }
        }
    }
}
```

Листинг 11.2. Интерфейс сервиса WCF (код на VB)

```
' NOTE: You can use the "Rename" command on the
' context menu to change the interface name "IService1"
' in both code and config file together.
<ServiceContract()>
Public Interface ICustomerService

    <OperationContract()>
    Function GetData(ByVal value As Integer) As String

    <OperationContract()>
    Function GetDataUsingDataContract(
          ByVal composite As CompositeType) As CompositeType

    ' TODO: Add your service operations here

End Interface

' Use a data contract as illustrated in the sample below
' to add composite types to service operations

<DataContract()>
Public Class CompositeType

    <DataMember()>
    Public Property BoolValue() As Boolean

    <DataMember()>
    Public Property StringValue() As String

End Class
```

В листингах 11.1 и 11.2 имеются два типа: `ICustomerService` и `CompositeType`. Оба этих типа были сгенерированы VS, чтобы предоставить пример того, как вы можете определить пример соглашения для сервиса. После разбора стандартного автоматически сгенерированного кода, нам необходимо модифицировать этот код, чтобы его можно было использовать с объектами `Customer`.

Начиная с интерфейса `ICustomerService`, две наиболее важные части кода — это атрибуты `ServiceContract` и `OperationContract`. Атрибут `ServiceContract` утверждает, что данный интерфейс определяет соглашение для Web-сервиса WCF. Без атрибута `ServiceContract` WCF просто не сможет распознать данный интерфейс. Атрибут `OperationContract` указывает методы, предоставляемые сервисом WCF. Без атрибута `OperationContract` метод не будет видимым как часть сервиса WCF.

Метод сервиса WCF может работать с любыми встроенными типами для параметров и возвращаемых типов, что демонстрируется методом `GetData`, который принимает параметр типа `int` и возвращает строковое значение. Когда вы работаете с индивидуально определяемыми типами, вам требуется дополнительный синтаксис, чтобы указать, какие части этого типа являются частью соглашения (contract). Типы представляют собой параметры и возвращаемые типы методов сервиса и, в дополнение к интерфейсу, тоже являются частью соглашения (contract).

Метод `GetDataUsingDataContract` представляет собой пример метода, использующего индивидуально определяемый составной тип (`CompositeType`), в качестве параметра, и возвращаемый тип. Будучи индивидуально определенным составным типом, `CompositeType` имеет атрибуты, помогающие определить соглашение (contract): `DataContract` и `DataMember`. Атрибут `DataContract` идентифицирует `CompositeType` как тип, который может быть включен в соглашение сервиса WCF. Без атрибута `DataContract` тип не может быть включен в соглашение сервиса. Атрибут `DataMember` выделяет члены типа, которые являются частью соглашения для данного сервиса. Без атрибута `DataMember` член типа не будет видимым как часть соглашения (contract).

Создание собственных соглашений

Мы не будем явно конструировать собственные типы данных для `DataContracts`, как было показано для `CompositeType` в листингах 11.1 и 11.2. Вместо этого мы воспользуемся встроенной возможностью LINQ to SQL, которая дает сущностям LINQ to SQL атрибут `DataContract`. Чтобы использовать сущности LINQ to SQL, создайте новый элемент **LINQ to SQL** в том же проекте, что и сервис WCF, и добавьте таблицу **Customer** в окно визуального конструктора (designer). Затем выполните щелчок мышью по поверхности визуального конструктора (не по сущности **Customer**) и просмотрите свойства. Установите для параметра **Serialization Mode** значение **Unidirectional**, как показано на рис. 11.2.

Теперь, вместо создания индивидуально определенного типа и копирования данных сущности LINQ to SQL в индивидуальный тип с возвратом индивидуального типа, можно просто выполнить запрос LINQ to SQL и вернуть объект-сущность LINQ to SQL.

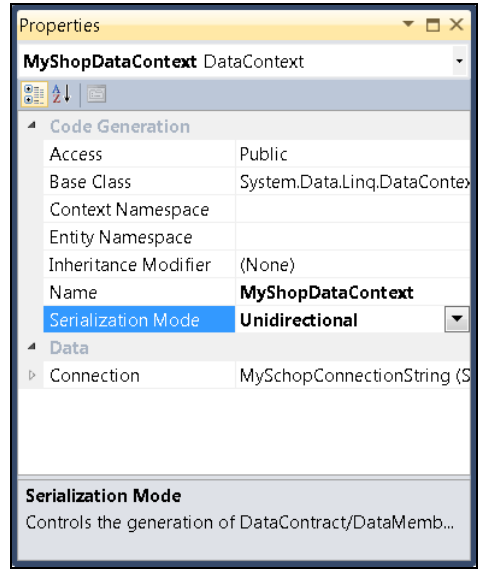


Рис. 11.2. Настройка свойства **Serialization Mode** для LINQ to SQL

Мы начали с индивидуальной настройки соглашения (contract) при изменении имени `IService1` на `ICustomerService`, но здесь требуется продолжение. Нужно определить методы, которые станут частью соглашения `CustomerService`: `GetCustomers`, `GetCustomer`, `InsertCustomer`, `UpdateCustomer` и, наконец, `DeleteCustomer`. На практике методов будет гораздо больше, чем вам требуется, например, для индивидуальной настройки соглашения с сервисом под нужды вашего приложения, но рассматриваемые здесь методы типичны и охватывают наиболее широко распространенные сценарии, с которыми вы столкнетесь в повседневной практике, и представляющие наиболее типичные образцы работы, которую вам придется выполнять. В листингах 11.3 и 11.4 показаны модификации, которые вам потребуется внести в интерфейс `ICustomerService`, чтобы поддерживать наиболее типичные операции по работе с клиентами. После внесения в ваш код изменений, показанных в листингах 11.3 и 11.4, ваше приложение будет выдавать ошибки компиляции до тех пор, пока вы не реализуете интерфейс `ICustomerService` в соответствии с инструкциями, приведенными в следующем разделе. Поэтому внесите предложенные изменения и продолжайте читать до следующего раздела.

Листинг 11.3. Реализация соглашения сервиса WCF (код на C#)

```
[ServiceContract]
public interface ICustomerService
{
        [OperationContract]
        Customer GetCustomer(int custID);

        [OperationContract]
        List<Customer> GetCustomers();

        [OperationContract]
        int InsertCustomer(Customer cust);

        [OperationContract]
```

```
        void UpdateCustomer(Customer cust);

        [OperationContract]
        void DeleteCustomer(int custID);
}
```

Листинг 11.4. Реализация соглашения сервиса WCF (код на VB)

```
<ServiceContract()>
Public Interface ICustomerService

        <OperationContract()>
        Function GetCustomer(ByVal custID As Integer) As Customer

        <OperationContract()>
        Function GetCustomers() As List(Of Customer)

        <OperationContract()>
        Function InsertCustomer(ByVal cust As Customer) As Integer

        <OperationContract()>
        Sub UpdateCustomer(ByVal cust As Customer)

End Interface
```

Вы уже знаете, как задать интерфейс, а в предыдущем разделе объяснялись цели атрибутов `ServiceContract` и `OperationContract`. В листингах 11.3 и 11.4 показано, что все, что вам необходимо сделать — это указать методы, которые вы хотите включить в соглашение как составную часть.

Иногда бывают ситуации, когда вам требуется вернуть индивидуально определенный тип из сервиса WCF. Например, если вам требуется заполнить раскрывающийся список, то все, что вам требуется — это ключ для значения и имя — для текста. Поэтому вы можете создать индивидуальный класс `CustomerLookup`, как показано в листингах 11.5 и 11.6. Этот класс задает атрибуты `DataContract` и `DataMember`. Код, представленный в листингах 11.5 и 11.6, демонстрирует, как запрограммировать индивидуально определенный тип, если вам требуется решить эту задачу.

Листинг 11.5. Индивидуально определенный тип для соглашения сервиса WCF (код на C#)

```
[DataContract]
public class CustomerLookup
{
        [DataMember]
```

```
        public int CustomerID { get; set; }

        [DataMember]
        public string CustomerName { get; set; }
}
```

Листинг 11.6. Индивидуально определенный тип для соглашения сервиса WCF (код на VB)

```
<DataContract()>
Public Class CustomerLookup

        <DataMember()>
        Public Property CustomerID() As Integer

        <DataMember()>
        Public Property CustomerName() As String

End Class
```

Использование индивидуально определенных типов для целей создания элементов управления, предназначенных для просмотра и поиска на уровне пользовательского интерфейса, например, таких, как класс `CustomerLookup` в листингах 11.5 и 11.6, открывает возможность обмениваться только необходимой информацией, а не целыми объектами, часть данных из состава которых использоваться не будет. Учитывая возможность работы через медленные сетевые соединения, ограничение объема информации, которой ваше приложение обменивается с Web-сервисом, может повысить производительность вашего приложения.

Теперь, когда соглашение (contract) создано, необходимо написать класс, который будет его реализовать.

Реализация логики с помощью классов WCF

Соглашение, созданное в предыдущем разделе, играет важную роль, потому что оно указывает, что именно требуется реализовать. Как вы уже знаете, интерфейсы только указывают члены, из которых состоит соглашение, но при этом вам требуется написать класс, содержащий код, который реализует соглашение. В этом разделе будет продемонстрирована реализация интерфейса `ICustomerService` с помощью класса `CustomerService`.

Первое, что вам потребуется сделать — это переименовать файл Service1.cs (Service1.vb — в VB), присвоив ему новое имя CustomerService.cs (CustomerService.vb — для VB), и щелкнуть мышью по кнопке **Yes**, когда VS запросит вашего согласия на изменение кода. В листингах 11.7 и 11.8 показан автоматически сгенерированный VS-код класса сервиса WCF.

Листинг 11.7. Автоматически сгенерированный код для класса, реализующего сервис WCF (код на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfDemoCS
{
    // NOTE: You can use the "Rename" command on the
    // "Refactor" menu to change the class name "Service1"
    // in both code and config file together.
    public class CustomerService : ICustomerService
    {
            public string GetData(int value)
           {
                   return string.Format("You entered: {0}", value);
           }

           public CompositeType GetDataUsingDataContract(
                     CompositeType composite)
           {
                     if (composite == null)
                     {
                               throw new ArgumentNullException("composite");
                     }
                     if (composite.BoolValue)
                     {
                               composite.StringValue += "Suffix";
                     }
                     return composite;
           }
    }
}
```

Листинг 11.8. Автоматически сгенерированный код для класса, реализующего сервис WCF (код на VB)

```
' NOTE: You can use the "Rename" command on the
' context menu to change the class name "Service1"
' in both code and config file together.
Public Class Service1
```

```
    Implements ICustomerService

    Public Function GetData(
        ByVal value As Integer) As String
        Implements ICustomerService.GetData
        Return String.Format("You entered: {0}", value)
    End Function

    Public Function GetDataUsingDataContract(
        ByVal composite As CompositeType) As CompositeType
        Implements ICustomerService.GetDataUsingDataContract
        If composite Is Nothing Then
                Throw New ArgumentNullException("composite")
        End If
        If composite.BoolValue Then
                composite.StringValue &= "Suffix"
        End If
        Return composite
    End Function

End Class
```

Методы класса `CustomerService` в листингах 11.7 и 11.8 демонстрируют реализации "скелета" интерфейса `ICustomerService`. Как вы помните, листинги 11.3 и 11.4 предоставляли новые методы для интерфейса `ICustomerService`, так что код, представленный в листингах 11.7 и 11.8, выдавал ошибки компиляции, потому что он не реализует методы интерфейса `ICustomerService`. Чтобы решить эту проблему, удалите методы `GetData` и `GetDataUsingDataContract` из класса `CustomerService`. Затем выделите идентификатор `ICustomerService` в файле CustomerService.cs, в результате чего слева от идентификатора `ICustomerService` появится символ подчеркивания. Задержите курсор над этим подчеркиванием, чтобы открыть меню, предлагающее возможности реализации интерфейса `ICustomerService`. Выберите из этого меню опцию **Implement Interface 'ICustomerService'**, в результате чего в составе класса `CustomerService` будет автоматически сгенерирован код, представляющий собой "скелет" для каждого из членов интерфейса `ICustomerService`. Стандартные реализации выбрасывают исключение `NotImplementedException`, и это означает, что вам необходимо реализовать код этих методов, основываясь на интерфейсе `ICustomerService`. В листингах 11.9 и 11.10 показаны реализации интерфейса `ICustomerService` в классе `CustomerService`. Если вы используете C#, добавьте код для каждого метода. Если вы используете VB, где не предлагается такой поддержки рефакторинга интерфейсов, как в C#, добавьте все методы в класс `CustomerService` вручную, в точности, как указано в листинге 11.10.

Листинг 11.9. Реализация сервиса WCF на C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfDemoCS
{
        public class CustomerService : ICustomerService
        {
            public Customer GetCustomer(int custID)
            {
                var ctx = new MyShopDataContext();

                var customer =
                    (from cust in ctx.Customers
                     where cust.CustomerID == custID
                     select cust)
                     .SingleOrDefault();

                return customer;
            }

            public List<Customer> GetCustomers()
            {
                var ctx = new MyShopDataContext();

                return
                    (from cust in ctx.Customers
                     select cust)
                     .ToList();
            }

            public int InsertCustomer(Customer cust)
            {
                var ctx = new MyShopDataContext();

                ctx.Customers.InsertOnSubmit(cust);

                ctx.SubmitChanges();

                return cust.CustomerID;
            }

            public void UpdateCustomer(Customer cust)
```

```
        {
            var ctx = new MyShopDataContext();

            var customer =
                (from cst in ctx.Customers
                 where cst.CustomerID == cust.CustomerID
                 select cst)
                 .SingleOrDefault();

            if (customer != null)
            {
                 customer.Age = cust.Age;
                 customer.Birthday = cust.Birthday;
                 customer.Income = cust.Income;
                 customer.Name = cust.Name;

                 ctx.SubmitChanges();

            }
        }

        public void DeleteCustomer(int custID)
        {
            var ctx = new MyShopDataContext();

            var customer =
                (from cst in ctx.Customers
                 where cst.CustomerID == custID
                 select cst)
                 .SingleOrDefault();

            if (customer != null)
            {
                ctx.Customers.DeleteOnSubmit(customer);

                ctx.SubmitChanges();
            }
        }
    }
}
```

Листинг 11.10. Реализация сервиса WCF на VB

```
' NOTE: You can use the "Rename" command on the context
' menu to change the class name "Service1" in both code
' and config file together.
Public Class CustomerService
    Implements ICustomerService

    Public Function GetCustomer(ByVal custID As Integer) As Customer
```

```
    Implements ICustomerService.GetCustomer

    Dim ctx As New MyShopDataContext

    Dim customer =
        (From cust In ctx.Customers
         Where cust.CustomerID = custID
         Select cust).SingleOrDefault()

    Return customer

End Function

Public Function GetCustomers() As List(Of Customer)
    Implements ICustomerService.GetCustomers

    Dim ctx As New MyShopDataContext

    Return (From cust In ctx.Customers
            Select cust).ToList()

End Function

Public Sub UpdateCustomer(ByVal cust As Customer)
    Implements ICustomerService.UpdateCustomer

    Dim ctx As New MyShopDataContext

    Dim customer = (From cst In ctx.Customers
                    Where cst.CustomerID = cust.CustomerID
                    Select cst).SingleOrDefault()

    If Not (customer Is Nothing) Then

       With customer
            .Age = cust.Age
            .Birthday = cust.Birthday
            .Income = cust.Income
            .Name = cust.Name
      End With

      ctx.SubmitChanges()

    End If

End Sub

Public Sub DeleteCustomer(ByVal custID As Integer)
```

```
        Dim ctx As New MyShopDataContext

        Dim customer = (From cst In ctx.Customers
                        Where cst.CustomerID = custID
                        Select cst).SingleOrDefault()

        If Not (customer Is Nothing) Then

            ctx.Customers.DeleteOnSubmit(customer)

            ctx.SubmitChanges()
        End If

    End Sub
End Class
```

Реализация класса `CustomerService` аналогична тому, что вы уже видели в предыдущих главах. Разница заключается в том, что в данном случае реализация представляет собой Web-сервис, который должен использоваться иначе. Вскоре мы перейдем к разделу, в котором вопросы использования Web-сервиса будут рассматриваться отдельно, но вам необходимо понимать, что Web-сервис представляет собой компонент, взаимодействие с которым необходимо поддерживать через сеть. В предшествующих главах вы уже видели код, который работает с данными, интегрированными в код приложения. Однако, Web-сервисам необходим хостинг на сервере, например, такой хостинг может предоставлять Internet Information Services (IIS), а код, потребляющий услуги Web-сервиса, должен подключаться и поддерживать коммуникации через IIS. В следующем разделе будет подробно рассказано о хостинге, который IIS предоставляет Web-сервисам.

Хостинг для сервиса WCF

Среда разработки VS автоматически предоставляет хостинг вашему сервису, но, в конечном итого, вам понадобится выполнить развертывание вашего сервиса на Internet Information Services (IIS) — Web-сервер, который предоставляет хостинг приложениям .NET. Инструкции, приведенные в данном разделе, представляют собой общее руководство, поясняющее в общих чертах, как происходит процесс развертывания. Вполне возможно, что в вашем случае, вследствие различия в операционных системах и сервисных пакетах (service packs) у вас этот процесс в некоторых деталях будет отличаться. Кроме того, возможно, что на вашем компьютере окажется другое сочетание операционной системы и версии IIS. Кроме того, поведение различных версий операционных систем (например, Windows Server 2003 и Windows Server 2008) может различаться в деталях, причем эти различия, какими бы незначительными они ни казались на первый взгляд, могут оказаться весьма существенными. Все эти проблемы не относятся к VS как к продукту, в таких ситуациях вам необходимо будет внимательно прочесть документацию по вашей

операционной системе, чтобы надлежащим образом сконфигурировать IIS и параметры безопасности операционной системы. Хотя поведение операционной системы не является функцией VS, в данном разделе будет приведено небольшое руководство, которое поможет вам правильно сориентироваться при решении такого рода проблем.

Общие процедуры настройки хостинга

В целях обеспечения высокого уровня безопасности, в стандартной конфигурации ОС Windows, предлагаемой по умолчанию, IIS не устанавливается. Существуют различные версии Windows, предназначенные для рабочих станций и серверов, поэтому я приведу лишь общее описание процедуры установки IIS. В качестве первого шага вам необходимо открыть окно **Control Panel** в вашей операционной системе Windows. Старые версии Windows в этом окне предоставляли ссылку **Add And Remove Programs**, а в новейших версиях эта опция называется **Programs And Features**. В качестве следующего шага вам необходимо воспользоваться опцией для установки и удаления компонентов Windows. Найдите в списке опцию для установки IIS и запустите процедуру установки. Не забудьте включить опцию поддержки FTP (File Transfer Protocol), если вы планируете осуществлять развертывание с помощью FTP. FTP — это протокол Internet, который позволяет работать с файлами. Он весьма полезен для развертывания приложений, поскольку с его помощью очень удобно перемещать файлы между серверами. На более новых версиях Windows вам потребуется активизировать поддержку ASP.NET. Как это делается, будет показано в следующем разделе.

Как только вы установите IIS, вы сможете предоставить хостинг своему приложению. В версиях Windows, предназначенных для рабочих станций, IIS 6 поддерживает только один Web-сайт. На серверных платформах или при условии использования IIS 7 и более новых версий можно поддерживать множество Web-сайтов. Чтобы создать Web-сайт, вам потребуется либо создать виртуальный каталог (в случае с настольными версиями IIS 6) или Web-приложение. Вы можете сделать это, открыв IIS, через меню **Администрирование** (Administrative Tools). Как правило, соответствующую опцию можно найти и в Панели управления (Control Panel). Найдите опцию **Web Sites**, щелкните по ней правой кнопкой мыши и из контекстного меню выберите команду **Create New Web Application**. Если вы пользуетесь IIS 6 на настольном компьютере, вам потребуется спуститься на уровень ниже, выполнить щелчок правой кнопкой мыши по опции **Default Web Site** и выбрать из контекстного меню команду **Create Virtual Directory**. Выполняя пошаговые инструкции программы-мастера, не изменяйте значений, предложенных по умолчанию. Единственное, что вам потребуется сделать — это указать путь к виртуальному каталогу или имя сайта и физический путь. Имя виртуального каталога/сайта представляет собой путь, который пользователь должен вводить в поле адреса. Физический путь представляет собой папку в вашей файловой системе, где будет размещаться приложение. По умолчанию предлагается значение `c:\inetpub`, в предположении того, что сама операционная система установлена на диск `C:`.

Установка IIS 7 в Windows 7

Чтобы установить IIS 7 в Windows 7, проделайте следующие шаги:

1. Выберите из меню команды **Пуск** (Start) | **Панель управления** (Control Panel) | **Программы** (Programs) | **Программы и компоненты** (Programs And Features). На экране появится окно Удаление или изменение программы (Uninstall Or Change A Program), показанное на рис. 11.3.

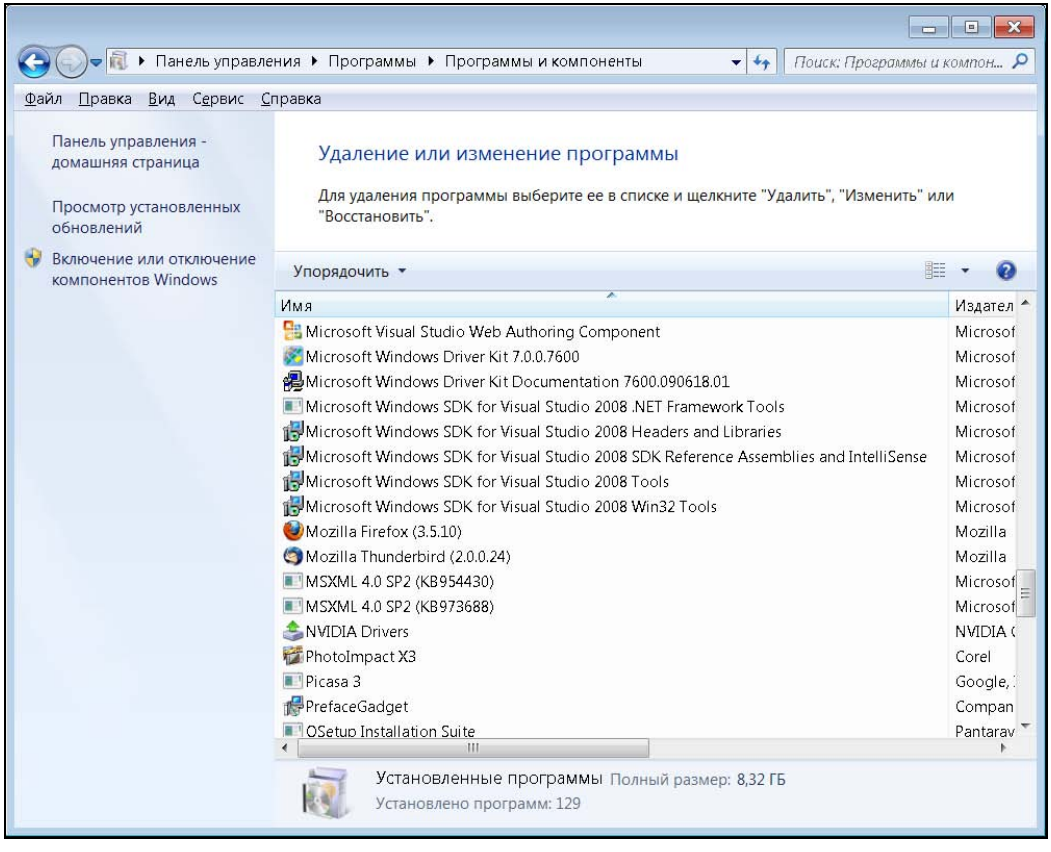


Рис. 11.3. Окно установки и удаления программ в Windows 7

2. Щелкните мышью по ссылке **Включение или отключение компонентов Windows** (Turn Windows features on or off). Подтвердите запрос UAC, после чего на экране появится окно **Компоненты Windows** (Windows Features), показанное на рис. 11.4.
3. Найдите в списке компонентов Windows опцию **Службы IIS** (Internet Information Services) и выберите опции для установки IIS. В рассматриваемом примере активизирован протокол FTP, который предоставляет один из вариантов, которые могут быть использованы для развертывания Web-сайта. Убеди-

тесь в том, что в ветви **Совместимость управления IIS 6** (IIS 6 Management Compatibility) установлен флажок **Совместимость конфигурации метабазы IIS и IIS6** (Ensure IIS Metabase and IIS 6 configuration compatibility). Нажмите кнопку **OK**, и сервер IIS будет установлен.

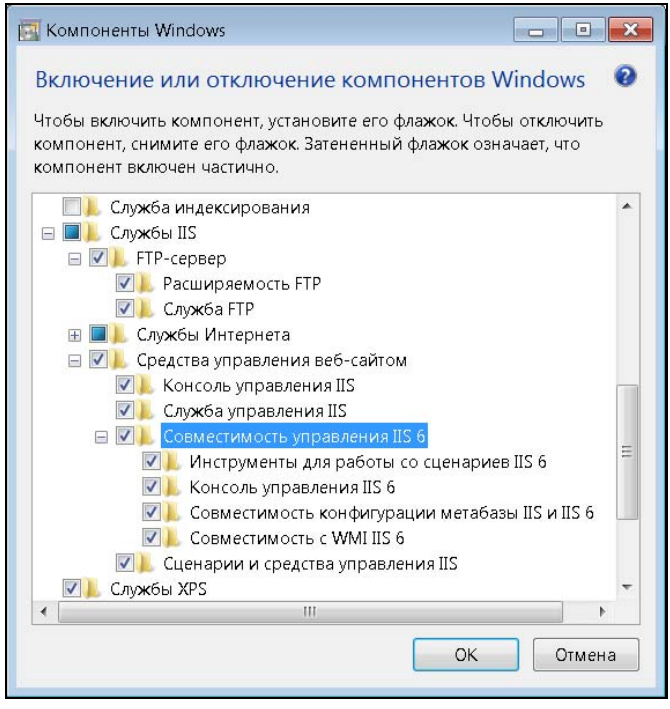


Рис. 11.4. Окно **Компоненты Windows** (Windows Features)

Порядок установки имеет значение

IIS предоставляет хостинг сервисам ASP.NET и WCF Web, причем для этого сервисы ASP.NET и WCF Web требуют специальной конфигурации. Ваша задача упростится, если вы установите IIS до установки VS. В этом случае VS по умолчанию правильно сконфигурирует настройки сервисов ASP.NET и WCF, так что IIS сможет предоставлять им хостинг. Если вы устанавливаете IIS уже после того, как установили VS, вам потребуется сконфигурировать сервисы ASP.NET и WCF, дав следующие команды в указанном порядке (команды вводятся в одну строку). Сначала конфигурируется ASP.NET:

```
"%windir%\Microsoft.NET\Framework\v4.0.30319\
aspnet_regiis.exe" -i -enable
```

Затем конфигурируется WCF:

```
"%WINDIR%\Microsoft.Net\Framework\v3.0\
Windows Communication Foundation\ServiceModelReg.exe" -r
```

Фактический путь к файлу aspnet_regiis.exe может отличаться от предложенного здесь, потому что в будущем номер версии `v4.x.x.x` может измениться. Поэтому прежде, чем давать первую команду, уточните местоположение нужного вам файла, например, с помощью Windows Explorer.

Создание Web-сайта под IIS 7 в Windows 7

Далее вам потребуется создать Web-сайт под IIS 7 в Windows 7. Для этой цели необходимо выполнить следующие операции:

1. Выберите команды **Пуск** (Start) | **Панель управления** (Control Panel) | **Система и безопасность** (System and Security) | **Администрирование** (Administrative Tools). На экране появится окно **Администрирование** (Administrative Tools), показанное на рис. 11.5.

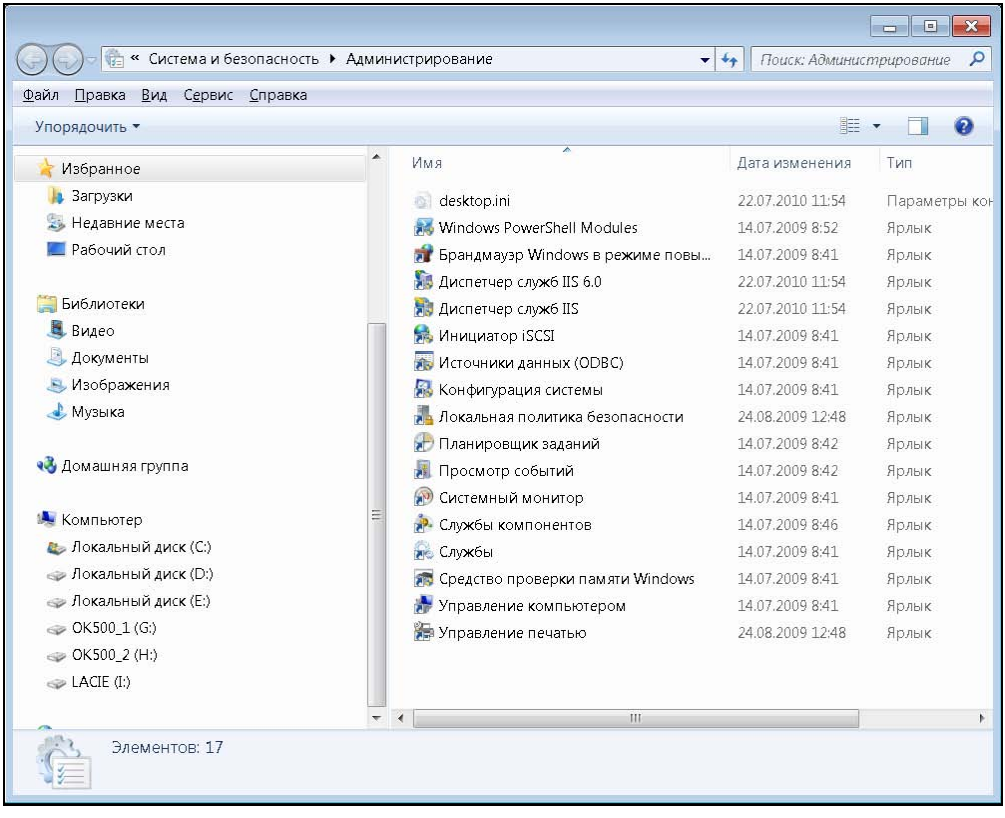


Рис. 11.5. Окно **Администрирование** (Administrative Tools)

2. Выполните двойной щелчок мышью по опции **Диспетчер служб IIS** (Internet Information Services (IIS) Manager), чтобы открыть окно Диспетчера служб IIS, показанное на рис. 11.6.
3. Щелкните правой кнопкой мыши по узлу **сайты** (Sites) в крайней левой панели и выберите из контекстного меню команду **Добавить веб-сайт** (Add Web Site) или в крайней правой панели **Действия** (Actions) пройдите сначала по ссылке **Просмотреть сайты** (View sites), а затем — по ссылке **Добавить веб-сайт** (Add Web Site). Раскроется окно добавления Web-сайта, показанное на рис. 11.7.

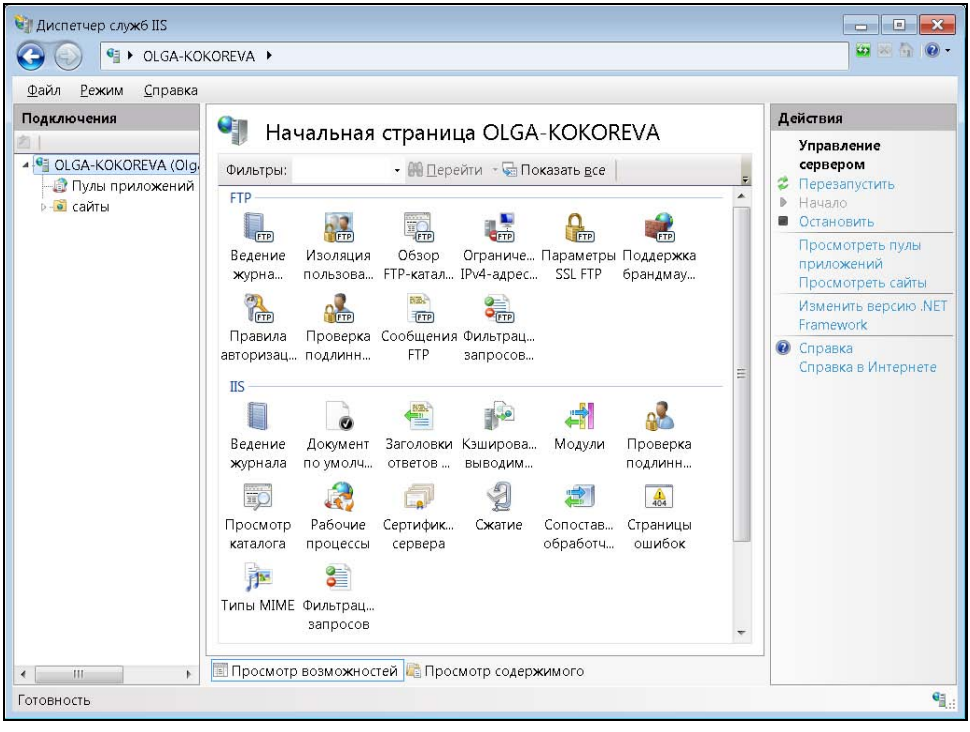


Рис. 11.6. Окно Диспетчера служб IIS

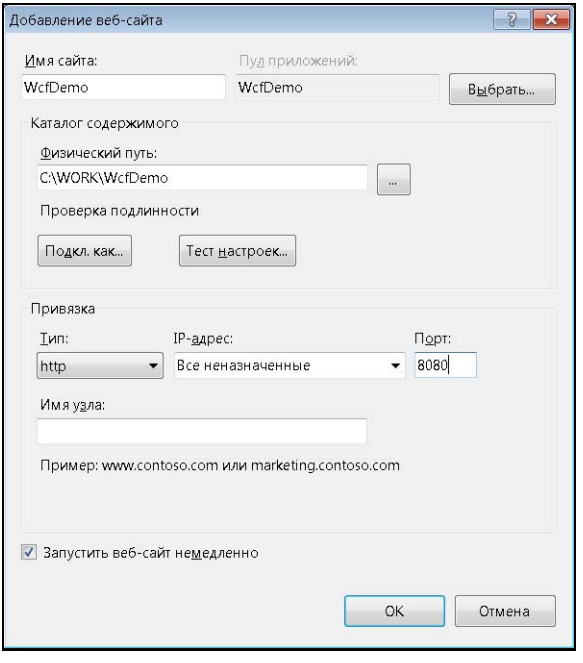


Рис. 11.7. Окно добавления Web-сайта

4. Дайте Web-сайту имя и укажите его физическое местоположение. Как показано на рис. 11.7, сайт из нашего примера называется **WcfDemo**, и он физически расположен в папке c:\WORK\WcfDemo. Обратите внимание на то, что поле **Порт** (Port) в разделе **Привязка** (Binding) указывает номер порта 8080. По умолчанию номер порта для Web-сайта имеет значение 80, но на этом порту можно иметь только один сайт. Как вариант, вы можете изменить имя хоста и сохранить предложенный по умолчанию порт 80. В данном случае используется порт 8080, так что адреса Web-сайтов не перекрываются. Если в вашей системе порт 8080 уже используется, возьмите другой (свободный) номер порта. Щелчок мышью по кнопке **OK** создаст Web-сайт.
5. Убедитесь в том, что в окне Диспетчера IIS выбран сайт **WcfDemo**, щелкните по нему правой кнопкой мыши и выберите из контекстного меню опцию **Добавить FTP-публикацию** (Add FTP Publishing). Как вариант, можно пройти по ссылке с таким же названием в панели **Действия** (Actions). На экране появится окно **Параметры привязки и SSL** (Binding And SSL Settings), показанное на рис. 11.8.

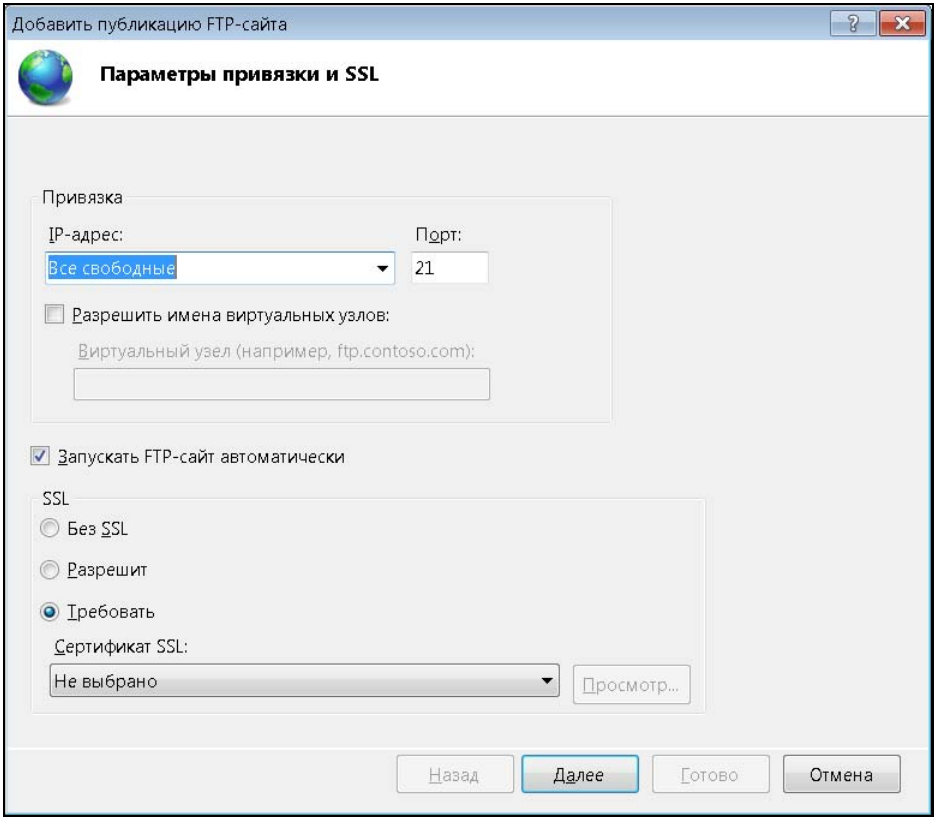


Рис. 11.8. Окно **Параметры привязки и SSL** (Binding And SSL Settings)

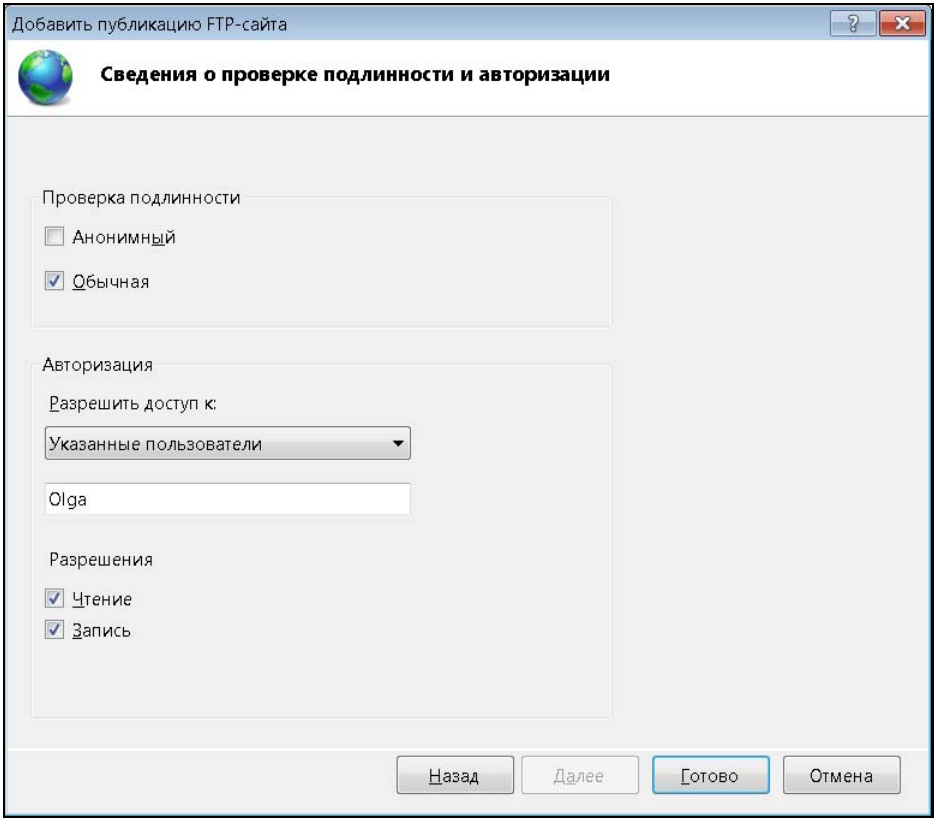


Рис. 11.9. Окно **Сведения о проверке подлинности и авторизации** (Authentication And Authorization)

6. Примите значения по умолчанию и нажмите кнопку **Далее** (Next), чтобы открыть окно **Сведения о проверке подлинности и авторизации** (Authentication And Authorization), показанное на рис. 11.9.
7. В окне **Сведения о проверке подлинности и авторизации** (Authentication And Authorization) укажите, какие пользователи должны получать доступ к вашему Web-сайту. Если вы установите флажок **Анонимный** (Anonymous), то доступ к вашему сайту сможет получать любой пользователь. Более безопасным вариантом является ограничение доступа с его предоставлением только тем пользователям и группам, которым вы доверяете. Базовая авторизация отображает окно с предложением зарегистрироваться для каждого, кто подключается к FTP-сайту. Нажатие кнопки **Готово** (Finish) позволит получать доступ к этому сайту по протоколу FTP.
8. В окне Диспетчера IIS выберите опцию **Пулы приложений** (Application Pools). Пул приложений представляет собой процесс, которому можно назначать Web-сайты. Это позволяет защитить Web-сайты друг от друга, поскольку, если один из процессов аварийно завершит работу, это не повлияет на Web-сайты в других

процессах. IIS создает пул приложений для вашего Web-сайта под тем же именем, под которым вы ранее создали сайт. Выполните двойной щелчок мышью по пулу приложений, названному по имени вашего Web-сайта, и установите для версии .NET Framework номер **v4**. Точный номер версии .NET Framework в будущем может меняться, поэтому вам следует убедиться в том, что установленный номер версии совпадает с тем, для которого вы компилировали ваше приложение в VS.

Как только ваш Web-сайт будет установлен и настроен, вы сможете приступить к развертыванию, которое будет обсуждаться в следующем разделе.

Развертывание сервиса WCF на IIS

Если вы хотите запускать сервис WCF в VS, вам нет необходимости делать что-то дополнительно, потому что VS уже при создании проекта устанавливает его так, чтобы запускать на встроенном сервере. Материалы, приведенные в данном разделе, призваны помочь вам выполнить развертывание IIS на Windows Server 2008. Если вы просто хотите запускать Web-сервис в VS, то этот раздел можно пропустить и продолжать чтение со следующего раздела, посвященного построению клиентского приложения, которое взаимодействует с Web-сервисом. К данному разделу можно вернуться впоследствии, когда необходимость в развертывании действительно возникнет.

Чтоб выполнить развертывание проекта Web-сервиса, вам потребуется получить адрес Web-сайта, модифицировать конфигурационный файл проекта и воспользоваться инструментом VS, который называется Publish.

> ***Совет***
>
> Чтобы выполнить публикацию, вам потребуется запустить VS от имени пользователя Администратор (Administrator). Чтобы сделать это, закройте VS (если среда VS запущена и работает), найдите пункт меню **Visual Studio** в меню **Пуск** (Start), щелкните по нему правой кнопкой мыши и из раскрывшегося контекстного меню выберите команду **Запуск от имени администратора** (Run As Administrator).

В предшествующем разделе, где обсуждалось создание Web-сайта, ему был назначен порт 8080. Хотя Web-сайт называется **WcfDemo**, он расположен на локальном компьютере в домене **localhost**. Если вы выполняете развертывание Web-сервиса на сайт, который имеет доменное имя, вам следует использовать это доменное имя. Например, доменное имя сайта сообщества, посвященного C#, C# Station, выглядит так: **csharp-station.com**, и, соответственно, интернет-адрес будет выглядеть так: **http://www.csharp-station.com**. Каждый Web-сервис, расположенный на этом сайте, имеет расширение имени файла *.svc, и, следовательно, имя, которое VS создаст, будет следующим: WcfDemoCS.CustomerService.svc. Таким образом, адрес для нашего сайта WcfDemo, расположенного на локальном компьютере, должен выглядеть так: **http://localhost:8080/WcfDemoCS.CustomerService.svc**.

Когда вы создаете новый проект типа **WCF Service**, VS добавляет в состав этого проекта файл с именем app.config. Этот файл принадлежит только VS. Файл

app.config никогда не распространяется вместе с вашим Web-сервисом, но на его основе генерируется файл web.config, который включается в процедуру развертывания. В проектах WPF VS использует файл app.config для генерации на его основе файла с именем *projectname*.exe.config, который расположен в той же папке, что и файл *projectname*.exe. Проекты типа **WCF Service** не генерируют конфигурационного файла в выводной папке, но они создают файл web.config при развертывании. Файл web.config вскоре будет обсуждаться отдельно.

В ходе разработки вы работаете с файлом app.config, который вы с легкостью найдете в составе проекта. Найдите и раскройте этот файл. Файл app.config содержит большое количество информации. В листинге 11.11 приведен лишь небольшой фрагмент этого файла, демонстрирующий наиболее характерные элементы конфигурации WCF.

Листинг 11.11. Адрес сервиса WCF в файле app.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
...
  <system.serviceModel>
  <bindings>
     ...
  </bindings>
  <client />
  <services>
    <service name="WcfDemoCS.CustomerService">
       ...
       <endpoint address="mex" binding="mexHttpBinding"
            contract="IMetadataExchange" />
       <host>
         <baseAddresses>
            <add baseAddress=
            "http://localhost:8732/Design_Time_Addresses
            /WcfDemoCS/CustomerService/" />
         </baseAddresses>
       </host>
    </service>
  </services>
...
  </system.serviceModel>
...
</configuration>
```

В листинге 11.11 приведены наиболее характерные элементы файла app.config — `configuration`, `system.serviceModel`, `services`, `service`, `host` и `baseAddresses`. Элемент **baseAddress** выделен полужирным шрифтом. В листинге он разбит на несколько строк, но в самом файле он должен быть написан в одну строку. Элемент

baseAddress объявляет, что приложения должны поддерживать коммуникации с сервисом через этот адрес. Это — адрес, сгенерированный VS для Web-сервера во время разработки сервиса WCF. Чуть ранее уже было показано, каким образом следует составлять адрес нашего приложения. Следовательно, когда вы начнете процедуру развертывания, вам необходимо будет закомментировать строки, указывающие Web-адрес времени разработки, и вместо них вписать фактический адрес, по которому будет производиться развертывание. Пример того, как могут выглядеть заменяемые строки файла app.config, приведен в листинге 11.12.

Листинг 11.12. Пример адреса для развертывания сервиса WCF

```
<baseAddresses>
<!--<add baseAddress=
"http://localhost:8732/Design_Time_Addresses/WcfDemoCS/Service1/" />-->
<add baseAddress=" http://localhost:8080/WcfDemoCS.CustomerService.svc " />
</baseAddresses>
```

Элементы <!-- и --> обозначают комментарии, и все, что заключено между ними, не интерпретируется как конфигурационная информация. Обратите внимание, как используется адрес развертывания (он не заключен в символы комментария). После развертывания вы можете закомментировать адрес развертывания, снять комментарии с адреса времени разработки и продолжить работу над сервисом WCF с помощью встроенного Web-сервиса VS.

В дополнение к редактированию элемента baseAddress, вам необходимо гарантировать, что будет обновлена строка соединения с базой данных в соответствии со средой, в которую производится развертывание. В среде разработки строка соединения с базой данных по умолчанию использует установку Integrated Security = true для регистрационной информации, что подразумевает использование идентификационной информации пользователя, зарегистрированного в системе на текущий момент. В результате в среде разработки приложение работает с регистрационной информацией пула приложений, которому назначен Web-сайт. Проблема с этой конфигурацией заключается в том, что пул приложений не имеет доступа к вашей базе данных. Наилучший подход состоит в том, чтобы создать пользовательскую учетную запись, предназначенную для использования только с вашим приложением, дать этому пользователю доступ к вашей базе данных, а затем написать строку соединения с базой данных, используя идентификационную информацию этого пользователя.

Создайте в вашей ОС Windows пользовательскую учетную запись, которая будет использоваться с SQL Server, и предоставьте этому пользователю доступ к вашей базе данных. Если вы пользуетесь версией SQL Server Express, неплохо будет скачать бесплатный продукт SQL Server Express Management Studio. В том, что касается переменных, которые могут повлиять на настройку безопасности, сверьтесь с документацией по SQL Server. В данной главе примеры будут использовать механизм аутентификации SQL. Итак, создайте пользовательскую учетную запись Windows или SQL и дайте этому пользователю доступ к базе данных **MyShop.**

После того как пользовательская учетная запись будет создана, раскройте файл app.config сервиса WCF и отредактируйте его таким образом, чтобы использовать идентификационную информацию этого пользователя. В целях обеспечения повышенного уровня безопасности, не забудьте задать для пользователя пароль. Пример того, как может выглядеть отредактированный фрагмент файла app.config, задающий строку соединения с базой данных и регистрационную информацию пользовательской учетной записи, приведен в листинге 11.13.

Листинг 11.13. Отредактированный фрагмент файла app.config, задающий строку соединения с базой данных

```
<add name=
"WcfDemoCS.Properties.Settings.MyShopConnectionString"
  connectionString=
"Data Source=.\sqlexpress;Initial Catalog=MyShop;
User ID=MyUserAccount;Password=G7b@H8m2a%lM6y;Pooling=False"
  providerName="System.Data.SqlClient" />
```

Чтобы выполнить развертывание, щелкните правой кнопкой мыши по имени вашего проекта Web-сервиса, **WcfDemo**, и из раскрывшегося контекстного меню выберите команду **Publish**. На экране появится окно **Publish WCF Service**, показанное на рис. 11.10.

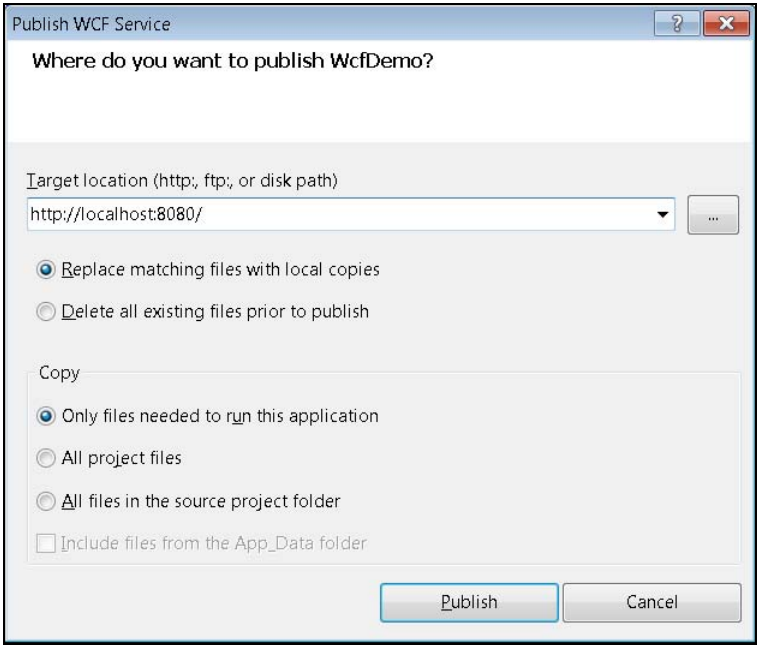


Рис. 11.10. Окно **Publish WCF Service**

В окне **Publish WCF Service** введите в поле **Target location** адрес, по которому вы собираетесь выполнить развертывание сервиса WCF. О том, как составляется этот адрес, уже рассказывалось ранее в этом разделе. При развертывании в вашем распоряжении имеются возможности заменить совпадающие файлы их локальными копиями или же удалить все существующие файлы в папке, предназначенной для развертывания. Обычно лучше всего копировать только те файлы, которые необходимы для запуска приложения, потому что в таком случае развертывание произойдет быстрее и, кроме того, такой подход безопаснее. Флажок **Include files from the App_Data folder** блокирован, так как проект сервиса WCF не содержит папки App_Data. Однако флажок, хоть и блокированный, все же присутствует, потому что тот же самый инструмент используется и для развертывания Web-сайтов ASP.NET, в составе которых может иметься папка App_Data.

Обычно папку App_Data предпочитают не распространять, потому что в ней может содержаться файл базы данных большого размера, что может существенно замедлить работу вашего приложения. Конечно, если на сервере, куда производится развертывание, установлен продукт SQL Server Express, которому требуется база данных в папке App_Data, этот флажок следует установить, чтобы включить базу данных в процесс развертывания. Чтобы осуществить развертывание сервиса, нажмите кнопку **Publish**.

Когда процесс развертывания завершится, в строке состояния VS появится сообщение **Publish Succeeded** или **Publish Failed**, в зависимости от успеха или неудачи процедуры. Если публикация завершается неудачей, откройте окно **Output**, чтобы выяснить причину сбоя. Проще всего сделать это, нажав клавиатурную комбинацию <CTRL>+<W>, <O>. Существует множество причин, по которым развертывание может потерпеть неудачу, поэтому внимательно прочтите сообщение об ошибке, и, возможно, вы сами сразу же выясните, что же пошло не так. Проверьте правильность настройки вашего Web-сайта — о том, как это делается, рассказывалось в предыдущем разделе. Если выяснить причину неудачи все же не удается, попробуйте найти нужную вам информацию на сайте Microsoft Developer Network (MSDN): **http://msdn.microsoft.com**, где можно выполнить поиск по обширной базе знаний (Knowledge Base). Как вариант, попробуйте скопировать сообщение об ошибке и выполните поиск по Интернету с помощью вашего предпочитаемого поисковика. Многие проблемы с развертыванием коренятся в неверной настройке IIS, так что не помешает узнать чуть больше о том, как работает IIS. Издательство McGraw-Hill выпустило отличную книгу Марти Мэттьюса (Marty Matthews), "Windows Server 2008: A Beginner's Guide", где можно найти немало ценной информации об IIS 7. Если вы выполняете развертывание на IIS 6, посмотрите предыдущее издание этой книги, описывающее Windows Server 2003.

Теперь, когда вы уже научились разрабатывать сервисы WCF и выполнять их развертывание, пора приступить к написанию программ, которые взаимодействуют с этим сервисом. Этот вопрос будет обсуждаться в следующем разделе.

Взаимодействие приложений с сервисом WCF

Любое приложение .NET может взаимодействовать с Web-сервисом. Фактически, одним из преимуществ, предоставляемых Web-сервисом, является то, что его функциональные возможности могут использоваться множеством приложений. Теоретически, любое приложение, работающее на любой платформе, может взаимодействовать с Web-сервисом, так как технологии, на которых основываются Web-сервисы, базируются на открытых стандартах, таких, как HTTP и XML. На практике, цель межплатформенных коммуникаций достигается за счет прогрессивных технологий, разработанных системными архитекторами и инженерами на основе углубленных знаний о деталях и внутренних механизмов работы Web-сервисов. Впрочем, чтобы начать работу, вам достаточно знать, что поддерживать коммуникации с Web-сервисами может любая из технологий .NET. В последующих нескольких разделах я продемонстрирую, как добиться того, чтобы ваши клиентские приложения могли взаимодействовать с Web-сервисами. Начнем с рассмотрения такой общей задачи, как создание ссылки на Web-сервис.

Создание ссылки на Web-сервис

Вне зависимости от того, какого типа приложение вы разрабатываете, вы можете создать ссылку на Web-сервис, иногда называемую привязкой к сервису. Для всех типов приложений это делается одинаково. Для того, чтобы протестировать эту возможность, подойдет любой проект — консольное приложение (Console), WPF, ASP.NET или Silverlight. Щелкните по имени проекта правой кнопкой мыши и выберите из раскрывшегося контекстного меню команду **Add Service Reference**. Вы увидите окно **Add Service Reference**, показанное на рис. 11.11.

Как вы, конечно, помните, в данной главе ранее уже говорилось о настройке Web-сервиса и конфигурировании адреса Web-сервиса. Теперь сосредоточимся именно на адресе, потому что именно по этому адресу вы производили развертывание Web-сервиса — этот адрес вводится в поле **Address** окна **Add Service Reference** (см. рис. 11.11). Если вы пользуетесь Web-сервером, встроенным в VS, и при этом проект Web-сервиса находится в одном и том же решении, то на этот случай предусмотрена очень удобная возможность автоматизировать эту задачу. Просто нажмите кнопку **Discover**, расположенную справа от поля адреса, и список Web-сервисов сразу же появится в соответствующем поле.

> ***Примечание***
>
> Тот адрес, который вы увидите на своем компьютере, будет отличаться от адреса, показанного на рис. 11.11, потому что в вашем случае будут использоваться другое имя проекта, номер порта и имя сервиса.

Если вам требуется использовать уже развернутый Web-сервис, просто введите адрес этого Web-сервиса в поле **Address**. Например, ранее в этой главе уже демонстрировалась процедура развертывания Web-сервиса на локальный сервер IIS и пояснялось, что для использования этого развернутого Web-сервиса вам следует ввести в поле

Address следующий адрес: **http://localhost:8080/WcfDemo.CustomerService.svc**. Естественно, в реальной жизни имена сервисов будут отличаться от **WcfDemo.CustomerService.svc** (это всего лишь пример, который мы использовали в этой главе). Чтобы определить реальное имя сервиса, перейдите в физический каталог, в который было выполнено развертывание сервиса, найдите там *.svc-файл и используйте его имя в качестве имени сервиса. Иногда могут возникать такие ситуации, когда вам потребуется пользоваться Web-сервисами сторонних компаний или других организаций. В этих случаях вы можете получить нужный адрес от разработчиков или сотрудников компании, предоставляющей этот Web-сервис, или же прочесть документацию и самостоятельно выяснить, какой адрес следует использовать. Если вы добавляете адрес вручную, то после его ввода нажмите кнопку **Go** (см. рис. 11.11).

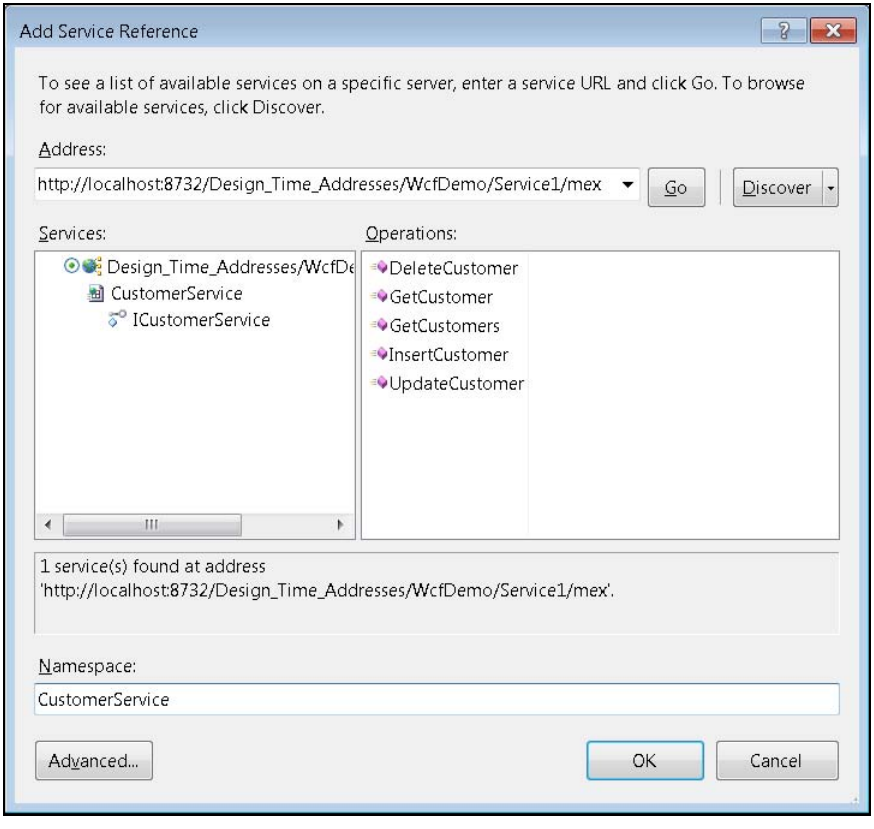


Рис. 11.11. Окно **Add Service Reference**

После того как вы либо нажмете кнопку **Discover** для автоматического обнаружения сервиса, либо введете адрес развертывания вручную, в списке **Services** появятся один или несколько сервисов. Если на данном этапе вы получите сообщение об ошибке, то это может говорить либо о том, что вы некорректно указали адрес,

либо о том, что сервис временно недоступен (по той или иной причине остановлен), либо (в том случае, если сервис находится в вашем собственном решении) о том, что сервис не может быть скомпилирован. В первую очередь, проверьте правильность ввода адреса, если вы вводили его вручную. Если вы ссылаетесь на проект, находящийся в составе вашего собственного решения, перекомпилируйте проект Web-сервиса, чтобы убедиться в том, что он не содержит ошибок и сборка проходит успешно. В случае ошибок, исправьте их, перекомпилируйте сервис и повторите попытку сослаться на него. Убедившись в том, что все проблемы на вашей стороне успешно решены, обратитесь за помощью к владельцам Web-сервиса.

Если с Web-сервером можно успешно поддерживать коммуникации, то вы увидите список предоставляемых им услуг. Просмотрите список сервисов и найдите интерфейс к сервису, в котором вы заинтересованы. На рис. 11.11 выбран сервис `ICustomerService`, а в расположенной справа панели **Operations** перечислены все доступные операции. Если вернуться к тому моменту, когда мы обсуждали создание сервиса `CustomerService`, то вы сразу же заметите, что в этом окне отображаются созданные нами интерфейс и методы класса. Если вы не видите нужного вам интерфейса или метода, проверьте атрибуты в коде — вам необходимо убедиться в том, что интерфейс имеет атрибут `ServiceContract`, а любые методы, которые должны быть видимыми, должны иметь атрибут `OperationContract`.

Web-сервис создаст в вашем проекте посредника (proxy), представляющего собой класс, посредством которого осуществляется все взаимодействие с Web-сервисом. Для этого будет использовано пространство имен по умолчанию, объявленное в свойствах вашего проекта. Пространство имен, которое предлагается по умолчанию в окне **Add Service Reference**, имеет стандартное значение `Service1`, и вам рекомендуется заменить это название на что-нибудь более осмысленное — например, `CustomerService`, как показано на рис. 11.11. В результате будет создан класс-посредник (proxy class) в пространстве имен `MyProjectNamespace.CustomerService`. Об этом нужно знать, потому что вам потребуется создать экземпляр класса-посредника, а для этого вы должны знать, в каком пространстве имен он располагается. Чтобы создать ссылку на сервис, нажмите кнопку **OK**. Ссылка на сервис будет выглядеть примерно так, как показано на рис. 11.12.

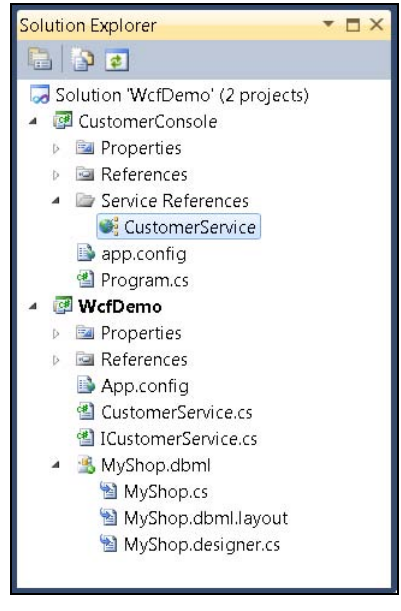


Рис. 11.12. Новая ссылка на сервис в составе проекта

Как показано на рис. 11.12, в составе проекта появится новая папка, озаглавленная Service References. Ссылка CustomerService в составе этой папки получает имя, которое вы указали для пространства имен в поле **Namespace** окна **Add Service Reference** (см. рис. 11.11).

Теперь, когда ссылка на сервис создана, вы можете использовать ее в любом приложении .NET.

В следующем разделе будет показано, как следует писать код, который будет взаимодействовать с вашим Web-сервисом.

Кодирование вызовов к Web-сервису

В данном разделе рассказывается о написании кода, обращающегося к Web-сервису. Сначала мы рассмотрим отдельные утверждения, которые обращаются к Web-сервису, снабженные необходимыми пояснениями, а затем приведем целостный листинг, в котором будет показано, как клиентское приложение "общается" с Web-сервисом. В нашем случае, для простоты, программа, пользующаяся услугами Web-сервиса, будет представлять собой простое консольное приложение, назовите его CustomerConsole. Чтобы протестировать приведенный здесь код на практике, создайте новое консольное приложение и добавьте код, приведенный в данном разделе, в состав метода `Main`. Полный код консольного приложения, поддерживающего коммуникации с Web-сервисом, будет приведен в листингах 11.26 (код на C#) и 11.27 (код на VB). Однако мы начнем с рассмотрения отдельных утверждений и постепенно соберем их воедино так, чтобы вы лучше поняли роль каждого из них во взаимодействии с Web-сервисом `CustomerService`, разработке и развертыванию которого были посвящены предшествующие разделы данной главы.

При создании ссылки на сервис, как было показано в предшествующем разделе, VS создает новый класс-посредник (proxy). Этот прокси-класс выглядит в точности так, как и класс вашего Web-сервиса, однако он не содержит аналогичного кода. Вместо этого прокси-класс занимается трансляцией вызовов от клиента и коммуникациями с Web-сервисом. Прокси-класс, созданный после добавления ссылки на сервис, как было показано в предшествующем разделе, называется `CustomerServiceClient` (рис. 11.13).

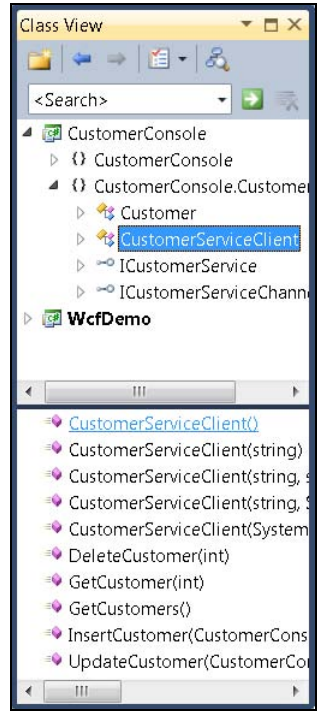


Рис. 11.13. Прокси-класс `CustomerServiceClient`, созданный после добавления ссылки на Web-сервис `CustomerService`

Не забудьте добавить для прокси-класса утверждение `using` (`Imports` — для VB). Так как пространство имен по умолчанию в нашем примере называется `CustomerConsole`, то пространство имен для прокси-класса Web-сервиса будет выглядеть так: `CustomerConsole.CustomerService`. Код на C#, который создает экземпляр прокси-класса, выглядит так:

```
var svc = new CustomerServiceClient();
```

Код на VB, выполняющий ту же самую задачу, будет выглядеть так:

```
Dim svc = New CustomerServiceClient
```

Как видите, имя прокси-класса формируется из имени, назначенному ссылке на сервис, к кон-

цу которого добавляется подстрока `Client`. Как и в случае с любым другим классом, вам необходимо создать экземпляр прокси-класса, с именем `svc`. Использование прокси приводит к тому, что ваш код будет воспринимать вещи таким образом, как если бы все находилось в одном проекте, тогда как в действительности прокси-класс делает вызов через `http` и отправляет пакет XML Web-сервису. Web-сервис транслирует код XML в вызов к методу, исполняет код для вызова метода и транслирует полученные результаты обратно в XML. В течение всего этого времени посредник (proxy) ожидает ответа от Web-сервиса. Получив ответ в форме пакета XML, прокси-класс транслирует его в объект .NET и передает его вызывающему коду. Если метод возвращает не тип, а пустое значение (`void`), то это значит, что возвращаемых значений не существует.

Имея ссылку на сервис (service reference), вы можете начинать взаимодействие с Web-сервисом. Фрагменты кода на C# и VB, приведенные в листингах 11.14 и 11.15, соответственно, создают в базе данных запись для нового клиента, вызывая метод `InsertCustomer` через прокси-класс Web-сервиса.

Листинг 11.14. Код на C#, создающий запись нового клиента через прокси-класс

```
var newCust = new Customer
{
        Age = 36,
        Birthday = new DateTime(1974, 8, 22),
        Income = 56000m,
        Name = "Venus"
};

var newCustID = svc.InsertCustomer(newCust);
```

Листинг 11.15. Код на VB, создающий запись нового клиента через прокси-класс

```
Dim newCust = New Customer

With newCust
     .Age = 36
     .Birthday = New DateTime(1974, 8, 22)
     .Income = 56000
     .Name = "Venus"
End With

Dim newCustID As Integer
newCustID = svc.InsertCustomer(newCust)
```

На данном этапе у вас может возникнуть вопрос — а откуда берется тип `Customer`? Как уже говорилось в предыдущем разделе этой главы, при обсуждении индивидуально определенных объектов, тип `Customer` представляет собой тип-

посредник (прокси-тип) для типа `Customer`, который был определен в LINQ to SQL. Поскольку мы установили для свойства **Serialization Mode** объекта **LINQ to SQL** значение **Unidirectional**, Web-сервис может передавать определение типа `Customer` с интерфейсом Web-сервиса, в результате чего появляется прокси-тип `Customer`.

Чтобы выполнить операцию вставки, следует воспользоваться ссылкой на прокси-класс, `svc`, для передачи экземпляра прокси-типа `Customer`. В листингах 11.16 и 11.17 показано, как получить запись указанного клиента от Web-сервиса.

Листинг 11.16. Получение записи указанного клиента от Web-сервиса (код на C#)

```
Customer cust = svc.GetCustomer(newCustID);
```

Листинг 11.17. Получение записи указанного клиента от Web-сервиса (код на VB)

```
Dim cust As New Customer
cust = svc.GetCustomer(newCustID)
```

В этом коде используется ссылка на прокси-сервис для вызова метода `GetCustomer` с указанием `ID` запрашиваемого клиента, в результате чего будет возвращен экземпляр прокси-типа `Customer`. Код, приведенный в листингах 11.18 и 11.19, показывает, каким образом можно обновить экземпляр `Customer`.

Листинг 11.18. Обновление экземпляра Customer (код на C#)

```
cust.Income = 49000m;
svc.UpdateCustomer(cust);
```

Листинг 11.19. Обновление экземпляра Customer (код на VB)

```
cust.Income = 49000
svc.UpdateCustomer(cust)
```

Ссылка `cust` в этом примере представляет ту же самую ссылку, которая была создана ранее. В рассматриваемых примерах мы обновляем только свойство `Income`. Далее, используется прокси-сервис для вызова метода `UpdateCustomer` с передачей ссылки на прокси-тип `Customer`. Если вам требуется просмотреть внесенные изменения, вы вновь можете воспользоваться методом `GetCustomer`, как показано в листингах 11.20 и 11.21.

Листинг 11.20. Просмотр внесенных изменений (код на C#)

```
Customer updatedCust = svc.GetCustomer(cust.CustomerID);
```

Листинг 11.21. Просмотр внесенных изменений (код на VB)

```
Dim updatedCust As Customer
updatedCust = svc.GetCustomer(cust.CustomerID)
```

Аналогичным образом, вы можете удалить запись клиента из базы данных. На C# это делается так:

```
svc.DeleteCustomer(updatedCust.CustomerID);
```

Аналогичный код на VB, выполняющий ту же задачу, выглядит так:

```
svc.DeleteCustomer(updatedCust.CustomerID)
```

Как и в предыдущем примере, здесь используется ссылка на класс-посредник для вызова метода `DeleteCustomer` с передачей `ID` клиента, запись которого была обновлена. Ссылка `updatedCust` была получена от предыдущего вызова к методу `GetCustomer`. Если вы захотите получить от Web-сервиса записи всех клиентов, вы можете воспользоваться методом `GetCustomers`, как показано в листингах 11.22 и 11.23.

Листинг 11.22. Получение от Web-сервиса записей всех клиентов (код на C#)

```
Customer[] customers = svc.GetCustomers();
```

Листинг 11.23. Получение от Web-сервиса записей всех клиентов (код на VB)

```
Dim customers As Customer()
customers = svc.GetCustomers()
```

Хотя код, приведенный в листингах 11.22 и 11.23, и похож на вызовы, продемонстрированные в предыдущих примерах, вы наверняка заметите, что в данном случае возвращаемое значение метода `GetCustomers` представляет собой массив значений типа `Customer`, `Customer[]` (`Customer()` — для VB). Однако Web-сервис определяет метод `GetCustomers` как возвращающий значение, представляющее собой список объектов типа `Customer`, `List<Customer>` (`List(Of Customer)` — в VB), как было указано в интерфейсе `ICustomerService` (см. листинги 11.3 и 11.4) и реализовано в классе `CustomerService` (см. листинги 11.9 и 11.10). Как вы помните, прокси-класс отвечает за трансляцию значения в формате XML, возвращаемого Web-сервисом, в объект или, как в данном случае, в коллекцию объектов. По умолчанию, прокси-класс транслирует все коллекции в массивы. Однако вы имеете возможность изменить тип возвращаемой коллекции. Для этого следует выполнить щелчок правой кнопкой мыши по папке Service Reference в составе вашего проекта и выбрать из контекстного меню команду **Configure Service Reference**. В результате будет открыто окно **Service Reference Settings**, показанное на рис. 11.14.

Большинство настроек, доступных в окне **Service Reference Settings**, предназначены для работы над сценариями, более сложными, нежели рассматриваемый простой пример. В данном случае нас интересует настройка **Collection Type** в группе опций **Data Type**. Замените предложенное по умолчанию значение **Collection Type**, выбрав из раскрывающегося списка значение **System.Collections.Generic.List** вместо предлагаемого по умолчанию значения **System.Array**, а затем нажмите кнопку **OK**. Затем измените код (см. листинги 11.22 и 11.23), содержащий вызов к методу `GetCustomers`, кодом, приведенным в листингах 11.24 и 11.25.

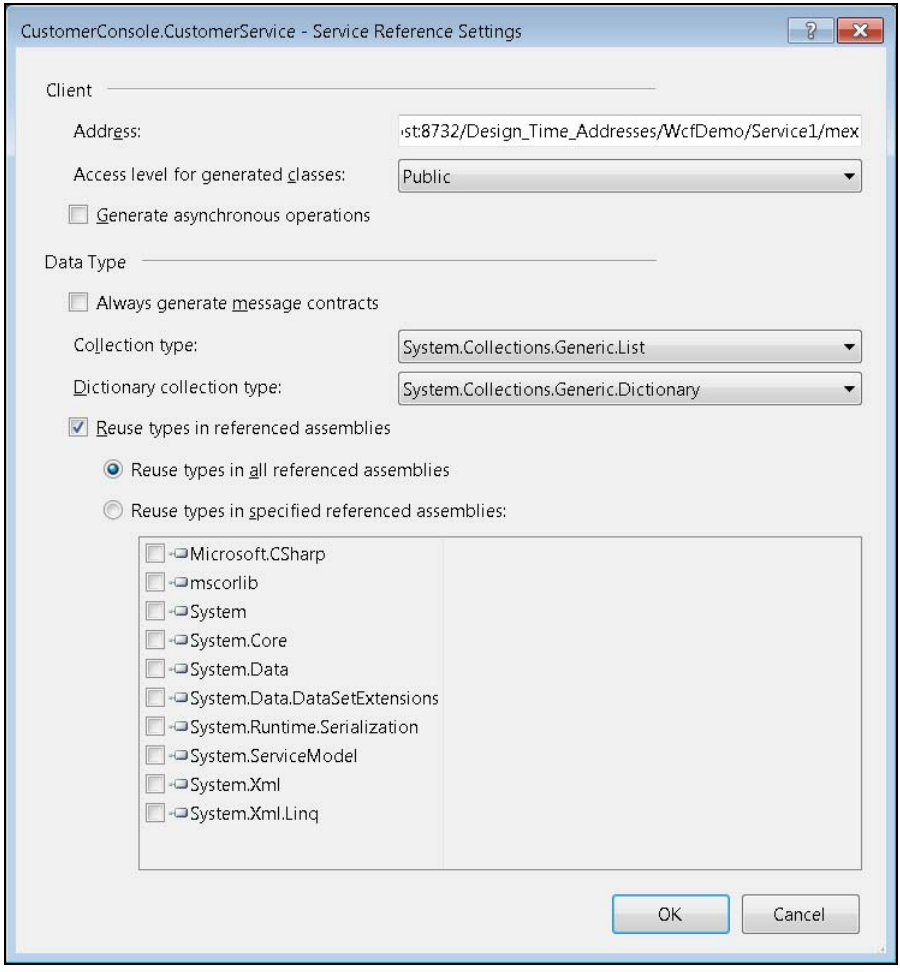


Рис. 11.14. Окно **Service Reference Settings**

Листинг 11.24. Модифицированный код на C#, содержащий вызов к методу GetCustomers

```
List<Customer> customers = svc.GetCustomers();
```

Листинг 11.25. Модифицированный код на VB, содержащий вызов к методу GetCustomers

```
Dim cust As New Customer
cust = svc.GetCustomer(newCustID)
```

В примерах, показанных в листингах 11.24 и 11.25, прокси-класс будет транслировать возвращаемые значения в типы `List<Customer>` (`List(Of Customer)` — для VB). Я умышленно продемонстрировал изменение типов возвращаемого значения

уже после создания Web-сервиса, чтобы продемонстрировать вам практику изменения типов возвращаемой коллекции. Однако вы можете изменить эту настройку сразу же после создания ссылки на Web-сервис. Вернитесь к рис. 11.11 — в нижней части окна **Add Service Reference** есть кнопка **Advanced**. Если вы нажмете эту кнопку при создании ссылки на сервис, то на экране появится окно **Service Reference Settings**, показанное на рис. 11.14, что позволит вам изменить тип возвращаемой коллекции еще при создании ссылки на сервис.

Итак, мы детально рассмотрели все операции по взаимодействию с Web-сервисом. Не забывайте о том, что точно такие же приемы и подходы применяются и для любого другого типа приложений .NET. Теперь давайте сведем все изложенные концепции воедино. Полный код приложения, использующего Web-сервис, приведен в листингах 11.26 и 11.27.

Листинг 11.26. Код консольного приложения на C#, взаимодействующего с Web-сервисом

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using CustomerConsole.CustomerService;

namespace CustomerConsole
{
        class Program
        {
                static void Main()
                {
                        var svc = new CustomerServiceClient();

                        var newCust = new Customer
                        {
                                Age = 36,
                                Birthday = new DateTime(1974, 8, 22),
                                Income = 56000m,
                                Name = "Venus"
                        };

                        var newCustID = svc.InsertCustomer(newCust);

                        Console.WriteLine("New Customer ID: " + newCustID);

                        Customer cust = svc.GetCustomer(newCustID);

                        Console.WriteLine("New Customer: " + cust.Name);

                        cust.Income = 49000m;
```

```
            svc.UpdateCustomer(cust);

            Customer updatedCust = svc.GetCustomer(cust.CustomerID);

            Console.WriteLine("Economic Adjustment: " + cust.Income);

            svc.DeleteCustomer(updatedCust.CustomerID);

            //Customer[] customers = svc.GetCustomers();
            List<Customer> customers = svc.GetCustomers();

            Console.WriteLine("\nAll Customers:\n");
            foreach (var custItem in customers)
            {
            Console.WriteLine(custItem.Name);
            }

            Console.ReadKey();
        }
    }
}
```

Листинг 11.27. Код консольного приложения на VB, взаимодействующего с Web-сервисом

```
Imports CustomerConsoleVB.CustomerService

Module Module1

    Sub Main()

        Dim svc = New CustomerServiceClient

        Dim newCust = New Customer

        With newCust
            .Age = 36
            .Birthday = New DateTime(1974, 8, 22)
            .Income = 56000
            .Name = "Venus"
        End With

        Dim newCustID As Integer

        newCustID = svc.InsertCustomer(newCust)

        Console.WriteLine("New Customer ID: " & newCustID)
```

```
            Dim cust As New Customer

            cust = svc.GetCustomer(newCustID)

            cust.Income = 49000

            svc.UpdateCustomer(cust)

            Dim updatedCust As Customer

            updatedCust = svc.GetCustomer(cust.CustomerID)

            Console.WriteLine("Economic Adjustment: " & cust.Income)

            svc.DeleteCustomer(updatedCust.CustomerID)

            Dim customers As List(Of Customer)

            'Dim customers As Customer()

            customers = svc.GetCustomers()

    End Sub

End Module
```

Развертывание клиентского приложения, взаимодействующего с Web-сервисом

При развертывании клиентского приложения, которое взаимодействует с Web-сервисом, вам необходимо обновить адрес сервиса в конфигурационном файле. Типы конфигурационных файлов могут меняться, в зависимости от типа приложения, которое вы создаете и затем распространяете. Конфигурационные файлы, используемые с приложениями типов, рассмотренных в данной книге, перечислены в табл. 11.1.

Таблица 11.1. *Конфигурационные файлы, используемые с клиентскими приложениями, рассмотренными в данной книге*

Тип приложения	Конфигурационный файл
Консольное (Console)	App.config
WPF	App.config
ASP.NET MVC	Web.config
Silverlight	ServiceReferences.ClientConfig

Вне зависимости от имени, в каждом конфигурационном файле присутствует элемент `system.serviceModel` с конфигурационными параметрами для Web-сервиса.

В листинге 11.28 показаны фрагменты конфигурационного файла, которые требуется изменить таким образом, чтобы получить возможность поддержания коммуникаций с Web-сервисом.

Листинг 11.28. Конфигурация клиентского приложения, взаимодействующего с Web-сервисом

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
        <system.serviceModel>
...
                <client>
                      <endpoint address="http://localhost:8732
                     /Design_Time_Addresses/WcfDemoCS/CustomerService /"
                          binding="wsHttpBinding"
                          bindingConfiguration="WSHttpBinding_ICustomerService"
                          contract="CustomerService.ICustomerService"
                          name="WSHttpBinding_ICustomerService">
...
        </system.serviceModel>
</configuration>
```

Просмотрев весь путь: `system.serviceModel`, `client`, `endpoint`, вы найдете атрибут `address`. В примере, показанном в листинге 11.28, в качестве значения атрибута `address` указан проект `WcfDemo` в составе того же самого решения, в состав которого входит и проект клиентского приложения. При развертывании вашего клиента, ему нужно будет указать путь для поддержания коммуникаций с Web-сервисом, развернутым на другой сайт. Код, приведенный в листинге 11.29, содержит модифицированный атрибут `address`, задающий путь клиенту для коммуникаций с Web-сервисом, развернутым на IIS, таким образом, как было показано ранее в этой главе.

Листинг 11.29. Модифицированный атрибут address, указывающий клиентскому приложению путь для коммуникаций с Web-сервисом, развернутым на IIS

```
<endpoint
address="http://localhost:8080/WcfDemoCS.CustomerService.svc"
binding="wsHttpBinding"
bindingConfiguration="WSHttpBinding_ICustomerService"
contract="CustomerService.ICustomerService"
name="WSHttpBinding_ICustomerService">
```

Значение `address` включает в свой состав имя файла, WcfDemoCS.CustomerService.svc, которое было автоматически сгенерировано при развертывании сервиса WcfDemo. Чтобы просмотреть имя этого файла, загляните в физическую папку, в которую было произведено развертывание Web-сервиса.

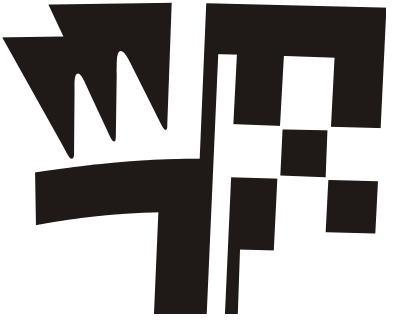
Создание Web-сервиса на Web-сайте

Материал, изложенный в данной главе, описывал создание Web-сервиса как отдельного проекта. При использовании этого подхода подразумевается, что вы имеете возможность сконфигурировать Web-сайт IIS для вашего Web-сервиса и можете создать еще один Web-сайт IIS для вашего приложения, если вы пишете Web-приложения, являющиеся клиентами Web-сервиса. Однако это не всегда возможно — например, вы не можете использовать этот подход, если выполняете развертывание на сервер стороннего Internet-провайдера, где вы можете иметь только один Web-сайт. На случай такой ситуации, вам предоставляется дополнительная опция добавления Web-сервиса к существующему Web-сайту.

Чтобы воспользоваться этой возможностью, создайте новый Web-сайт ASP.NET MVC. Щелкните правой кнопкой мыши по имени проекта, выберите из контекстного меню команды **Add | New Item** и создайте новый сервис WCF. Вы получите файл интерфейса, IService1.cs; адресуемый файл сервиса; класс реализации, Service1.svc; и файл Service1.svc.cs, который можно найти в ветви Service1.svc. Вся информация, изложенная ранее в этой главе, применима и для кодирования Web-сервиса как части Web-приложения. Вполне возможно, что этот вариант вам подойдет.

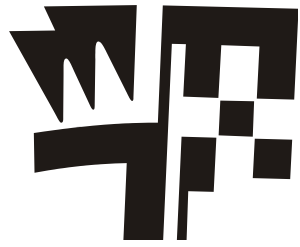
Заключение

В этой главе было рассказано о построении Web-сервиса, развертывании готовых Web-сервисов и написании клиентских приложений, взаимодействующих с Web-сервисом. Вы узнали об определении соглашений для Web-сервисов, включая интерфейсы и атрибуты. Далее было рассказано о реализации сервисов. В разделе, посвященном развертыванию, было рассказано о том, как предоставлять хостинг Web-сервису с помощью IIS и как использовать программу VS Publish Wizard для осуществления развертывания. Наконец, в заключительной части этой главы было рассказано о написании клиентских приложений, создающих ссылки на Web-сервис и поддерживающих с ним коммуникации.



ЧАСТЬ IV

Расширение возможностей VS 2010



Глава 12

Индивидуальная настройка среды разработки

В этой главе будут рассмотрены следующие ключевые концепции:

- Реализация индивидуальных шаблонов;
- Создание индивидуально настраиваемых автоматически генерируемых фрагментов кода (Custom Snippets);
- Написание макросов.

В дополнение к замечательным функциональным возможностям, которые были описаны в предшествующих главах, VS позволяет вам выполнять индивидуальную настройку среды разработчика. В этой главе будут обсуждаться такие средства индивидуальной настройки рабочей среды, как индивидуальные шаблоны (custom templates), индивидуально настраиваемые автоматически генерируемые фрагменты кода (custom snippets) и макросы (macros).

На протяжении всей книги вы начинали работу над каждой главой с создания новых проектов и добавления в их состав новых элементов, пользуясь при этом теми возможностями, которые предлагаются средой разработки VS. В некоторых случаях может оказаться желательным иметь собственный тип проекта, созданный "с нуля" или за счет индивидуальной настройки одного из существующих шаблонов в соответствии с вашими собственными потребностями. Это же применимо и к добавлению новых элементов проектов — например, специальных типов или файлов классов, для которых вы можете полностью изменить содержимое или создать новые элементы.

В *главе 2* было продемонстрировано использование автоматически генерируемых фрагментов кода (snippets), а в *главах 3* и *4* было показано, как с помощью автоматически генерируемых фрагментов быстро кодировать общие утверждения. Однако вы совершенно не обязаны ограничиваться использованием только готовых автоматических фрагментов — при желании вы можете создавать свои собственные. Кроме того, VS предлагает окно управления автоматически генерируемыми фрагментами, в котором вы можете их организовать, добавлять, удалять и всячески перенастраивать.

Каждый раз, когда вы замечаете, что вы часто выполняете повторяющиеся последовательности одних и тех же действий, будет весьма полезно записать эти последовательности с тем, чтобы в следующий раз не выполнять рутинную работу, а быстро выполнить свою задачу в автоматическом режиме. Например, если вы

обнаружите, что из раза в раз вы набираете одни и те же последовательности нажатия клавиш на клавиатуре, вам, естественно, захочется их записать, чтобы впоследствии все действия можно было выполнить как одну команду. В этой ситуации на помощь вам придут макросы (Macros).

Итак, приступим к изучению возможностей по индивидуальной настройке среды разработки VS. Начнем с разработки индивидуальных шаблонов.

Реализация индивидуальных шаблонов

Как уже неоднократно демонстрировалось в предшествующих главах этой книги, VS упрощает вашу работу по созданию новых проектов, предлагая стандартные шаблоны типовых проектов, в состав которых входят наиболее ходовые элементы проекта и код, представляющий собой "скелет" или "каркас" будущего приложения. В большинстве случаев это серьезно ускоряет работу по созданию нового проекта. По мере того как вы набираетесь опыта программирования с помощью VS, вы рано или поздно обнаружите, что вам нужны другие шаблоны проектов, которые содержат другие элементы и другой код, отличные от того, что предлагается в типовых шаблонах, поставляющихся в составе VS. В данном разделе как раз и будет продемонстрировано создание индивидуальных проектов, а также рассказано о том, как создавать собственные шаблоны проектов и их элементов.

Создание новых шаблонов проектов

Если вы работаете над одним проектом долгое время, вас вполне может удовлетворить такой подход, когда вы создаете его на основе одного из типовых шаблонов, а потом вносите одноразовые модификации. Однако если вы регулярно начинаете работать над новыми проектами, и каждый из ваших проектов имеет схожие элементы и схожий код, индивидуальная модификация шаблона проекта может оказаться очень полезной. Всегда найдется множество причин, по которым программисту может понадобиться индивидуально настроенный шаблон проекта — например, это может быть ситуация, когда вы постоянно добавляете в состав нового проекта одни и те же элементы, которые не включены в типовой шаблон по умолчанию, или же ситуация, когда вам из раза в раз приходится удалять из типового шаблона те элементы, которые вы никогда не используете в своих программах. Наконец, возможна и такая ситуация, когда вы постоянно меняете свойства элементов типового шаблона при обновлении версий. При этом вы можете не только создавать шаблоны на основе уже существующих, но и имеете возможность создать абсолютно новый шаблон нового проекта такого типа, которого еще не существует.

Пример, который будет рассмотрен в данном разделе, демонстрирует модификацию шаблона проекта ASP.NET MVC. Изменения, вносимые в шаблон, подразумевают удаление большинства кода, предоставляемого в составе шаблона. В данном случае предполагается, что после того, как вы создадите несколько приложений ASP.NET MVC, вы можете счесть лишними большинство стандартных файлов и предпочтете начинать с менее наполненной "скелетной" модели приложения, а необходимый код дописывать самостоятельно.

Модификация проекта

Простейший способ начать создание нового шаблона проекта заключается в том, чтобы запустить новый проект, тип которого имеет наиболее близкое сходство с тем проектом, который вы планируете создать. Если вы хотите начать "с чистого листа", проще всего создать проект нового консольного приложения (Console), потому что в таких проектах содержится меньше всего элементов, удаление которых не потребует длительного времени. В сценарии, рассматриваемом в данном разделе, мы создадим специализированный проект ASP.NET MVC, поэтому имеет смысл начать с создания нового проекта ASP.NET MVC. Для этой цели выполните следующие шаги:

1. Чтобы запустить новый проект, нажмите клавиатурную комбинацию <CTRL>+<SHIFT>+<N> и в качестве типа проекта выберите опцию **ASP.NET MVC 2 Web Application**. Дайте имя проекту и решению (в нашем примере используется имя **Custom ASP.NET MVC Web Application**) и укажите местоположение нового проекта (это может быть любая папка в файловой системе). Нажмите кнопку **OK**, чтобы создать новый проект. Далее, когда появится окно **Create Unit Test Project**, установите опцию **No, do not create a unit test project** и нажмите кнопку **OK**. VS создаст новое решение, содержащее один проект. О проектах этого типа и их стандартном содержимом рассказывалось в *главе 9*.
2. Откройте папку Controllers и удалите все ее содержимое.
3. Откройте папку Models и удалите все ее содержимое.
4. Откройте папку Views и удалите папки Account и Home, оставив в сохранности все остальное.
5. Откройте папку Shared, вложенную в папку Views, и удалите все ее содержимое.
6. Выполните двойной щелчок мышью по файлу Global.asax и закомментируйте вызов к `routes.MapRoute`.
7. Чтобы убедиться в том, что внесенные изменения корректны, выполните сборку проекта и запустите его. Выберите из меню команды **Build | Rebuild Solution** и убедитесь в том, что компиляция идет без ошибок. Затем нажмите клавишу <F5>, чтобы запустить приложение и дать VS возможность модифицировать файл Web.config. Так как вы закомментировали маршрут в файле Global.asax, и никаких файлов обнаружено не было, в браузере будет выведено сообщение `The resource can't be found`. Это нормально, поскольку подразумевается, что вы будете строить собственные контроллеры (controllers), модели (models) и представления (views), а также реализуете маршрутизацию самостоятельно.

Таким образом, вы модифицировали проект ASP.NET MVC так, чтобы создавать новые приложения, в состав которых не входит никаких ненужных вам преинсталлированных элементов. Для дальнейших индивидуальных настроек вы можете также заменить CSS-файл в папке Content и добавить ваши собственные библиотеки на JavaScript в папку Scripts. Внесите те изменения, которые вы считаете наиболее подходящими для вас, когда вы начинаете работу над новым проектом ASP.NET MVC. В следующем разделе мы трансформируем этот проект в шаблон, предназначенный для многократного использования.

Экспорт шаблона проекта

После того как вы сконфигурируете проект так, чтобы он в наибольшей степени соответствовал вашим потребностям, нужно сохранить его в качестве шаблона проекта. Выберите из меню команды **File | Export Template**, после чего на экране появится окно **Choose Template Type**, показанное на рис. 12.1. Выберите опцию **Project Template** и нажмите кнопку **Next**.

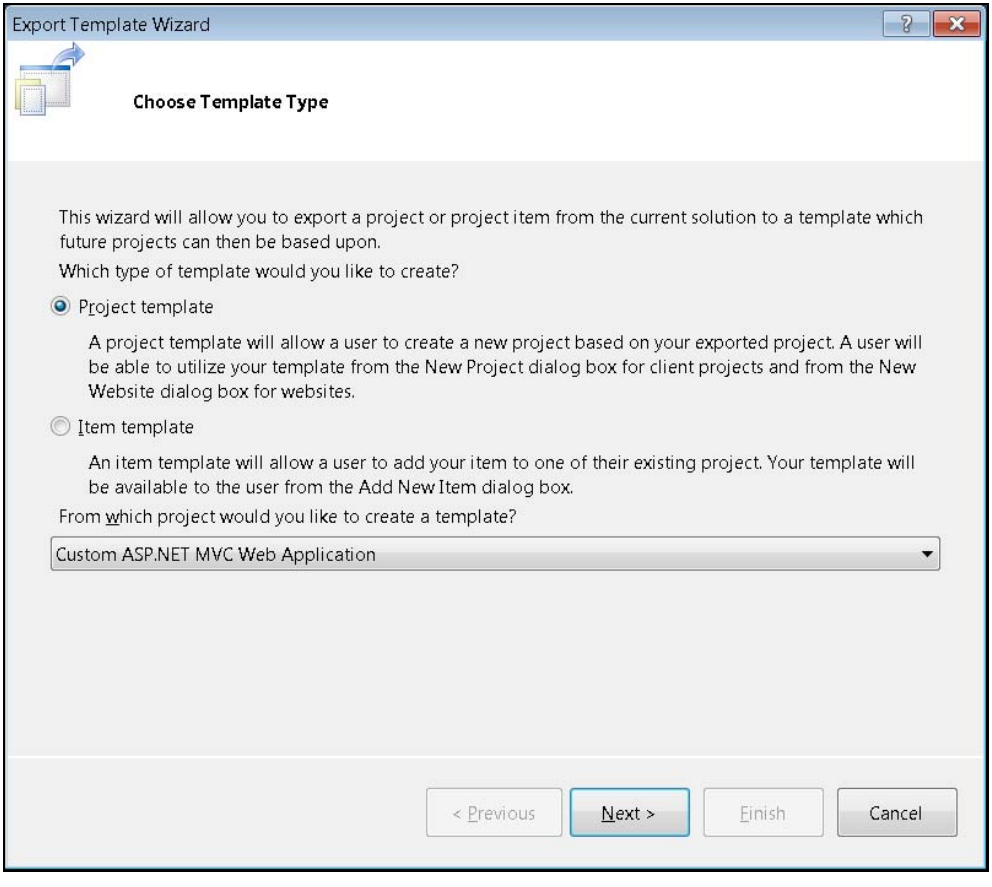


Рис. 12.1. Окно **Choose Template Type**

Следующим появится окно **Select Template Options**, показанное на рис. 12.2. В поле **Template Name** по умолчанию будет указано имя, которое вы присвоили новому проекту. При желании, вы можете изменить это имя. В поле **Template description** введите описание нового шаблона и его предполагаемое применение. Кроме того, если вы хотите ассоциировать с этим шаблоном какой-нибудь значок или картинку для предварительного просмотра (preview), нажмите кнопку **Browse**, расположенную справа от соответствующего поля, и выберите картинку, которую вы хотели бы ассоциировать с проектом. Как вы, разумеется, помните, в окне **New Project**

каждый тип проекта ассоциируется с соответствующим значком, и при выборе данного типа отображаются данные для предварительного просмотра. Опция **Automatically import the template into Visual Studio** сделает шаблон проекта доступным в окне **New Project**. Опция **Display an explorer window on the output files folder** позволит вам получать доступ к новому файлу, отображаемому в папке вывода (Output). Чтобы завершить процедуру экспорта, нажмите кнопку **Finish**.

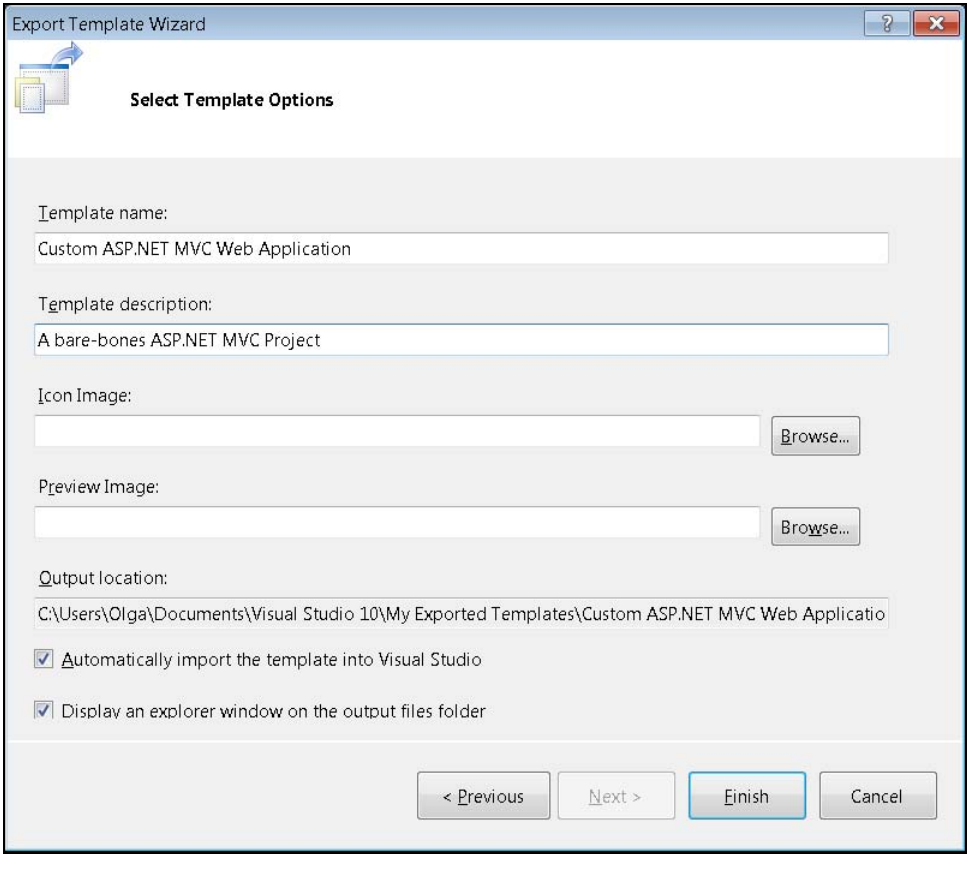


Рис. 12.2. Окно **Select Template Options**

После того как вы нажмете кнопку **Finish**, VS выполнит две задачи: во-первых, шаблон будет сохранен в выводной папке, и, во-вторых, этот шаблон станет доступным в VS. Выводная папка — это просто папка, в которой будет сохранен шаблон проекта, Custom ASP.NET Web Application.zip, содержащий всю информацию, необходимую VS для отображения шаблона и создания проекта на его основе в том случае, если вы выберете этот шаблон в окне **New Projects**.

Кроме того, вы можете предоставить индивидуальный шаблон приложения в общий доступ, чтобы им могли пользоваться другие разработчики. В следующем разделе будет рассказано о том, что следует сделать, чтобы шаблон проекта появился в VS.

Использование нового шаблона проекта

В инструкции по экспорту шаблона проекта, приведенной в предшествующем разделе, рекомендовалось установить опцию **Automatically import the template into Visual Studio**. Использование слова "импорт" может навести вас на мысль о том, что где-то "за кулисами" происходит некоторый магические процесс, и, до определенной степени, это действительно так. Однако все, что делает программа-мастер Export Template Wizard, сводится к копированию файла Custom ASP.NET MVC Web Application.zip из выводной папки в папку <*My Documents*>\Visual Studio 2010\Templates\ProjectTemplates, которая представляет собой локальное хранилище шаблонов проектов. Расположение папки <*My Documents*> может отличаться, в зависимости от той версии Windows, под управлением которой вы работаете. Как только файл появится в локальной папке шаблонов проектов, вы сможете убедиться в том, что он импортирован в VS. Для этого нажмите клавиатурную комбинацию <CTRL>+<SHIFT>+<N> и убедитесь в том, что ваш новый шаблон с именем **Custom ASP.NET MVC Web Application** появился в списке.

Если вы не установили опцию **Automatically import the template into Visual Studio** (см. рис. 12.2), то вы можете просто скопировать файл Custom ASP.NET MVC Web Application.zip в папку, представляющую собой локальное хранилище шаблонов проектов, и этот шаблон станет доступен в VS в окне **New Project**. Если вы предоставите файл Custom ASP.NET MVC Web Application.zip в общий доступ другим разработчикам, они тоже смогут скопировать его в свои локальные папки шаблонов.

Если вы удалите файл из локального хранилища шаблонов проектов, то он больше не появится в окне **New Project.**

Еще один вариант добавления шаблонов проектов заключается в копировании файлов проектов в следующую папку: \Program Files\Microsoft Visual Studio 10.0\Common7\IDE\ProjectTemplates, которая называется глобальным хранилищем шаблонов. В составе этой папки есть различные вложенные папки с именами CSharp, VisualBasic, Web и т. п. Каждая из этих папок соответствует структуре папок, отображаемой в окне **New Project**. В составе каждой из вложенных папок есть еще одна вложенная папка с именем, соответствующим коду набора параметров языковой настройки — например, для английского языка это 1033. Вам нужно скопировать свой файл индивидуально настроенного шаблона в папку с именем набора параметров языковой настройки, вложенную в папку, соответствующую названию категории проектов, к которым должен принадлежать ваш шаблон. Например, если вы хотите, чтобы проект появлялся в окне **New Project** в категории **Visual C# | Web**, скопируйте *.zip-файл проекта шаблона в папку \Program Files\Microsoft Visual Studio 10.0\Common7\IDE\ProjectTemplates\CSharp\Web.

В отличие от локальных шаблонов проектов, где все, что вам требуется — только скопировать файл, шаблоны проектов в глобальном хранилище автоматически не отображаются. Чтобы протестировать сценарий с глобальным хранилищем шаблонов, вам потребуется удалить индивидуальный шаблон проекта из вашей локальной папки шаблонов. Затем вам потребуется закрыть VS и запустить сеанс

командной строки, выбрав из меню следующие команды: **Пуск** (Start) | **Все программы** (All Programs) | **Microsoft Visual Studio 2010** | **Visual Studio Tools**. Наведите курсор на команду **Visual Studio Command Prompt (2010)**, щелкните правой кнопкой мыши и из раскрывшегося контекстного меню выберите команду **Запуск от имени Администратора** (Run As Administrator). Из командной строки дайте следующую команду, чтобы импортировать шаблоны проектов из глобального хранилища:

```
devenv /installvstemplates
```

На завершение работы команды потребуется несколько минут, но после того, как она отработает, вы увидите, что новый шаблон проявился в окне **New Project**. Если впоследствии вы решите, что данный шаблон в окне VS **New Project** больше не нужен, удалите шаблон проекта из глобального хранилища и снова запустите команду `devenv /installvstemplates`.

Теперь, когда вы научились создавать индивидуальные шаблоны проектов и пользоваться ими, давайте обсудим создание индивидуальных элементов проектов. Данная тема обсуждается в следующем разделе.

Создание шаблонов новых элементов

Иногда бывают такие ситуации, когда вы часто пользуетесь шаблонами некоторых элементов проекта, но при этом их содержимое не совсем удовлетворяет вашим потребностям, и вы вынуждены регулярно его менять. В таких случаях вы можете существенно повысить производительность своего труда, создав индивидуальные шаблоны элементов. Пример, приведенный в данном разделе, демонстрирует создание шаблона для перечислений (enums), которого на текущий момент в составе типовых элементов нет. Чтобы создать шаблон нового элемента, сначала необходимо создать файл, содержащий данный элемент, сохранить его, после чего новый шаблон элемента будет доступен для использования в ваших проектах.

Создание шаблона элемента

Простейший способ начать создавать новый шаблон элемента заключается в том, чтобы запустить новый проект, в который входит существующий элемент шаблона, наиболее близкий к тому типу шаблона, который вы хотите создать. Чтобы создать новый шаблон перечисления (enum), нам требуется всего лишь файл класса, поэтому нам подойдет любой тип проекта, к которому можно добавить файл класса. В примере, описываемом в данном разделе, мы создадим проект нового консольного приложения (Console), однако на самом деле тип проекта значения не имеет, поскольку нам из его состава требуется всего лишь единственный файл, на основе которого и будет создан шаблон. Чтобы создать новый шаблон элемента, проделайте следующие шаги:

1. Нажмите клавиатурную комбинацию <CTRL>+<SHIFT>+<N>, чтобы создать новый проект, и в окне **New Project** выберите в качестве типа проекта опцию **Console Application**. Назовите проект любым именем по собственному усмотрению и выберите местоположение для его сохранения. Имя проекта и папка,

в которой он будет сохранен, для наших целей большого значения не имеют, мы заинтересованы только в том, чтобы получить файл шаблона, а не сам проект. Нажмите кнопку **OK**, и новый проект будет создан. VS создаст для проекта новое решение. К настоящему моменту вы уже создали немало консольных приложений, описывавшихся в предыдущих главах. Созданный вами новый проект ничем от них не отличается.

2. Щелкните правой кнопкой мыши по имени проекта в окне **Solution Explorer**, выберите из контекстного меню команды **Add | New Item**, выберите в появившемся окне опцию **Code File**, назовите новый файл Enum.cs (Enum.vb — для VB) и нажмите кнопку **Add**. В результате в состав вашего проекта будет добавлен новый пустой файл.
3. Введите в этот файл код, представленный в листингах 12.1 (для проекта на C#) и 12.2 (для проекта на VB).

Листинг 12.1. Код на C#, предназначенный для создания шаблона элемента типа "перечисление" (enum)

```
/// <summary>
/// Enum description
/// </summary>
public enum MyEnum
{
        /// <summary>
        /// Item 1 description
        /// </summary>
        Item1,

        /// <summary>
        /// Item 2 description
        /// </summary>
        Item2
}
```

Листинг 12.2. Код на VB, предназначенный для создания шаблона элемента типа "перечисление" (enum)

```
''' <summary>
''' Enum description
''' </summary>
Public Enum MyEnum
       ''' <summary>
       ''' Item 1 description
       ''' </summary>
       Item1

       ''' <summary>
```

```
        ''' Item 2 description
        ''' </summary>
        Item2
End Enum
```

4. Сохраните файл.

Итак, теперь вы имеете файл, который может использоваться в качестве "каркаса" для новых элементов проекта типа "перечисление". Остается только экспортировать его в формате шаблона элемента. О том, как это делается, будет рассказано в следующем разделе.

Экспорт шаблона элемента

После того как вы написали каркасный код для нового элемента, необходимо сохранить этот файл в формате шаблона. В качестве первого шага, выберите из меню команды **File | Export Template**, в результате чего на экране появится окно **Choose Template Type**, показанное на рис. 12.3. Выберите в этом окне опцию **Item template** и нажмите кнопку **Next**.

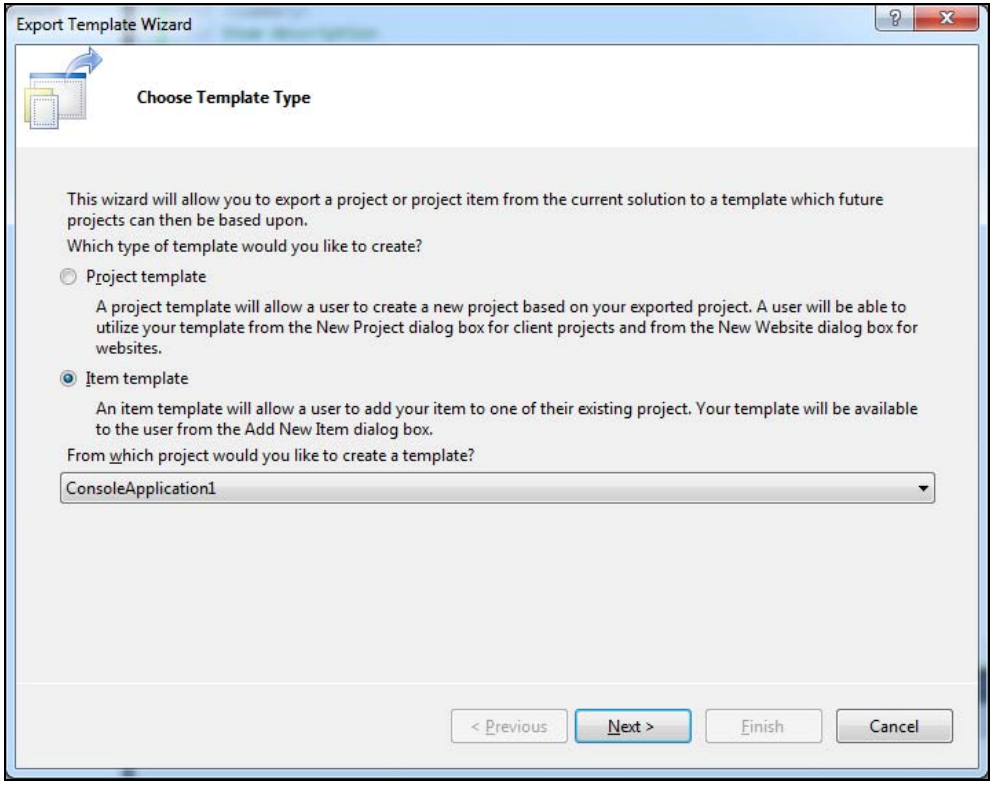


Рис. 12.3. Окно **Choose Template Type**

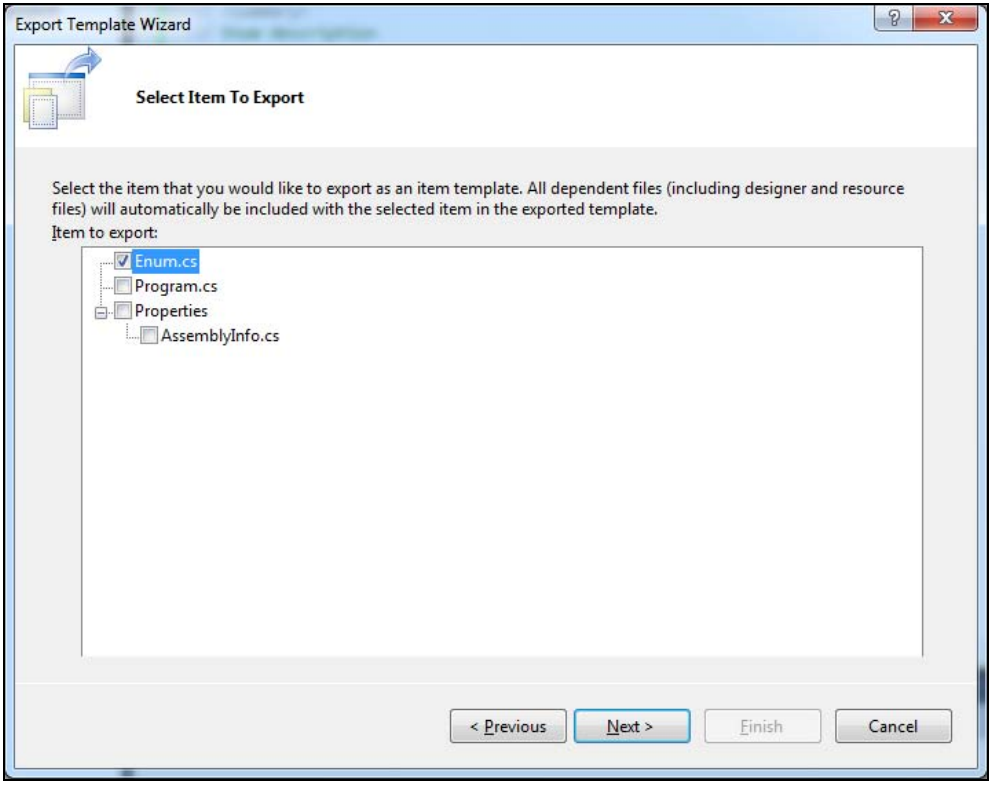


Рис. 12.4. Окно **Select Item To Export**

Далее появится окно **Select Item To Export**, показанное на рис. 12.4. В этом окне содержится список всех файлов, подходящих для создания элемента. Установите флажок для файла Enum.cs, который представляет собой единственный файл, в котором мы заинтересованы в данном примере, и нажмите кнопку **Next.**

Следующим появится окно **Select Item References**, показанное на рис. 12.5. В этом окне перечислены сборки, которые являются частью проекта, из которого вы извлекаете шаблон элемента. Установите флажки, соответствующие сборкам, которые потребуются этому элементу. В рассматриваемом примере нам необходимо гарантировать включение в шаблон сборки **System**. Когда вы выберете эту опцию, непосредственно под полем списка появится предупреждающее сообщение. Проигнорируйте это сообщение. В данном примере подразумевается, что на вашем компьютере в любом случае установлен пакет .NET Framework, и, значит, сборка System.dll будет доступна всегда. Для продолжения нажмите кнопку **Next.**

На рис. 12.6 показано окно **Select Template Options**, в котором вам необходимо указать, как новый шаблон элемента будет представлен в окне **New Items**, которое появляется, когда пользователь выбирает команды **Add** | **New Item**, чтобы добавить в состав проекта новый элемент. По умолчанию поле **Template name** содержит имя проекта, и рекомендуется, чтобы вы изменили это имя на то имя, под которым должен фигурировать шаблон (в нашем примере — Enum).

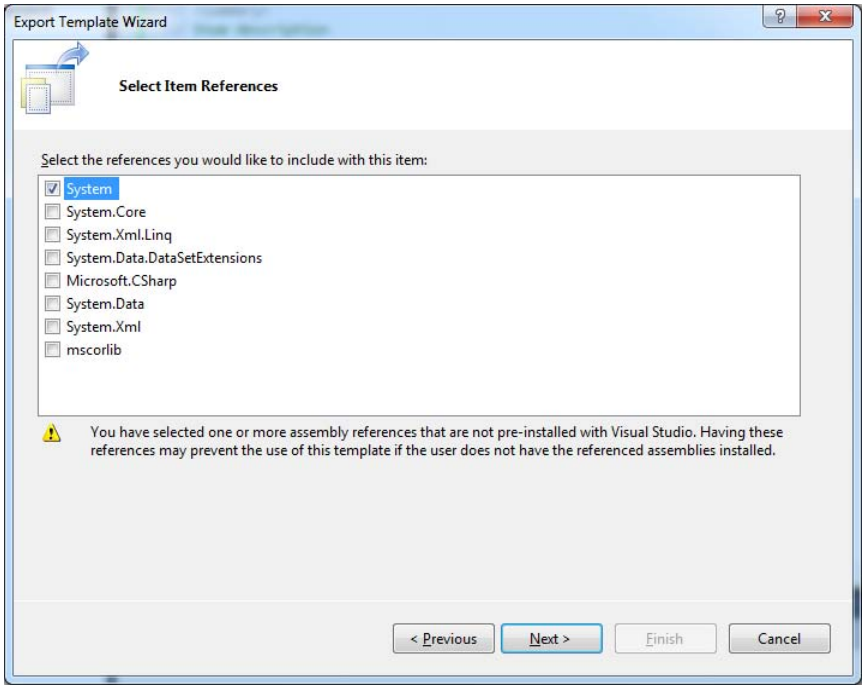


Рис. 12.5. Окно **Select Item References**

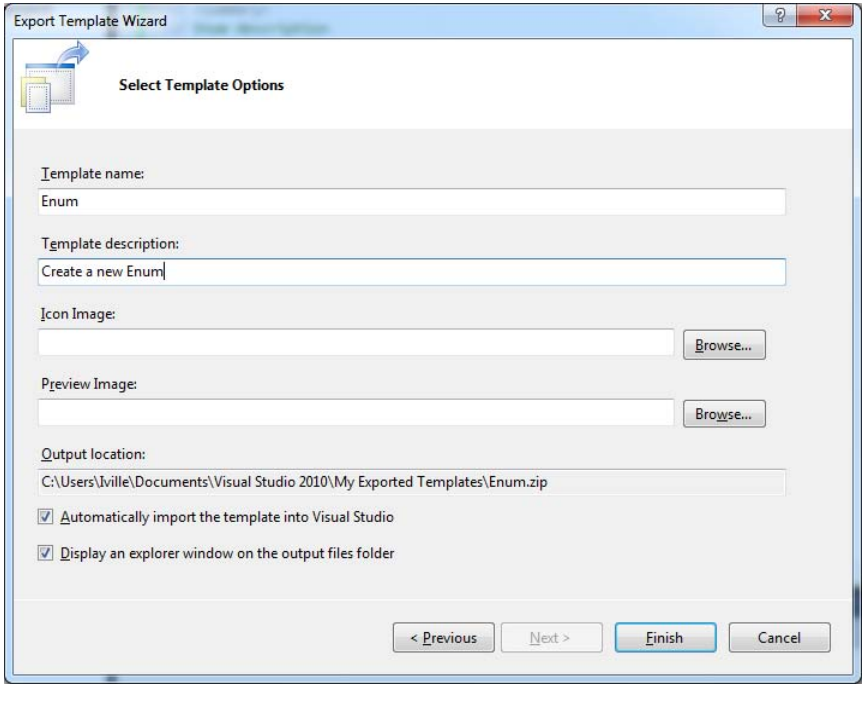


Рис. 12.6. Окно **Select Template Options**

В поле **Template description** рекомендуется ввести строку, которая содержит краткое описание предназначения шаблона элемента. Если вы хотите ассоциировать с шаблоном индивидуальный значок (icon) или изображение для предварительного просмотра (preview), то вы можете нажать кнопки **Browse**, расположенные на правой границе соответствующих полей **Icon Image** и **Preview Image** и выбрать файлы, которые будут ассоциированы с вновь создаваемым шаблоном. Как вы помните, в окне **New Item** с каждым элементом проекта ассоциируется значок, а в правой панели окна отображается изображение для предварительного просмотра проекта. Если вы установите флажок **Automatically import the template into Visual Studio**, то шаблон элемента станет доступен в окне **New Item**. Установка флажка **Display an explorer window on the output files folder** позволит получать доступ к новому файлу, сохраненному в выводную папку, через Windows Explorer.

После того как вы нажмете кнопку **Finish**, VS сохранит файл шаблона элемента в выводной папке и сделает его доступным в VS. Выводная папка — это просто папка, в которой будет сохранен новый шаблон элемента, Enum.zip, который содержит всю информацию, необходимую VS для того, чтобы отобразить шаблон в окне **New Item** и создать на его основе новый элемент проекта. Естественно, новый шаблон можно использовать совместно с другими разработчиками. В следующем разделе будет рассказано о том, что необходимо проделать, чтобы новый шаблон элемента появился в VS.

Использование шаблона элемента

В предшествующем разделе говорилось, что при экспорте шаблона элемента следует установить флажок **Automatically import the template into Visual Studio**, чтобы файл Enum.zip был скопирован из выводной папки в папку <My Documents>\Visual Studio 2010\Templates\ItemTemplates, которая представляет собой локальное хранилище шаблонов элементов проектов. Естественно, расположение папки <My Documents> может меняться, в зависимости от той версии Windows, в которой вы работаете. После того как файл появится в папке локальных элементов шаблонов, вы можете убедиться в том, что он импортирован в VS. Для этого щелкните правой кнопкой мыши по имени проекта в окне **Solution Explorer** (откройте один из существующих проектов или создайте новый), нажмите клавиатурную комбинацию <CTRL>+ +<SHIFT>+<A> и убедитесь, что в списке элементов проекта, доступных для выбора в окне **New Item**, появился элемент **Enum.**

Если вы не установили флажка **Automatically import the template into Visual Studio** (см. рис. 12.6), вы можете вручную скопировать файл Enum.zip в локальное хранилище шаблонов элементов, и шаблон элемента после этого появится в VS. Если вы хотите, вы можете использовать файл Enum.zip совместно с другими разработчиками. Для этого они должны будут скопировать данный файл в свое локальное хранилище шаблонов элементов.

Если вы удалите файл из папки локальных шаблонов элементов, он больше не будет отображаться в окне VS **New Item**. Еще один вариант добавления шаблонов элементов заключается в их копировании в папку \Program Files\Microsoft Visual Studio 10.0\Common7\IDE\ItemTemplates, которая называется глобальным хранилищем шаблонов

элементов. В составе папки, представляющей собой глобальное хранилище шаблонов элементов, имеется структура вложенных папок, включая такие, как CSharp, VisualBasic, Web и т. д. Каждая из этих вложенных папок соответствует папке в окне VS **New Item**. В составе каждой папки имеется еще одна вложенная папка с именем, соответствующим коду языковой настройки (например, для английского языка это будет код 1033), и именно в эту папку вам нужно копировать шаблоны для тех категорий, в составе которых они должны появляться.

В отличие от шаблонов, которые содержатся в локальных папках, шаблоны, находящиеся в глобальных хранилищах, в окне **New Item** автоматически не отображаются. Чтобы протестировать сценарий с глобальными шаблонами, удалите шаблон элемента из вашей папки локальных шаблонов. Затем закройте VS и выберите из меню следующие команды: **Пуск** (Start) | **Все программы** (All Programs) | **Microsoft Visual Studio 2010** | **Visual Studio Tools**. Щелкните правой кнопкой мыши по опции **Visual Studio Command Prompt (2010)** и выберите из контекстного меню команду **Запуск от имени Администратора** (Run As Administrator). Запустите следующую команду:

```
devenv /installvstemplates
```

На завершение работы этой команды может потребоваться несколько минут, но, когда она отработает, новый шаблон элемента появится в окне **New Item**. Если впоследствии вы решите, что какой-то из шаблонов вам больше не нужен, удалите его из папки, представляющей собой глобальное хранилище шаблонов, и дайте команду `devenv /installvstemplates` еще раз.

В данном разделе было продемонстрировано использование шаблонов новых проектов и их элементов. Однако иногда бывают и такие ситуации, когда в процессе программирования вам требуется добавлять довольно типичные и часто повторяющиеся фрагменты кода. Эта задача автоматизируется при помощи автоматически генерируемых фрагментов кода (snippets), которые будут рассматриваться в следующем разделе.

Создание индивидуальных автоматически генерируемых фрагментов кода

Если вы пользуетесь автоматически генерируемыми фрагментами кода VS (snippets), которые описывались в *главе 2*, то вы уже наверняка поняли, как много времени они помогают сэкономить при написании стандартных, широко распространенных и часто повторяющихся блоков кода. Возможно, со временем вам захочется дописать собственные автоматически генерируемые фрагменты для таких конструкций, которые этой возможностью не охвачены. Особенно часто так поступают разработчики C#, заметившие, что для VB автоматически генерируемых фрагментов предоставляется гораздо больше. И даже, если вы работаете на VB, где автоматически генерируемые фрагменты представлены в изобилии, вы все равно можете посчитать, что некоторые блоки кода следует реализовать в виде автоматически генерируемых фрагментов, и повысить свою индивидуальную производительность труда еще больше.

Создание автоматически генерируемого фрагмента кода

В VB уже имеются автоматически генерируемые фрагменты кода для `Sub` и `Function`, а вот в C# их нет. Так как C# не представляет такого большого количества автоматически генерируемых фрагментов кода, как VB, в этом разделе я продемонстрирую процесс создания автоматических фрагментов кода для C#. Впрочем, для VB этот процесс выполняется совершенно аналогично. Чтобы создать новый автоматически генерируемый фрагмент кода, вы можете воспользоваться одним из существующих файлов автоматических фрагментов или начать создание нового файла "с нуля". Сначала давайте рассмотрим процесс создания автоматического фрагмента на основе уже существующего файла.

Исследование существующих автоматических фрагментов

Автоматически генерируемые фрагменты кода (Snippets), поставляющиеся в составе VS, расположены в папке \Program Files\Microsoft Visual Studio 10.0\. В составе этой папки имеются вложенные папки для каждого из поддерживаемых языков (VC#, VB, XML и т. д.), внутри которых и хранятся автоматические фрагменты для соответствующего языка. В этих папках вы найдете вложенную папку Snippets, а в ней — одну или несколько папок, имена которых совпадают с кодами параметров языковой настройки (например, 1033 — для английского языка). Для некоторых языков папки с параметрами языковой настройки могут находиться на более высоком уровне иерархии, чем папки Snippets и, соответственно, наоборот. Вне зависимости от этого, вам нужно найти папку Snippets для нужного вам языка, в которой вы найдете файлы с расширениями .snippet. Для C# путь к нужной папке будет выглядеть следующим образом: \Program Files\Microsoft Visual Studio 10.0\VC#\Snippets\1033. Кроме того, в составе самой папки Snippets будут содержаться и другие вложенные папки, которые служат для классификации автоматических фрагментов и их распределения по категориям.

Давайте раскроем один из файлов с расширением имени файла .snippet, поскольку в его составе есть ряд конструкций, которые позволят вам быстро разобраться с тем, как же именно работают автоматические фрагменты кода. Возможно, проще всего будет открыть пустой файл, нажав клавиатурную комбинацию <CTRL>+<N>, выбрав в появившемся окне опцию **Visual C# Class**, дать новому файлу любое имя по своему усмотрению, а затем воспользоваться автоматическим фрагментом `for`, прежде чем двигаться куда-либо далее. Как вариант, вы можете вернуться к *главе 2* и просмотреть описание автоматического фрагмента `for`.

Расширение имени файла .snippet зарегистрировано как ассоциирующееся с VS, поэтому вы можете выполнить двойной щелчок мышью по одному из файлов с расширением .snippet, расположенных в папке Snippets, чтобы открыть этот файл в окне редактора кода VS. В листинге 12.3 показано содержимое файла автоматически генерируемого фрагмента для цикла `for`.

Листинг 12.3. Автоматически генерируемый фрагмент кода для цикла for

```
<?xml version="1.0" encoding="utf-8" ?>
<CodeSnippets  xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
        <CodeSnippet Format="1.0.0">
                <Header>
                        <Title>for</Title>
                        <Shortcut>for</Shortcut>
                        <Description>Code snippet for 'for' loop</Description>
                        <Author>Microsoft Corporation</Author>
                        <SnippetTypes>
                                <SnippetType>Expansion</SnippetType>
                                <SnippetType>SurroundsWith</SnippetType>
                        </SnippetTypes>
                </Header>
                <Snippet>
                        <Declarations>
                                <Literal>
                                        <ID>index</ID>
                                        <Default>i</Default>
                                        <ToolTip>Index</ToolTip>
                                </Literal>
                                <Literal>
                                        <ID>max</ID>
                                        <Default>length</Default>
                                        <ToolTip>Max length</ToolTip>
                                </Literal>
                        </Declarations>
                        <Code Language="csharp"><![CDATA[for (int $index$ = 0; $index$
                        < $max$; $index$++)
                        {
                        $selected$ $end$
                        }]]>
                        </Code>
                </Snippet>
        </CodeSnippet>
</CodeSnippets>
```

Как показано в листинге 12.3, автоматический фрагмент представляет собой файл формата XML, в котором все данные определяются открывающими и закрывающими тэгами, организованными в иерархическую структуру. Внутри тэгов `CodeSnippet` находятся элементы `Header` и `Snippet`.

Внутри элемента `Header` имеется элемент `Shortcut`, который определяет префикс, который вам необходимо ввести, чтобы редактор кода VS мог использовать автоматически генерируемый фрагмент. Тэги `Title` и `Description` определяют, что

отображает функция Intellisense при определенной последовательности нажатий клавиш. Тэг `Author` указывает, кто написал автоматический фрагмент.

Элемент `SnippetTypes` определяет два способа использования автоматического фрагмента: расширение (`Expansion`) и обтекание кода вокруг выделенного фрагмента (`SurroundsWith`). В *главе 2* было описано большое количество автоматических фрагментов, работающих по принципу расширения (Expansion). Однако автоматические фрагменты, работающие по методу обтекания (SurroundsWith) тоже очень полезны. Чтобы использовать автоматический фрагмент, работающий по принципу обтекания (SurroundsWith), выделите фрагмент кода, вокруг которого необходимо расположить обтекающий код, нажмите клавиатурную комбинацию <CTRL>+<SPACE> и затем выберите автоматический фрагмент. После того как фрагмент будет выбран, в редакторе VS появится шаблон фрагмента, блоки которого окружают выделенный текст. Так как цикл `for` имеет блок, который может содержать выражения, имеет смысл реализовать для фрагмента как обтекание (SurroundsWith), так и расширение (Expansion).

Элемент `Snippet`, приведенный в листинге 12.3, содержит элементы `Declarations` и `Code`, причем декларации используются в коде. При обсуждении работы шаблонов автоматических фрагментов не забывайте, что курсор размещается в блоках кода, которые вы редактируете, и перемещение между этими блоками осуществляется нажатием клавиши <TAB>. Блоки кода, которые подлежат заполнению, соответствуют элементам `Literal` в декларации.

Каждый элемент `Literal` имеет `ID`, который используется в элементе `Code`, чтобы определить, где располагается элемент `Literal`. Элемент `Default` описывает данные, которые шаблон отображает по умолчанию, до того, как вы начнете ввод с клавиатуры. Каждый раз, когда вы заполняете шаблон автоматического фрагмента, вы можете задержать курсор мыши над полем данных, в результате чего на экране появится всплывающая подсказка, описывающая, какую информацию следует вводить в поле данных. Эта всплывающая подсказка определяется элементом `Tooltip` в определении автоматического фрагмента. `ID` для каждого литерала определяется в элементе `Code`.

Внутри элемента `Code` располагается код автоматического фрагмента. Переменные в коде с префиксами и суффиксами в виде знака доллара (`$`) помогают определять, как работает шаблон автоматического фрагмента. Обратите внимание, что переменные `$index$` и `$max$` соответствуют элементам `Literal` в составе элемента `Declarations`; именно здесь и необходимо вводить элементы данных при заполнении шаблона автоматически генерируемого фрагмента в VS. Переменная `$end$` определяет, где оказывается курсор после того, как заполнение шаблона для автоматического фрагмента будет закончено (т. е. после нажатия клавиши <ENTER> в шаблоне автоматического фрагмента). Вам необходимо разместить переменную `$end$` там, где разработчику обычно хотелось бы продолжать ввод с клавиатуры. Переменная `$selected$` используется с автоматическими фрагментами, работающими по принципу обтекания (`SurroundsWith`), определяя отношения между выделенным текстом и расположением кода автоматического фрагмента.

Теперь, когда вы в общих чертах познакомились с автоматическими фрагментами, необходимо научиться создавать индивидуальные фрагменты, определенные в соответствии с вашими потребностями.

Написание новых автоматических фрагментов

Чтобы создать новый автоматический фрагмент, вы можете либо начать работу с модификации одного из существующих автоматических фрагментов или начать свою работу с нуля, создав пустой файл. Если вы хотите работать с существующим автоматическим фрагментом, найдите и откройте тот файл с расширением .snippet, который ближе всего соответствует тому фрагменту, который вы хотите создать. О том, как это делается, было рассказано в предшествующем разделе. Если вы решили начать всю работу "с чистого листа", то и для этого есть автоматический фрагмент, позволяющий вам быстро создавать новые автоматические фрагменты.

Как уже говорилось в предшествующем разделе, автоматические фрагменты представляют собой XML-файлы. К счастью, VS предоставляет удобный встроенный редактор XML, который поддерживает автоматические фрагменты XML. Таким образом, когда говорится о создании автоматического фрагмента "с нуля", это не совсем соответствует действительности, поскольку даже здесь VS позволяет автоматизировать и упростить задачу. Следующая пошаговая процедура демонстрирует, каким образом можно очень быстро создать автоматический фрагмент кода, который можно использовать для добавления метода C# в состав класса.

1. Запустите VS и нажмите клавиатурную комбинацию <CTRL>+<N>, чтобы создать новый XML-файл. Если вы открываете файл из одного из существующих проектов, вам потребуется ввести имя файла. Назовите файл так: meth.snippet. Новый XML-файл состоит из единственной строки, которая называется префиксом XML.
2. Перейдите в следующую строку за префиксом XML, нажмите клавиатурную комбинацию <CTRL>+<K>+<X>, введите с клавиатуры последовательность символов **sn**, а затем из списка, отображенного функцией Intellisense, выберите элемент **Snippet** и нажмите клавишу <ENTER>. Вы увидите шаблон XML для создания автоматического фрагмента с полями `Title`, `Author`, `Shortcut`, `Description`, `ID` и `Default` (рис. 12.7).
3. Введите данные, перемещаясь между полями шаблона нажатиями клавиши <TAB> следующим образом: в поле `Title` введите значение `Method Snippet`, в поле `Author` введите ваше имя, в поле `Shortcut` — строку `meth`, в поле `Description` — строку `Create a New Method`, в поле `ID` — строку `access`, а в поле `Default` — строку `public`. Закончив ввод информации, нажмите клавишу <ENTER>.
4. Результирующий шаблон все еще нуждается в дополнительном вводе кода и определений элементов шаблона. Для этого заполните элемент `Code` и добавьте элементы `Literal`. Сначала модифицируйте элемент `Code` так, как показано в листинге 12.4.

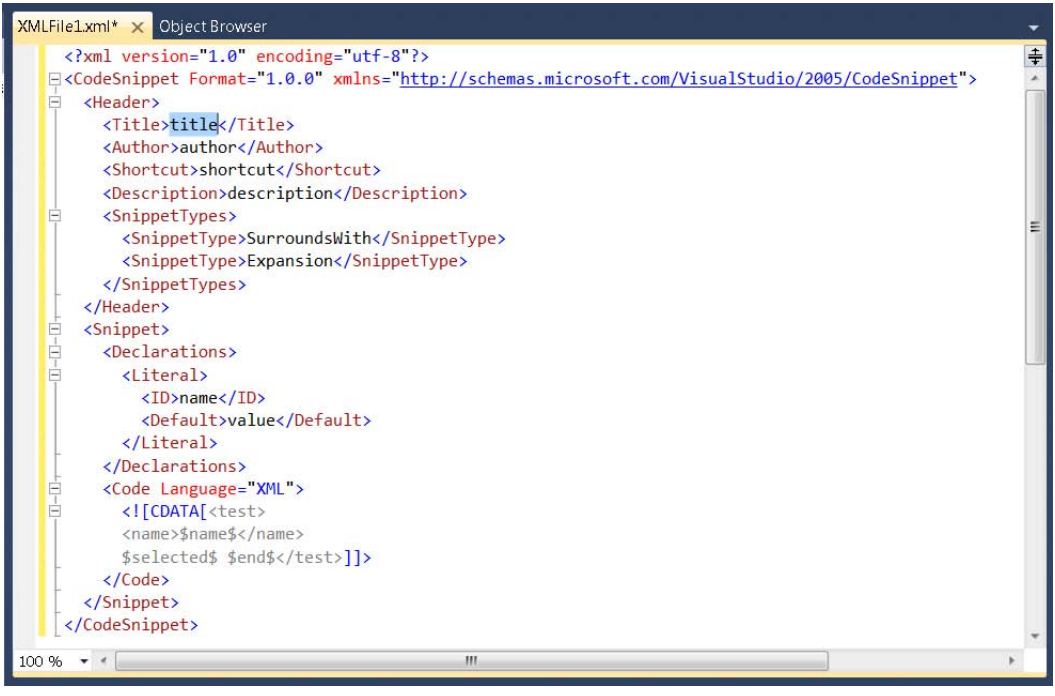


Рис. 12.7. Шаблон XML для создания автоматического фрагмента

Листинг 12.4. Модификации, которые необходимо внести в элемент Code шаблона автоматического фрагмента

```
<Code Language="csharp">
    <![CDATA[$access$ $return$ $methodName$($paramList$)
    {
            $end$
    }
    ]]>
</Code>
```

5. Кроме элемента `access`, код, приведенный в листинге 12.4, содержит переменные `return`, `methodName` и `paramList`. Для каждой из этих переменных добавьте элементы `Literal`, где `ID` представляет собой имя каждой из переменных, элемент `Default` для `return` установлен на значение `void`, `methodName` — на `MethodName`, а `paramList` — на `int p1`.
6. Сохраните файл и назовите его meth.snippet. В следующем разделе будет рассказано о том, куда поместить этот файл, чтобы новый автоматический фрагмент был доступен для использования, а пока просто поместите его в любую папку, где вы впоследствии легко сможете его найти. Между прочим, в диалоговом окне **Save File** для поля **Save A Type** имеется опция **Snippet Files (*.snippet)**, кото-

рую вам необходимо выбрать с тем, чтобы гарантировать, что сохраненный файл автоматического фрагмента имеет корректное расширение имени файла.

Теперь у вас имеется работоспособный автоматический фрагмент. Его полный код приведен в листинге 12.5. Кроме того, обратите внимание, что для каждого элемента `Literal` имеется атрибут `Tooltip`, призванный помочь пользователю правильно заполнить все поля автоматического фрагмента. Заметим также, что атрибут `Language` элемента `Code` имеет значение `csharp`, а не `C#`. Хотя, на первый взгляд, это может показаться незначительными нюансами, но все эти несущественные мелочи могут сделать файл автоматического фрагмента некорректным. Наилучшим подходом к устранению неполадок будет открытие похожего автоматического фрагмента и изучение его формата с тем, чтобы убедиться, не допустили ли вы где-то синтаксических ошибок и опечаток. В следующем разделе будет рассказано о том, что следует делать дальше с файлом автоматического фрагмента, чтобы его можно было начать использовать.

Листинг 12.5. Индивидуально заданный автоматический фрагмент для создания метода на C#

```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippet Format="1.0.0"
xmlns="http://schemas.microsoft.com
/VisualStudio/2005/CodeSnippet">
    <Header>
        <Title>Method Snippet</Title>
        <Author>Joe Mayo</Author>
        <Shortcut>meth</Shortcut>
        <Description>Create a New Method</Description>
        <SnippetTypes>
            <SnippetType>SurroundsWith</SnippetType>
            <SnippetType>Expansion</SnippetType>
        </SnippetTypes>
    </Header>
    <Snippet>
        <Declarations>
            <Literal>
                <ID>access</ID>
                <Default>public</Default>
                <ToolTip>Access modifier</ToolTip>
            </Literal>
            <Literal>
                <ID>return</ID>
                <Default>void</Default>
                <ToolTip>Return value</ToolTip>
            </Literal>
            <Literal>
```

```
            <ID>methodName</ID>
            <Default>MethodName</Default>
            <ToolTip>Name of Method</ToolTip>
        </Literal>
        <Literal>
            <ID>paramList</ID>
            <Default>int p1</Default>
            <ToolTip>
                Comma-separated list of parameters
            </ToolTip>
        </Literal>
    </Declarations>
    <Code Language="csharp">
        <![CDATA[
            $access$ $return$ $methodName$($paramList$)
            {
                    $end$
            }]]>
    </Code>
  </Snippet>
</CodeSnippet>
```

Управление библиотекой автоматических фрагментов

Чтобы иметь возможность пользоваться вновь созданным автоматическим фрагментом, его необходимо скопировать либо в одну из предназначенных для хранения автоматических фрагментов папок VS, либо воспользоваться инструментом VS, который называется диспетчером сниппетов (Code Snippets Manager). В данном разделе будет рассказано о том, как пользоваться новым автоматическим фрагментом, который вы создали в предыдущем разделе, и как, при желании, предоставить его в общий доступ.

Управление библиотекой автоматических фрагментов

Локальные автоматические фрагменты располагаются в папке \Users\<your name>\Documents\Visual Studio 2010\Code Snippets. В этой папке существует вложенная иерархическая структура папок с автоматическими фрагментами для всех поддерживаемых языков и технологий, причем в этих папках тоже могут быть вложенные папки, помогающие организовать автоматические фрагменты (snippets). Скопируйте свой новый автоматический фрагмент (файл с расширением .snippet) в одну из этих папок, например, Visual C#\My Code Snippets, и ваш вновь созданный автоматический фрагмент сразу же станет доступным для использования.

Папка, являющаяся локальным хранилищем автоматических фрагментов, делает его доступным вам (когда вы регистрируетесь в системе от своего имени). Кроме того, при желании вы можете сделать свой автоматический фрагмент доступным

каждому из пользователей, регистрирующихся на вашем компьютере. Для этого его нужно скопировать в папку, представляющую собой глобальное хранилище автоматических фрагментов, путь к которой выглядит так: \Program Files\Microsoft Visual Studio 10.0\. В составе этой папки тоже есть вложенная иерархическая структура папок для всех поддерживаемых языков и технологий, например, VC# — для C# или VB — для VB. Внутри этих папок вы увидите или папки с именами, соответствующими кодам национальных языков (например, для английского языка это будет код 1033) или папку Snippets. Пролистайте всю эту вложенную структуру, и вы увидите множество файлов автоматических фрагментов с расширением .snippet и дополнительные вложенные папки, помогающие в организации автоматических фрагментов. Скопируйте файл в одну из этих папок, и он сразу же станет доступен не только вам, но и другим пользователям.

Работа с этой сложной иерархической структурой папок может быть достаточно неудобной, поэтому VS предлагает специальный инструмент, помогающий организовать ваши автоматические фрагменты — Диспетчер сниппетов (Snippets Manager).

Использование инструмента Code Snippets Manager

Инструмент Code Snippets Manager позволяет вам импортировать новые автоматические фрагменты и оптимальным образом организовать их хранение. Чтобы воспользоваться этим инструментом, выберите из меню команды **Tools | Code Snippets Manager** или нажмите клавиатурную комбинацию <CTRL>+<K>, <CTRL>+<B>. На экране появится окно **Code Snippets Manager**, показанное на рис. 12.8.

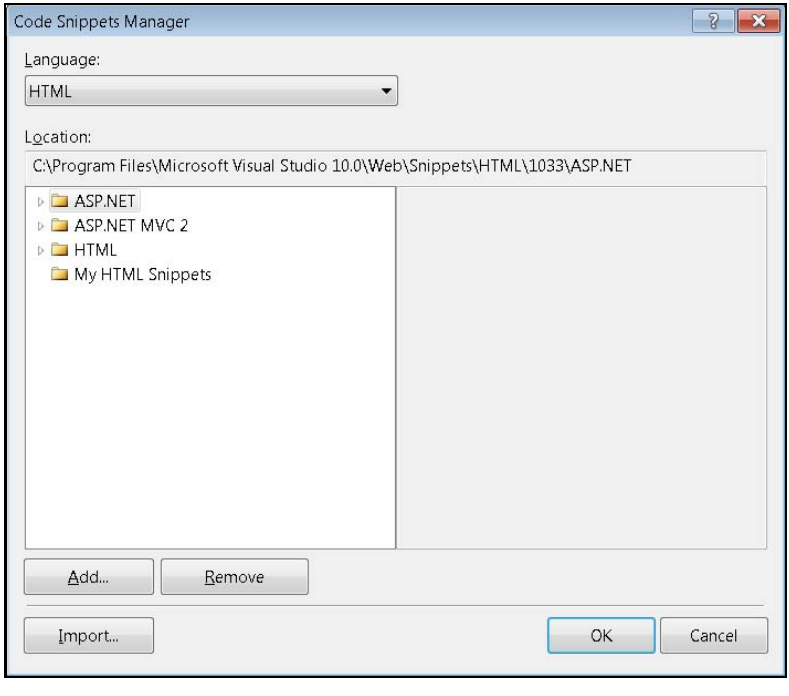


Рис. 12.8. Окно **Code Snippets Manager**

Раскрывающийся список **Language** позволяет выбирать из списка типы автоматических фрагментов кода, с которыми вы можете работать. В расположенном ниже поле **Location** показано, как организованы автоматические фрагменты. Кнопки **Add** и **Remove** позволяют вам управлять папками. Если вы хотите сделать доступными новые автоматические фрагменты, воспользуйтесь кнопкой **Import.**

Как видите, автоматические фрагменты серьезно упрощают процедуру быстрого написания типового кода. Однако в VS есть и еще более мощная возможность — макросы (macros), к обсуждению которых мы приступим в следующем разделе.

Написание макросов

Если вам не хватает тех средств повышения индивидуальной производительности труда, которые предоставляют вам индивидуально разработанные автоматические фрагменты кода (custom snippets), вам следует сделать еще один шаг вперед и начать пользоваться макросами. Макрос (macro) — это повторяющаяся последовательность действий, которую можно записать, и затем повторно воспроизводить столько раз, сколько потребуется. Макросы окажутся очень полезны в тех ситуациях, когда вы, работая в VS, многократно повторяете одну и ту же последовательность действий. В этом разделе будет показано, как создать макрос и как его запустить, на примере написания блока кода для проверки (валидизации) строк.

Запись макроса

При создании бизнес-объектов обычной и общепринятой практикой является проверка (валидизация) входных параметров с тем, чтобы убедиться в их корректности и допустимости. Одной из проверок является принудительная передача параметра вызывающим кодом. Пример, приведенный в этом разделе, демонстрирует написание макроса, проверяющего, не получают ли передаваемые параметры строкового типа значений `null`, не являются ли они пустыми строками или символами пробела/табуляции.

Начнем с создания нового проекта консольного приложения и добавим к нему новый файл класса, содержащий метод, имитирующий добавление в базу данных записи для нового клиента. Код данного метода на C# и VB приведен в листингах 12.6 и 12.7, соответственно.

Листинг 12.6. Код метода на C#, имитирующий добавление в базу данных записи для нового клиента

```
using System;

class Customer
{
        public int AddNewCustomer(string firstName, string lastName)
        {
                int newCustID = 0;
```

```
                // Logic to add customer

                return newCustID;
        }
}
```

Листинг 12.7. Код метода на VB, имитирующий добавление в базу данных записи для нового клиента

```
Public Class Customer

    Function AddNewCustomer(
        ByVal firstName As String,
        ByVal lastName As String) As Integer

        Dim newCustID As Integer = 0

        ' Logic to add customer

        Return newCustID

    End Function

End Class
```

В данном случае в методе `AddNewCustomer` нас интересуют параметры `firstName` и `lastName`. Каждый раз при работе с данными вам обычно требуется убедиться в том, что вводимая информация корректна и допустима. При обработке пользовательского ввода прием некорректной информации — это обычное дело, даже если в вашем пользовательском интерфейсе реализована валидизация пользовательского ввода. Например, код, приведенный в листингах 12.8 и 12.9, вызывает метод `AddNewCustomer`, передавая ему передачу некорректных аргументов.

Листинг 12.8. Код на C#, демонстрирующий передачу методу некорректных параметров

```
class Program
{
        static void Main()
        {
                string firstName = "Joe";
                string lastName = null;

                Customer cust = new Customer();
                cust.AddNewCustomer(firstName, lastName);
        }
}
```

Листинг 12.9. Код на VB, демонстрирующий передачу методу некорректных параметров

```
Module Module1

    Sub Main()

        Dim firstName As String = "Joe"
        Dim lastName As String = Nothing

        Dim cust As New Customer
        cust.AddNewCustomer(firstName, lastName)

    End Sub
End Module
```

В примере, приведенном в листингах 12.8 и 12.9, параметр `firstName` корректен, потому что содержит допустимое имя. Однако обратите внимание, что параметр `lastName` получает значение `null` (`Nothing` — в VB). Это вызовет исключение `NullReferenceException`, когда метод `AddNewCustomer` попытается выполнить строковую операцию над параметром. Код, который вызывает метод `AddNewCustomer`, потенциально может вызвать исключение `NullReferenceException` или (в предположении того, что значение `null` будет воспринято как некорректное) вы просто сохраните некорректные данные.

Так как `AddNewCustomer` не имеет реализации, все сказанное относится к области догадок, но зато позволяет предположить, какие проблемы могут возникнуть, если вы дадите программе возможность принять некорректные данные.

Макрос, который будет продемонстрирован в этом разделе, продемонстрирует выполнение проверки строковых параметров, чтобы исключить ввод таких значений, как `null`, пустые строки или пробелы, за счет выбрасывания исключения `ArgumentNullException`. Это не даст вызывающему коду передать некорректные данные и выведет для пользователя поясняющее сообщение, почему так происходит. Чтобы создать макрос, найдите в коде позицию, где должен запускаться макрос (там, где это уместно и возможно), запустите запись макроса, выполните нужные действия в среде VS, а затем остановите запись. В чем-то этот алгоритм напоминает запись телепередачи с помощью видеорекордера, когда вы запускаете процесс записи, просматриваете передачу до определенного момента, а затем останавливаете запись. Для записи последовательности действий, выполняющих проверку передаваемых параметров и сохранения ее в качестве макроса, проделайте следующее:

1. Щелкните мышью по параметру `firstName` в методе `AddNewCustomer`, чтобы курсор находился в пределах идентификатора параметра `firstName`. Это важно, поскольку нам требуется имя параметра в нашем коде.
2. Начните запись макроса, выбрав из меню команды **Tools** | **Macros** | **Record TemporaryMacro** или нажав клавиатурную комбинацию <CTRL>+<SHIFT>+<R>.

3. Если вы работаете на C#, нажмите следующую последовательность клавиатурных комбинаций: <CTRL>+<←>, <CTRL>+<SHIFT>+<→> и <CTRL>+<C>. Для VB нажмите следующие клавиатурные комбинации: <CTRL>+<←>, <CTRL>+ +<SHIFT>+<→>, <SHIFT>+<←> и <CTRL>+<C>. Таким образом, вы скопируете имя параметра.
4. Для C#, нажмите клавиатурную комбинацию <CTRL>+<F>, чтобы открыть окно **Find And Replace**, введите в поле **Find What** символ **{**, щелкните мышью по кнопке **Find Next**, закройте окно **Find And Replace**, нажмите клавишу <END> и нажмите клавишу <ENTER>. Для VB нажмите клавишу <END> и нажмите клавишу <ENTER>. Это переместит курсор в начальную позицию для ввода кода.
5. Введите с клавиатуры символы **if** и дважды нажмите клавишу <TAB> (автоматически генерируемый фрагмент кода `if`), затем введите в поле условия строку **string.IsNullOrWhiteSpace**, нажмите клавиатурную комбинацию <CTRL>+<V>, чтобы вставить имя параметра в качестве аргумента, затем снова введите с клавиатуры символ **)**. Для C#, нажмите клавишу <ENTER>. Для VB, нажмите клавишу <↓>. Курсор переместится в тело утверждения **if** (в точности так же, как и следовало бы ожидать при работе с автоматическим фрагментом утверждения `if`). Это позволяет задать проверку корректности для параметра, чтобы убедиться в том, что он не получает значения `null` (`Nothing` — в VB), не является пустой строкой или символом пробела либо табуляции.
6. Введите с клавиатуры следующую строку: **throw new ArgumentNullException("**, затем нажмите клавиатурную комбинацию <CTRL>+<V>, чтобы вставить имя параметра, введите с клавиатуры символы **", "**, снова нажмите клавиатурную комбинацию <CTRL>+<V>, чтобы вставить имя параметра, нажмите клавишу пробела и введите заключительную строку **value is not valid.")**. Для C#, введите символ точки с запятой, **;**, чтобы завершить строку. Эти действия будут выполнены в том случае, если в качестве параметра передано недопустимое значение: будет выброшено исключение и выведено сообщение о том, что введенное значение некорректно.
7. Нажмите клавишу <↓>, затем нажмите клавишу <ENTER>. Курсор переместится за пределы кода, что удобно в тех случаях, если вы хотите продолжать ввод с этой позиции.
8. Выберите из меню команды **Tools** | **Macros** | **Stop Recording TemporaryMacro** или нажмите клавиатурную комбинацию <CTRL>+<SHIFT>+<R>, чтобы завершить запись.

Итак, вы записали макрос. Чтобы проверить правильность выполнения действий, просмотрите модифицированный код метода `AddNewCustomer`, который теперь должен выглядеть так, как показано в листингах 12.10 и 12.11.

Листинг 12.10. Модифицированный код метода AddNewCustomer после записи макроса (код на C#)

```
using System;


class Customer
```

```
{
        public int AddNewCustomer(string firstName, string lastName)
        {
                if (string.IsNullOrWhiteSpace(firstName))
                {
                        throw new ArgumentNullException(
                                "firstName",
                                "firstName value is not valid.");
                }

                int newCustID = 0;

                // Logic to add customer

                return newCustID;
        }
}
```

Листинг 12.11. Модифицированный код метода AddNewCustomer после записи макроса (код на VB)

```
Public Class Customer

    Function AddNewCustomer(
        ByVal firstName As String,
        ByVal lastName As String) As Integer

        If String.IsNullOrWhiteSpace(firstName) Then
                Throw New ArgumentNullException(
                        "firstName",
                        "firstName value is not valid.")
        End If

        Dim newCustID As Integer = 0

        ' Logic to add customer

        Return newCustID

    End Function

End Class
```

В коде, приведенном в листингах 12.10 и 12.11, аргументы `ArgumentNullException` разделены по отдельным строкам в целях соответствия форматированию, на самом же деле это утверждение должно вводиться одной строкой. Далее, нам следует

протестировать наш макрос, запустив его. Щелкните мышью по параметру `lastName` и выберите из меню команды **Tools** | **Macros** | **Run TemporaryMacro** или нажмите клавиатурную комбинацию <CTRL>+<SHIFT>+<P>. В результате этого вы получите код, показанный в листинге 12.12.

Листинг 12.12. Код, полученный в результате запуска макроса

```
public int AddNewCustomer(string firstName, string lastName)
{
        if (string.IsNullOrWhiteSpace(lastName))
        {
            throw new ArgumentException("lastName", "lastName value is not
                                    valid.");
        }
        if (string.IsNullOrWhiteSpace(firstName))
        {
            throw new ArgumentException("firstName", "firstName value is
                                    not valid.");
        }

        int newCustID = 0;

        // Logic to add customer

        return newCustID;
}
```

Теперь вы можете запускать этот макрос для каждого из строковых параметров вашего класса и быстро выполнять проверку корректности их ввода. Единственной проблемой на данный момент является то, что наш макрос будет немедленно перезаписан, как только вы начнете запись нового макроса. Кроме того, если вы выйдете из VS, то макрос сохранен не будет. Поэтому в следующем разделе мы обсудим операции, необходимые для сохранения макроса.

Сохранение макроса

Чтобы иметь возможность повторного использования макросов в следующих сеансах, их необходимо сохранять. Чтобы сохранить только что записанный макрос, выберите из меню команды **Tools** | **Macros** | **Save TemporaryMacro**. VS сохранит макрос **TemporaryMacro** и раскроет окно **Macro Explorer**, показанное на рис. 12.9.

VS использует имя **TemporaryMacro** для всех записываемых макросов. Поэтому, если вы хотите сохранить его для последующего использования, макрос необходимо переименовать. Переименуйте макрос, дав ему, например, имя **ValidateStringParameter**. Для этого щелкните правой кнопкой мыши по макросу **TemporaryMacro** в окне **Macro Explorer** и из контекстного меню выберите команду **Rename**.

В окне **Macro Explorer** вы можете добавлять новые проекты макросов (Macro Projects), которые представляют собой контейнеры для хранения модулей макросов. Для этой цели выполните щелчок по строке **Macros** и выберите из контекстного меню команду **New Macro Project**. Если ваши коллеги предоставляют в ваше распоряжение свои проекты макросов, щелкните правой кнопкой мыши по строке **Macros** и выберите из контекстного меню команду **Load Macro Project**, затем найдите в нужных папках файловой системы проект и загрузите его. Модули макросов содержат отдельные макросы, и вы можете открывать контекстное меню для любого проекта макроса, например, **MyMacros** или **Samples** (см. рис. 12.9) и выбрать из него команду **New Module** для добавления новых модулей. Конечно, доступ ко всем этим командам доступен и через команды меню **Tools | Macros**.

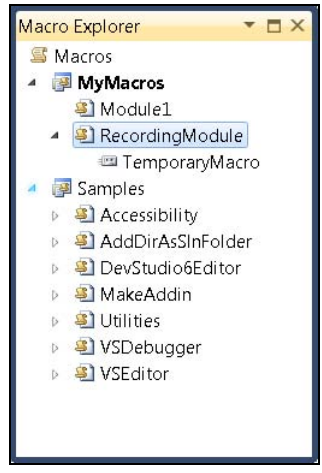


Рис. 12.9. Окно **Macro Explorer**

Чтобы запустить один из имеющихся у вас макросов, выполните по нему двойной щелчок мышью в окне **Macro Explorer**.

Чтобы модифицировать макрос, вы можете либо перезаписать его, либо отредактировать его код. О редактировании макросов рассказывается в следующем разделе.

Редактирование макросов

Макросы содержат редактируемый код, за счет чего вы можете модифицировать ранее записанные макросы или писать собственные практически "с нуля". Чтобы отредактировать макрос, щелкните по его имени правой кнопкой мыши в окне **Macro Explorer** и из раскрывшегося контекстного меню выберите команду **Edit**. На экране появится окно редактора макросов, показанное на рис. 12.10. В рассматриваемом примере это окно содержит код макроса **ValidateStringParameter**, запись которого была произведена в предшествующем разделе.

Как показано на рис. 12.10, редактор открывает макрос в окне редактирования кода. Языком программирования макросов является VB, так что, если обычно вы предпочитаете программировать на C#, я рекомендую вам вернуться к *главам 2 — 4* и перечитать те их части, которые относятся к VB. Функции редактора макросов во многом очень похожи на стандартные функциональные возможности интегрированной среды разработки VS, с тем лишь исключением, что теперь вы работаете с проектами макросов и модулями макросов. В листингах 12.13 и 12.14 приведен код макроса **ValidateStringParameter**. Обратите внимание, что макросы, как для C#, так и для VB, пишутся на VB. Однако код макроса для C#, написанный на VB, работает в среде кода C#, а макрос для VB, написанный на VB, предназначен для работы с кодом VB.

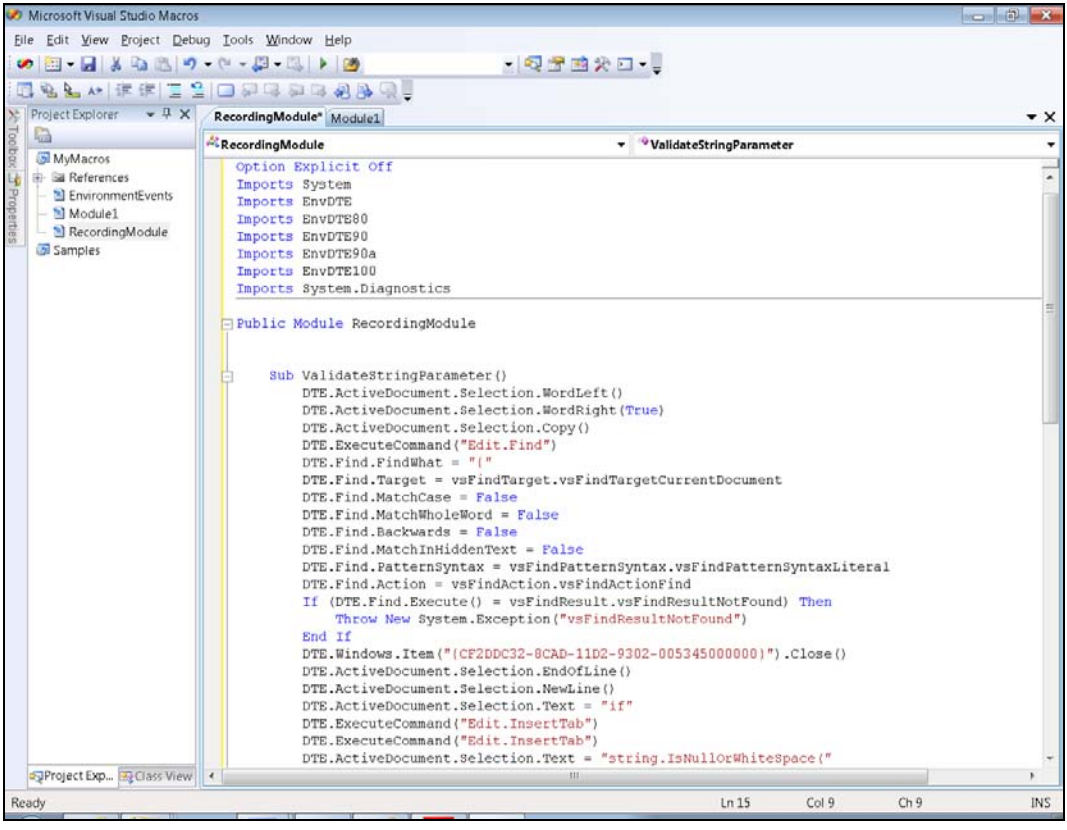


Рис. 12.10. Окно редактора макросов

Листинг 12.13. Код макроса ValidateStringParameter, написанный для работы с кодом на C# (обратите внимание, что сам макрос написан на VB!)

```
Option Strict Off
Option Explicit Off
Imports System
Imports EnvDTE
Imports EnvDTE80
Imports EnvDTE90
Imports EnvDTE90a
Imports EnvDTE100
Imports System.Diagnostics

Public Module RecordingModule


    Sub ValidateStringParameter()
```

```
        DTE.ActiveDocument.Selection.WordLeft()
        DTE.ActiveDocument.Selection.WordRight(True)
        DTE.ActiveDocument.Selection.Copy()
        DTE.ExecuteCommand("Edit.Find")
        DTE.Find.FindWhat = "{"
        DTE.Find.Target = vsFindTarget.vsFindTargetCurrentDocument
        DTE.Find.MatchCase = False
        DTE.Find.MatchWholeWord = False
        DTE.Find.Backwards = False
        DTE.Find.MatchInHiddenText = False
        DTE.Find.PatternSyntax = vsFindPatternSyntax.vsFindPatternSyntaxLiteral
        DTE.Find.Action = vsFindAction.vsFindActionFind
        If (DTE.Find.Execute() = vsFindResult.vsFindResultNotFound) Then
            Throw New System.Exception("vsFindResultNotFound")
        End If
        DTE.Windows.Item("{CF2DDC32-8CAD-11D2-9302-005345000000}").Close()
        DTE.ActiveDocument.Selection.EndOfLine()
        DTE.ActiveDocument.Selection.NewLine()
        DTE.ActiveDocument.Selection.Text = "if"
        DTE.ExecuteCommand("Edit.InsertTab")
        DTE.ExecuteCommand("Edit.InsertTab")
        DTE.ActiveDocument.Selection.Text = "string.IsNullOrWhiteSpace("
        DTE.ActiveDocument.Selection.Paste()
        DTE.ActiveDocument.Selection.Text = ")"
        DTE.ExecuteCommand("Edit.BreakLine")
        DTE.ActiveDocument.Selection.Text = "throw new ArgumentNullException("""
        DTE.ActiveDocument.Selection.Paste()
        DTE.ActiveDocument.Selection.Text = "@"
        DTE.ActiveDocument.Selection.DeleteLeft()
        DTE.ActiveDocument.Selection.Text = ""","""
        DTE.ActiveDocument.Selection.Paste()
        DTE.ActiveDocument.Selection.Text = " value is not valid"");"
        DTE.ActiveDocument.Selection.LineDown()
        DTE.ActiveDocument.Selection.NewLine()
    End Sub
End Module
```

Листинг 12.14. Код макроса ValidateStringParameter, написанный для работы с кодом на VB (обратите внимание, что сам макрос написан на VB!)

```
Option Strict Off
Option Explicit Off
Imports System
Imports EnvDTE
Imports EnvDTE80
Imports EnvDTE90
```

```
Imports EnvDTE90a
Imports EnvDTE100
Imports System.Diagnostics

Public Module RecordingModule

    Sub ValidateStringParameter()
        DTE.ActiveDocument.Selection.WordLeft()
        DTE.ActiveDocument.Selection.WordRight(True)
        DTE.ActiveDocument.Selection.CharLeft(True)
        DTE.ActiveDocument.Selection.Copy()
        DTE.ActiveDocument.Selection.EndOfLine()
        DTE.ActiveDocument.Selection.NewLine()
        DTE.ActiveDocument.Selection.Text = "if"
        DTE.ExecuteCommand("Edit.InsertTab")
        DTE.ExecuteCommand("Edit.InsertTab")
        DTE.ActiveDocument.Selection.Text = "string.IsNullOrEmpty("
        DTE.ActiveDocument.Selection.Paste()
        DTE.ActiveDocument.Selection.Text = ")"
        DTE.ActiveDocument.Selection.LineDown()
        DTE.ActiveDocument.Selection.Text =
"throw new ArgumentNullException("""
        DTE.ActiveDocument.Selection.Paste()
        DTE.ActiveDocument.Selection.Text = """, """
        DTE.ActiveDocument.Selection.Paste()
        DTE.ActiveDocument.Selection.Text =
" value is not valid."")"
        DTE.ActiveDocument.Selection.LineDown()
        DTE.ActiveDocument.Selection.NewLine()
    End Sub
End Module
```

В листингах 12.13 и 12.14 все пространства имен, которые начинаются с подстроки `EnvDTE`, имеют код, позволяющий вам манипулировать средой VS. Сам макрос представляет собой процедуру (`Sub`) внутри модуля (`Module`).

Каждое из утверждений макроса соответствует определенному шагу, который вы осуществляли при записи макроса в соответствии с процедурой, описанной в предшествующем разделе. Например, окно **Find And Replace** имеет несколько опций, которые макрос выбирает или заполняет данными в зависимости от того, какова его цель.

Открыть макрос в редакторе макросов бывает очень полезно, если вы хотите внести быстрое изменение, не прибегая к перезаписи макроса. Например, представьте себе, что при записи макроса вы забыли нажать ту или иную клавишу или просто допустили опечатку. В этому случае вам достаточно будет просто отредактировать код, сохранить файл, закрыть редактор макросов и затем перезапустить

макрос. На практике, с макросом для C# существует одна проблема, заключающаяся в том, что он будет работать только в том файле, в котором его записали. Этой проблемы не существует для макросов, написанных для кода на VB. Чуть далее, я покажу вам, как решить эту проблему, а пока вернемся к окну редактора макросов.

В среде VS вы можете открыть окно редактора макросов, выбрав из меню команды **Tools** | **Macros** | **Macros IDE**, запустить новый проект, добавить к нему модуль, а затем добавить процедуру (`Sub`) к модулю (`Module`) в качестве нового макроса. Затем можно начать кодирование макроса, введя строку **DTE.** и воспользовавшись технологией Intellisense для нахождения различных частей IDE. Загадочный параметр для `Windows.Item`, `{CF2DDC32-8CAD-11D2-9302-005345000000}`, предназначен для окна **Find And Replace**. Этот параметр называется глобально уникальным идентификатором (Globally Unique Identifier, GUID). Параметры GUID часто используются как специализированные идентификаторы программных компонентов, и в VS этот GUID применяется для уникальной идентификации различных инструментов. Таким образом, строка `DTE.Windows.Item("{CF2DDC32-8CAD-11D2-9302-005345000000}").Activate()` представляет собой способ сослаться на окно **Find And Replace** и открыть его.

Проблема с макросом для кода на C#, приведенного в листинге 12.13, возникает из-за того, что этот макрос будет работать только в файле Customer.cs. Если вы создаете новый класс с именем `Product` в файле Product.cs и добавляете к нему метод `AddNewProduct` так, как это показано в листинге 12.15, то макрос **ValidateStringParameter** попытается открыть файл Customer.cs, что не даст вам желаемого результата.

Листинг 12.15. Макрос ValidateStringParameter не будет работать в этом файле на C#

```
using System;

namespace ConsoleApplication1
{
        class Product
        {
                public int AddNewProduct(string productName)
                {
                        int newProdID = 0;

                        // Logic to add product

                        return newProdID;
                }
        }
}
```

В коде на VB, приведенном в листинге 12.16, такой проблемы не произойдет.

Листинг 12.16. В этом коде на VB наш макрос ValidateStringParameter работает отлично

```
Public Class Product

    Function AddNewProduct(ByVal productName As String) As Integer

        Dim newProdID As Integer = 0

         ' Logic to add product

        Return newProdID

    End Function

End Class
```

Чтобы решить проблему с работоспособностью макросов для кода на C#, откройте файл Customer.cs и обратите внимание на то, что макрос имеет в своем составе три утверждения, касающиеся файла Customer.cs. Закомментируйте эти выражения, как показано в листинге 12.17.

Листинг 12.17. Выражения в макросе для кода на C#, которые нужно закомментировать

```
…
'DTE.Windows.Item("Customer.cs").Activate()
…
'DTE.Windows.Item("Customer.cs").Activate()
DTE.Find.FindWhat = "{"
…
'DTE.Windows.Item("Customer.cs").Activate()
…
```

Если вы пишете ваш собственный макрос, наиболее быстрый способ определить, какой код вам следует написать, заключается в записи макроса в среде VS (<CTRL>+<SHIFT>+<R>), использовании функций VS, которые требуется закодировать, с последующей остановкой записи (<CTRL>+<SHIFT>+<R>) и сохранением макроса. Затем раскройте редактор макросов и скопируйте нужные вам части кода. Эта техника особенно полезна для определения поведения окон наподобие **Find And Replace**, как это было продемонстрировано в предыдущем разделе. Чтобы изучить макросы более серьезно, посмотрите на макросы, расположенные в составе папки Samples (см. рис. 12.9), которые иллюстрируют различные подходы к написанию макросов.

Заключение

В этой главе вы познакомились с различными приемами по индивидуальной настройке среды VS. Здесь рассказывалось об индивидуальной настройке проектов и сохранении работы в виде индивидуальных шаблонов проектов и их элементов. Этот подход дает возможность настроить проекты и работать с ними так, как удобнее всего именно вам. Далее, мы перешли к созданию индивидуальных автоматических фрагментов кода, модификации существующих автоматических фрагментов и созданию абсолютно новых. Затем было рассказано о том, как организовать автоматические фрагменты и предоставить их в общий доступ с помощью инструмента Code Snippets Manager. Наконец, в заключительной части главы мы познакомились с макросами, научились записывать макросы, пользоваться редактором макросов для модификации существующих макросов и написания новых.

Макросы представляют собой очень мощное средство, но VS только им не ограничивается. В следующей главе мы обсудим другие возможности по расширению функциональных возможностей интегрированной среды разработки (IDE) — а именно вы научитесь писать дополнительные модули (Add-Ins).

Глава 13

Расширение возможностей Visual Studio 2010

В данной главе будут обсуждаться следующие ключевые концепции:

- Создание дополнительных модулей (Add-In) с помощью VS;
- Какие типы дополнительных модулей можно создавать;
- Развертывание дополнительных модулей.

В предшествующих главах в основном обсуждались способы использования VS, разработка различных видов приложений и некоторые индивидуальные настройки среды разработки VS. В частности, в предшествующей главе обсуждались макросы, предоставляющие возможность автоматизировать выполнение периодически повторяющихся последовательностей действий, давая доступ практически ко всем возможностям, предлагаемым VS. Однако возможности VS этим не ограничиваются. Если сделать еще один шаг вперед, то выяснится, что функции VS можно расширить и за пределы макросов. В данной главе мы обсудим расширение возможностей VS при помощи специальных программных компонентов, которые называются дополнительными модулями (Add-In).

В принципе, дополнительный модуль (Add-In) представляет собой программный компонент, который позволяет реализовать новые функции, которых в VS до сих пор не было. Дополнительный модуль подключается к VS, и его можно запускать из среды VS, так, как если бы он был частью VS. В этой главе показано, как проходит весь процесс создания дополнительного модуля — от начала разработки и до установки и развертывания. Здесь будет продемонстрировано, как добавить дополнительные возможности в состав в VS с помощью дополнительного модуля. Помимо создания дополнительных модулей, в этой главе указано, в каком направлении следует двигаться, чтобы правильно определить, как получить доступ к различным компонентам VS. Приведенный в этой главе пример представляет собой дополнительный модуль, который находит все клавиатурные комбинации, существующие в VS, и выводит их в окно **Output**. После описания процесса написания дополнительного модуля будет рассказано о том, как выполнить его развертывание, чтобы новый дополнительный модуль мог быть загружен в VS. Начнем с обсуждения того, как VS помогает в создании дополнительных модулей.

Разработка дополнительного модуля Visual Studio

Как и при создании других типов проектов VS, вы можете запустить программу Add-In Project Wizard и для написания дополнительных модулей (Add-In), специальных программных модулей, призванных расширить функциональные возможности VS. В следующем разделе будет рассказано о том, как запустить программу-мастер Add-In Project Wizard, а также будут обсуждаться полученные результаты.

Запуск программы-мастера Add-In Project Wizard

Чтобы начать работу над проектом дополнительного модуля VS, действовать нужно точно так же, как и при запуске любого другого нового проекта. Разница состоит только в том, что программа-мастер Add-In Project Wizard задает вопросов больше, чем аналогичные программы-мастера, создающие проекты других типов. Приведенное описание пошаговой процедуры создания нового дополнительного модуля VS поясняет каждый шаг, который вам необходимо выяснить, и дает пояснения ко всем вопросам, на которые вам потребуется ответить.

1. Запустите VS и нажмите клавиатурную комбинацию <CTRL>+<SHIFT>+<N>, чтобы открыть окно **New Project**. В этом окне выберите опции **Other Project Types | Extensibility**, и вы увидите два варианта проектов дополнительных модулей, которые вы можете создать: **Visual Studio Add-In** (дополнительный модуль Visual Studio) и **Shared Add-In** (разделяемый дополнительный модуль). Опцию **Shared Add-In** следует применять для создания дополнительных модулей Microsoft Office (Microsoft Office Add-In). Опция **Visual Studio Add-In** называется так, потому что она предназначена для создания дополнительных модулей, расширяющих функциональные возможности самого продукта Visual Studio. Это — именно то, чем мы и будем заниматься в данной главе. Вид экрана после выбора опций **Other Project Types | Extensibility** показан на рис. 13.1.
2. Выберите опцию **Visual Studio Add-In**. Дайте новому проекту имя KeystrokeFinder, укажите папку, в которой хотите создать новый проект, и нажмите кнопку **OK**. Нажмите кнопку **Next**, чтобы закрыть экран-приветствие и перейти к экрану **Select A Programming Language**, показанному на рис. 13.2.
3. В окне **Select a Programming Language** выберите язык программирования, на котором собираетесь писать свой дополнительный модуль. Язык C++ в этой книге не обсуждается, но вы можете выбрать одну из опций, предложенных для этого языка, если вы хорошо им владеете. В рассматриваемом примере я рекомендую вам выбрать либо C#, либо VB, базовую информацию о синтаксисе которых можно найти в *главах 2, 3* и *4*. Выбрав язык программирования, на котором вы собираетесь работать, нажмите кнопку **Next**. На экране появится окно **Select An Application Host**, показанное на рис. 13.3.

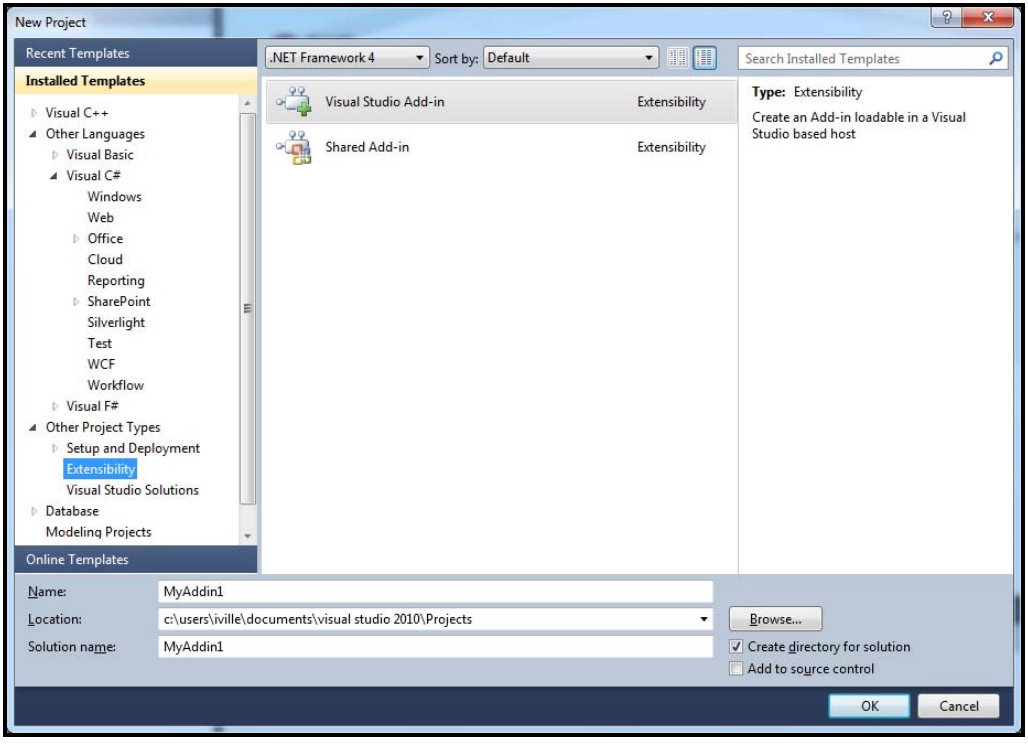


Рис. 13.1. Выбор опции создания **Visual Studio Add-In** в окне **New Project**

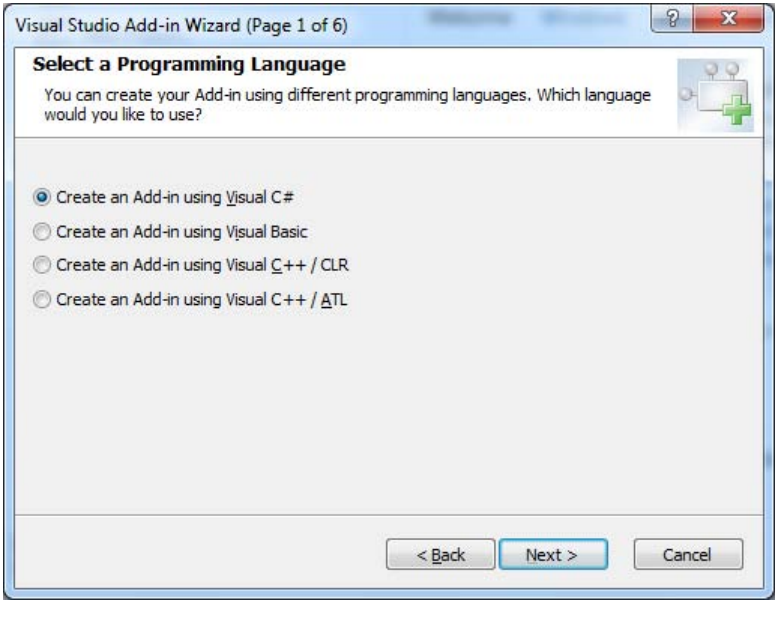


Рис. 13.2. Окно **Select A Programming Language**

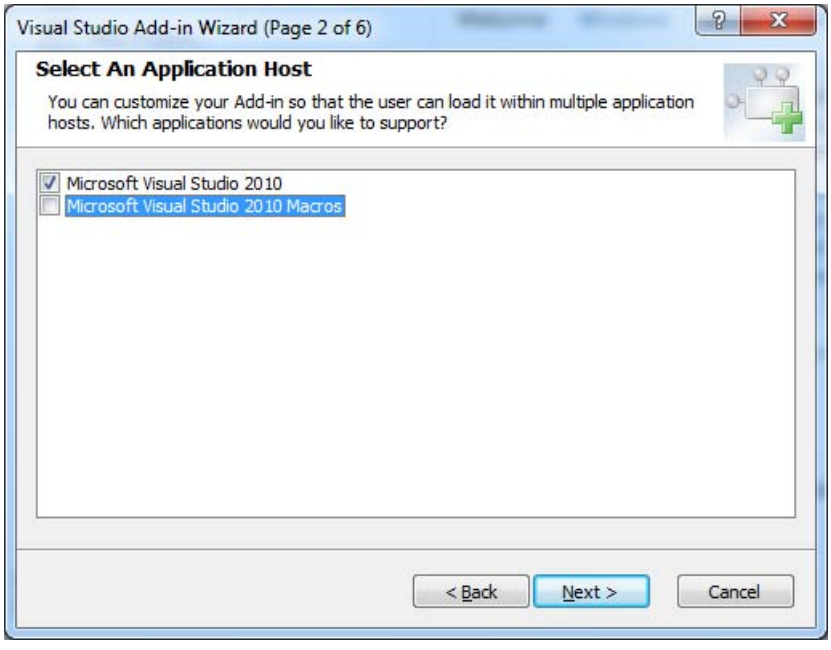


Рис. 13.3. Окно **Select An Application Host**

4. В окне **Select An Application Host** для выбора доступны две опции: **Microsoft Visual Studio 2010** и **Microsoft Visual Studio 2010 Macros**. Если вы выберете опцию **Microsoft Visual Studio 2010**, то ваш дополнительный модуль будет работать в среде VS, обсуждению которой посвящена практически вся данная книга. Если выбрана опция **Microsoft Visual Studio 2010 Macros**, то ваш дополнительный модуль будет работать с редактором макросов (Macro Editor), который обсуждался в заключительной части *главы 12*. В рассматриваемом примере дополнительного модуля нам нужна только опция **Microsoft Visual Studio 2010**, а опция **Microsoft Visual Studio 2010 Macros** не нужна. Поэтому установите флажок **Microsoft Visual Studio 2010** и нажмите кнопку **Next**, чтобы открыть следующее окно, **Enter a Name and Description**, показанное на рис. 13.4.
5. Изначально поля, доступные в окне **Enter a Name and Description** заполнены стандартными значениями по умолчанию — **No Name provided** и **No Description provided**. Удалите эти значения по умолчанию и введите имя (Name) и описание (Description), которыми вы хотите снабдить свой дополнительный модуль. Дополнительный модуль (Add-In) получит имя, которое вы укажете здесь, а описание этого модуля пользователи смогут прочесть, работая с инструментом VS Add-In Manager, который будет рассматриваться далее в этой главе. Заполнив поля, нажмите кнопку **Next**, и на экране появится окно **Choose Add-In Options**, показанное на рис. 13.5.

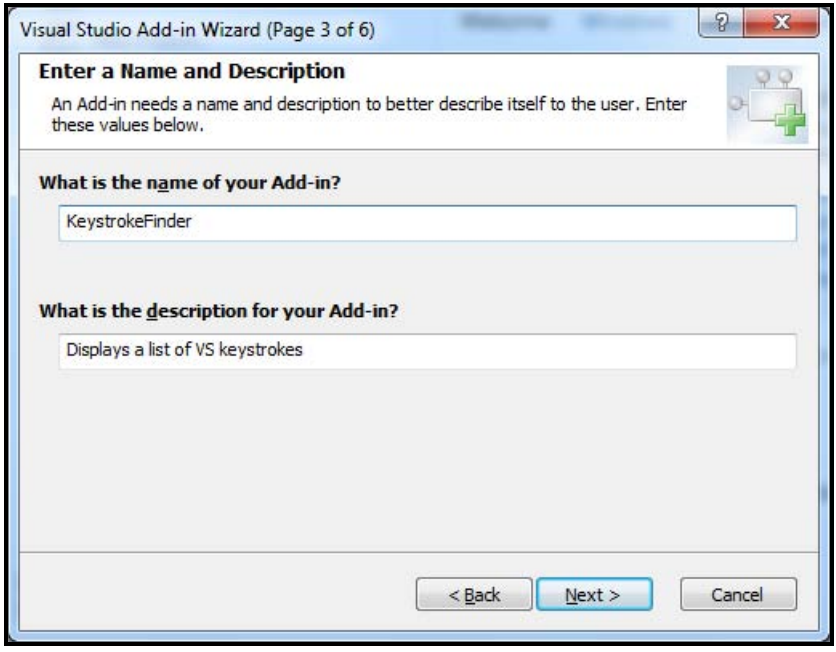


Рис. 13.4. Окно **Enter a Name and Description**

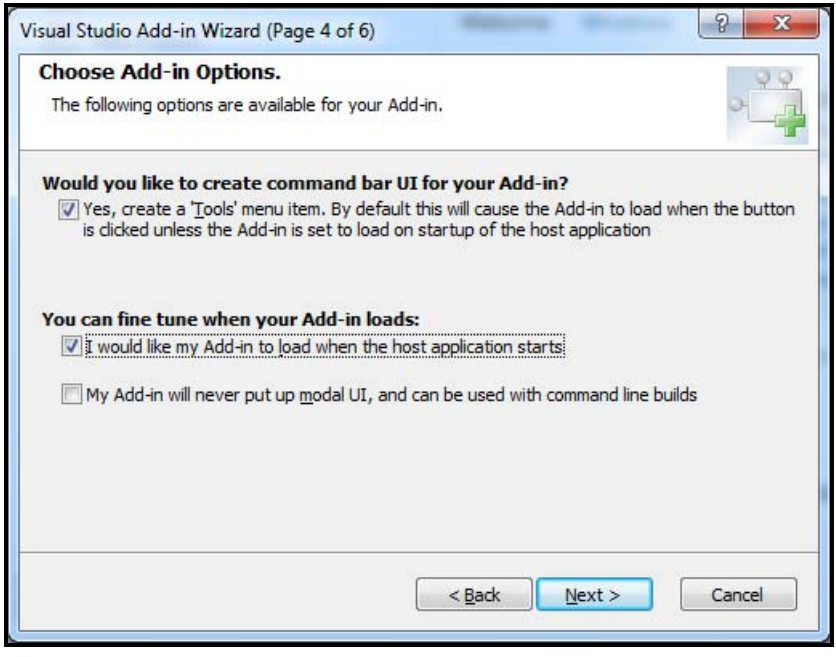


Рис. 13.5. Окно **Choose Add-In Options**

6. Первая из опций, доступных в окне, показанном на рис. 13.5, **Would you like to create command bar UI for your Add-In?**, позволяет добавить пункт меню в состав меню **Tools**. Новый пункт меню будет называться по имени, которое вы дали своему новому дополнительному модулю. Установите этот флажок. Кроме того, установите и следующий флажок, который позволяет загружать дополнительный модуль при запуске VS. В качестве альтернативного варианта, вы можете загружать дополнительный модуль вручную с помощью инструмента Add-In Manager, который будет обсуждаться далее в этой главе. Наконец, третья опция позволяет указывать, хотите ли вы, чтобы ваш дополнительный модуль работал, если VS запускается через командную строку. В предыдущей главе был приведен пример запуска VS через командную строку при установке глобальных шаблонов проектов путем запуска команды `devenv /installvstemplates`. Всплывающее модальное окно (это то самое окно, в котором вы должны нажать кнопку **OK**, чтобы закрыть его) остановит работу командной строки, потому что она ожидает уведомления от модельного окна. Если операции, выполняемые из командной строки, работают в фоновом режиме по расписанию, то у вас не будет никакого способа отправить это уведомление, поэтому задания, запущенные из командной строки, выполнены не будут. Поэтому устанавливайте этот флажок только в тех случаях, когда это не мешает выполнению заданий из командной строки. В нашем примере следует установить первые две опции, а последнюю оставить неактивированной. Затем нажмите кнопку **Next**, чтобы перейти в окно **Choosing 'Help About' Information**, показанное на рис. 13.6.

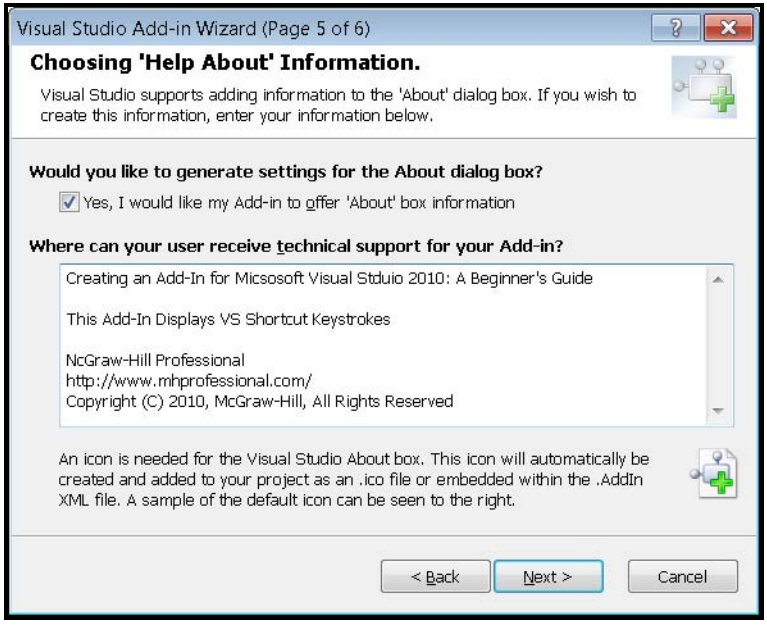


Рис. 13.6. Окно **Choosing 'Help About'**

7. При желании, вы можете отображать для вашего дополнительного модуля (Add-In) окно **About**. Для этого установите флажок **Yes, I would like my Add-In to offer 'About' box information**. Далее, перейдите в расположенное ниже текстовое поле и отредактируйте текст, который должен отображаться в окне **About**. Щелкните мышью по кнопке **Next**, а в следующем окне, озаглавленном **Summary**, нажмите кнопку **Finish**.

Примерно через минуту VS создаст новое решение и проект, который будет содержать все элементы, упрощающие создание дополнительного модуля. В следующем разделе мы рассмотрим типовые элементы проекта нового дополнительного модуля.

Исследование решения, созданного программой-мастером Add-In Wizard

После того как отработает программа-мастер Add-In Project Wizard, VS создаст решение, в составе которого будет проект, содержащий код, который представляет собой "каркас" будущего приложения. Вам потребуется не только разобраться с тем, какие файлы будут вам доступны, но и понять, какие предоставляются интерфейсы, и научиться правильно реализовывать методы этих интерфейсов. Если понимание интерфейсов вызывает у вас некоторые сложности, то сейчас самое время перечитать главу 4, чтобы освежить в памяти необходимую информацию. Основная информация, которую вам требуется знать об этом проекте, заключается в следующем: он содержит новые ссылки (references), класс `Connect` и парочку файлов *.AddIn. Все компоненты нового проекта показаны на рис. 13.7, и в последующих нескольких разделах мы обсудим их в том порядке, в котором они представлены в окне **Solution Explorer**.

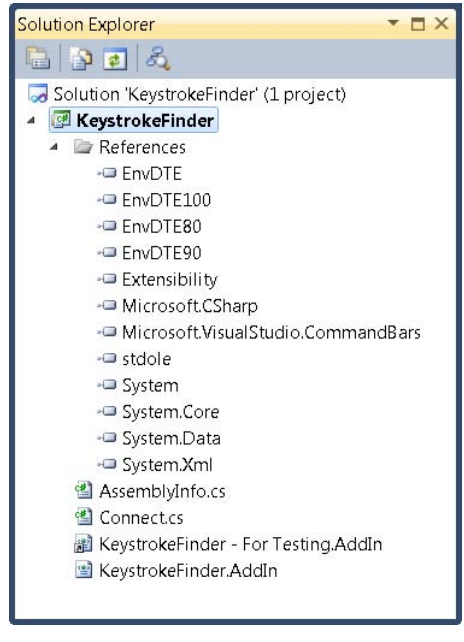


Рис. 13.7. Проект нового дополнительного модуля (Add-In) в окне **Solution Explorer**

В папке **References** (см. рис. 13.7) перечислены ссылки на сборки (assembly references). Возможно, у вас сразу же возникнет вопрос о том, что же представляют собой сборки, имена которых содержат подстроку `EnvDTE`. Если разобрать структуру имени, то подстрока `Env` представляет собой сокращение от `environment` (среда), а аббревиатура DTE расшифровывается как Development Tools

Extensibility (объектная модель DTE, "средства расширения инструментария разработчика"). Таким образом, сборки, имена которых содержат подстроку `EnvDTE`, представляют собой сборки, содержащие код, позволяющий расширять среду разработки VS. Каждая из таких сборок реализует функциональные возможности, присущие конкретной версии VS: сборка `EnvDTE` предназначена для VS.NET (первой версии VS, которая поддерживала разработку для платформы .NET) и VS 2003, сборка `EnvDTE80` предназначена для VS 2005, `EnvDTE90` — для VS 2008, и, наконец, `EnvDTE100` — для VS 2010 (тема настоящей книги). Причина, по которой вам нужны ссылки на все версии `EnvDTE`, заключается в том, что каждая новая версия основывается на предыдущей с добавлением новых функциональных возможностей, а не замещает старую версию полностью. Следовательно, иногда вам потребуются классы, интерфейсы или методы, снабженные номерами, например, `IDTExtensibility` и `IDTExtensibility2`, где `IDTExtensibility2` представляет собой более новую версию с дополнительными членами. Интерфейс `IDTExtensibility2` мы обсудим чуть позже, а сейчас очень важно разобраться с тем, как каждая версия сборок `EnvDTE` управляет более новыми версиями кода. Предложенная схема предлагает наращивание новых функциональных возможностей с выходом каждой новой версии VS, не жертвуя при этом обратной совместимостью.

Класс `Connect` содержит код, который взаимодействует с VS, благодаря чему и работают дополнительные модули Add-In. Не стоит забывать, что мы в данном случае имеем дело с проектом VS, и он ведет себя так же, как и другие проекты, которые вы можете создавать. Вы имеете возможность добавлять классы, реализующие любые функциональные возможности, и добавлять в класс `Connect` код, который будет вызывать ваши классы, организовать код по папкам, добавлять к решению библиотеки классов и вызывать код из этих библиотек. Класс `Connect` будет подробно исследоваться в следующем разделе.

Кроме того, внимания заслуживают и такие компоненты проекта, как файлы с расширением *.AddIn. Они представляют собой файлы развертывания. Когда-то давно, в старые времена, вам необходимо было редактировать реестр Windows с тем, чтобы зарегистрировать и сконфигурировать модули расширения, но теперь это не так. Конфигурацию дополнительных модулей теперь следует выполнять в файлах *.AddIn, которые содержат код XML. Далее в этой главе мы подробно рассмотрим "анатомию" файлов *.AddIn и научимся манипулировать этими файлами для осуществления развертывания.

Кроме того, внимательно посмотрев на файлы *.AddIn в окне **Solution Explorer**, вы, несомненно, обнаружите, что один из них представляет собой ярлык (shortcut), потому что его значок снабжен миниатюрным изображением "стрелки", которая, как известно, и отличает ярлыки от обычных файлов. Этот файл используется в отладочных целях. Если вы просмотрите свойства этого файла, то вы увидите, что он ссылается на папку Documents\Visual Studio 2010\Addins\<*project_name*> (где *project_name* — это имя вашего проекта). Когда вы будете отлаживать ваше приложение, VS использует отладочный файл *.AddIn для загрузки дополнительного модуля в новую копию VS. Манипуляции над дополнительным модулем вы будет осуществлять в новой копии VS, а ваша текущая копия VS, работая в отладочном

режиме, сможет работать с точками останова (breakpoints) и выполнять отладку дополнительного модуля (Add-In).

Теперь, ознакомившись с ключевыми компонентами проекта дополнительного модуля, предназначенного для расширения функциональных возможностей VS, мы можем приступить к подробному изучению класса `Connect` и его членов, которые взаимодействуют с VS, обеспечивая возможность работы вашего дополнительного модуля.

Изучение структуры класса Connect

Класс `Connect` реализует два интерфейса, `IDTExtensibility2` и `IDTCommandTarget`, и содержит несколько членов. Прежде чем приступать к изучению кода, давайте поговорим об интерфейсах, их членах и их предназначении.

Предназначение интерфейсов (`IDTExtensibility2` и `IDTCommandTarget`) заключается в том, чтобы помогать управлять временем жизни дополнительного модуля. VS интерпретирует эти интерфейс, но не имеет никакой информации о коде, который вы пишете. Следовательно, вам необходимо "перебросить мостик" между вашим кодом и VS, предоставив VS всю информацию, необходимую для того, чтобы дополнительный модуль (Add-In) мог работать. Чтобы сделать это, вы пользуетесь классом `Connect`, который реализует интерфейсы `IDTExtensibility2` и `IDTCommandTarget`. Затем вам необходимо будет ввести код методов, являющихся членами класса `Connect`, которые реализуют эти интерфейсы. Когда VS взаимодействует с интерфейсами, исполняется ваш код, который их реализует.

В чем-то этот алгоритм напоминает беседу людей, говорящих на разных языках. Чтобы понять друг друга и предмет обсуждения, они нуждаются в общем языке, на котором каждый из них говорит вполне удовлетворительно для того, чтобы понимать друг друга. Этот общий язык, которым владеют оба собеседника, в данном случае играет роль интерфейса между ними.

Первый интерфейс, который нам требуется обсудить — это интерфейс `IDTExtensibility2`, цель которого заключается в том, чтобы дать VS возможность загружать и выгружать дополнительные модули. Загрузка и выгрузка дополнительных модулей играет важную роль, потому что VS загружает модули расширения при запуске и выгружает их при закрытии. Существует ряд действий, которые необходимо предпринять, в зависимости от того, когда загружается дополнительный модуль, и от типа информации, к которой вам необходимо получить доступ. Например, когда дополнительный модуль (Add-In) загружается впервые, вам может потребоваться выполнить специальные операции наподобие конфигурирования или вывода запросов пользователю, предлагающих ему зарегистрировать ваш дополнительный модуль. Члены `IDTExtensibility2` с краткими описаниями их целей и задач перечислены в табл. 13.1.

Второй интерфейс, реализуемый классом `Connect` — это интерфейс `IDTCommandTarget`. При сборке дополнительного модуля вам нужен некоторый способ, при помощи которого интегрированная среда разработки VS IDE могла выполнить этот дополнительный модуль. Например, вы можете создать именованную команду, которая предоставит дополнительный модуль как одну из команд,

которые появляются в меню **Tools**. Каждый раз, когда пользователь будет выбирать этот пункт меню, будет выполняться именованная команда, запускающая код дополнительного модуля. `IDTCommandTarget` представляет собой интерфейс, который VS использует для исполнения вашего дополнительного модуля. В табл. 13.2 перечислены члены интерфейса `IDTCommandTarget` и описано их предназначение.

Таблица 13.1. *Члены интерфейса* `IDTExtensibility2`

Член	Предназначение
`OnAddInsUpdate`	Реакция на изменение состояния дополнительного модуля (ручная загрузка или выгрузка)
`OnBeginShutdown`	Дополнительный модуль загружен, и в этот момент пользователь закрывает VS
`OnConnection`	Дополнительный модуль загружается
`OnDisconnection`	Дополнительный модуль выгружается
`OnStartupComplete`	VS запускается и загружает дополнительный модуль

Таблица 13.2. *Члены интерфейса* `IDTCommandTarget`

Член	Предназначение
`Exec`	Вызывается VS для исполнения дополнительного модуля
`QueryStatus`	Вызывается VS для того, чтобы определить, должна ли команда быть активной, невидимой или поддерживаемой

Каждый из методов, реализующих интерфейсы `IDTExtensibility2` и `IDCommandTarget`, реализуется классом `Connect`. В листинге 13.1 представлены все эти члены и "каркасный" код будущего приложения. Код, представленный в листинге 13.1, написан на C#. При изучении этого кода очень полезно изучить обзорную информацию об интерфейсах из табл. 13.1 и 13.2, а также внимательно изучить комментарии, описывающие предназначение "каркасного" кода. Если вы решите писать дополнительный модуль на VB, то эти комментарии будут точно такими же. Некоторые из комментариев относятся на хост-приложение, роль которого будут исполнять либо интегрированная среда разработки VS IDE, либо редактор макросов, в зависимости от того, какие опции вы выбрали в соответствующем окне программы-мастера Add-In Project Wizard (см. рис. 13.3).

Листинг 13.1. "Каркасный" код класса Connect

```
using System;
using Extensibility;
using EnvDTE;
using EnvDTE80;
using Microsoft.VisualStudio.CommandBars;
using System.Resources;
```

```
using System.Reflection;
using System.Globalization;

namespace KeystrokeFinder
{
    /// <summary> The object for implementing an Add-in.</summary>
    /// <see also class='IDTExtensibility2' />
    public class Connect : IDTExtensibility2, IDTCommandTarget
    {
        /// <summary>Implements the constructor for the Add-in object.
        /// Place your initialization code within this method.</summary>
        public Connect()
        {
        }

        /// <summary>Implements the OnConnection method of the IDTExtensibility2
        /// interface. Receives notification that the Add-in is being
        /// loaded.</summary>
        /// <param term='application'>Root object of the host application.</param>
        /// <param term='connectMode'>Describes how the Add-in is being
        /// loaded.</param>
        /// <param term='addInInst'>Object representing this Add-in.</param>
        /// <seealso class='IDTExtensibility2' />
        public void OnConnection(object application, ext_ConnectMode connectMode,
                                 object addInInst, ref Array custom)
        {
            _applicationObject = (DTE2)application;
            _addInInstance = (AddIn)addInInst;
            if(connectMode == ext_ConnectMode.ext_cm_UISetup)
            {
                object []contextGUIDS = new object[] { };
                Commands2 commands = (Commands2)_applicationObject.Commands;
                string toolsMenuName = "Tools";

                // Place the command on the tools menu.
                // Find the MenuBar command bar, which is the top-level command
                // bar holding all the main menu items:
                Microsoft.VisualStudio.CommandBars.CommandBar menuBarCommandBar =
               ((Microsoft.VisualStudio.CommandBars.CommandBars)
                _applicationObject.CommandBars) ["MenuBar"];

                //Find the Tools command bar on the MenuBar command bar:
                CommandBarControl toolsControl =
                             menuBarCommandBar.Controls[toolsMenuName];
```

```
            CommandBarPopup toolsPopup = (CommandBarPopup)toolsControl;

            //This try/catch block can be duplicated if you wish
            //to add multiple commands to be handled by your Add-in,
            //  just make sure you also update the QueryStatus/Exec method
            //  to include the new command names.
            try
            {
                //Add a command to the Commands collection:
                Command command = commands.AddNamedCommand2(_addInInstance,
                   "KeystrokeFinder", "KeystrokeFinder", "Executes the command
                   for KeystrokeFinder", true, 59, ref contextGUIDS,
                         (int)vsCommandStatus.vsCommandStatusSupported+
                         (int)vsCommandStatus.vsCommandStatusEnabled,
                         (int)vsCommandStyle.vsCommandStylePictAndText,
                          vsCommandControlType.vsCommandControlTypeButton);

                //Add a control for the command to the tools menu:
                if((command != null) && (toolsPopup != null))
                {
                    command.AddControl(toolsPopup.CommandBar, 1);
                }
            }
            catch(System.ArgumentException)
            {
                //If we are here, then the exception is probably
                //because a command with that name
                //  already exists. If so there is no need to
                //  recreate the command and we can
                //  safely ignore the exception.
            }
        }
    }

    /// <summary>Implements the OnDisconnection method of the
    /// IDTExtensibility2 interface. Receives notification that the Add-in is
    /// being unloaded.</summary>
    /// <param term='disconnectMode'>Describes how the Add-in is being
    /// unloaded.</param>
    /// <param term='custom'>Array of parameters that are host
    /// application specific.</param>
    /// <seealso class='IDTExtensibility2' />
    public void OnDisconnection(ext_DisconnectMode disconnectMode,
                                ref Array custom)
    {
```

```
}

/// <summary>Implements the OnAddInsUpdate method of the IDTExtensibility2
/// interface. Receives notification when the collection of Add-ins has
/// changed.</summary>
/// <param term='custom'>Array of parameters that are host application
/// specific.</param>
/// <seealso class='IDTExtensibility2' />
public void OnAddInsUpdate(ref Array custom)
{
}

/// <summary>Implements the OnStartupComplete method of the
/// IDTExtensibility2 interface. Receives notification that the host
/// application has completed loading.</summary>
/// <param term='custom'>Array of parameters that are host application
/// specific.</param>
/// <seealso class='IDTExtensibility2' />
public void OnStartupComplete(ref Array custom)
{
}

/// <summary>Implements the OnBeginShutdown method of the IDTExtensibility2
/// interface. Receives notification that the host application is being
/// unloaded.</summary>
/// <param term='custom'>Array of parameters that are host application
/// specific.</param>
/// <seealso class='IDTExtensibility2' />
public void OnBeginShutdown(ref Array custom)
{
}

/// <summary>Implements the QueryStatus method of the IDTCommandTarget
/// interface. This is called when the command's availability is
/// updated</summary>
/// <param term='commandName'>The name of the command to determine state
/// for.</param>
/// <param term='neededText'>Text that is needed for the command.</param>
/// <param term='status'>The state of the command in the user
/// interface.</param>
/// <param term='commandText'>Text requested by the neededText
/// parameter.</param>
/// <seealso class='Exec' />
public void QueryStatus(string commandName, vsCommandStatusTextWanted
            neededText, ref vsCommandStatus status, ref object commandText)
```

```
        {
            if(neededText ==
                     vsCommandStatusTextWanted.vsCommandStatusTextWantedNone)
            {
                if(commandName == "KeystrokeFinder.Connect.KeystrokeFinder")
                {
                    status =
                     (vsCommandStatus)vsCommandStatus.vsCommandStatusSupported|
                      vsCommandStatus.vsCommandStatusEnabled;
                    return;
                }
            }
        }

        /// <summary>Implements the Exec method of the IDTCommandTarget interface.
        /// This is called when the command is invoked.</summary>
        /// <param term='commandName'>The name of the command to execute.</param>
        /// <param term='executeOption'>Describes how the command should be
        /// run.</param>
        /// <param term='varIn'>Parameters passed from the caller to the command
        /// handler.</param>
        /// <param term='varOut'>Parameters passed from the command handler to the
        /// caller.</param>
        /// <param term='handled'>Informs the caller if the command was handled or
        /// not.</param>
        /// <seealso class='Exec' />
        public void Exec(string commandName, vsCommandExecOption executeOption,
                         ref object varIn, ref object varOut, ref bool handled)
        {
            handled = false;
            if(executeOption == vsCommandExecOption.vsCommandExecOptionDoDefault)
            {
                if(commandName == "KeystrokeFinder.Connect.KeystrokeFinder")
                {
                    handled = true;
                    return;
                }
            }
        }
        private DTE2 _applicationObject;
        private AddIn _addInInstance;
    }
}
```

Итак, вы получили обзорную информацию об интерфейсах `IDTExtensibility2` и `IDTCommandTarget`, о том, что они делают и в чем заключается их предназначе-

ние. В следующем разделе вы начнете дописывать собственный код к методам интерфейса с тем, чтобы наш дополнительный модуль KeystrokeFinder начал выполнять полезную работу.

Добавление функциональных возможностей в дополнительный модуль

При реализации функциональных возможностей дополнительного модуля, наибольший интерес для вас представляет перехват вызова к `Exec`, методу, который VS вызывает каждый раз, когда пользователь выбирает из меню **Tools** команду, которая должна запускать ваш дополнительный модуль. В данном разделе будет также рассмотрена парочка дополнительных методов: `OnConnection`, метод, который выполняет большую часть инициализационного кода, и метод `QueryStatus`, который удобен для управления статусом элемента меню, запускающего ваш дополнительный модуль. Сначала рассмотрим метод `OnConnection`, потому что сначала нужно ознакомиться с тем, как осуществляется инициализация дополнительного модуля.

Изучение метода OnConnection

Как вы уже знаете, класс `Connect` реализует различные методы интерфейсов с тем, чтобы среда разработки VS могла вызывать эти методы, которые обеспечивают исполнение вашего дополнительного модуля. Одним из основных методов является метод `OnConnection`, который является членом интерфейса `IDTExtensibility2`. VS вызывает метод `OnConnection`, когда загружается ваш дополнительный модуль. При вызове метода `OnConnection`, VS передает ему четыре параметра, которые вы можете использовать для инициализации дополнительного модуля. Программа-мастер Add-In Project Wizard, рассмотренная ранее в этой главе, генерирует большую часть "каркасного" кода, который использует значения параметров метода `OnConnection` для инициализации дополнительного модуля. Хотя пример, рассматриваемый в этой главе, и не модифицирует метод `OnConnection`, понимание принципов работы этого кода очень важно для понимания процесса инициализации дополнительного модуля и влияния этого процесса на код, который вы будете писать впоследствии. Для начала давайте еще раз посмотрим на параметры метода `OnConnection` и изучим сгенерированный код.

Параметры метода OnConnection

Метод `OnConnection` принимает четыре параметра. Каждый из этих параметров VS передает методу `OnConnection`; эти параметры содержат все сведения, необходимые для инициализации дополнительного модуля. В табл. 13.3 перечислены все эти параметры и кратко описано их предназначение.

Таблица 13.3. *Параметры метода* `OnConnection`

Член	Тип	Предназначение
`application`	Тип времени компиляции — `Object`, но во время исполнения тип определяется версией VS, с которой вы работаете. Например, в более старых версиях VS тип времени исполнения был `DTE`, а в VS 2010 типом времени исполнения для параметра `application` будет `DTE2`	`Application` представляет собой родительский объект для всей модели автоматизации, принятой в VS. Он используется для доступа ко всем окнам, командам и другим компонентам интегрированной среды разработки (IDE)
`connectMode`	Перечисление (Enum) типа `ext_ConnectMode`	Этот параметр позволяет определить, когда и как загружается дополнительный модуль. В последующем разделе будет показано, как метод `OnConnection` считывает значение этого параметра для того, чтобы определить, когда дополнительный модуль загружается впервые
`addInInst`	Тип времени компиляции — `Object`, но во время исполнения этот параметр имеет тип `AddIn`	Этот параметр ссылается на сам дополнительный модуль, позволяя вам изучить свойства этого дополнительного модуля
`custom`	Массив (`Array`)	В примере, рассматриваемом в данной главе, не используется. Однако представьте себе ситуацию, когда вам необходимо реализовать интерфейс. Помимо VS 2010, вы можете иметь другое хост-приложение, поддерживающее дополнительные модули расширения, реализующие интерфейс `IDTExtensibility2`. Эти хост-приложения могут использовать параметр `custom` с типом массива (`array`) для передачи информации, специфичной для данных приложений. Следовательно, параметр `custom` представляет собой еще одну точку расширения, позволяющую сделать интерфейс `IDTExtensibility2` еще более гибким

Изучение кода, сгенерированного для метода OnConnection

Как вы уже знаете, основной целью метода `OnConnection` является помощь в инициализации дополнительного модуля расширения. В предыдущем разделе мы уже рассмотрели параметры, передаваемые VS данному методу, и рассмотрели смысл каждого из этих параметров. В листингах 13.2 и 13.3 показан код, генерируемый VS после завершения работы программы-мастера Add-In Project Wizard. Этот код отражает результаты выбора опций дополнительного модуля таким образом, как было показано на рис. 13.5.

Листинг 13.2. Код метода OnConnection на C#

```
public void OnConnection(
    object application, ext_ConnectMode connectMode,
    object addInInst, ref Array custom)
{
    _applicationObject = (DTE2)application;
    _addInInstance = (AddIn)addInInst;
    if(connectMode == ext_ConnectMode.ext_cm_UISetup)
    {
            object []contextGUIDS = new object[] { };
            Commands2 commands =
              (Commands2)_applicationObject.Commands;
            string toolsMenuName = "Tools";

            Microsoft.VisualStudio.CommandBars.CommandBar
                menuBarCommandBar = ((
                    Microsoft.VisualStudio.CommandBars.CommandBars)
                    _applicationObject.CommandBars)["MenuBar"];
            CommandBarControl toolsControl =
                menuBarCommandBar.Controls[toolsMenuName];
            CommandBarPopup toolsPopup =
                (CommandBarPopup)toolsControl;

            try
            {
                Command command = commands.AddNamedCommand2(
                    _addInInstance, "KeystrokeFinder",
                    "KeystrokeFinder",
                    "Executes the command for KeystrokeFinder",
                    true, 59, ref contextGUIDS,
                    (int)vsCommandStatus
                         .vsCommandStatusSupported+
                    (int)vsCommandStatus.vsCommandStatusEnabled,
                    (int)vsCommandStyle
                         .vsCommandStylePictAndText,
                     vsCommandControlType
                         .vsCommandControlTypeButton);

                if((command != null) &&
                     (toolsPopup != null))
                {
                       command.AddControl(
                           toolsPopup.CommandBar, 1);
                }
```

```
        }
        catch(System.ArgumentException)
        {
        }
    }
}
```

Листинг 13.3. Код метода OnConnection на VB

```
Public Sub OnConnection(
    ByVal application As Object,
    ByVal connectMode As ext_ConnectMode,
    ByVal addInInst As Object,
    ByRef custom As Array) Implements IDTExtensibility2.OnConnection
    _applicationObject = CType(application, DTE2)
    _addInInstance = CType(addInInst, AddIn)

    If connectMode = ext_ConnectMode.ext_cm_UISetup Then

        Dim commands As Commands2 =
            CType(_applicationObject.Commands, Commands2)
        Dim toolsMenuName As String = "Tools"

        Dim commandBars As CommandBars =
            CType(_applicationObject.CommandBars, CommandBars)
        Dim menuBarCommandBar As CommandBar =
            commandBars.Item("MenuBar")

        Dim toolsControl As CommandBarControl =
            menuBarCommandBar.Controls.Item(toolsMenuName)
        Dim toolsPopup As CommandBarPopup =
            CType(toolsControl, CommandBarPopup)

        Try

            Dim command As Command =
                commands.AddNamedCommand2(
                    _addInInstance, "KeystrokeFinderVB",
                    "KeystrokeFinderVB",
                    "Executes the command for KeystrokeFinderVB",
                    True, 59, Nothing,
                    CType(vsCommandStatus.vsCommandStatusSupported,
                        Integer) +
                    CType(vsCommandStatus.vsCommandStatusEnabled,
                        Integer),
                    vsCommandStyle.vsCommandStylePictAndText,
```

```
                    vsCommandControlType.vsCommandControlTypeButton)

            command.AddControl(toolsPopup.CommandBar, 1)
        Catch argumentException As System.ArgumentException
        End Try

    End If
End Sub
```

Разбиение кода, приведенного в листингах 13.2 и 13.3, на структурные единицы демонстрирует роль метода `OnConnection` и как он влияет на последующий код. Первая часть метода получает ссылки на два важных объекта: `application` и `addInInst`. Фрагменты кода, приведенные в листингах 13.4 и 13.5, показывают, как получить ссылки на эти объекты и преобразовать их к типам `DTE2` и `AddIn`, соответственно. Ссылки на `_applicationObject` и `_addInInstance` являются полями класса `Connect`, играющими важную роль, потому что теперь другие методы класса смогут получать доступ к этим объектам.

Листинг 13.4. Код на C#, получающий ссылки на объекты application и addInInst

```
_applicationObject = (DTE2)application;
_addInInstance = (AddIn)addInInst;
```

Листинг 13.5. Код на VB, получающий ссылки на объекты application и addInInst

```
_applicationObject = CType(application, DTE2)
_addInInstance = CType(addInInst, AddIn)
```

Остальной код в методе `OnConnection` устанавливает команду из меню **Tools**, как и было указано при создании проекта (см. рис. 13.5). Однако это происходит только один раз — при первом запуске приложения. Чтобы убедиться в том, что команда меню создается только один раз, код проверяет, установлен ли параметр `connectMode` на значение `ext_ConnectMode.ext_cm_UISetup`, как показано в листингах 13.6 и 13.7. Остальной код метода `OnConnection` будет исполняться только в том случае, если это условие выполняется.

Листинг 13.6. Проверка установки параметра connectMode (код на C#)

```
if(connectMode == ext_ConnectMode.ext_cm_UISetup)
```

Листинг 13.7. Проверка установки параметра connectMode (код на VB)

```
If connectMode = ext_ConnectMode.ext_cm_UISetup Then
```

Когда код выполняется в первый раз, то выполняется утверждение `if` в листингах 13.6 и 13.7. В результате будет создан пункт меню для дополнительного модуля

KeystrokeFinder в меню **Tools**. Примеры кода, приведенные далее в этом разделе, все содержатся внутри только что рассмотренного утверждения `if`; это полезно знать, поскольку это помогает ориентироваться в объектной модели VS.

Код, приведенный в листингах 13.8 и 13.9, использует `_applicationObject` для получения списка команд, который представляет собой список всех действий, которые вы можете предпринять в VS. Как говорилось ранее, `_applicationObject` имеет тип `DTE2` и служит родительским объектом для получения доступа к функциональным возможностям VS.

Листинг 13.8. Код на C#, использующий _applicationObject для получения списка команд

```
Commands2 commands =
    (Commands2)_applicationObject.Commands;
```

Листинг 13.9. Код на VB, использующий _applicationObject для получения списка команд

```
Dim commands As Commands2 =
    CType(_applicationObject.Commands, Commands2)
```

В объектной модели автоматизации VS, элемент меню называется `CommandBar`. Таким образом, вы получаете ссылку на коллекцию `CommandBars`, опять же посредством `_applicationObject`, для получения ссылки на объект `MenuBar`, представляющий собой главное меню VS и присвоенный `menuBarCommandBar` (листинги 13.10 и 13.11).

Листинг 13.10. Код на C#, получающий ссылку на коллекцию CommandBars

```
Microsoft.VisualStudio.CommandBars.CommandBar
    menuBarCommandBar = ((
        Microsoft.VisualStudio.CommandBars.CommandBars)
        _applicationObject.CommandBars)["MenuBar"];
```

Листинг 13.11. Код на VB, получающий ссылку на коллекцию CommandBars

```
Dim commandBars As CommandBars =
    CType(_applicationObject.CommandBars, CommandBars)
Dim menuBarCommandBar As CommandBar =
    commandBars.Item("MenuBar")
```

В коллекции `CommandBars`, `menuBarCommandBar`, затем производится поиск коллекции `Controls`, представляющей собой список элементов главного меню. Цель заключается в том, чтобы найти **Tools**, присвоенное `toolsControl` так, как показано в листингах 13.12 и 13.13.

Листинг 13.12. Код на C#, выполняющий поиск меню Tools

```
string toolsMenuName = "Tools";
CommandBarControl toolsControl =
    menuBarCommandBar.Controls[toolsMenuName];
```

Листинг 13.13. Код на VB, выполняющий поиск меню Tools

```
Dim toolsMenuName As String = "Tools"
Dim toolsControl As CommandBarControl =
    menuBarCommandBar.Controls.Item(toolsMenuName)
```

В объектной модели автоматизации VS индивидуальное меню представляет собой объект `CommandBarPopup`, присвоенный переменной `toolsPopup`. На C# это присваивание выглядит так:

```
CommandBarPopup toolsPopup = (CommandBarPopup)toolsControl;
```

На VB это делается так:

```
Dim toolsPopup As CommandBarPopup = CType(toolsControl, CommandBarPopup)
```

Теперь вы имеете ссылку на меню, в которое следует добавить новый пункт меню для нового модуля расширения. Теперь вы готовы добавить команду, используя для этого метод `AddNamedCommand2`. Не забудьте, что ранее рассмотренный код присвоил эти команды из объекта приложения переменной `commands`. Быстрый взгляд на аргументы метода `AddNamedCommand2` даст вам представление о том, что происходит: код передает ссылку модулю расширения, передает имя и описание пункта меню, а также указывает статус команды (поддерживаемая и активная). Элемент меню получит изображения и текст, и команда имеет тип `button` (можно щелкнуть по ней мышью). Если вам нужна подробная информация об этом вызове, самое время просмотреть документацию. Хотя общее понимание основных интерфейсов, таких, как `OnConnection` для `IDTExtensibility2`, имеет очень важное значение, запоминание каждого вызова API не обязательно будет лучшим подходом, особенно на начальных этапах освоения. Вызов метода `AddNamedCommand2` показан в листингах 13.14 и 13.15.

Листинг 13.14. Вызов метода AddNamedCommand2 (код на C#)

```
Command command = commands.AddNamedCommand2(
    _addInInstance, "KeystrokeFinder",
    "KeystrokeFinder",
    "Executes the command for KeystrokeFinder",
    true, 59, ref contextGUIDS,
    (int)vsCommandStatus
         .vsCommandStatusSupported+
    (int)vsCommandStatus.vsCommandStatusEnabled,
    (int)vsCommandStyle
```

```
            .vsCommandStylePictAndText,
        vsCommandControlType
            .vsCommandControlTypeButton);
```

Листинг 13.15. Вызов метода AddNamedCommand2 (код на VB)

```
Dim command As Command =
    commands.AddNamedCommand2(
       _addInInstance, "KeystrokeFinderVB",
        "KeystrokeFinderVB",
        "Executes the command for KeystrokeFinderVB",
        True, 59, Nothing,
        CType(vsCommandStatus.vsCommandStatusSupported,
             Integer) +
        CType(vsCommandStatus.vsCommandStatusEnabled,
             Integer),
        vsCommandStyle.vsCommandStylePictAndText,
        vsCommandControlType.vsCommandControlTypeButton)
```

Метод `AddNamedCommand2` возвращает объект `Command`, `command`, который должен быть помещен среди команд VS, так, чтобы пользователь щелчком мыши мог вызвать ваш модуль расширения. Эту задачу выполняет следующее присваивание, которое добавляет команду в меню **Tools**. Как вы, вероятно, помните из предыдущих примеров, код искал и получил ссылку на меню **Tools**. Убедившись в том, что как `command`, так и `toolsPopup` ссылаются на допустимые объекты (это будет наилучшим подходом), необходимо поместить команду на первую позицию в меню **Tools**. Эту задачу выполняет код, приведенный в листингах 13.16 и 13.17.

Листинг 13.16. Добавление команды в меню Tools (код на C#)

```
if((command != null) &&
    (toolsPopup != null))
{
    command.AddControl(
       toolsPopup.CommandBar, 1);
}
```

Листинг 13.17. Добавление команды в меню Tools (код на VB)

```
command.AddControl(toolsPopup.CommandBar, 1)
```

На этом зона ответственности метода `OnConnection` заканчивается. Если у вас реализован собственный код для реализации модуля расширения, то его размещение в методе `OnConnection` будет хорошим решением. Приведенный пример был полезен тем, что теперь вы знаете, как следует получать доступ к меню и командам VS. Кроме того, этот пример также продемонстрировал и важность главного

объекта `application` и его использование в качестве начальной точки для получения доступа к другим компонентам VS.

Как вы, вероятно, помните, метод `OnConnection` присваивает главный объект приложения, `application`, переменной `_applicationObject`, которая является полем класса `Connect`. Это важно, потому что теперь вы имеете доступ к главному объекту `application`, и в следующем разделе будет продемонстрировано его использование для выполнения вашего дополнительного модуля посредством метода `Exec`.

Реализация метода Exec

Каждый раз, когда пользователь запускает ваш дополнительный модуль, VS вызывает метод `Exec` интерфейса `IDTCommandTarget`. Метод `Exec` играет важную роль, потому что именно сюда вы добавляете код, который реализует поведение вашего модуля расширения. В предшествующих разделах обсуждался код, сгенерированный VS, а вот в листингах 13.18 и 13.19 приведен код метода `Exec`, который вам необходимо ввести самостоятельно, чтобы новый дополнительный модуль **KeystrokeFinder** мог работать. Цель рассматриваемого в этой главе дополнительного модуля заключается в перечислении всех команд VS и ассоциированных с ними клавиатурных комбинаций. Список команд должен выводиться в окно **Output**. Листинги 13.18 и 13.19 содержат код метода `Exec` для дополнительного модуля **KeystrokeFinder**.

Листинг 13.18. Код метода Exec для нового дополнительного модуля KeystrokeFinder (код на C#)

```
public void Exec(
    string commandName, vsCommandExecOption executeOption,
    ref object varIn, ref object varOut, ref bool handled)
{
    handled = false;
    if(executeOption ==
        vsCommandExecOption.vsCommandExecOptionDoDefault)
    {
            if (commandName ==
                "KeystrokeFinder.Connect.KeystrokeFinder")
            {
                OutputWindow outWin =
                   _applicationObject.ToolWindows.OutputWindow;
                OutputWindowPane outPane =
                    outWin.OutputWindowPanes.Add(
                        "Keyboard Shortcuts");
                outPane.Activate();

                foreach (Command cmd in
                        _applicationObject.Commands)
                {
```

```
                    object[] cmdBindings =
                          cmd.Bindings as object[];

                    if (cmdBindings.Length > 0)
                    {
                              string bindingStr =
                                  string.Join(", ", cmdBindings);
                              outPane.OutputString(
                                  "Command: " + cmd.Name +
                                  ", Shortcut: " + bindingStr +
                                  "\n");
                    }
                }

                handled = true;
                return;
            }
        }
}
```

Листинг 13.19. Код метода Exec для нового дополнительного модуля KeystrokeFinder (код на VB)

```
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 =
                "KeystrokeFinderVB.Connect.KeystrokeFinderVB" Then

                 Dim outWin As OutputWindow =
                        _applicationObject.ToolWindows.OutputWindow

                 Dim outPane As OutputWindowPane =
                        outWin.OutputWindowPanes.Add(
                            "Keyboard Shortcuts")

                outPane.Activate()

                For Each cmd As Command In _applicationObject.Commands

                    Dim cmdBindings As Object() =
```

```
                    CType(cmd.Bindings, Object())

                If cmdBindings.Length > 0 Then

                    Dim bindingStr As String =
                        String.Join(", ", cmdBindings)

                    outPane.OutputString(
                       "Command: " & cmd.Name &
                       ", Shortcut: " & bindingStr &
                       Environment.NewLine)

                End If

            Next

            handled = True
            Exit Sub
        End If
    End If
End Sub
```

Параметр `executeOption` метода `Exec` позволяет вам определить, следует ли запросить пользовательский ввод, выполнить действие или отобразить подсказку — все это опции `vsCommandExecOption`. Все, что вам требуется сделать — выполнить проверку опции, а затем выполнить действие, указанное текущим значением `executeOption`. В модуле расширения, рассматриваемом в данном примере, мы проверяем только значение `vsCommandExecOptionDoDefault`, которое указывает на необходимость выполнения операции. На C# эта проверка выполняется так:

```
if(executeOption == vsCommandExecOption.vsCommandExecOptionDoDefault)
```

На VB та же самая проверка выполняется так:

```
If executeOption = vsCommandExecOption.vsCommandExecOptionDoDefault Then
```

Пример, рассматриваемый в этой главе, имеет только одну команду, однако вы впоследствии можете реализовать и дополнительные команды, добавив их в метод `OnConnection`. Поэтому добавьте в код утверждение `if`, чтобы убедиться в том, что выполняете именно ту команду, которую требуется. На C# эта команда выглядит так:

```
if (commandName == "KeystrokeFinder.Connect.KeystrokeFinder")
```

На VB аналогичная команда выглядит так:

```
If commandName = "KeystrokeFinderVB.Connect.KeystrokeFinderVB" Then
```

Как уже неоднократно говорилось ранее, объект `application` представляет собой стартовую точку для доступа ко всем объектам VS. Так как нам необходимо осуществлять вывод в окно **Output**, код получает доступ к свойству `ToolWindows` объекта `application`, который предоставляет доступ ко множеству окон VS. Код,

приведенный в листингах 13.20 и 13.21, получает ссылку на `OutputWindow`, добавляет новую панель и активирует ее.

Листинг 13.20. Код на C#, получающий ссылку на OutputWindow, добавляющий новую панель и активирующий ее

```
OutputWindow outWin =
        _applicationObject.ToolWindows.OutputWindow;
OutputWindowPane outPane =
        outWin.OutputWindowPanes.Add(
                "Keyboard Shortcuts");
outPane.Activate();
```

Листинг 13.21. Код на VB, получающий ссылку на OutputWindow, добавляющий новую панель и активирующий ее

```
Dim outWin As OutputWindow =
    _applicationObject.ToolWindows.OutputWindow
Dim outPane As OutputWindowPane =
    outWin.OutputWindowPanes.Add(
        "Keyboard Shortcuts")
outPane.Activate()
```

Вернувшись к объекту `application`, мы должны получить доступ к коллекции `Commands`, используя цикл `foreach` для получения доступа к каждому объекту `Command`. Имя каждой команды задается свойством `Name`. Свойство `Bindings` представляет собой коллекцию клавиатурных комбинаций для `command`. Некоторые команды не имеют клавиатурных комбинаций, на что указывает пустая коллекция `Bindings` (ее свойство `Length` должно быть установлено на 0), и такие команды мы пропускаем. Код, приведенный в листингах 13.22 и 13.23, иллюстрирует цикл, пробегающий все команды VS и выводящий имя каждой команды и ассоциированные с ней клавиатурные комбинации в окно **Output**.

Листинг 13.22. Код на C#, выводящий в окно Output команды VS и ассоциированные с ними клавиатурные комбинации

```
foreach (Command cmd in
    _applicationObject.Commands)
{
    object[] cmdBindings =
        cmd.Bindings as object[];

    if (cmdBindings.Length > 0)
    {
            string bindingStr =
                string.Join(", ", cmdBindings);
```

```
            outPane.OutputString(
                "Command: " + cmd.Name +
                ", Shortcut: " + bindingStr +
                "\n");
    }
}
handled = true;
```

Листинг 13.23. Код на CV, выводящий в окно Output команды VS и ассоциированные с ними клавиатурные комбинации

```
For Each cmd As Command In _applicationObject.Commands

    Dim cmdBindings As Object() =
        CType(cmd.Bindings, Object())

    If cmdBindings.Length > 0 Then

        Dim bindingStr As String =
            String.Join(", ", cmdBindings)

        outPane.OutputString(
            "Command: " & cmd.Name &
            ", Shortcut: " & bindingStr &
            Environment.NewLine)

    End If

Next

handled = True
```

Обратите внимание, что условие `handled` мы установили на `true`, тем самым давая VS возможность узнать о том, что код распознал команду и выполнил ее. Помимо предоставления пользователям возможности выполнения модуля расширения (Add-In), вам необходимо гарантировать, что команда правильно отображает свой статус, и ее работа построена логично. Этот вопрос мы обсудим в следующем разделе.

Установка статуса команды с помощью QueryStatus

Когда среда разработки VS работает с вашим дополнительным модулем, она должна вызывать метод `QueryStatus` интерфейса `IDTCommandTarget`, чтобы гарантировать правильность отображения команды. Код, приведенный в листингах 13.24 и 13.25, демонстрирует стандартную реализацию метода `QueryStatus`.

Листинг 13.24. Стандартная реализация метода QueryStatus на C#

```
public void QueryStatus(
    string commandName,
    vsCommandStatusTextWanted neededText,
    ref vsCommandStatus status,
    ref object commandText)
{
    if(neededText ==
        vsCommandStatusTextWanted
            .vsCommandStatusTextWantedNone)
        {
             if(commandName ==
                "KeystrokeFinder.Connect.KeystrokeFinder")
            {
                status =
                   (vsCommandStatus)
                    vsCommandStatus.vsCommandStatusSupported|
                    vsCommandStatus.vsCommandStatusEnabled;
                return;
            }
        }
}
```

Листинг 13.25. Стандартная реализация метода QueryStatus на VB

```
Public Sub QueryStatus(
    ByVal commandName As String,
    ByVal neededText As vsCommandStatusTextWanted,
    ByRef status As vsCommandStatus,
    ByRef commandText As Object) Implements IDTCommandTarget.
QueryStatus

    If neededText =
       vsCommandStatusTextWanted.vsCommandStatusTextWantedNone Then

       If commandName =
            "KeystrokeFinderVB.Connect.KeystrokeFinderVB" Then

            status =
                CType(vsCommandStatus.vsCommandStatusEnabled +
                vsCommandStatus.vsCommandStatusSupported,
                vsCommandStatus)
       Else
```

```
            status = vsCommandStatus.vsCommandStatusUnsupported

        End If

    End If

End Sub
```

Метод `QueryStatus`, приведенный в листингах 13.24 и 13.25, проверяет свойство `commandName` с тем, чтобы убедиться, что он работает именно с тем модулем расширения, который требуется. Если это так, он устанавливает параметр `status` на комбинацию значений из перечисления `vsCommandStatus`. В листингах 13.24 и 13.25 команда имеет статус поддерживаемой и активной.

Итак, в рассмотренном примере было продемонстрировано создание модуля расширения. В следующем разделе будет продемонстрировано развертывание модулей расширения.

Развертывание модуля расширения

В развертывании дополнительных модулей принимают участие два файла *.AddIn и *.dll. Файл *.AddIn содержит регистрационную информацию для вашего модуля расширения, а *.dll представляет собой выходную сборку библиотеки классов, которая содержит сам модуль расширения.

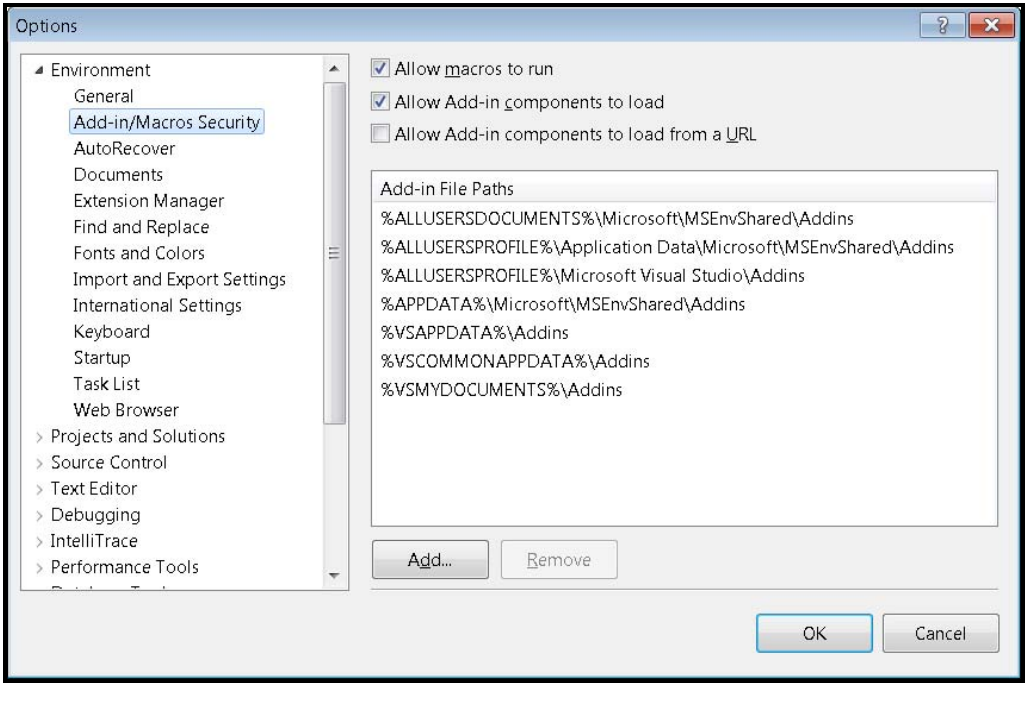


Рис. 13.8. Опции безопасности для макросов и дополнительных модулей

Чтобы выполнить развертывание дополнительного модуля, скопируйте файл *.AddIn в папку, известную VS. Существует специальный набор папок, известных VS, но вы можете добавить в их состав и собственные папки. Чтобы просмотреть настройки VS, выберите из меню команды **Tools** | **Options**, в раскрывшемся окне **Options** выберите опции **Environment** | **Add-in/Macros Security** (рис. 13.8). В этом окне имеются и опции, определяющие, возможен ли запуск макросов, можно ли загружать дополнительные модули (Add-Ins), а также можно ли загружать дополнительные модули через Интернет.

В дополнение к файлу *.AddIn вам необходимо указать, где должна располагаться библиотека классов для модуля расширения. По умолчанию программа-мастер Add-In Project Wizard предполагает, что файлы *.dll располагаются там же, где и файлы *.AddIn. В листинге 13.26 показано содержимое файла *.AddIn. Расположение файла *.dll задается элементом `Assembly`, который может указывать путь в файловой системе или URL.

Листинг 13.26. Содержимое файла *.AddIn

```
<?xml version="1.0" encoding="UTF-16" standalone="no"?>
<Extensibility xmlns=
"http://schemas.microsoft.com/AutomationExtensibility">
    <HostApplication>
        <Name>Microsoft Visual Studio</Name>
        <Version>10.0</Version>
    </HostApplication>
    <Addin>
        <FriendlyName>Keystroke Finder</FriendlyName>
        <Description>
Displays a List of VS Shortcut Keystrokes.
        </Description>
        <AboutBoxDetails>
Creating an Add-...
        </AboutBoxDetails>
        <AboutIconData>...</AboutIconData>
        <Assembly>KeystrokeFinder.dll</Assembly>
        <FullClassName>
KeystrokeFinder.Connect
        </FullClassName>
        <LoadBehavior>1</LoadBehavior>
        <CommandPreload>1</CommandPreload>
        <CommandLineSafe>0</CommandLineSafe>
    </Addin>
</Extensibility>
```

Еще один способ работы с дополнительными модулями предоставляет инструмент Add-In Manager, который можно запустить, выбрав из меню команды **Tools** | **Add-in Manager**. На рис. 13.9 показано окно **Add-In Manager** с новым модулем расширения **KeystrokeFinder** в списке доступных дополнительных модулей. Уста-

новка флажка для доступных модулей расширения немедленно загружает их (сброс флажков, соответственно, выгружает). Если установить флажок в столбце **Startup**, то дополнительный модуль должен загружаться при запуске VS. Если установить флажок в столбце **Command Line**, то дополнительный модуль будет загружаться и в том случае, когда пользователь запускает VS из командной строки командой `devenv.exe`.

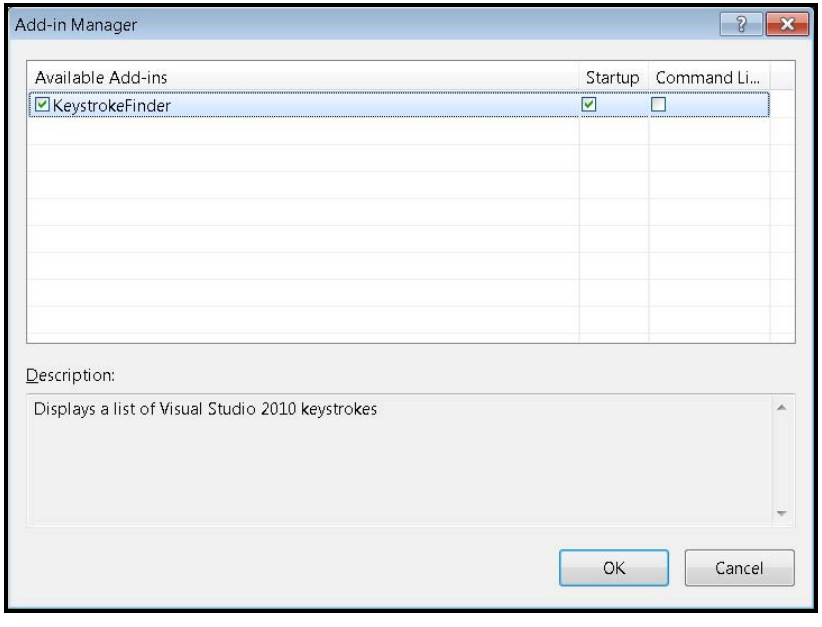


Рис. 13.9. Окно **Add-In Manager**

Когда дополнительный модуль будет развернут и загружен, пользователь сможет запускать его, выбрав из меню команды **Tools | KeystrokeFinder**. Когда модуль расширения работает, окно **Output** будет отображать список команд и соответствующих им клавиатурных комбинаций. Чтобы просмотреть результаты, откройте окно **Output**, нажав клавиатурную комбинацию <CTRL>+<W>+<O> перед запуском модуля расширения.

Итак, теперь вы знаете, как создавать и развертывать модули расширения. Однако для дальнейшего изучения вам необходимы некоторые инструкции и советы, без которых написать собственный модуль расширения будет довольно затруднительно. Эти советы будут даны в следующем разделе.

В каком направлении двигаться дальше

Как вы уже знаете из предыдущих разделов, объект `application` является центральным объектом в разработке дополнительных модулей, расширяющих функциональные возможности VS. Каждый раз, когда вам требуется найти что-либо, пользуйтесь справочной информацией по объекту `application`, нажмите на кла-

виатуре символ точки (.), и технология Intellisense предложит вам различные свойства — например, команды и окна.

При просмотре свойств объекта `application` сверяйтесь с документацией VS, где подробно рассказано о том, что означает каждое свойство, и предоставлены примеры кода, иллюстрирующего, как это свойство работает. Иногда, впрочем, может оказаться и так, что в документации нет примеров кода, а сам текст написан не таким уж простым и понятным языком, как хотелось бы. В таких случаях, вам может потребоваться провести самостоятельную исследовательскую работу. В вашем распоряжении будут такие инструменты, как точки останова в отладчике и окно **Immediate.** Установите точку останова на один из методов модуля расширения и изучите значение объекта. Чтобы определить, что содержится внутри объекта, откройте окно **Immediate**, введите имя объекта, введите символ точки (.), и функция Intellisense поможет вам найти нужные вам свойства.

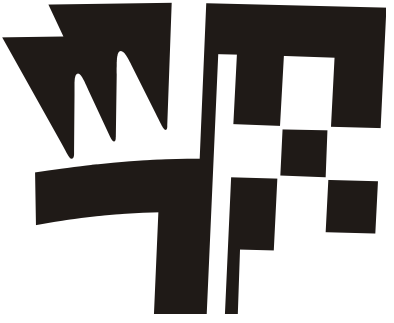
В некоторых ситуациях вы будете иметь дело со свойствами, которые представляют собой коллекции. В этом случае вы можете написать код метода, необходимого для получения доступа к коллекции, и добавить в него цикл `foreach` (`For Each` — для VB), и выводить значения членов коллекции в окно **Output.**

Заключение

В данной главе вы пошагово прошли процедуру, необходимую для написания собственного модуля, расширяющего функциональные возможности VS. В разделах этой главы последовательно описывалось, как запустить новый проект дополнительного модуля. Этот процесс похож на процедуру создания проектов других типов, за исключением того, что программа-мастер, создающая такие проекты, задает больше вопросов. Далее были описаны роли всех компонентов проекта и было показано содержимое самого модуля расширения, его методов и интерфейсов, а также рассмотрен код, представляющий собой "каркас" будущего модуля расширения, автоматически генерируемый программой-мастером Add-In Project Wizard. Наконец, в заключение этой главы мы обсудили код, который добавляется в состав модуля расширения, который выводит в окно **Output** список команд и ассоциированных с ними клавиатурных комбинаций. Этот процесс был призван продемонстрировать, каким образом можно программно получать доступ к различным компонентам VS. Наконец, заключительные разделы этой главы были посвящены развертыванию дополнительных модулей и управлению ими, а также были даны советы и рекомендации по дальнейшему изучению доступных вам возможностей программирования модулей расширения.

Эта глава — последняя в данной книге, но она одновременно представляет собой и отправную точку в длинном и интересном пути изучения возм+ожностей программирования с помощью Microsoft Visual Studio 2010. Я искренне надеюсь, что книга вам понравится и поможет вам приобрести новые знания и навыки. Успехов вам на этом пути!

Джо Майо (Joe Mayo)



ПРИЛОЖЕНИЯ

Приложение 1

Введение в XML

Расширяемый язык разметки (Extensible Markup Language, XML) представляет собой открытый стандарт для кросс-платформенного обмена документами. Изначально язык XML применялся для представления данных, но с тех пор он уже давно перерос эту планку и теперь включает в свой состав не только технологии реализации пользовательских интерфейсов, но даже и логику исполнения программы. Хотя язык XML нашел себе множество областей практического применения, в данной книге основное внимание уделяется его использованию с такими технологиями, как ASP.NET, Silverlight и Windows Presentation Foundation (WPF), которые обсуждались в книге. В основном, в данной книге обсуждается использование XML для построения пользовательских интерфейсов, хотя в комбинации с каждой из перечисленных технологий и областей применения работа с кодом XML имеет свою специфику. Например, в ASP.NET язык XML используется в комбинации с HTML (XHTML). Такие технологии, как Silverlight и WPF, используют XML Application Markup Language (XAML), название которого произносится "Zamel." Прежде чем изучать XHTML или XAML, необходимо краткое введение в основы языка XML. Таким введением и призвано послужить данное приложение. Если вы уже знакомы с XML, то это приложение можно рассматривать в качестве краткого справочника, который поможет освежить в памяти информацию, необходимую для понимания материалов, излагаемых в книге. Хотя это краткое введение в XML не даст вам полной информации, оно все же должно помочь вам понять основные принципы использования этого языка при работе в Visual Studio 2010.

Редактор кода XML в VS 2010

VS 2010 позволяет разработчикам создавать собственные документы XML. Для этой цели предоставляется встроенный редактор кода XML (XML editor). Существуют различные способы открыть новый документ XML как в рамках проекта, так и независимо, без создания какого-либо проекта. Если вы просто хотите создать новый файл формата XML вне зависимости от какого бы то ни было проекта, выберите из меню команды **File** | **New** | **File**, а затем выберите опцию **XML File** и нажмите кнопку **OK**. При сохранении файл можно переименовать (например, Customer.xml). Чтобы отредактировать файл формата XML в рамках проекта, выберите из меню опции **Add** | **New Item**, затем выберите список **Data**, выберите опцию

XML File, присвойте имя новому файлу (например, Customer.xml) и щелкните мышью по кнопке **OK**. Откроется окно редактора со встроенной поддержкой Intellisense — это явно лучше, чем Notepad. В листинге П1.1 показано содержимое документа XML, который содержит данные о клиентах.

Листинг П1.1. Пример документа XML

```
<?xml version="1.0" encoding="utf-8" ?>
<customer id="7">
    <name>Joe</name>
    <address>123 4th St</address>
</customer>
```

Как видно из примера, приведенного в листинге П1.1, документ XML представляет собой читаемый текст. Он содержит данные, и смысл этих данных специфичен для приложений, которые их используют. В последующих разделах данного приложения мы подробно разберем содержимое листинга П1.1 и дадим пояснения к каждой из его частей.

Префиксы XML

В начале документа, приведенного в листинге П1.1, содержится префикс XML, выглядящий следующим образом:

```
<?xml version="1.0" encoding="utf-8" ?>
```

По этому префиксу приложения, читающие документ, могут убедиться в том, что это действительно документ XML. Номер версии является самоочевидным. Кодировка важна, потому что она указывает формат текста на двоичном уровне. Если одно из ваших приложений должно передавать данные другому, важно, чтобы оба приложения, во-первых, могли прочитать документ и, во-вторых, использовали одну и ту же кодировку. Кодировка utf-8 используется по умолчанию, и для целей данной книги ее достаточно.

Угловые скобки, < и >, определяют разметку XML. Для префикса файла, содержимое заключается между последовательностями символов <? и ?>, но, как будет показано в последующих разделах, большинство других элементов разметки отличаются.

Элементы XML

Элементами XML, представленными в листинге П1.1, являются `customer`, `name` и `address`. Каждый элемент определяется парой соответствующих друг другу (открывающий и закрывающий) символов разметки. В общем случае, разметка соответствует следующему образцу:

```
<elementName>value</elementName>
```

В только что приведенном примере `elementName` — это имя элемента XML, а `value` — это данные, ассоциированные с указанным элементом. Элементы всегда имеют открывающий и закрывающий тэги. Закрывающий тэг всегда следует за открывающим (между парой тэгов может существовать еще одна пара вложенных тэгов, которые определяют вложенный элемент). Закрывающие тэги всегда содержат наклонную косую черту, которая предшествует имени элемента.

Значение в только что приведенном примере иногда может быть пустым. Пустое значение указывает на то, что элементу не присвоено никакого значения. Значение может состоять из одного или нескольких элементов, например, в листинге П1.1 есть значение `Customer`, которое содержит элементы `name` и `address`. В листинге П1.1 поле имени имеет значение `Joe`, а поле адреса — значение `123 4th St`. В дополнение к элементам вы можете устанавливать атрибуты. Об атрибутах рассказывается в следующем разделе.

Атрибуты

Атрибут присваивается элементу, имеющему единственное значение, например:

```
<elementName attributeName="attributeValue">
    elementValue
</elementName>
```

Обратите внимание, что атрибут `attributeName` находится за открывающим тэгом элемента. Он содержит знак равенства и присваиваемое значение, которое заключается в кавычки. Одному элементу можно присвоить множество атрибутов, которые в этом случае должны быть отделены друг от друга пробелами. Не забывайте, что атрибуты могут иметь только одно значение, а если вам требуется определить множество значений, то нужно пользоваться элементами.

Примерами атрибутов в листинге П1.1 являются `version` и `encoding` в префиксе, а также `id` для элемента `customer`.

Пространства имен

Еще одной важной частью языка XML, которую необходимо хорошо понимать, являются пространства имен. В *главе 2* рассказывалось о том, как пространства имен (namespaces) в C# и VB помогают уникальной идентификации кода в пределах заданного пространства имен. В XML пространства имен имеют схожее предназначение. В случае с примером, приведенным в листинге П1.1, существует элемент `customer`, но с данными этого элемента работает множество различных программ. В разных программах объект `customer` может быть определен по-разному, и вам необходимы средства, позволяющие различать эти объекты. Здесь вам и придут на помощь пространства имен. Вам необходимо определить ваш объект `customer` в пределах пространства имен по вашему выбору. То же самое должны делать и другие разработчики — определять уникальное пространство имен для своих объектов `customer`. Таким образом, путаницы между вашими программами не возникнет, даже если они и сделают попытку чтения не тех данных. В листинге П1.2 продемонстри-

ровано, каким образом следует использовать пространства имен, чтобы сделать свой объект `customer` уникальным.

> ***Совет***
>
> Наверняка вы уже заметили, что пространства имен в листинге П1.2 выглядят как Web-адреса. Однако это просто совпадение. Такое использование пространств имен представляет собой общепринятую практику, применяемую для того, чтобы повысить шансы на то, что пространство имен будет уникальным. В действительности, пространство имен представляет собой только строку, которая привлекает внимание новичков. Например, **http://mcgraw-hill.com/vs2010bg** представляет собой иное пространство имен, отличное от **http://mcgraw-hill.com/vs2010bg/** всего лишь из-за лишнего символа наклонной косой черты на конце второй строки. Таким образом, если вы допустите эту ошибку, возможно, что ваша программа не распознает данные как имеющие правильный формат только из-за того, что эти данные находятся в другом пространстве имен, отличном от того, которое ожидает "увидеть" ваша программа. Не забывайте, что пространство имен — это уникальная строка, а не Web-адрес.

Листинг П1.2. Пример пространства имен XML

```
<?xml version="1.0" encoding="utf-8" ?>
<customer id="7"
    xmlns="http://mcgraw-hill.com/vs2010bg"
    xmlns:a="http://somedomain.com/addresses">
    <name>Joe</name>
    <a:address>123 4th St</a:address>
</customer>
```

Пространства имен указываются путем помещения атрибута элемента `xmlns`, с префиксом или без него. Атрибут `xmlns` без префикса указывает пространство имен по умолчанию для всех элементов, где располагается пространство имен, и для всех дочерних элементов, где располагается пространство имен. Это значит, что объекты `customer` и `name` находятся в пространстве имен **http://mcgraw-hill.com/vs2010bg**.

Пространства имен могут иметь префиксы, что помогает указать области, применительно к которым они действуют В примере, приведенном в листинге П1.2, имеется тэг `xmlns:a`, где `a` — это префикс для пространства имен **http://somedomain.com/addresses**. Удобство префиксов в том, что они помогают сделать код XML более удобочитаемым. В листинге П1.2 пространство имен `address` выделяется префиксом `a:`, в форме `<a:address>`, чтобы подчеркнуть, что адрес принадлежит к пространству имен **http://somedomain.com/addresses**. Без префикса вам пришлось бы писать элемент `address` в следующей неудобочитаемой форме:

```
< http://somedomain.com/addresses:address>
 123 4th St
</ http://somedomain.com/addresses:address>
```

Разрывы строки здесь добавлены исключительно для удобочитаемости, но на практике читается только само значение, но не пробелы или разрывы строк.

Меню XML

Когда вы открываете в VS документ XML, вы увидите меню XML, в котором есть опции для запуска, отладки и профилировки документов XML Transformation (XSLT) и для работы со схемами. XSLT используется работающей программой или утилитой для перевода документа XML из одной формы в другую. Схема XML — это документ XML, который описывает допустимый формат другого документа XML. Для документа XML схема XML представляет собой то же самое, что определение таблицы SQL для хранящихся в ней данных. Как XSLT, так и схемы выходят за рамки данной книги, а здесь они просто упомянуты с тем, чтобы вы знали, какие средства вам, в принципе, доступны на тот случай, когда они вам понадобятся.

Конфигурирование опций XML Opt

Выберите из меню команды **Tools** | **Options**, чтобы открыть окно VS **Options**. В окне **Options** выберите узел **Text Editor XML**, и вы сможете сконфигурировать множество опций, ассоциирующихся с написанием документов XML. Например, вы можете включить опцию нумерации строк или опцию форматирования тегов.

Заключение

Итак, в данном приложении вы получили необходимый минимум информации для работы с XML в VS. Теперь вы знаете, как создаются документы XML, и знакомы с их базовыми компонентами — префиксами (prefixes), элементами (elements), атрибутами (attributes) и пространствами имен (namespaces). Кроме того, вы ознакомились с поиском опций, позволяющих выполнить индивидуальную настройку ваших документов XML — включая их редактирование. XML представляет собой фундамент, на котором основываются XAML и XHTML, которые будут рассмотрены в следующем приложении. Наконец, информация, приведенная в этом приложении, необходима для понимания материалов, излагаемых в главах самой книги.

Приложение 2

Введение в XAML

Язык XAML (XML Application Markup Language), произносится как "Zamel", представляет собой основанный на XML язык, предназначенный для построения пользовательских интерфейсов. XAML применяется при построении приложений Windows Presentation Foundation (WPF) и Silverlight. Приложения WPF предназначены для работы на настольных компьютерах, в то время как Silverlight — это технология для разработки Web-приложений. Впрочем, WPF и Silverlight имеют между собой много общего, особенно в той части, которая касается программирования с использованием XAML. Следовательно, данное приложение, вкратце описывающее язык XAML, призвано помочь вам в проектировании пользовательских интерфейсов, с применением макетов, общих для WPF и Silverlight. Если вы еще не знакомы с XAML, то это приложение будет очень полезно прочесть перед тем, как приступать к чтению глав, посвященных разработке приложений WPF и Silverlight, с тем, чтобы вы могли извлечь максимум пользы, работая с этими технологиями. В целях простоты изложения, чтобы не отвлекать ваше внимание на мелкие и не всегда нужные подробности, я буду демонстрировать применение XAML на примере разработки приложения WPF. Однако следует иметь в виду, что все, о чем рассказывается в этом приложении, применимо не только по отношению к WPF, но и по отношению к Silverlight. Прежде чем приступать к чтению данного приложения, внимательно прочтите приложение 1, которое должно послужить для вас кратким введением в язык XML, что упростит для вас понимание базовых синтаксических правил языка XML.

Запуск проекта WPF

По мере того как вы читаете эту книгу, посвященную разработке приложений с помощью VS, вы постепенно начнете набирать опыт работы с кодом на XAML в среде разработчика VS (VS IDE). Как уже говорилось ранее, для описания общих принципов работы с кодом XAML мы будем использовать проект приложения WPF, просто потому, что в его состав входит меньшее количество файлов, и по своей структуре проекты WPF проще, чем проекты Silverlight.

Чтобы начать работу над новым проектом WPF, выберите из меню команды **File | New | Project** и в раскрывшемся окне **New Project** выберите опцию **WPF Application**. Дайте новому проекту любое имя по своему усмотрению и нажмите

кнопку **OK**. Когда откроется новый проект, в его составе вы обнаружите файл Window1.xaml. Содержимое этого файла будет примерно таким, как показано в листинге П2.1.

Листинг П2.1. Новый файл формата XAML, создаваемый при запуске нового проекта WPF

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">

    <Grid>
    </Grid>
</Window>
```

В VS макет, предлагаемый для Window1.xaml по умолчанию, отображается в верхней половине окна, где расположена область визуального конструирования пользовательского интерфейса. В нижней части окна находится область кода XAML. Вы можете полностью занять все окно для просмотра документа XAML. Для этого подхватите мышью верхнюю границу области кода XAML и перетяните ее до верхней границы окна. После этого все окно будет представлять собой область редактора кода XAML. Первое, на что следует обратить внимание при просмотре кода, приведенного в листинге П2.1, это то, что сам документ представляет собой XML-файл, содержащий элементы (elements), атрибуты (attributes) и пространства имен (namespaces). Каждый из элементов имеет свое предназначение, которое будет описано в последующих разделах.

Элементы как классы

Чтобы документ XAML представлял собой осмысленный код, элементы должны быть ассоциированы с классами. Элемент `Window` в листинге П2.1 ассоциирован с классом, который называется `WpfApplication1.MainWindow`, на что указывает атрибут `x:Class`. Префикс `x` задает псевдоним для пространства имен `http://schemas.microsoft.comwinfx/2006/xaml`, где определен атрибут `Class`. Устанавливая соответствие между элементом и классом, вы даете VS возможность компиляции XAML в исполняемый код.

Обратите внимание, что пространством имен по умолчанию является пространство имен `http://schemas.microsoft.com/winfx/2006/xaml/presentation`, которое определяет, как элементы с префиксами будут компилироваться в код. Важно понимать, что при написании кода XAML, вы создаете документ, который в ходе компиляции будет транслироваться в исполняемый код.

Атрибуты как свойства

Атрибуты `Title`, `Height` и `Width` представляют собой атрибуты элемента `Window` в листинге П2.1. Когда VS компилирует XAML, каждый из атрибутов элемента транслируется в свойство класса, в который транслируется сам этот элемент. Говоря более конкретно, класс `WpfApplication1.MainWindow` будет иметь свойства `Title`, `Height` и `Width`. Для каждого из свойств будет установлено значение, присвоенное соответствующему атрибуту.

Исполнение документа XAML

Имейте в виду, что данное приложение не является кратким учебным курсом по WPF, а напротив, его основная цель заключается в объяснении принципов работы XAML. Тем не менее, данное приложение содержит достаточно информации, чтобы помочь вам понять, что происходит, когда код XAML компилируется и исполняется. Нажмите клавишу <F5> или щелкните мышью по кнопке **Start Debugging** на инструментальной панели, чтобы запустить эту программу. В результате вы увидите окно, подобное тому, что представлено на рис. П2.1.

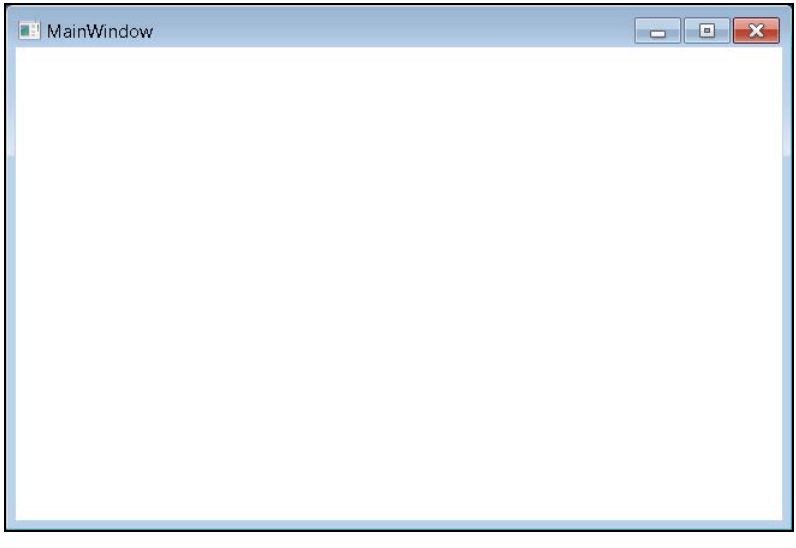


Рис. П2.1. Исполнение XAML

На рис. П2.1 показано, как исполняется элемент `Window`, создающий обычное окно приложения со строкой заголовка, кнопками **minimize** и **close**, а также рамками. Кроме того, на данной иллюстрации вы можете наблюдать результаты применения атрибутов оконного элемента (Window), где в строке заголовка отображается заголовок окна (**MainWindow**), а размеры окна задаются атрибутами `Height` и `Width`.

Этот пример иллюстрирует мощь и гибкие возможности XAML, благодаря которым вы можете получить отнюдь не примитивный результат, самостоятельно не написав ни единой строчки кода на C# или VB. Конечно, весь код XAML транслируется в исполняемый код, но декларативная природа XAML позволяет вам указать, какие результаты вы хотели бы получить, без необходимости указывать, как именно они должны быть получены. XAML позволяет вам экономить время, которое в противном случае для получения эквивалентных результатов должно было бы быть потрачено на сложное программирование. Вместо этого вы без малейших усилий получаете работоспособный исполняемый код.

Элементы свойств

Итак, вы уже видели, как атрибуты преобразуются в свойства. В дополнение к атрибутам, XAML имеет элементы свойств (property elements), являющиеся дочерними элементами, причем свойству могут быть назначены один или несколько других элементов. Примером элемента свойства может служить свойство `Content` командной кнопки (`Button`). Кнопка (`Button`) представляет собой класс как в WPF, так и в Silverlight. Щелчком мышью по кнопке пользователь обычно производит то или иное действие в вашей программе. Свойство `Content` класса Button определяет то, что видит пользователь. Чтобы описать разницу между атрибутом свойства (property attribute) и элементом свойства (property element), рассмотрим пример, в котором имеется и то, и другое. В качестве примера возьмем свойство `Content` класса `Button`.

В листинге П2.2 показан класс `Button` со свойством `Content`, установленным как атрибут.

Листинг П2.2. Класс Button со свойством Content, установленным как атрибут

```
Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Button Content="Click Me" />
    </Grid>
</Window>
```

Как видите, в листинге П2.2 окно (`Window`) содержит элемент `Button`, у которого есть атрибут `Content`, содержащий текст. На рис. П2.2 показано, как выглядит это окно во время исполнения.

Мощной возможностью XAML являются элементы свойств, которые позволяют осуществлять сложную разметку, которая будет присвоена свойству класса. В случае с классом `Button`, мы расширяем свойство `Content` до элемента свойства (property element), чтобы продемонстрировать, как с помощью XAML добавить

другое содержимое, а не только текст. В листинге П2.3 приведен расширенный пример разметки в классе `Button` из листинга П2.2. Теперь разметка расширена таким образом, чтобы содержать изображение вместо текста. В тексте для повышения удобочитаемости я добавил разрыв строки для значения атрибута `Source`.

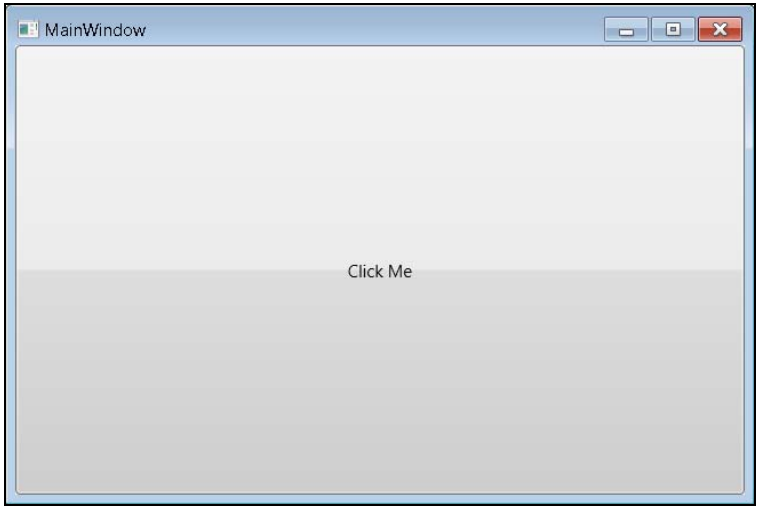


Рис. П2.2. Кнопка (Button) с атрибутом Content, установленным как текст

Листинг П2.3. Расширенная разметка (теперь кнопка вместо текста содержит изображение)

```
<Button>
    <Button.Content>
        <Image Source=
           <Image Source= "C:\Work\Emperor_Penguins.jpg" />
    </Button.Content>
</Button>
```

Вместо установки атрибута `Content` в рассматриваемом примере используется синтаксис элемента свойства (property element), где дочерний элемент называется <`parentElementName.attributeName`>. Преимущество синтаксиса с использованием элемента `property` в коде из листинга П2.3 заключается в том, что теперь свойство `Content` установлено таким образом, чтобы отображать картинку. При использовании синтаксиса с применением атрибутов, вы были ограничены только текстом, а применение синтаксиса элемента `property` позволяет вам поместить на кнопку все, что угодно, например, картинку. Конечно, вы можете воспользоваться картинкой, несущей смысловую нагрузку, а изображение, добавленное мною, служит только для демонстрации возможностей. На рис. П2.3 показана новая кнопка с изображением.

Рис. П2.3. Кнопка со свойством `Content` может отображать картинки

Совет

VS предоставляет в распоряжение пользователей редактора XAML поддержку Intellisense. Вы можете поместить курсор между открывающим и закрывающим тэгами, нажать клавишу <ENTER> и задать отступ начальной позиции курсора между начальным и конечным тэгами. Начиная с этого момента, вы можете ввести символ угловой скобки (<) и начать пользоваться возможностями Intellisense, чтобы выбирать нужные вам элементы и атрибуты, позволяющие реализовать синтаксис элемента `property`.

Расширения разметки

Еще одна возможность расширения, предлагаемая XAML, — это расширения разметки (markup extensions), которые позволяют вам устанавливать атрибуты для ссылки на другие значения. Обычное применение для расширений разметки заключается в привязке данных и использовании ресурсов. Привязка данных (Data binding) — это обычная практика ассоциирования данных с элементом управления пользовательского интерфейса. Например, если вам необходимо отобразить на экране регистрационную запись клиента, вы можете привязать каждое свойство объекта `customer` к определенным областям экрана. Так, например, имя клиента можно привязать к текстовому полю (`TextBox`) на экране. Примеры привязки данных к элементам управления пользовательского интерфейса были приведены в *главе 8*, посвященной разработке приложений WPF, и *главе 10*, посвященной разработке приложений Silverlight. На текущий момент, мы сконцентрируем внимание на расширениях разметки и рассмотрим пример, в котором осуществляется привязка ресурса к элементу.

Ресурсом называется тип объекта или значения, которые могут использоваться многими элементами управления. Например, вы можете определить специальный цвет для кнопок на вашем экране в одном фрагменте кода, а затем использовать расширения разметки, чтобы из всех кнопок сослаться на один и тот же ресурс. Таким образом, вы можете изменить цветовой ресурс в одном месте, и все кнопки, ссылающиеся на этот ресурс, изменят свой цвет автоматически. В листинге П2.4 определен ресурс типа "кисть" (brush resource) конкретного цвета и демонстрируется способ ссылаться на эту кисть из множества кнопок при помощи расширения разметки.

Листинг П2.4. Применение расширений разметки при работе с ресурсами

```
<Window x:Class="WpfApplication1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <SolidColorBrush x:Key="ButtonBrush" Color="Yellow" />
    </Window.Resources>
    <StackPanel>
        <Button Background="{StaticResource ResourceKey=ButtonBrush}"
Content="Button One" />
        <Button Background="{StaticResource ResourceKey=ButtonBrush}"
Content="Button Two" />
    </StackPanel>
</Window>
```

Элемент `Window.Resources` в листинге П2.4 представляет собой элемент свойства окна. Он содержит элемент `SolidColorBrush` со свойством `Color`, установленным на `Yellow`. Все интерфейсные элементы в WPF и Silverlight отрисовываются кистями (brushes), которые определяют цвета, градиенты, изображения, медийные файлы или орнаменты. В данном случае мы обошлись единственным цветом, для чего вполне подходит элемент `SolidColorBrush`. "Фокус" здесь не в том, что такое "кисть", а в том, что кисть — это ресурс, с помощью которого мы можем продемонстрировать использование расширений разметки, применяемых для доступа к этому ресурсу. Важно присвоить ключ каждому ресурсу, поскольку этот ключ будет использоваться расширениями разметки для идентификации этого ресурса.

Расширение разметки присваивается атрибутам `Background` элементов `Button`, как показано в листинге П2.4. Расширения разметки заключены в фигурные скобки. Внутри фигурных скобок указывается тип расширения и ассоциированные с ним атрибуты. В листинге П2.4 используется тип расширения `StaticResource`, благодаря чему вы можете ссылаться на этот ресурс. Атрибут `ResourceKey` расширения `StaticResource` указывает конкретный ресурс, который должен использоваться. Значение `ButtonBrush` соответствует ключу ресурса `SolidColorBrush`. Таким образом, значение атрибута `BackGround` элементов `Button` — `StaticResource` для ресурса `SolidColorBrush`, цвет которого установлен на желтый (`Yellow`). Это означает, что кнопки будут желтыми.

Чтобы оценить удобство использования ресурсов, представьте себе ситуацию, в которой вам пришлось бы отдельно устанавливать атрибут `BackGround` для каждой кнопки вместо того, чтобы воспользоваться расширением разметки `StaticResource`. Далее, подумайте, какой объем работы вам придется проделать, если вам вдруг захочется изменить фон всех кнопок, вручную перекодируя каждую из них. Однако расширение разметки `StaticResource` позволяет изменить цвет для ресурса `SolidColorBrush`, и после этого атрибут `BackGround` для всех кнопок изменится автоматически, не требуя никакой дополнительной работы. На рис. П2.4 показано окно с двумя кнопками. Попробуйте ввести код из листинга П2.4 в редактор XAML, откомпилируйте и запустите приложение и убедитесь в том, что кнопки действительно желтые!

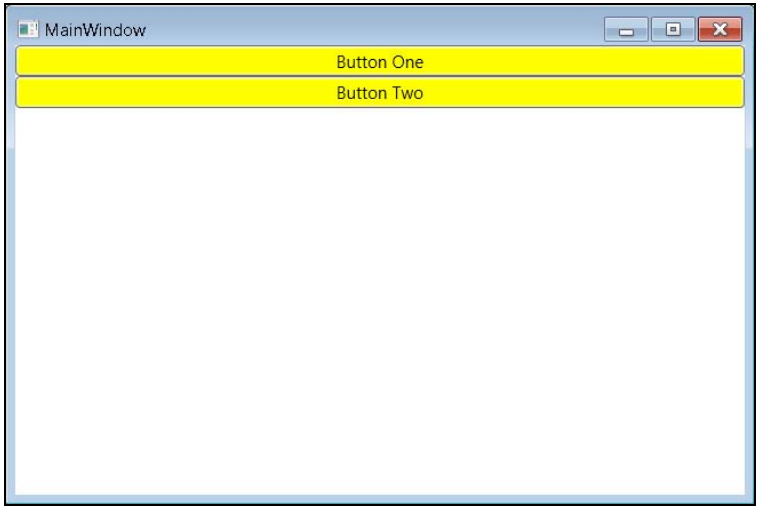


Рис. П2.4. Две кнопки, использующие один и тот же ресурс за счет расширений разметки

Заключение

Это небольшое приложение представляет собой краткое введение в XAML — особый тип документов XML, который применяется для построения пользовательских интерфейсов при разработке приложений WPF и Silverlight. После прочтения этого введения вы должны научиться устанавливать соответствия между элементами и классами, а также между атрибутами и свойствами классов. Кроме того, вы должны были научиться указывать элементы свойств (property elements) таким образом, чтобы получить больше возможностей контроля над пользовательским интерфейсом, нежели за счет использования одних только атрибутов. Наконец, вы узнали о синтаксисе расширений разметки и рассмотрели пример применения расширения разметки `StaticResource` для ссылки на ресурсы. Теперь вы должны быть готовы к восприятию глав о разработке приложений WPF и Silverlight, где XAML интенсивно используется для разработки пользовательских интерфейсов.

Приложение 3

О русской версии Visual Studio 2010

Официальный релиз Microsoft Visual Studio 2010 и .NET Framework 4 состоялся 12 апреля 2010 года. Мероприятие было глобальным, и запуск почти синхронно состоялся по всему земному шару. В России на мероприятии, посвященном запуску Microsoft Visual Studio 2010, была анонсирована и русская версия этого продукта, которая официально была представлена Microsoft на конференции ReMIX (**http://remix.ru/**). Visual Studio 2010 стала первой и пока единственной полностью локализованной на русский язык средой разработки. На русский язык были переведены все ее компоненты, в том числе:

- Обновленный интерфейс (рис. П3.1), включая технологию автозаполнения Intellisense, отладчик и всплывающие подсказки;
- Документация по всей платформе разработки Microsoft — библиотека MSDN (**http://msdn.microsoft.com/ru-ru/library/default.aspx**), которая распространяется бесплатно и доступна в режиме онлайн;
- Сервер для организации командной разработки Team Foundation Server 2010;
- Локализованы все редакции Visual Studio 2010. Следующие версии и обновления будут также выходить на русском языке.

Надо ли говорить о том, какого колоссального труда это потребовало? По данным самой корпорации Microsoft, которые были озвучены на официальной презентации, было в совокупности переведено более 20 миллионов слов! Над этим проектом работали около 1500 лингвистов и IT-специалистов, в числе которых были представители российского сообщества разработчиков и сферы высшего образования.

Русская версия Visual Studio 2010 была выложена в открытый доступ 21 мая, и одновременно с этим для загрузки стали доступны пробные и Express-версии (**http://www.microsoft.com/express/Downloads/**). В настоящее время русские версии Visual Studio 2010 доступны в таких программах, как

WebsiteSpark (**http://www.microsoft.com/web/websitespark/default.aspx**),

DreamSpark (**http://dreamspark.ru/**) и

BizSpark (**http://ms-start.ru/Programs/BizSpark.aspx**).

Впрочем, некоторые программисты не слишком хорошо понимают, зачем нужно локализовать интерфейс IDE. Вот только одна типичная цитата с форумов: "Я не совсем понимаю, для чего разработчику это нужно, ладно документация, но интерфейс среды-то зачем переводить, там же ничего непонятно будет по-русски".

Понять их можно (ведь сколько до этого было не совсем удачно локализованных версий самых разных программных продуктов). Однако в данном случае эти опасения не совсем обоснованны. На начальном этапе, конечно же, были и баги — вот, например, статья "26 багов в интерфейсе и локализации Visual Studio 2010 Rus" (**http://www.net-next.ru/habranews/37687.html**). В принципе, их наличие и послужило причиной того, что при подготовке книги использовалась все-таки английская, а не русская версия.

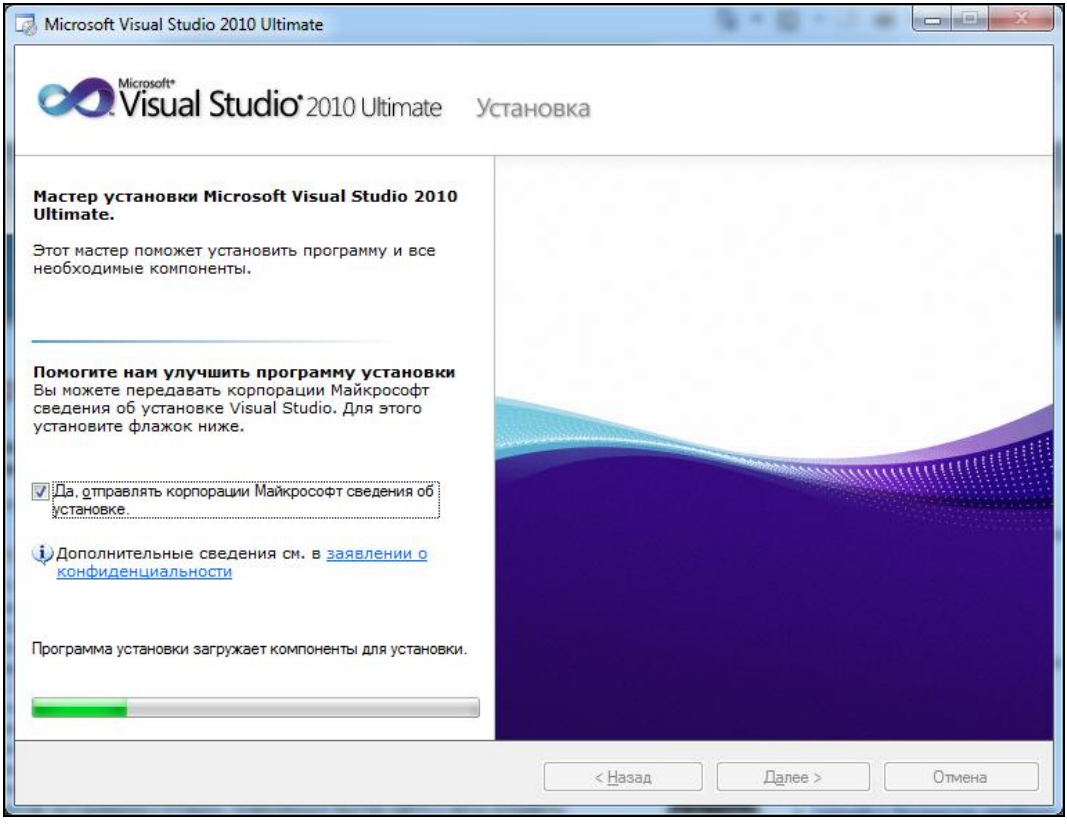


Рис. П3.1. Начиная с самой процедуры инсталляции Microsoft Visual Studio 2010 локализованы все компоненты среды разработчика

Но к настоящему времени почти все эти недочеты уже исправлены — кроме одного, а именно некоторой непоследовательности в обращении с логическими (Boolean) выражениями. Посмотрите: в окне **Параметры** (Options) в раскрывающихся списках доступны опции **True** и **False** (рис. П3.2), и при этом в том же окне, но в другой группе опций с ними соседствуют **Истина** и **Ложь** (рис. П3.3). И то, и другое нормально, но все же хотелось бы единообразия. Впрочем, это уже такой несущественный недочет, упоминание о котором выглядит как мелочная придирка.

Все остальные баги, описанные в упомянутой статье, к настоящему моменту уже исправлены.

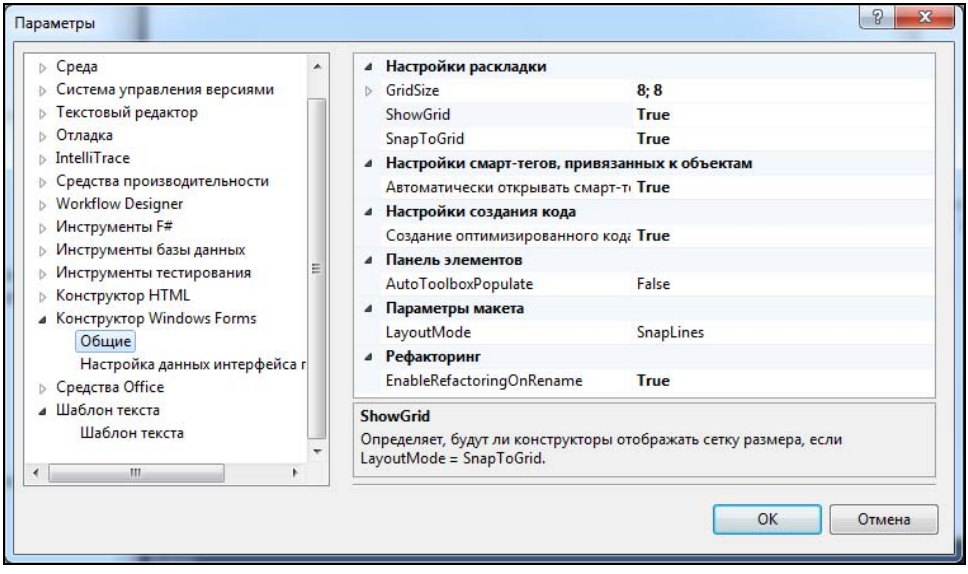


Рис. П3.2. Часть раскрывающихся списков предлагает для выбора традиционные **True** и **False**

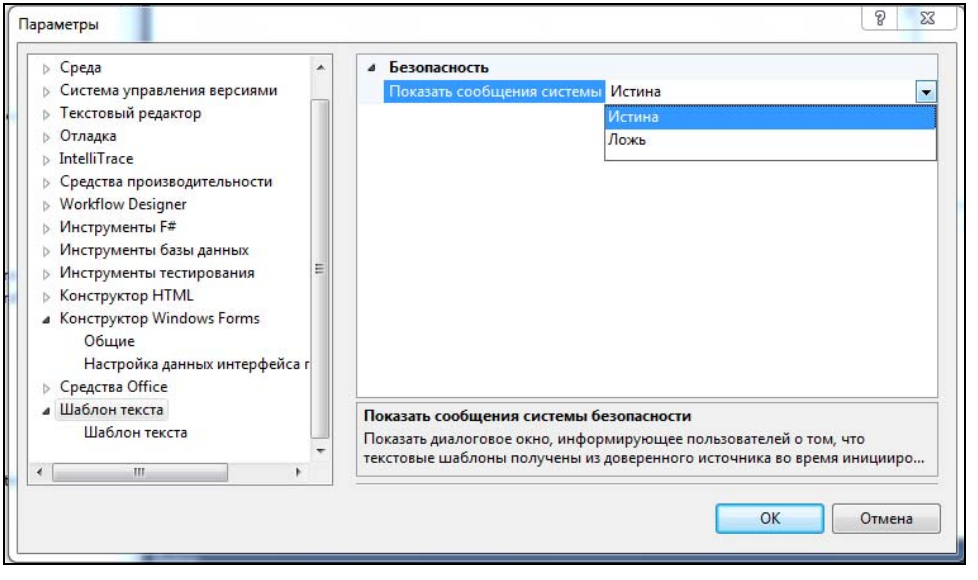


Рис. П3.3. При этом другая часть точно таких же раскрывающихся списков, но для другой группы опций, предлагает локализованные варианты

Наконец, нельзя не упомянуть, что возможно одновременное использование русской и английской версий Visual Studio, а языки можно переключать в настройках среды. Кроме того, можно использовать интерфейс на английском языке, а документацию на русском и наоборот. Иными словами, все сделано для вашего удобства — берите и пользуйтесь!

Предметный указатель