Алексей Голощапов

Microsoft® Visual Studio 2010

Санкт-Петербург
«БХВ-Петербург»
2011

УДК 681.3.068
ББК 32.973.26-018.1
Г60

Голощапов А. Л.

Г60 Microsoft® Visual Studio 2010. — СПб.: БХВ-Петербург, 2011. — 544 с.: ил. + CD-ROM — (В подлиннике)

ISBN 978-5-9775-0617-5

Рассмотрены приемы работы в интегрированной среде разработки Microsoft Visual Studio 2010, а также новые технологии и элементы среды, предназначенные для создания современных приложений. Описана работа с решениями, проектами, редакторами и визуальными конструкторами. Описывается создание различных типов приложений: с помощью технологий Windows Presentation Foundation и Windows Forms, создание веб-приложений с помощью технологий ASP.NET, MVC, AJAX, jQuery, Silverlight. Рассматривается проектирование и развертывание баз данных, а также создание приложений для работы с базами данных с использованием технологий LINQ, Entity Framework, ASP.NET Dynamic Data, технология создания служб Windows Communication Foundation, управление рабочими процессами с помощью Windows Workflow Foundation, локализация и развертывание приложений. Материал книги сопровождается практическими примерами в тексте и на компакт-диске.

Для программистов

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

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

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.01.11.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 43,86.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

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

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

ISBN 978-5-9775-0617-5

Оглавление

Введение

Новая версия интегрированной среды разработки Microsoft Visual Studio 2010 продолжает дальнейшее развитие семейства продуктов Visual Studio и содержит полный набор инструментов для управления всеми этапами жизненного цикла разработки программного обеспечения, начиная с проектирования архитектуры и заканчивая тестированием и развертыванием приложений. Visual Studio обеспечивает разработчикам широкие возможности для эффективного создания современного программного обеспечения.

На кого рассчитана эта книга

Эта книга предназначена для программистов-практиков. В ней не содержатся сведения об особенностях программирования на языке C#.NET. Эта книга о создании практических приложений с помощью инструментов, предоставляемых Visual Studio 2010. Книга подробно описывает практически все технологии, платформы и библиотеки, доступные в последней версии интегрированной среды разработки.

Материал в книге излагается достаточно детально, однако с учетом ограниченного объема книги невозможно полностью осветить все аспекты программирования в среде Visual Studio 2010. Основное назначение данной книги — предоставить читателю необходимую информацию для дальнейшей полноценной и эффективной работы в новой версии Visual Studio.

Я надеюсь, что эта книга будет полезной и ценной любому человеку, заинтересованному в разработке приложений в среде Visual Studio 2010. Начинающие программисты с базовыми знаниями C#.NET смогут начать свою профессиональную деятельность, освоив по книге новую версию Visual Studio 2010. Профессиональные программисты познакомятся с основными функциональными возможностями новой среды разработки, которые смогут использовать в своей деятельности.

В целом эта книга содержит много практической информации, которая пригодится вам независимо от вашего опыта и профессиональных интересов, и должна помочь вам в освоении и использовании новой версии Visual Studio.

Краткое описание глав

Книга состоит из 5 частей, которые содержат 23 главы, и одного приложения.

Первая часть книги — "Основы Visual Studio 2010". В ней приводятся общие сведения о среде разработки, ее настройке и отладке приложений.

- *Глава 1. Общие сведения о Visual Studio 2010.*

  Новая версия Visual Studio включает большое число превосходных новых возможностей и ключевых обновлений. В этой главе описываются усовершенствования и нововведения интегрированной среды разработки Visual Studio 2010.

- *Глава 2. Настройка рабочей среды Visual Studio 2010.*

  Глава посвящена настройкам интегрированной среды разработки, которые хранятся в группах, называемых параметрами. Можно настроить и хранить такие параметры, как отображение окна инструментов, структура окон, расположение команд меню, имена меню, отображение шаблонов в диалоговом окне **New Projects** и сочетания клавиш.

- *Глава 3. Интегрированная среда разработки.*

  Рассматривается ключевая идея VS.NET — это единая среда разработки, которая использует одинаковую для всех языков логику создания приложений, общий набор различных программных компонентов, в том числе библиотек классов.

- *Глава 4. Отладка приложений.*

  В этой главе будет представлен краткий обзор возможностей отладки приложений, предлагаемых средой Visual Studio 2010, с концентрацией внимания на тех аспектах, которые могут оказаться новыми для некоторых категорий разработчиков.

- *Глава 5. Создание приложений Windows Forms.*

  В этой главе рассматривается создание проектов на основе Windows Forms. Здесь будут описаны принципы создания пользовательского интерфейса под Windows, переключения и обмен данными между формами в приложениях. Хотя технология Windows Forms далеко не новая, на ее основе было создано огромное количество приложений, которые следует поддерживать, и любому программисту просто необходимо знать принципы работы этой технологии.

Вторая часть книги — "Технологии доступа к данным" — описывает инструменты для разработки баз данных и различные технологии доступа к данным, как новые, так и уже известные с прошлых версий Visual Studio.

- *Глава 6. Проектирование баз данных в Visual Studio 2010.*

  Глава знакомит читателя с тем, как при помощи инструментов Visual Database Tools создаются базы данных и их объекты, а также как созданные базы данных наполнить тестовой информацией с помощью инструментов, предоставляемых средой Visual Studio.

- *Глава 7. Технология доступа к данным ADO.NET.*

  Рассматриваются общие принципы работы с технологией ADO.NET. Технология ADO.NET предоставляет доступ к таким источникам данных, как SQL Server и XML. Пользовательские приложения, создаваемые для работы с данными, могут использовать ADO.NET для соединения с этими источниками данных и для получения, обработки и обновления имеющихся в них данных.

- *Глава 8. Работа с автономными данными в ADO.NET.*

  Основная задача использования ADO.NET — это работа с данными в автономном режиме. В этой главе изучаются объекты `DataSet` и `DataAdapter` работы с отсоединенными источниками данных.

- *Глава 9. LINQ.*

  В этой главе рассматривается LINQ (Language Integrated Query), технология, которая соединяет мир объектов с миром данных. LINQ позволяет разработчикам формировать прямо в программном коде запросы, основанные на наборах, без использования дополнительного языка запросов.

- *Глава 10. Entity Framework.*

  Здесь описывается Entity Data Model — спецификация для определения данных, используемых приложениями, построенными на основе платформы Entity Framework. Приложения используют определенные моделью EDM сущности и связи для работы с данными.

Третья часть книги — "Создание уровня презентаций" — это самая большая часть, здесь рассматриваются различные веб-технологии и библиотеки, доступные для разработки в среде Visual Studio 2010: веб-формы ASP.NET, AJAX, библиотека jQuery, ASP.NET Dynamic Data, ASP.NET MVC 2, Windows Presentation Foundation и технология Silverlight.

- *Глава 11. Веб-формы ASP.NET.*

  В этой главе рассматривается технология для создания веб-приложений ASP.NET. Платформа ASP.NET обеспечивает полноценную объектную модель. Элементы управления, размещаемые на веб-страницах, предлагают богатый набор функциональности.

- *Глава 12. Стили и темы.*

  В этой главе изучается создание каскадных таблиц стилей и тем, управление ими, а также применение их к вашим страницам. Стили и темы определяют общий дизайн веб-сайта, для изменения стиля вы должны будете сделать это в одном месте — и изменения произойдут везде, где используется данный стиль.

- *Глава 13. Мастер-страницы и управление навигацией.*

  Эта глава научит созданию полноценных веб-сайтов. Чтобы построить профессиональный веб-сайт, используют мастер-страницы и элементы, управляющие навигацией по веб-сайту.

- *Глава 14. ASP.NET AJAX.*

  В этой главе рассматривается технология AJAX. Эта технология позволяет усовершенствовать пользовательский интерфейс веб-приложений за счет асинхрон-

ного обмена и динамических манипуляций веб-страницей на стороне клиента. Пользователь может взаимодействовать с серверными функциями и данными без необходимости полного обновления страниц.

- *Глава 15. Библиотека jQuery.*

  Библиотека jQuery обеспечивает функциональность для выбора, обхода и управления элементами веб-страниц. Она предоставляет всевозможные специальные эффекты. Библиотека jQuery позволяет легко отделять код от дизайна. В итоге написанный код проще читается и более надежен.

- *Глава 16. ASP.NET Dynamic Data.*

  В этой главе изучается платформа ASP.NET Dynamic Data, позволяющая легко создавать приложения ASP.NET на основе данных. Это достигается путем автоматического определения метаданных модели данных во время выполнения и вывода из них функциональности пользовательского интерфейса. Механизм формирования шаблонов позволяет получить функциональный веб-узел для просмотра и изменения данных.

- *Глава 17. ASP.NET MVC.*

  В этой главе рассматривается платформа Model-View-Controller (MVC) — архитектурный принцип, согласно которому веб-приложение делится на компоненты. Такое разделение облегчает управление отдельными частями приложения, что упрощает их разработку, изменение и тестирование. ASP.NET MVC — это альтернатива разработке веб-страниц ASP.NET.

- *Глава 18. Windows Presentation Foundation.*

  В этой главе изучается технология Windows Presentation Foundation (WPF) — система нового поколения для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем. С помощью WPF можно создавать широкий спектр как автономных, так и размещенных в браузере приложений.

- *Глава 19. Silverlight.*

  В этой главе рассматривается технология Silverlight, которая является подмножеством Windows Presentation Foundation (WPF), она значительно расширяет возможности элементов браузера для создания пользовательского интерфейса. Silverlight позволяет создавать впечатляющие графику, анимацию и другие клиентские мультимедийные функции.

Четвертая часть книги — "Сервисы и коммуникации" — посвящена разработке уровня коммуникаций между приложениями.

- *Глава 20. Windows Communication Foundation.*

  В этой главе рассматривается технология Windows Communication Foundation (WCF), которая является единой моделью программирования Microsoft для построения служб. Она позволяет разработчикам строить безопасные надежные решения с поддержкой транзакций и возможностью межплатформенной интеграции.

- *Глава 21. Windows Workflow Foundation.*

  В этой главе рассматривается новая версия Windows Workflow Foundation. Технология Windows Workflow Foundation является моделью, в которой вы можете определять и выполнять процессы с использованием набора строительных блоков — действий (activities).

Пятая часть книги — "Локализация и развертывание приложений" — посвящена созданию интернациональных приложений и развертыванию приложений на клиентских компьютерах.

- *Глава 22. Локализация приложений.*

  Для успешной работы приложений на международном рынке необходима поддержка различных региональных стандартов. Платформа .NET Framework обладает интегрированной инфраструктурой, которая позволяет создавать интернациональные приложения.

- *Глава 23. Развертывание приложений.*

  Для разработчиков .NET в среде Visual Studio доступны обширные возможности для создания проектов развертывания и способы, которыми могут быть конфигурированы проекты развертывания, в первую очередь, для больших и сложных приложений с использованием технологии Windows Installer.

Исходные коды примеров

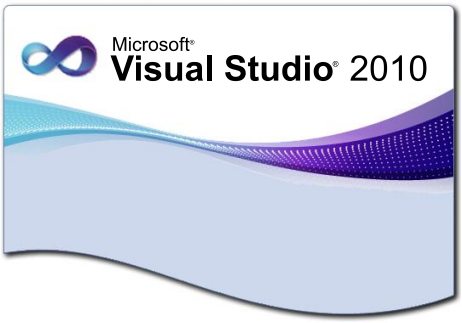
На диске, прилагаемом к книге, находятся все исходные коды примеров. Установка примеров описана в *приложении*.

Книга содержит полные исходные коды всех программ, однако некоторые листинги, для экономии места и во избежание ненужного дублирования информации, содержат только изменения программного кода относительно предыдущих листингов. Такое сокращение позволяет не только экономить место, но и улучшить понимание программного кода, делая акцент только на новой функциональности.

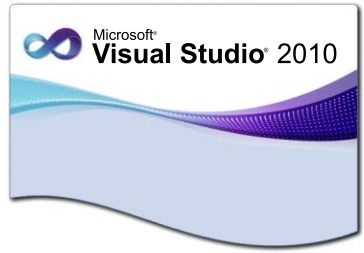
Благодарности

В первую очередь хочу поблагодарить своих родных и близких за оказанную моральную поддержку в процессе написания этой книги. Отдельная благодарность издательству "БХВ-Петербург", заместителю главного редактора Игорю Владимировичу Шишигину и всем сотрудникам издательства, которые помогали мне в создании этой книги.

ЧАСТЬ I



Основы Visual Studio 2010



ГЛАВА 1

Общие сведения о Visual Studio 2010

Visual Studio 2010 была спроектирована с целью максимально упростить процесс написания кода, его отладки и компиляции в сборку, предназначенную для поставки конечным потребителям. С помощью Visual Studio 2010 можно выполнять почти любые виды работ, связанных с разработкой, отладкой и тестированием программного обеспечения.

Версии Visual Studio 2010

Visual Studio 2010 распространяется в нескольких изданиях:

- Professional;
- Premium;
- Ultimate;
- Test Professional.

Visual Studio 2010 Professional предназначена в основном для отдельных разработчиков или небольших групп разработки программного обеспечения.

Visual Studio 2010 Premium, в отличие от Professional, содержит дополнительные инструменты для отладки кода и диагностики кода, например, инструменты для статистического анализа кода, создания метрик кода и профилирования. Она также позволяет осуществлять автоматизированное тестирование пользовательского интерфейса. Для работы с базами данных Visual Studio 2010 Premium содержит инструменты для управления изменениями баз данных, модульного тестирования и генерации тестовых данных для наполнения БД. Visual Studio 2010 Premium позволяет создавать сложные масштабируемые и высококачественные приложения.

Visual Studio 2010 Ultimate является самым полным изданием Visual Studio и дополнительно предоставляет большой набор средств для управления циклом жизни приложения.

Visual Studio 2010 Test Professional — это специализированный набор средств для тестирования приложений и, в отличие от Visual Studio 2010 Professional,

Premium и Ultimate, не предназначена для непосредственной разработки программного обеспечения. Visual Studio 2010 Test Professional используется вместе с основными редакциями Visual Studio для совместной работы разработчиков программного обеспечения и тестеров в течение всего жизненного цикла создания приложения.

Усовершенствования интегрированной среды разработки Visual Studio

Новая версия Visual Studio включает большое число новых возможностей и ключевых обновлений.

Новая стартовая страница

Стартовая страница Visual Studio 2010 приобрела новый вид и дополнительные функциональные возможности (рис. 1.1). Область содержимого с вкладками **Get Started**, **Guidance and Resources**, **Latest News** предоставляет ссылки к разнообраз-

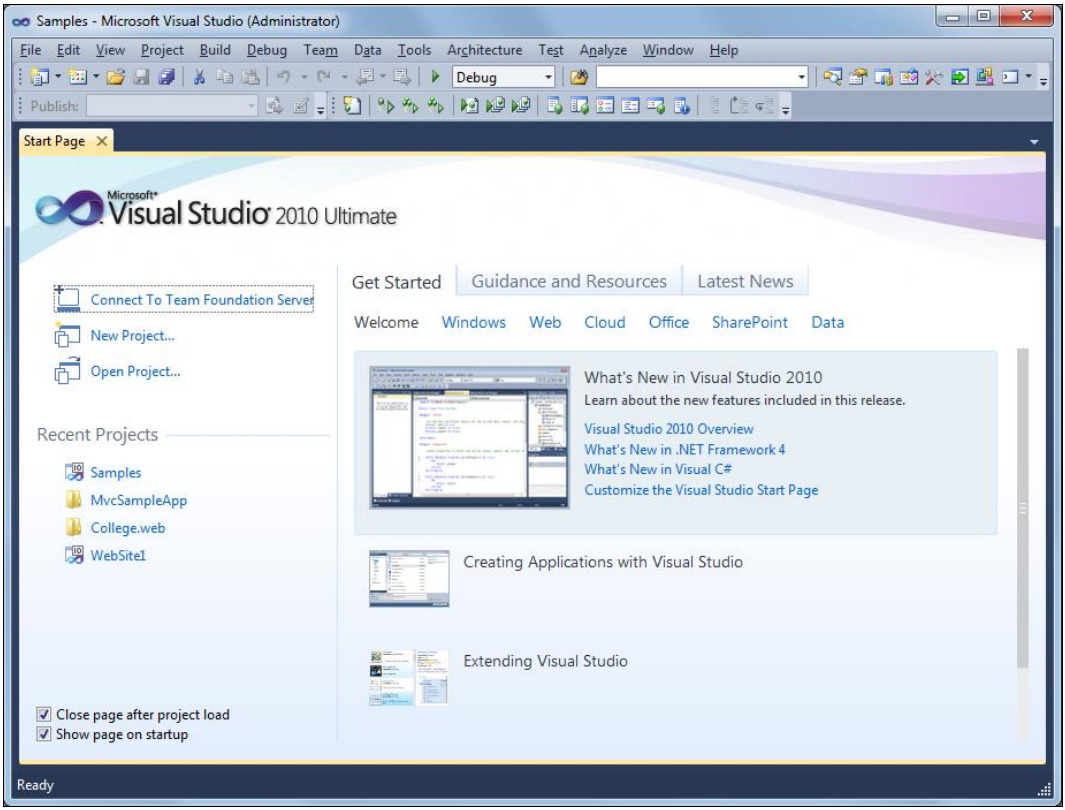


Рис. 1.1. Стартовая страница Visual Studio 2010

ным сгруппированным по категориям справочным материалам. В эту область загружаются ресурсы MSDN, ресурсы сообщества и настраиваемый веб-канал новостей.

На стартовой странице также изменился список **Recent Projects** (Последние проекты), который теперь позволяет добавлять или удалять проекты в этом списке одним щелчком мыши.

Поддержка нескольких мониторов

В Visual Studio 2010 окна документов, например редакторы кода и визуальные конструкторы, теперь являются самостоятельными окнами и, при необходимости, могут размещаться вне главного окна среды разработки. Например, можно перетащить редактор кода или страницу справки за пределы главного окна среды разработки, чтобы можно было видеть редактор кода рядом с визуальным конструктором форм и разместить их на разных мониторах (рис. 1.2).

Выделение ссылок в коде

В Visual Studio 2010 редакторы кода получили дополнительную функциональность, облегчающую работу с кодом. В новой версии Visual Studio появилась удобная функция выделения ссылок в программном коде. При выделении в редакторе кода какой-либо переменной, объекта, метода, свойства, типа или другого символа в коде автоматически выделяются все вхождения этого объекта, чтобы разработчик мог увидеть, где в программном коде используется выделенный элемент (рис. 1.3).

Масштабирование

В любом окне редактирования кода или документа среды разработки Visual Studio 2010 теперь возможно быстро увеличить или уменьшить масштаб, нажав и удерживая клавишу <Ctrl>, и одновременно вращая колесо прокрутки мыши. Можно также масштабировать некоторые текстовые окна инструментов, например окно **Output** (Вывод).

Однако функция масштабирования не работает в рабочих областях конструирования или в окнах инструментов, содержащих значки, например **Toolbox** (Панель инструментов) или **Solution Explorer** (Обозреватель решений).

Выделение области

В Visual Studio 2010 появилась новая удобная функция редактора кода — возможность прямоугольного выделения и редактирования выбранного вертикального блока текста (рис. 1.4).

В предыдущих версиях Visual Studio можно было выбрать прямоугольный фрагмент текста, удерживая нажатой клавишу <Alt> и выделяя фрагмент с помощью мыши. Затем выделенный текст можно было копировать или удалить. В Visual Studio 2010 в функцию выделения области добавлены возможности вставки текста.

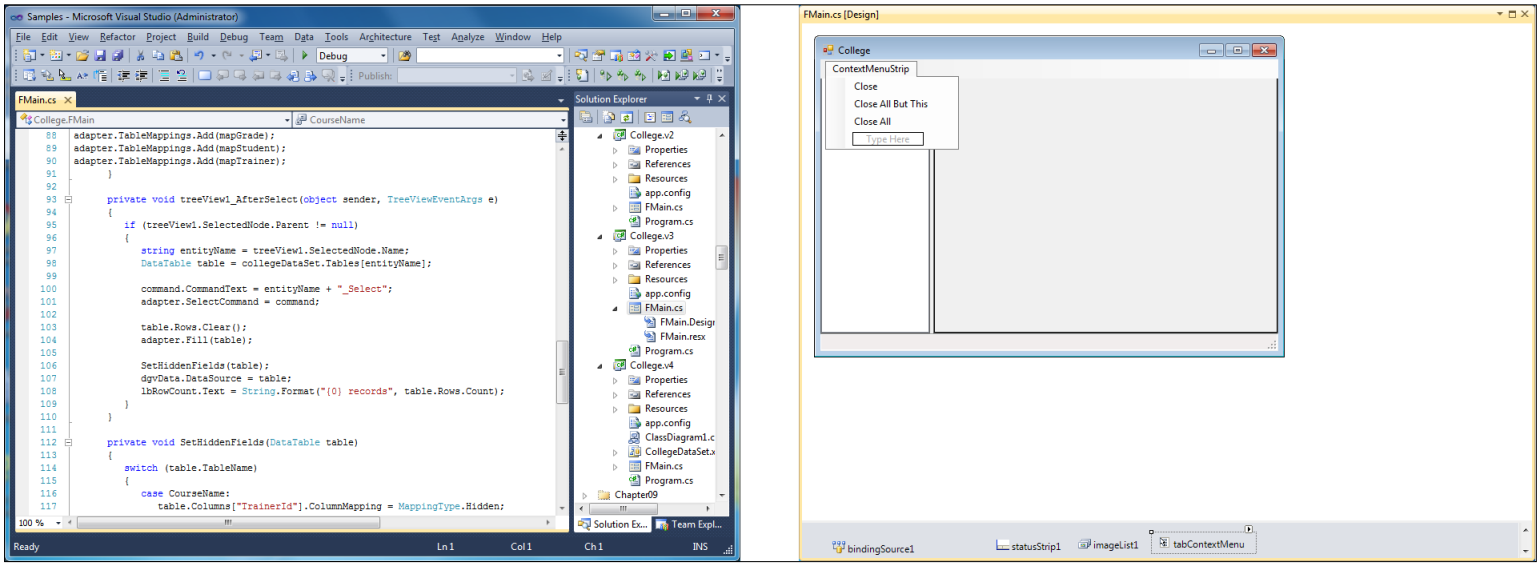


Рис. 1.2. Размещение окна конструктора вне главного окна IDE

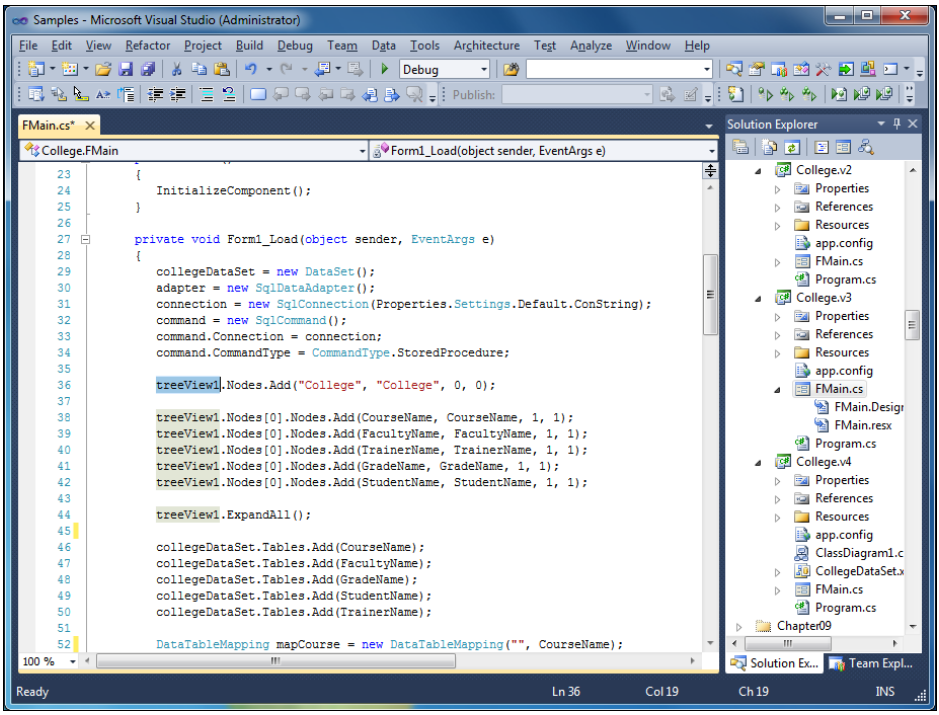


Рис. 1.3. Выделение вхождений объекта `treeView1` в коде программы

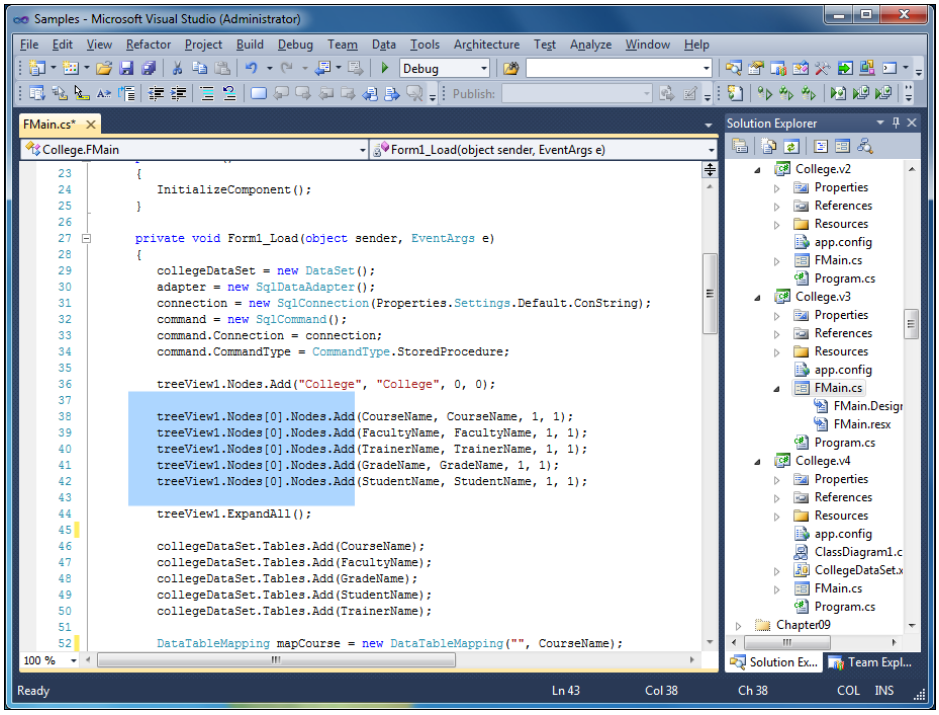


Рис. 1.4. Выделение области кода с помощью мыши

Выделение и редактирование вертикального блока текста можно использовать для работы в коде программы с группой операторов, например, изменения модификаторов доступа, установки новых полей или добавления комментариев.

Тестирование и отладка приложений

Среда разработки Visual Studio 2010 теперь может создавать заглушки кода для новых типов и членов, до их определения. В результате можно сначала писать тесты, а затем создавать код, необходимый для компиляции этих тестов. Кроме того, теперь технология IntelliSense предоставляет режим предложений, который предотвращает автоматическое заполнение типа или элемента, который еще не был определен в программном коде.

Для упрощения отладки приложений в Visual Studio 2010 была добавлена новая функциональность. Например, в окне **Threads** предоставляются возможности фильтрации, поиска и расширения стека вызовов, а также группирования. Кроме того, теперь можно упорядочивать точки прерывания, выполнять по ним поиск и совместно использовать их с другими разработчиками.

Для разработки приложений с применением технологий параллельнлго программирования были добавлены новые окна **Parallel Stacks** и **Parallel Tasks**, помогающие в отладке параллельного кода.

Усовершенствования разработки приложений ASP.NET

В Visual Studio 2010 усовершенствована разработка приложений на ASP.NET. В ASP.NET 4 были расширены возможности кэширования вывода, появился диспетчер предварительной загрузки приложений.

В Visual Studio 2010 включена новая версия архитектурного шаблона MVC. В новой версии появились строго типизированные вспомогательные функции, а также функции на основе шаблонов, которые позволяют автоматически генерировать UI для сущностей.

В конструкторе веб-страниц в Visual Studio 2010 поддерживается смешанное представление (представление с разделением). Такая функциональность позволяет просматривать файл одновременно и в графическом представлении, и в представлении исходного кода (рис. 1.5).

Усовершенствования конструкторов для Windows Presentation Foundation и Silverlight

В Visual Studio 2010 внесены значительные усовершенствования конструктора, помогающие создавать приложения Windows Presentation Foundation (WPF). Например, после добавления источника данных в проект можно создавать элементы

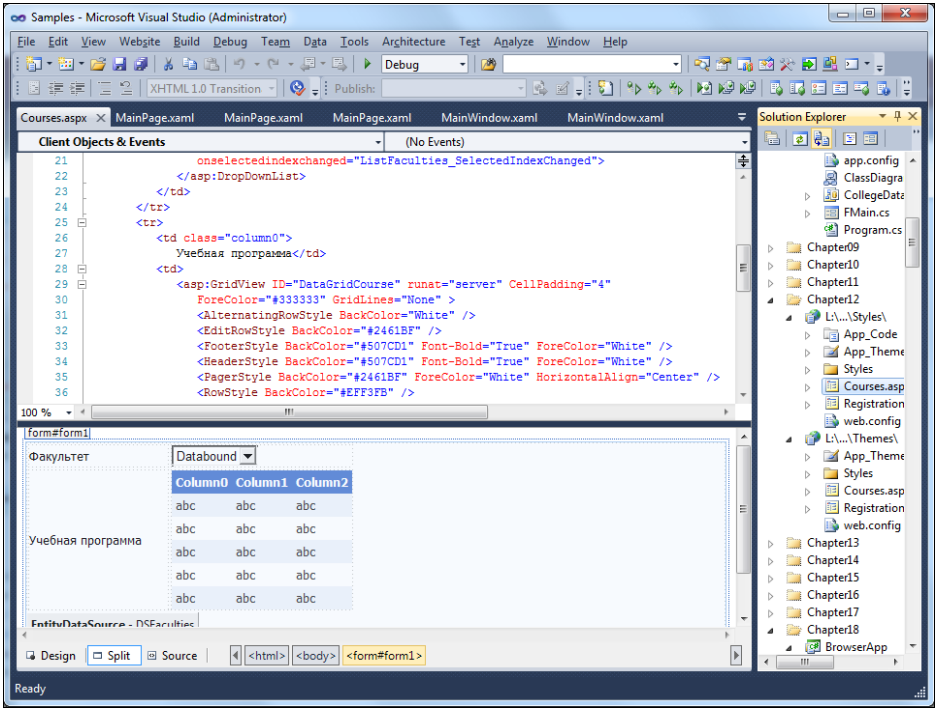


Рис. 1.5. Конструктор веб-страниц с разделением представлений

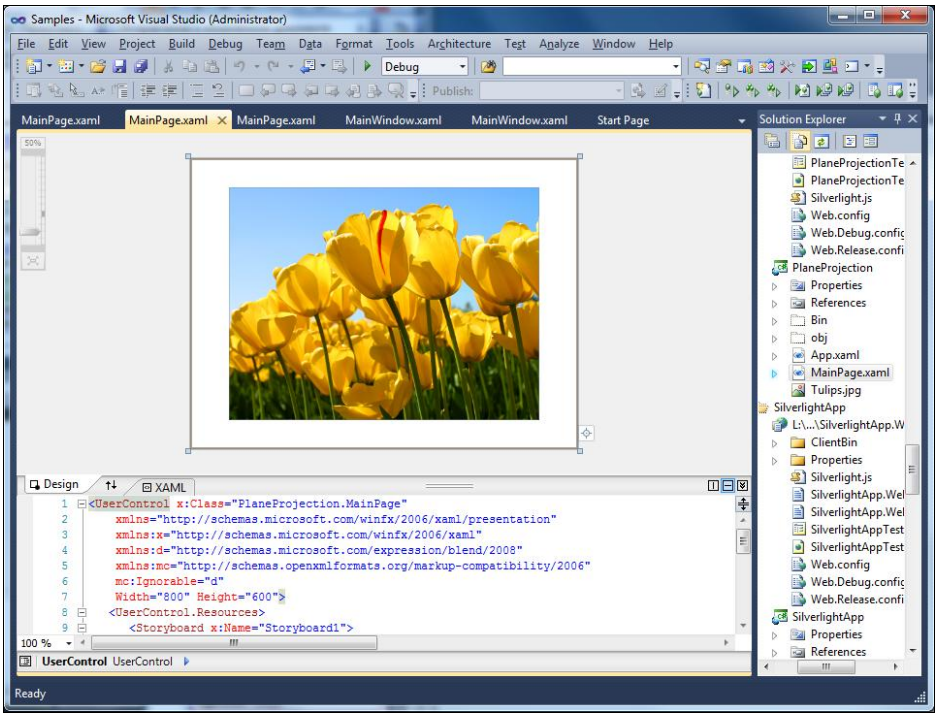


Рис. 1.6. Конструктор Silverlight

управления Windows Presentation Foundation с привязкой к данным путем перетаскивания элементов из окна источников данных в визуальном конструкторе WPF.

В предыдущей версии Visual Studio поддержка визуального конструктора для проектов Silverlight была ограничена окном предварительного просмотра, доступным только для чтения. В Visual Studio 2010 функциональность визуального конструктора для Silverlight не отличается от визуального конструктора Windows Presentation Foundation. Теперь при разработке проектов Silverlight (рис. 1.6) можно выбирать и располагать элементы в рабочей области визуального конструктора с помощью мыши так же, как и в проектах Windows Presentation Foundation.

Новая версия Windows Workflow Foundation

Технология Windows Workflow Foundation в .NET Framework 4.0 претерпела значительные изменения, была улучшена ее производительность. Теперь рабочие процессы еще проще создавать, выполнять и поддерживать, а также реализовывать узел с новыми функциями. Кроме того, полностью поменялся конструктор Windows Workflow Foundation (рис. 1.7).

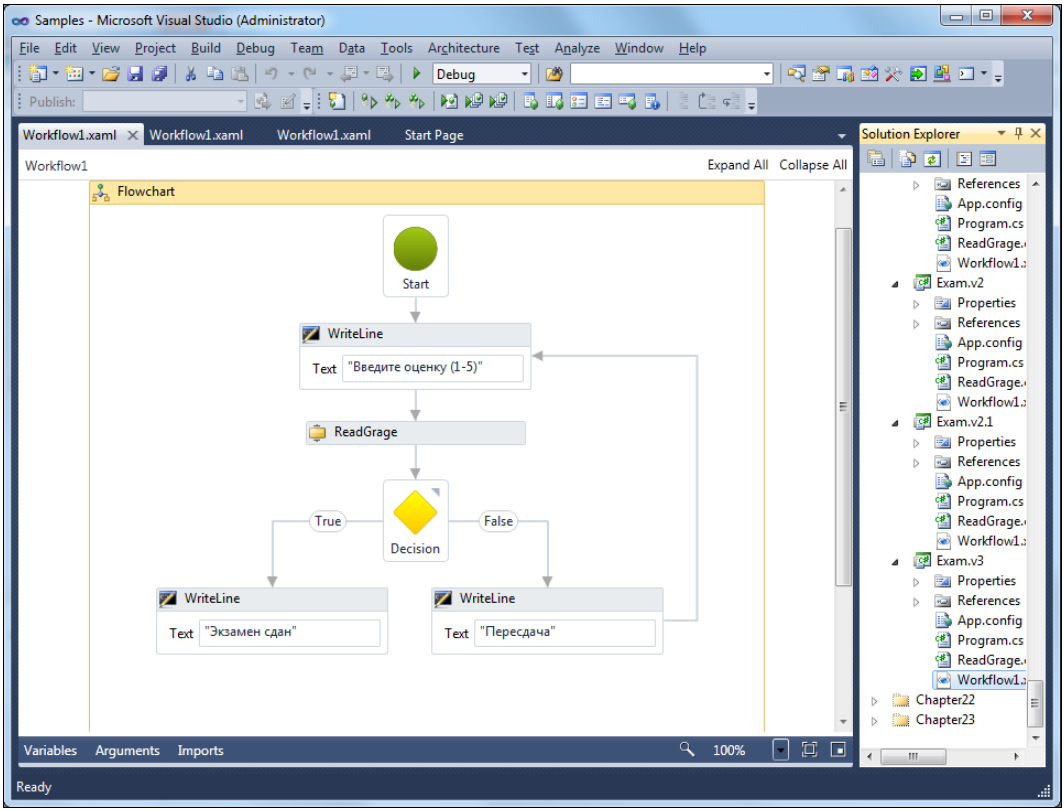


Рис. 1.7. Новый конструктор Windows Workflow Foundation

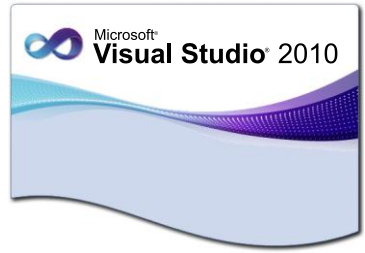
Однако в новой версии Windows Workflow Foundation теперь отсутствует поддержка рабочих потоков конечных автоматов (State Machine Workflow), хотя, возможно, что State Machine Workflow снова появится в следующих версиях Windows Workflow Foundation.

Резюме

Visual Studio 2010, безусловно, является большим и сложным продуктом. Основная идея Visual Studio — это реализация единой среды разработки, которая использует одинаковую для всех языков логику создания приложений, общий набор различных программных компонентов, в том числе библиотек классов.

Дополнительная фунциональность, появившаяся в новой версии, еще больше укрепляет позиции Visual Studio 2010 как универсального инструмента разработки, способного обеспечить разработчикам быстрое и качественное создание современного программного обеспечения.

ГЛАВА 2



Настройка рабочей среды Visual Studio 2010

Прежде чем приступить к написанию приложений в новой среде разработки, для более эффективной и удобной работы надо настроить параметры среды. Регулировка параметров позволяет настраивать среду разработки в соответствии с вашим стилем разработки.

Первая загрузка интегрированной среды разработки приведет к появлению диалогового окна, которое попросит вас выбрать настройки среды разработки. Для того чтобы помочь в этом вопросе, Microsoft создала целый набор настроек среды, которые сконфигурированы под обычные типы разработчиков. Например, если вы настроите свою среду разработки на работу с языком C#, то в диалоговом окне создания нового проекта **New Project** будут первыми располагаться шаблоны проектов на языке C#.

Параметры настройки среды

В Visual Studio настройки интегрированной среды разработки хранятся в группах, называемых *параметры настройки среды*. Значения параметров могут быть основаны на различных действиях по разработке, а также на предпочтениях пользователя. Можно настроить и хранить такие параметры, как отображение окна инструментов, структура окон, расположение команд меню, имена меню, отображение шаблонов в диалоговом окне **New Project** и сочетания клавиш.

Параметры — это возможности настройки частей интегрированной среды разработки, например, структуры окон, поведения редактора, фрагментов кода IntelliSense и параметров диалогового окна. Параметры можно сохранить, экспортировать, импортировать или сбросить для упрощения использования интегрированной среды разработки. По умолчанию действующие параметры сохраняются в файле Currentsettings.vssettings.

При первом запуске Visual Studio необходимо выбрать одну из предварительно определенных коллекций параметров. На рис. 2.1 показано окно выбора параметров. Диалоговое окно выбора настроек среды по умолчанию **Choose Default Environment Settings** появляется только при первом запуске Visual Studio. При

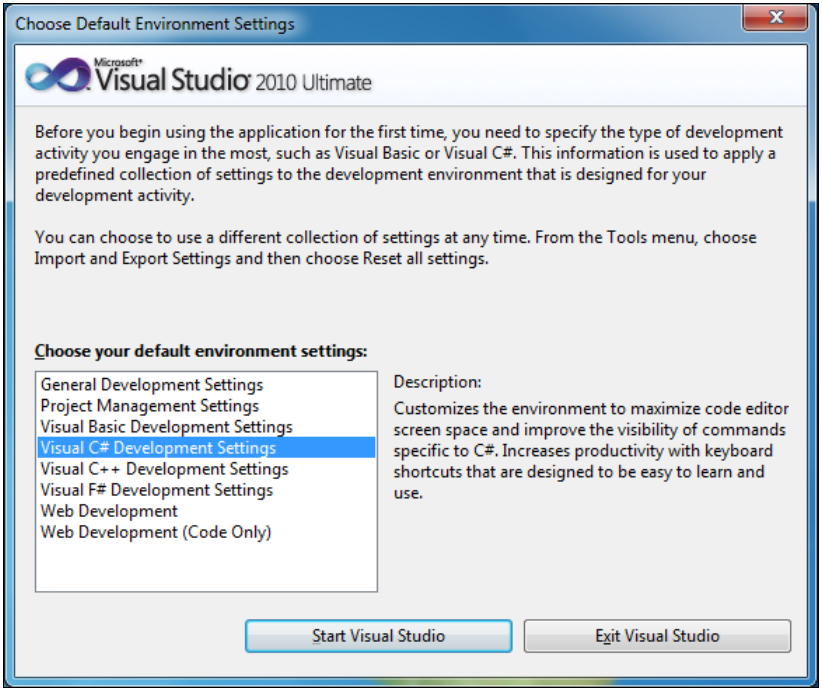


Рис. 2.1. Окно выбора настроек среды Visual Studio

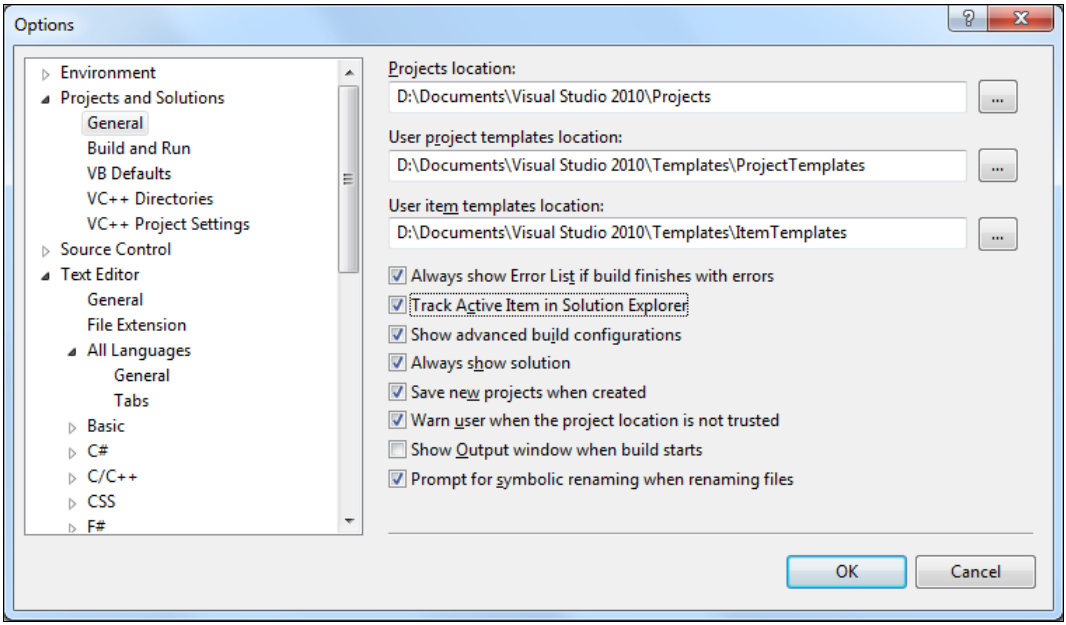


Рис. 2.2. Диалоговое окно выбора вариантов настроек

последующих запусках вы напрямую попадете в интегрированную среду разработки Visual Studio 2010.

Каждая коллекция параметров разработана таким образом, чтобы соответствовать конкретному стилю разработки. Ее можно изменять в процессе работы. При каждом изменении параметра, отслеживаемого Visual Studio, например, изменении цвета закомментированного кода в редакторе, изменение автоматически сохраняется. Visual Studio применяет активные параметры автоматически при каждом запуске Visual Studio.

Параметры среды разработки можно менять, выбрав в главном меню пункт **Tools | Options**. При этом откроется диалоговое окно **Options**, в котором содержится множество параметров, которые можно менять в зависимости от предпочтений пользователя (рис. 2.2).

Миграция параметров настроек

Visual Studio 2010 поддерживает миграцию настроек. Если, например, у вас имеется инсталлированная система Visual Studio 2008 (или более ранняя версия), то Visual Studio 2010 позволит вам при первом запуске выполнить миграцию настроек. Можно экспортировать параметры редактора, проектов, отладки, шрифтов, цветов и других функций среды разработки в файл, чтобы их можно было импортировать на другой компьютер и использовать на нем. Также можно перенести параметры с предыдущей версии Visual Studio для использования в другой версии Visual Studio.

Можно заменить как все активные параметры, так и некоторые из них. Если импортируемый vssettings-файл параметров содержит параметры для всех категорий, при импорте этого файла будут заменены все существующие параметры. При импорте файла с расширением vssettings, содержащего подмножество категорий параметров, или при удалении категорий параметров в импортируемом файле с расширением vssettings можно заменить только выбранные параметры, а все остальные активные параметры не будут изменены.

При наличии нескольких версий Visual Studio, установленных на одном компьютере, параметры можно переносить из одной версии в другую. Обычно активный файл параметров содержит два типа параметров:

- предварительно определенные параметры, которые устанавливаются при первом запуске Visual Studio;
- пользовательские параметры, которые создаются при изменении любых предварительно определенных параметров.

По умолчанию все действующие параметры настроек среды сохраняются в файле Currentsettings.vssettings.

Для переноса настроек на новый компьютер используется мастер импорта и экспорта настроек **Import and Export Settings Wizard**. Этот мастер позволяет импортировать настройки, экспортировать их в файл или восстанавливать настройки в один из вариантов по умолчанию, предлагаемых интегрированной средой.

Для управления настройками среды используется команда меню **Tools | Import and Export Settings**. На рис. 2.3 показана первая страница **Import and Export Settings Wizard**. Эта страница позволяет выбирать экспорт или импорт настроек.

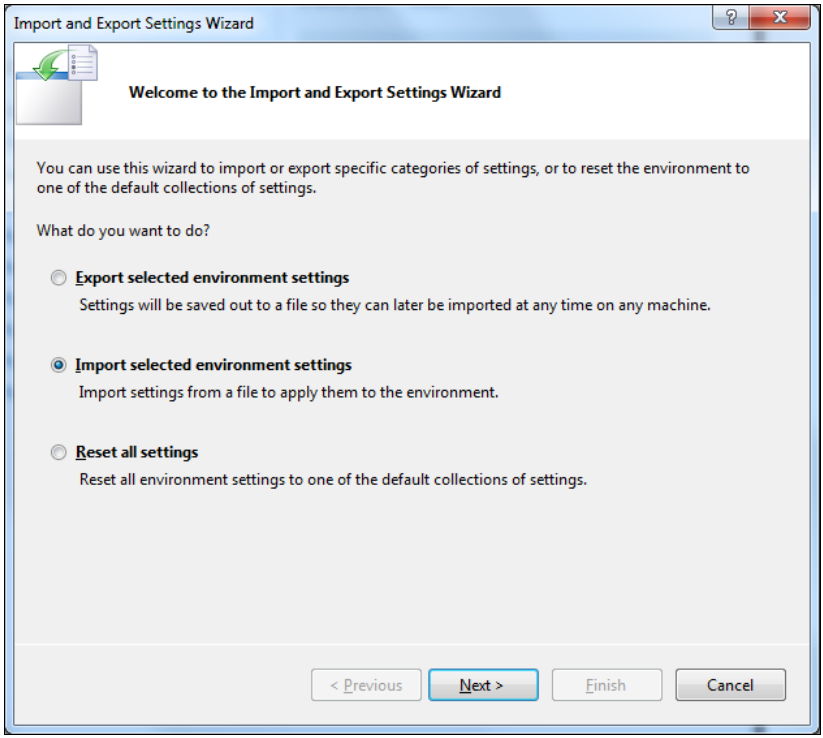


Рис. 2.3. Мастер импорта и экспорта настроек

Вы можете запустить мастер **Import and Export Settings Wizard** и выбрать импорт настроек из этого файла. На странице мастера **Save Current Settings** (Сохранить текущие параметры), представленной на рис. 2.4, можно сохранить текущие параметры для создания резервной копии или импортировать новые параметры, заменив текущие, чтобы продолжить работу без сохранения текущих параметров.

При импорте вы можете выбирать вариант из коллекций настроек. По умолчанию имеется несколько коллекций настроек (рис. 2.5). Кроме того, вы можете указать файл со своими настройками.

После создания файла с расширением vssettings его можно использовать на других компьютерах для импорта параметров в Visual Studio. Можно также загрузить файлы параметров, созданные другими программистами, и применить их параметры в вашем экземпляре Visual Studio.

Если импортируемый файл параметров содержит параметры для всех категорий, при импорте этого файла будут заменены все существующие параметры. Если файл содержит лишь некоторые категории параметров, то при импорте файла будут заменены только эти категории, а все прочие параметры останутся без изменений.

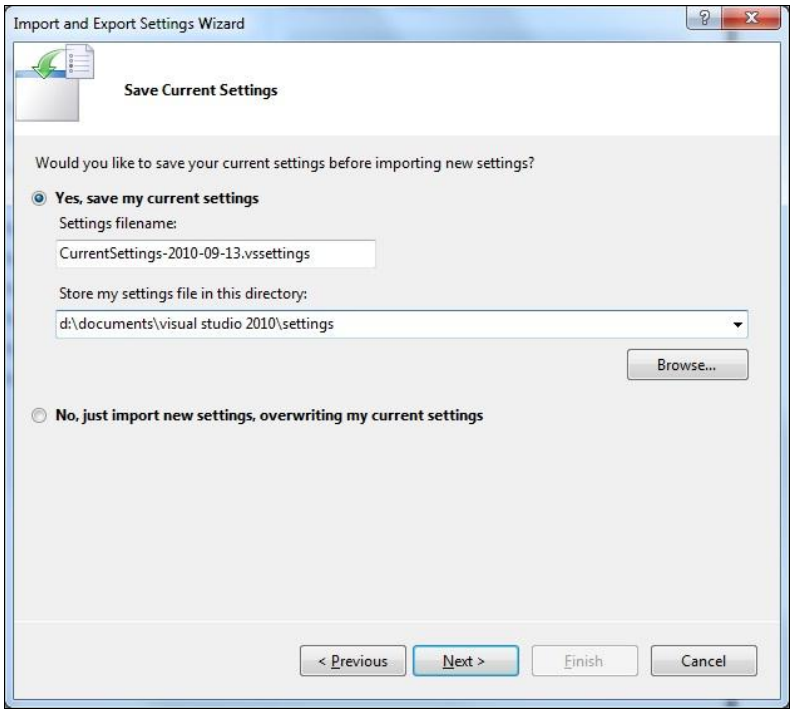


Рис. 2.4. Страница сохранения текущих настроек

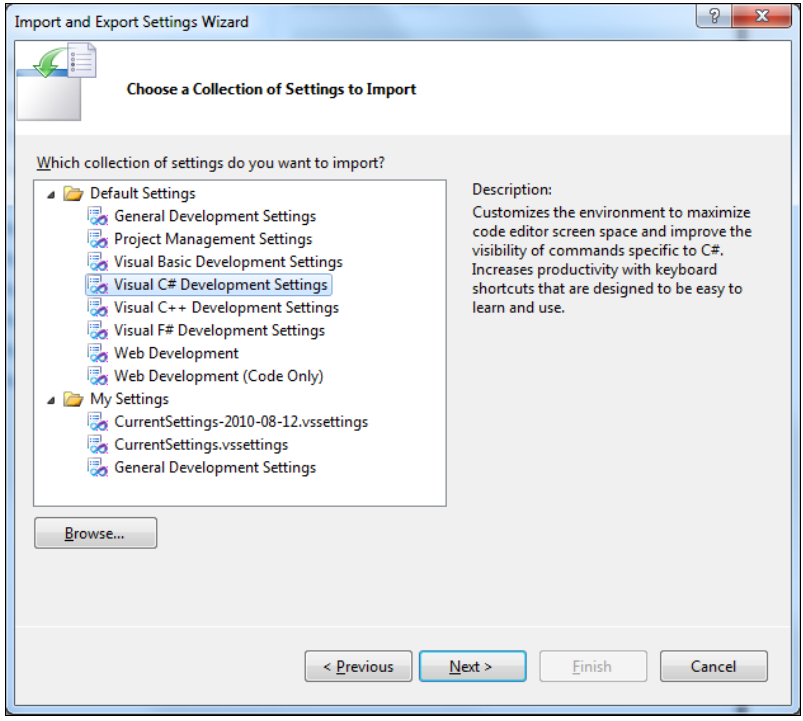


Рис. 2.5. Выбор коллекции настроек для импорта

Это позволяет вам выбирать настройки, которые вы планируете импортировать. Например, вы можете импортировать шрифты и виды форматирования кода, но вам не нужны все остальные его настройки. На рис. 2.6 показана страница мастера с набором доступных параметров настройки среды разработки. Параметры, помеченные черно-желтым треугольником с восклицательным знаком, содержат данные, связанные с параметрами безопасности компьютера.

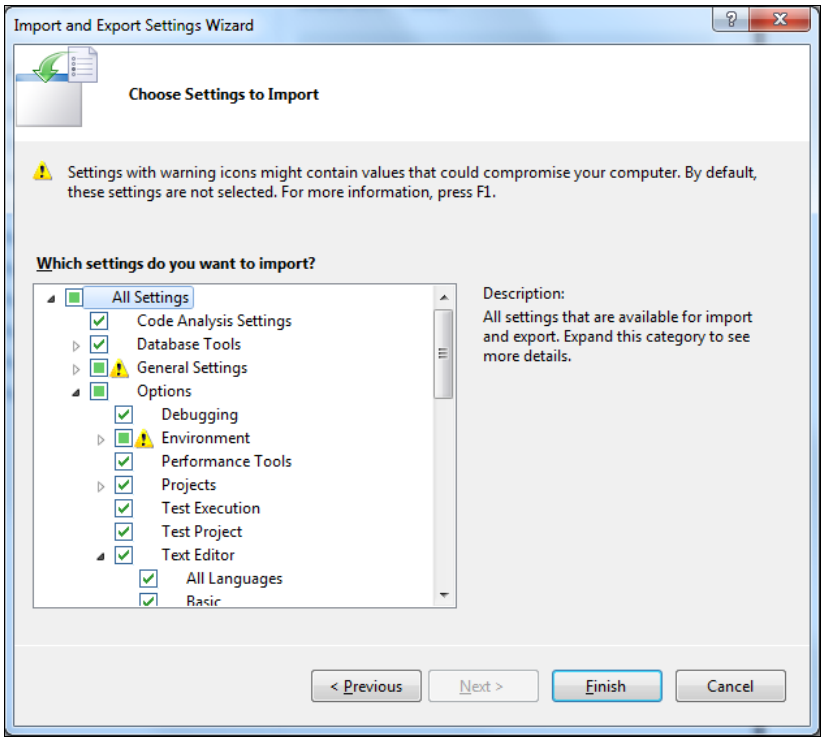


Рис. 2.6. Выбор настроек для импорта

При экспорте настроек можно указать, какие из них следует применить. Страница мастера с выбором настроек для экспорта аналогична странице импорта настроек. Обратите внимание, что можно выбрать только те настройки, которые хотите экспортировать. Например, можно не экспортировать все категории параметров и пометить только нужные категории, не изменяя остальные параметры.

При экспорте настроек также создается файл с расширением vssettings. Этот файл можно затем передать другим пользователям. Его можно использовать также для миграции настроек с одного компьютера на другой и с одной версии интегрированной среды на другую.

Start Page

При первом запуске Visual Studio отображается пустая интегрированная среда разработки, показанная на рис. 2.7, и стартовая страница **Start Page**. Стартовая

страница содержит различные ссылки на веб-сайты для разработчиков и дает возможность запускать новые проекты или открывать уже существующие.

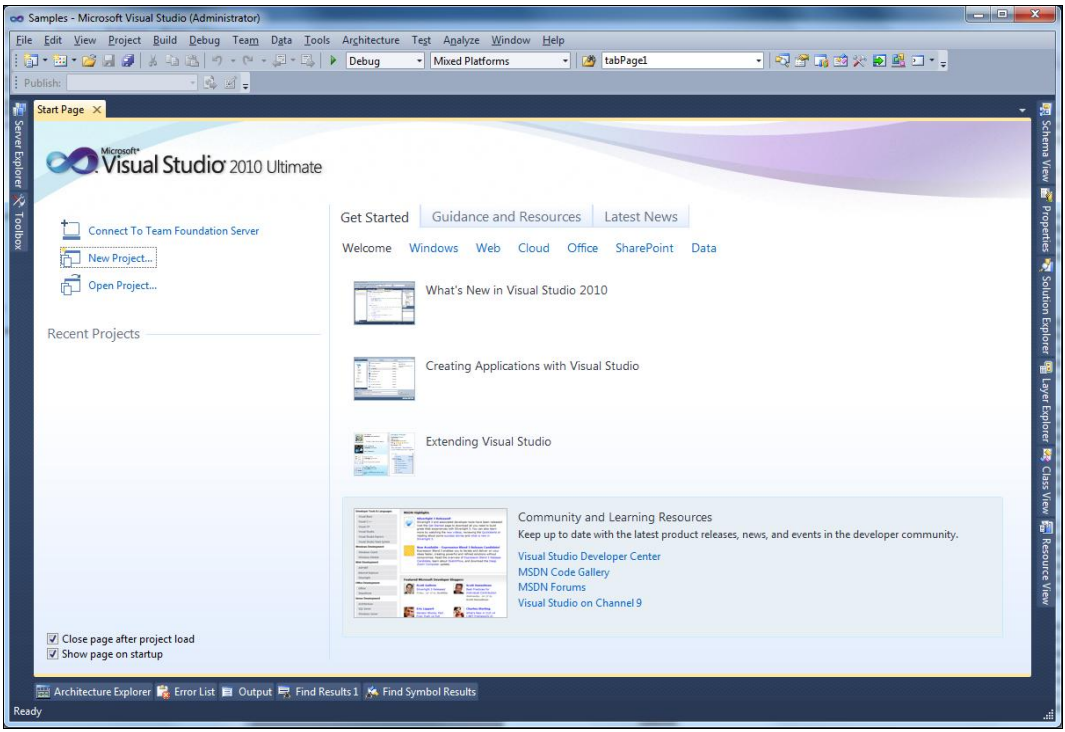


Рис. 2.7. Стартовая страница Visual Studio

Start Page также обеспечивает быстрый доступ к онлайн-ресурсам: сведениям о предстоящих выпусках продуктов и конференциях, а также о последних статьях по разработке приложений.

> ***Примечание***
>
> Стартовая страница всегда открыта по умолчанию. Если эта страница закрыта, для ее открытия в меню **View** (Вид) выберите **Start Page**.

Start Page разделена на три основных раздела:

- раздел команд, в котором отображаются команды **Connect To Team Foundation Server** (Соединение с сервером командной разработки), **New Project** и **Open Project**;
- список **Recent Projects** (Последние проекты**)**;
- область содержимого с вкладками **Get Started** (Начало работы), **Guidance and Resources** (Руководства и ресурсы) и **Latest News** (RSS-канал новостей).

В нижней части стартовой страницы находятся параметры **Close page after project load** (Закрыть страницу после загрузки проекта) и **Show page on startup**

(Показывать страницу при запуске), которые задают параметры отображения стартовой страницы Visual Studio.

Документация по продукту

Visual Studio позволяет получать доступ к справочной документации MSDN напрямую из среды разработки. Справочная система Microsoft представляет собой единую систему для работы с содержимым, получаемым из библиотеки MSDN Online, и с установленным локально содержимым. Клиентская часть справочной системы, называемая *средством просмотра справки (Microsoft)*, состоит из двух компонентов, таких как:

- агент библиотеки справки;
- диспетчер библиотеки справки.

Агент библиотеки справки представляет собой приложение области уведомлений, которое служит для отображения локального содержимого в браузере по умолчанию. Диспетчер библиотек справки служит для добавления, обновления и удаления автономного содержимого, а также для управления параметрами справки.

Панель навигации при работе с автономным вариантом справки была упрощена и стала похожа на работу с библиотекой MSDN Online. Содержание позволяет легко получать представление о расположении текущего и соседних разделов.

В Visual Studio справка интегрирована со средой разработки (IDE), чтобы пользователь мог получить все необходимые ему сведения с учетом контекста разработки (рис. 2.8).

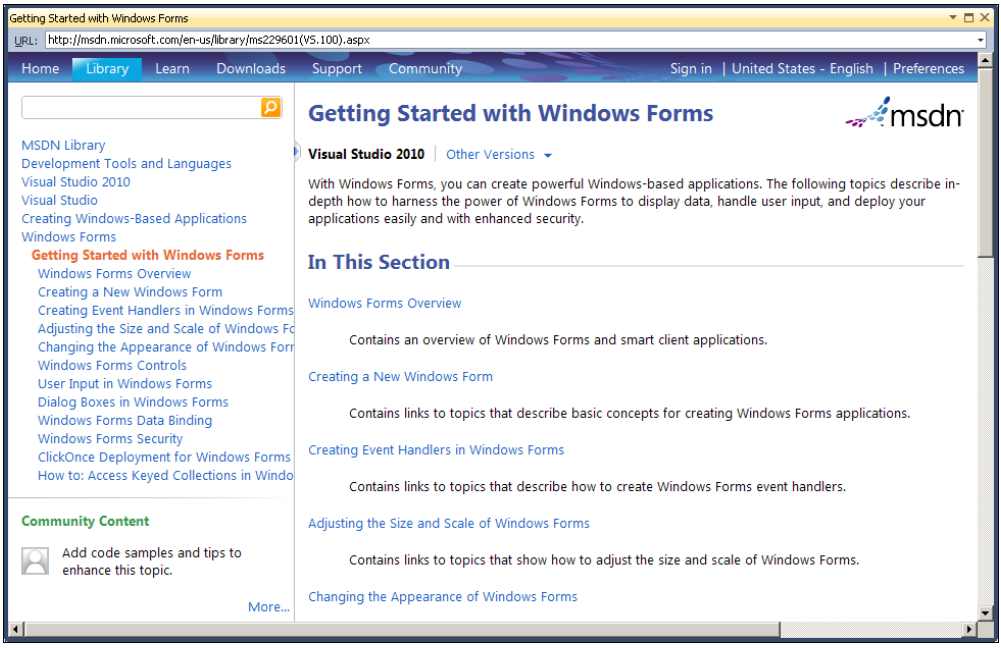


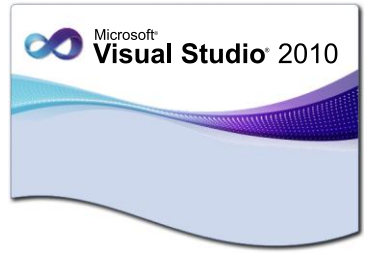
Рис. 2.8. Справка в окне браузера

Динамическая справка, вызываемая нажатием клавиши <F1>, отображает разделы документации, основываясь на выделенных элементах интерфейса или выводимых сообщениях об ошибках.

При разработке объектов и кодировании их функций в редакторах можно использовать функцию завершения операторов, чтобы получить сведения о синтаксисе, необходимом для ключевых слов, методов и свойств отдельного языка. Если необходимо подробнее изучить функциональные возможности библиотеки или объекта, можно использовать обозреватель объектов. Если требуется дополнительная информация об использовании и возможностях отдельного элемента языка программирования, можно также воспользоваться динамической справкой <F1>, чтобы вывести раздел справочника по языку.

Резюме

Visual Studio 2010 и .NET Framework 4.0 добавляют много новых функциональных возможностей в уже полнофункциональный программный продукт, каким является Visual Studio. Применение Visual Studio 2010 увеличивает производительность труда разработчиков при создании приложений, предназначенных для новой версии .NET Framework.



ГЛАВА 3

Интегрированная среда разработки

Чтобы эффективно управлять компонентами, используемыми на этапе разработки, например ссылками, подключениями данных, папками и файлами, в Visual Studio предусмотрены два типа контейнеров. Эти контейнеры называются *решениями* и *проектами*. Также Visual Studio предоставляет папки решений для того, чтобы структурировать связанные проекты по группам и затем выполнять действия над этими группами проектов. Частью интегрированной среды разработки является интерфейс для просмотра и управления этими контейнерами и связанными с ними элементами — **Solution Explorer**.

Создание проектов

Проект — это основная единица, с которой работает программист. Он выбирает тип проекта, а Visual Studio создает шаблон проекта в соответствии с выбранным типом.

В Visual Studio редко создается пустой проект, к которому затем добавляется код. Вместо этого вы указываете среде Visual Studio тип проекта, который вы хотите создать, и среда разработки генерирует файлы и программный код, которые служат основой для выбранного типа проекта. После этого вы можете работать над приложением, добавляя ваш код к созданному шаблону проекта. Например, если требуется создать проект, основанный на Windows Forms, то среда Visual Studio 2010 сгенерирует проект с пустой формой. Если потребуется создать консольное приложение, то Visual Studio сгенерирует файл program.cs, в котором уже будут основные пространства имен, класс `Program` и точка входа в приложение — статический метод `Main()`.

При создании проекта Visual Studio также определяется, должен ли проект компилироваться в консольное приложение, библиотеку классов, приложения Windows Forms, Windows Presentation Foundation и др. Выбранный тип проекта также указывает компилятору, какие внешние библиотеки необходимо подключить. Конечно, вы можете изменить все эти параметры настройки в процессе создания приложения.

Шаблоны Visual Studio

Ряд предопределенных шаблонов проекта и шаблонов элементов проекта устанавливаются при установке среды разработки Visual Studio. Эти шаблоны можно использовать для создания основного контейнера проекта и предварительного набора элементов, необходимых для разработки приложения, класса, элемента управления или библиотеки. Можно также использовать один из многих шаблонов элементов проекта для создания, например, приложения Windows Forms или страницы Web Forms, чтобы индивидуально изменять приложение по мере его разработки.

Шаблоны проектов Visual Studio предоставляют многократно используемую и настраиваемую основу для проектов и элементов, позволяющую ускорить процесс разработки приложений, поскольку избавляют программистов от необходимости создания новых проектов и элементов "с нуля".

Эти шаблоны предоставляют пользователям отправную точку для создания новых или расширения текущих проектов. Шаблоны проектов предоставляют файлы, необходимые для конкретного типа проекта, включают стандартные ссылки на сборки и задают свойства проекта и параметры компилятора по умолчанию. Шаблоны элементов могут варьироваться по сложности от одного пустого файла с правильным расширением имени до элементов из нескольких файлов, включая, например, файлы исходного кода с кодом-основой, файлы сведений о конструкторе и внедренные ресурсы.

В дополнение к установленным шаблонам можно разработать собственные шаблоны проектов или использовать шаблоны, созданные сторонними производителями. Все шаблоны проектов и элементов, независимо от того, поставляются они с Visual Studio или созданы сторонними производителями, работают одинаковым образом и состоят из следующих элементов:

- файлов, которые будут созданы при выборе этого типа шаблона. Сюда входят все файлы с исходным кодом, внедренные ресурсы, файлы проекта и т. д.;
- одного файла с расширением vstemplate. Этот файл содержит метаданные, которые предоставляют Visual Studio сведения, необходимые для отображения шаблона в диалоговом окне **New Project**.

Эти файлы сжаты в ZIP-файл и находятся в папке Program Files\Microsoft Visual Studio 10.0\Common7\IDE\ItemTemplatesCache\CSharp.

Создание проекта

Вы можете создать новый проект, выбирая **File | New | Project** в меню Visual Studio. При этом откроется диалоговое окно **New Project** с разнообразными видами проектов. Для рассмотрения работы в интегрированной среде разработки мы создадим простое консольное приложение с именем ConsoleApp (рис. 3.1).

Консольное приложение позволит ограничиться небольшим объемом кода и сосредоточиться на наиболее важных вопросах изучения работы в среде разработки Visual Studio 2010.

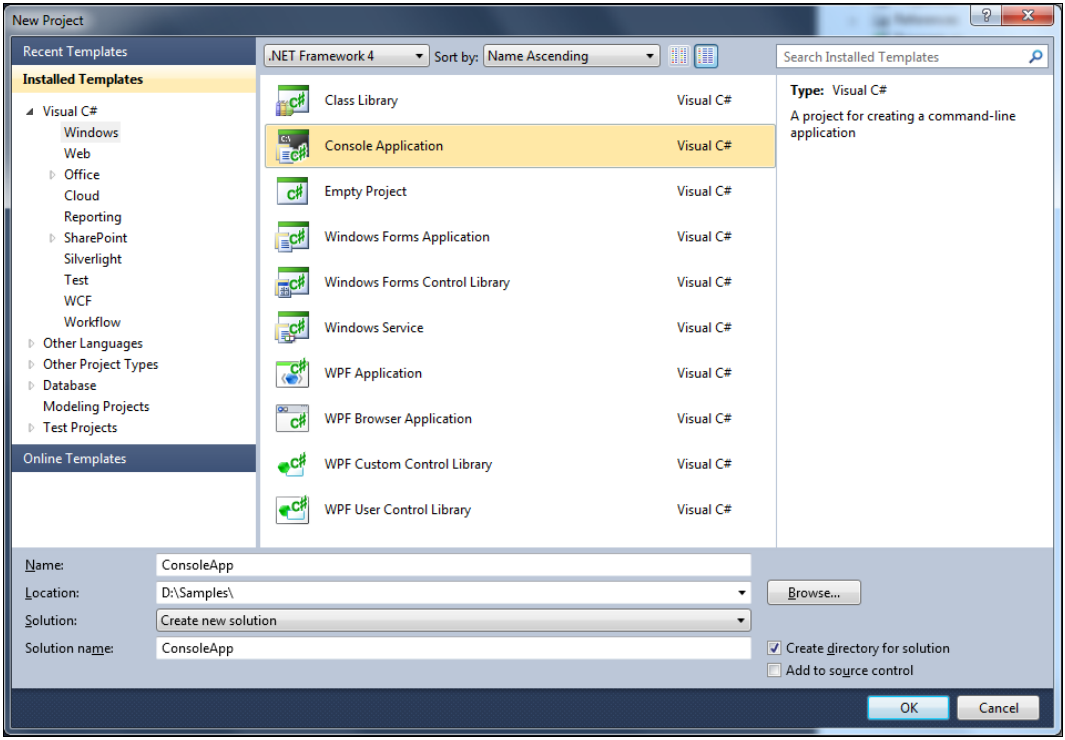


Рис. 3.1. Диалоговое окно **New Project**

На рис. 3.2 показано окно **Solution Explorer**, в котором отображается дерево файлов проекта, файл с кодом Program.cs и дополнительный файл AssemblyInfo.cs в папке Properties.

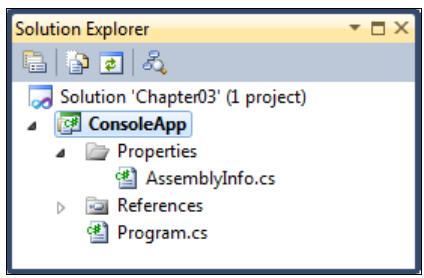


Рис. 3.2. Дерево проекта в окне **Solution Explorer**

Таков консольный проект, построенный по умолчанию, я не буду подробно рассматривать его содержимое, т. к. вы наверняка уже создавали подобные проекты.

Существует несколько способов откомпилировать и выполнить программу из среды Visual Studio. Например, можно нажать комбинацию клавиш <Ctrl>+ +<Shift>+<B> или выбрать в меню пункт **Build | Build Solution**. В качестве альтернативы можно щелкнуть по кнопке **Build**, расположенной на одноименной панели инструментов.

Чтобы выполнить программу без отладчика, можно нажать комбинацию клавиш <Ctrl>+<F5> или выбрать в меню пункт **Debug | Start Without Debugging**.

Программу можно выполнить, не выделяя компиляцию в отдельный этап. В зависимости от выбранных параметров среда IDE сохранит файл, откомпилирует и выполнит его.

Решения

В предыдущем разделе Visual Studio фактически уже создал решение — хотя это решение содержит пока только один проект в окне **Solution Explorer** (см. рис. 3.2), который представляет древовидную структуру, определяющую ваше решение.

Решение может содержать несколько проектов, тогда как проект обычно содержит несколько элементов. Фактически решение — это набор всех проектов, которые составляют пакет создаваемых вами программ.

Окно **Solution Explorer** позволяет вам группировать и управлять множеством файлов, которые составляют ваше приложение. Решение обычно содержит несколько проектов. Проект группирует относящиеся к нему файлы, как показано на рис. 3.3. Например, вы можете создать веб-сайт, приложение Windows Forms, библиотеку классов, консольное приложение и т. д.

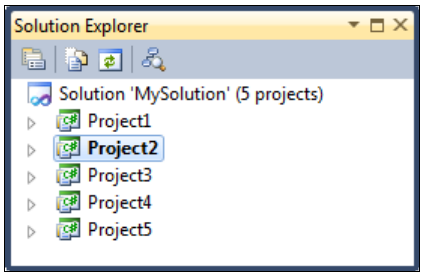


Рис. 3.3. Дерево проектов в окне **Solution Explorer**

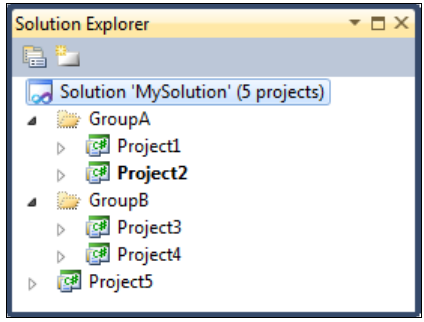


Рис. 3.4. Группировка проектов по папкам

Если решение состоит из множества проектов различного типа, их можно группировать по папкам (рис. 3.4). Папки решения являются средством структуризации в окне обозревателя решений; соответствующие папки Windows не создаются, однако лучше структурировать проекты на диске таким же образом, как и в решении. Для создания папки щелкните правой кнопкой мыши на значке решения, выберите пункт меню **Add | New Solution Folder** и присвойте папке имя.

Добавление проектов в решение

Вы можете создать новый проект двумя способами:

- выбрать **New Project** в меню **File** (поскольку вы уже сделали);
- выбрать **Project Add New** в меню **File**.

Если вы выберете команду **Add to solution**, в существующее решение с консольным проектом добавится новый проект, как показано на рис. 3.5.

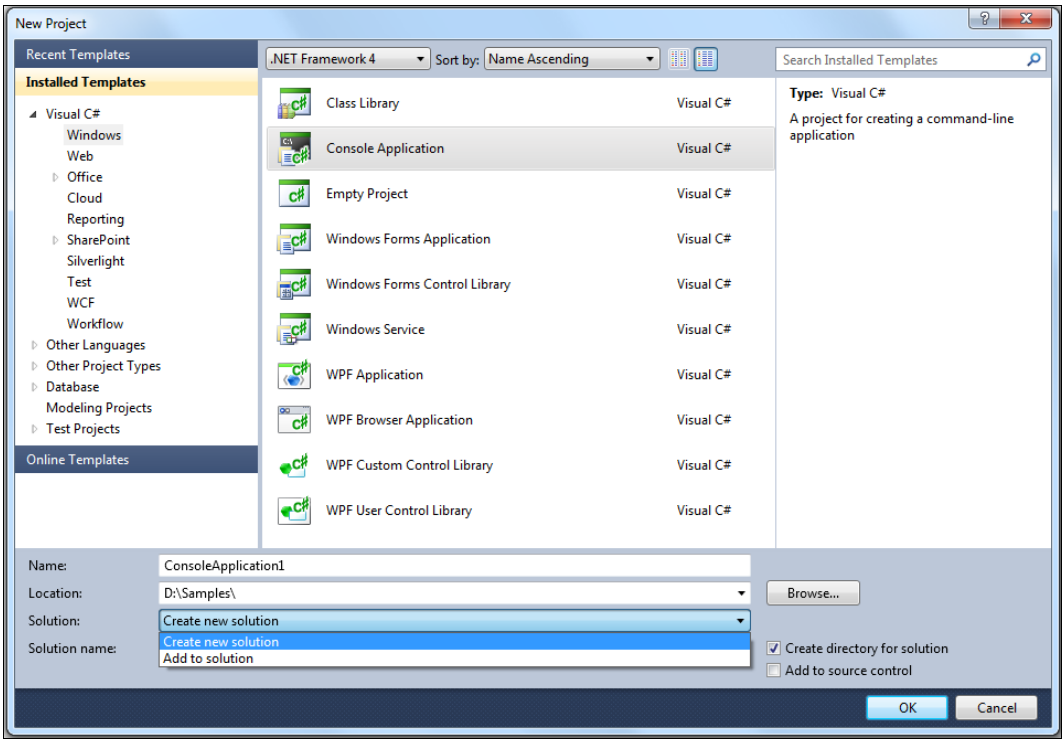


Рис. 3.5. Добавление нового проекта в решение

В соответствии с языковой независимостью решения, в Visual Studio новый проект не обязательно должен быть написан на C#. Допускается возможность размещения проекта C#, проекта Visual Basic и проекта C++ в одном и том же решении. Однако в нашей книге мы будем работать с проектами на языке C# .NET.

Если требуется переименовать какой-нибудь проект или файл проекта, лучше это сделать в окне **Solution Explorer**, т. к. среда разработки Visual Studio автоматически обновит любую ссылку на этот файл в другом проекте. Если же вы переименуете файл через Windows Explorer, Visual Studio не будет в состоянии определить местонахождение этого файла, и тогда потребуется вручную отредактировать проект и решение.

Установка стартового проекта

Когда вы компилируете решение, то компилируются все содержащиеся в нем проекты, но может выполняться только один из них. Поэтому необходимо отдельно указывать среде Visual Studio, какой из проектов должен запускаться на выполнение при нажатии клавиши <F5> или выборе команды **Start**. Если у вас имеется

один исполняемый файл и несколько библиотек, к которым он обращается, то ясно, что запускаться должен именно исполняемый файл. Стартовый проект будет отображаться в дереве решений жирным шрифтом (см. рис. 3.3 и 3.4 ранее в этой главе).

Конфигурации *Debug* и *Release*

При отладке в код добавляются дополнительные строки, обеспечивающие отображение важной отладочной информации. Совершенно очевидно, что было бы предпочтительно полностью удалить эти строки из исполняемого файла, прежде чем включать его в поставляемое программное обеспечение. Такое удаление можно выполнить и вручную, но разве эта задача не упростилась бы намного, если бы имелась возможность каким-то образом просто пометить соответствующие операторы, чтобы компилятор мог игнорировать их при компиляции окончательной версии приложения перед его поставкой?

Компиляция программного продукта на стадии отладки отличается от компиляции окончательной версии. Все, что Visual Studio должна сделать, для того чтобы обеспечить поддержку различных вариантов компоновки проекта, — это сохранить различные наборы вариантов компоновки, которые называются *конфигурациями*. Когда вы создаете проект, Visual Studio автоматически предоставляет вам две конфигурации, соответствующие отладочной (**Debug**) и окончательной (**Release**) версиям продукта:

- конфигурация **Debug** указывает на то, что оптимизация выполняться не должна, а дополнительная отладочная информация, наоборот, должна включаться в исполняемый код;
- конфигурация **Release** указывает на то, что оптимизация должна выполняться, а отладочная информация не должна включаться в исполняемый код.

Вам также предоставляется возможность самостоятельно определить параметры конфигурации. Это может потребоваться в тех случаях, когда вы, например, хотите настроить отдельные варианты компоновки проекта, предназначенные для использования на уровне профессионалов или на уровне предприятий, чтобы можно было поставлять две версии данного программного обеспечения.

Выбор конфигурации

В связи с тем, что среда Visual Studio в состоянии хранить подробную информацию о нескольких конфигурациях, возникает очевидный вопрос: каким образом она определяет, какую именно конфигурацию следует использовать для компоновки того или иного проекта? Суть ответа состоит в следующем: всегда существует активная конфигурация, которую Visual Studio и будет применять для компоновки проекта. Также обратите внимание, что конфигурации устанавливаются для каждого проекта в отдельности, а не для решения в целом.

По умолчанию, когда вы создаете проект, в качестве активной устанавливается конфигурация **Debug**. Чтобы установить в качестве активной другую конфигура-

цию, следует выбрать команду меню **Build | Configuration Manager**. При этом откроется окно свойств решения (рис. 3.6). Доступ к этой команде можно получить также через выпадающее меню основной панели инструментов Visual Studio.

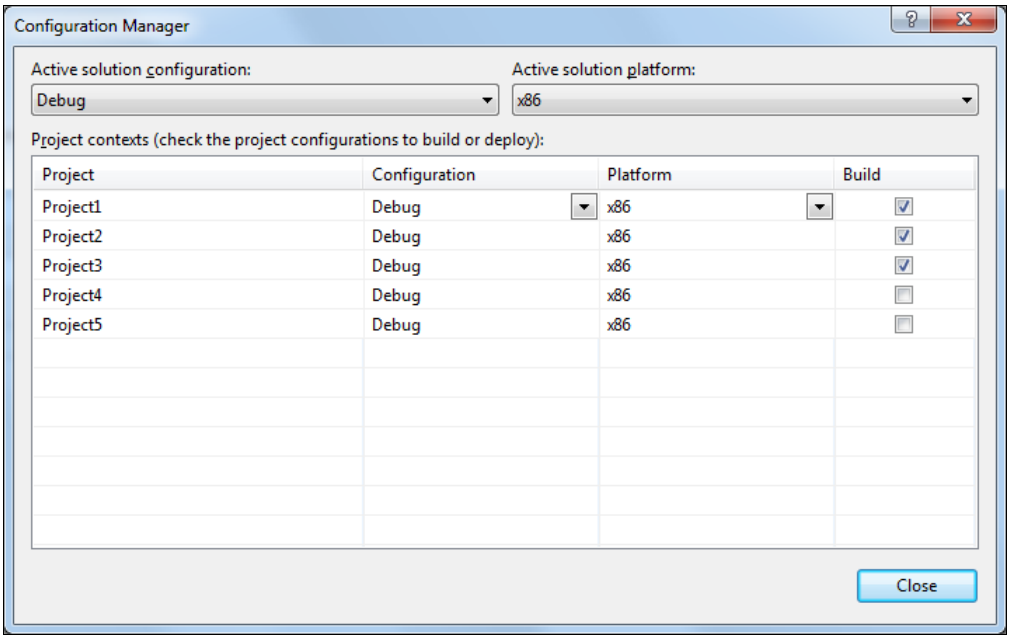


Рис. 3.6. Установка активной конфигурации

Редактирование конфигурации

Помимо выбора активной конфигурации вы также можете просматривать конфигурации и редактировать их. Для этого необходимо выделить соответствующий проект в окне **Solution Explorer**, а затем выбрать команду меню **Projects Properties**. В результате этого на экране отобразится диалоговое окно с весьма сложной структурой. Это же диалоговое окно можно вызвать и иначе, щелкнув правой кнопкой мыши на имени проекта в окне **Solution Explorer** и выбрав в открывшемся контекстном меню команду **Properties**.

В этом диалоговом окне отображается древовидная структура, позволяющая просматривать и редактировать множество общих аспектов проекта. Для полного их обсуждения в данной книге просто не хватило бы места, однако наиболее важные из них мы все же рассмотрим. На рис. 3.7 показан вид вкладки, на которой отображаются доступные свойства для отдельного приложения. При необходимости, например, можно поменять название сборки или пространство имен по умолчанию.

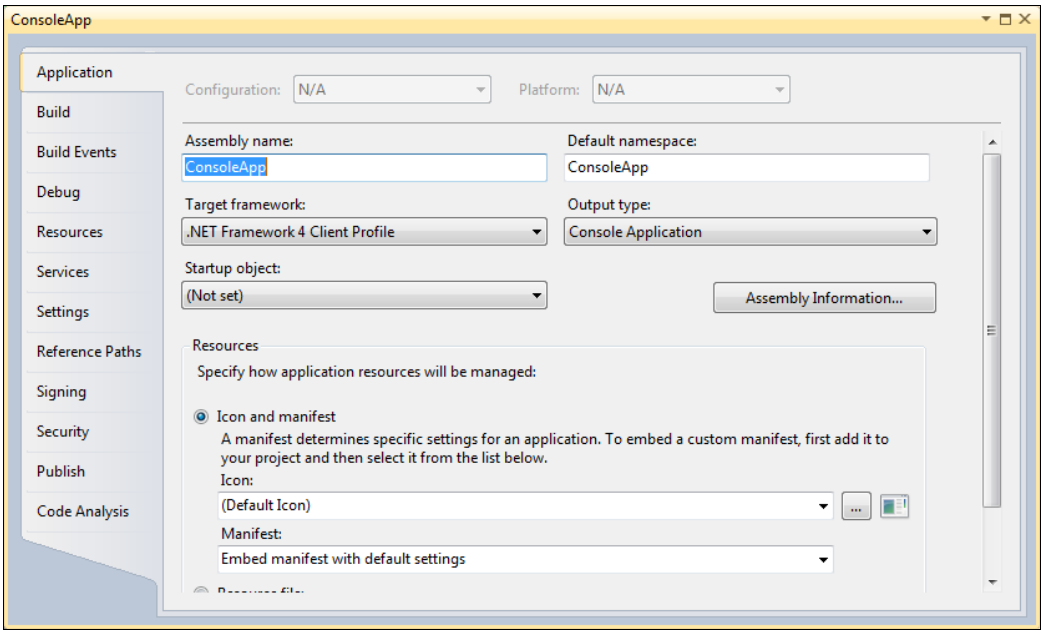


Рис. 3.7. Вкладка **Application** свойств проекта

Рефакторинг кода

Многие программисты разрабатывают свои приложения, кодируя сначала функциональные возможности, а затем они переделывают приложения, чтобы сделать их более управляемыми и более читаемыми. *Рефакторинг*, если кто незнаком с этим термином, является процессом изменения внутренней структуры программы, не затрагивающий ее внешнего поведения и имеющий целью облегчить понимание ее работы.

Поэтому среда Visual Studio 2010 включает ряд инструментальных средств для рефакторинга кода. Вы можете найти эти инструменты под опцией **Refactor** в главном меню Visual Studio. Инструменты для рефакторинга кода значительно упрощают выполнение рефакторизации кода не только в пределах одной страницы, но и по всему приложению. Кроме того, вы также получаете возможность выполнять следующие действия:

- изменять имена методов, локальных переменных, полей и множества других элементов;
- извлекать методы из выделенных фрагментов кода;
- извлекать интерфейсы на основании набора существующих элементов типов;
- превращать локальные переменные в параметры;
- переименовывать или переупорядочивать параметры.

В процессе создания учебных программ, описанных в этой книге, а также в дальнейшем, при создании ваших собственных приложений, вы сможете убедиться

в том, что новые средства для рефакторинга кода, предлагаемые средой разработки Visual Studio 2010, обеспечивают прекрасные возможности для того, чтобы сделать код более понятным, удобочитаемым и лучше структурированным.

Система окон

Visual Studio предоставляет множество окон, которые отображают информацию, необходимую для создания приложений. Некоторые окна, такие как **Solution Explorer**, которое мы уже рассмотрели ранее в этой главе, отображены по умолчанию, другие, например отладочные, появляются при запуске приложения в режиме отладки.

Полный список доступных окон расположен в главном меню Visual Studio под опцией **View** (рис. 3.8).

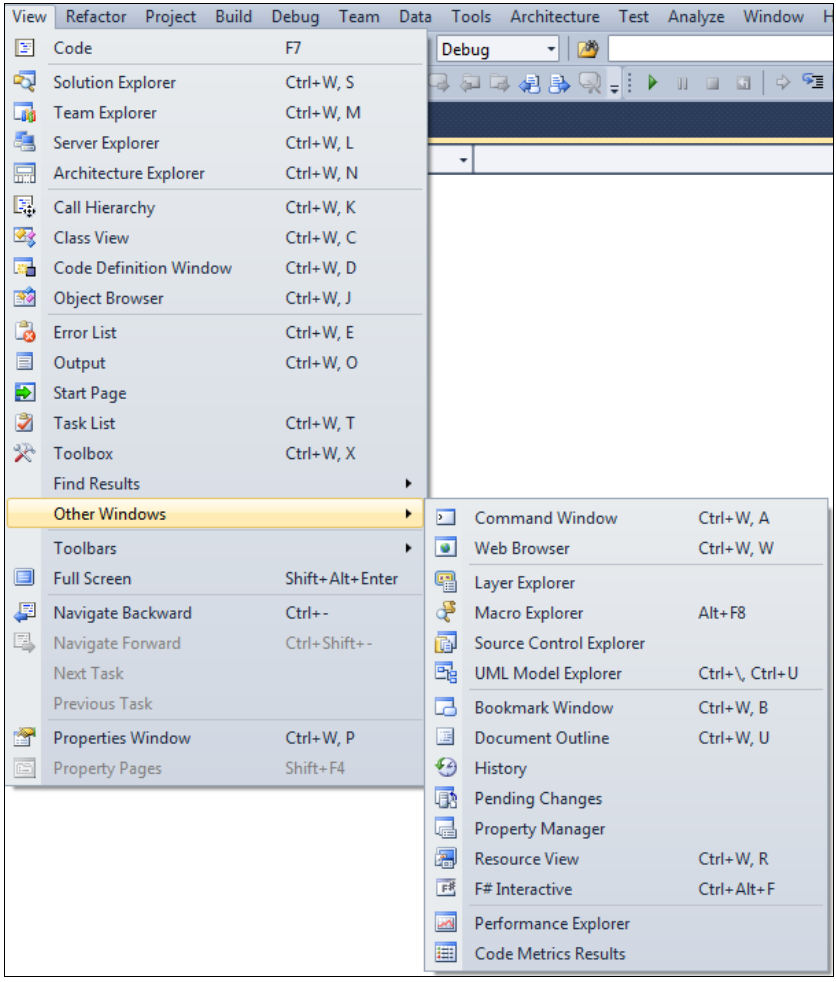


Рис. 3.8. Меню **View**

В целом, интегрированная среда разработки содержит два типа окон:

- окна инструментов;
- окна документов.

Размер области для просмотра и редактирования кода устанавливается в зависимости от размещения окон в интегрированной среде разработки.

Расположение окон

Окна инструментов и документов можно упорядочить перетаскиванием, командами меню **Window**, или щелкнув правой кнопкой мыши заголовок перемещаемого окна. Панели инструментов можно перетаскивать или упорядочивать с помощью диалогового окна **Customize** (Настройка), доступ к которому можно получить через команду меню **Tools | Customize.**

Любое окно инструментов или окно документов может быть отстыковано от интегрированной среды разработки и помещено в любое место на рабочем столе. Если два связанных окна документов отображаются одновременно, при изменении содержимого в любом из них обновляются оба этих окна.

Окна инструментов можно прикрепить к одной из сторон фрейма интерфейса IDE. При перетаскивании окна инструментов в новое расположение в интегрированной среде разработки появляется маркер в виде ромба. Маркер помогает закрепить окно инструментов на одной из четырех сторон интегрированной среды разработки или во фрейме редактирования.

Чтобы переместить закрепляемое окно без его привязки к какому-либо месту, нужно при его перетаскивании держать нажатой клавишу <Ctrl>.

На рис. 3.9 изображены специальные маркеры, которые появляются при перетаскивании окна инструментов или документов к центру интегрированной среды разработки.

При перетаскивании окон появляется маркер в виде ромба. Четыре стрелки ромба указывают на четыре стороны панели редактирования. Если окно является окном инструментов, то дополнительные четыре стрелки указывают на углы окна IDE.

Когда перетаскиваемое окно достигнет нужного расположения, наведите указатель на соответствующую часть ромба-маркера. Указанная область будет отображена затемненной. Чтобы закрепить окно, отпустите кнопку мыши, и оно останется прикрепленным в выбранном месте среды разработки.

Например, если обозреватель решений закреплен на правой стороне среды разработки, а вы хотите закрепить его на левой стороне, перетащите обозреватель решений в центр среды разработки, наведите указатель на самую левую стрелку ромба и отпустите кнопку мыши.

Кроме того, окно инструментов можно прикрепить к боковой, верхней или нижней части окна среды разработки, перетащив его в сторону до появления второго маркера в форме ромба. Щелкните одну из четырех стрелок, чтобы закрепить окно рядом с данной частью боковой стороны окна.

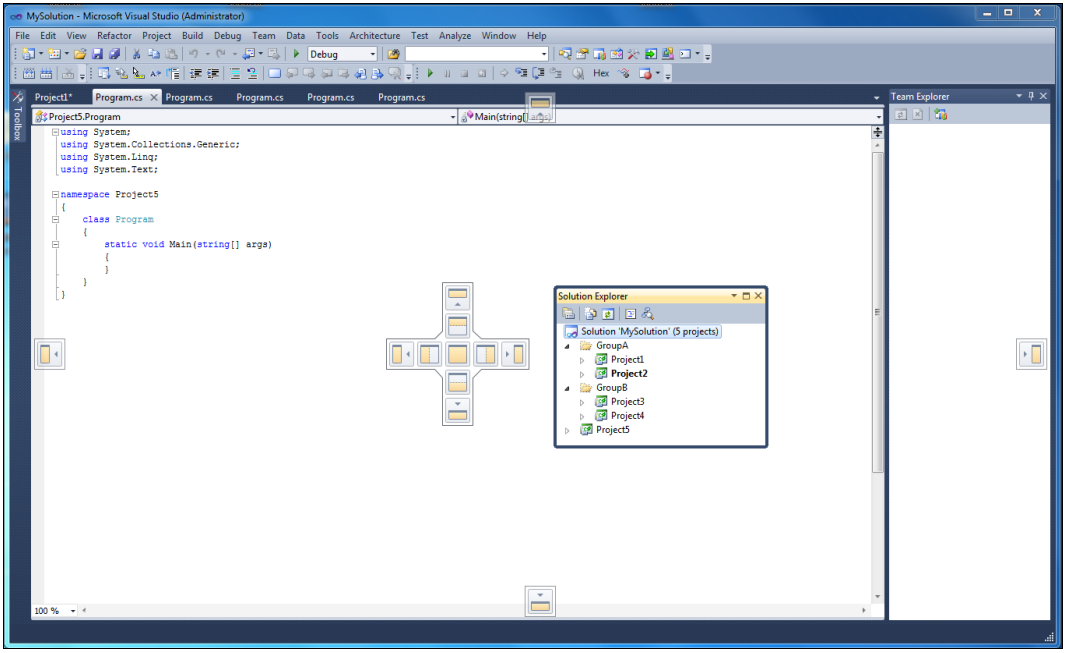


Рис. 3.9. Перетаскивание и закрепление окна

Все окна инструментов, названия которых встречаются в меню **View**, поддерживают возможность *автоматического скрытия* (значок канцелярской кнопки в заголовке окна). Автоматическое скрытие сдвигает окно в сторону, когда активно другое окно. Когда окно скрыто, его имя и значок отображаются на вкладке на краю интегрированной среды разработки. Чтобы снова сделать окно активным, наведите указатель на вкладку, и оно вернется в поле зрения.

Окно *Properties*

Окно свойств **Properties** — основной инструмент настройки формы и ее компонентов. Содержимое этого окна представляет собой весь список свойств выбранного в данный момент компонента или формы. Вызывается это окно несколькими способами:

- в меню **View** выбираем пункт **Properties Window** (или используем клавишу <F4>);
- на выбранном объекте щелкаем правой кнопкой мыши и в контекстном меню находим пункт **Properties**;
- выбираем объект и нажимаем клавишу <F4>;
- просто выбираем объект и переходим в окно **Properties**.

При создании проекта в окне **Properties** отображаются свойства проекта. Кроме того, окно **Properties** позволяет управлять размером, внешним видом и поведением создаваемых элементов управления.

Окно **Properties** также группирует схожие свойства в наборы (для облегчения доступа). На рис. 3.10 показано окно **Properties**.

Обратите внимание, что по умолчанию это окно группирует схожие свойства в разделы при помощи категорий **Advanced**, **Misc** и др. При желании можно отключить группировку свойств по категориям и отсортировать список свойств в алфавитном порядке путем нажатия значка **AZ** на панели инструментов. Наконец, окно **Properties** позволяет привязать события элемента управления к коду внутри нашего приложения.

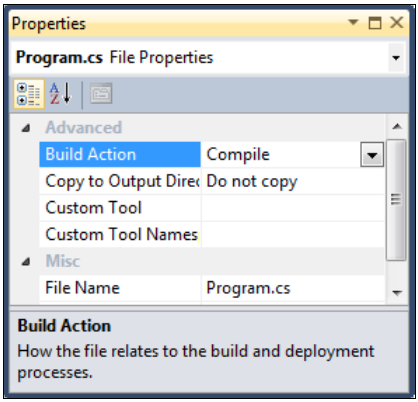


Рис. 3.10. Окно **Properties**

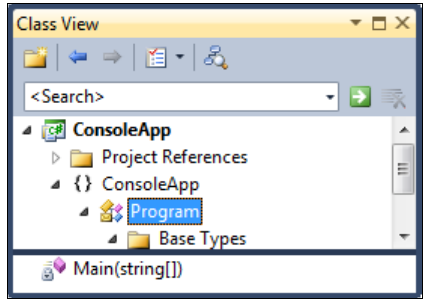


Рис. 3.11. Окно **Class View**

Окно *Class View*

Окно просмотра структуры классов **Class View** позволяет перемещаться в коде по выбранному объекту и содержит методы, классы, данные всего листинга проекта.

В Visual Studio окно **Class View** в действительности выступает не как самостоятельное окно, а как вкладка окна **Solution Explorer**. По умолчанию окно **Class View** даже не появляется в окне **Solution Explorer**. Чтобы отобразить его на экране, выберите в главном меню пункт **View | Class View** или воспользуйтесь сочетанием клавиш <Ctrl>+<Shift>+<C>.

В окне **Class View** (рис. 3.11) отображается иерархия пространств имен и классов, присутствующих в вашем коде, в виде дерева, которую вы можете развернуть для получения более подробной информации о том, какие классы содержатся в пространствах имен и какие элементы содержатся в классах.

Удобной возможностью, предлагаемой окном **Class View**, является то, что если вы щелкнете правой кнопкой мыши на имени любого элемента, к исходному коду которого у вас имеется доступ, то в открывшемся контекстном меню будет присутствовать команда **Go To Definition** (Перейти к определению), позволяющая перейти к тому месту программы в окне редактора кода, в котором находится определение данного элемента. Того же результата можно добиться, дважды щелкнув на элементе в окне **Class View** или щелкнув правой кнопкой мыши на названии эле-

мента в редакторе исходного кода и выбрав ту же команду в открывшемся контекстном меню.

Кроме того, контекстное меню позволяет добавить в класс поле, метод, свойство или индексатор. Это означает, что вы устанавливаете детальные характеристики соответствующего элемента в диалоговом окне, и для вас автоматически генерируется соответствующий код.

Окно *Object Browser*

В процессе написания кода часто требуется информация о том, какие методы и другие элементы программного кода доступны в базовых классах и других библиотеках, на которые в ваших сборках имеются ссылки. Для этого предназначено окно браузера объектов **Object Browser**. В Visual Studio 2010 для доступа к этому окну надо выбрать в главном меню пункт **View | Object Browser**.

Окно **Object Browser** по своему внешнему виду напоминает окно **Class View**. Окно **Object Browser** тоже отображает древовидное представление структуры классов вашего приложения, позволяя просматривать члены каждого класса. Незначительное отличие пользовательского интерфейса состоит в том, что члены класса отображаются на отдельной панели, а не в дереве класса. Действительно важным отличием является то, что с помощью этого окна вы можете просмотреть не только пространства имен и классы, входящие в состав вашего проекта, но и все сборки, на которые в вашем проекте имеются ссылки. На рис. 3.12 представлен вид окна **Object Browser** при просмотре класса `Console`, входящего в состав базовых классов .NET.

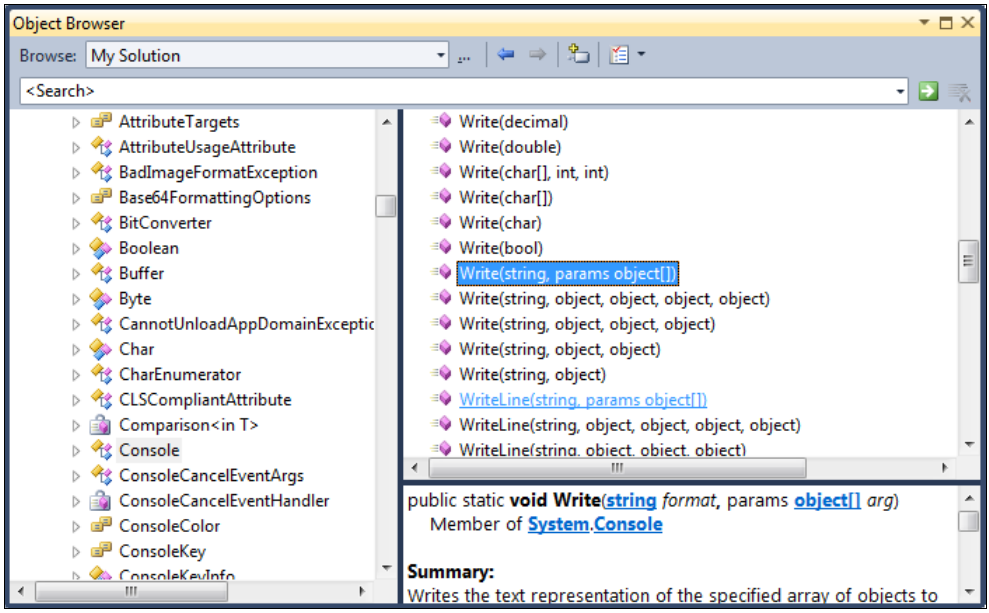


Рис. 3.12. Окно **Object Browser**

При работе с окном **Object Browser** необходимо учитывать, что классы в нем группируются сначала по сборкам, в которых они находятся, и лишь затем по пространствам имен. К сожалению, поскольку пространства имен для классов часто разбросаны по нескольким сборкам, нахождение определенного класса может оказаться затруднительным, если вы не знаете, какой сборке он принадлежит.

Окно *Code Definition*

В окне **Code Definition** (Определение кода) выводится исходный код для объекта или элемента. Если исходный код для этого элемента недоступен, то интегрированная среда разработки отображает метаданные в виде исходного кода в окне **Code Definition**.

Например, если в редакторе расположить курсор в пределах слова Console, то метаданные типа `Console` будут показаны в виде исходного кода в окне **Code Definition**. Исходный код напоминает объявление класса `Console`, но не содержит его реализации. Внешний вид окна **Code Definition** показан на рис. 3.13.

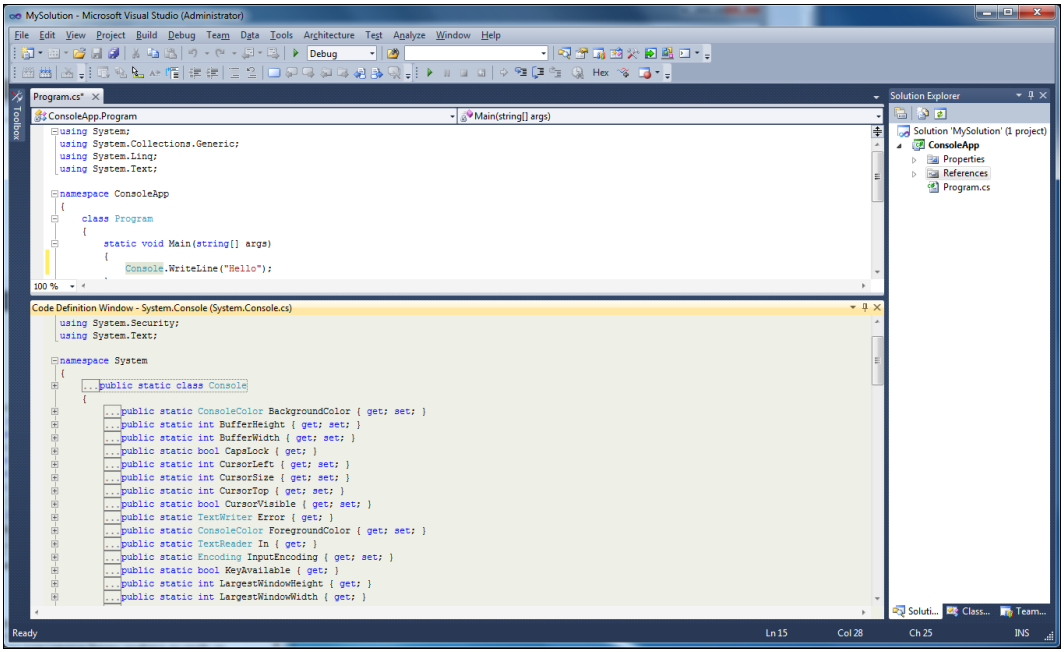


Рис. 3.13. Окно **Code Definition**

Если требуется просмотреть объявление элемента, который присутствует в окне **Code Definition**, щелкните правой кнопкой мыши этот элемент и выберите команду **Goto definition**.

Окно *Call Hierarchy*

В Visual Studio 2010 появилась новое окно **View Call Hierarchy** для просмотра иерархии вызовов, позволяющее узнать, где в вашем исходном коде используется определенный метод или свойство, а также позволяющее быстро пройтись по графу дерева вызовов в рамках вашего исходного кода.

Для просмотра иерархии вызовов выберите название метода или свойства в вашем исходном коде и либо нажмите комбинацию клавиш <Ctrl>+<K>, <Ctrl>+<T>, либо щелкните правой кнопкой мыши и выберите из контекстного меню пункт **View Call Hierarchy**. В результате появится вспомогательное окно **Call Hierarchy**, которое по умолчанию располагается под редактором кода (рис. 3.14).

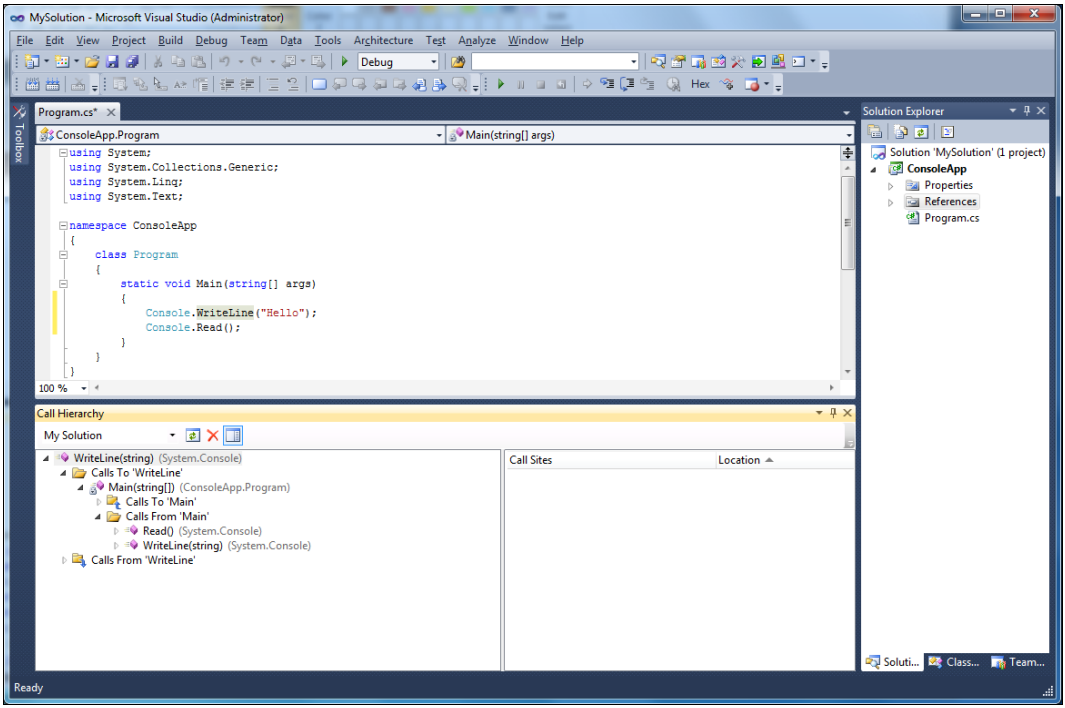


Рис. 3.14. Окно **Call Hierarchy**

Таким образом, в ходе разработки вы можете быстро проходить по множеству файлов с исходным кодом вашего решения, чтобы лучше понять взаимосвязи между классами и методами.

Окно *Server Explorer*

Окно **Server Explorer**, показанное на рис. 3.15, используется для получения сведений о компьютерах в сети в процессе написания кода.

Окно **Server Explorer** позволяет получать информацию о соединениях с базами данных, службах, журналах регистрации событий, а также другую подобную информацию.

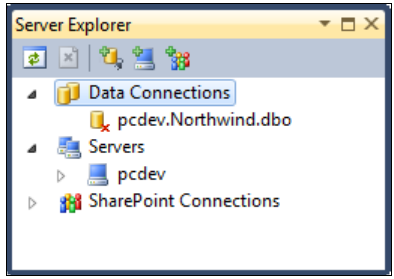


Рис. 3.15. Окно **Server Explorer**

Окно **Server Explorer** связано с окном **Properties**, поэтому, например, если вы развернете дочерний узел **Services** и щелкнете на некоторой службе, то свойства этой службы отобразятся в окне **Properties**.

Окно *Task List*

В Visual Studio поддерживается список задач — окно **Task List**. В программу включаются комментарии с описанием действий, которые предполагается выполнить в будущем; тип задачи определяется специальным ключевым словом, следующим после знака комментария. Всего определены три встроенные категории задач — **TODO**, **HACK** и **UNDONE**. Комментарии с задачами выводятся в окне, вызываемом командой **View | Other Windows | Task List** (или комбинацией клавиш <Ctrl>+<Alt>+<K>). Пример окна **Task List** показан на рис. 3.16.

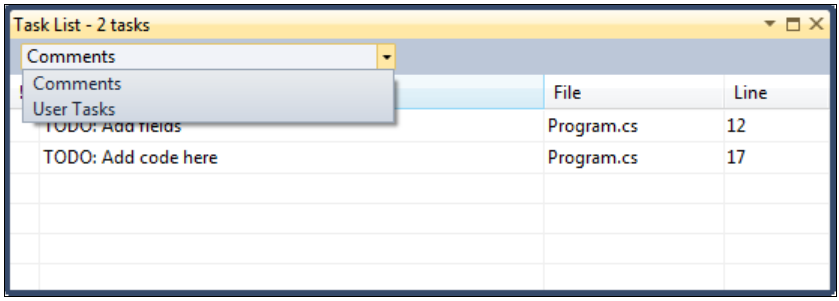


Рис. 3.16. Окно **Task List**

Такие комментарии с задачами, оставленные в коде, может автоматически находить **Task List** и сообщать о них в своем окне. При открытии проекта он сканируется на наличие этих комментариев, и если они встречаются, то создаются записи в списке задач. Такие комментарии в исходном тексте программы начинаются с признака комментария применяемого языка программирования, за которым следует маркер примечания, двоеточие и текст, который будет отображаться в списке **Task List**, например:

```
// TODO: Написать запрос к базе данных
```

Специальные маркеры, которые ищет **Task List**, определяются в диалоговом окне **Options**, показанном на рис. 3.17, где можно добавить новые маркеры и удалить или изменить существующие.

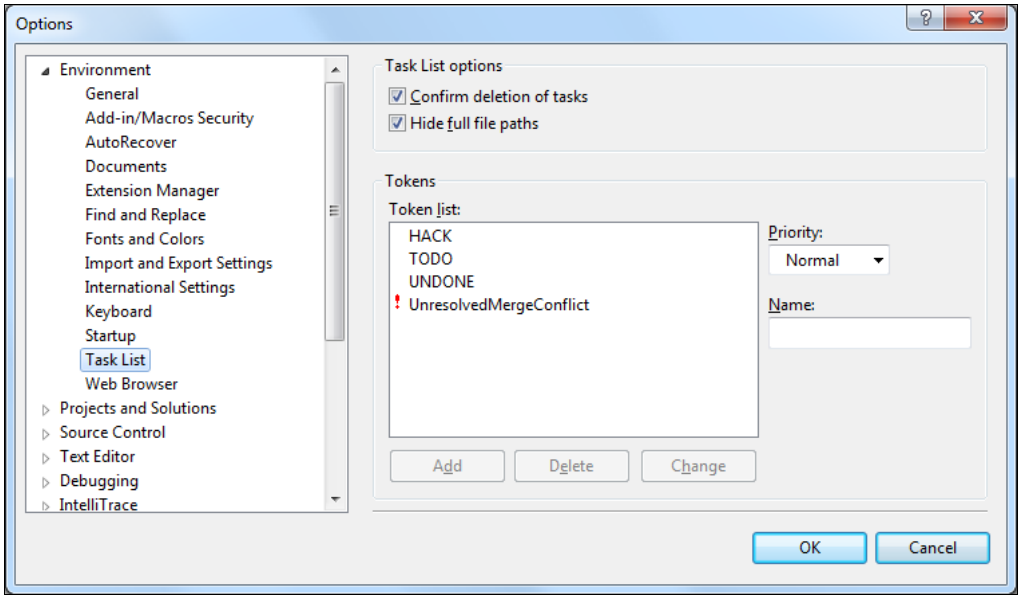


Рис. 3.17. Окно добавления новых маркеров для **Task List**

Редакторы кода

Visual Studio 2010 имеет несколько текстовых редакторов (редакторов кода). Все редакторы кода, независимо от языка, на которым пишется код, построены по одинаковому принципу и предоставляют основной набор функциональных возможностей, такие как подсветка синтаксиса, поля выбора, способность сворачивать вложенные элементы.

Форматирование кода

Редактор кода обрабатывает отступы и пробельные символы для того, чтобы ваш код был понятным и читабельным. Он обеспечивает технологию IntelliSense и дописывание операторов (для того чтобы освободить вас от необходимости поиска или запоминания каждой библиотеки объектных модулей или ключевого слова). Он группирует код в блоки, обеспечивает расцветку ключевых слов и комментариев, выделяет ошибки, выделяет новый код относительно ранее компилировавшегося. Вообще, редактор кода Visual Studio делает очень много для того, чтобы ваш труд был производительным. Рисунок 3.18 показывает код для консольного приложения из этой главы.

На рис. 3.18 обратите внимание на небольшие знаки "–" слева окна, так отмечаются пункты, где редактор кода предполагает начало нового блока программы. С их помощью код группируется в логические области. Вы можете использовать знак минуса для того, чтобы закрыть целый класс, метод, свойство или другую подобную группу. Эта возможность позволяет вам скрывать тот код, с которым вы в данный момент не работаете.

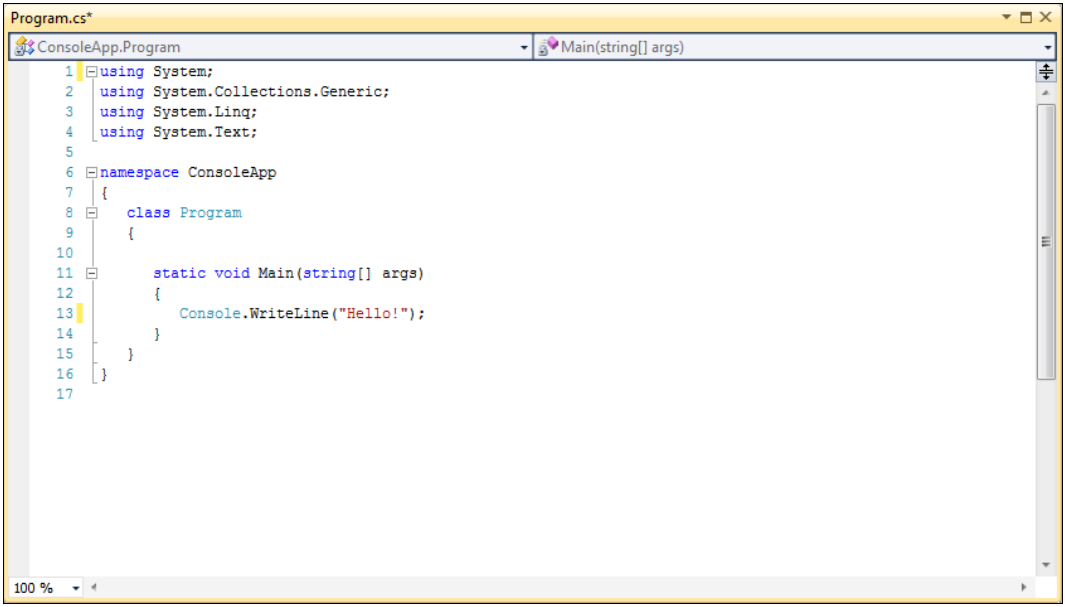


Рис. 3.18. Группировка программного кода в блоки

Вы можете также создавать ваши собственные (именованные) области кода для этой же цели. Если необходима дополнительная группировка кода, вы можете указать свои собственные блоки свертывания кода с помощью директив препроцессора `#region` и `#endregion` (рис. 3.19).

Редактор кода автоматически обнаруживает директиву `#region` и помещает новый знак "–" около директивы, как показано в рис. 3.20, разрешая пользователю закрыть область. Включение этого кода в области означает, что вы можете заставить редактор кода закрывать блоки программы, отмечая область директивой `#region`. Компилятор игнорирует директивы и компилирует код программы как обычно.

- Новый код внутри областей помечается цветной линией (см. рис. 3.19 и 3.20). Желтый цвет используется для нового кода, который еще не сохранен. Линия становится зеленой после сохранения и исчезает, когда вы закроете и вновь откроете файл. Эта функциональная возможность позволяет вам отслеживать места выполненных изменений в коде. Имя открытого кодового файла показано в заголовке окна кода. Звездочка указывает, что код уже изменился с момента последнего сохранения.

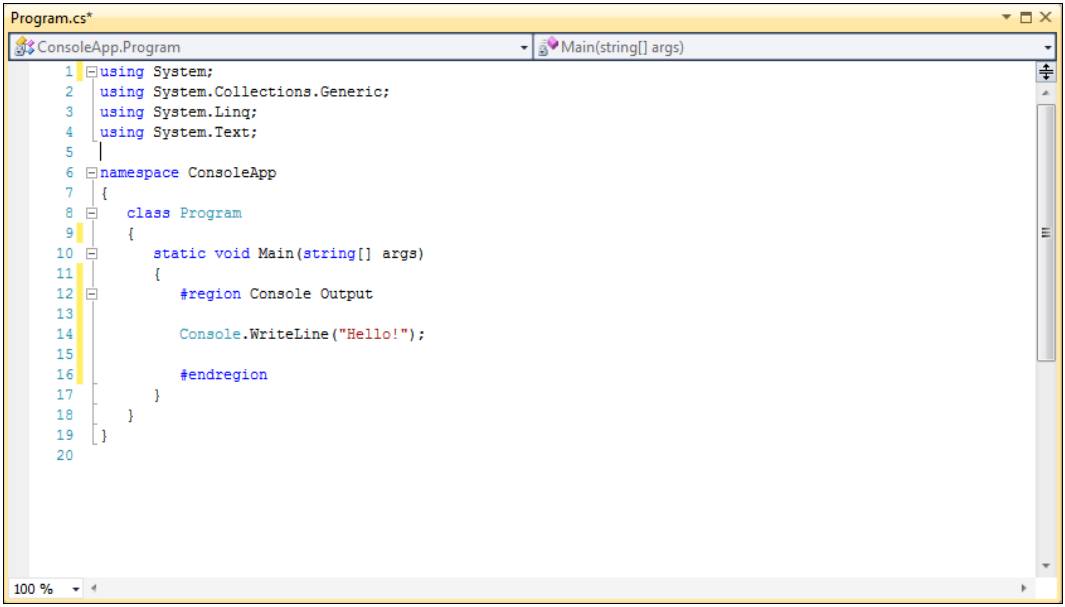


Рис. 3.19. Создание именованной области кода

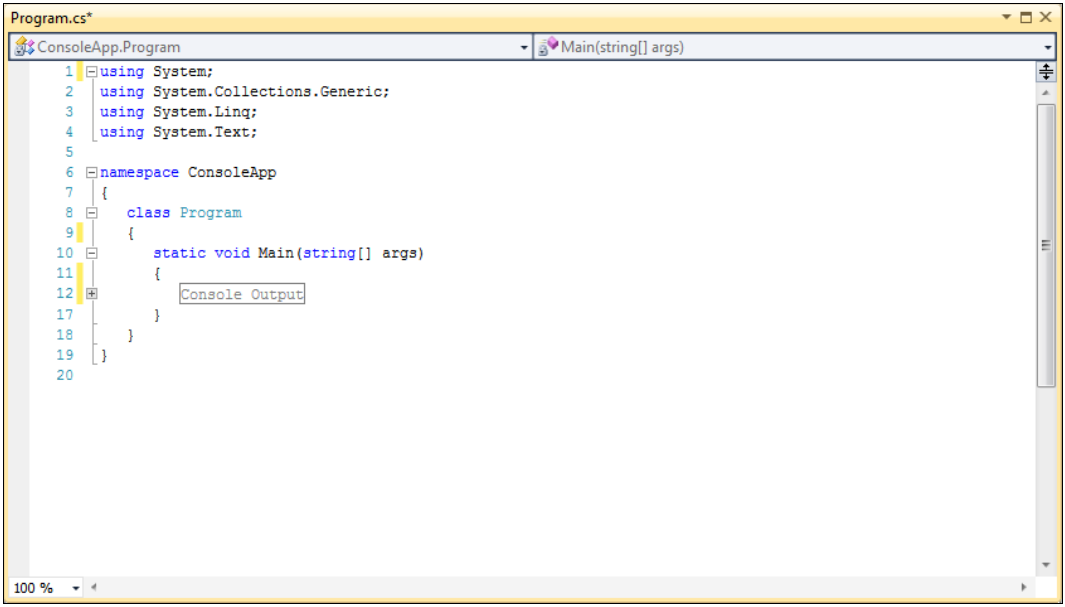


Рис. 3.20. Закрытый блок программного кода

- При наборе кода запускается функция IntelliSense. Для быстрого нахождения в списке нужного элемента вы можете использовать клавиши со стрелками. При наведении указателя мыши на элемент вам будут показаны подробности данного элемента (текст подсказки будет справа). Вы можете нажать клавишу <Tab> для дописывания элемента внутри IntelliSense.

- Код выделяется различными цветами. По умолчанию ключевые слова имеют синий цвет, комментарии — зеленый, текст — черный, пользовательские типы — голубой, строковые значения — красный цвет и т. д.
- Два выпадающих списка в верхней части редактора кода позволяют вам перемещаться между классами в файле (левый выпадающий список) и методами, полями и свойствами данного класса (правый выпадающий список).

Редактор кода также проводит анализ написанного кода и подчеркивает большинство синтаксических ошибок короткой волнистой линией. При наведении курсора мыши на подчеркнутый текст появляется подсказка, объясняющая ошибку.

Если вас не устраивает форматирование и подсветка синтаксиса, предлагаемые редактором кода по умолчанию, можно использовать диалоговое окно **Options**, которое вызывается из главного меню командой **Tools | Options**, для изменения фонового цвета редактора или цвета и шрифта различного текста внутри редактора. Вы можете также включить нумерацию строк и управлять отступами (табуляцией) и пробельными символами. Можно также настроить язык и специфические для редактора опции. Например, на рис. 3.21 показано диалоговое окно **Options** для выбора параметров шрифта.

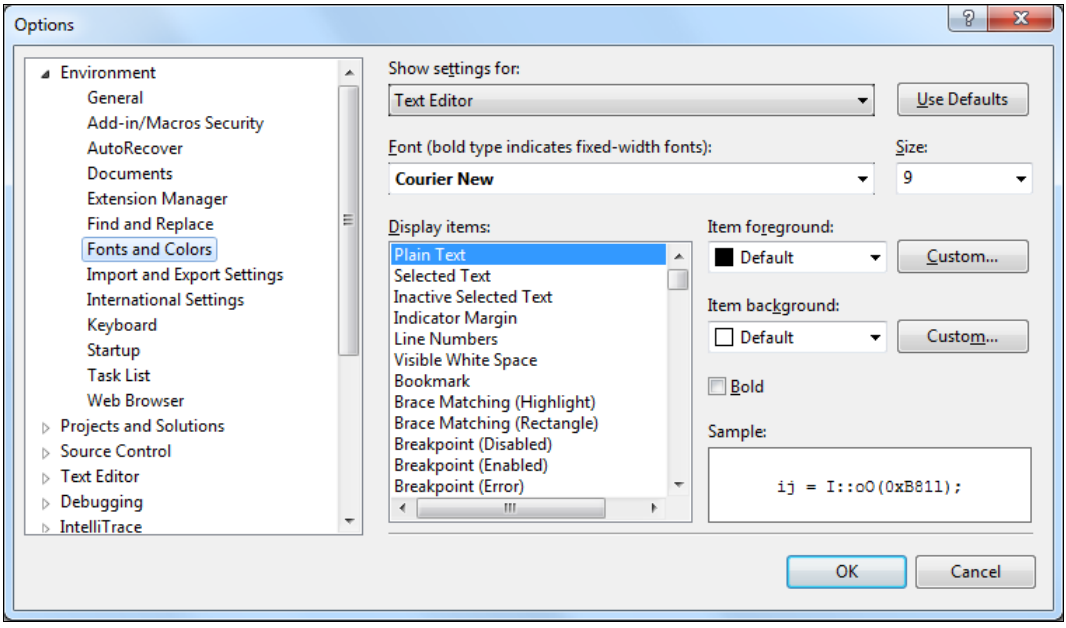


Рис. 3.21. Диалоговое окно **Options** с настройками шрифтов и цветов

В каждой компании, занимающейся разработкой программного обеспечения, свои стандарты оформления кода. Параметры форматирования для каждого из языков, поддерживаемых в IDE Visual Studio, устанавливаются в диалоговом окне **Options** в узле **Text Editor**. В этом узле вы можете манипулировать большим количеством настроек для текстового редактора. Например, можно настроить автоматическое форматирование кода в редакторе (рис. 3.22).

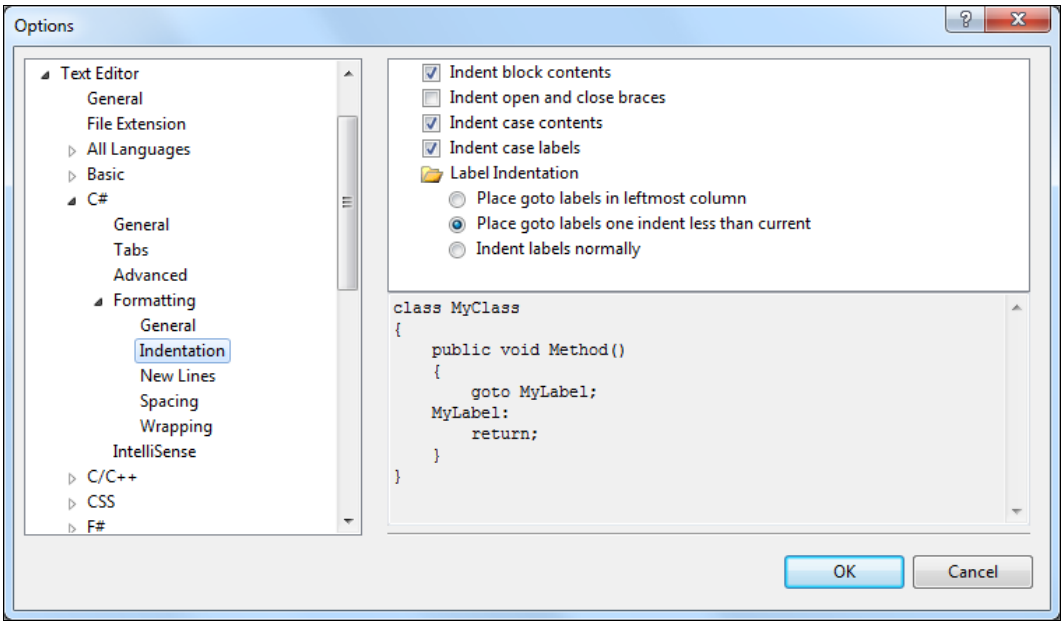


Рис. 3.22. Управление форматированием кода в диалоговом окне **Options**

Если требуется, чтобы все фигурные скобки в коде стояли на отдельных строках, или, наоборот, чтобы они находились в начале строки, с которой начинается блок кода, то можете настроить нужное форматирование кода в узле **Text Editor**.

Текстовые редакторы также имеют несколько инструментов для отображения нумерации строк, пометки строк кода, поиска и замены текста в исходных файлах.

Отображение нумерации строк

Нумерацию строк можно включить для любого документа, загруженного в редактор кода. Нумерация строк включается в диалоговом окне **Options**, в узле **Text Editor | All Languages | General**.

При перемещении по коду большого размера очень полезна возможность сразу перейти на определенную строку кода. Для такого перехода нажмите в редакторе комбинацию клавиш <Ctrl>+<G>. Это приведет к появлению диалогового окна **Go To Line** (Перейти на строку), показанного на рис. 3.23. В этом окне имеется поле

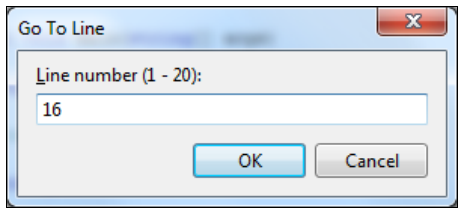


Рис. 3.23. Переход на строку

для указания номера строки, на которую необходимо перейти, и даже указана возможная "область" для перехода (приводится диапазон номеров строк для текущего документа). Ввод требуемого номера строки приведет к переходу курсора на начало указанной строки.

Окно *Bookmarks*

Окно **Bookmarks** (Закладки) позволяет помечать определенную строку кода. Закладки помогают решить проблему перемещения по большим кодовым файлам. После размещения закладки на строке кода вы в любое время сможете мгновенно вернуться к этой строке. Когда вам приходится работать со множеством закладок, вы можете свободно перемещаться между помеченными закладками строками кода. Это чрезвычайно полезная возможность. Если вы — разработчик, которому приходится иметь дело с большой базой исходных кодов, то в этих кодах неизбежно будут иметься интересные места, которые вы захотите просмотреть в редакторе. Вспомним, что окно текстового редактора имеет средство навигации (раскрывающиеся списки типов и членов), однако это не лучшие инструменты для таких случаев, когда интересующая вас строка может быть произвольным оператором, похороненным под миллионами строк кода.

Закладки визуально отображаются в поле индикаторов текстового редактора. Например, на рис. 3.24 закладка установлена на строке 14.

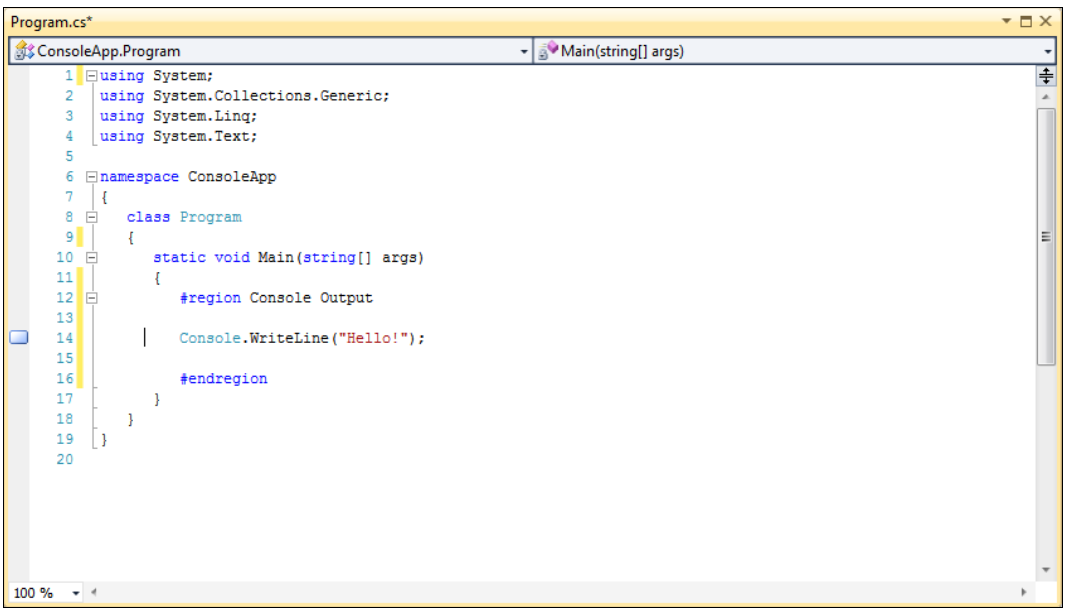


Рис. 3.24. Закладка в редакторе кода

Для того чтобы добавить закладку или перемещаться по закладкам, необходимо использовать либо панель инструментов текстового редактора, либо окно **Bookmarks**. Вы можете увидеть окно **Bookmarks** (рис. 3.25) посредством выбора

в меню команды **View | Bookmark Window**. Это окно представляет собой панель инструментов для операций с закладками и список всех имеющихся закладок вместе с их реальным физическим местоположением — имя файла и номер строки в этом файле.

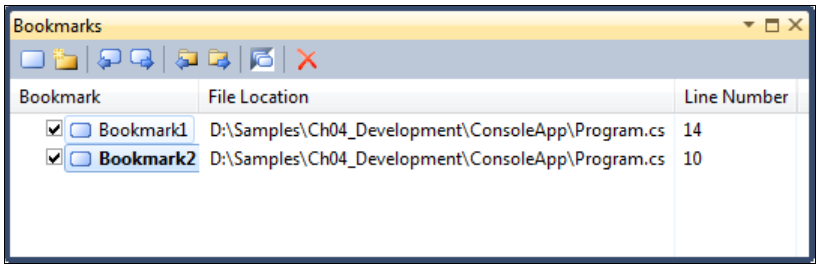


Рис. 3.25. Окно закладок **Bookmarks**

Для того чтобы установить закладку на данную строку кода, необходимо сначала поместить курсор в текстовом редакторе на нужную строку, а затем нажать кнопку **Toggle Bookmark**.

Такой же процесс используется и для снятия закладки. Нажатие кнопок **Forward** и **Back** в окне закладок приведет к перемещению курсора редактора вперед и назад по всем имеющимся закладкам. Список закладок этого окна является также полезным механизмом для быстрого включения или отключения закладки при помощи флажка рядом с закладкой и для переименования закладки. Щелчок правой кнопкой мыши по закладке позволит вам переименовать ее в нечто более осмысленное, нежели "Bookmark1".

Окно **Bookmarks** также позволяет создать каталог закладок и структурировать закладки по темам. Каталоги создаются кнопкой **New Folder** панели инструментов окна. Затем вы можете создать закладку и переместить ее в требуемый каталог закладок.

Визуальные конструкторы и панели инструментов

Visual Studio 2010 также поставляется с множеством визуальных конструкторов. Они позволяют создавать элементы, которые составляют ваше приложение. Эти элементы включают: формы Windows, веб-формы, диаграммы классов, схемы XML и т. д. *Визуальный конструктор* — это холст, на котором вы при помощи мыши создаете различные элементы посредством перетаскивания, изменения размеров и т. д.

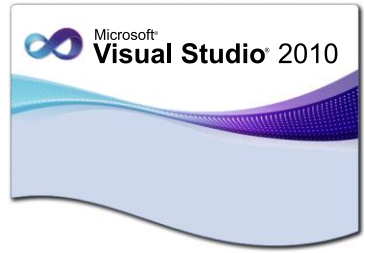
Среда Visual Studio 2010 предоставляет на инструментальных панелях большое количество компонентов для разработки приложений. Категории компонентов, доступных через инструментарий, зависят от типа проекта.

Все визуальные конструкторы, независимо от их типа, работают практически одинаково. Они занимают центральное место внутри интегрированной среды раз-

работки и имеют вид окон с вкладками, окруженных различными меню, панелями инструментов и прочими панелями. Эти конструкторы мы изучим в процессе создания различных типов приложений на протяжении всей книги.

Резюме

Чтобы быстро и эффективно создавать приложения, необходимо в совершенстве владеть приемами работы в IDE Visual Studio 2010. Среда разработки Visual Studio 2010 была спроектирована с целью максимально упростить процесс написания кода, его отладки и компиляции в сборку, предназначенную для поставки конечным потребителям. Работая в Visual Studio 2010, вы получаете в свое распоряжение тщательно продуманную среду, с помощью которой вы можете выполнять почти любые виды работ, связанных с созданием ваших приложений.



ГЛАВА 4

Отладка приложений

В среде Visual Studio 2010 предусмотрен мощный набор средств построения и отладки. Благодаря конфигурациям построения можно выбирать компоненты для построения, исключать компоненты, а также определять, как будут построены выбранные проекты и для какой платформы.

В этой главе будет представлен краткий обзор возможностей отладки приложений, предлагаемых средой Visual Studio 2010, с концентрацией внимания на тех аспектах, которые могут оказаться новыми для некоторых категорий разработчиков.

Основные типы ошибок

В процессе написания программного кода возможно появление трех видов ошибок:

- *синтаксические ошибки* (syntax errors) возникают, если компилятор не в состоянии понять переданный ему исходный текст;
- *логические ошибки* (logical errors) не препятствуют компиляции и исполнению программы, но приводят к неожиданным результатам;
- *ошибки периода выполнения* (run-time errors) проявляются при попытке выполнить недопустимое действие во время исполнения программы.

Синтаксические ошибки можно обнаружить сразу. Код программы при наличии синтаксических ошибок не будет скомпилирован. После завершения построения приложения можно использовать отладчик для обнаружения и устранения таких проблем, как логические и семантические ошибки, обнаруженные во время выполнения. Рассмотрим процесс обнаружения и устранения этих ошибок в среде Visual Studio 2010 более подробно.

Синтаксические ошибки

Синтаксические ошибки возникают, если компилятор не может скомпилировать предоставленный ему код, например, из-за опечатки в ключевом слове, пропуска

знака препинания или использования неверной конструкции. Любая из этих ошибок не позволит компилятору интерпретировать такой код.

Среда разработки Visual Studio 2010 облегчает поиск синтаксических ошибок, автоматически обнаруживая их при компоновке проекта и выделяя в исходном тексте подсветкой. Список обнаруженных ошибок также выводится в окне **Error List** (рис. 4.1).

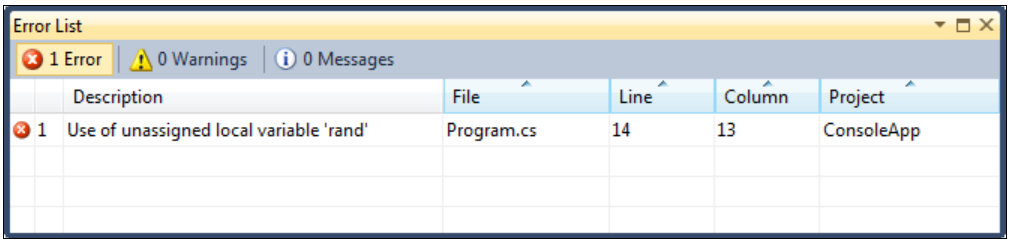


Рис. 4.1. Окно **Error List**

Если дважды щелкнуть левой кнопкой мыши описание ошибки в окне **Error List**, среда разработки переместит курсор в соответствующую строку в окне кода и выделит ошибку подсветкой — обычно этого достаточно, чтобы ее распознать и исправить. Программисту остается только исправить найденную ошибку. Если для исправления ошибки требуется дополнительная информация, можно воспользоваться встроенной или онлайновой документацией MSDN или другими ресурсами сообщества разработчиков.

Логические ошибки

Логическую ошибку констатируют, если приложение корректно компилируется и исполняется, но не выдает ожидаемые результаты. Диагностировать такие ошибки труднее, чем остальные, из-за отсутствия указаний на источник ошибки. Причиной логической ошибки могут быть такие, на первый взгляд, незначительные промахи, как неверно поставленная точка в десятичной дроби или лишняя итерация в операторе цикла.

Чтобы обнаружить логическую ошибку, необходимо разработать и предоставить приложению тестовые данные, а затем проанализировать результаты выполнения программы. Часто для нахождения логической ошибки приходится анализировать строку за строкой написанный код. Для этого применяется режим пошагового исполнения кода, который будет рассмотрен далее в этой главе.

Ошибки периода выполнения

Ошибки периода выполнения возникают при попытке приложения выполнить недопустимую операцию. К этой категории относятся деление на ноль и действия, запрещенные политиками безопасности: попытка их исполнения приводит к генерации исключения защиты.

При возникновении ошибки периода выполнения генерируется *исключение* (exception), предоставляющее описание ошибки. Это специальный класс, который служит для передачи сведений об ошибке другим компонентам приложения.

Чтобы познакомиться с инструментами отладки и их настройкой, напишем простую консольную программу, которая генерирует массив из десяти случайных чисел и выводит числа в окно консоли. В программе есть ошибка — выход в цикле `for` за пределы массива. Код программы приведен в листинге 4.1.

Листинг 4.1. Пример программы с ошибкой

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

namespace ConsoleApp
{
   class Program
   {
      static void Main(string[] args)
      {
         int[] numbers = new int[10];
         Random rand = new Random();

         for (int i = 0; i <= 10; i++)
         {
            numbers[i] = rand.Next();
            Console.WriteLine("{0}\t{1}", i, numbers[i]);
         }
      }
   }
}
```

Компиляция кода программы проходит успешно, но при выполнении приложения отладчик Visual Studio реагирует на ошибку выходом в код и показывает строку с ошибкой. На рис. 4.2 показан вид редактора Visual Studio при генерации исключения во время выполнения программы.

При появлении ошибки в редакторе кода выделяется строка, на которой была сгенерирована ошибка. Окно в правой верхней части изображения — это помощник **Exception Assistant**. Он предоставляет подробности по исключению и предлагает советы по поиску ошибки и устранению проблемы. Из этого окна можно получить доступ к поиску в интерактивной системе помощи дополнительной информации по этому исключению.

В нижней части главного окна среды разработки находится еще несколько дополнительных полезных окон (рис. 4.2). Слева расположено окно **Locals**. Это окно

отображает значения локальных переменных в коде программы в момент генерации исключения.

В правой нижней части главного окна среды разработки находится окно стека вызовов **Call Stack**. Это окно показывает последовательность, в которой различные компоненты приложения были вызваны. Окно **Call Stack** можно использовать для перехода к участкам кода, отображающимся в стеке.

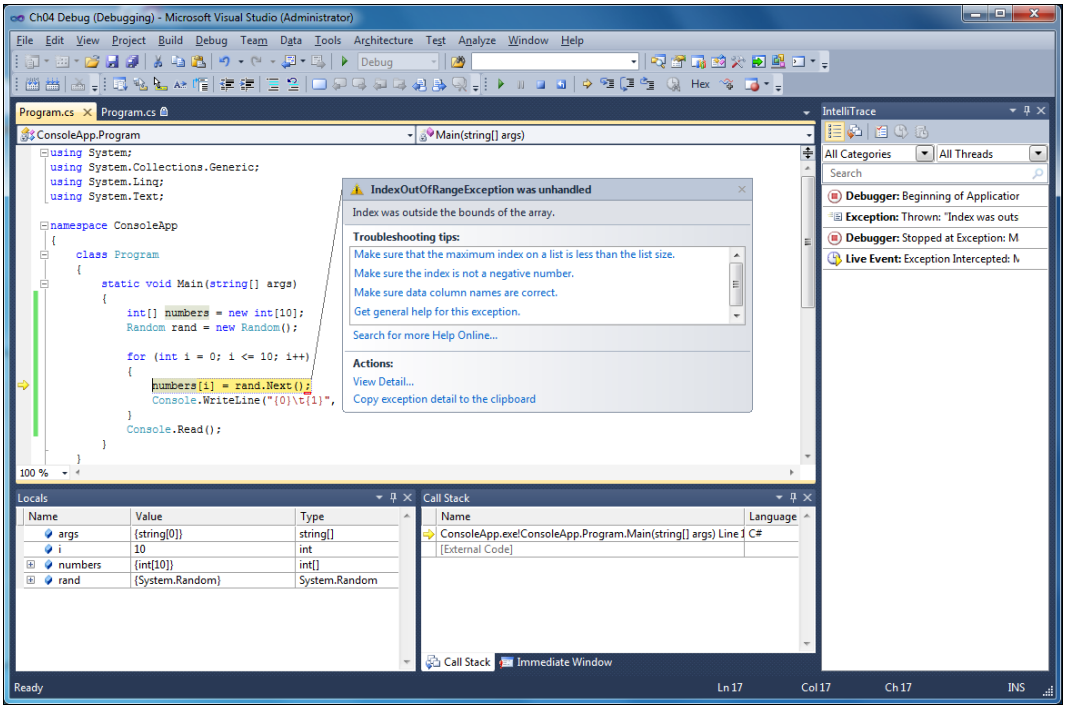


Рис. 4.2. Генерация исключения при выполнении программы

Неактивная закладка рядом с окном **Call Stack** дает вам доступ к окну интерпретаций **Immediate Window**, которое позволяет вводить команды кода и получать результаты в редакторе. Далее в этой главе мы подробнее рассмотрим эти окна и работу с ними.

Отладчик Visual Studio

Отладчик, встроенный в Visual Studio 2010, является одним из самых больших и сложных инструментов интегрированной среды разработки. Учитывая обширную функциональность, мы не сможем описать все возможные сценарии, которые могут встретиться при разработке приложений. Однако мы надеемся описать в данном разделе наиболее часто используемые функциональные возможности.

Меню **Debug** и соответствующая панель инструментов предоставляют доступ к запуску сеансов отладки, пошаговому прохождению кода, управлению точками

прерывания, а также и ко многим функциональным возможностям отладки в Visual Studio 2010.

В неактивном режиме меню **Debug** предоставляет возможности для запуска сеанса отладки, прикрепления к выполняющемуся процессу и для доступа к некоторым из отладочных окон.

При запуске приложения в режиме отладки в меню **Debug** добавляется несколько дополнительных опций. Эти опции включают функции перемещения по коду, перезапуск сеанса и доступ к дополнительным окнам отладки.

С помощью панели инструментов **Debug** вы можете управлять сеансом отладки. Например, можно начать или продолжить сеанс отладки, остановить выполняющийся сеанс, выполнить пошаговый проход кода и т. д.

Вы можете управлять множеством опций отладки Visual Studio в диалоговом окне **Options**. Узел **Debugging** в дереве опций дает доступ к этим переключателям отладочных опций. На рис. 4.3 показаны общие настройки отладки в диалоге **Options**. Это окно доступно в главном меню в разделе **Tools.**

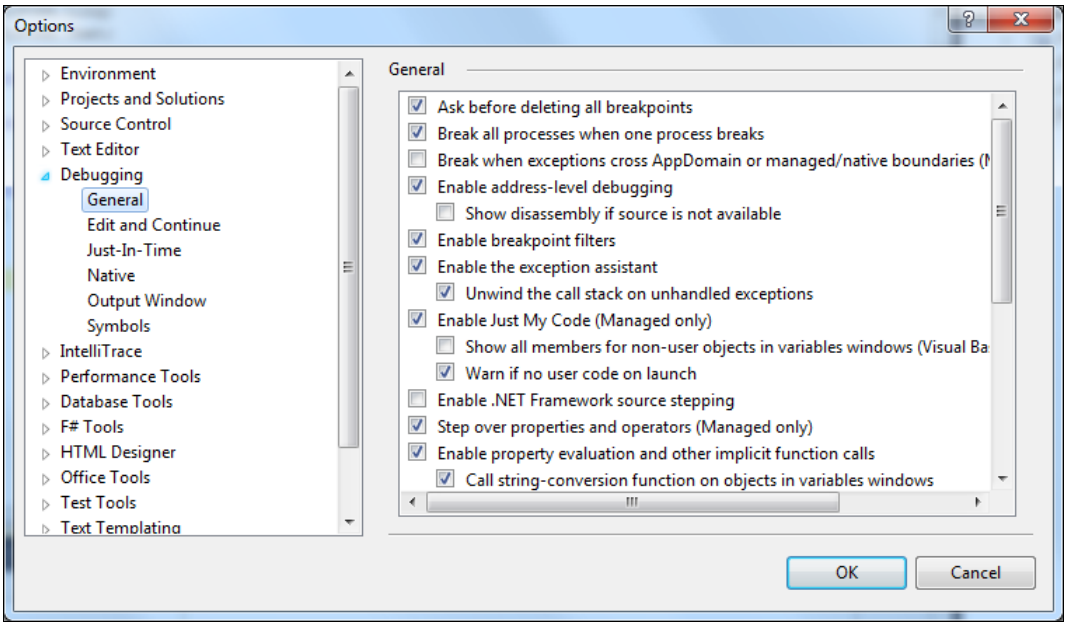


Рис. 4.3. Диалоговое окно **Options**

Узел **Debugging** дает доступ к включению и выключению многих опций отладки, например установки фильтров точек прерывания, выводу окна предупреждающего сообщения и др.

Точки прерывания

Можно назначать определенные строки кода или задавать условия, при исполнении которых отладчик непременно остановит выполнение приложения. Такие

процедуры называются *установкой точек прерывания* (breakpoints), они позволяют останавливать исполнение программы в заданном месте или при определенных обстоятельствах. Точки прерывания задаются несколькими способами:

- *по функции* — останавливают исполнение по достижении определенной строки функции;
- *по адресу в файле* — останавливают исполнение приложения по достижении определенного места в файле исходного текста;
- *по адресу памяти* — останавливают приложение при обращении к определенному адресу памяти.

Visual Studio 2010 имеет для точек прерывания несколько значков. Внешний вид значков отображает различные типы точек прерывания. Например, сплошной кружок — это обычная точка прерывания, а пустой кружок представляет деактивированную точку прерывания.

Настройка точки прерывания

Точки прерывания по функции применяются чаще всего, их устанавливают одним из трех способов:

1. Щелкнув серую область в левой части окна кода напротив нужной строки — в результате точка прерывания устанавливается в этой строке.

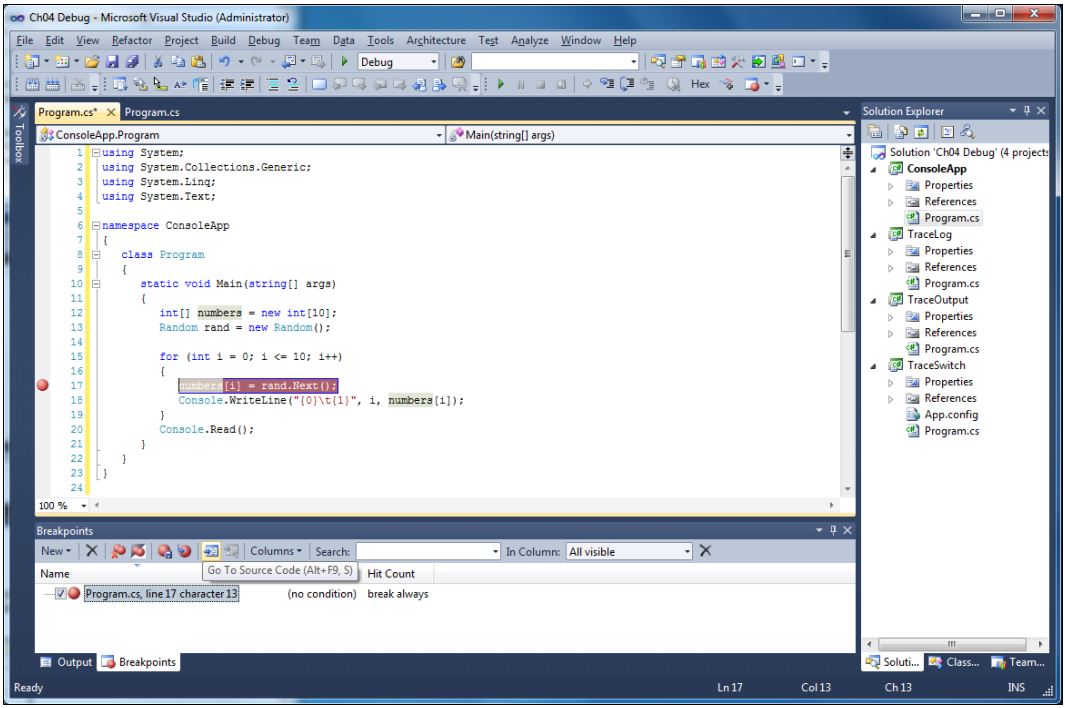


Рис. 4.4. Настройка точки прерывания

2. Щелкнув правой кнопкой нужную строку и выбрав из контекстного меню команду **Insert Breakpoint** (Вставить точку прерывания).
3. Выбрав команду **New Breakpoint** (Новая точка прерывания) из меню **Debug** или из контекстного меню редактора кода и установив соответствующие параметры точки прерывания в окне **New Breakpoint**.

Самый часто используемый способ настройки точки прерывания: сначала надо найти строку кода, на которой вы хотите остановить отладчик, затем вы щелкаете по этой строке в поле индикаторов редактора кода. При этом в поле индикаторов появляется красный кружок, и строка кода выделяется красным цветом (рис. 4.4). Щелчок мышью на кружке удаляет точку прерывания.

Окно *Breakpoints*

Для централизованного управления всеми точками прерывания применяется окно **Breakpoints**. Его можно вызвать через меню **Debug | Windows | Breakpoints**. Внешний вид окна показан на рис. 4.5.

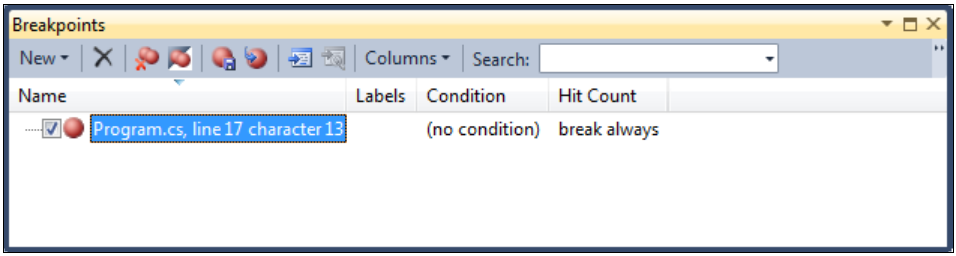


Рис. 4.5. Окно **Breakpoints**

В окне **Breakpoints** отображаются все точки прерывания, установленные в проекте с указанием их размещения и условий, определенных для каждой точки прерывания. В меню **Columns** выбирают для отображения дополнительные столбцы со сведениями о точках прерывания. В этом окне также можно деактивировать точку прерывания, сняв соответствующий флажок. Кроме того, кнопка в верхнем левом углу этого окна позволяет создать новые и удалить существующие точки прерывания, а также очистить список или отключить все точки прерывания.

Точку прерывания, ставшую ненужной, можно удалить либо отключить, если она потребуется в дальнейшем. Как уже говорилось, для управления точками прерывания предназначено окно **Breakpoints**. Вы также можете удалить или перевести в неактивное состояние точку прерывания в редакторе кода.

Для удаления точки прерывания достаточно щелкнуть ее правой кнопкой мыши на серой панели слева от редактора кода. Для временного отключения точки прерывания щелкните ее правой кнопкой мыши в редакторе кода и выберите **Disable Breakpoint**. Таким образом, она будет переведена в неактивное состояние.

Окно **Breakpoints** также дает вам доступ к каждой отдельной точке прерывания. Оно служит стартовой точкой для настройки многих опций, связанных с точками прерывания. Например, вы можете деактивировать отдельную точку прерывания,

сняв галочку около точки прерывания в списке точек прерывания. Кроме того, вы можете настроить множество свойств и условий, связанных с точкой прерывания. На рис. 4.6 показаны деактивированная точка отслеживания и контекстное меню, связанное с точкой прерывания.

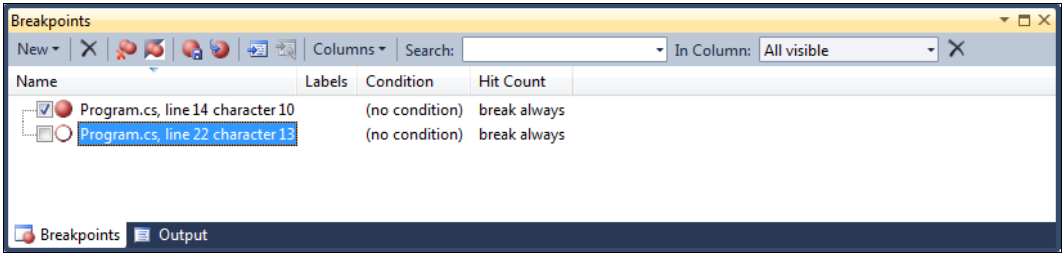


Рис. 4.6. Управление отдельной точкой прерывания

Обратите внимание, что из этого контекстного меню вы можете удалить точку прерывания, а также перейти к соответствующему ей исходному коду. Однако более важно то, что здесь есть доступ к настройке условий и фильтров, связанных с данной точкой прерывания. Далее мы опишем использование всех этих опций.

Настройка точки прерывания функции

Точка прерывания функции — это точка прерывания, которая настраивается в диалоговом окне **New Breakpoint**. Она называется точкой прерывания функции потому, что обычно устанавливается на начало функции (хотя это и не обязательно). В диалоговом окне **New Breakpoint** вы можете вручную настроить функцию, на которой вы хотите прервать выполнение, строку кода в функции или даже символ в строке.

Если при вызове этого диалогового окна ваш курсор находится на функции или на вызове функции, то имя функции будет автоматически внесено в диалоговое окно. Вы можете также ввести название функции в этом окне.

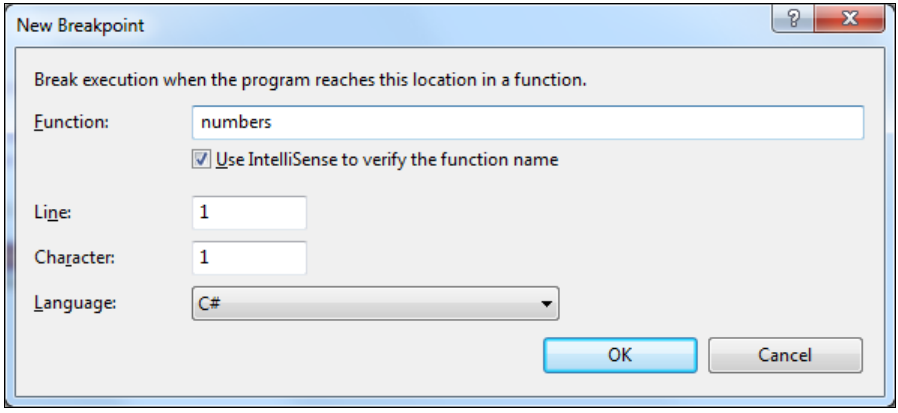


Рис. 4.7. Окно **New Breakpoint**

При установке новой точки прерывания через окно **New Breakpoint** можно задать условия, определяющие, будет ли остановлено исполнение приложения по достижении этой точки прерывания. Если щелкнуть кнопку **Condition** (Условие), откроется окно **Breakpoint Condition**, где предлагается ввести некоторое выражение (рис. 4.7). В результате точка прерывания активизируется, если при вычислении выражения получится `true`.

Прерывание на основе условий

Часто одной установки простой точки прерывания недостаточно (либо она неэффективна). Например, если вы ищете выполнение в вашем коде определенного условия (которое, возможно, вызывает исключительное состояние), то вам лучше делать прерывания по этому условию.

Это сэкономит время на постоянные входы в функции, при которых вы только изучаете несколько элементов данных и видите, что ваше условие не выполнено. Имеется несколько типов условий, которые вы можете добавить к точке прерывания:

- **Location**;
- **Condition**;
- **Hit Count**;
- **Filter**;
- **When Hit**.

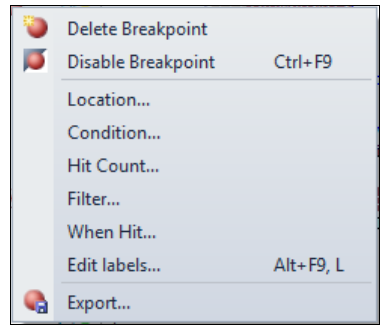


Рис. 4.8. Контекстное меню для точки прерывания

Вы добавляете условие к точке прерывания в окне **Breakpoints**. Выберите точку прерывания и нажмите правую кнопку мыши. Это действие активирует контекстное меню для данной точки прерывания (рис. 4.8), из которого можно выбрать тип условия прерывания.

Все типы условий для точек прерывания устанавливаются при помощи специализированных окон, которые мы подробно рассмотрим далее.

Окно *File Breakpoint*

Вы можете отредактировать точку прерывания так, чтобы прерывание происходило в определенном месте файла. Большинство точек прерывания работает именно так. То есть они знают файл, номер строки и символ, на котором должно произойти прерывание выполнения кода программы.

На рис. 4.9 показан пример окна **File Breakpoint**, открытого через опцию **Location** контекстного меню точки прерывания. Вы можете также использовать эту функциональную возможность для быстрой установки точки прерывания в конкретной строке без необходимости поиска в коде программы.

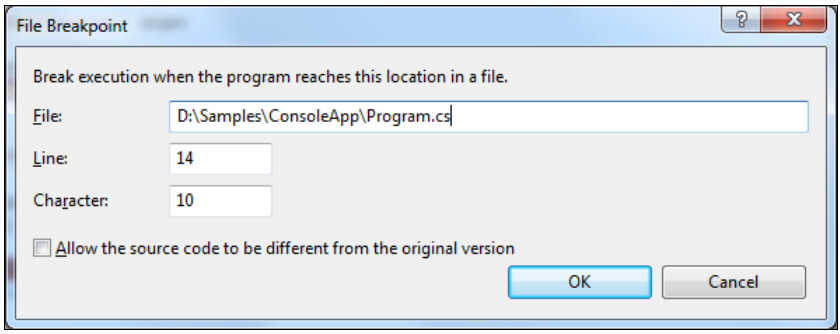


Рис. 4.9. Диалоговое окно **File Breakpoint**

Окно *Breakpoint Condition*

Условие точки прерывания позволяет при выполнении программы выйти в отладчик в случае, когда должно выполнится некоторое условие. Например, если известно, что ошибка в программе появляется только при определенных значениях переменной. Настройка условия точек прерывания — это удобное средство для поиска таких ошибок.

Для настройки условия выделите точку прерывания, для которой вы хотите добавить условие. Затем выберите опцию **Condition** из контекстного меню (через щелчок правой кнопкой мыши). Это активирует диалоговое окно **Breakpoint Condition** (рис. 4.10). При настройке условия у вас есть два варианта: **Is true** и **Has changed**. Вариант **Is true** позволяет вам настроить логическое условие, при выполнении которого произойдет выход из отладчика на соответствующую строку кода.

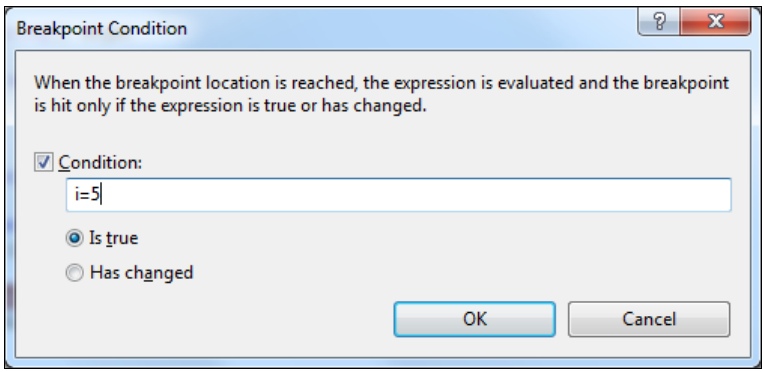


Рис. 4.10. Диалоговое окно **Breakpoint Condition**

Например, в случае с нашим приложением из листинга 4.1 можно добавить `i=5` в условие **Is true** для точки прерывания (где `i` — переменная нашего цикла `for`). В результате отладчику будет дано указание останавливаться на этой строке кода только тогда, когда будет выполнено условие `i=5`. На рис. 4.10 показано это условие в диалоговом окне. Там же имеются и две опции для условий.

Опция **Has changed** означает, что выходить из кода нужно только при изменении значения выражения. Первый проход по вашему коду устанавливает первое

значение выражения. Если после этого данное значение изменяется, то отладчик делает прерывание в данной строке. Эта возможность может быть полезной тогда, когда у вас есть поля или свойства с начальными значениями и вы хотите отследить изменения этих значений. Кроме того, опция **Has changed** может быть полезна в циклах и операторах `if...then`, когда вас интересует только одно — изменил ли ваш код данное конкретное значение.

Информация о точках прерывания между сеансами отладки сохраняется. То есть когда в конце рабочего дня вы закрываете Visual Studio, то по вашему возвращению все точки прерывания окажутся на своих местах. Это окупает время, потраченное на настройку сложных режимов отладки. Они могут оставаться в вашем приложении и включаться (выключаться) по мере необходимости.

Окно *Breakpoint Hit Count*

Используя команду **Hit Count**, вы сообщаете отладчику, что хотите прервать выполнение тогда, когда данная строка кода выполнится определенное количество раз. Обычно можно найти более удобное условие прерывания, чем **Hit Count**. Однако эта функция полезна в тех случаях, когда вы не можете указать реальное условие, но знаете, что когда вы проходите через функцию определенное количество раз, то начинаются проблемы. Кроме того, опция **Hit Count** может быть более полезной в сценариях с точками отслеживания, когда вы выдаете данные о том, что происходит в вашем коде. Возможно, вам будет удобно выдавать эти данные только время от времени.

На рис. 4.11 показано диалоговое окно **Breakpoint Hit Count**. Обратите внимание, что этот моментальный снимок экрана был сделан во время активного сеанса отладки. Вы можете добавить любые из этих условий к точкам прерывания во время активного сеанса отладки.

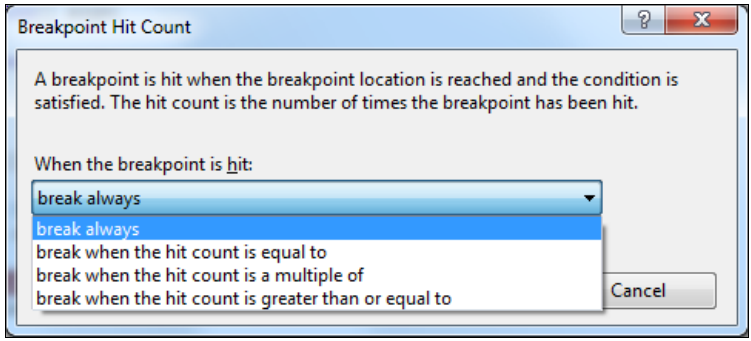


Рис. 4.11. Настройка количества прохождений точки прерывания

Это диалоговое окно предоставляет вам несколько опций для настройки количества попаданий. В раскрывающемся списке **When the breakpoint is hit** имеются следующие варианты:

- **break always** — прерывать всегда, не использует счетчик количества попаданий (по умолчанию);

- **break when the hit count is equal to** — прерывать тогда, когда значение счетчика равно указанному числу;
- **break when the hit count is a multiple of** — прерывать тогда, когда значение счетчика кратно указанному числу;
- **break when the hit count is greater than or equal to** — прерывать тогда, когда значение счетчика больше или равно указанному числу.

При необходимости можно совместно использовать все условия для точек прерывания в одной точке прерывания. Например, добавить к данной точке прерывания условие и фильтр, что дает возможность создавать при помощи точек прерывания специальные сценарии для отладки приложения.

Окно *Breakpoint Filter*

Среда разработки предоставляет диалоговое окно **Breakpoint Filter** (Фильтр точек прерывания). Фильтры точек прерывания позволяют вам указать конкретный компьютер, процесс или поток, в котором должна произойти остановка выполнения программы. Например, если ошибка появляется только на определенном компьютере, в определенном процессе или потоке, т. е. возможность настроить такое прерывание при помощи фильтра.

Для настройки фильтров точек прерывания можно задать имя целевого компьютера или для процесса задать его имя или идентификатор. Имеется возможность создавать комбинации условий при помощи операций `&` (и), `||` (или) и `!` (не). Тонкая настройка фильтров точек прерывания позволяет установить прерывание в определенном потоке конкретного процесса на конкретном компьютере. На рис. 4.12

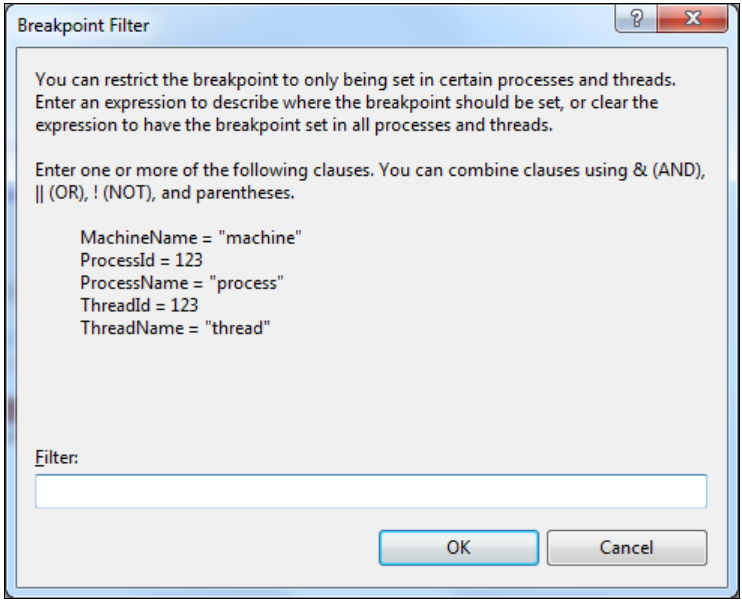


Рис. 4.12. Окно **Breakpoint Filter**

показано диалоговое окно **Breakpoint Filter**, в котором можно настроить необходимые фильтры точек прерывания.

Окно *When Breakpoint Is Hit*

Точки отслеживания позволяют выдать данные в окно **Output** (это окно будет описано далее в этой главе), когда встретилась определенная точка прерывания. После этого вы можете выйти в отладчик, как и в случае с обычной точкой прерывания, или продолжить выполнение приложения. Эта возможность может быть очень полезной тогда, когда вы хотите поддерживать регистрацию событий, происходящих в приложении при отладке. Затем вы можете просмотреть журнал событий, чтобы получить информацию о конкретных условиях и порядке выполнения (когда происходит исключение).

Вы можете настроить точки отслеживания явным образом — при помощи щелчка правой кнопкой мыши по строке кода и последующего выбора пункта **Insert Tracepoint** в меню **Breakpoint**. Кроме того, выбор команды **When Hit** из контекстного меню точки прерывания в окне **Breakpoints** активирует диалоговое окно точки отслеживания **When Breakpoint Is Hit** (рис. 4.13).

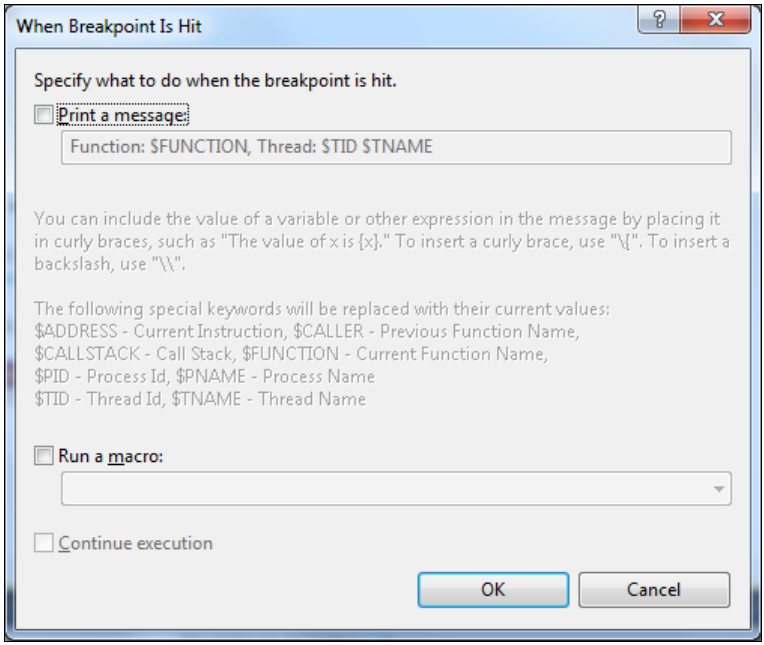


Рис. 4.13. Окно **When Breakpoint Is Hit**

Имеющиеся в диалоговом окне **When Breakpoint Is Hit** опции включают выдачу сообщения в окно вывода, выполнение макроса и продолжение выполнения. Вы можете выбрать любую комбинацию этих опций. Первая (выдача сообщения) позволяет вам выдать данные о вашей функции.

Есть несколько ключевых слов, которые допускается использовать для выдачи данных, например:

- `$function` — для имени функции;
- `$caller` — для имени вызывающей функции.

Полный список ключевых слов отображается в поясняющем тексте окна **When Breakpoint Is Hit**. Вы можете также вывести значения ваших переменных, заключив имена переменных в фигурные скобки.

Флажок **Continue execution** (Продолжать выполнение) позволяет вам указать, действительно ли это настоящая точка отслеживания или это точка прерывания, которая содержит отслеживающее действие. Если вы выбираете продолжение выполнения, то получаете только отслеживающее действие (сообщение и/или макрос). Если же вы даете указание, что продолжать выполнение программы не нужно, то получаете отслеживающее действие плюс остановку отладчика на данной строке кода точно так же, как и на обычной точке прерывания. При этом по существу применяется действие **When Hit** к обычной точке прерывания.

Если включить флажок **Run a macro** (Выполнить макрос), то диалоговое окно отобразит список всех загруженных в среду разработки макросов, из которого можно выбрать нужный макрос.

Также есть возможность объединять действия отслеживания с условиями. При этом действие происходит только в том случае, когда выполняется условие точки прерывания.

Пошаговое прохождение кода

При отладке программы чаще всего используется пошаговый проход по строкам кода и изучение данных, выдаваемых приложением и отладчиком. Пошаговое прохождение заключается в изучении строки кода, выполнении этой строки и анализе результатов выполнения. Далее этот процесс повторяется для последующих строк программного кода.

Начало отладки приложения

Команда входа в пошаговое прохождение кода **Step Into** доступна в меню **Debug** и на панели инструментов (эта команда также выполняется при нажатии клавиши <F11>). Если выполнить команду **Step Into** для приложения, которое в данный момент не выполняется в режиме отладки, приложение будет откомпилировано и запущено, и в окне отладки вы получите первую строку для пошагового прохождения кода. Это и есть вход в код вашего приложения. На рис. 4.14 показано приложение в режиме отладки при вызове команды **Step Into**.

Вызов команды **Step Over** из меню **Debug** в тот момент, когда ваше приложение находится в состоянии покоя, приведет к тому же самому, что и вызов **Step Into.** То есть приложение будет откомпилировано и запущено в сеансе отладки на первую строку кода.

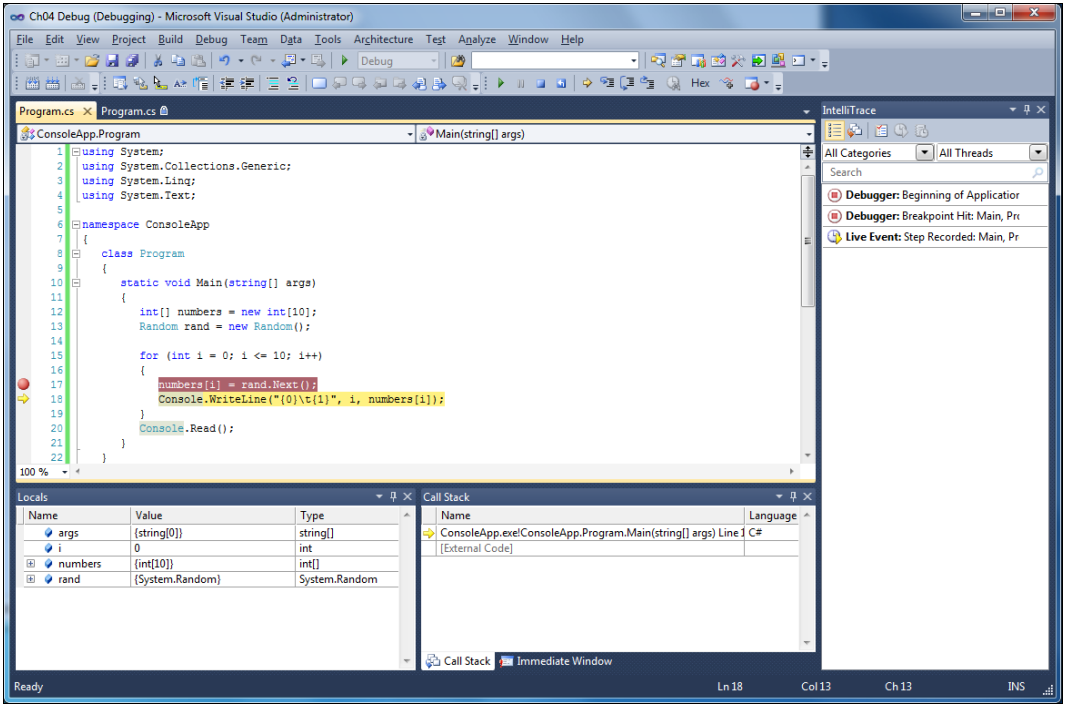


Рис. 4.14. Использование **Step Into** для запуска приложения

Одной из более удобных функциональных возможностей набора инструментов отладки является функция **Run to Cursor** (Выполнить до курсора). Если установить курсор на конкретную строку в коде программы и выполнить команду **Run to Cursor**, приложение скомпилируется и будет выполняться до тех пор, пока не дойдет до той строки кода, где был установлен курсор. В этой точке отладчик прерывает приложение и выделяет строку кода для выполнения пошагового прохождения по коду программы. Поэтому функция **Run to Cursor** является эффективным средством привести отладчик на указанную программистом область кода, которая требует отладки. Фактически это невидимая временная точка прерывания

Сеанс отладки приложения можно также запустить командой **Start Debugging** (Начало отладки). Это зеленая стрелка "воспроизведение" в меню **Debug** или на панели инструментов. Отладку приложения также можно запустить нажатием клавиши <F5>. При этом начнется сеанс отладки, но выход в код не произойдет, если только при выполнении программы не будет сгенерировано исключение или не попадется точка прерывания. Эта операция используется, если в программном коде применяется большое количество точек прерывания.

Если ваше приложение выполняется, и вы хотите войти в режим прерывания, то вы можете сделать это в любое время либо при помощи команды **Break All** (Прервать все) из меню **Debug** или панели инструментов, либо при помощи комбинации клавиш <Ctrl>+<Alt>+<Break>. Команда **Break All** представлена на панели инструментов значком с символом паузы. Нажатие этой кнопки останавливает ваше приложение на следующей выполняющейся строке и позволяет вам получить ин-

формацию из отладчика. Команда **Break All** особенно полезна в том случае, когда требуется прервать длительный процесс или цикл в коде программы.

Прохождение по коду

Во время сеанса отладки у вас есть три основных варианта продвижения по вашему коду. Вы можете войти в строку или функцию, пропустить данную функцию или выйти из функции. Давайте рассмотрим каждый из них.

Команда **Step Into** (Вход) — <F11> — позволяет вам продвигаться по вашему коду по одной строке. Вызов этой команды выполнит текущую строку кода и поместит ваш курсор на следующую выполняемую строку. Важное различие между **Step Into** и другими похожими командами состоит в том, как **Step Into** обрабатывает строки кода, в которых содержатся вызовы методов. Если вы находитесь на строке кода, которая вызывает другой метод вашего решения, то вызов **Step Into** перенесет вас на первую строку этого метода (при условии, что у вас загружены соответствующие отладочные символы).

Команда **Step Over** (Пропуск) — <F10> — позволяет вам сохранять фокус в текущей процедуре (не заходя в вызываемые ею методы). То есть вызов **Step Over** приведет к выполнению строки за строкой, но не заведет вас в вызовы функций, конструкторы или вызовы свойств. Конечно, любое исключение в пропущенной функции приведет к прерыванию работы отладчика и выходу в код (как обычно).

Команда выхода из пошагового режима **Step Out** (Выход), вызываемая нажатием комбинации клавиш <Shift>+<F11>, — это еще один полезный инструмент. Он позволяет вам дать указание отладчику закончить выполнение текущего метода (который вы отлаживаете) и вернуться в режим прерывания выполнения программы сразу после его завершения. Это очень удобно тогда, когда требуется пропустить большой фрагмент кода данного метода, который написан без ошибок (или все ошибки устранены) и не требует пошагового выполнения. Кроме того, в случае необходимости можно войти в функцию для отладки только ее части, а затем выйти из нее.

Продолжение отладки

Если в сеансе отладки вы ушли с выполняющегося кода, то найти дорогу назад часто бывает очень трудно. Выполнявшаяся строка потерялась в одном из многих открытых окон с кодом. К счастью, для того чтобы вернуться назад, вы можете использовать кнопку **Show Next Statement** (значок с желтой стрелкой) на панели инструментов **Debug**. Это вернет вас на ту строку, которая выполнялась в момент остановки отладчика.

Когда программа находится в сеансе отладки, команда **Start Debugging** (или **Run**) изменяется на **Continue** (Продолжение). Команда **Continue** доступна тогда, когда вы приостановили выполнение на строке кода в отладчике. Эта команда дает возможность продолжить выполнение приложения без пошагового прохождения по строкам кода программы. При помощи команды **Continue** вы даете указание отлад-

чику продолжать выполнение программы до тех пор, пока не произойдет исключение или не сработает точка прерывания.

Окончание отладки

Вы можете закончить сеанс отладки несколькими способами. Один из самых часто используемых методов — это прекратить выполнение приложения. При завершении приложения произойдет также и завершение сеанса отладки.

Имеется также пара способов и в окне **Debug**. Команда **Terminate All** завершает все процессы, к которым прикреплен отладчик, и завершает сеанс отладки. Есть также опция **Detach All**, которая открепляет отладчик от всех выполняющихся процессов без их завершения. Опция **Detach All** требуется в случае временного прикрепления к выполняющемуся процессу, который должен остаться работающим и после завершения отладки.

Отладочные окна

Среда Visual Studio 2010 предоставляет множество окон, позволяющих отслеживать исполнение программы. Нахождение нужных данных поможет вам быстро обнаружить ошибки и быстро их исправить. Visual Studio старается дать вам данные там, где они нужны. Например, окно всплывающей подсказки **DataTips** показывает вам значения переменных прямо в редакторе кода. Есть еще много примеров того, как Visual Studio показывает вам отладочные данные тогда и там, где они вам нужны. В следующих разделах мы опишем эти случаи (и не только их). Одни из них, такие как окна **Output** и **Command**, доступны в режиме **Design**, а другие, например окно **Locals**, видны только при отладке.

Наиболее часто применяемые при отладке окна **Locals**, **Autos** и **Watch** позволяют отслеживать и редактировать значения переменных в коде программы, что удобно при тестировании процедур путем передачи им различных аргументов.

Окно *Output*

В окне **Output** отображаются все данные, которые приложение выводит в командной строке в ходе его компиляции и исполнения: операторы `Debug` и `Trace` (об этих операторах будет рассказано далее в этой главе), а также сообщения, которы-

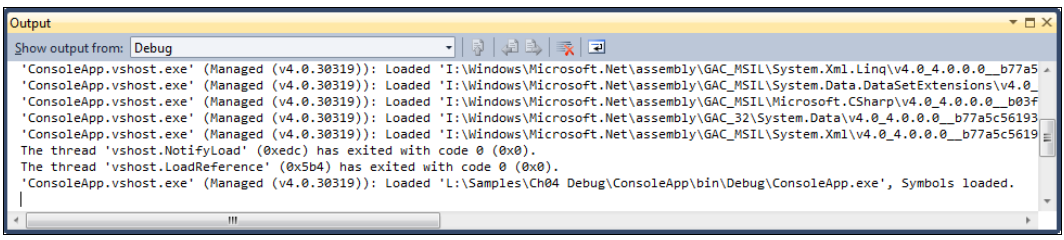


Рис. 4.15. Окно **Output**

ми приложение уведомляет о загрузке сборок. При отладке приложения в окне **Output**, как правило, просматривают уведомления оператора `Debug`. По умолчанию окно **Output** видно; в противном случае его можно вызвать, выбрав из меню команду **View | Other Windows | Output** (рис. 4.15).

Окно *Locals*

Во время выполнения программы в режиме отладки в окне локальных переменных **Locals** можно отслеживать значения всех переменных текущей процедуры. В нем отображаются три столбца: **Name** (Имя), **Value** (Значение) и **Type** (Тип).

Чтобы отобразить окно **Locals** во время отладки приложения, необходимо выбрать в меню **Debug** команду **Windows | Locals**.

Окно **Locals** показывает все переменные и их значения для текущей области видимости отладчика и, таким образом, можно получить информацию о том, что происходит в текущем методе. Переменные в окне **Locals** автоматически настраиваются отладчиком Visual Studio. Эти переменные организованы в список и отсортированы по имени в алфавитном порядке.

Сложные переменные, например экземпляры классов и структур, показаны в виде дерева с узлами, раскрывая которые можно получить доступ к значениям их членов.

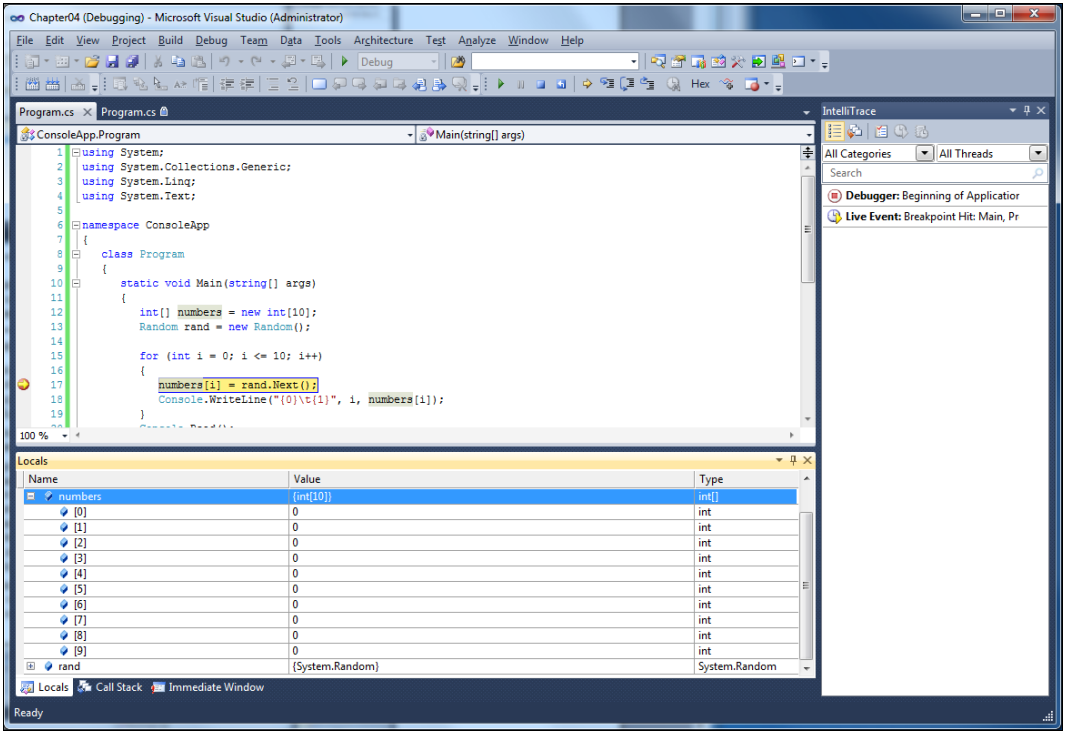


Рис. 4.16. Окно **Locals**

При передаче управления от метода к методу содержимое окна **Locals** меняется, отображая только локальные переменные.

В окне **Locals** также можно изменять значение переменной. Для этого следует выделить в столбце **Value** значение нужной переменной и ввести новое значение. Однако таким способом возможна корректировка только значений для простых типов, например строковых и числовых переменных. Модифицировать переменные-объекты, помещая в них ссылки на другие экземпляры класса или структуры, нельзя, зато можно изменять значения их членов. Измененные значения в столбце **Value** подсвечиваются красным цветом.

На рис. 4.16 показан пример окна **Locals**. В нем вы можете видеть приложение нашего примера, которое приостановлено внутри цикла `for`. По мере установки значений результаты отображаются в столбце **Value** окна **Locals**.

Окно *Autos*

Окно **Autos** напоминает окно **Locals**, но его возможности ограничены. В нем отображаются те же столбцы, что и в окне **Locals**, но в них содержатся только переменные из текущей и предыдущей строк кода. В окне **Autos** можно изменять значение переменных так же, как и в **Locals**.

Чтобы вызывать окно **Autos** во время отладки приложения, выберите из меню **Debug** команду **Windows | Autos**.

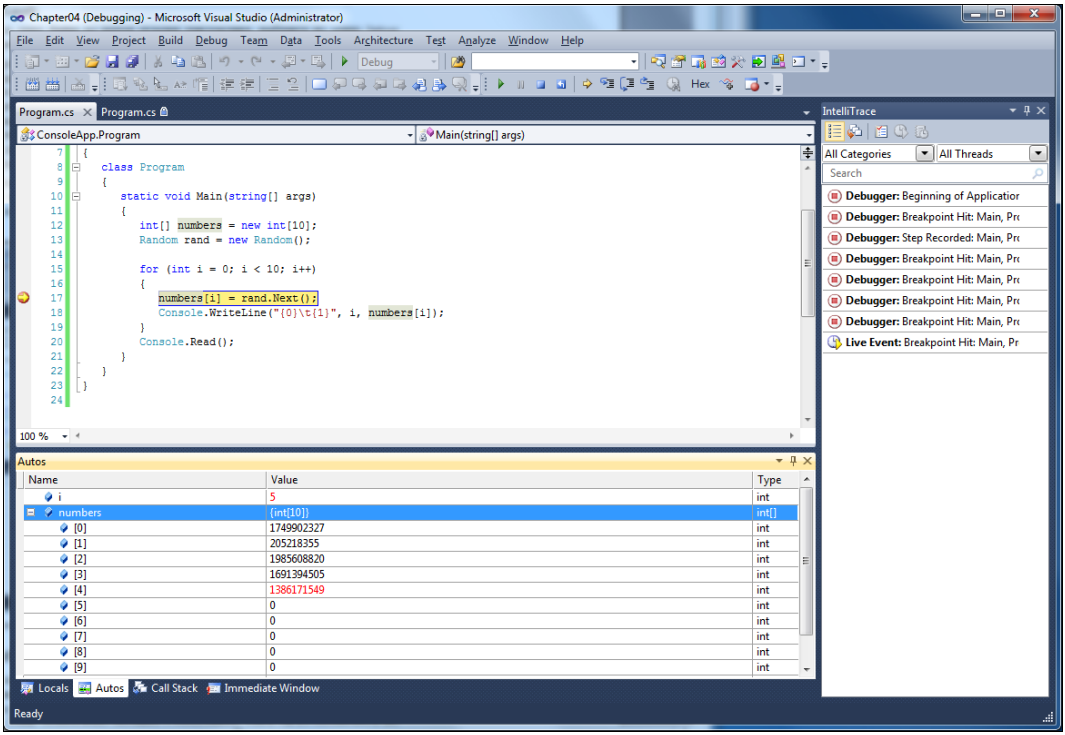


Рис. 4.17. Окно **Autos**

Использовать окно **Autos** удобно при отладке больших программ, когда отображение всех локальных переменных дает слишком много информации, чтобы в ней можно было разобраться. Это окно отображает значения всех переменных и выражений, имеющихся в текущей выполняющейся строке кода или в предыдущей строке кода. Это позволяет вам действительно сосредоточиться только на значениях, которые вы в данный момент отлаживаете.

На рис. 4.17 показано окно **Autos** для строки кода. Обратите внимание на разницу между окнами **Locals** и **Autos**, а также на то, что среда Visual Studio даже добавила в список наблюдения специфические выражения, которые отличаются от кода.

Окно *Watch*

Окна **Watch** предназначены для наблюдения за переменными. Доступ к окнам **Watch** можно получить из меню или панели инструментов **Debug**. Четыре окна **Watch** называют **Watch 1**, **Watch 2**, **Watch 3** и **Watch 4**. Эти окна **Watch** позволяют настроить четыре списка элементов, за которыми можно вести наблюдение во время выполнения программы. Использование отдельных списков переменных удобно в случае, когда каждый список относится к разным областям видимости в коде приложения.

В окно **Watch** можно добавить переменную, чтобы отслеживать, как изменяются ее значения. Переменные не удаляются из окна **Watch**, даже когда они уходят из

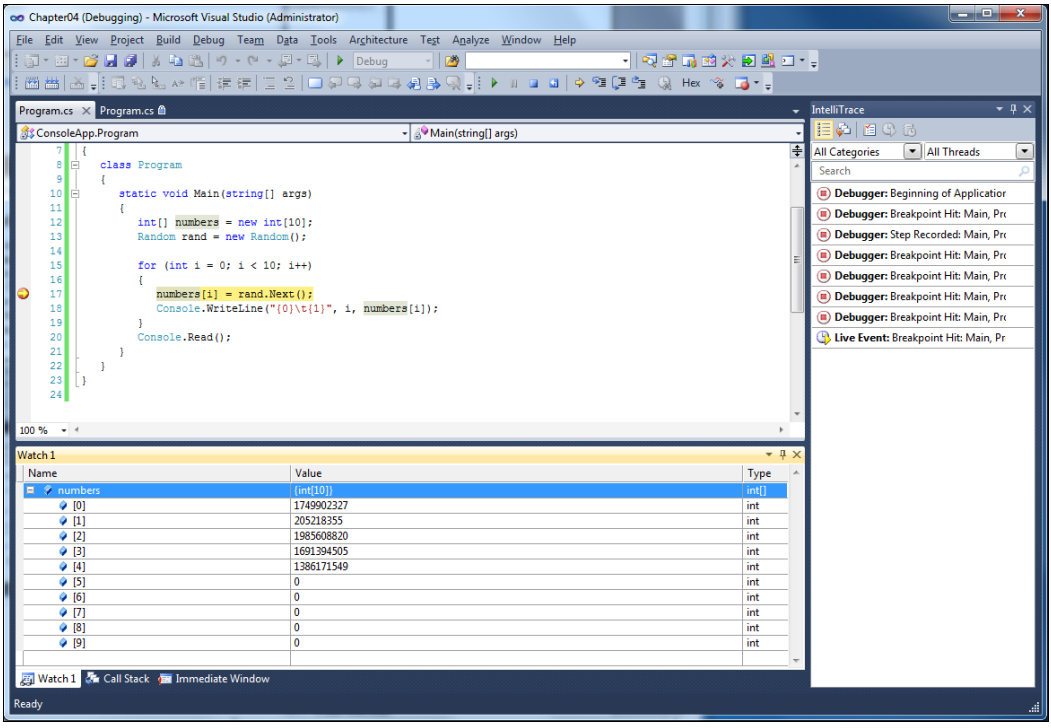


Рис. 4.18. Окно **Watch**

области видимости. Так же как и в окнах **Locals** и **Autos**, в окне **Watch** отображаются столбцы **Name** (Имя), **Value** (Значение) и **Type** (Тип), содержащие сведения о переменных; здесь также разрешается изменять текущие значения простых переменных путем ввода новых значений в столбец **Value**.

Добавить переменную в окно **Watch** можно одним из следующих способов:

- щелкнуть мышью пустую строку в окне **Watch** и ввести в нее имя переменной;
- в редакторе кода щелкнуть правой кнопкой мыши переменную и из контекстного меню выбрать команду **Add Watch**;
- щелкнуть переменную в редакторе кода, окнах **Autos** либо **Locals** и перетащить ее в окно **Watch**;
- перетащить выделенный элемент в окно **Watch** с помощью мыши.

Среда Visual Studio 2010 позволяет открывать несколько окон **Watch**, что дает возможность отслеживать одновременно несколько наборов переменных. Окно **Watch** представлено на рис. 4.18.

Окно *QuickWatch*

Диалоговое окно **QuickWatch** (рис. 4.19) предназначено для быстрого расчета значения переменной. В столбцах **Name** (Имя), **Value** (Значение) и **Type** (Тип) этого окна отображаются сведения только об одной переменной; из него переменную можно добавить в окно **Watch**, а также изменить ее значение, введя новое значение в столбец **Value**.

Для вызова диалогового окна **QuickWatch** необходимо в редакторе кода щелкнуть правой кнопкой мыши на переменной и выбрать из контекстного меню опцию **QuickWatch**.

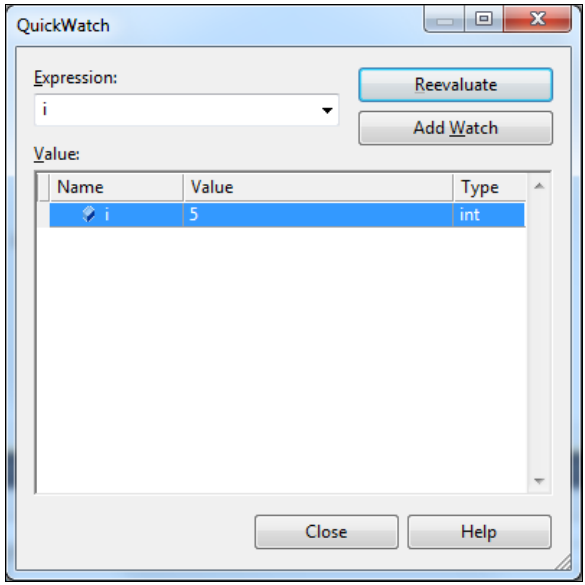


Рис. 4.19. Окно **QuickWatch**

Из окна **QuickWatch** можно также добавить переменную или выражение в окно **Watch**. Для этого в редакторе кода надо выделить требуемый объект, щелкнуть на нем правой кнопкой мыши и выбрать в контекстном меню пункт **Add Watch**. При этом выделенная переменная будет помещена в окно **Watch.**

В окне **QuickWatch** вы можете писать выражения и добавлять их в окно **Watch**. Добавленный в окно **QuickWatch** элемент будет вычислен при нажатии кнопки **Reevaluate**. Нажатие кнопки **Add Watch** отправит переменную в окно **Watch 1.**

Окно *Command* в режиме *Immediate*

Окно **Command** в режиме **Immediate** применяют при отладке для исполнения процедур, расчета значений выражений или изменения значений переменных. Чтобы открыть окно **Command** в режиме **Immediate**, щелкните команду **Debug | Windows | Immediate**.

Окно **Command** в режиме **Immediate** позволяет исполнить любой допустимый оператор, но не разрешает объявлять переменные и объекты. В режиме пошагового исполнения кода можно ввести в окно **Command** оператор или имя метода так же, как в редакторе кода. Нажав клавишу <Enter>, вы заставите Visual Studio исполнить введенный оператор. Завершив исполнение оператора или метода, приложение вернется в режим пошагового исполнения.

Когда приложение находится в режиме пошагового исполнения, значения выражений и переменных можно вводить в окно **Command** напрямую, предваряя их знаком вопроса. Например, таким образом перемножить текущие значения `x1` и `x2`, получив результат в окне **Command**:

```
?   x1*x2
```

Всплывающие подсказки данных *DataTips*

Всплывающие подсказки данных **DataTips** позволяют выделять переменную или выражение в коде программы и получить информацию о значении переменной прямо в окне редактора кода. Для получения всплывающей подсказки надо поместить указатель мыши над именем соответствующей переменной в окне редактора кода. При этом рядом с этой переменной появляется небольшое всплывающее окошко, в котором будет отображено значение данной переменной и которое, кроме того, может быть расширено для представления более подробной информации.

Всплывающие подсказки **DataTips** аналогичны действию окна **QuickWatch**. Однако гораздо проще навести курсор на необходимую переменную, и ее данные отобразятся в подсказке **DataTip.**

Пример всплывающей подсказки данных показан на рис. 4.20.

Щелчок правой кнопкой мыши по знаку "плюс" для этой переменной разворачивает множество членов этого объекта. Вы можете прокрутить этот список при помощи стрелки в нижней части окна. Вы можете также щелкнуть правой кнопкой мыши по любому члену списка и отредактировать его значение, скопировать его или добавить в окно наблюдения.

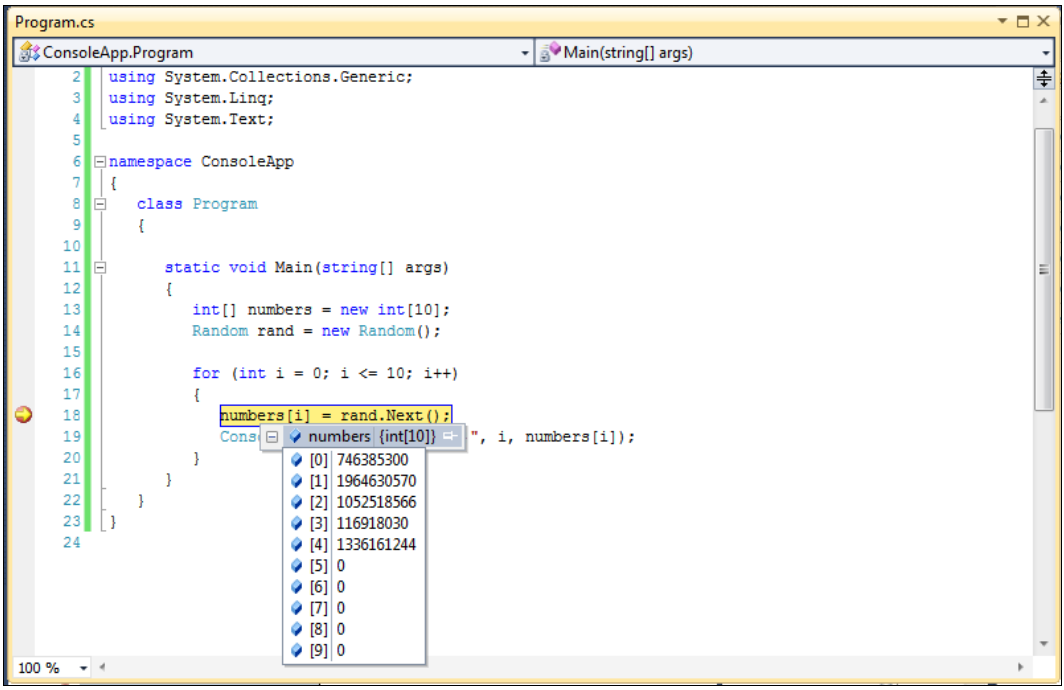


Рис. 4.20. Просмотр значения переменной во всплывающей подсказке

Окна визуализации данных

Visual Studio предлагает быстрый и простой способ доступа к данным внутри объекта с помощью *окон визуализации данных*. Эти окна предназначены для представления данных объекта определенным осмысленным образом.

Среда разработки Visual Studio 2010 предоставляет несколько типов окон визуализации данных:

- **HTML** — показывает диалоговое окно в виде браузера, где HTML интерпретирован так, как его будет видеть пользователь;
- **XML** — показывает XML в структурированном формате;
- **Text** — показывает строковое значение в легком для чтения формате;
- **DataSet** — показывает содержимое объектов `DataSet`, `DataView` и `DataTable`.

В следующих главах книги при создании и отладке приложений вам придется активно пользоваться этими окнами.

Исключения

Обработка исключений в коде гарантирует, что приложение сможет всегда успешно справляться с возможными проблемами, однако в процессе отладки кода

приложения, автоматическая обработка исключений, возникающих во время отладки, часто является нежелательной, особенно в тех случаях, когда обработка исключения предусматривает сброс данных и прекращение выполнения приложения. Однако при отладке приложения требуется, чтобы отладчик помог установить причину генерации исключения.

При возникновении исключения среда времени выполнения пытается найти соответствующий обработчик даже в том случае, если в программном коде он не предусмотрен. Когда выяснится, что такой обработчик отсутствует, выполнение вашей программы будет прекращено. Генерация исключения сопровождается очисткой стека вызовов, в результате становится невозможным просмотр значения переменных, поскольку все они покинут пределы области видимости.

Чтобы изменять поведение программы при генерации исключения, Visual Studio предоставляет окно **Exceptions**, которое можно открыть в меню **Debug | Exceptions** (рис. 4.23), которое позволяет указать, что должно происходить при возбуждении исключения. Доступны два возможных варианта выбора:

- продолжить выполнение программы;
- прекратить выполнение программы (в этом случае отладчик начинает самостоятельно выполнять оператор `throw`).

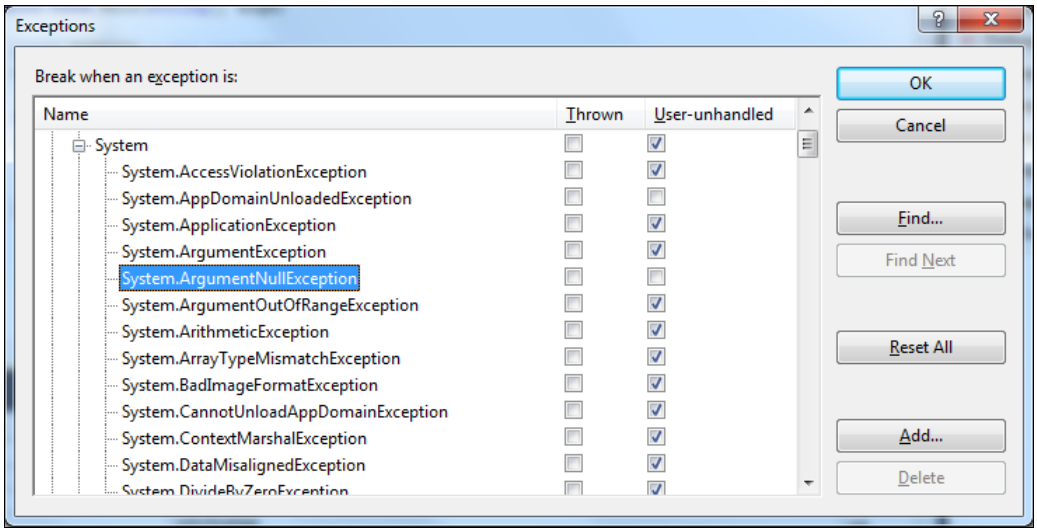


Рис. 4.21. Окно **Exceptions**

Среда Visual Studio не в состоянии автоматически отслеживать написанные вами пользовательские классы исключений, но вы можете вручную добавить их в список и тем самым указать, какие из них должны приводить к немедленному прекращению выполнения приложения. Для добавления пользовательских классов исключений надо щелкнуть по кнопке **Add** и ввести имя нужного класса исключений.

Классы *Debug* и *Trace*

Классы `Debug` и `Trace` позволяют генерировать и записывать в журналы сообщения со сведениями о состоянии приложения, не прерывая его исполнения. Класс `Debug` позволяет генерировать и записывать в журнал во время выполнения сообщения с описанием состояния программы, а класс `Trace` — создавать инструменты для диагностики даже для скомпилированных приложений.

Вывод трассировки в окно *Output*

Классы `Trace` и `Debug` содержат статические методы, которые позволяют проверять условия во время выполнения и записывать результаты в журнал. Данные, генерируемые ими, отображаются в окне **Output** (рис. 4.22), где их можно просматривать при отладке, и записываются в коллекцию `Listeners`. Коллекция `Listeners` содержит группу классов, получающих данные от `Trace` и `Debug`, их называют *слушателями*. Классы `Trace` и `Debug` используют общую коллекцию `Listeners`, предоставляя одни и те же данные всем слушателям из этого набора.

Разные виды слушателей записывают данные и текстовые файлы или журналы событий. После завершения исполнения программы трассировочные данные, записанные слушателями, анализируют, чтобы найти ошибки в приложении. Трассировка полезна и при настройке программ. Пример программы с использованием трассировки приведен в листинге 4.2.

Листинг 4.2. Программа с выводом трассировочных сообщений

```
using System;
using System.Diagnostics;
using System.IO;

namespace ConsoleApp
{
   class Program
   {
      static void Main(string[] args)
      {
         int[] numbers = new int[10];
         Random rand = new Random();

         for (int i = 0; i < 10; i++)
         {
            numbers[i] = rand.Next();
            Console.WriteLine("{0}\t{1}", i, numbers[i]);
            Trace.WriteLine(String.Format("{0}:\t{1}\t{2}",
               DateTime.Now, i, numbers[i]));
         }
         Trace.Flush();
```

```
            Console.Read();
        }
    }
}
```

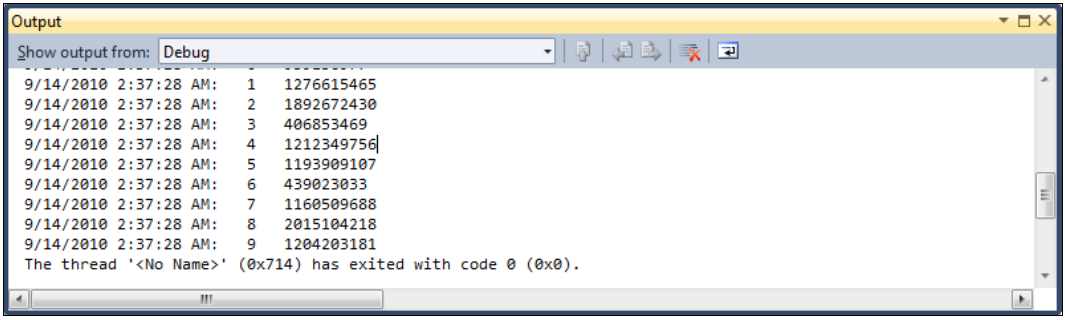


Рис. 4.22. Данные трассировки в окне **Output**

Классы `Trace` и `Debug` предоставляют одинаковые наборы методов и свойств и записывают результаты в коллекцию `Listeners`. Единственное различие между этими классами в том, что по умолчанию операторы `Debug` удаляются при компиляции приложения в Release-версию, а операторы `Trace` там остаются. Поэтому класс `Debug` обычно применяют для отладки на стадии разработки, а `Trace` — для тестирования и настройки после компиляции окончательной версии приложения.

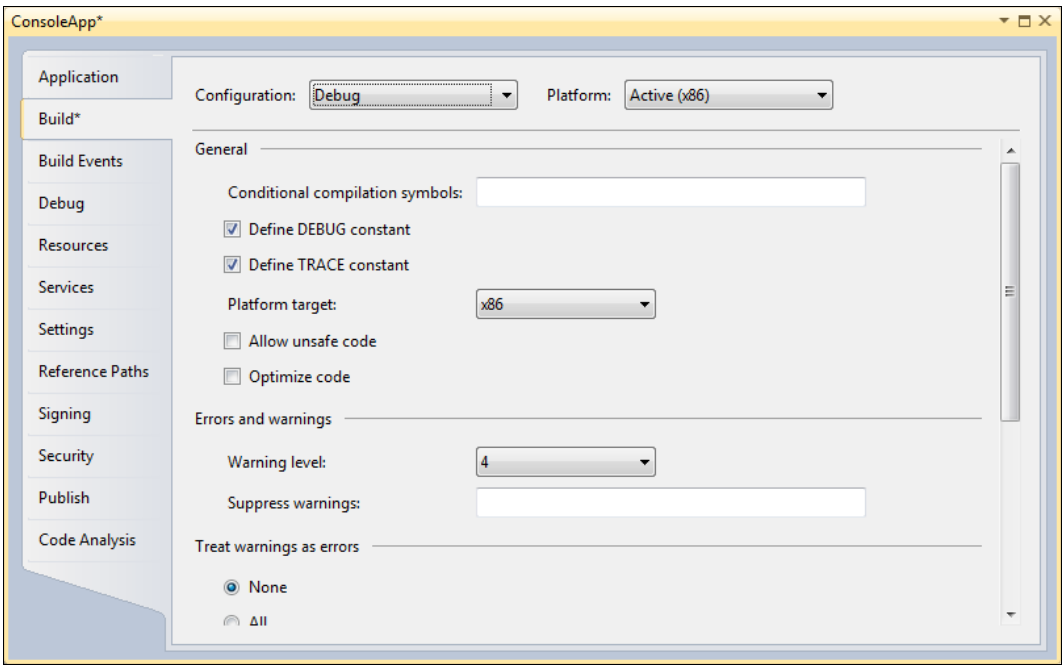


Рис. 4.23. Установка флагов `DEBUG` и `TRACE` в опциях компилятора

Компилятор игнорирует вызовы DEBUG и TRACE, если они не определены как символ условной компиляции. Флажки для определения DEBUG и TRACE расположены в окне свойств проекта на вкладке **Build** (рис. 4.23), их можно включать или отключать в зависимости от того, требуется ли вам производить запись трассировочной или отладочной информации.

Запись данных в набор *Listeners*

Все данные, сгенерированные классами Trace и Debug, направляются в набор Listeners. Это специальный набор, упорядочивающий классы, способные получать данные от Trace, и предоставляющий доступ к этим классам. Каждый член набора Listeners в полном объеме получает данные, генерированные классами Trace и Debug, способ обработки этих данных зависит от слушателя.

Данные, генерированные операторами Trace и Debug, передаются членам набора Trace.Listeners — объектам, способным получать трассировочные данные и обрабатывать их. Существует три вида трассировочных слушателей:

- DefaultTraceListener;
- TextWriterTraceListener;
- EventLogTraceListener.

По умолчанию набор Listeners инициализируется единственным членом, экземпляром класса DefaultTraceListener. Он создается автоматически и получает данные от Trace и Debug даже в отсутствие других слушателей, добавленных явно. Полученные данные DefaultTraceListener передает отладчику и отображает в окне **Output** среды разработки Visual Studio (как в прошлом разделе). Чтобы создать журнал трассировочных сообщений, с которым можно работать независимо от отладчика, необходимо добавить по крайней мере один объект Listener.

Для записи генерируемых классами Trace и Debug данных в набор Listeners служат методы этих классов:

- Write() — записывает строку в набор Listeners;
- WriteLine()— записывает в набор Listeners строку, завершая ее символом возврата каретки;
- WriteIf()— записывает строку в набор Listeners, если заданное булево выражение имеет значение true;
- WriteLineIf()— записывает в набор Listeners строку, завершенную символом возврата каретки, если заданное булево выражение имеет значение true;
- Assert()— записывает сообщение в набор Listeners, если заданное булево выражение имеет значение false, и открывает окно с текстом этого сообщения;
- Fail() — создает проверку, которая автоматически завершается неудачей без проверки условия. При этом в набор Listeners записывается сообщение и отображается окно с текстом этого сообщения.

Трассировочные сообщения можно форматировать, изменяя отступы в тексте, вызывая методы Indent() и Unindent() и устанавливая свойства IndentSize и

IndentLevel. Эти методы и свойства позволяют создать иерархию сообщений об ошибках. Свойство IndentSize возвращает или устанавливает число пробелов, составляющее один уровень отступа, а IndentLevel — уровень отступа для текущего сообщения. Метод Indent увеличивает, а Unindent — уменьшает значение IndentLevel на единицу.

Чтобы записать данные в набор Listeners, необходимо вызвать соответствующий метод:

- Write() или WriteLine() для безусловной записи;
- WriteIf() или WriteLineIf(), чтобы записать данные, если проверка условия дает true;
- Fail() для безусловной записи результата и отображения сообщения;
- Assert(), чтобы записать данные, если проверка условия дает true, и отобразить сообщение.

Данные, генерируемые классом TextWriterTraceListener, записываются в текстовый файл, в объект Stream либо в объект TextWriter. Объекты Stream и TextWriter также применяются для записи данных в текстовые файлы. Чтобы создать объект TextWriterTraceListener для записи данных, сгенерированных классом Trace, в текстовый файл, необходимо создать или открыть файл, куда будут записываться данные, после этого следует создать экземпляр TextWriterTraceListener, передав ему созданный открытый файл. В завершение следует добавить TextWriterTraceListener.Listeners.

Пример записи трассировочных событий в файл показан в листинге 4.3.

Листинг 4.3. Запись трассировочных событий в файл

```
using System;
using System.Diagnostics;
using System.IO;

namespace ConsoleApp
{
   class Program
   {
      static void Main(string[] args)
      {
         // Создайте экземпляр объекта FileStream,
         // указывающий на нужный текстовый файл
         FileStream traceLog = new FileStream(
            "C:\\TraceLog.txt", FileMode.OpenOrCreate);

         // Создайте экземпляр TextWriterTraceListener,
         // передав ему новый объект FileStream
         TextWriterTraceListener textListener =
            new TextWriterTraceListener(traceLog);
```

```
            // Добавьте созданный объект Listener к набору Listeners.
            Trace.Listeners.Add(textListener);

            int[] numbers = new int[10];
            Random rand = new Random();

            for (int i = 0; i < 10; i++)
            {
               numbers[i] = rand.Next();
               Console.WriteLine("{0}\t{1}", i, numbers[i]);
               textListener.WriteLine(String.Format(
                  "{0}:\t{1}\t{2}", DateTime.Now, i, numbers[i]));
            }

            // Сбросить буфер на диск
            Trace.Flush();
            traceLog.Close();
         }
      }
   }
```

Если открыть существующий файл с помощью `FileMode.OpenOrCreate`, содержимое файла будет перезаписано. Чтобы добавить новые даннные к существующему файлу, объявляйте объект `FileStream` с помощью `FileMode.Append`. Пример записанных в файл трассировочных данных представлен на рис. 4.24.

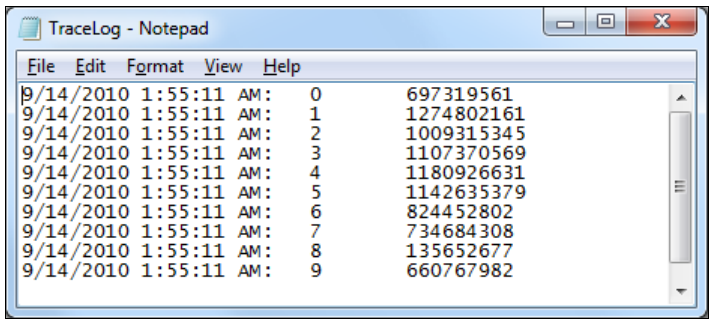


Рис. 4.24. Текстовый файл с данными трассировки

Для записи трассировочных данных в объект `EventLog` применяют объект `EventLogTraceListener`. В принципе, запись трассировочных данных в `EventLog` не отличается от записи в текстовый файл. Необходимо создать объект `EventLogTraceListener`, передав ему новый объект `EventLog` (его следует создать заранее) или ссылку на существующий объект журнала событий, и добавить его к набору `Listeners`. После этого результаты `Trace` будут записываться в указанный объект `EventLog` в виде объектов `EventLogEntry`.

Пример программы, использующей запись трассировочных данных в журнал событий, представлен в листинге 4.4.

Листинг 4.4. Запись трассировочных данных в журнал событий

```
using System;
using System.Diagnostics;
using System.IO;

namespace ConsoleApp
{
   class Program
   {
      static void Main(string[] args)
      {
         // Создаем экземпляр журнала событий EventLog
         EventLog log = new EventLog("ConsoleApp Debug Log");
         log.Source = "Trace Output";

         // Создаем объект EventLogTraceListener и передаем ему
         // объект EventLog
         EventLogTraceListener eventlogListener =
            new EventLogTraceListener(log);

         for (int i = 0; i < 10; i++)
         {
            numbers[i] = rand.Next();
            Console.WriteLine("{0}\t{1}", i, numbers[i]);
            eventlogListener.WriteLine(String.Format(
               "{0}\t{1}", i, numbers[i]));
         }
         Trace.Flush();

         Console.Read();
      }
   }
}
```

Если выполнить программу, а затем открыть **Event Viewer**, то в директории **Application and Services Logs** мы увидим новый журнал событий **ConsoleApp Debug Log** для нашего приложения, показанный на рис. 4.25.

Трассировочные переключатели

Данные, генерируемые операторами `Debug` и `Trace`, интенсивно используются при отладке, но после компиляции и выпуска окончательной версии приложения трассировку включают только в особых случаях.

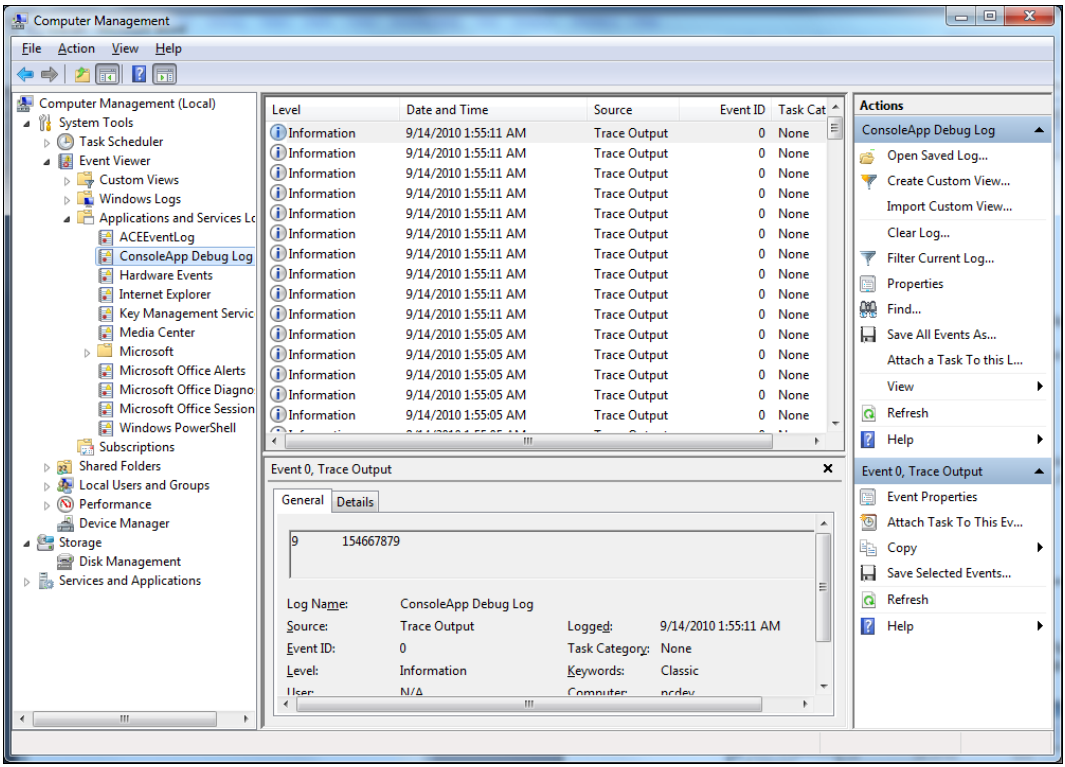


Рис. 4.25. Запись трассировочных данных в журнал событий

Библиотека базовых классов .NET Framework содержит два типа трассировочных переключателей. Первый, `BooleanSwitch`, возвращает значение типа `Boolean`, что делает его похожим на обычный переключатель, не имеющий промежуточных положений. Второй, `TraceSwitch` поддерживает параметр с пятью возможными значениями, позволяющий установить для переключателя нужный тип результата.

Включать и выключать отображение данных, генерируемых операторами `Trace`, можно средствами трассировочных переключателей, которыми управляют через конфигурационный файл приложения.

Трассировочные переключатели разрешено включать и отключать в компилированном приложении через его конфигурационный файл — файл с расширением config в формате XML, в котором хранится необходимая приложению информация. Конфигурационный файл располагается в одном каталоге с исполняемым файлом.

Конфигурационные файлы есть не у всех приложений. Если у вашего приложения, использующего трассировочные переключатели, такого файла нет, его необходимо создать.

При создании трассировочного переключателя необходимо предоставить параметр `DisplayName`, который определяет имя переключателя в конфигурационном файле. При редактировании конфигурационного файла следует указать имя переключателя и целочисленное значение, которое необходимо ему присвоить. Для

объектов типа `BooleanSwitch` значение 0 деактивирует переключатель, а любые значения, отличные от нуля, активируют его.

Класс `TraceSwitch` имеет свойство `TraceLevel`, значения которого представляют различные уровни ошибок. Это свойство способно принимать любое из пяти значений перечисления `TraceLevel`:

- `TraceLevel.Off` (0) деактивирует объект `TraceSwich`;
- `TraceLevel.Error` (1) определяет вывод кратких сообщений об ошибках;
- `TraceLevel.Warning` (2) позволяет выводить сообщения об ошибках и предупреждения;
- `Trace Level.Info` (3) разрешает выводить сообщения об ошибках, предупреждения и краткие информационные сообщения;
- `TraceLevel.Verbose` (4) позволяет выводить сообщения об ошибках, предупреждения и подробные сведения об исполнении программы.

Чтобы создать конфигурационный файл и настроить трассировочные переключатели, необходимо сначала создать в программе объекты переключателей с соответствующими значениями свойства `DisplayName`. Затем, если у вашего приложения нет файла конфигурации (с расширением config), необходимо его создать. Для этого выберите в меню **Project** опцию **Add New Item**, в открывшемся диалоговом окне **Add New Item** выберите **Application Configuration File** и назовите новый файл ConsoleApp.config, как показано на рис. 4.26.

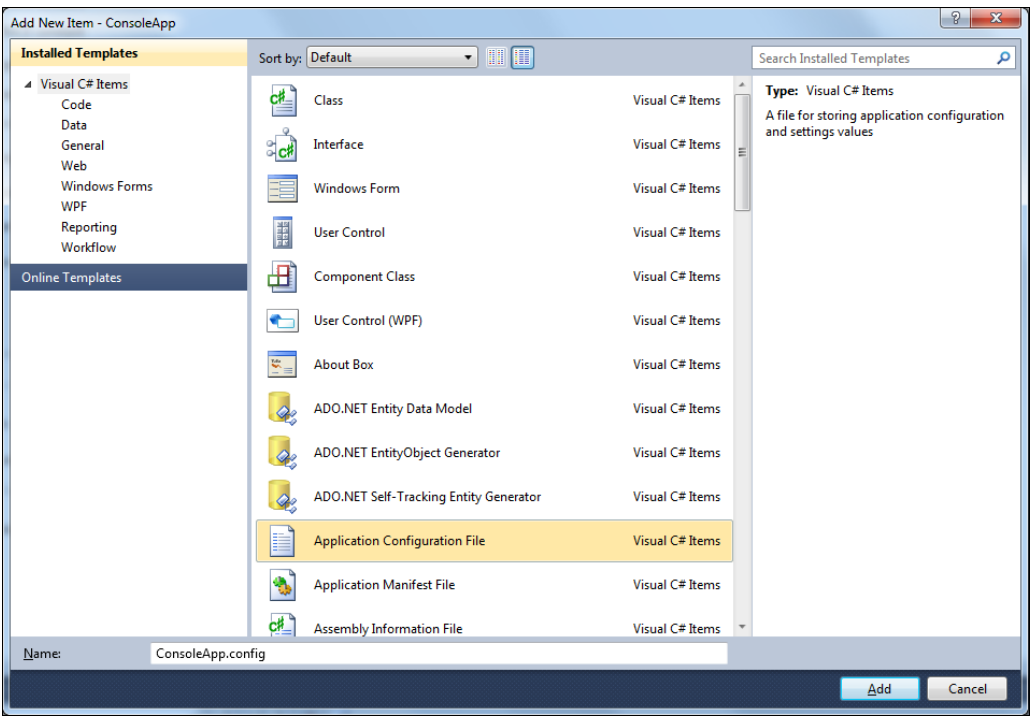


Рис. 4.26. Добавление файла конфигурации в приложение

Между тегами <configuration> и </configuration> добавьте XML-код, объявляющий трассировочные переключатели IsTraceOn (типа BooleanSwitch) и SwitchLevel (типа TraceSwitch) и присваивающий им значения, как показано в листинге 4.5.

Листинг 4.5. Трассировочные переключатели

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.diagnostics>
      <switches>
         <add name="IsTraceOn" value="1" />
         <add name="SwitchLevel" value="3" />
      </switches>
   </system.diagnostics>
</configuration>
```

В коде листинга 4.5 значение IsTraceOn установлено в true, а значение SwitchLevel — в TraceLevel.Info. Чтобы включить или выключить любой из переключателей, достаточно будет изменить его значение в конфигурационном файле, не перекомпилируя приложения.

Экземпляры классов BooleanSwitch и TraceSwitch создаются так же, как экземпляры любого другого класса. Конструкторы этих типов переключателей требуют два параметра: DisplayName, определяющий имя переключателя, отображаемое в пользовательском интерфейсе, и Description, который задает краткое описание переключателя, например:

```
BooleanSwitch bSwitch = new BooleanSwitch("IsTraceOn", "Tracing");
TraceSwitch trSwitch = new TraceSwitch("SwitchLevel", "Tracing");
```

С помощью трассировочных переключателей операторы Trace проверяют, следует ли записывать трассировочные данные. Методы Trace.WriteIf и Trace.WriteLineIf позволяют проверить, активен ли переключатель трассировки, и узнать определяемый им уровень трассировки:

```
Trace.WriteLineIf(bSwitch.Enabled == true, "Trace Boolean switch");
Trace.WriteLineIf(trSwitch.Level == TraceLevel.Info, "Trace level switch");
```

Операторы Trace не подключаются к трассировочным переключателям автоматически, это необходимо написать программисту в коде программы. Подключение переключателей трассировки в программе показано в листинге 4.6.

Листинг 4.6. Подключение переключателей трассировки в программе

```
using System;
using System.Diagnostics;

namespace ConsoleApp
{
   class Program
```

```
   {
      static void Main(string[] args)
      {
         int[] numbers = new int[10];
         Random rand = new Random();

         BooleanSwitch bSwitch = new BooleanSwitch(
            "IsTraceOn", "Tracing");
         TraceSwitch trSwitch = new TraceSwitch(
            "SwitchLevel", "Tracing");

         Trace.WriteLineIf(bSwitch.Enabled == true,
             "Trace Boolean switch:");
         for (int i = 0; i < 10; i++)
         {
            numbers[i] = rand.Next();
            Console.WriteLine("{0}\t{1}", i, numbers[i]);

            Trace.WriteLineIf(bSwitch.Enabled == true, String.Format(
               "{0}:\t{1}\t{2}", DateTime.Now, i, numbers[i]));
         }

         Trace.WriteLineIf(trSwitch.Level == TraceLevel.Info,
            "Trace level switch:");
         for (int i = 0; i < 10; i++)
         {
            numbers[i] = rand.Next();
            Console.WriteLine("{0}\t{1}", i, numbers[i]);

            Trace.WriteLineIf(trSwitch.Level==TraceLevel.Info,
               String.Format("{0}:\t{1}\t{2}",
                  DateTime.Now,i, numbers[i]));
         }
         Trace.Flush();

         Console.Read();
      }
   }
}
```

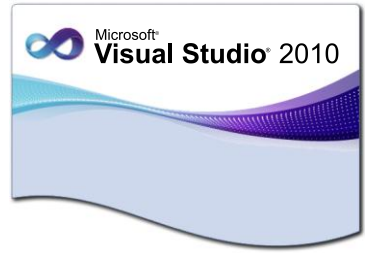
При исполнении кода, создающего трассировочный переключатель, приложение проверяет наличие в файле конфигурации сведений об этом переключателе. Для каждого переключателя проверка производится один раз при запуске приложения. Чтобы изменить параметры трассировочного переключателя после его создания, необходимо остановить приложение, внести соответствующие изменения в файл конфигурации и запустить приложение снова.

Примечание

При использовании методов, выполняющих безусловную запись, например `Trace.Write()` или `Trace.WriteLine()`, результаты трассировки записываются во всех случаях, независимо от состояния переключателей.

Резюме

Visual Studio 2010 предоставляет многочисленные инструменты для наблюдения за исполнением и отладки программы. Отладочные окна позволяют отслеживать значения переменных программы во время ее исполнения. Имеются также дополнительные окна для наблюдения за самыми разными данными приложения, что значительно облегчает процесс отладки приложений.



ГЛАВА 5

Создание приложений Windows Forms

Форма — это основной элемент пользовательского интерфейса, основа для построения приложения под Windows. Обычно формы отображают логически связанные данные и содержат инструментальные средства, позволяющие пользователю получать информацию, необходимую для продолжения работы.

Это основа взаимодействия пользователя с приложением на любом уровне. Для поддержки необходимой функциональности к приложению добавляют соответствующие элементы управления и меню. Дизайн интерфейса очень важен, поскольку выдержанный в едином стиле и логично организованный интерфейс облегчает освоение и эффективное использование программы.

Несмотря на то, что для построения пользовательского интерфейса появились более эффективные технологии, такие как Windows Presentation Foundation (WPF), приложения Windows Forms будут еще актуальны, по крайней мере, в течение ближайших нескольких лет.

Создание проекта Windows Forms Application

В Visual Studio основой для создания приложений Windows Forms является шаблон проекта Windows Forms Application. В диалоговом окне **Add New Project** выберите этот шаблон, дайте проекту имя WindowsFormsApp и нажмите кнопку **OK** (рис. 5.1).

Шаблон проекта Windows Forms состоит по умолчанию из класса формы и из статического класса `Program`. После создания проекта вы получаете пустую форму, открытую в визуальном конструкторе Windows Forms.

В файле Program.cs (листинг 5.1) расположен класс `Program`, в котором определен метод `Main()` — точка входа по умолчанию любого C#-приложения. С этого метода начинается выполнение программы.

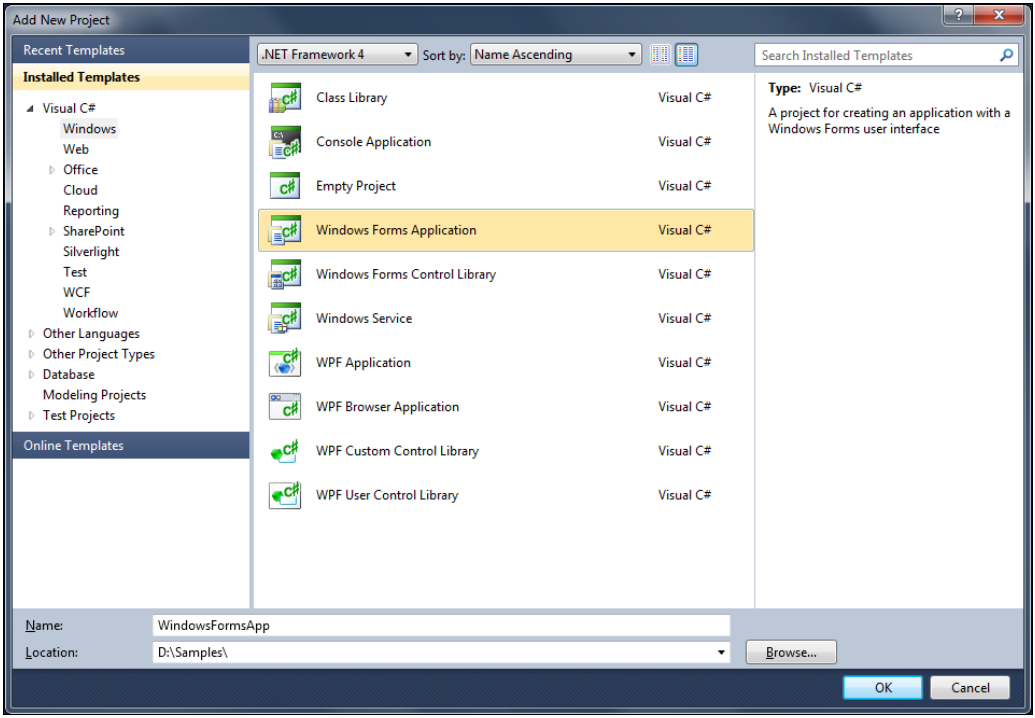


Рис. 5.1. Создание нового проекта Windows Forms

Листинг 5.1. Файл Program.cs

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

namespace WidowsFormsApp
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Обратите внимание, что заголовку метода `Main()` предшествует атрибут `STAThread`. `STAThread` устанавливает модель организации поточной обработки (threading model), а именно модель с однопоточным управлением, т. е. такую обработку данных, когда все объекты выполняются в едином процессе (single-threaded apartment — STA). Windows-приложение может использовать одну из двух моделей: с однопоточным или многопоточным управлением.

Метод `Application.Run()` отвечает за запуск стандартной очереди сообщений приложения. У метода `Application.Run()` есть три перегруженных варианта:

- `Run()`
- `Run(ApplicationContext)`
- `Run(Form)`

Класс `Application` содержит несколько очень полезных функциональных возможностей. Он обеспечивает статические методы и свойства для управления запуском и завершением приложения и получения доступа к сообщениям Windows, которые обрабатываются приложением. Вот некоторые наиболее полезные из этих методов и свойств:

- `CommonAppDataPath` — путь для данных, которые характерны для всех пользователей приложения. Как правило, это BasePath\Company Name\Product Name\Version, где BasePath — C:\Documents and Settings*%username%*\ ApplicationData. Если он не будет существовать, то путь будет создан;
- `ExecutablePath` — путь и имя исполняемого файла;
- `LocalUserAppDataPath` — подобен `CommonAppDataPath`, за исключением того, что это свойство поддерживает передвижение;
- `MessageLoopStartupPath` — возвращает `true`, если цикл обработки сообщения существует на текущем потоке, иначе `false`. Подобен `ExecutablePath`, за исключением того, что имя файла не возвращено;
- `AddMessageFilter()` — используется для предварительной обработки сообщения. Этот метод добавляет фильтр сообщений для мониторинга сообщений Windows при их маршрутизации к местам назначения;
- `ExitThread()` — заканчивает циклы обработки сообщений в текущем потоке и закрывает все окна в потоке;
- `Exit()` — заканчивает все выполняющиеся в настоящее время циклы сообщения, а затем закрывает все окна приложения после обработки сообщений и выходит из приложения.

В примере класса `Program` из листинга 5.1 объект `Form1` станет *главной формой приложения*. Это означает, что когда эта форма закрывается, приложениях завершает свою работу.

Сам класс формы `Form1` разделен на два файла. Причина в том, что начиная с Visual Studio 2005 используется возможность частичных (partial) классов и среда разработки выделяет весь код, сгенерированный визуальным конструктором,

в отдельный файл. Если используется имя по умолчанию — `Form1`, то эти два файла будут называться Form1.cs и Form1.Designer.cs. Если у вас не включена опция **Show All Files** (Показать все файлы) в меню **Project**, то вы не увидите в окне **Solution Explorer** файла Form1.Designer.cs.

В файле Form1.cs, приведенном в листинге 5.2, мы видим только операторы `using` и конструктор класса с вызовом метода `InitializeComponents()`. Сгенерированная средой разработки вторая часть — код класса `Form1`, помещенная в файл Form1.Designer.cs, показана в листинге 5.3.

Листинг 5.2. Файл Form1.cs

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

namespace WindowsFormsApplication6
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Листинг 5.3. Файл Form1.Designer.cs

```
namespace WidowsFormsApp
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
```

```
        /// <param name="disposing">true if managed resources should be
        /// disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support — do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.Text = "Form1";
        }
        #endregion
    }
}
```

Все кодирование должно осуществляться только в файле Form1.cs. Файл Form1.Designer.cs не должен редактироваться непосредственно разработчиком. Единственное исключение из этого правила — если необходима специальная обработка завершения формы в методе `Dispose()`, о чем будет рассказано далее.

Рассмотрим теперь строку кода:

```
private System.ComponentModel.IContainer components = null;
```

Класс формы поддерживает интерфейс `IDisposable`, поэтому, когда компонент будет добавлен к объекту-контейнеру коллекции компонентов, в этом контейнере производится проверка того, что компоненты инициализированы должным образом при создании формы и освобождаются при закрытии формы. Это можно увидеть в методе `Dispose()`:

```
protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}
```

При вызове метода `Dispose()` происходит вызов метода `Dispose()` объекта дочернего компонента, и, т. к. объект содержит другие компоненты, они также будут уничтожены.

Теперь рассмотрим сгенерированный средой разработки конструктор класса формы, который находится в файле Forml.cs:

```
public Form1()
{
   InitializeComponent();
}
```

Сам метод `InitializeComponent()` расположен в Forml.Designer.cs и должен инициализировать любые элементы управления, которые могут находиться на форме. В этом методе также инициализируют свойства формы.

Этот метод привязан к дизайнеру Windows Forms в Visual Studio. Когда вы делаете изменения в форме при использовании **Form Designer**, например, ставите на форму кнопку или другой элемент управления, эти изменения сразу отразятся в методе `InitializeComponent()`. Если вы сделаете какие-либо изменения в теле метода `InitializeComponent()` вручную, а затем через дизайнер Windows Forms, то ваши изменения, сделанные вручную, будут потеряны. Метод `InitializeComponent()` переписывается после каждого изменения в конструкторе Windows Forms. Если вы должны добавить дополнительный код инициализации для формы или средств управления и компонентов на форме, сделайте это после вызова `InitializeComponent()`.

Иерархия классов Windows Forms

Обратите внимание на использование наследования для создания класса `MainForm`:

```
public class Form1 : Form
```

Класс `Form` — один из главных классов в пространстве имен `System.Windows.Forms`. Важность понимания иерархии становится очевидной во время проектирования приложений на основе Windows Forms. Иерархия классов приведена на рис. 5.2.

Наверху иерархии стоит класс `Object`. В .NET все типы данных (как структурные, так и ссылочные) производятся от единого общего предка: класса `System.Object`. Класс `Object` определяет общее полиморфное поведение для всех типов данных в .NET.

Наследование от класса `MarshalByRefObject` гарантирует, что обращение к этому типу будет производиться по ссылке вместо создания локальной копии. Таким образом, к примеру, если мы обратимся к форме на удаленном компьютере, мы будем работать со ссылкой на нее, а не с локальной копией.

Класс `Component` обеспечивает инфраструктуру, необходимую для того, чтобы его можно было перетаскивать и помещать на поле конструктора, а также, чтобы он мог включать в себя другие элементы управления.

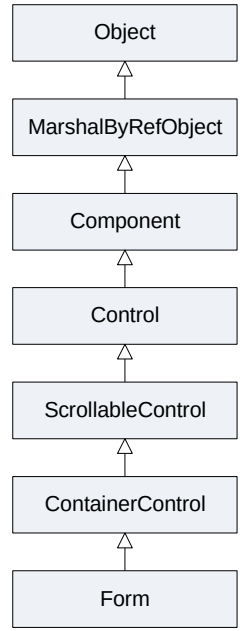


Рис. 5.2. Иерархия классов формы

Следующий базовый класс, обеспечивающий форме нужную фунциональность, — класс `Control`. Этот класс определяет общие черты для всех типов, относящихся к элементам графического интерфейса. `Control` позволяет настраивать размер и местонахождение окна, обрабатывать ввод с клавиатуры и мыши. Класс `Control` предоставляет основные функциональные возможности для создания внешнего вида приложения. Класс `Control` обеспечивает большой список функциональных возможностей для классов-наследниковв. Список слишком длинный, чтобы перечислить его здесь. В целом класс `Control` определяет множество методов, которые могут быть замещены в производных классах для реагирования на разные события (например, события взаимодействия с пользователем).

В классе `ScrollableControl` определено всего несколько членов, главное назначение которых — обеспечить поддержку вертикальной и горизонтальной полос прокрутки. Наиболее часто используемыми типами этого класса являются свойства `AutoScroll` и `AutoScrollMinSize`. Эти свойства обеспечивают автоматическое появление полос прокрутки в тех ситуациях, когда содержимое формы не умещается в границах формы (например, если пользователь уменьшил ее размер). Свойство `AutoScrollMinSize` позволяет задать минимальный размер формы, при котором всегда будут появляться полосы прокрутки. Класс `ScrollableControl` позволяет также принудительно обеспечивать появление полос прокрутки при помощи отдельных типов `HScrollBar` и `VScrollBar`).

Члены класса `ContainerControl` позволяют управлять фокусом — т. е. выделением отдельных элементов на форме (например, переход фокуса может осуществляться с помощью клавиши <Tab>). Чаще всего возможности этого класса используются в тех случаях, когда на форме присутствует множество элементов управления, и мы хотим обеспечить пользователю удобство перехода по ним посредством клавиатуры. При помощи членов этого класса мы можем программным образом получить информацию о том, какой элемент управления выделен в настоящий момент, принудительно передать фокус определенному элементу управления и т. п.

Наконец, класс `Form` является сердцем клиентского приложения Windows. Из приведенной на рис. 5.2 иерархии наследования можно предположить, что форма может быть контейнером для других элементов, может иметь полосы прокрутки, когда содержащиеся элементы не соответствуют области клиента, и имеет многие из тех же самых свойств, методов и событий, которые есть у других элементов.

Свойства формы

У формы множество свойств, которые определяют ее облик. Их значения можно просматривать и изменять в дизайнере через окно свойств формы (рис. 5.3).

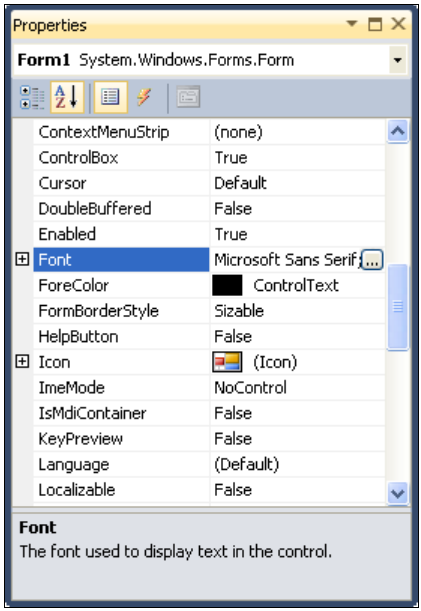


Рис. 5.3. Окно свойств формы

Некоторые свойства, такие как `Font`, в действительности заключают в себе несколько значений, каждое из которых влияет на вид формы по-своему. Щелкнув в окне **Properties** значок "+" рядом с именем свойства, вы сможете просмотреть и изменить каждое из этих значений. Для некоторых свойств, таких как `BackColor` и `ForeColor`, значения определяют средствами специального редактора.

Свойства `BackColor` и `ForeColor` — цвета, которые используются на форме. Свойство `ForeColor` определяет цвет текста переднего плана. Поскольку выбор цвета важен для дизайна пользовательского интерфейса, следует отнестись к нему очень внимательно. Чтобы облегчить чтение форм, следует применять высоко контрастные цветовые схемы.

Свойство `BackGroundImage` позволяет использовать изображение вместо однотонного фона. Если задано фоновое изображение, то изменение свойства `BackColor` формы отразится на значении одноименного свойства всех размещенных на ней элементов управления, но никак не повлияет на саму форму.

Свойство `ShowInTaskBar` получает или задает значение, указывающее, отображается ли форма на панели задач Windows.

Свойство `FormBorderstyle` устанавливает тип границы, которая появляется вокруг формы. Это свойство использует перечисление `FormBorderstyle`. Значения могут быть следующие:

- `Sizable`
- `SizableToolWindow`

- Fixed3D
- FixedDialog
- Fixedsingle
- FixedToolWindow
- None

Большинство значений этих свойств очевидно, за исключением двух инструментальных окон со стилями SizableToolWindow и FixedToolWindow — окна не будут появляться на панели задач, независимо от того, как установлено свойство ShowInTaskBar. Также инструментальные окна не будут показываться в списке окон, когда пользователь нажмет комбинацию клавиш <Alt>+<Tab>.

Для позиционирования окна формы на экране при ее первом отображении используется свойство StartPosition, которое принимает значение из перечисления FormStartPosition;

- CenterParent — форма центрируется в клиентской области родительской формы;
- CenterScreen — форма центрируется на текущем экране;
- Manual — местоположение формы основано на свойстве Location;
- WindowsDefaultBounds — форма располагается в позиции по умолчанию, определяемой Windows, и имеет размеры по умолчанию;
- WindowsDefaultLocation — форма располагается в позиции по умолчанию, определяемой операционной системой.

Методы формы

Методы применяются для исполнения тех или иных действий. Методы, являющиеся членами класса, выполняют действия, составляющие функциональность данного класса. Любая форма инкапсулирует базовый набор функций, унаследованный от класса System.Windows.Forms.Form, куда входят методы, управляющие отображением формы и доступом к ней в пользовательском окружении. Вот некоторые из них:

- Show() — загружает экземпляр класса формы в память, отображает его на экране и передает ему фокус ввода, при этом свойство Visible автоматически устанавливается в true. Если экземпляр формы уже загружен, но пока не видим (например, если его свойство Visible установлено в false), вызов метода Show() даст тот же результат, что и установка свойства Visible в true.
- ShowDialog() — выполняет те же действия, что и Show(), но делает окно формы модальным. Это означает, что другим формам приложения не удастся получить фокус, пока не закрыта форма, показанная при помощи метода ShowDialog(). Сделав окно формы модальным, вы заставите пользователя выполнить некоторое действие на этой форме, и только после этого он сможет продолжить работу с приложением.

- `Activate()` — активирует форму, и она получает фокус ввода. Если вызвать этот метод в окне приложения, у которого в текущий момент нет активных форм, окно этого приложения на панели задач начинает мигать. В любом случае активировать разрешено только видимую форму. Если вызвать этот метод из невидимой формы, он просто вернет управление.
- `Hide()` — делает форму невидимой. Форма остается в памяти, но останется невидимой, пока не будет вызван метод `Show()` или свойство `Visible` этой формы не будет установлено в `true`. Метод `Hide()` устанавливает свойство `Visible` в `false` (в принципе, аналогичный эффект достигается при установке этого свойства напрямую).
- `Close()`— закрывает форму. Этот метод закрывает все удерживаемые формой ресурсы и помечает ее как мусор, подлежащий сбору. После вызова метода `Close()` сделать форму видимой, вызвав метод `Show()`, не удастся, поскольку ресурсы формы уже освобождены. Вызов `Close()` на стартовой форме приложения завершает приложение.

Для вызова любого из этих методов необходима ссылка на форму, т. е. в памяти должен быть заранее созданный экземпляр формы. При запуске приложение автоматически создает экземпляр стартовой формы в дополнение к тем экземплярам форм, которые создаются в коде.

Жизненный цикл и события формы

На протяжении жизненного цикла формы генерируется ряд событий. В этом разделе мы рассмотрим те события, которые генерируются во время создания, функционирования и уничтожения формы. На рис. 5.4 показаны различные стадии (и соответствующие события) от зарождения формы и до ее закрытия.

Процесс создания формы важен для понимания принципов создания приложений Windows Forms. Для экземпляра Windows Forms события происходят в следующем порядке:

1. `Load` — генерируется, когда экземпляр формы впервые загружается в программу, т. е. при первом вызове метода `Form.Show` или `Form.ShowDialog` для экземпляра формы.
2. `Activated` — это событие многократно срабатывает в течение жизни формы. Оно генерируется, когда форма получает фокус. Так, это происходит при вызове методов `Show()`, `ShowDialog()` и `Activate()`. Обработчик события `Activated` применяют для автоматической передачи фокуса определенному элементу управления формы, а также для изменения цвета активного элемента управления, чтобы сделать его заметным пользователю. `Deactivated` генерируется, когда форма теряет фокус. Это происходит из-за взаимодействия пользователя с интерфейсом либо при вызове методов `Hide()` или `Close()`, однако метод `Close()` генерирует это событие, только если закрываемая форма является активной. Это событие применяют для проверки данных, введенных пользователем. События `Activated` и `Deactivated` генерируются только при перемещении фокуса в пределах прило-

жения: если переключиться на другое приложение и обратно, ни одно из них не сработает.

3. `Shown` вызывается только при первом отображении формы; последующее свертывание, развертывание, восстановление, скрытие, отображение, перерисовка и отмена действительности формы не приводит к созданию этого события.
4. `Closing` — генерируется, когда текущая форма начинает закрываться (но еще не закрыта), например при вызове метода `Form.Close` или щелчке кнопки **Close**. Это событие позволяет проверить, все ли действия, обязательные для данной формы, выполнены, например, заполнены ли все обязательные поля. В зависимости от результатов проверки, можно прервать процесс закрытия и оставить форму открытой.
5. `Closed` — генерируется после закрытия формы. Подобно событию `Closing`, это происходит при вызове метода `Form.Close` или когда пользователь закрывает форму вручную. Событие `Closed` генерируется вслед за событием `Closing` после исполнения его обработчиков. Событие `Closed` позволяет исполнить любой код для очистки после закрытия формы.

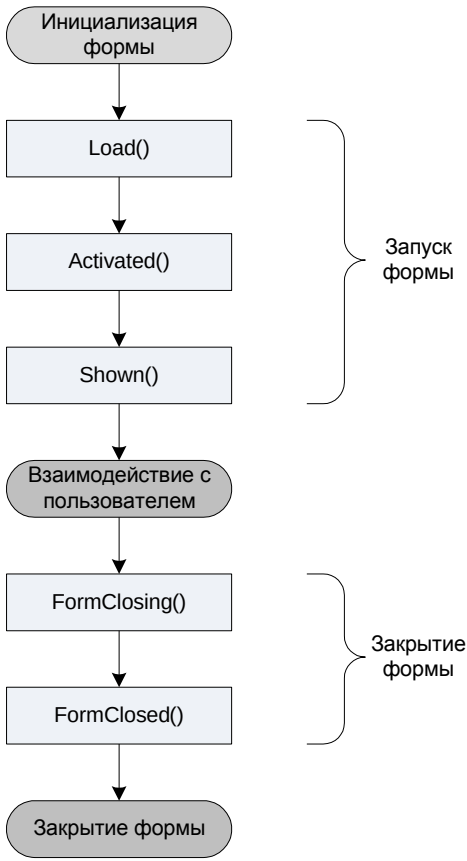


Рис. 5.4. События формы

Создание обработчика событий

Для реагирования на событие формы вам нужно сначала создать обработчик события. Если разработчик хочет заставить приложение выполнять некоторые действия в ответ на событие, он пишет обработчик события (event handler) — метод, исполняемый при генерации события. Например, в обработчик события `Deactivate` можно вставить код, проверяющий заполнение обязательных полей формы.

Окно **Properties** (Свойства) среды Visual Studio предоставляет быстрый механизм для описания обработчика события. Сначала надо выбрать интересующую вас форму. Затем нажать кнопку **Events** (События) на панели инструментов окна **Properties**. Теперь окно покажет вам список всех событий, определенных на форме. Двойной щелчок по событию создаст пустую процедуру обработчика события и откроет ее в редакторе кода. Обработчик события будет иметь корректный список аргументов и соблюдать установленные стандарты именования обработчиков событий (обычно это `Объект_ИмяСобытия`). На рис. 5.5 показан список доступных событий формы в окне **Properties**.

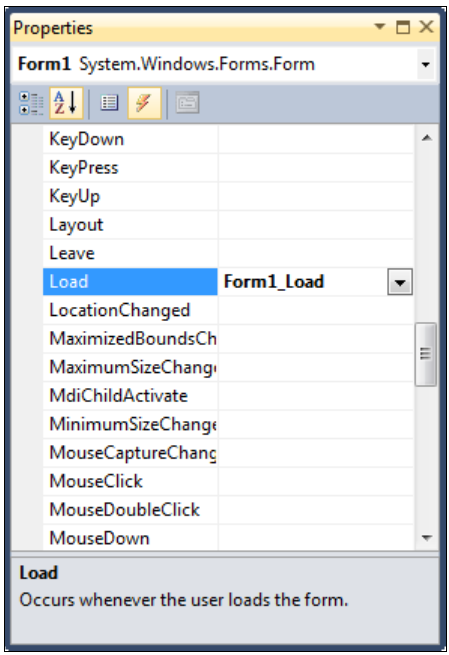


Рис. 5.5. События формы в окне свойств **Properties**

Компоновка и позиционирование элементов управления

Кратко рассмотрим элементы управления, которые формируют либо дополняют функциональность приложения. Их перетаскивают на форму с инструментальной панели **Toolbox**. Некоторые из них служат для получения ввода от пользователя, другие элементы выполняют базовые задачи по взаимодействию с пользователем.

Кроме того, предусмотрены специализированные компоненты, реализующие более сложные операции по взаимодействию с другими частями приложения. Компоненты похожи на элементы управления: это тоже готовые блоки кода, инкапсулирующие определенную функциональность. Основное отличие между ними в том, что элементы управления видимы, а компоненты — нет (по крайней мере в период выполнения).

Для форм Windows общепринятым является наличие нескольких основных компонентов: меню, панелей инструментов и строк состояния — все это стандартные элементы для каждого Windows-приложения.

Для создания внешнего вида формы используется визуальный дизайнер Windows Forms и панель **Toolbox**. Элементы управления перетаскивают на форму с инструментальной панели **Toolbox**. При размещении элемента управления на форме его код автоматически добавляется к приложению.

Привязка и закрепление элементов

Привязка — это концепция фиксирования статического (привязанного) положения левой, верхней, правой или нижней границы элемента управления внутри границ самой формы. Например, привязка метки к верхней и левой границе формы (так делается по умолчанию) приведет к тому, что метка будет сохранять свое положение вне зависимости от того, как будет изменяться размер формы. Свойство `Anchor` любого элемента управления может быть настроено на любую комбинацию значений `Top`, `Left`, `Bottom` и `Right`. Браузер свойств элемента управления предоставляет удобный виджет редактора свойств (рис. 5.6), который графически выделяет те стороны элемента управления, которые привязаны.

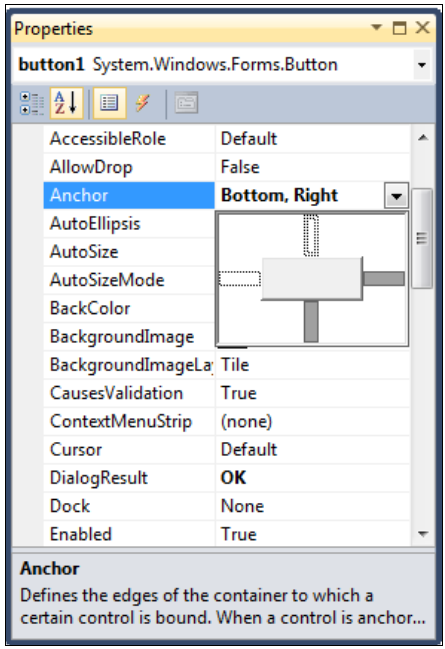


Рис. 5.6. Настройка свойства `Anchor`

Привязка противоположных сторон элемента управления имеет интересный эффект. Поскольку каждая сторона должна сохранять свое положение относительно сторон формы, то сам элемент управления будет растягиваться по вертикали или по горизонтали (в зависимости от того, какие стороны были привязаны — `Top` и `Bottom` или `Right` и `Left`). Фактически это именно то, что вам нужно для текстового поля: вы хотите, чтобы его ширина и высота изменялись при изменении размера формы. При привязке всех сторон элемента управления вы получите поведение, показанное на рис. 5.7; элемент управления автоматически подстроил свои размеры (при этом от разработчика не потребовалось написания никакого кода). По умолчанию элементы управления обычно привязываются по верхней и левой стороне.

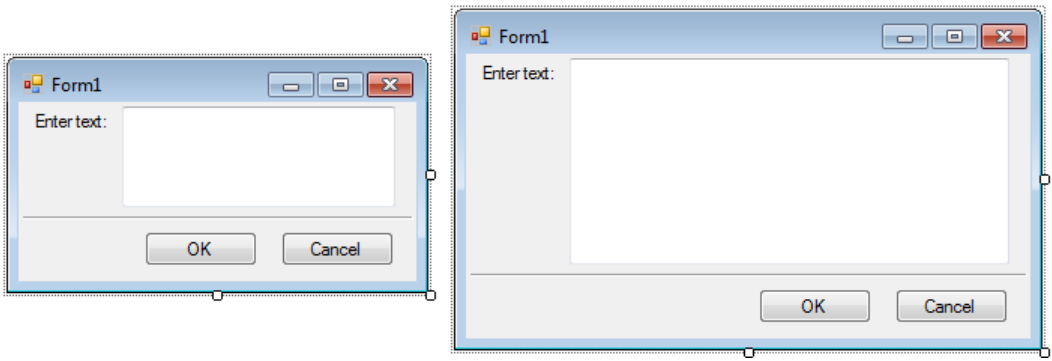


Рис. 5.7. Привязка элементов управления

Привязка также решает проблему позиционирования командных кнопок. Если вы измените их свойство `Anchor` на `Bottom, Right` — то они закрепятся к правой нижней стороне формы, что соответствует их рекомендуемому расположению на форме. Поскольку вы не закрепляете противоположные стороны элементов управ-

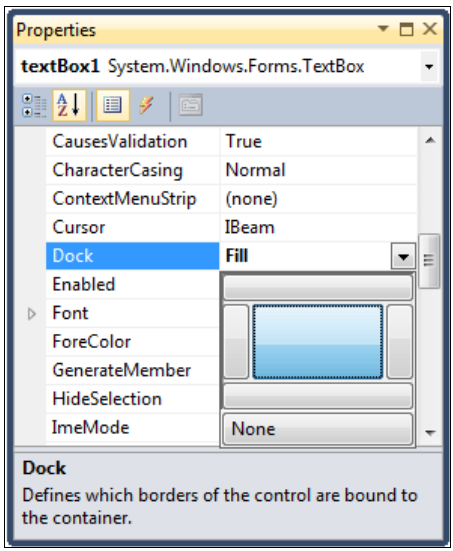


Рис. 5.8. Настройка свойства `Dock`

ления, то вы не заставляете кнопки изменять свой размер; они просто перемещаются для сохранения расстояния от правой и нижней границ формы.

С помощью свойства `Dock` можно задавать границы элемента управления, прикрепленные к его родительскому элементу управления, а также определять способ изменения размеров элемента управления при изменении размеров родительского элемента управления. Так же, как и в случае со свойством `Anchor`, окно **Properties** предоставляет графический инструмент для настройки свойства `Dock` элемента управления (рис. 5.8).

Приложения с несколькими формами

Большинство приложений Windows Froms содержат более одной формы. Дополнительные формы всегда можно добавить в проект во время разработки. Рассмотрим теперь взаимодействие форм в приложении.

Назначение стартовой формы

Если приложение Windows Forms содержит несколько форм, следует одну из них назначить стартовой. Давайте добавим в проект еще одну форму. Для этого щелкните правой кнопкой мыши на значке проекта в окне **Solution Explorer** и в контекстном меню выберите опцию **Add | New Item**, в появившемся окне **Add New Item** выберите **Windows Form** (рис. 5.9).

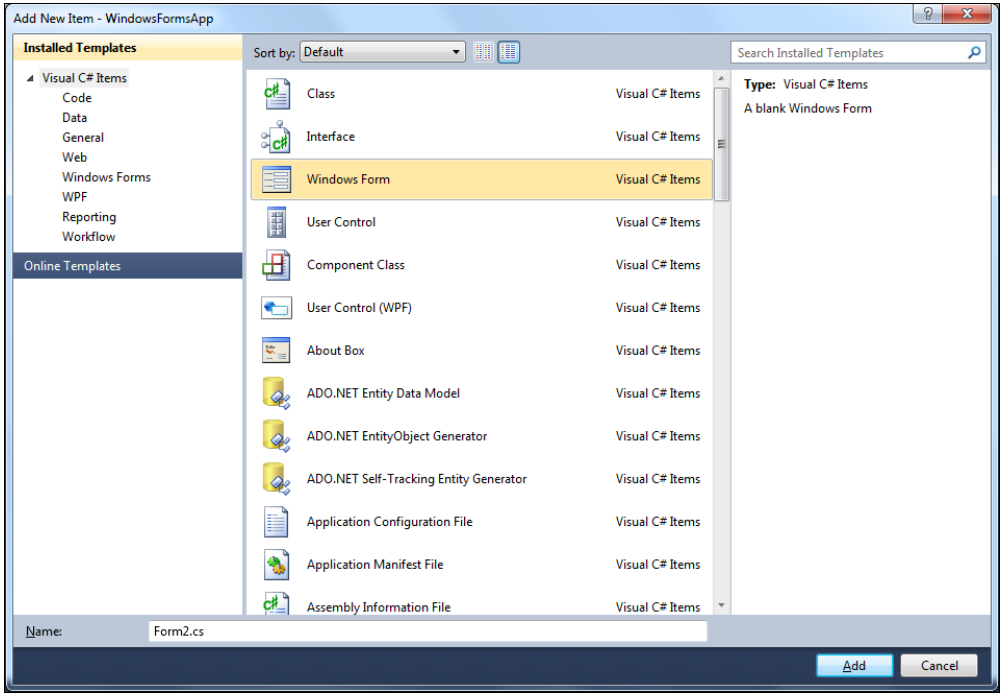


Рис. 5.9. Добавление новой формы в проект

По умолчанию класс новой формы будет иметь имя `Form2`. Теперь у нас в приложении две формы, однако, если запустить приложение, снова отобразится форма `Form1`. Переопределить стартовую форму можно в классе `Program` (файл Program.cs). В методе `Main()` есть вызов метода `Application.Run()`, в качестве параметра которому передается экземпляр стартовой формы, в нашем случае — `Form1`. Замените параметр в методе `Application.Run()` на `Form2`, как показано в листинге 5.4.

Листинг 5.4. Назначение стартовой формы

```
static void Main()
{
   ...
   Application.Run(new Form2());
}
```

Теперь при запуске приложения будет отображена форма `Form2`.

Переключение между формами

Окна бывают модальными и немодальными. При отображении другой формы используются два метода:

- `Show()` — для вызова немодальной формы;
- `ShowDialog()` — для вызова модальной формы.

Немодальные окна не делают своих владельцев неактивными — пользователь может в любой момент перейти к главной форме, не закрывая такой диалог. При отображении на экране модального диалога его владелец, которым обычно является главная форма приложения, временно становится неактивным.

Модальное (диалоговое) окно появляется на экране при вызове `ShowDialog()`. Вызов метода `ShowDialog()` — блокирующий; эта функция не возвращает управления, пока диалог не закроется. Ее возвращаемое значение указывает, как был закрыт диалог. Результат диалога формы является значением, которое возвращается из формы при ее отображении в виде модального диалогового окна. Если форма отображается в виде диалогового окна, установка этого свойства со значением из перечисления `DialogResult` устанавливает значение результата диалогового окна для формы, скрывает модальное диалоговое окно и возвращает элемент управления вызывающей форме. Это свойство обычно устанавливается свойством `DialogResult` элемента управления `Button` на форме. Когда пользователь выбирает элемент управления `Button`, значение, назначенное свойству `DialogResult` этой кнопки, назначается свойству `DialogResult` формы. Перечисление `DialogResult` имеет следующие значения:

- `None`
- `OK`
- `Cancel`
- `Abort`
- `Retry`
- `Ignore`
- `Yes`
- `No`

Давайте рассмотрим вызов формы на практическом примере. На форме Form1 разместим элемент TextBox, присвоим ему имя tbResult (свойство Name) и установим свойство ReadOnly в true. Добавим также элемент Label с надписью (свойство Text) **Data from Form2:** и кнопку с именем bForm2 и надписью **Call Form2** (рис. 5.10).

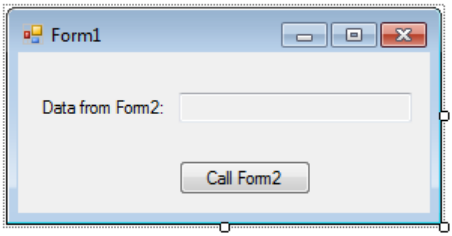


Рис. 5.10. Внешний вид формы Form1 в дизайнере форм

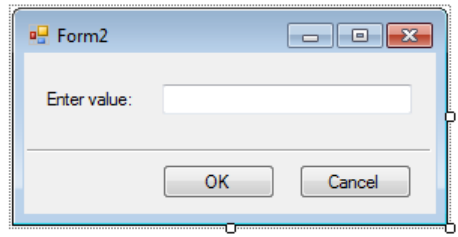


Рис. 5.11. Внешний вид формы Form2 в дизайнере форм

На форме Form2 разместим элемент TextBox, присвоим ему имя tbValue, элемент Label с надписью **Enter value:** и две кнопки: одну с именем bOK и надписью **OK** и кнопку с именем bCancel и надписью **Cancel** (рис. 5.11).

Для кнопки bOK установим свойство DialogResult в DialogResult.OK, а для кнопки bCancel — в DialogResult.Cancel. Щелчком мыши на кнопке bOK создадим обработчик события bOK_Click() и в класс Form2 добавим свойство Expression, куда будем записывать и откуда будем читать данные, вводимые пользователем в текстовое поле tbValue. Код класса Form2 представлен в листинге 5.5.

Листинг 5.5. Код класса Form2

```
namespace FormSwitch
{
   public partial class Form2 : Form
   {
      public Form2()
      {
         InitializeComponent();
      }

      public string Expression {get; set;}

      private void bOK_Click(object sender, EventArgs e)
      {
          Expression = tbValue.Text;
      }
   }
}
```

В классе `Form1` создадим обработчик события на вызывающей кнопке и в теле обработчика напишем код инициализации и вызова формы `Form2` с выводом результатов, как показано в листинге 5.6.

Листинг 5.6. Код класса `Form1`

```
namespace FormSwitch
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void bForm2_Click(object sender, EventArgs e)
      {
         Form2 childForm = new Form2();

         // Если пользователь закроет окно кнопкой OK
         if (childForm.ShowDialog() == DialogResult.OK)
         {
            tbResult.Text = childForm.Expression;
         }
      }
   }
}
```

Запустите приложение. Работа приложения показана на рис. 5.12. Возвращаемое из метода `ShowDialog()` значение `DialogResult.OK` означает, что пользователь щелкнул кнопку **OK**. Если `ShowDialog()` возвращает значение, отличное от `DialogResult.OK`, то вызывающая форма игнорирует введенные данные.

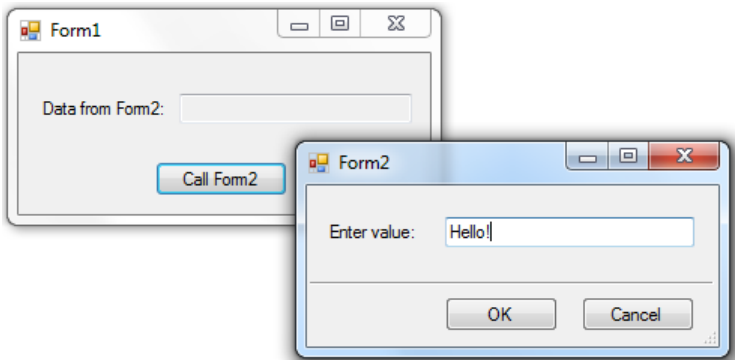


Рис. 5.12. Обмен данными между формами

Принципы создания пользовательского интерфейса

Интерфейс прежде всего должен учитывать потребности конечных пользователей приложения, которых принято называть целевой аудиторией. Если целевая аудитория программы известна, разработка пользовательского интерфейса не представляет трудностей. Рационально сконструированный интерфейс облегчает освоение и применение программы, а неудачный вызывает у клиентов ощущение бессилия и снижает эффективность работы с приложением. В результате пользователи все чаще игнорируют его и даже могут полностью отказаться от него. Для поддержки необходимой функциональности к приложению добавляют соответствующие элементы управления и меню.

Меню

Элемент управления `MenuStrip` представляет контейнер для структуры меню формы. Можно добавить объекты `ToolStripMenuItem` в объект `MenuStrip`, который представляет отдельные команды в структуре меню. Каждый объект `ToolStripMenuItem` может быть командой для приложения или родительским меню для других элементов вложенного меню.

Элемент управления `MenuStrip` дает возможность визуально конструировать систему главного меню формы. Перетаскивание этого элемента управления из панели `Toolbox` на пустую форму автоматически прикрепит меню к верхнему краю формы

После того как вы поместили на форму этот элемент управления, выбор элемента `MenuStrip` активирует смарт-тег. Смарт-тег позволяет автоматически вставить в меню стандартные пункты и выставить свойства `RenderMode`, `Dock` и `GripStyle`.

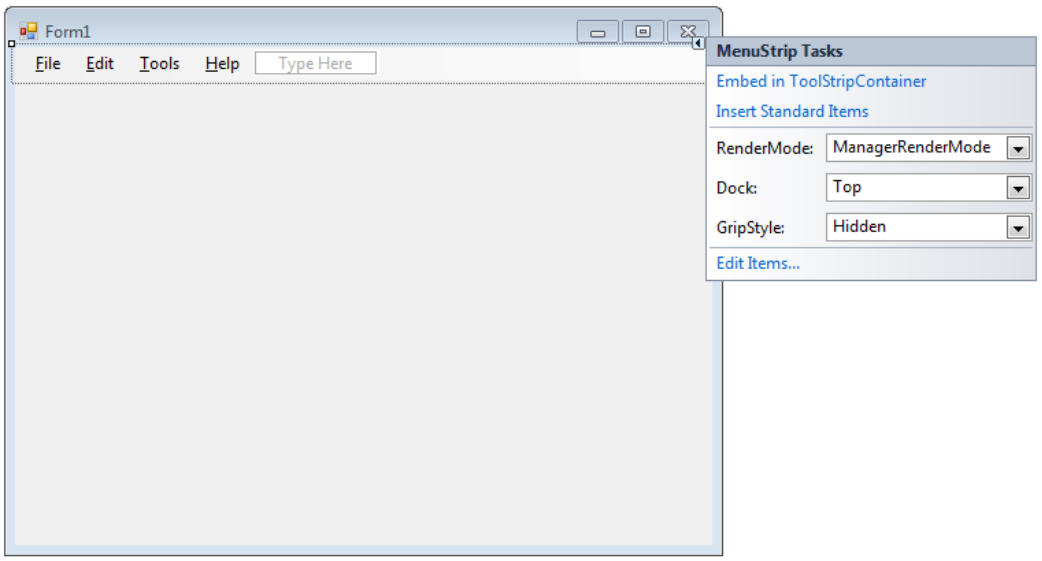


Рис. 5.13. Меню с добавленными стандартными пунктами

Использование возможности автоматического оснащения полоски меню стандартным набором пунктов меню экономит несколько минут по сравнению с ручным вариантом выполнения этой работы (рис. 5.13).

Дизайнер добавляет не только стандартные пункты меню верхнего уровня **File**, **Edit**, **Tools** и **Help**, но и подпункты в каждом меню.

Если вы хотите вручную добавить пункты меню в строку меню, то вы можете использовать местозаполнитель внутри полоски на месте надписи **Type Here**. При вводе новой опции меню в местозаполнителе становятся видимыми дополнительные местозаполнители, а в строку меню добавляется новый пункт (рис. 5.14).

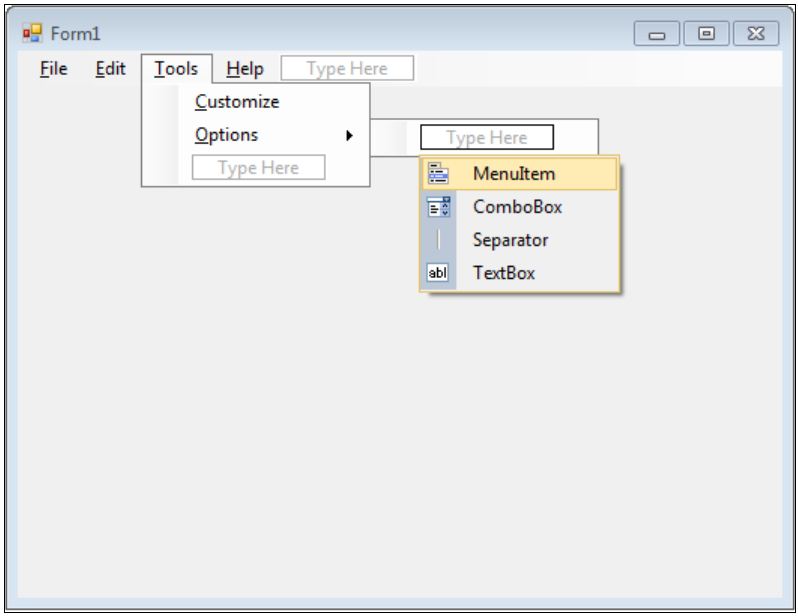


Рис. 5.14. Добавление пунктов меню

Панель инструментов

Следующий элемент для формы — это панель инструментов. Панели инструментов реализуются при помощи элементов управления `ToolStrip`. Элементы управления `ToolStrip` могут содержать различные дочерние элементы управления; каждый из них наследует от базового класса `ToolStripItem`.

Фактически интерактивные функции компоновки полоски инструментов работают так же, как и у полоски меню: перетаскивание элемента управления на форму приведет к тому, что пустой элемент управления `ToolStrip` будет прикреплен к верхнему краю формы непосредственно под меню, и вы сможете быстро добавить в панель инструментов `ToolStrip` набор стандартных элементов при помощи ее смарт-тега в правой части элемента, выбрав пункт **Insert Standard Items** (Вставить стандартные элементы).

Элемент управления ToolStrip — контейнерный элемент, используемый при создании панели инструментов, главного меню и строки состояния. Элемент управления ToolStrip используется непосредственно для панелей инструментов и служит базовым классом для элементов StatusStrip и MenuStrip, представляющих главное меню и строку состояния.

Элемент управления ToolStrip использует набор элементов управления, классы которых являются производными от класса ToolStripItem. Основные свойства элементов управления ToolStripItem — это Image и Text. Графические изображения могут быть установлены либо через свойство Image, либо с использованием элемента управления ImageList и установкой свойства ImageList элемента управления ToolStrip. Затем могут быть установлены свойства ImageIndex индивидуальных элементов управления.

Форматирование текста в ToolStripItem управляется свойствами Font, TextAlign и TextDirection. Свойство TextAlign устанавливает выравнивание текста относительно элемента управления. Это может быть любое значение из перечисления ControlAlignment. По умолчанию принимается MiddleRight.

Свойство TextDirection устанавливает ориентацию текста. Значения могут быть любыми из перечисления ToolStripTextDirection, а именно — Horizontal, Inherit, Verticai270 и Vertical90. Vertical270 поворачивает текст на 270 градусов, а Vertical90 — на 90 градусов.

Свойство DisplayStyle управляет тем, отображается ли текст, изображение, то и другое или ни то ни другое на поверхности элемента управления. Когда AutoSize установлено в true, ToolStripItem будет изменять свой размер, потому потребуется минимальное пространство.

Порядок по *z*-координате играет важную роль в размещении прикрепленных элементов управления. Прикрепленные элементы упорядочиваются на форме по возрастанию их *z*-индекса. Например, если вы выделите ToolStrip и выберете команду **Send to Back**, то порядок контейнеров MenuStrip и ToolStrip изменится таким образом, чтобы поместить ToolStrip первым (наверху формы), а MenuStrip — вторым (сразу под ToolStrip).

На рис. 5.15 показана форма с добавленным элементом управления ToolStrip со стандартными элементами управления.

Для заполнения панели инструментов используется диалоговое окно **Items Collection Editor**. Для открытия окна **Items Collection Editor** в смарт-теге **SmartStrip** выберите пункт **Edit Items**. Редактор обеспечивает прямой доступ ко всем свойствам вложенного элемента управления, а также позволяет вам редактировать, удалять и переупорядочивать элементы внутри полосы состояния (рис. 5.16).

Каждому пункту меню могут быть назначены горячие клавиши. Когда горячая клавиша назначена, она может быть отображена в меню установкой значения свойства ShowShortCutKey в true.

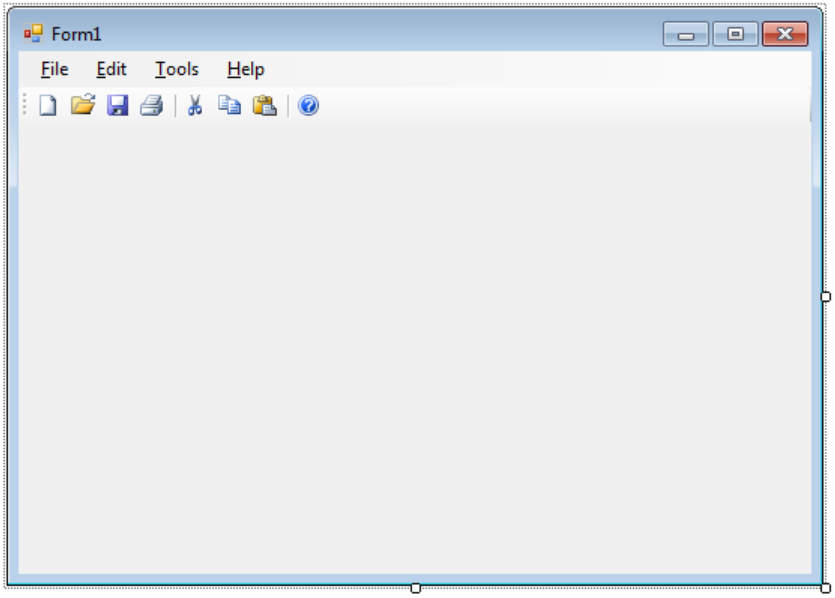


Рис. 5.15. Форма со стандартными меню и панелью инструментов

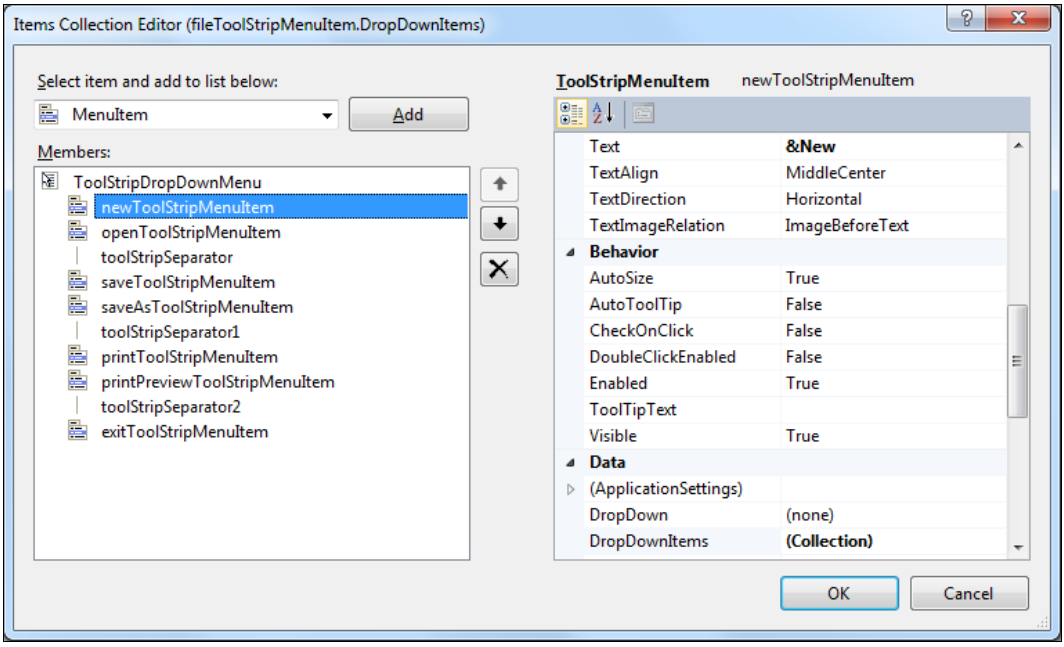


Рис. 5.16. Редактор **Items Collection Editor**

Строка состояния

Строки состояния дают пользователю информацию о текущем статусе приложения, о продвижении некоего действия, о выделенном на форме объекте и т. д.

В режиме конструирования (рис. 5.17) вы видите кнопку выпадающего списка, в котором имеется выбор всех четырех поддерживаемых дочерних элементов управления элемента `StatusStrip`.

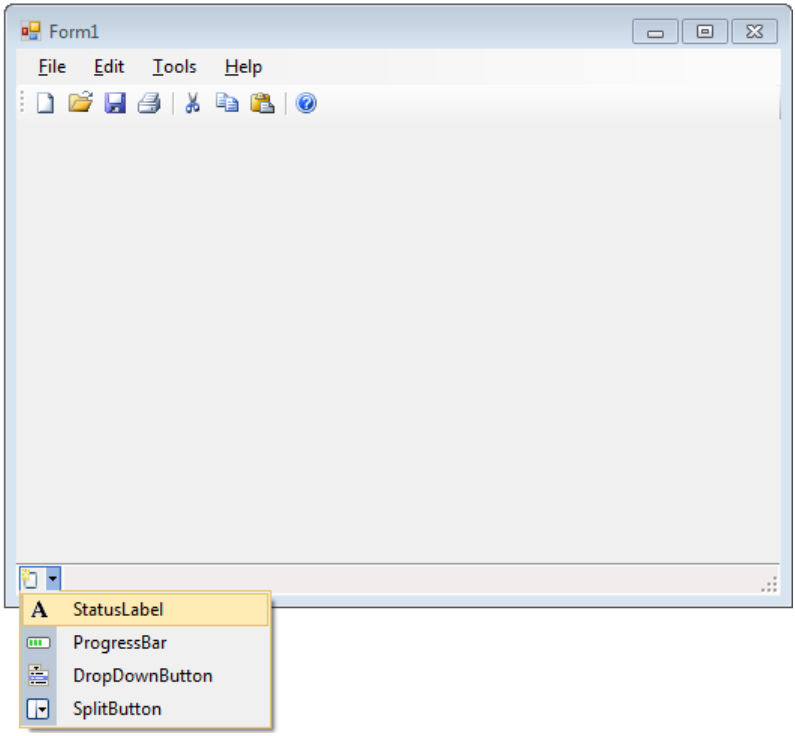


Рис. 5.17. Создание строки состояния

По умолчанию в элементе управления `StatusStrip` панели отсутствуют. Дочерние элементы управления можно добавлять внутри `StatusStrip` путем упорядочения потоком слева направо. Возможности редактирования "на месте" очень удобны для быстрого добавления дочерних элементов управления для `StatusStrip`. Если необходима более детальная настройка дочерних элементов управления `StatusStrip`, можно использовать диалоговое окно **Items Collection Editor**.

Использование контейнерных элементов

Контейнеры — это элементы управления, предназначенные для содержания других элементов управления. Для создания сложного дизайна вы можете использовать контейнеры в сочетании со свойствами `Anchor` и `Dock` элементов управления. Несмотря на то, что имеется множество различных контейнерных элементов

управления, самыми часто применяемыми являются элементы `FlowLayoutPanel`, `TableLayoutPanel` и `SplitContainer`.

Классы `TableLayoutPanel` и `FlowLayoutPanel` наследуют от класса `Panel`. Этот класс предоставляет возможности очень высокого уровня для группировки элементов управления. Это полезно для компоновки, поскольку вы можете сгруппировать несколько элементов управления посредством размещения их внутри одной панели. После этого вы можете работать с ними как с одной группой; например, деактивация панели приведет к деактивации всех ее дочерних элементов управления. Классы `TableLayoutPanel` и `FlowLayoutPanel` используют эту функциональность, а также добавляют возможность динамически влиять на позиционирование своих дочерних элементов управления.

Элемент управления `SplitContainer` состоит из двух панелей с линейкой или разделителем между ними. Пользователь может перемещать эту линейку, изменяя размеры панелей. При изменении размеров панелей элементы управления, содержащиеся на них, также могут изменять свой размер.

На рис. 5.18 показано два элемента управления `SplitContainer`: два разделенных контейнера, причем один из них располагается на панели, находящейся в другом контейнере. Эти элементы используются для обеспечения возможности как вертикального, так и горизонтального изменения размеров панелей на форме.

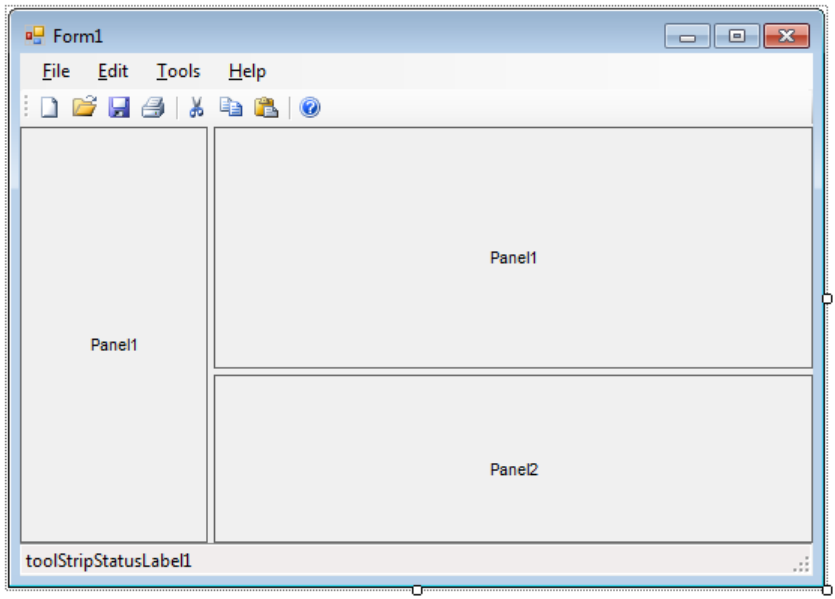


Рис. 5.18. Горизонтальный `SplitContainer` встроен в правый вертикальный `SplitContainer`

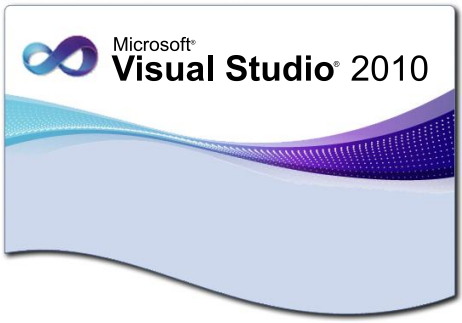
Первый элемент управления `SplitContainer` имеет вертикальный разделитель. Правая панель первого контейнера невидима, т. к. она является родительским контейнером для второго элемента управления `SplitContainer`, разделенного по горизонтали на панели `Panel1` и `Panel2`.

Перетаскиванием вправо или влево вертикальной линии разделителя можно изменять занимаемое правой панелью пространство на форме. Аналогично можно перемещать горизонтальную линию разделителя между панелями `Panel1` и `Panel2` второго контейнера `SplitContainer`.

Резюме

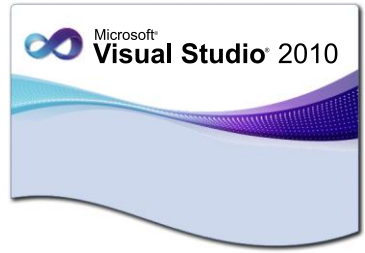
В этой главе было дано представление о базовых принципах построения клиентских Windows-приложений. Были описаны стандартные элементы управления, образующие иерархию классов пространства имен `Windows.Forms`, и рассмотрены различные их свойства и методы. Windows Forms — очень обширная тема, и на ее изучение может уйти несколько недель. Продолжить освоение этого материала можно со знакомства с многочисленными элементами управления, определенными в пространстве имен `System.Windows.Forms`. Однако, чтобы сэкономить время на освоение Visual Studio и не утомлять читателя бесконечными перечислениями элементов управления Windows Forms, я решил, что их изучение лучше провести параллельно с освоением другой большой темы — технологии доступа к данным ADO.NET, которую мы будем проходить на протяжении следующих нескольких глав этой книги, и параллельно изучим работу практически всех элементов управления Windows Forms.

ЧАСТЬ II



Технологии доступа к данным

ГЛАВА 6



Проектирование баз данных в Visual Studio 2010

Большинство создаваемых коммерческих приложений предназначены для обработки информации, хранящейся в базах данных. Visual Studio предоставляет множество средств разработки, которые призваны помочь в процессе создания приложений, получающих доступ к данным.

Эта глава о том, как вы можете управлять базами данных и создавать при помощи Visual Studio 2010 и SQL Server приложения, которые могут работать с данными. Здесь мы рассмотрим инструменты Visual Studio 2010 для работы с базами данных SQL Server и создадим свою учебную базу данных, с которой мы будем работать на протяжении всей книги, используя ее в качестве источника данных при разработке различных приложений.

Создание базы данных в Visual Studio

С помощью **Server Explorer** можно создать новый экземпляр базы данных SQL Server с нуля. Если до этого вы уже работали с SQL Management Studio 2008 или с SQL Management Studio 2005, вы увидите, что создание баз данных с помощью **Server Explorer** похоже на работу с базами данных в SQL Management Studio.

В этом разделе мы создадим базу данных в SQL Server 2008, которая будет использоваться в качестве источнка данных для наших учебных приложений на протяжении всех следующих глав.

Рассмотрим разработку базы данных, которая содержит информацию об учебном процессе вуза: списки студентов, перечень изучаемых предметов, сведения о преподавательском составе, сведения о занятиях, результаты экзаменов по предметам. База будет небольшая, только для учебного примера, и будет содержать всего пять таблиц:

- `Student`
- `Course`
- `Faculty`
- `Trainer`
- `Grade`

Примечание

Если у вас уже есть опыт разработки баз данных в SQL Management Studio или в прошлой версии Visual Studio, можете пропустить этот раздел и воспользоваться готовой базой данных, которая находится на диске с примерами в каталоге Samples\Database, и перейти к *разд. "Шаблоны проектов баз данных"* этой главы.

В окне **Server Explorer** в узле **Data Connections** есть список всех установленных подключений к базам данных. Для запуска процесса создания базы данных щелкните правой кнопкой мыши по узлу **Data Connections** и в контекстном меню выберите пункт **Create New SQL Server Database**. В появившемся одноименном диалоговом окне (рис. 6.1) вам будет необходимо указать имя сервера, тип SQL-аутентификации и название базы данных.

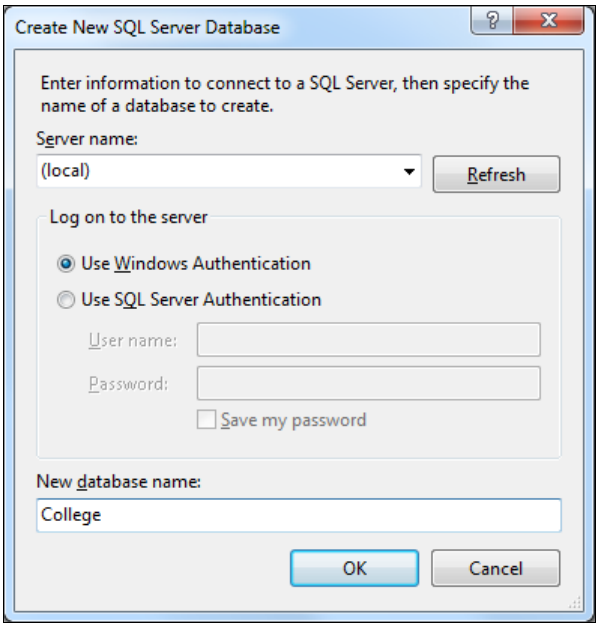


Рис. 6.1. Создание новой базы данных

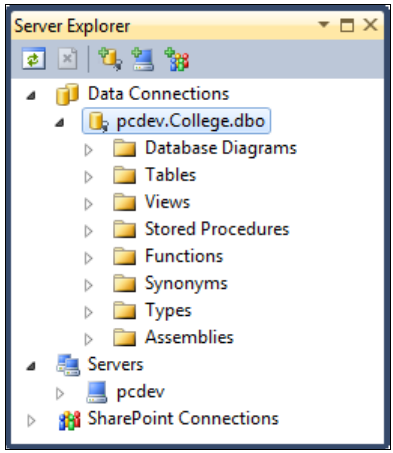


Рис. 6.2. База данных `College` в узле **Data Connections**

После нажатия кнопки **OK** база данных будет создана и в узле **Data Connections** в окне **Server Explorer** будет добавлено подключение к базе `College`, как показано на рис. 6.2.

Под значком, показывающим подключение к базе данных `College`, расположены каталоги для объектов баз данных. Эти каталоги являются стартовой точкой для создания внутри базы данных соответствующих объектов.

Определение таблиц

Для создания и редактирования таблиц в Visual Studio используется конструктор таблиц **Table Designer**. Если в окне **Server Explorer** в существующем подключении щелкнуть правой кнопкой мыши по каталогу **Tables** и выбрать пункт **Add New**

Table, в главной панели документов интегрированной среды откроется окно визуального конструктора таблиц **Table Designer**.

Этот конструктор похож на конструктор таблиц в SQL Management Studio: в нем вы добавляете строку для каждого столбца, который вы хотите определить в таблице. Для каждого столбца таблицы указывается имя столбца, тип данных и возможность использования значений типа `null`. В нижней части окна **Table Designer** расположена панель **Column Properties** (Свойства столбца), которая предоставляет доступ к свойствам каждого столбца таблицы (рис. 6.3).

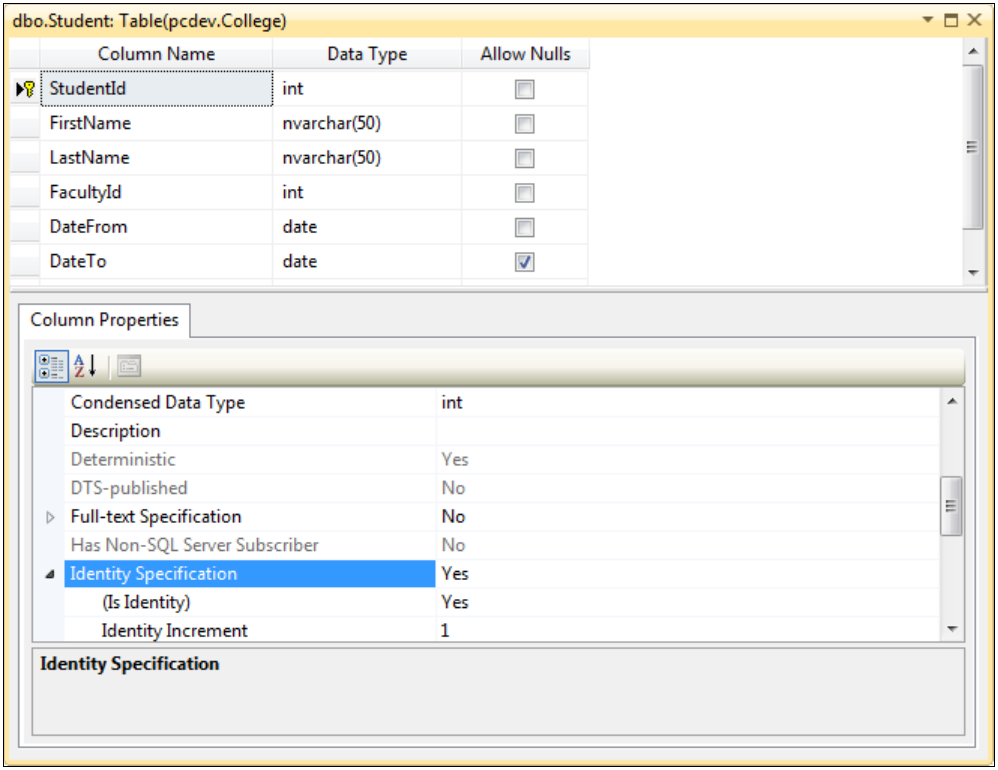


Рис. 6.3. Определение столбцов таблицы

Для создания таблиц можно использовать скрипт, показанный в листинге 6.1.

Листинг 6.1. SQL-скрипт для создания таблиц базы данных `College`

```
USE [College]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE TABLE [dbo].[Trainer](
   [TrainerId] [int] IDENTITY(1,1) NOT NULL,
   [FirstName] [nvarchar](50) NOT NULL,
   [LastName] [nvarchar](50) NOT NULL,
   [DateFrom] [date] NOT NULL,
   [DateTo] [date] NULL,
 CONSTRAINT [PK_Trainer] PRIMARY KEY CLUSTERED([TrainerId] ASC)
)
GO

CREATE TABLE [dbo].[Faculty](
   [FacultyId] [int] IDENTITY(1,1) NOT NULL,
   [FacultyTitle] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Faculty] PRIMARY KEY CLUSTERED([FacultyId] ASC)
)

CREATE TABLE [dbo].[Course](
   [CourseId] [int] IDENTITY(1,1) NOT NULL,
   [CourseTitle] [nvarchar](50) NOT NULL,
   [TrainerId] [int] NOT NULL,
   [FacultyId] [int] NOT NULL,
 CONSTRAINT [PK_Course] PRIMARY KEY CLUSTERED([CourseId] ASC)
)
GO

CREATE TABLE [dbo].[Student](
   [StudentId] [int] IDENTITY(1001,1) NOT NULL,
   [FirstName] [nvarchar](50) NOT NULL,
   [LastName] [nvarchar](50) NOT NULL,
   [FacultyId] [int] NOT NULL,
   [DateFrom] [date] NOT NULL,
   [DateTo] [date] NULL,
 CONSTRAINT [PK_Employee] PRIMARY KEY CLUSTERED ([StudentId] ASC)
)
GO

CREATE TABLE [dbo].[Grade](
   [GradeId] [int] IDENTITY(1,1) NOT NULL,
   [StudentId] [int] NOT NULL,
   [CourseId] [int] NOT NULL,
   [GradeValue] [int] NOT NULL,
   [Date] [date] NOT NULL,
 CONSTRAINT [PK_Grade] PRIMARY KEY CLUSTERED([GradeId] ASC)
)
GO
```

Создание диаграммы базы данных

Для создания диаграммы базы данных щелкните правой кнопкой мыши по узлу **Database Diagrams** (Диаграммы базы данных) в окне **Server Explorer** и выберите пункт **Add New Diagram** (Добавить новую диаграмму). Откроется пустая диаграмма, и визуальный дизайнер незамедлительно покажет диалоговое окно для добавления таблиц в диаграмму (рис. 6.4).

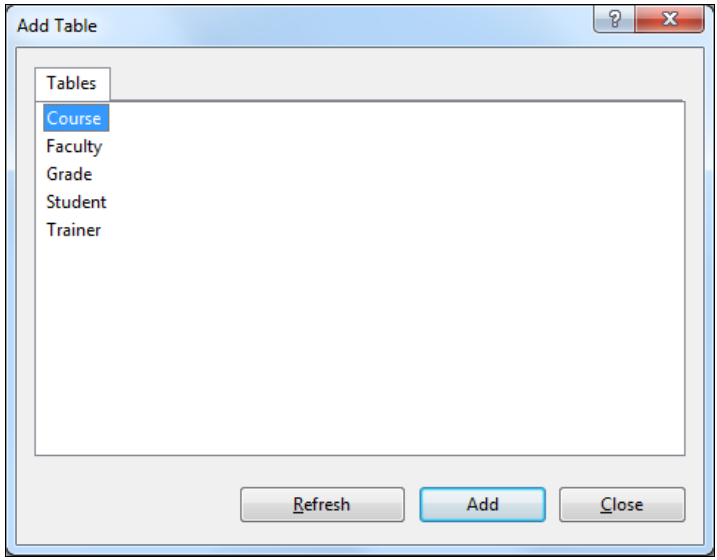


Рис. 6.4. Добавление таблиц в диаграмму

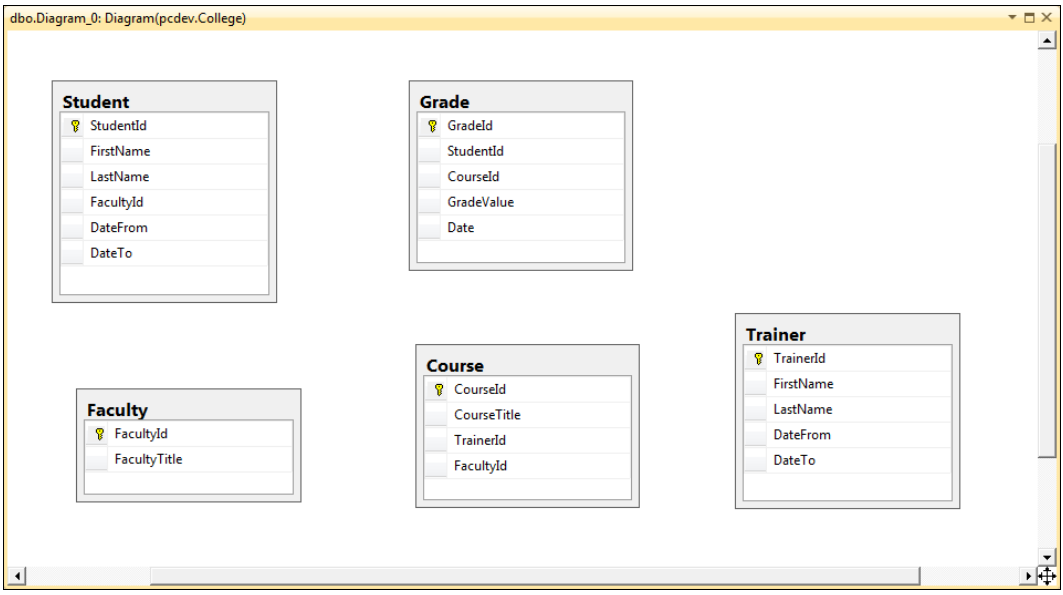


Рис. 6.5. Окно **Database Diagram Designer**

После того как вы добавите несколько таблиц, диаграмма покажет графическое представление столбцов таблиц и всех имеющихся между таблицами связей. Диаграмма полностью интерактивная и по своим функциональным возможностям соответствует редактору диаграмм в SQL Server Management Studio. Диаграмма для нашей базы данных, пока без связей между таблицами, показана на рис. 6.5.

Таблицы можно редактировать прямо внутри диаграммы. Для изменения столбца нужно щелкнуть мышью на столбце таблицы, выделив его, а затем ввести имя столбца или изменить тип данных этого столбца, либо правило использования значений типа `null`. Для добавления нового столбца надо заполнить новую строку в представлении таблицы в диаграмме.

Создание связей между таблицами

Связи между таблицами можно легко определить прямо внутри диаграммы. Для этого надо просто перетащить столбец первичного ключа из одной таблицы в столбец внешнего ключа другой таблицы. При этом появится диалоговое окно **Foreign Key Relationships** (Связи внешнего ключа), показанное на рис. 6.6.

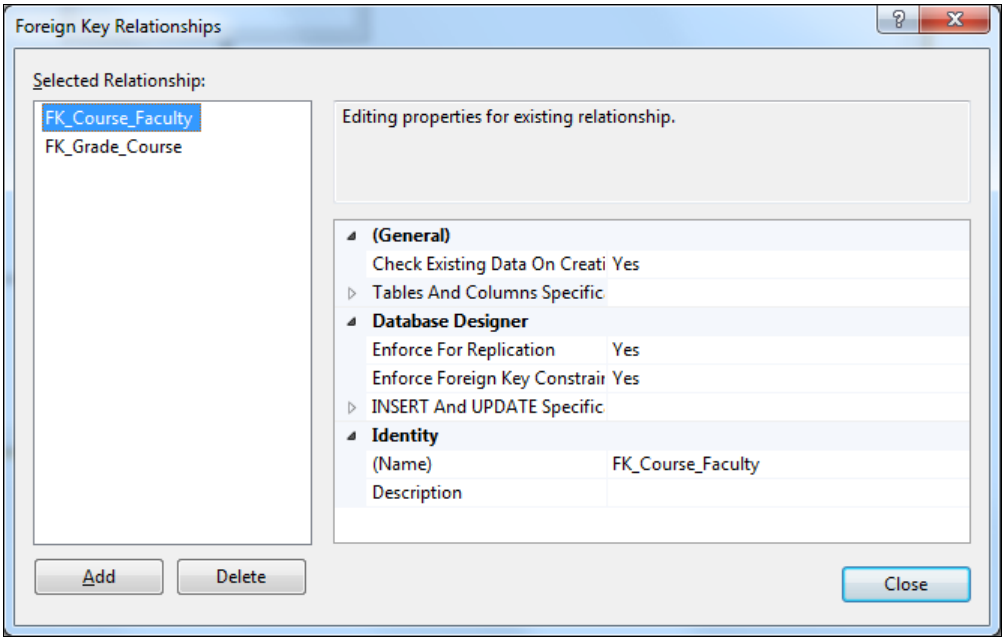


Рис. 6.6. Создание внешнего ключа

Для выбора ключевых столбцов в таблицах в окне свойств связи (справа) надо в строке **Tables And Columns Specification** (Определение таблиц и столбцов**)** нажать кнопку с многоточием и в открывшемся диалоговом окне **Tables and Columns** (Таблицы и столбцы) выбрать требуемую таблицу и ее столбцы. На рис. 6.7 показано создание связи "один-ко-многим" между таблицей `Trainer` и таблицей `Course`. Таблица `Course` имеет столбец `TrainerId`, который будет внешним ключом для первичного ключа таблицы `Trainer`.

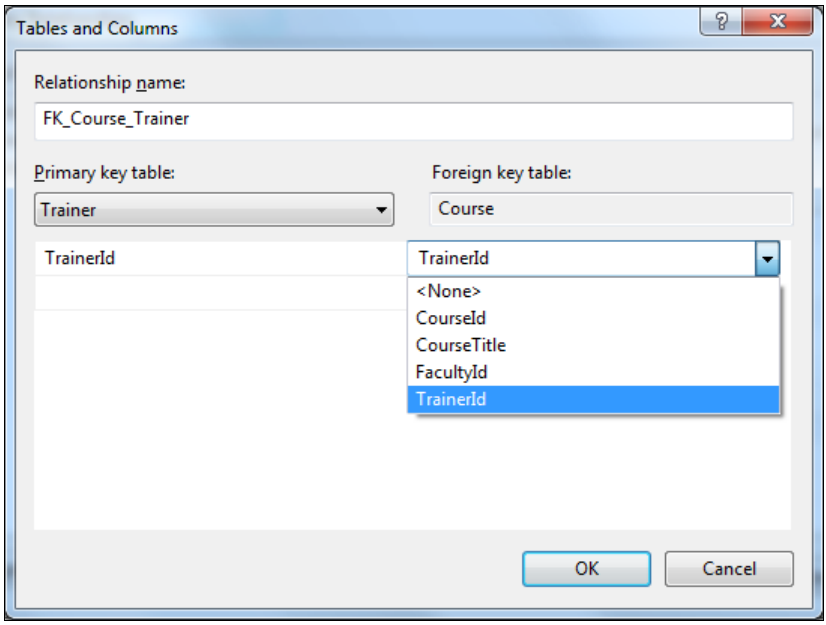


Рис. 6.7. Определение ключей

Связи на диаграмме изображаются в виде линии между двумя таблицами. Линия обозначает направление связи ключиком на стороне первичного ключа и символом бесконечности на стороне внешнего ключа. В результате вы должны создать связи между таблицами, как показано на рис. 6.8.

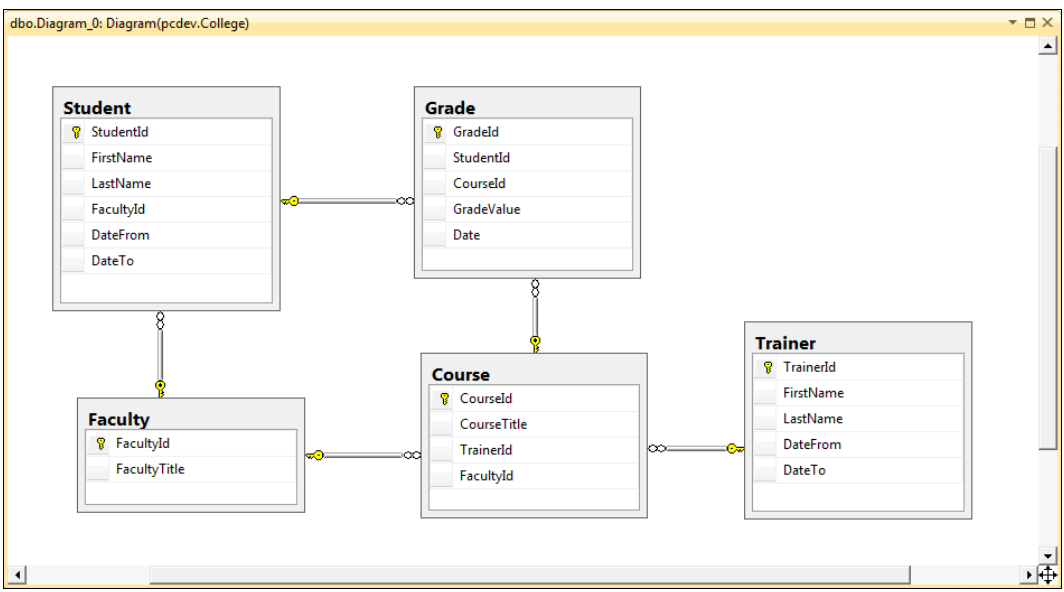


Рис. 6.8. Добавление связей между таблицами

Также вы можете для экономии времени, если у вас уже есть опыт работы в SQL Server Management Studio, просто исполнить скрипт из листинга 6.2, который создаст необходимые связи между таблицами.

Листинг 6.2. SQL-скрипт для создания связей между таблицами

```
Use [College]
GO

ALTER TABLE [dbo].[Course]  WITH CHECK ADD  CONSTRAINT [FK_Course_Faculty]
FOREIGN KEY([FacultyId])
REFERENCES [dbo].[Faculty] ([FacultyId])
GO
ALTER TABLE [dbo].[Course] CHECK CONSTRAINT [FK_Course_Faculty]
GO

ALTER TABLE [dbo].[Course]  WITH CHECK ADD  CONSTRAINT [FK_Course_Trainer]
FOREIGN KEY([TrainerId])
REFERENCES [dbo].[Trainer] ([TrainerId])
GO
ALTER TABLE [dbo].[Course] CHECK CONSTRAINT [FK_Course_Trainer]
GO

ALTER TABLE [dbo].[Grade]  WITH CHECK ADD  CONSTRAINT [FK_Grade_Course] FOREIGN
KEY([CourseId])
REFERENCES [dbo].[Course] ([CourseId])
GO
ALTER TABLE [dbo].[Grade] CHECK CONSTRAINT [FK_Grade_Course]
GO

ALTER TABLE [dbo].[Grade]  WITH CHECK ADD  CONSTRAINT [FK_Grade_Student]
FOREIGN KEY([StudentId])
REFERENCES [dbo].[Student] ([StudentId])
GO
ALTER TABLE [dbo].[Grade] CHECK CONSTRAINT [FK_Grade_Student]
GO

ALTER TABLE [dbo].[Student]  WITH CHECK ADD  CONSTRAINT [FK_Student_Faculty]
FOREIGN KEY([FacultyId])
REFERENCES [dbo].[Faculty] ([FacultyId])
GO
ALTER TABLE [dbo].[Student] CHECK CONSTRAINT [FK_Student_Faculty]
GO
```

Разработка хранимых процедур

Следующим шагом мы создадим хранимые процедуры для нашей базы данных. Для создания хранимой процедуры щелкните правой кнопкой мыши по каталогу

хранимых процедур в нашей базе данных в **Server Explorer** и выберите пункт **Add New Stored Procedure** (Добавить новую хранимую процедуру). В редакторе **SQL Editor** откроется шаблон для хранимой процедуры. Редактор **SQL Editor** аналогичен редактору кода Visual Studio. Несмотря на то, что в нем нет технологии IntelliSense, он поддерживает подсветку синтаксиса, точки прерывания, а также обычные функции редактирования SQL-кода.

На рис. 6.9 показан шаблон для создания хранимой процедуры в окне редактора **SQL Editor**.

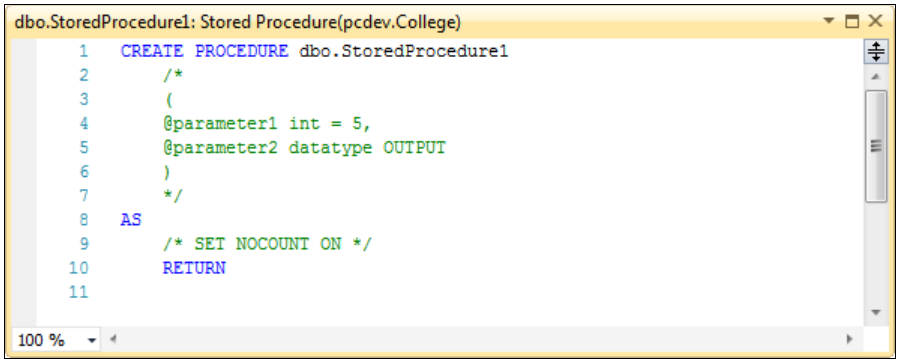


Рис. 6.9. Начало создания новой хранимой процедуры

Примеры хранимых процедур, необходимых для выборки, вставки, модификации и удаления данных для сущности `Student`, приведены в листингах 6.3—6.6.

Листинг 6.3. Хранимая процедура `Student_Select`

```
CREATE PROCEDURE Student_Select (
   @studentId int = NULL,
   @facultyId int = NULL)
AS
BEGIN
   SET NOCOUNT ON
   SELECT
      [Student].[StudentId],
      [Student].[FirstName],
      [Student].[LastName],
      [Faculty].[FacultyId],
      [Faculty].[FacultyTitle],
      [Student].[DateFrom],
      [Student].[DateTo]
    FROM [Student]
    LEFT OUTER JOIN [Faculty] ON [Faculty].[FacultyId]=[Student].[FacultyId]
    WHERE
       (@studentId IS NULL OR [Student].[StudentId] = @studentId) AND
       (@facultyId IS NULL OR [Faculty].[FacultyId] = @facultyId)
END
```

Листинг 6.4. Хранимая процедура `Student_Insert`

```
CREATE PROCEDURE Student_Insert (
      @firstName nvarchar(50),
      @lastName nvarchar(50),
      @facultyId int,
      @dateFrom date,
      @dateTo date = NULL)
AS
BEGIN
   SET NOCOUNT ON
   INSERT INTO [Student] (
      [FirstName],
      [LastName],
      [FacultyId],
      [DateFrom],
      [DateTo])
   VALUES (
      @firstName,
      @lastName,
      @facultyId,
      @dateFrom,
      @dateTo)
   RETURN @@ERROR
END
```

Листинг 6.5. Хранимая процедура `Student_Update`

```
CREATE PROCEDURE Student_Update (
      @studentId int,
      @firstName nvarchar(50),
      @lastName nvarchar(50),
      @facultyId int,
      @dateFrom date,
      @dateTo date = NULL)
AS
BEGIN
   SET NOCOUNT ON
   UPDATE [Student]
   SET
      [FirstName] = @firstName,
      [LastName] = @lastName,
      [FacultyId] = @facultyId,
      [DateFrom] = @dateFrom,
      [DateTo] = @dateTo
```

```
   WHERE [StudentId] = @studentId
   RETURN @@ERROR
END
```

Листинг 6.6. Хранимая процедура `Student_Delete`

```
CREATE PROCEDURE Student_Delete (@studentId int = NULL)
AS
BEGIN
   SET NOCOUNT ON
   DELETE FROM [Student]
   WHERE [StudentId] = @studentId
   RETURN @@ERROR
END
```

Для остальных сущностей хранимые процедуры пишутся аналогично приведенным для сущности `Student`. Для экономии места в книге они здесь не приводятся, вы их можете найти на диске с примерами: Database\CollegeScripts\CreateStoredProcedures.sql.

При написании кода процедуры окно редактора будет выделять и заключать в рамку синего цвета определенные части процедуры. Эти части в рамке представляют собой операторы SQL, которые можно редактировать при помощи **Query Designer**. Таким образом, редактор указывает, что он распознал в процедуре оператор SQL. Пример выделения блока показан на рис. 6.10.

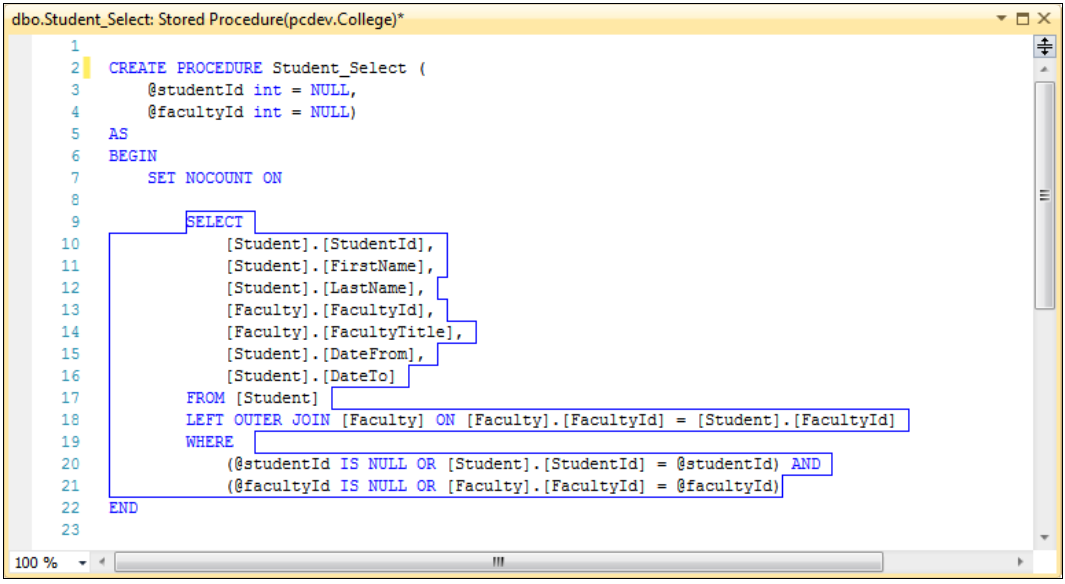


Рис. 6.10. Операторы SQL в хранимой процедуре

Операторы SQL в хранимой процедуре можно также создавать при помощи дизайнера **Query Builder**. Если вы щелкнете правой кнопкой мыши внутри рамки и в контекстном меню выберете пункт **Design SQL Block**, то откроется дизайнер **Query Builder**, окно которого изображено на рис. 6.11.

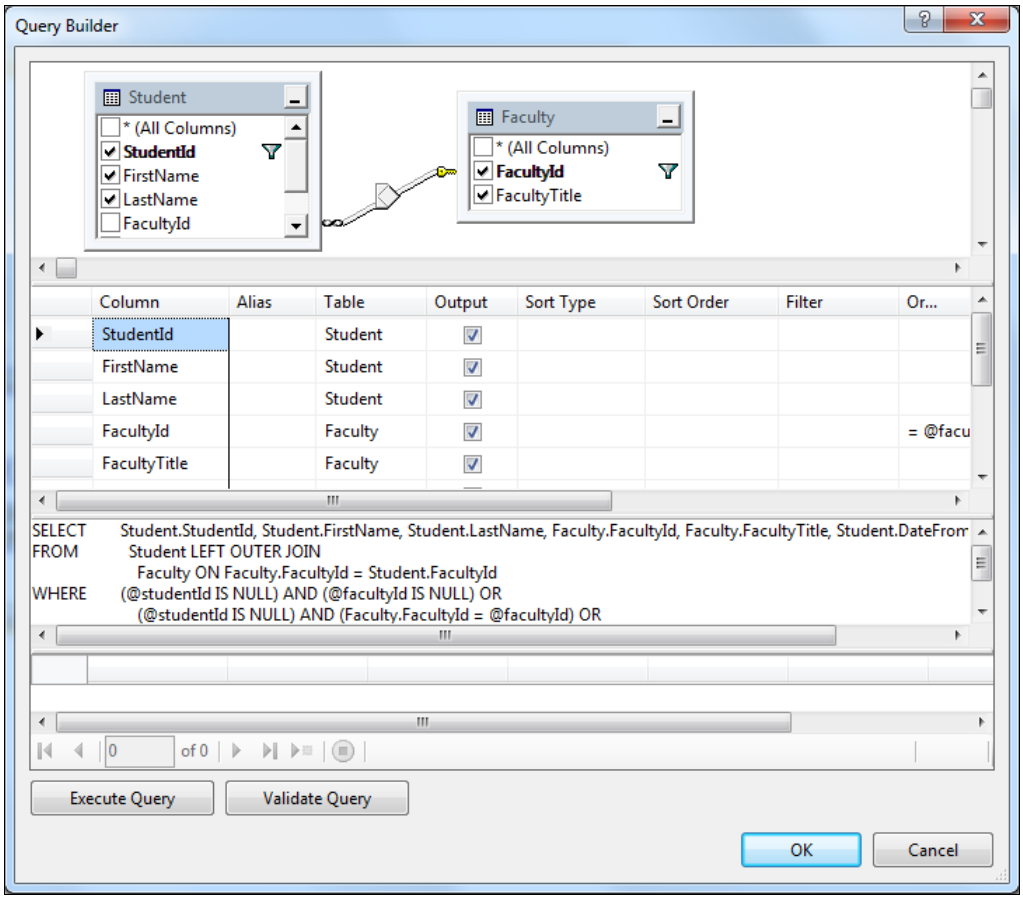


Рис. 6.11. Конструирование запроса для хранимой процедуры

Вы можете создавать SQL-операторы, перетаскивая мышью объекты базы данных в дизайнер **Query Builder**. Нажатие **OK** в диалоговом окне дизайнера сохранит запрос в хранимой процедуре (при этом обновится код в окне редактора).

Query Builder можно также использовать для вставки новых блоков SQL в хранимую процедуру (а не только для редактирования уже существующих операторов SQL). Для этого в окне **SQL Editor** надо щелкнуть правой кнопкой мыши по той строке процедуры, в которой вы хотите разместить новый запрос. В появившемся меню надо выбрать пункт **Insert SQL**. Теперь для создания соответствующего SQL-кода можно использовать **Query Designer**. После того как нажатием кнопки **OK** вы закроете диалоговое окно, новый SQL-оператор будет вставлен в процедуру и может быть сохранен в базе данных.

Отладка хранимых процедур

Когда хранимая процедура открыта в окне **SQL Editor**, вы можете настроить в ней точку останова. После этого вы можете щелкнуть правой кнопкой мыши по названию хранимой процедуры в дереве **Server Explorer** и выбрать пункт **Step Into Stored Procedure** (рис. 6.12).

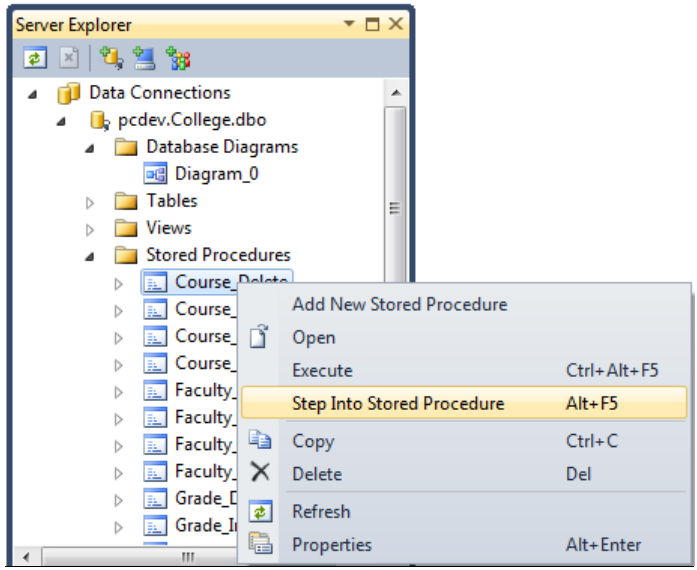


Рис. 6.12. Отладка хранимой процедуры

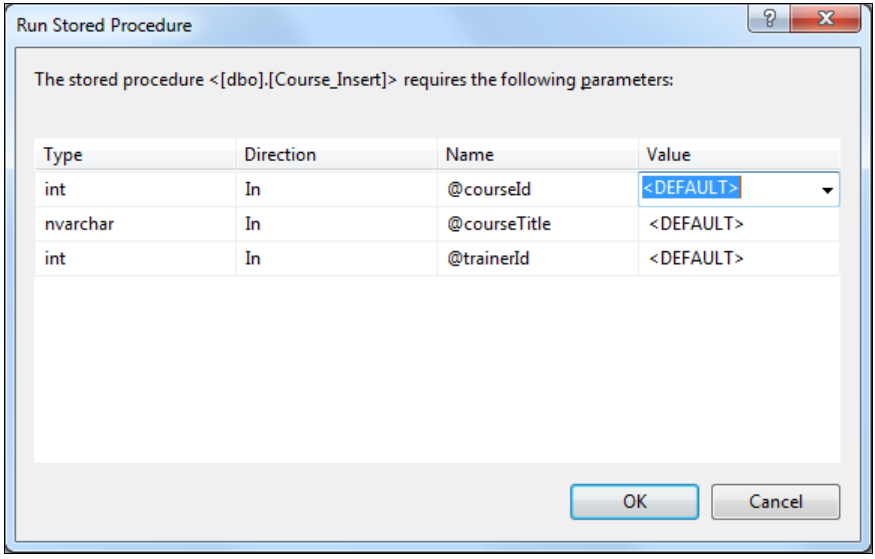


Рис. 6.13. Ввод значений параметров в **SQL Debugger**

Отладчик SQL умеет работать также и с параметрами. Если хранимая процедура использует параметры, то отладчик покажет диалоговое окно для ввода значений параметров (рис. 6.13). Вы можете быстро пробежать по списку параметров и задать соответствующие значения.

После нажатия кнопки **OK** хранимая процедура будет выполнена. Если вы настроили точку останова, то выполнение на ней остановится (для указания текущей выполняющейся строки в редакторе используется желтая стрелка — точно так же, как и в окне редактора кода). При остановке выполнения хранимой процедуры вы можете использовать уже известные вам окна **Locals** и **Watch** для отладки кода процедуры, описанные в *главе 4*, посвященной отладке приложений.

Шаблоны проектов баз данных

В проектах баз данных используется все та же система шаблонов проектов и создания нового проекта, что и в других типах проектов Visual Studio.

Поскольку проект базы данных является всего лишь представлением реальной базы данных, то изменения в проекте можно делать без боязни повредить реальную базу данных. И поскольку эти изменения производятся с определенным набором файлов (которые могут находиться в той же системе управления исходными кодами, что и остальные файлы VSTS), то разработчик базы данных может участвовать в том же самом процессе отслеживания элементарных работ, что и остальные члены проектной команды.

Visual Studio 2010 помогает выполнять создание и развертывание новой базы данных в системе управления версиями. Схема проекта базы данных создается путем импорта схемы из базы данных — шаблоны SQL Server 2005 Wizard, SQL Server 2008 Wizard, или создания скриптов для развертывания новой базы данных. Visual Studio 2010 удобно использовать для изменения автономного представления базы данных и последующего развертывания этих изменений.

Проект базы данных это, по существу, автономная версия базы данных. Он отображает базу данных при помощи набора SQL-файлов, которые содержат схему и определения объектов для таблиц, индексов, а также хранимых процедур. С помощью проектов баз данных можно создавать новые базы данных а также обновлять имеющиеся базы данных. Проекты баз данных позволяют использовать при разработке баз данных методы управления версиями и проектами, как это происходит при работе с управляемым или машинным кодом.

В диалоговом окне **Add New Project** проекты SQL Server находятся в категории **Database**. Давайте выберем шаблон мастера SQL Server 2008 Wizard и пройдем по всем экранам мастера, для того чтобы выполнять анализ и проверку объектов в проекте базы данных (рис. 6.14).

После нажатия кнопки **OK** откроется начальное окно мастера создания базы данных **New Database Project Wizard**, показанное на рис. 6.15.

На следующей странице мастера **Project Properties** (Свойства проекта) необходимо указать, как мы хотим организовать проект и какую схему использовать.

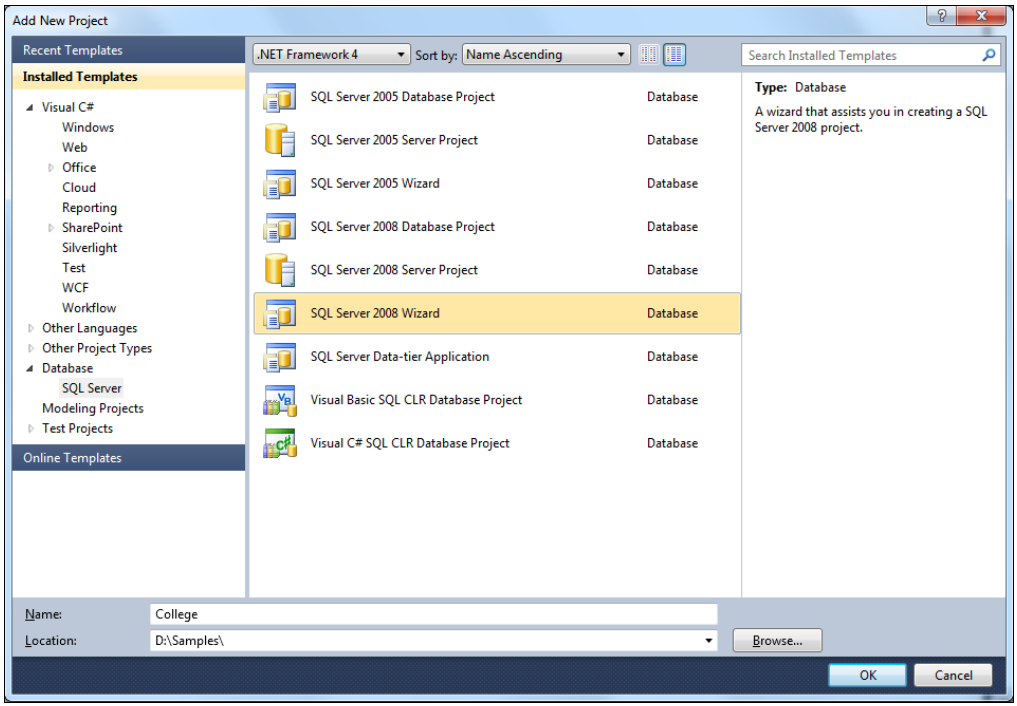


Рис. 6.14. Проекты базы данных

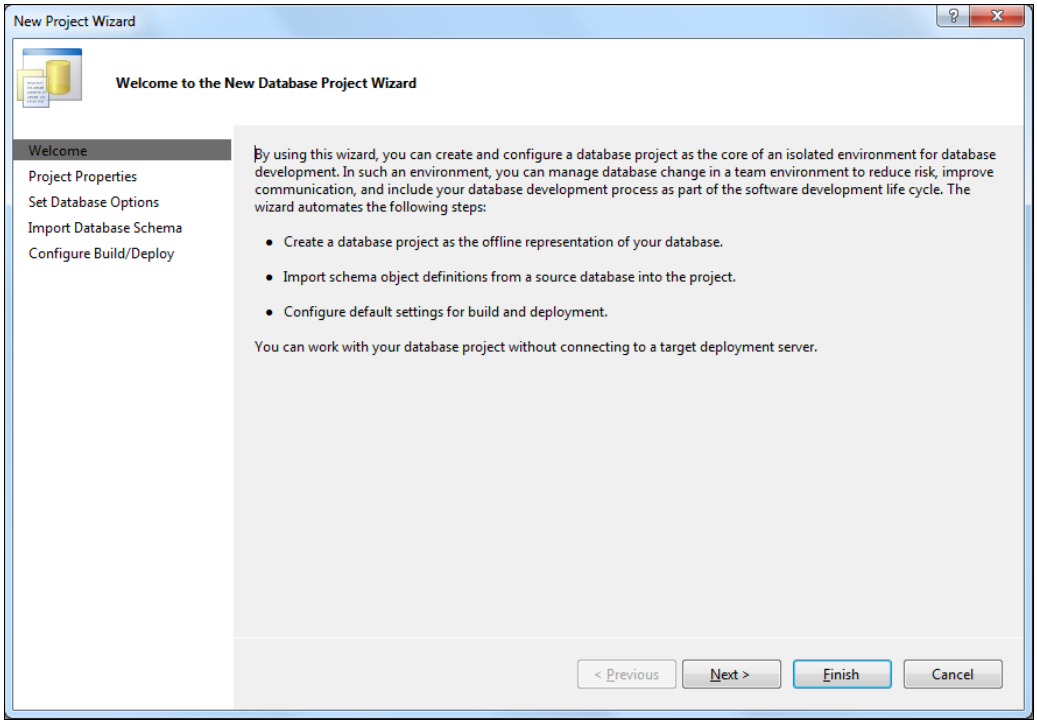


Рис. 6.15. Начальная страница мастера создания проекта базы данных

Для организации проекта есть два варианта:

- **By schema** (По схеме);
- **By type of object** (По типу объекта).

При выборе варианта **By type of object** интегрированная среда Visual Studio создаст каталог объектов схемы с подкаталогами для таких объектов вашей базы данных, как таблицы и хранимые процедуры. Опция **By schema** сгруппирует ваши объекты проекта по тому типу схемы, к которому они принадлежат (рис. 6.16).

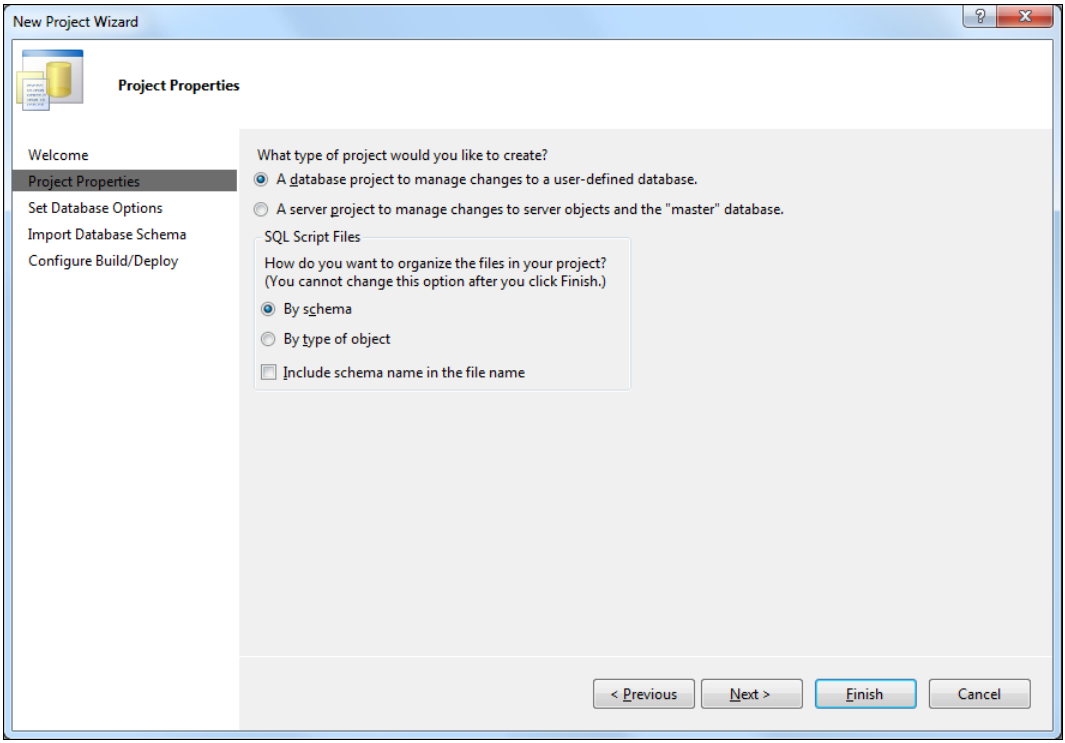


Рис. 6.16. Окно мастера со страницей выбора свойств проекта

Следующая страница мастера **Set Database Options** (Установка настроек базы данных) указывает используемый тип сопоставления, а также различные настройки сопоставления и хранения, которые показаны на рис. 6.17. Можно оставить настройки по умолчанию. Фактически на следующей странице мастера у вас будет опция импорта этих настроек из рабочей базы данных (в дополнение к ее схеме).

Посредством реинжиниринга существующей базы данных в составляющие ее объекты среда Visual Studio дает возможность нам создавать копии базы данных, что в свою очередь позволяет разработчикам работать в своей собственной "песочнице" и не беспокоиться о повреждении производственного хранилища данных. На рис. 6.18 показана страница мастера для импорта схемы базы данных **Import Database Schema**.

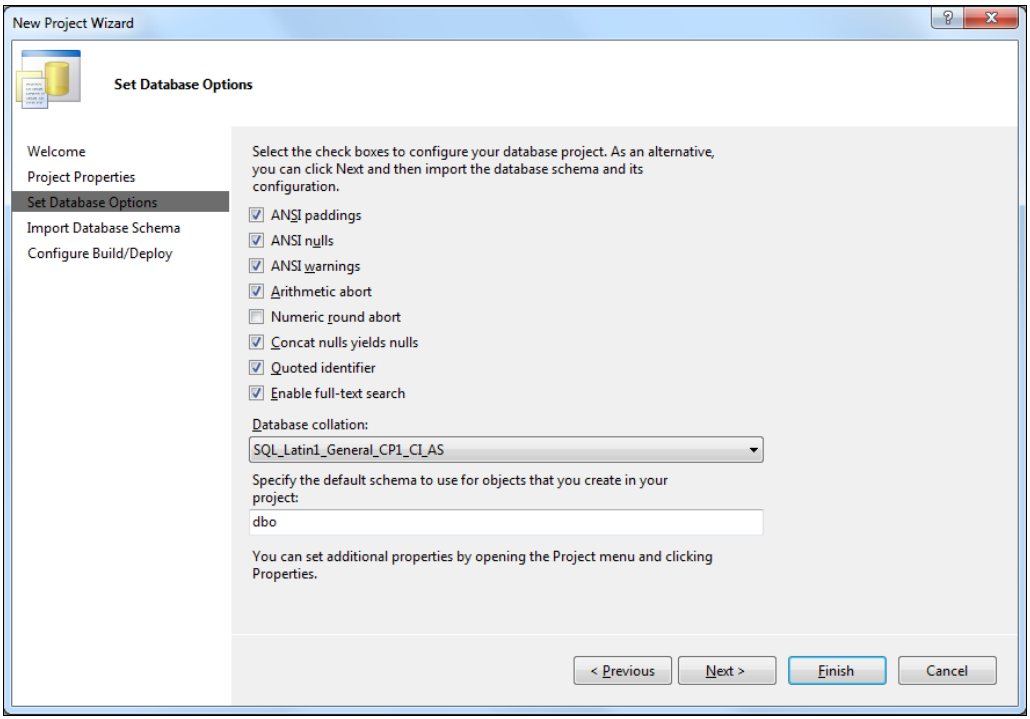


Рис. 6.17. Установка настроек базы данных

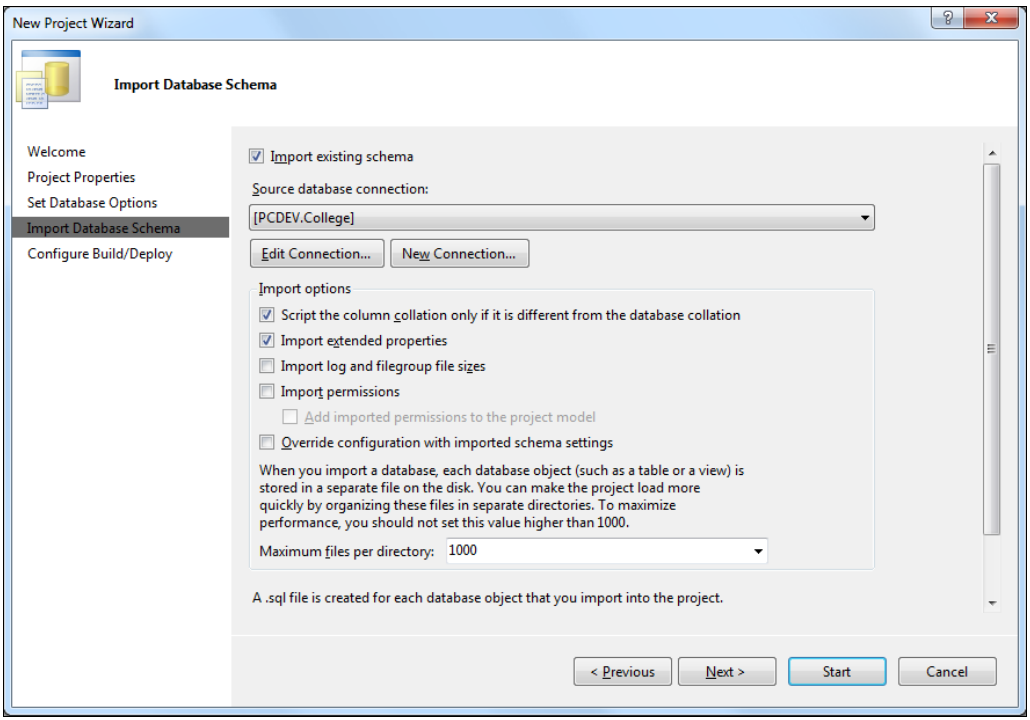


Рис. 6.18. Импорт схемы базы данных

Нажав кнопку **New Connection**, с помощью появившегося диалогового окна **Connection Properties** создайте подключение к нашей базе данных, а затем переходите к следующей странице мастера.

Страница мастера **Configure Build and Deploy** (Конфигурация сборки и развертывания) проекта базы данных определяет, как Visual Studio будет собирать и развертывать проект базы данных. Особый интерес здесь представляют настройки целевой базы данных. Существуют два основных сценария развертывания:

- Развернуть проект в новую базу данных.
- Развернуть проект в уже существующую базу данных.

Это позволяет либо создать новый экземпляр базы данных на основе описанной в проекте схемы, либо обновить существующую базу данных для того, чтобы использовалась описанная в проекте схема. Страница мастера **Configure Build and Deploy** представлена на рис. 6.19.

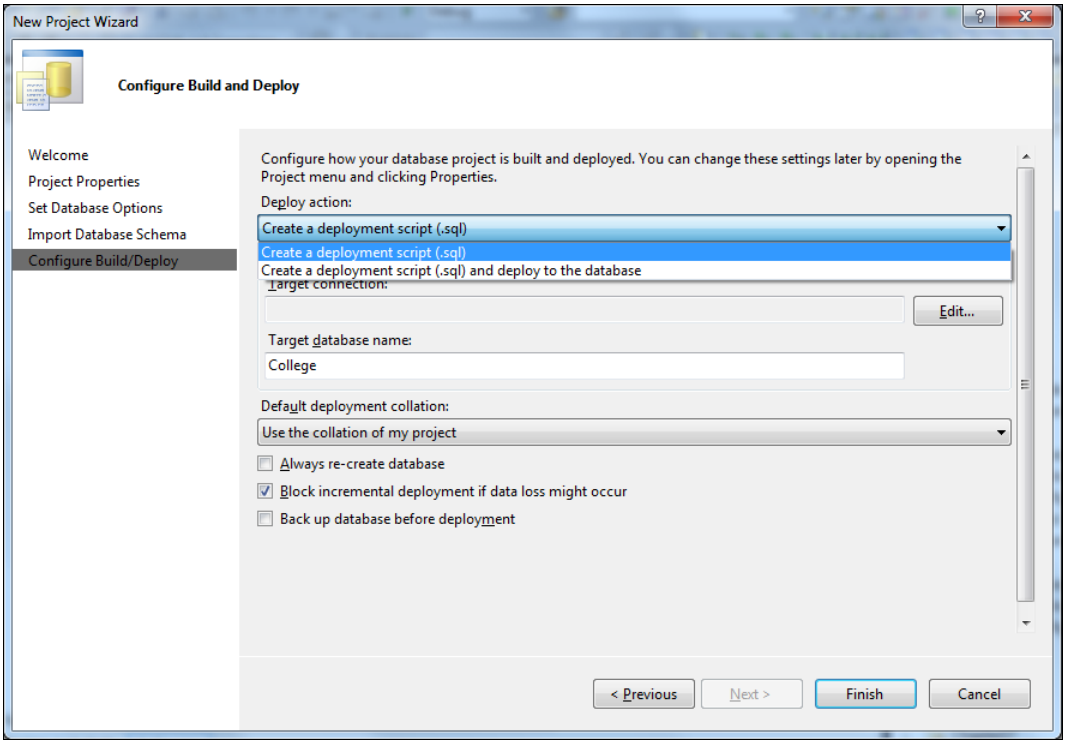


Рис. 6.19. Страница мастера **Configure Build and Deploy**

Мы задаем целевую базу данных посредством указания подключения, имени базы данных, а также местоположения файлов базы данных на сервере. Последнее поле принимается по умолчанию равным типу сопоставления исходной базы данных (которое Visual Studio определила по указанной на предыдущем экране информации).

Финальная страница мастера **Summary** показана на рис. 6.20. На данной стадии Visual Studio уже имеет всю информацию для создания исходного проекта. Нажатие кнопки **Finish** запустит генерирование проекта в Visual Studio.

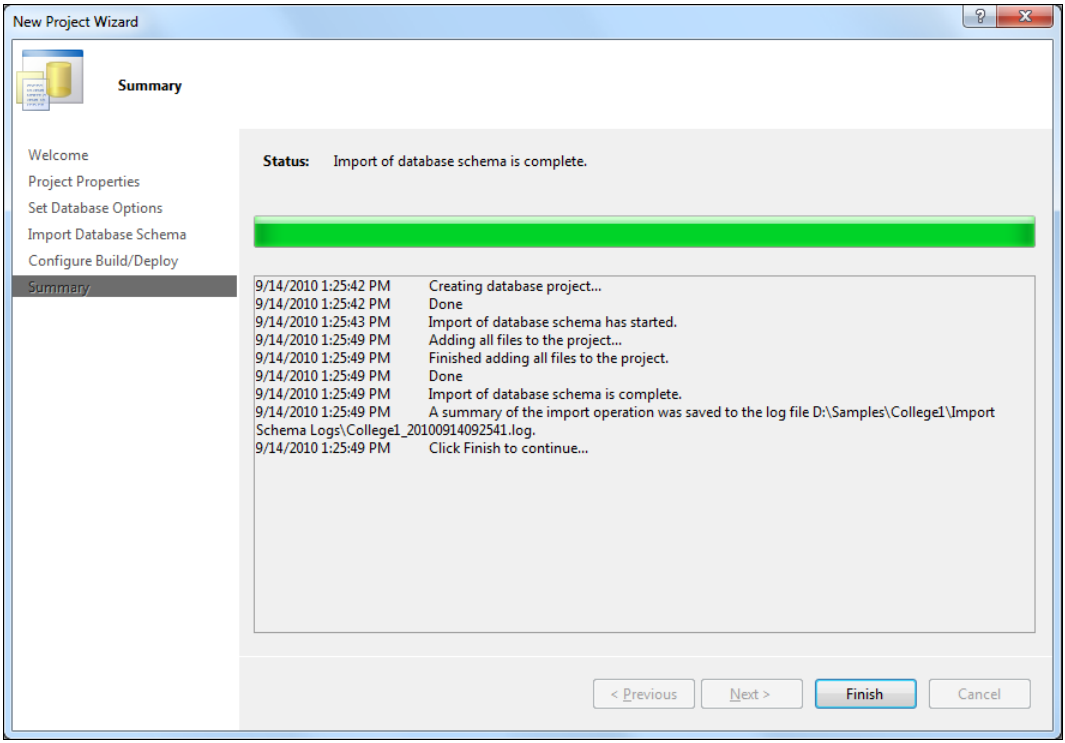


Рис. 6.20. Импорт схемы базы данных `College`

После создания проект по умолчанию будет иметь структуру каталогов, показанную на рис. 6.21.

Как видно из структуры папок, проект базы данных имеет определенные каталоги для хранения скриптов, запросов и ссылок на базы данных. Например, объекты схемы базы данных находятся в корневом каталоге **Schema Objects**, внутри которого они организованы по своим типам. Каждый объект базы данных — это может быть таблица, хранимая процедура, индекс, ключ или ограничение — представлен одним SQL-файлом. Кроме файлов схемы, у нас есть каталоги для хранения планов создания данных (подробнее о них далее в этой главе), а также для скриптов пред- и постразвертывания.

Скрипты, которые размещены в каталогах Pre-Deployment и Post-Deployment, будут выполнены непосредственно перед развертыванием базы данных и сразу после него. Скрипты Pre-Deployment и Post-Deployment полезны в том случае, когда вы развертываете базу данных в тестовой среде. Вы можете использовать скрипты Pre-Deployment для инициализации развертывания базы данных, а скрипты Post-Deployment для очистки после завершения развертывания базы данных.

Скрипты SQL в проекте базы данных могут создавать различные объекты баз данных, например таблицы, представления, триггеры, хранимые процедуры и т. д. Запросы, которые были созданы при помощи визуального конструктора запросов **Query/View Designer**, могут быть также непосредственно сохранены в проекте базы данных.

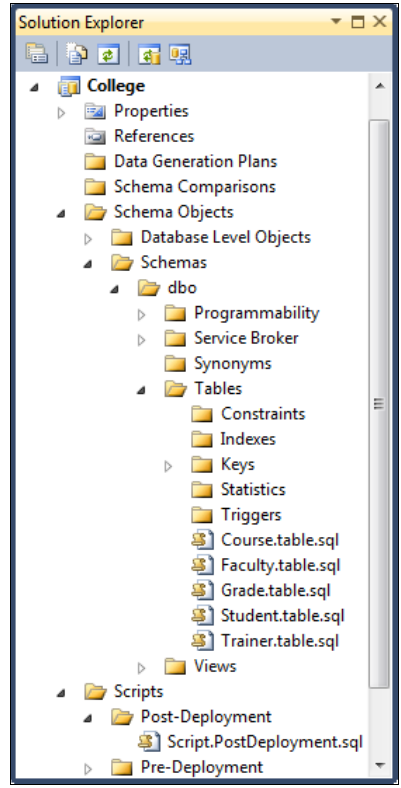


Рис. 6.21. Структура проекта базы данных `College`

Автоматическое генерирование скриптов

Существуют два основных типа автоматически генерируемых скриптов: скрипты создания и скрипты изменений. *Скрипты создания* используются для создания нового объекта базы данных: таблицы, хранимой процедуры и т. п. *Скрипты изменений* применяются для обновления существующего объекта базы данных.

Концепция скриптов изменений аналогична, но есть два отличия: скрипты изменений могут автоматически генерироваться только для таблиц, и скрипт будет учитывать только изменения в таблице (саму таблицу создавать не будет). Дизайнер **Table Designer** обеспечивает генерирование скриптов изменений. После выполнения изменений таблицы внутри визуального дизайнера выберите **Table Designer | Generate Change Script**. Скрипт будет создан и помещен в каталог **Change Scripts** текущего проекта.

Поскольку Visual Studio генерирует скрипт изменений при помощи отслеживания ваших действий по редактированию и сравнения их с существующей в базе данных структурой таблицы, то вы можете сгенерировать скрипт изменений только до того, как ваши изменения будут сохранены в базе данных.

Выполнение скриптов

После создания скрипта его можно выполнить на целевой базе данных. Для этого запустите скрипт в **Solution Explorer** при помощи щелчка правой кнопкой мыши по файлу скрипта. В появляющемся меню имеются два варианта запуска скрипта: команда **Run** и команда **Run On**. Команда **Run** прогонит скрипт по базе данных, указанной в ссылке, на базу данных по умолчанию. Команда **Run On** позволяет вам вручную указать в списке ссылок на базы данных ту базу данных, которая является целью вашего скрипта.

Представление схемы

Кроме уже знакомого представления **Solution Explorer** проекты баз данных имеют также и окно **Schema View** (рис. 6.22). Это окно доступно из главного меню **View | Database Schema View**.

Поскольку окно **Schema View** сосредоточено исключительно на объектах баз данных, то вы не найдете здесь таких объектов, как планы создания данных или скрипты пред- и постразвертывания. Окно **Schema View** имеет также замечательную функцию визуальной индикации наличия синтаксических ошибок в объектах баз данных.

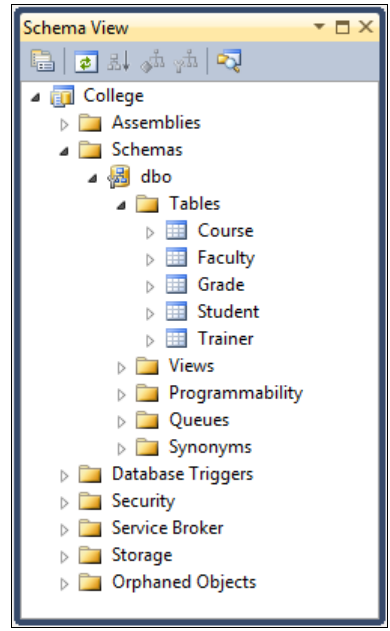


Рис. 6.22. Окно **Schema View** проекта базы данных

Создание тестовых данных

Еще одним ключевым действием жизненного цикла разработки баз данных является создание тестовых данных. Само собой разумеется, что невозможно протестировать и уровень приложения, и уровень базы данных (приложения для работы с базой данных) при отсутствии данных в базе. Это общая для проектов баз данных проблема. В идеальном случае надо было бы тестировать на реальных данных, но с использованием таких наборов данных связано множество проблем. Реальные

данные часто содержат конфиденциальную информацию, которая не может находиться в открытом доступе.

Для создания тестовых данных Visual Studio имеет генератор данных. Он работает как дополнение к инструменту импорта схемы: если мы обследуем производственную базу данных и выполним реинжиниринг ее схемы, то сможем сделать то же самое и с данными.

Генератор данных использует профиль данных производственной базы для создания своего собственного алгоритма генерирования случайных (но релевантных) данных в целевой базе данных.

План создания данных — это объект, который мы можем добавить в наш проект базы данных для создания тестовых наборов данных. Эти планы будут видны в окне **Solution Explorer** в каталоге **Data Generation Plan**. Для добавления плана в ваш проект щелкните правой кнопкой мыши по названию проекта, выберите пункт **Add New Item**, а затем выберите шаблон **Data Generation Plan**, как показано на рис. 6.23.

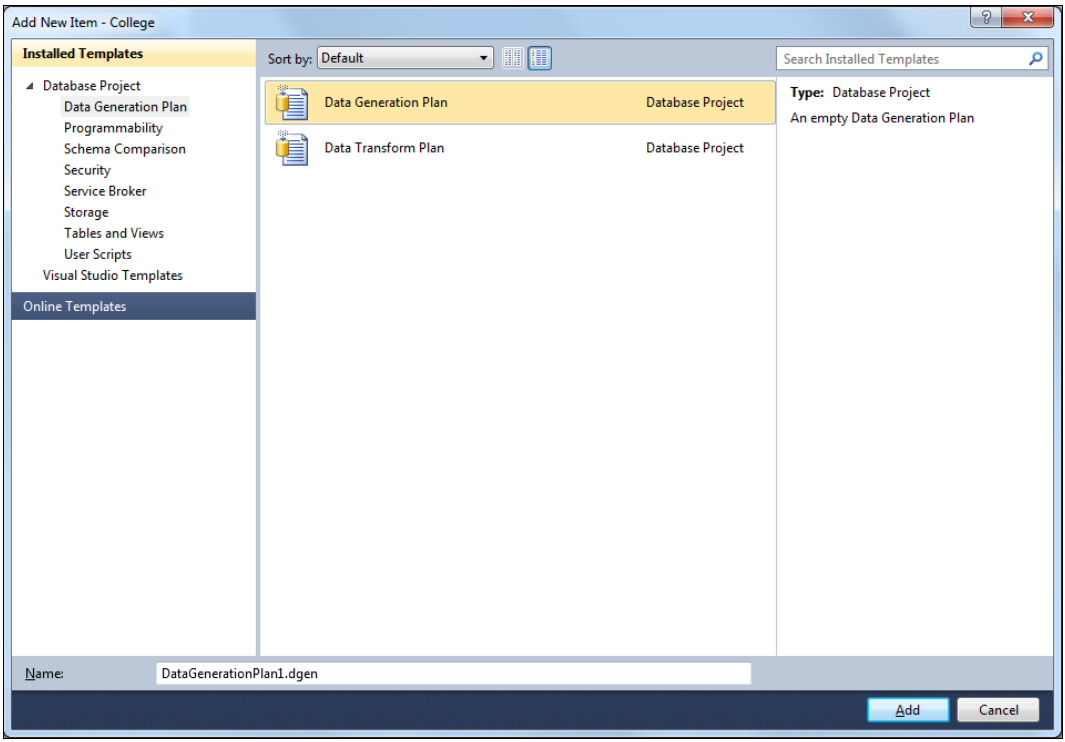


Рис. 6.23. Добавление плана создания данных

После того как план будет добавлен в ваш проект, он обследует объекты схемы проекта, а затем загрузит их в окно визуального дизайнера **Data Generation Plan Designer** (рис. 6.24).

В окне визуального дизайнера мы управляем всеми аспектами плана создания данных. Верхняя панель дизайнера показывает нам все таблицы нашего проекта.

Мы можем включить любой столбец в план создания данных, помечая флажок рядом с названием объекта.

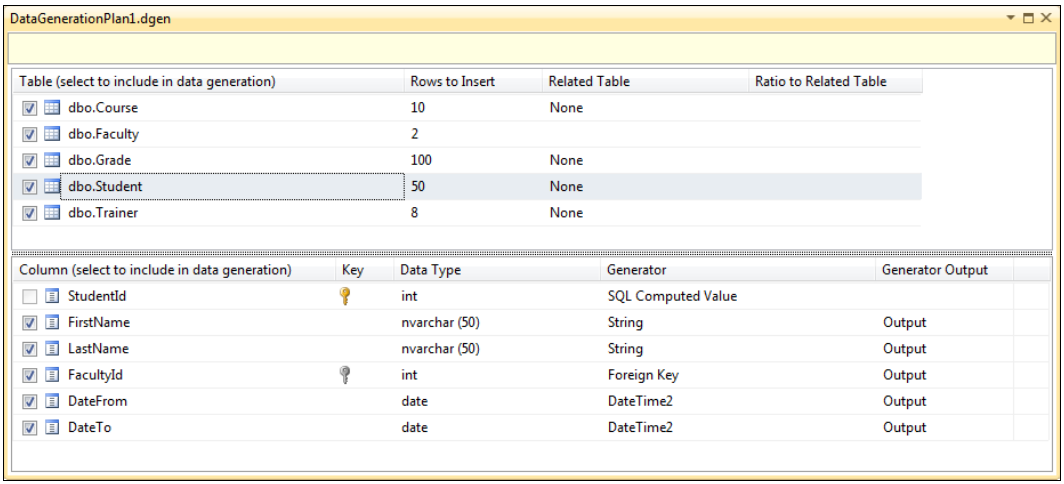


Рис. 6.24. Визуальный дизайнер **Data Generation Plan Designer**

Кроме названий таблиц мы видим также и число, представляющее количество строк, которые будут вставлены в таблицу. По умолчанию план вставит 50 строк в каждую таблицу. Это значение можно изменить на любое число строк (его необходимо ввести в ячейку таблицы).

Список имеющихся таблиц будет заполнен в соответствии с внешними ключами и связями, описанными в схеме базы данных. После выбора связанной таблицы мы можем указать соотношение, и это число автоматически настроит количество строк, которые будут вставлены в таблицу на стороне "многие".

Настройка генераторов данных

Нижняя панель визуального дизайнера данных дает нам предварительный просмотр данных для любой выбранной в верхней панели таблицы. Например, после выбора таблицы `Student` мы сможем увидеть строки, которые автоматически сгенерированы для вставки в эту таблицу генератором данных (рис. 6.25).

Каждый генератор выдает данные на базе случайного начального значения и соответствует целевой схеме. В нашем примере с таблицей `Student` мы видим, что для столбца `StudentId` не было сгенерировано значений, поскольку в схеме базы данных этот столбец описан как идентифицирующий; значения для этого столбца будут предоставлены самой базой данных. В данных примера вы увидите числа там, где должны быть числа; строки там, где должны быть строки; и временные метки там, где должны быть данные даты/времени.

Но данные не безукоризненные. Например, мы видим, что столбцы имен содержат данные, которые имеют вид строк, но не похожи на настоящие имена. Для того

чтобы данные стали более реалистичными, нам необходимо настроить некоторые свойства отдельных генераторов.

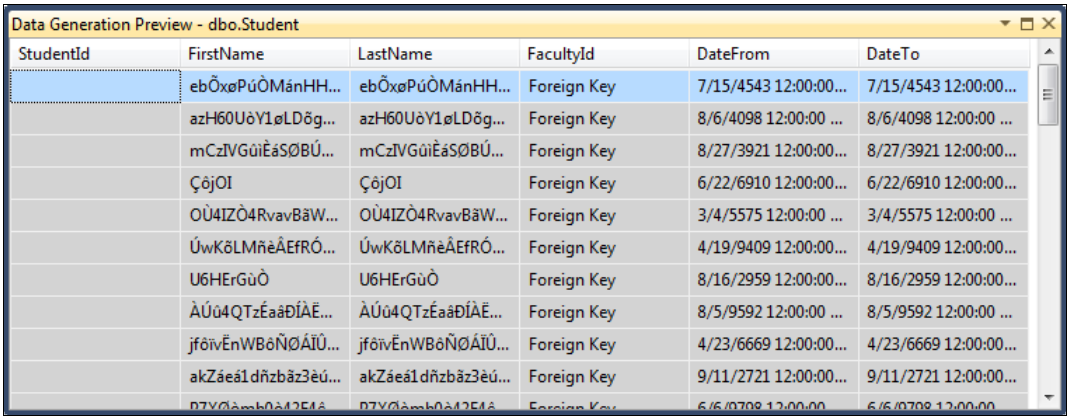


Рис. 6.25. Предварительный просмотр данных

Visual Studio имеет генераторы для всех основных типов данных. Кроме того, имеются генераторы и для более сложных типов:

- внешние ключи;
- регулярные выражения;
- данные на основе запросов.

Изменение свойств генератора

Каждый тип генератора имеет собственные уникальные свойства, которые можно настраивать. Для числовых генераторов мы можем указать минимальное и максимальное значения. Помимо управления такими параметрами, как минимальное и максимальное значения (или начальное значение), мы можем также указать распределение для строковых генераторов (при помощи настройки минимальной и максимальной длины генерируемых строк).

Как уже упоминалось, у нас есть возможность реализовать более сложные генераторы. Давайте в качестве примера рассмотрим столбец `FirstName` таблицы `Student`. Поскольку базовый генератор этого делать не умеет, то мы можем вместо него использовать генератор регулярных выражений.

Для изменения присвоенного столбцу генератора необходимо прокрутить окно **Column Details** до столбца **Generator**. Щелчок по столбцу покажет раскрывающийся список со всеми поддерживаемыми типами генераторов. В нашем случае необходимо выбрать **Regular Expression** (рис. 6.26).

После замены генератора мы можем использовать окно свойств для ввода нашего выражения, которое будет генерировать правдоподобные обращения (рис. 6.27).

После этого, открыв панель **Data Generation Preview**, можно увидеть, что данные уже обновлены в соответствии с настройками генератора.

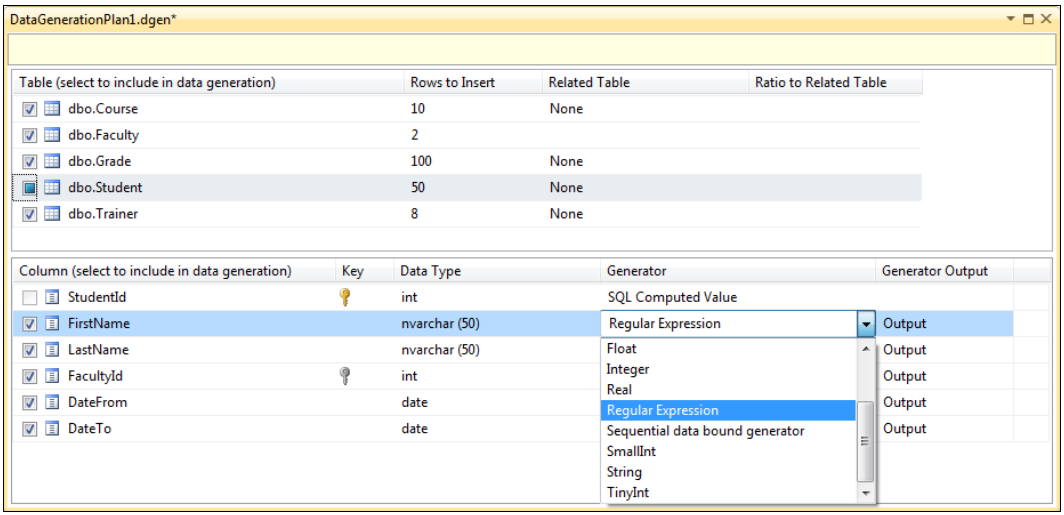


Рис. 6.26. Выбор типа генератора данных

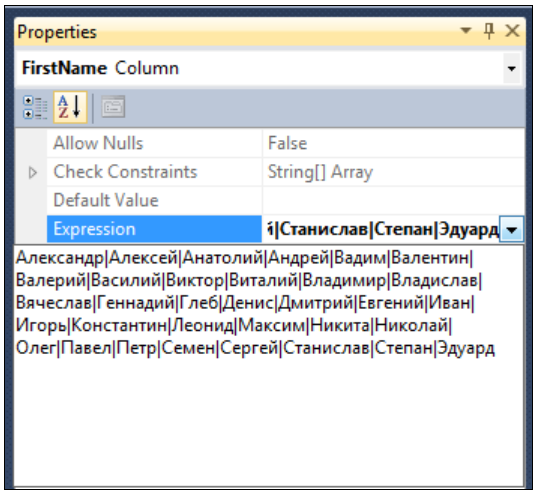
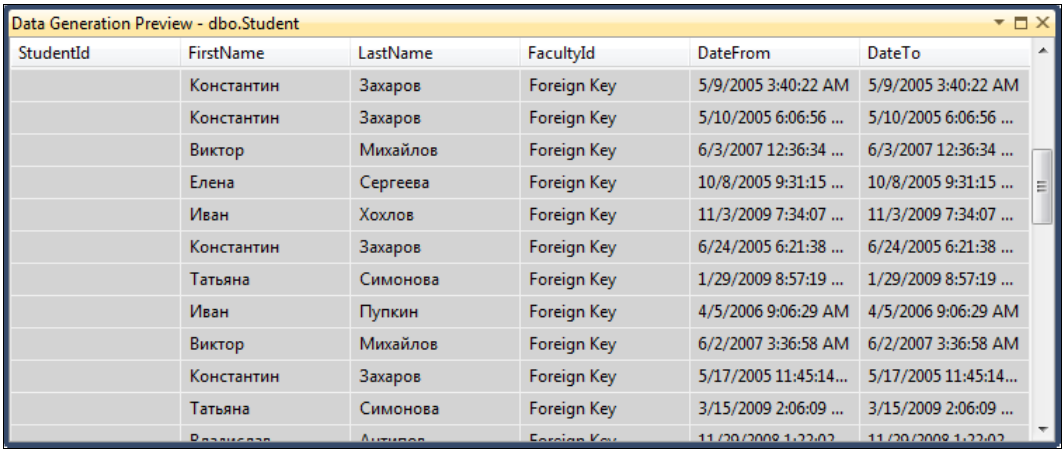


Рис. 6.27. Изменение присвоенного генератора

Создание данных

После того как мы настроили план создания данных так, как того хотели, последним шагом необходимо выполнить этот план. Самый простой способ для этого — щелкнуть по значку **Generate Data** в панели инструментов **Data Generator** или в главном меню выбрать опцию **Data | Data Generator | Generate Data**. Сначала вам будет необходимо указать то подключение к базе данных, которое будет использоваться (которое, в свою очередь, уже определяет и целевую базу данных). Здесь вы получите запрос, хотите ли вы удалять уже существующие в таблице строки перед тем, как создавать новые данные. После получения ответов на эти во-

просы начнется создание данных. Результатом будет создание базы данных с информационным наполнением, вполне соответствующим реальным данным, для хранения которых эта база и разрабатывается (рис. 6.28).


Data Generation Preview - dbo.Student

StudentId	FirstName	LastName	FacultyId	DateFrom	DateTo
	Константин	Захаров	Foreign Key	5/9/2005 3:40:22 AM	5/9/2005 3:40:22 AM
	Константин	Захаров	Foreign Key	5/10/2005 6:06:56 ...	5/10/2005 6:06:56 ...
	Виктор	Михайлов	Foreign Key	6/3/2007 12:36:34 ...	6/3/2007 12:36:34 ...
	Елена	Сергеева	Foreign Key	10/8/2005 9:31:15 ...	10/8/2005 9:31:15 ...
	Иван	Хохлов	Foreign Key	11/3/2009 7:34:07 ...	11/3/2009 7:34:07 ...
	Константин	Захаров	Foreign Key	6/24/2005 6:21:38 ...	6/24/2005 6:21:38 ...
	Татьяна	Симонова	Foreign Key	1/29/2009 8:57:19 ...	1/29/2009 8:57:19 ...
	Иван	Пупкин	Foreign Key	4/5/2006 9:06:29 AM	4/5/2006 9:06:29 AM
	Виктор	Михайлов	Foreign Key	6/2/2007 3:36:58 AM	6/2/2007 3:36:58 AM
	Константин	Захаров	Foreign Key	5/17/2005 11:45:14...	5/17/2005 11:45:14...
	Татьяна	Симонова	Foreign Key	3/15/2009 2:06:09 ...	3/15/2009 2:06:09 ...

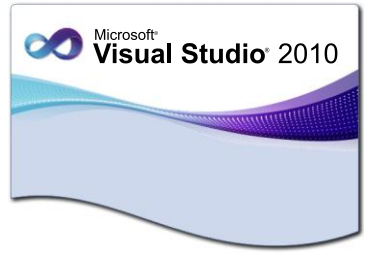
Рис. 6.28. Создание тестовых данных

При желании можно немного подредактировать сгенерированные средой Visual Studio данные, например в столбцах `DateFrom` и `DateTo`, для более правильного соответствия реальным данным.

Резюме

Среда Visual Studio предоставляет эффективные инструменты для проектирования баз данных и, что самое важное, позволяет наполнять базы тестовыми данными. Познакомив вас со всеми этими замечательными встроенными инструментами, мы показали вам путь повышения эффективности использования средств Visual Studio для разработки баз данных.

ГЛАВА 7



Технология доступа к данным ADO.NET

ADO.NET — это технология, позволяющая приложению взаимодействовать с данными. Большинству приложений требуется для работы тот или иной тип доступа к данным. Клиентским приложениям необходимо взаимодействовать с централизованной базой данных, хранилищами данных в формате XML или локальными базами данных, работающими на клиентских машинах. Технология ADO.NET предоставляет простые в применении средства доступа к данным.

ADO.NET предоставляет доступ к таким источникам данных, как SQL Server и XML, а также к источникам данных, предоставляемым посредством OLE DB и ODBC. Пользовательские приложения, использующие общие данные, могут использовать ADO.NET для соединения с этими источниками данных и для получения, обработки и обновления имеющихся в них данных.

Архитектура данных ADO.NET

В Visual Studio .NET имеется множество встроенных мастеров и дизайнеров, которые помогут быстро и эффективно создать архитектуру доступа к данным во время разработки и оснастить приложение надежным механизмом доступа к данным, затратив минимум усилий на написание кода. Наряду с этим все возможности объектной модели ADO.NET доступны программно, что позволяет реализовать нестандартные функции или создавать приложения, ориентированные на нужды пользователя.

Доступ к данным в ADO.NET основан на использовании двух компонентов:

- *провайдера данных* (Data Provider), выполняющего функции посредника при взаимодействии программы и баз данных;
- *набора данных* — объекта `DataSet`, в котором данные хранятся на локальном компьютере.

Провайдеры данных

Связь с базой данных создается и поддерживается при помощи провайдера данных. Провайдерами данных .NET Framework являются компоненты, которые специально сконструированы для обработки данных и быстрого, однопроходного доступа к данным только для чтения. В действительности *провайдер* — это набор классов, взаимосвязанных компонентов, обеспечивающих эффективный высокопроизводительный доступ к данным (рис. 7.1). Любой провайдер данных состоит из четырех компонентов:

- `Connection` — обеспечивает подключение к источнику данных;
- `Command` — применяется для управления источником данных; позволяет исполнять команды, не возвращающие данных, например `INSERT`, `UPDATE` и `DELETE`, либо команды, возвращающие объект `DataReader` (такие как `SELECT`). Объект `Command` позволяет обращаться к командам базы данных для возврата данных, изменения данных, выполнения хранимых процедур и передачи или получения сведений о параметрах;
- `DataReader` — предоставляет доступный только для однонаправленного чтения набор записей, подключенный к источнику данных. `DataReader` обеспечивает высокопроизводительный поток данных из источника данных;
- `DataAdapter` — заполняет отсоединенный объект `DataSet` или `DataTable` и обновляет его содержимое. `DataAdapter` предоставляет мост между объектом `DataSet` и источником данных. `DataAdapter` использует объекты `Command` для выполнения команд SQL на источнике данных, для загрузки `DataSet` с данными и согласования изменений данных, выполненных в `DataSet`, вновь с источником данных.

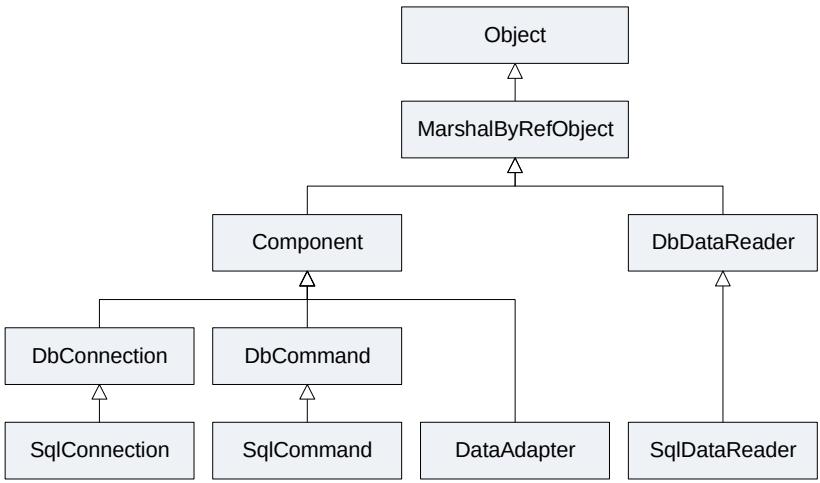


Рис. 7.1. Иерархия классов провайдеров данных ADO.NET

Объект `Connection` представляет соединение с базой данных. Visual Studio поддерживает два класса `Connection`: `SqlConnection`, предназначенный для подключе-

ния к SQL Server, и OleDbConnection, который применяется для подключения к самым разным БД. Все данные, необходимые для открытия канала связи с базой данных, хранятся в свойстве ConnectionString объекта Connection, этот объект также поддерживает ряд методов, позволяющих обрабатывать данные с применением транзакций. Объект Command также представлен двумя классами — SqlCommand и OleDbCommand. Он позволяет исполнять команды над базой данных, используя для обмена данными установленное соединение.

Примечание

В книге будет рассматриваться работа только с сервером баз данных Microsoft SQL Server.

При помощи объектов Command можно исполнять хранимые процедуры, команды SQL, а также операторы, возвращающие целые таблицы.

Объект SqlDataReader предоставляет поток с набором записей базы данных, доступный только для однонаправленного чтения. В отличие от других компонентов провайдера данных, создавать экземпляры DataReader напрямую не разрешается, его можно получить при помощи методов ExecuteReader объекта Command: метод SqlCommand.ExecuteReader() возвращает объект SqlDataReader. Поскольку в любой момент времени в памяти находится только одна строка, использование объекта SqlDataReader почти не снижает производительность системы, но требует монопольного доступа к открытому объекту Connection в течение времени жизни объекта SqlDataReader.

DataAdapter — это основной класс ADO.NET, обеспечивающий доступ к отсоединенным данным. В сущности, он выполняет функции посредника во взаимодействии между БД и объектом DataSet. При вызове метода Fill() объект DataAdapter заполняет DataTable или DataSet данными, полученными из базы данных. После обработки данных, загруженных в память, можно записать модифицированные данные в базу данных, вызвав метод Update() объекта DataAdapter.

При вызове метода Update() все измененные данные копируются из объекта DataSet в базу данных с исполнением соответствующей команды InsertCommand, DeleteCommand или UpdateCommand.

Организация доступа к данным

Доступ к данным в ADO.NET осуществляется в такой последовательности:

1. Объект Connection устанавливает между приложением и базой данных соединение, напрямую доступное объектам Command и DataAdapter.
2. Объект Command позволяет исполнять команды непосредственно над базой данных. Если исполненная команда возвращает несколько значений, Command открывает к ним доступ через объект DataReader.
3. Полученные результаты можно обрабатывать напрямую, используя код приложения, либо через объект DataSet, заполнив его при помощи объекта DataAdapter.

Для обновления базы данных также применяют объекты `Command` или `DataAdapter`.

Объект *DataSet*

Объект `DataSet` — это представление в памяти компьютера данных, изолированных от источника данных. Этот объект можно также рассматривать как локальную копию фрагмента БД. Объект `DataSet` хранится в памяти, его содержимым разрешено манипулировать и обновлять независимо от БД, играющей роль источника данных. При необходимости объект `DataSet` может служить шаблоном для обновления серверной БД.

Объект `DataSet` содержит набор объектов `DataTable` (этот набор может быть и пустым, т. е. не содержать ни одного объекта `DataTable`). Каждый объект `DataTable` представляет в памяти компьютера одну таблицу. Структура объекта `DataTable` определяется двумя наборами: `DataColumns`, куда входят все столбцы таблицы, которую представляет объект `DataTable`, и набором ограничений таблицы. Вместе эти два набора составляют схему таблицы. В `DataTable` также входит набор `DataRows`, где, собственно, и хранятся данные объекта `DataSet`.

Наконец, `DataSet` содержит набор `ExtendedProperties`, в котором хранятся дополнительные данные объекта `DataSet`.

Класс `DataSet` в ADO.NET специально сконструирован для доступа к данным независимо от источника данных. Поэтому он может быть использован со многими разными источниками данных, с XML-данными или для управления данными, локальными для приложения. `DataSet` содержит коллекцию одного или нескольких объектов `DataTable`, состоящих из строк и столбцов данных, а также первичный ключ, внешний ключ, ограничение и связанные сведения о данных в объектах `DataTable`.

Подключение к базе данных

Есть несколько способов подключения к БД в приложении. Проще всего это сделать при помощи графических инструментов Visual Studio 2010 во время разработки.

Для управления текущими соединениями с источниками данных служит окно **Server Explorer**. Текущие соединения с источниками данных, доступные в Visual Studio 2010, отображаются в окне **Server Explorer** в виде узлов дерева **Data Connections**.

В предыдущей главе, создавая базу данных `College`, мы уже создали соединение с ней, которое будет отображено в окне **Server Explorer** в виде узла **Data Connections**. Чтобы добавить к проекту соединение с базой данных, достаточно перетащить нужный узел **Data Connections** из окна **Server Explorer** в окно дизайнера формы. При этом автоматически создается новый объект `SqlConnection`, который конфигурируется для подключения к нужной БД.

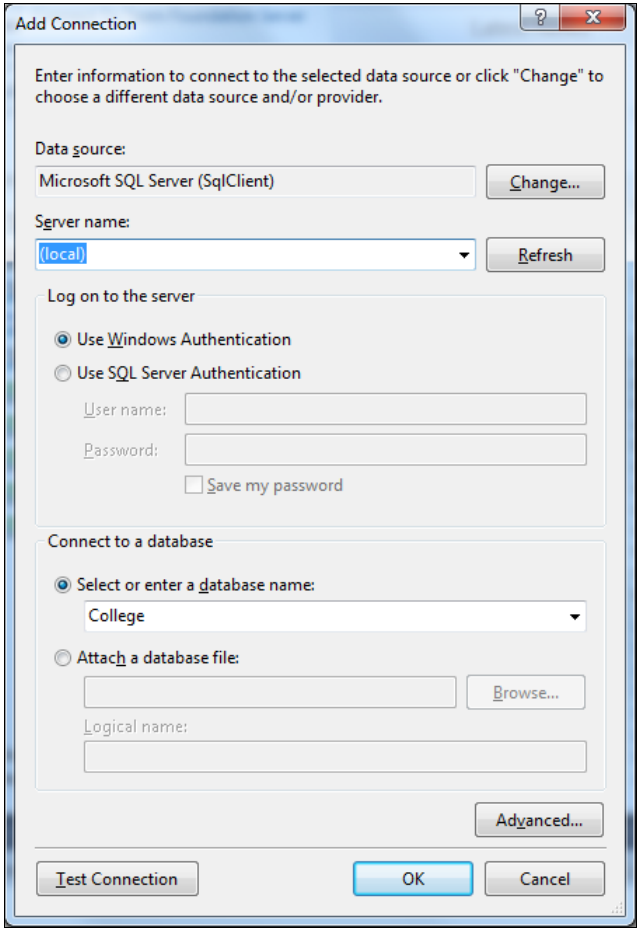


Рис. 7.2. Окно создания нового соединения **Add Connection**

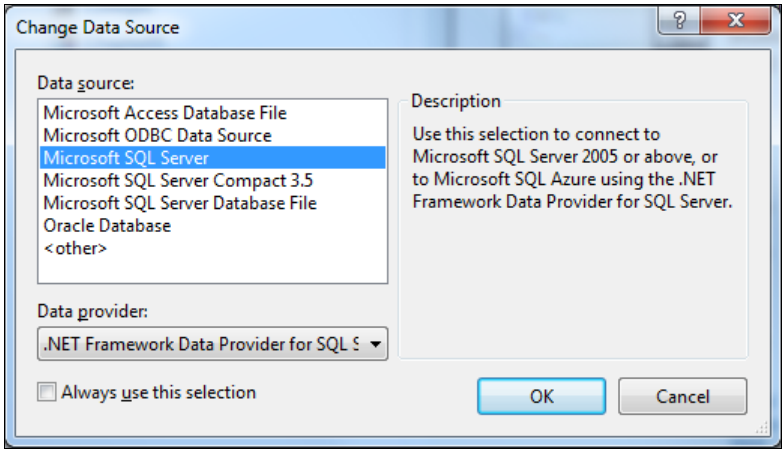


Рис. 7.3. Окно **Change Data Source**

Окно **Server Explorer** также позволяет создать новое соединение, щелкнув правой кнопкой узел **Data Connections** и выбрав из контекстного меню команду **Add Connection** (рис. 7.2).

Окно **Add Connection** позволяет выбрать провайдер данных. При нажатии кнопки **Change** появится диалоговое окно **Change Data Source**, которое предоставляет вам выбрать провайдер для подключения к базе данных (рис. 7.3).

Настроив новое подключение, можно проверить соединение, нажав кнопку **Test Connection**, при успешном выполнении соединения появится окно с текстом "Test connection succeeded".

Объект *Connection*

Создать объект `Connection` в коде программы довольно просто. Для этого надо создать новый объект `Command` и присвоить его свойству `ConnectionString` содержимое строки соединения с базой данных, как показано в листинге 7.1.

Листинг 7.1. Создание объекта `Command` в коде программы

```
SqlConnection con = new SqlConnection();
con.ConnectionString =
   "Data Source=(local);Initial Catalog=College;Integrated Security=True";
```

Можно также задать строку подключения в качестве параметра в конструкторе объекта `Connection`:

```
SqlConnection con = new SqlConnection(string connectionString);
```

Строку подключения к базе данных можно взять из окна свойств соединения с базой данных из узла **Data Connections** в окне **Server Explorer** (рис. 7.4).

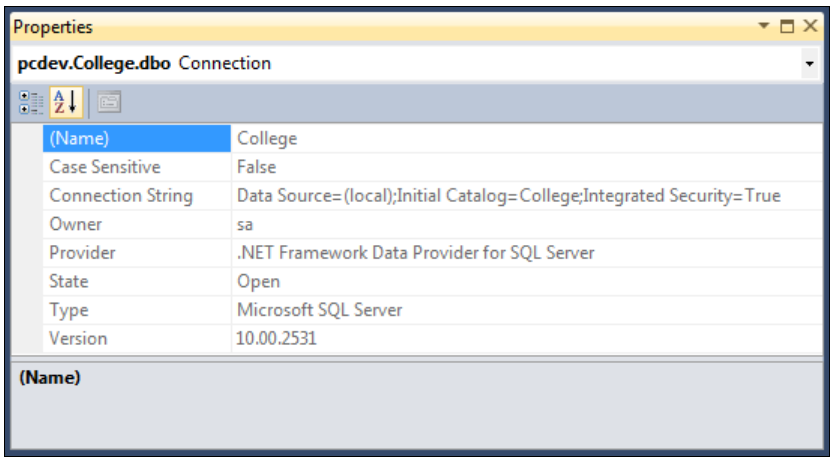


Рис. 7.4. Окно свойств соединения с базой данных `College`

Объект *Command*

Объект `Command` содержит ссылку на хранимую процедуру БД или оператор SQL и способен исполнить этот оператор на источнике данных, используя активное соединение. Объект `Command` также содержит все данные, необходимые для исполнения команды: ссылку на активное соединение, текст команды и ее параметры.

Подобно другим классам компонентов провайдера данных, класс `Command` представлен двумя вариантами. Первый, `OleDbCommand`, разработан для взаимодействия с самыми разными типами БД, а второй — `SqlCommand` — только для баз данных SQL Server.

Для класса `Command` достаточно активного соединения, взаимодействие с объектом `DataAdapter` ему не требуется. В силу этих причин объекты `Command` очень быстро и эффективно взаимодействуют с базами данных, позволяя выполнять различные операции:

- исполнять команды, не возвращающие значения, например `INSERT`, `UPDATE` и `DELETE`;
- исполнять команды, возвращающие единственное значение;
- исполнять команды языка Database Definition Language, например `CREATE TABLE` и `ALTER`;
- работать с объектом `DataAdapter`, возвращающим объект `DataSet`;
- возвращать результирующий набор непосредственно через экземпляр объекта `DataReader` — это самый быстрый способ доступа к данным, он особенно удобен, если данные требуются только для чтения;
- возвращать результирующий набор в виде потока XML — эту возможность поддерживает только класс `SqlCommand`;
- возвращать результирующий набор, созданный на основе нескольких таблиц или в результате исполнения нескольких операторов.

У объекта `Command` есть свойство `CommandType`, которое определяет тип команды, содержащийся в свойстве `CommandText`, оно может принимать одно из следующих значений:

- `Text` — заставляет рассматривать значение свойства `CommandText` как текст команды SQL, при этом в свойстве `CommandText` должен быть один или несколько допустимых операторов SQL, разделенных точкой с запятой. В последнем случае операторы SQL выполняются по порядку;
- `StoredProcedure` — в этом случае в свойстве `CommandText` необходимо указать имя существующей хранимой процедуры — она будет исполнена при вызове данной команды;
- `TableDirect` — в этом случае в свойстве `CommandText` должно быть имя одной или нескольких таблиц. При исполнении эта команда вернет все столбцы и строки таблиц, заданных свойством `CommandText`.

Например, для выборки данных из таблицы `Student` надо написать код, представленный в листинге 7.2.

Листинг 7.2. Создание объекта `Command`

```
SqlCommand cmd = new SqlCommand();
cmd.Connection = con;
cmd.CommandType = CommandType.Text;
cmd.CommandText = "SELECT * FROM Student";
```

Свойство `Connection` устанавливается в соответствии с типом соединения: для объекта `SqlCommand` требуется соединение на основе объекта `SqlConnection`, а для `OleDbCommand` — соединение на основе `OleDbConnection`.

Объект `Command` поддерживает три метода:

- `ExecuteNonQuery()` — исполняет команды, не возвращающие данные, например `INSERT`, `UPDATE` и `DELETE`;
- `ExecuteScalar()` — исполняет запросы к БД, возвращающие единственное значение;
- `ExecuteReader()` — возвращает результирующий набор через объект `DataReader`.

Все эти методы исполняют на источнике данных команду, представленную объектом `Command`, отличаются они возвращаемым значением. Метод `ExecuteNonQuery()` — самый простой из них, он не возвращает никаких значений. Этот метод обычно применяют для вызова хранимых процедур или команд SQL, таких как `INSERT`, `UPDATE` или `DELETE`.

Метод `ExecuteScalar()` возвращает только значение первого поля первой строки, извлеченной заданной командой, независимо от того, сколько строк выбрано этой командой в действительности. Метод `ExecuteReader()` возвращает неизменяемый объект `DataReader`, допускающий только последовательный однонаправленный просмотр без использования объекта `DataAdapter`. Если не требуется модифицировать содержимое базы данных или как-то иначе манипулировать им, этот способ извлечения данных является самым быстрым и эффективным.

Класс `SqlCommand` поддерживает еще один интересный метод — `ExecuteXmlReader()`, возвращающий результирующий набор в формате XML; результаты возвращаются в виде неизменяемого объекта `XmlReader`, доступного только для последовательно однонаправленного просмотра.

Объект *DataReader*

Исполнить с помощью объекта `Command` команду, возвращающую набор, можно методом `ExecuteReader()`. Он возвращает объект `DataReader`.

Объект `DataReader` — это упрощенный объект, обеспечивающий быстрое и эффективное последовательное однонаправленное чтение данных. Объект `DataReader` позволяет перебирать записи результирующего набора, передавая нужные значения напрямую коду приложения. При этом данные разрешается просматривать только в одном направлении: нельзя вернуться к записи, прочитанной ранее. Кроме того,

объект `DataReader` ориентирован на использование постоянных соединений и, пока он существует, требует монопольного доступа к активному соединению.

Объект `DataReader` нельзя создать напрямую, это делается путем вызова метода `ExecuteReader` объекта `Command`. Подобно другим членам классов провайдеров данных, у каждого класса `DataProvider` есть собственный класс `DataReader`. Например, объект `SqlCommand` возвращает `SqlDataReader`.

При вызове метода `ExecuteReader` объект `Command` исполняет представленную им команду и создает объект `DataReader` соответствующего типа, который можно записать в переменную ссылочного типа.

Получив ссылку на объект `DataReader`, можно просматривать записи, загружая нужные данные в память. У нового объекта `DataReader` указатель чтения устанавливается на первую запись результирующего набора. Чтобы сделать ее доступной, следует вызвать метод `Read`. Если запись доступна, метод `Read` переводит указатель объекта `DataReader` к следующей записи и возвращает `true`, в противном случае метод `Read` возвращает `false`. Таким образом, метод `Read` используют для перебора записей в цикле `while`, например, так:

```
SqlDataReader rdr = cmd.ExecuteReader();
while (rdr.Read())
{
   ...
}
```

Для перебора записей также можно использовать цикл `foreach`. Класс `SqlDataReader` содержит метод `GetEnumerator`, и при каждом шаге цикла он будет возвращать объект `DbDataRecord`, представляющий одну запись, который можно сохранить, например, в `ArrayList` (листинг 7.3).

Листинг 7.3. Создание `DataReader` и заполнение данными объекта `ArrayList`

```
// открываем соединение
con.Open();
SqlDataReader rdr = cmd.ExecuteReader();
ArrayList records = new ArrayList();
if (rdr.HasRows)
{
   foreach (DbDataRecord rec in rdr)
   {
      records.Add(rec);
   }
}
// закрываем соединение
con.Close();
```

Выполнение метода `ExecuteReader()` требует открытого соединения с базой данных. Закончив чтение данных, вызовите метод `Close()` объекта `DataReader`, чтобы закрыть соединение.

При чтении записи с помощью объекта DataReader значения отдельных полей доступны через индексатор или свойство по умолчанию в виде массива объектов, к элементам которого разрешается обращаться по индексу:

```
object o = rdr[3];
```

или по имени поля:

```
object o = rdr["CustomerID"];
```

При этом способе доступа DataReader предоставляет все значения в виде объектов, хотя из DataReader допустимо извлекать и типизированные данные. Более подробно об этом рассказано далее.

Прочитав данные с помощью DataReader, следует вызвать метод Close(), чтобы закрыть DataReader, в противном случае объект DataReader будет удерживать монопольный доступ к активному соединению, сделав его недоступным другим объектам. При вызове метода ExecuteReader(), установив свойство CommandBehavior в CloseConnection, вы автоматически закроете соединение, не вызывая метод Close() явно.

Приложение для чтения данных

Имея теперь представление об архитектуре доступа к данным, давайте напишем приложение для работы с нашей базой данных. Создадим проект Windows Forms Application с именем College и с одной формой, которую назовем FMain.

На форму добавим элементы StatusStrip и DataGridView для отображения табличных данных и зададим ему свойство Dock значением DockStyle.Fill. В StatusStrip добавим метку StatusLabel с именем lbRecordsCount для отображения количества записей (рис. 7.5).

Для формы создадим обработчик события Load, в тело которого мы будем писать код для чтения информации из базы данных.

Сначала надо создать объект Connection. Пример создания объекта Connection был показан в листинге 7.1 этой главы, однако в реальных приложениях строку соединения не задают жестко в коде программы, а выносят во внешний файл настроек приложения. Откройте окно свойств проекта и выберите вкладку **Settings**. Создайте новый параметр, который будет хранить строку соединения с базой данных (рис. 7.6):

- **Name** — ConString;
- **Type** — Connection string;
- **Scope** — Application;
- **Value** — Data Source=(local);Initial Catalog=College;Integrated Security=True.

При этом в файле конфигурации приложения появится параметр соединения с базой данных, как показано в листинге 7.4.

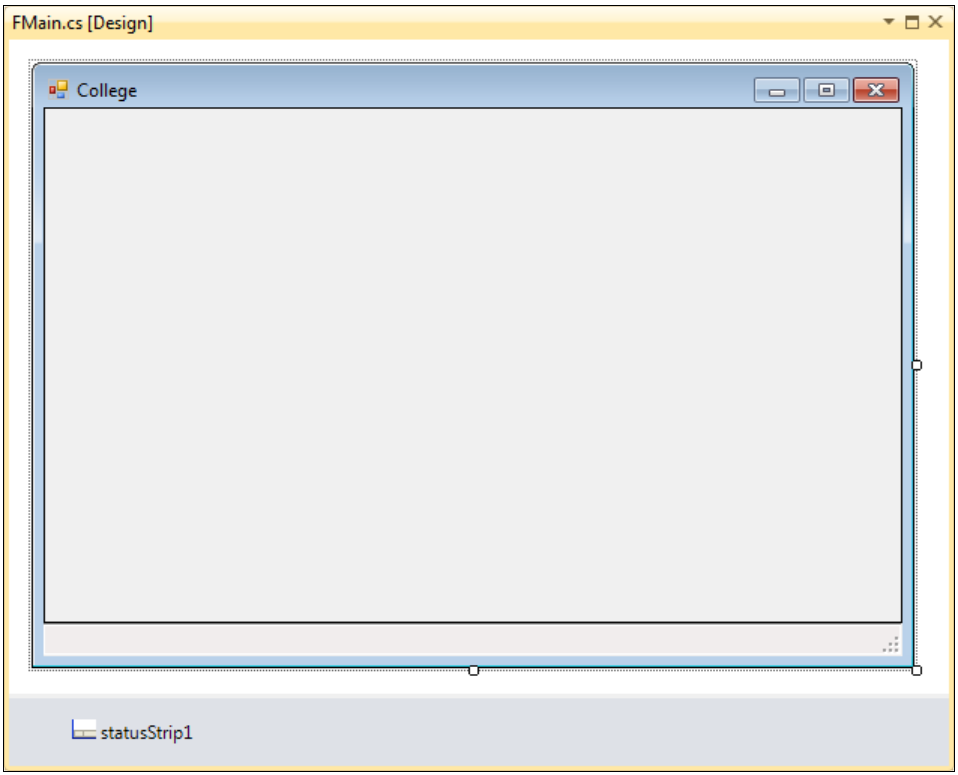


Рис. 7.5. Дизайн формы приложения для чтения данных

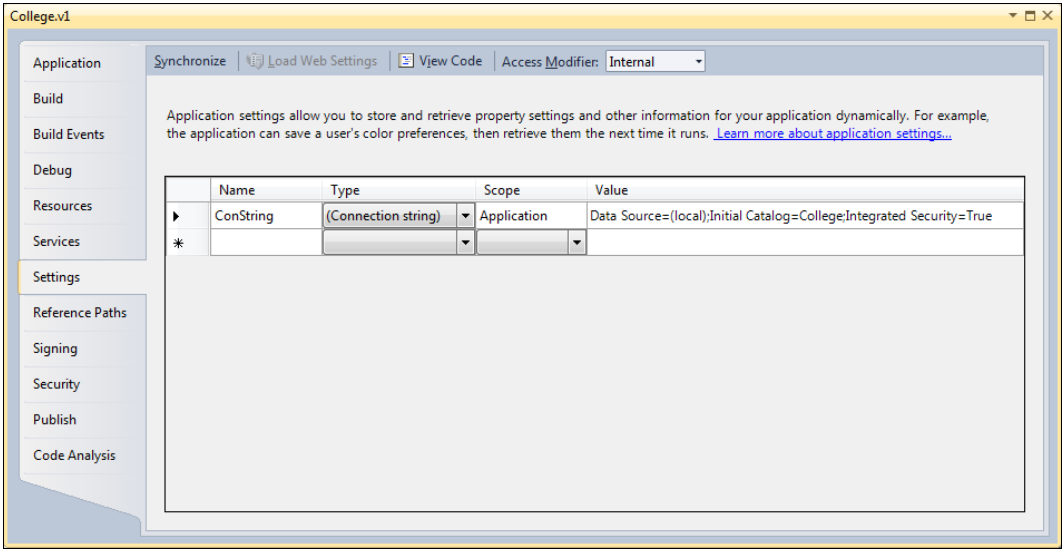


Рис. 7.6. Вкладка **Settings** проекта с созданным параметром соединения с базой данных

Листинг 7.4. Файл конфигурации приложения

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="College.Properties.Settings.ConString"
            connectionString=
      "Data Source=(local);Initial Catalog=College;Integrated Security=True"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
</configuration>
```

Теперь передать строку соединения из конфигурационного файла в объект `Command` можно следующим образом:

```
SqlConnection con = new SqlConnection();
con.ConnectionString = Properties.Settings.Default.ConString;
```

Далее нужно создать объект `Command`. Для создания объекта `Command` используйте код из листинга 7.2 этой главы. Затем надо вызвать метод `ExecuteReader()` для получения объекта `SqlDataReader` и сохранить данные, например, в объекте `ArrayList` (или в `DataTable`).

У элемента `DataGridView` есть свойство для привязки данных `DataSource`, которое задает источник данных типа `Object` (а это может быть `ArrayList`, `DataTable` и другие типы), для которого `DataGridView` отображает данные. Код для чтения данных поместим в тело созданного обработчика события `Load` формы, как показано в листинге 7.5 (пример на диске находится в каталоге Chapter07\College.v1).

Листинг 7.5. Приложение для чтения данных

```
using System;
using System.Collections;
using System.Data;
using System.Data.Common;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace College
{
   public partial class FMain : Form
   {
      public FMain()
      {
         InitializeComponent();
      }
```

```
        private void FMain_Load(object sender, EventArgs e)
        {
            SqlConnection con = new SqlConnection();
            con.ConnectionString = Properties.Settings.Default.ConString;

            SqlCommand cmd = new SqlCommand();
            cmd.Connection = con;
            cmd.CommandType = CommandType.Text;
            cmd.CommandText = "SELECT * FROM Student";

            con.Open();
            SqlDataReader rdr = cmd.ExecuteReader();
            ArrayList records = new ArrayList();
            if (rdr.HasRows)
            {
                foreach (DbDataRecord rec in rdr)
                {
                    records.Add(rec);
                }
            }
            con.Close();
            dgvData.DataSource = records;
            lbRecordCount.Text = "Count: " + records.Count;
        }
    }
}
```

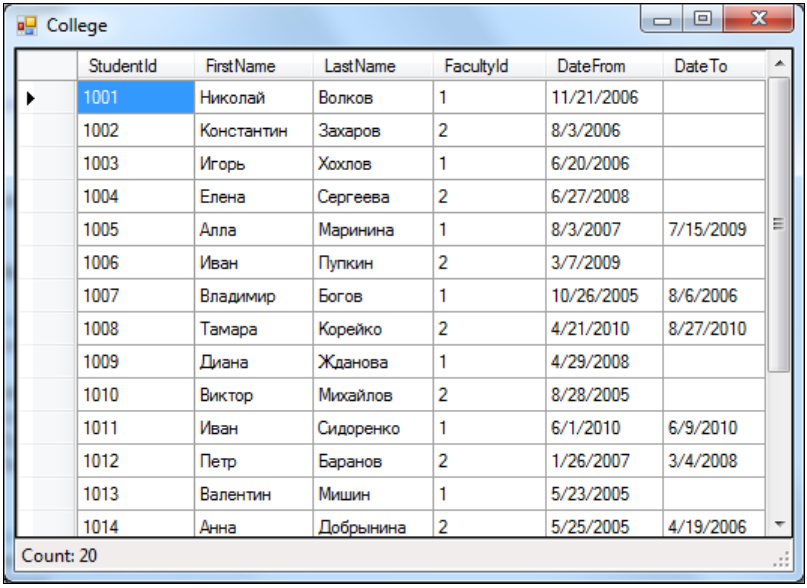


Рис. 7.7. Вывод данных в `DataGridView`

Запустив приложение, вы увидите, что элемент DataGridView отобразит записи из таблицы Student нашей базы данных (рис. 7.7).

Передача параметров в объект *Command*

Параметры — это значения, которыми заполняют поля подстановки, введенные в текст команды во время разработки. Каждый параметр представлен экземпляром класса SqlParameter. Параметры хранятся в свойстве Parameters объекта Command. В период выполнения значения параметров считываются из этого свойства и подставляются в оператор SQL либо передаются хранимой процедуре.

Объекты Command поддерживают набор Parameters, в котором хранятся объекты Parameter соответствующего типа. Объект Parameter выполняет преобразование параметров из типа, используемого в приложении, в тип, используемый в БД. Поскольку эти свойства взаимосвязаны, при изменении значения одного из них значение другого автоматически изменяется и преобразуется в соответствующий поддерживаемый тип. Аналогичным образом связаны свойства DbType и SqlType объектов SqlParameter, свойство SqlType указывает тип SQL, представленный этим параметром.

Свойство Direction объекта параметра определяет, является ли этот параметр входным или выходным. Возможные значения для этого свойства — Input, Output, InputOutput или ReturnValue — указывают, должен ли параметр хранить значение, возвращаемое хранимой процедурой или функцией.

На члены набора Parameters ссылаются в коде по индексу либо по имени, заданному свойством ParameterName. Свойства Precision, Scale и Size определяют длину и точность значения параметров. Свойства Precision и Scale применяются с числовыми и десятичными параметрами и определяют разрядность и длину дробной части значения свойства Value соответственно, а свойство Size применяется с двоичными и строковыми параметрами и представляет максимальную длину такого поля. Свойства SourceColumn и SourceVersion используются с параметрами, привязанными к полям объекта DataTable. Свойство SourceColumn указывает поле для поиска или сопоставления значений, а свойство SourceVersion — версию поля для редактирования. Свойство Value содержит значение параметра.

Переделаем нашу программу. Для примера возьмем хранимую процедуру Grade_Select и построим запрос, возвращающий оценки одного студента. Для вызова хранимой процедуры установим свойство CommandType в StoredProcedure, запишем в свойство CommandText имя хранимой процедуры. Внесем в код программы соответствующие изменения, как показано в листинге 7.6 (пример на диске находится в каталоге Chapter07\College.v2).

Листинг 7.6. Построение параметризованного запроса

```
...
cmd.Connection = con;
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = "Grade_Select";
```

```
cmd.Parameters.Add(new SqlParameter("@studentId", SqlDbType.Int));
cmd.Parameters["@studentId"].Value = 1011;
...
```

Запустите приложение. Результат вывода параметризованного запроса для таблицы `Grade` показан на рис. 7.8.

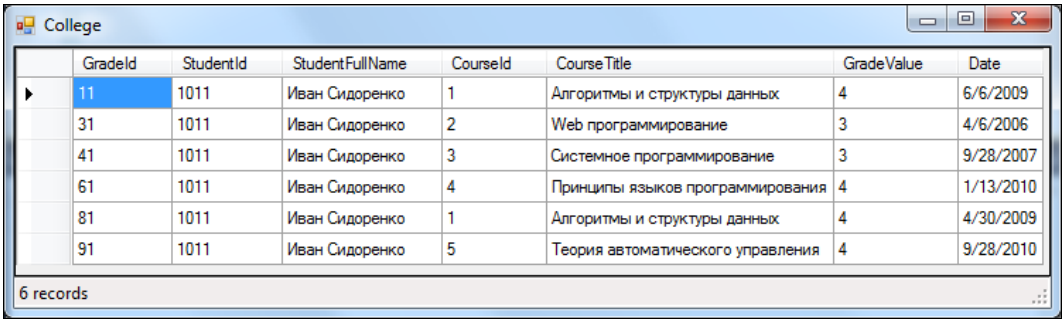


Рис. 7.8. Вывод данных параметризованного запроса

Использование типизированного объекта *DataReader*

Для работы с данными в приложении почти всегда необходимо получать из базы строго типизированные данные, а не типа `Object`. Объект `DataReader` предоставляет методы для извлечения данных заданного типа из результирующего набора. Например, метод `GetBoolean()` извлекает значения типа `Boolean`. Если известен тип данных некоторого столбца, можете воспользоваться методами объекта `DataReader` для извлечения из этого столбца строго типизированных данных. Например:

```
string str = rdr.GetBoolean(1);
```

При использовании такого способа извлечения данных вы должны указать порядковый номер, а не имя поля. Если известно только имя поля, можно определить его порядковый номер, вызвав метод `GetOrdinal`, например, так:

```
int ord = rdr.GetOrdinal("LastName");
string lastName = rdr.GetString(ord);
```

Если тип столбца неизвестен, можно воспользоваться методом `GetFieldType()`, который возвращает тип `System.Type`, являющийся типом данных объекта. В качестве параметра этому методу передается порядковый номер столбца.

Пример с извлечением строго типизированных данных из `SqlDataReader` показан в листинге 7.7 (пример на диске находится в каталоге Chapter07\College.v3).

Листинг 7.7. Код обработчика события `Load()` формы

```
private void FMain_Load(object sender, EventArgs e)
{
   SqlConnection con = new SqlConnection();
```

```
    con.ConnectionString = Properties.Settings.Default.ConString;
    SqlCommand cmd = new SqlCommand();
    cmd.Connection = con;
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.CommandText = "Grade_Select";
    cmd.Parameters.Add(new SqlParameter("@studentId", SqlDbType.Int));
    cmd.Parameters["@studentId"].Value = 1011;

    con.Open();

    SqlDataReader rdr = cmd.ExecuteReader();
    DataTable dt = new DataTable();

    for (int i = 0; i < rdr.FieldCount; i++)
    {
       dt.Columns.Add(
          new DataColumn(rdr.GetName(i), rdr.GetFieldType(i)));
    }
    while (rdr.Read())
    {
       DataRow row = dt.NewRow();
       for (int i = 0; i < rdr.FieldCount; i++)
       {
          row[i] = rdr.GetValue(i);
       }
       dt.Rows.Add(row);
    }
    con.Close();
    dgvData.DataSource = dt;
    toolStripStatusLabel1.Text = String.Format("{0} records",
         dt.Rows.Count);
}
```

Записи из объекта `SqlDataReader` сохраняются в `DataTable`, строго типизированные столбцы которой создаются динамически, во время выполнения программы, а затем `DataTable` заполняется записями из базы данных. Сама выборка данных аналогична полученной в предыдущей программе (см. рис. 7.7).

Использование строго типизированных данных позволяет производить не только чтение, но и их правильное использование и валидацию при модифицировании этих данных в программе.

Модификация данных

Модифицировать и передавать данные в подсоединенном режиме очень просто. Для обновления данных обычно используют метод `ExecuteNonQuery`, который выполняет запрос, не создавая объект `DataReader` для приема его результатов.

Значения возвращаемых параметров и параметров вывода становятся доступными по завершении вызова метода `ExecuteNonQuery`. Метод `ExecuteNonQuery` возвращает целое число, соответствующее количеству записей, затронутых запросом.

Добавим в наше приложение возможность добавления, изменения и удаления записей.

На главную форму добавим элементы `MenuStrip` и `ToolStrip` и поместим на них кнопки для добавления, изменения и удаления данных, а также для обновления содержимого `DataGridView` (рис. 7.9).

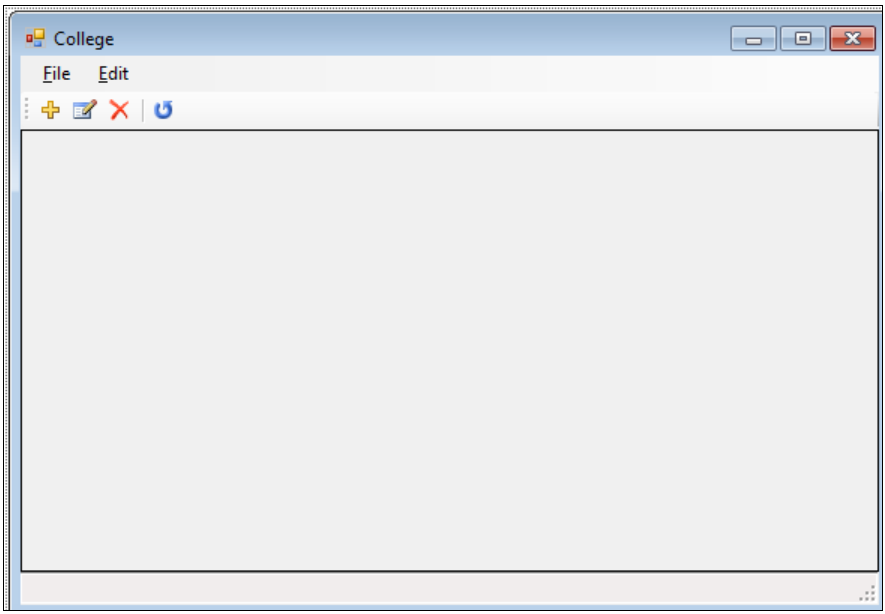


Рис. 7.9. Главная форма приложения

Для добавления и модификации записей создадим отдельную форму. Конечно, можно просто вводить данные напрямую в `DataGridView`, но в реальных приложениях хорошим стилем программирования пользовательского интерфейса считается создание отдельной формы для ввода и модификации данных. На форме будет пять полей ввода: два элемента `TextBox` для имени и фамилии (`tbFirstName`, `tbLastName`), элемент `ComboBox` для выбора факультета (`cbFaculty`) и два поля `DateTimePicker` для ввода дат (`dtDateFrom` и `dtDateTo`). Также добавьте компонент `ErrorProvider` для валидации вводимых данных. Дизайн формы представлен на рис. 7.10.

Для выбора факультета надо наполнить данными выпадающий список `cbFaculty`. Для этого нужно выполнить запрос, вызвав хранимую процедуру `Faculty_Select` без параметров, которая вернет список факультетов, и привязать данные (объект `DataTable`) к элементу `ComboBox`. У `ComboBox` есть свойство `DataSource`, значение которого задается так же, как и значение свойства `DataSource` элемента управления `DataGrid`.

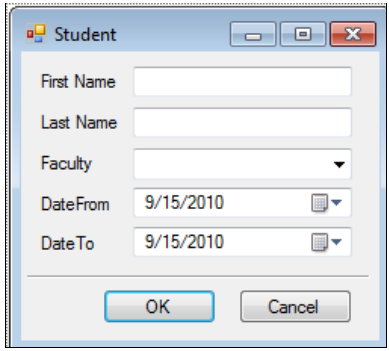


Рис. 7.10. Форма для ввода и модификации данных

При создании экземпляра `DataTable` мы присваиваем столбцам таблицы имена полей из объекта `SqlDataReader`. Свойство `DisplayMember` определяет, значениями какого столбца элемент `ComboBox` заполняет список, и принимает имя соответствующего столбца `DataTable`. В нашем случае это имя столбца `FacultyName` объекта `DataTable`. Затем надо задать свойству `ValueMember` имя столбца `FacultyId` объекта `DataTable`:

```
DataTable dt = new DataTable();
...
cbFaculty.DataSource = dt;
cbFaculty.ValueMember = "FacultyId";
cbFaculty.DisplayMember = "FacultyName";
cbFaculty.SelectedValue = -1;
```

В конструкторе формы будут два параметра: модифицируемая запись из базы данных, передаваемая как `DataRow`, и переменная типа `bool`, указывающая на действие — обновление записи или создание новой записи. Код класса формы `FStudent` представлен в листинге 7.8 (полный код примера находится в каталоге Chapter07\College.v4 на прилагаемом к книге диске).

Листинг 7.8. Код класса формы `FStudent`

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;

namespace College
{
   public partial class FStudent : Form
   {
      private DataRow row;
      private bool isUpdate;
```

```
public FStudent(DataRow inputRow, bool update)
{
   InitializeComponent();
   row = inputRow;
   isUpdate = update;
}

private void FStudent_Load(object sender, EventArgs e)
{
   SqlConnection con = new SqlConnection(
      College.Properties.Settings.Default.ConString);
   SqlCommand com = new SqlCommand();
   com.Connection = con;
   com.CommandType = CommandType.StoredProcedure;
   com.CommandText = "Faculty_Select";
   con.Open();
   SqlDataReader rdr = com.ExecuteReader();
   DataTable dt = new DataTable();
   for (int i = 0; i < rdr.FieldCount; i++)
   {
      dt.Columns.Add(new DataColumn(rdr.GetName(i),
         rdr.GetFieldType(i)));
   }
   while (rdr.Read())
   {
      DataRow r = dt.NewRow();
      for (int i = 0; i < rdr.FieldCount; i++)
      {
         r[i] = rdr.GetValue(i);
      }
      dt.Rows.Add(r);
   }
   cbFaculty.DataSource = dt;
   cbFaculty.ValueMember = "FacultyId";
   cbFaculty.DisplayMember = "FacultyName";
   cbFaculty.SelectedValue = -1;

   if (isUpdate)
   {
      this.Text = "Update student";
      tbFirstName.Text = row["FirstName"].ToString();
      tbLastName.Text = row["LastName"].ToString();
      cbFaculty.SelectedValue = (int)row["FacultyId"];
      dtDateFrom.Value = (DateTime)row["DateFrom"];
   }
```

```
        else
        {
            this.Text = "Insert new student";
        }

        if (row["DateTo"] == System.DBNull.Value)
        {
            dtDateTo.Value = dtDateTo.MaxDate;
        }
    }

    private void control_Validating(object sender, CancelEventArgs e)
    {
        Control c = (Control)sender;
        if (c.Text.Length == 0)
        {
            errProvider.SetIconAlignment(c, ErrorIconAlignment.MiddleRight);
            errProvider.SetError(c, "This field must be not empty!");
            c.Focus();
        }
        else
        {
            errProvider.SetError(c, "");
        }
    }

    private void bOK_Click(object sender, EventArgs e)
    {
        if (tbFirstName.Text.Length > 0 &&
            tbLastName.Text.Length > 0 &&
            cbFaculty.Text.Length > 0)
        {
            row["FirstName"] = tbFirstName.Text;
            row["LastName"] = tbLastName.Text;
            row["FacultyId"] = (int)cbFaculty.SelectedValue;
            row["DateFrom"] = dtDateFrom.Value;
            if (dtDateTo.Value == dtDateTo.MaxDate)
                row["Dateto"] = System.DBNull.Value;
            else
                row["DateTo"] = dtDateTo.Value;
        }
        else
        {
            this.DialogResult = DialogResult.None;
        }
    }
```

```
      public DataRow Row
      {
         get { return row; }
         set { row = value; }
      }
   }
}
```

На главной форме `FMain` загрузку данных в `DataGridView` выделим в отдельный метод, представленный в листинге 7.9.

Листинг 7.9. Загрузка данных в `DataGridView`

```
private void LoadData()
{
   SqlCommand cmd = new SqlCommand();
   cmd.Connection = connection;
   cmd.CommandType = CommandType.StoredProcedure;
   cmd.CommandText = "Student_Select";

   connection.Open();
   SqlDataReader rdr = cmd.ExecuteReader();
   DataTable dt = new DataTable();

   for (int i = 0; i < rdr.FieldCount; i++)
   {
      dt.Columns.Add(new DataColumn(rdr.GetName(i), rdr.GetFieldType(i)));
   }

   while (rdr.Read())
   {
      DataRow row = dt.NewRow();
         for (int i = 0; i < rdr.FieldCount; i++)
         {
            row[i] = rdr.GetValue(i);
         }
         dt.Rows.Add(row);
   }

   connection.Close();
   dgvData.DataSource = dt;
   toolStripStatusLabel1.Text = String.Format("{0} records", dt.Rows.Count);
}
```

Метод `LoadData()` будет вызываться при первой загрузке данных в форму и при обновлении данных. Реализация обработчика события `Load` формы представлена в листинге 7.10.

Листинг 7.10. Реализация обработчика события Load формы

```
private SqlConnection connection;
...
private void Form1_Load(object sender, EventArgs e)
{
   connection = new SqlConnection(
       Properties.Settings.Default.ConString);
   LoadData();
}
```

Для кнопок вставки, модификации, удаления данных напишем обработчики событий, код которых представлен в листингах 7.11—7.14.

Листинг 7.11. Реализация обработчика события кнопки добавления новой записи

```
private void bNew_Click(object sender, EventArgs e)
{
   FStudent fStudent = new FStudent(((DataTable)dgvData.DataSource).NewRow(),
false);
   if (fStudent.ShowDialog() == DialogResult.OK)
   {
      SqlCommand com = new SqlCommand();
      com.Connection = connection;
      com.CommandType = CommandType.StoredProcedure;
      com.CommandText = "Student_Insert";
      com.Parameters.Add(new SqlParameter("@firstName",
                         SqlDbType.NVarChar));
      com.Parameters["@firstName"].Value = fStudent.Row["FirstName"];
      com.Parameters.Add(new SqlParameter("@lastName", SqlDbType.NVarChar));
      com.Parameters["@lastName"].Value = fStudent.Row["LastName"];
      com.Parameters.Add(new SqlParameter("@facultyId", SqlDbType.Int));
      com.Parameters["@facultyId"].Value = fStudent.Row["FacultyId"];
      com.Parameters.Add(new SqlParameter("@dateFrom", SqlDbType.Date));
      com.Parameters["@dateFrom"].Value = fStudent.Row["DateFrom"];
      connection.Open();

      if (com.ExecuteNonQuery() > 0)
      {
         MessageBox.Show("Add Student Successfully");
      }

      connection.Close();
      LoadData();
      dgvData.Rows[dgvData.Rows.Count - 1].Selected = true;
   }
}
```

Листинг 7.12. Реализация обработчика события кнопки модификации записи

```
private void bEdit_Click(object sender, EventArgs e)
{
   int id = (int)dgvData.SelectedRows[0].Cells["StudentId"].Value;
   DataTable dt = (DataTable)dgvData.DataSource;
   DataRow[] rows = dt.Select(String.Format("StudentId = {0}", id));
   FStudent fStudent = new FStudent(rows[0], true);

   if (fStudent.ShowDialog() == DialogResult.OK)
   {
      SqlCommand com = new SqlCommand();
      com.Connection = connection;
      com.CommandType = CommandType.StoredProcedure;
      com.CommandText = "Student_Update";

      com.Parameters.Add(new SqlParameter(
            "@studentId", SqlDbType.NVarChar));
      com.Parameters["@studentId"].Value = id;
      com.Parameters.Add(new SqlParameter(
            "@firstName", SqlDbType.NVarChar));
      com.Parameters["@firstName"].Value = fStudent.Row["FirstName"];
      com.Parameters.Add(new SqlParameter(
             "@lastName", SqlDbType.NVarChar));
      com.Parameters["@lastName"].Value = fStudent.Row["LastName"];
      com.Parameters.Add(new SqlParameter(
             "@facultyId", SqlDbType.NVarChar));
      com.Parameters["@facultyId"].Value = fStudent.Row["FacultyId"];
      com.Parameters.Add(new SqlParameter(
             "@dateFrom", SqlDbType.Date));
      com.Parameters["@dateFrom"].Value = fStudent.Row["DateFrom"];
      com.Parameters.Add(new SqlParameter("@dateTo", SqlDbType.Date));
      com.Parameters["@dateTo"].Value = fStudent.Row["DateTo"];

      connection.Open();
      com.ExecuteNonQuery();
      connection.Close();
      LoadData();
   }
}
```

Листинг 7.13. Реализация обработчика события кнопки удаления записи

```
private void bDelete_Click(object sender, EventArgs e)
{
   int id = (int)dgvData.SelectedRows[0].Cells["StudentId"].Value;
```

```
    if (MessageBox.Show("Delete this record?", "Confirm delete",
          MessageBoxButtons.YesNo) == DialogResult.OK)
    {
       SqlCommand com = new SqlCommand();
       com.Connection = connection;

       com.CommandType = CommandType.StoredProcedure;
       com.CommandText = "Student_Delete";
       com.Parameters.Add(new SqlParameter(
             "@studentId", SqlDbType.NVarChar));
       com.Parameters["@studentId"].Value = id;

       connection.Open();
       com.ExecuteNonQuery();
       connection.Close();
       LoadData();
    }
}
```

Листинг 7.14. Реализация обработчика события кнопки обновления данных

```
private void bRefresh_Click(object sender, EventArgs e)
{
   LoadData();
}
```

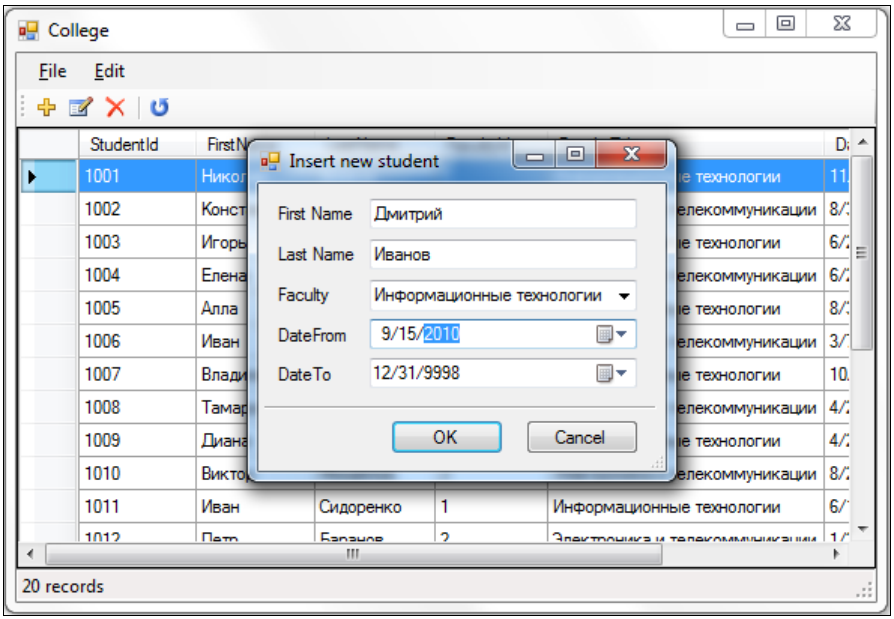


Рис. 7.11. Добавление новой записи

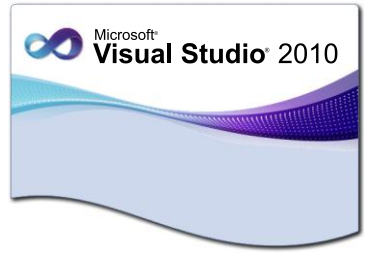
Полный код приложения вы можете найти на диске в каталоге Chapter07\College.v4. Скомпилируйте и запустите приложение и протестируйте его работоспособность. Внешний вид приложения представлен на рис. 7.11.

Резюме

Доступ к данным — чрезвычайно важная часть приложения. В этой главе мы рассмотрели основы технологии доступа к данным ADO.NET: подключение к базе данных, создание объектов `Connection`, `Command` и `DataReader`.

При разработке приложений для работы с данными доступ к ним нужно реализовать правильно для эффективной работы этого приложения. Рассмотренный в этой главе объект `DataReader` — это простой объект, обеспечивающий быстрое и эффективное последовательное однонаправленное чтение данных. Объект `DataReader` позволяет перебирать записи результирующего набора данных. При этом данные возможно просматривать только в одном направлении. Кроме того, объект `DataReader` ориентирован на использование постоянных соединений с базой данных и, пока он существует, требует монопольного доступа к активному соединению, т. е. работает в подключенном к источнику данных режиме.

В следующих главах мы будем рассматривать организацию доступа к данным в автономном режиме, который является более эффективным решением при создании многопользовательских распределенных систем для работы с данными.



ГЛАВА 8

Работа с автономными данными в ADO.NET

Основная задача использования ADO.NET — это работа с данными в автономном режиме. Следовательно, нам нужен объект для хранения автономных данных. В силу автономности данных реализация этого объекта не должна зависеть от источника данных, поскольку он играет роль универсального посредника. Другими словами, реализация объекта, хранящего автономные данные, не должна учитывать особенности исходного источника данных, например, для него не должно быть разницы, является ли источником данных Microsoft SQL Server или Oracle. Этим посредником между подключенным и автономным миром является объект `DataAdapter`.

Объект *DataAdapter*

Объекты `DataAdapter` обеспечивают связь между источником данных и объектом `DataSet` (или `DataTable`), управляя обменом данных и контролируя их передачу. Одни приложения осуществляют одностороннюю передачу данных, а другим требуется не только извлекать, но и модифицировать данные источника. Объект `DataAdapter` способен извлекать данные, заполнять объекты `DataSet` или `DataTable` и при необходимости обновлять содержимое источника данных. Так как адаптер данных является посредником между источником данных и объектом для хранения автономных данных, его конкретная реализация зависит от типа источника данных. Для работы с базами данных SQL Server используется специализированный класс `SqlDataAdapter`, а для работы с базами данных Oracle — класс `OracleDataAdapter`.

У класса `SqlDataAdapter` имеются четыре основных свойства для работы с источником данных:

- `SelectCommand` — содержит текст или объект команды, осуществляющей выборку данных из базы данных; при вызове метода `Fill()` эта команда исполняется и заполняет объект `DataTable` или объект `DataSet`;
- `InsertCommand` — содержит текст или объект команды, осуществляющий вставку строк в таблицу;

- `DeleteCommand` — содержит текст или объект команды, осуществляющий удаление строки из таблицы;
- `UpdateCommand` — содержит текст или объект команды, осуществляющий обновление значений в базе данных.

Чтобы заполнить `DataTable` данными, следует вызвать метод `Fill()` объекта `DataAdapter`. Этот метод исполняет команды, заданные свойством `SelectCommand`, на соединении, указанном свойством `Connection`, и заполняет `DataSet` данными, которые возвращает исполненная команда. Методу `Fill()` необходимо передать целевой объект, которым может быть `DataTable`, например:

```
DataTable dt = new DataTable();
myDataAdapter.Fill(dt);
```

Обратите внимание, что взаимодействия с объектом `Connection` не происходит. При исполнении команды, вызванной методом `Fill()`, соединение открывается только на время извлечения данных, после чего сразу же закрывается. Таким образом, после извлечения данные становятся отсоединенными и ими можно манипулировать в коде независимо от базы данных, а при необходимости ее можно и обновить.

Рассмотрим это на практическом примере. Создадим новый проект Windows Forms. На форме будут расположены элементы управления `DataGridView` и `StatusBar`. Такую форму вы уже создавали в *главе 7* (см. рис. 7.5). Можете использовать ее в качестве шаблона для данного проекта. Код приложения представлен в листинге 8.1.

> ***Примечание***
>
> Полный код приложения на диске, прилагаемом к книге, находится в каталоге Chapter08\College.v1.

Листинг 8.1. Использование `DataAdapter` для передачи данных в приложение

```
using System;
using System.Data;
using System.Data.Common;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace College
{
   public partial class FMain : Form
   {
      public FMain()
      {
         InitializeComponent();
      }
```

```
    private void Form1_Load(object sender, EventArgs e)
    {
        DataTable table = new DataTable();
        table.TableName = "Course";

        SqlDataAdapter adapter = new SqlDataAdapter();
        SqlConnection _connection = new SqlConnection(
              Properties.Settings.Default.ConString);
        table.Rows.Clear();
        SqlCommand com = new SqlCommand();
        com.Connection = _connection;
        com.CommandType = CommandType.StoredProcedure;
        com.CommandText = "Course_Select";

        adapter.SelectCommand = com;
        adapter.Fill(table);
        table.Columns["TrainerId"].ColumnMapping = MappingType.Hidden;
        table.Columns["FacultyId"].ColumnMapping = MappingType.Hidden;

        dgvData.DataSource = table;
        lbRowCount.Text = String.Format("{0} records", table.Rows.Count);
    }
  }
}
```

Внешний вид приложения, отображающего таблицу `Course`, представлен на рис. 8.1.

Как видно из листинга 8.1, один и тот же объект `DataTable` может быть многократно передан методу `Fill()`, при вызове которого у объекта `DataAdapter` заполняется данными `DataTable`.

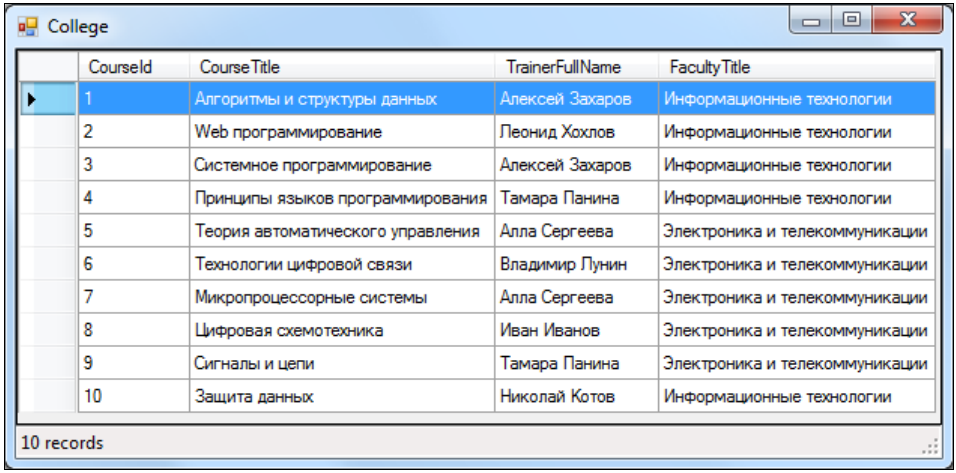


Рис. 8.1. Заполнение данных с использованием `DataAdapter`

Взаимодействие объектов *DataAdapter* и *DataSet*

В предыдущей главе мы уже кратко упоминали об объекте `DataSet` при рассмотрении архитектуры доступа к данным. Теперь рассмотрим `DataSet` более подробно, а также изучим взаимодействие `DataSet` с объектом `DataAdapter`.

Объект `DataSet` — это центральный компонент архитектуры доступа, основанной на использовании автономных данных. Объекты `DataSet` представляют данные, загруженные в память. Для заполнения `DataSet` используется объект `DataAdapter`.

В принципе, объект `DataSet` представляет собой копию базы данных, частичную или полную. Получение доступа к данным объекта `DataSet` не требует взаимодействия с базой данных, а модификация данных, загруженных в `DataSet`, не отражается на содержимом базы данных вплоть до ее обновления.

Хотя объект `DataSet` не соединен с их источником данных, он представляет полный набор данных, включая таблицы, связи между ними и ограничения. Поскольку `DataSet` является отсоединенным представлением данных, он способен хранить информацию из нескольких источников. Взаимодействие между `DataSet` и базой данных протекает под управлением объекта `DataAdapter`.

Объект `DataAdapter` инкапсулирует функциональность, необходимую для заполнения `DataSet` данными и обновления базы данных, играя роль "моста" между соединением и объектом `DataSet`. В свойстве `Connection` объекта `DataAdapter` хранится ссылка на объект соединения, представляющий базу данных и позволяющий манипулировать ее содержимым. Объект `DataAdapter` также поддерживает свойство `SelectCommand`. Это свойство содержит объект `Command`, который представляет команды, осуществляющие выборку данных из источника. Как и у других компонентов провайдера данных, реализация объекта `DataAdapter` специфична для типа провайдера: `SqlDataAdapter` разработан специально для взаимодействия с SQL Server, а `OleDbDataAdapter` позволяет подключать самые разные источники данных.

Теперь создадим приложение, способное загружать информацию из всех таблиц нашей базы данных.

> ***Примечание***
>
> Полный код приложения на прилагаемом к книге диске находится в каталоге Chapter08\College.v2.

Поместите на форму элемент `SplitContainer`, в левой части которого расположите элемент управления `TreeView`, который будет отображать дерево объектов нашей базы данных, а в правой части поместите элемент управления `DataGridView` для отображения записей выбранного объекта базы данных.

Также поместите на форму компонент `ImageList`, который будет контейнером иконок для элемента `TreeView`. Поместите в `ImageList` две иконки для отображения корневого и дочерних узлов дерева. Подходящие иконки можно взять в библиотеке VS2010 Image Library, поставляемой вместе с Visual Studio 2010. Внешний вид формы в дизайнере представлен на рис. 8.2.

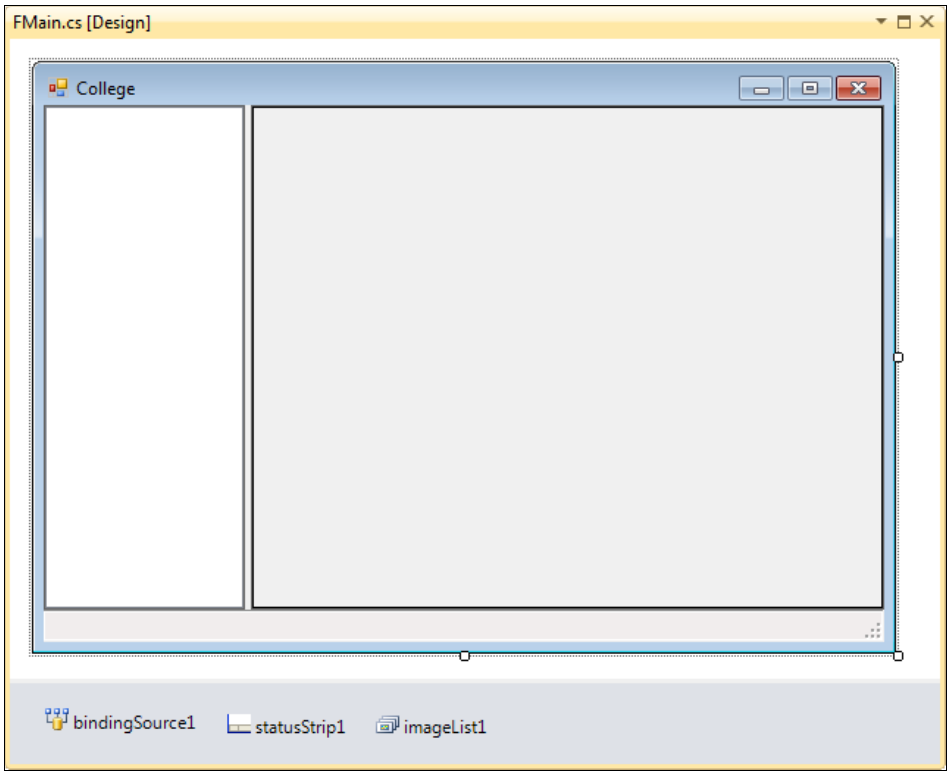


Рис. 8.2. Дизайн формы приложения

В окне **Properties** для элемента `TreeView` свяжите свойство `ImageList` с нашим компонентом `ImageList`.

Для названий объектов базы данных будем использовать именованные константы. В тексте программы имена объектов базы данных будут многократно использоваться, и применение именованных констант, что является хорошим стилем кодирования, упростит обращения к объектам и уменьшит вероятность ошибок, которые могут появиться при использовании строковых значений имен объектов. Набор констант-членов класса представлен в листинге 8.2.

Листинг 8.2. Именованные константы класса `FMain`

```
private const string CourseEntity = "Course";
private const string FacultyEntity = "Faculty";
private const string GradeEntity = "Grade";
private const string StudentEntity = "Student";
private const string TrainerEntity = "Trainer";
```

Для класса `FMain` создайте группу закрытых переменных-членов класса, как показано в листинге 8.3.

Листинг 8.3. Закрытые переменные класса `FMain`

```
private DataSet collegeDataSet;
private SqlDataAdapter adapter;
private SqlConnection connection;
private SqlCommand command;
```

В теле обработчика события `Load` формы поместите код инициализации переменных и заполнения дерева объектов, приведенный в листинге 8.4.

Листинг 8.4. Обработчик события `Load` формы

```
private void Form1_Load(object sender, EventArgs e)
{
   collegeDataSet = new DataSet();

   collegeDataSet.Tables.Add(CourseEntity);
   collegeDataSet.Tables.Add(FacultyEntity);
   collegeDataSet.Tables.Add(GradeEntity);
   collegeDataSet.Tables.Add(StudentEntity);
   collegeDataSet.Tables.Add(TrainerEntity);

   adapter = new SqlDataAdapter();

   connection = new SqlConnection(Properties.Settings.Default.ConString);

   command = new SqlCommand();
   command.Connection = connection;
   command.CommandType = CommandType.StoredProcedure;

   treeView1.Nodes.Add("College", "College", 0, 0);

   treeView1.Nodes[0].Nodes.Add(CourseEntity, CourseEntity, 1, 1);
   treeView1.Nodes[0].Nodes.Add(FacultyEntity, FacultyEntity, 1, 1);
   treeView1.Nodes[0].Nodes.Add(TrainerEntity, TrainerEntity, 1, 1);
   treeView1.Nodes[0].Nodes.Add(GradeEntity, GradeEntity, 1, 1);
   treeView1.Nodes[0].Nodes.Add(StudentEntity, StudentEntity, 1, 1);

   treeView1.ExpandAll();
}
```

Для элемента управления `TreeView` создайте обработчик события `AfterSelect`. Он будет срабатывать при выборе узла в дереве объектов и динамически подключать разные наборы данных из нашей базы данных `College`. Код обработчика события `AfterSelect` представлен в листинге 8.5.

Листинг 8.5. Обработчик события `AfterSelect` дерева объектов

```
private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
{
   if (treeView1.SelectedNode.Parent != null)
   {
      string entityName = treeView1.SelectedNode.Name;
      DataTable table = collegeDataSet.Tables[entityName];

      // построение запроса для хранимых процедур
      command.CommandText = entityName + "_Select";
      adapter.SelectCommand = command;

      // заполнение данными
      table.Rows.Clear();
      adapter.Fill(table);

      // закрываем отображение лишних полей
      SetHiddenFields(table);

      dgvData.DataSource = table;
      lbRowCount.Text = String.Format("{0} records", table.Rows.Count);
   }
}
```

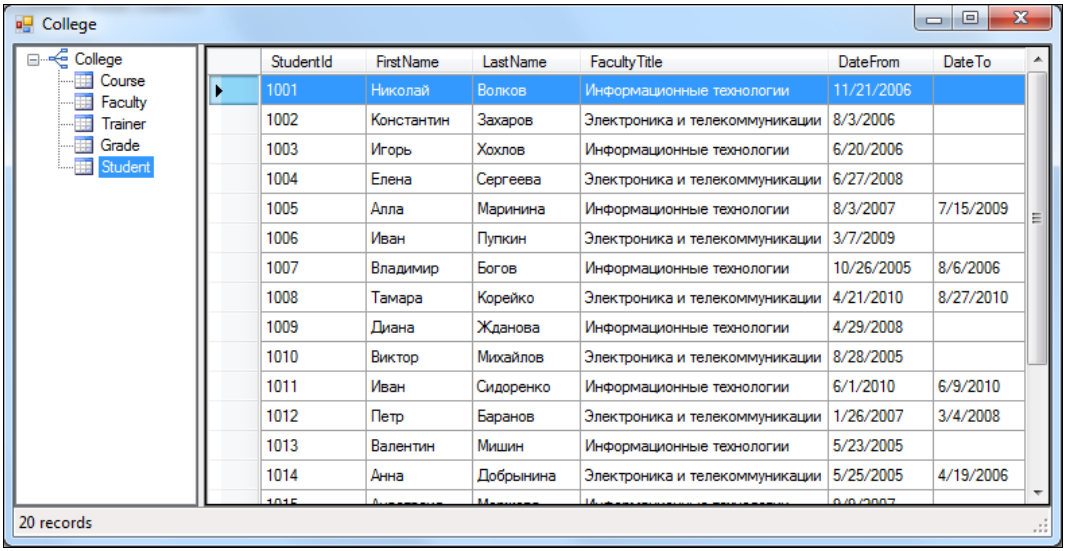
В программе напишем метод `SetHiddenFields()`, который закрывает ненужные для пользователя поля, например в объекте `Student` это поле идентификатора внешнего ключа `FacultyId`. Пользователю необходимо только поле с названием факультета, и нет необходимости выводить его идентификатор. Код метода `SetHiddenFields()` представлен в листинге 8.6.

Листинг 8.6. Метод `SetHiddenFields()`

```
private void SetHiddenFields(DataTable table)
{
   switch (table.TableName)
   {
      case CourseEntity:
         table.Columns["TrainerId"].ColumnMapping = MappingType.Hidden;
         table.Columns["FacultyId"].ColumnMapping = MappingType.Hidden;
         break;
      case FacultyEntity:
         break;
      case GradeEntity:
         table.Columns["StudentId"].ColumnMapping = MappingType.Hidden;
         table.Columns["CourseId"].ColumnMapping = MappingType.Hidden;
         break;
```

```
        case StudentEntity:
            table.Columns["FacultyId"].ColumnMapping = MappingType.Hidden;
            break;
        case TrainerEntity:
            break;
    }
}
```

Скомпилируйте и запустите приложение. Его внешний вид показан на рис. 8.3. Теперь мы можем полностью просматривать все объекты в нашей базе данных.



StudentId	FirstName	LastName	FacultyTitle	DateFrom	DateTo
1001	Николай	Волков	Информационные технологии	11/21/2006	
1002	Константин	Захаров	Электроника и телекоммуникации	8/3/2006	
1003	Игорь	Хохлов	Информационные технологии	6/20/2006	
1004	Елена	Сергеева	Электроника и телекоммуникации	6/27/2008	
1005	Алла	Маринина	Информационные технологии	8/3/2007	7/15/2009
1006	Иван	Пупкин	Электроника и телекоммуникации	3/7/2009	
1007	Владимир	Богов	Информационные технологии	10/26/2005	8/6/2006
1008	Тамара	Корейко	Электроника и телекоммуникации	4/21/2010	8/27/2010
1009	Диана	Жданова	Информационные технологии	4/29/2008	
1010	Виктор	Михайлов	Электроника и телекоммуникации	8/28/2005	
1011	Иван	Сидоренко	Информационные технологии	6/1/2010	6/9/2010
1012	Петр	Баранов	Электроника и телекоммуникации	1/26/2007	3/4/2008
1013	Валентин	Мишин	Информационные технологии	5/23/2005	
1014	Анна	Добрынина	Электроника и телекоммуникации	5/25/2005	4/19/2006

Рис. 8.3. Приложение для просмотра базы данных `College`

Однако пока остается одна проблема — названия полей объектов базы данных имеют не очень понятные для пользователя имена. Далее в этой главе мы покажем, как можно решить эту проблему.

Реализация отображения при выборке данных

Имена полей, хранящихся в базе данных объектов, часто не годятся для нормального восприятия их пользователями. Например, в нашем приложении все поля на английском языке и имеют непонятные для рядового пользователя сокращения.

В ADO.NET можно реализовать отображение на основе объектов `DataTableMapping` и `DataColumnMapping`. Отображение устанавливает взаимно-однозначную трансляцию между именами столбцов результирующего набора и именами объектов `DataColumn`.

При заполнении DataSet адаптер данных просматривает свое свойство TableMappings, чтобы определить, указал ли разработчик правила отображения. По умолчанию свойство TableMappings пусто, так что имена столбцов, используемые в базе данных, применяются и в соответствующих объектах DataTable.

Для использования отображений ADO.NET необходимо создать объект DataTableMapping. Этот объект позволяет выполнять отображение между двумя именами одной и той же таблицы, а кроме того, содержит свойство ColumnMappings — коллекцию объектов DataColumnMapping, выполняющих отображения имен различных столбцов таблицы. После создания этого объекта и добавления в него всех нужных отображений столбцов сам объект можно занести в свойство TableMappings объекта DataAdapter.

У класса DataTableMapping есть конструктор, принимающий две строки. Первая строка задает имя исходной таблицы и чувствительна к регистру букв. Это имя должно совпадать с именем таблицы, используемой в процессе заполнения или обновления, которое выполняет объект DataAdapter. Если не указать имя исходной таблицы, то объект DataAdapter назначает стандартное имя "Table". Второй параметр содержит имя объекта DataTable в DataSet. Объект DataTableMapping содержит коллекцию ColumnMappings, в нее должны входить все столбцы, которые требуется отобразить из базы данных в DataSet. Вот пример создания объекта DataTableMapping:

```
DataTableMapping tableMap = new DataTableMapping("", CourseName);
```

После создания объекта DataTableMapping в нем надо заполнить коллекцию ColumnMappings, которая должна содержать объекты DataColumnMapping. Объект DataColumnMapping содержит отношение между столбцом в базе данных и столбцом в DataSet. Для его создания нужно указать две строки: первая строка задает имя столбца в источнике данных, а вторая — имя столбца, видимое в DataSet:

```
DataColumnMapping colMap = new DataColumnMapping(
   "CourseTitle", "Название курса"));
tableMap.ColumnMappings.Add(colMap)
```

Чтобы реализовать отображение в нашем приложении, добавьте код из листинга 8.7 в обработчик события Load формы.

> ***ПРИМЕЧАНИЕ***
>
> Полный код приложения можно найти на диске, прилагаемом к книге, в каталоге Chapter08\College.v3.

Листинг 8.7. Создание отображения

```
DataTableMapping mapCourse = new DataTableMapping("", CourseName);
mapCourse.ColumnMappings.Add(new DataColumnMapping(
   "CourseId", "№"));
mapCourse.ColumnMappings.Add(new DataColumnMapping(
   "CourseTitle", "Название курса"));
mapCourse.ColumnMappings.Add(new DataColumnMapping(
   "TrainerFullName", "Имя и фамилия преподавателя"));
```

```
mapCourse.ColumnMappings.Add(new DataColumnMapping(
   "FacultyTitle", "Факультет"));

DataTableMapping mapFaculty = new DataTableMapping("", FacultyName);
mapFaculty.ColumnMappings.Add(new DataColumnMapping(
   "FacultyId", "№"));
mapFaculty.ColumnMappings.Add(new DataColumnMapping(
   "FacultyTitle", "Факультет"));

DataTableMapping mapGrade = new DataTableMapping("", GradeName);
mapGrade.ColumnMappings.Add(new DataColumnMapping(
   "GradeId", "№"));
mapGrade.ColumnMappings.Add(new DataColumnMapping(
   "StudentFullName", "Имя и фамилия студента"));
mapGrade.ColumnMappings.Add(new DataColumnMapping(
   "CourseTitle", "Название курса"));
mapGrade.ColumnMappings.Add(new DataColumnMapping(
   "GradeValue", "Оценка"));
mapGrade.ColumnMappings.Add(new DataColumnMapping(
   "Date", "Дата экзамена"));

DataTableMapping mapStudent = new DataTableMapping("", StudentName);
mapStudent.ColumnMappings.Add(new DataColumnMapping(
   "StudentId", "Перс. код"));
mapStudent.ColumnMappings.Add(new DataColumnMapping(
   "FirstName", "Имя"));
mapStudent.ColumnMappings.Add(new DataColumnMapping(
   "LastName", "Фамилия"));
mapStudent.ColumnMappings.Add(new DataColumnMapping(
   "FacultyTitle", "Факультет"));
mapStudent.ColumnMappings.Add(new DataColumnMapping(
   "DateFrom", "Дата поступления"));
mapStudent.ColumnMappings.Add(new DataColumnMapping(
   "DateTo", "Дата выпуска"));

DataTableMapping mapTrainer = new DataTableMapping("", TrainerName);
mapTrainer.ColumnMappings.Add(new DataColumnMapping(
   "TrainerId", "№"));
mapTrainer.ColumnMappings.Add(new DataColumnMapping(
   "FirstName", "Имя"));
mapTrainer.ColumnMappings.Add(new DataColumnMapping(
   "LastName", "Фамилия"));
mapTrainer.ColumnMappings.Add(new DataColumnMapping(
   "FacultyTitle", "Факультет"));
mapTrainer.ColumnMappings.Add(new DataColumnMapping(
   "DateFrom", "Дата приема"));
mapTrainer.ColumnMappings.Add(new DataColumnMapping(
   "DateTo", "Дата увольнения"));
```

```
adapter.TableMappings.Add(mapCourse);
adapter.TableMappings.Add(mapFaculty);
adapter.TableMappings.Add(mapGrade);
adapter.TableMappings.Add(mapStudent);
adapter.TableMappings.Add(mapTrainer);
```

Теперь при запуске приложения названия столбцов отображаются на понятном для пользователя языке (рис. 8.4).

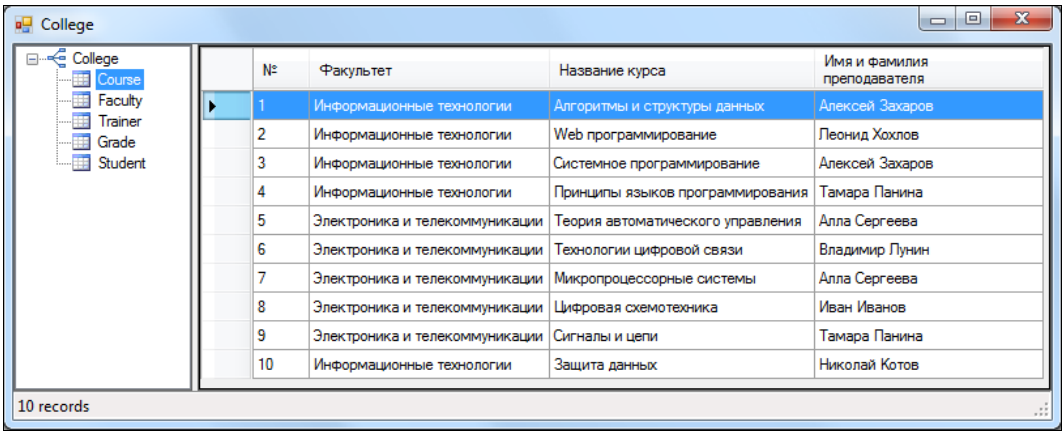


Рис. 8.4. Приложение с использованием отображения для названий полей данных

Объект *DataSet* со строгим контролем типов

Объекты `DataSet`, с которыми мы работали ранее в этой главе, не являлись строго типизированными. Это означает, что каждый элемент данных в таком `DataSet` имеет тип `object`. Чтобы выполнить над таким элементом данных действия, специфичные для некоторого типа (например `string`, `DateTime` и др.), этот объект следует преобразовать в соответствующий тип. При использовании нестрого типизированных переменных есть большая вероятность появления ошибок приведения типов. Эти проблемы разрешаются с помощью строго типизированных `DataSet`.

Типизированный объект `DataSet` не является встроенным в .NET Framework классом. Это класс, сгенерированный средой разработки Visual Studio, являющийся непосредственным потомком класса `DataSet` и позволяющий уточнять свойства и методы на основе указанной схемы источника данных. Типизированный `DataSet` содержит собственные классы для объектов `DataTable` и `DataRow`.

В целом объекты `DataSet` со строгим контролем типов упрощают процесс разработки, облегчая написание кода для доступа и изменения данных.

Примечание

Кроме строго типизированных `DataSet`, существуют более современные технологии доступа к данным, которые мы будем рассматривать в *главах 9* и *10*. Однако с ис-

пользованием строго типизированных `DataSet` было написано огромное количество приложений, которые необходимо сопровождать, модифицировать их код, поэтому вам необходимо иметь представление об этой технологии доступа к данным.

Создание источника данных

Создать строго типизированный `DataSet` для заданного источника данных в среде Visual Studio очень легко. Для этого даже не надо писать ни одной строчки кода. Для примера такой программы с использованием строго типизированного `DataSet` создадим новое приложение Windows Forms и откроем форму в дизайнере форм.

Примечание

Полный код приложения можно найти на диске, прилагаемом к книге, в каталоге Chapter08\College.v4.

Затем откроем окно **Data Sources**. Если это окно закрыто, выберите в главном меню пункт **Data | Show Data Sources** и в открывшемся окне **Data Sources** (по умолчанию оно отображается на левой панели инструментов) нажмите на ссылку **Add New Data Source**. При этом откроется первая страница мастера **Data Source Configuration Wizard**, которая предлагает выбрать тип источника данных. Выберите в качестве источника данных **Database** (рис. 8.5).

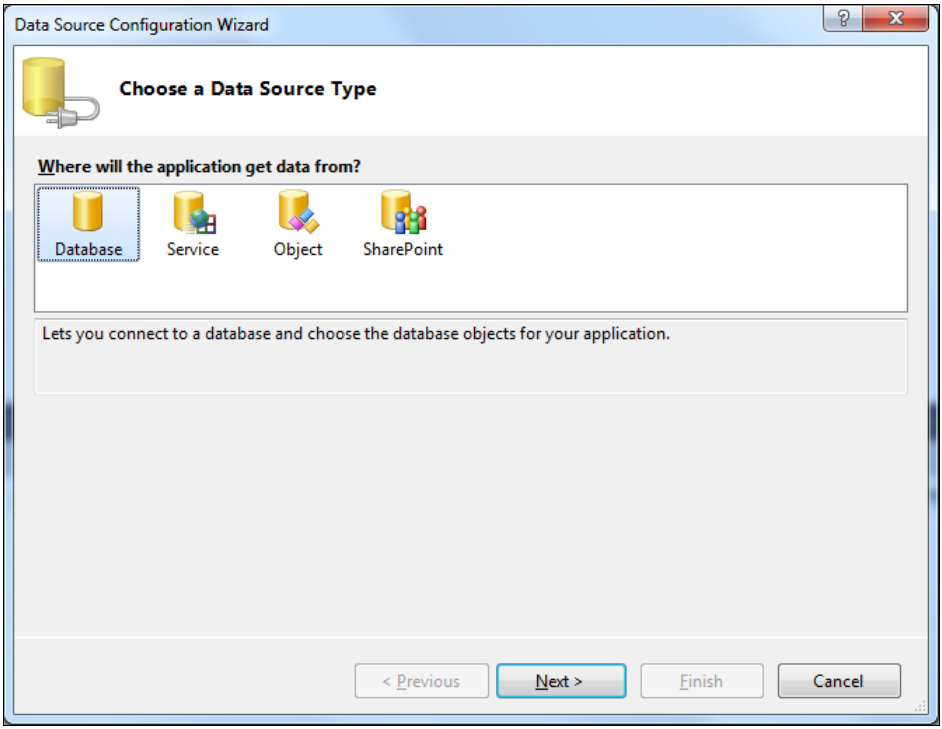


Рис. 8.5. Страница мастера для выбора типа источника данных

Следующая страница мастера предлагает вам выбрать модель базы данных. Выберите **Dataset** (рис. 8.6).

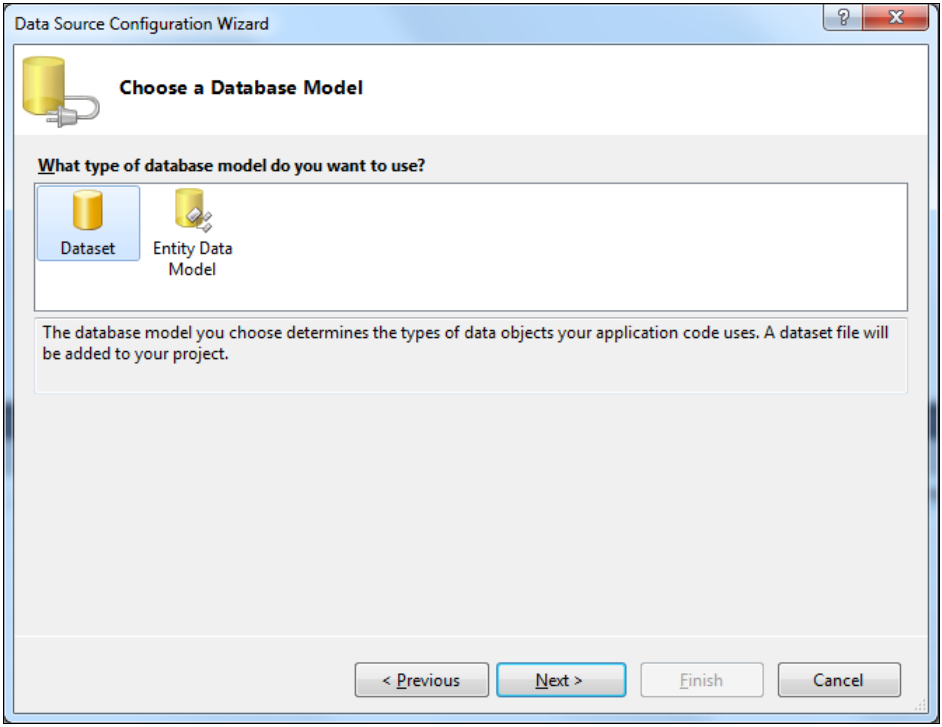


Рис. 8.6. Страница мастера для модели данных

Дальше вам предлагается создать подключение к базе данных (рис. 8.7). Подключение к базе данных должно присутствовать в списке, если вы выполняли примеры из *глав 6* и *7*. Если его нет — создайте новое подключение, нажав кнопку **New Connection**. Процесс создания нового подключения был описан в предыдущей главе.

Далее выберите нужные объекты в базе данных `College`. В нашем случае выберите таблицы (рис. 8.8).

После нажатия кнопки **Finish** в окне **Data Sources** появится новый источник данных — типизированный `DataSet` с именем `CollegeDataSet`, содержащий пять объектов, соответствующих таблицам в базе данных (рис. 8.9).

При создании источника данных к проекту также добавятся новые файлы, содержащие код для `CollegeDataSet`. Если открыть файл CollegeDataSet.xsd в дизайнере, вы увидите все таблицы со связями из базы данных `College` (рис. 8.10).

Затем перетащите мышью один из объектов, например `Student`, на форму. При этом среда сгенерирует код для `DataGridView` с привязкой к объекту `Student`, элемент `BindingNavigator`, расположенный на панели инструментов, и дополнительные объекты для привязки данных: экземпляры классов `CollegeDataSet`, `BindingSource`, типизированный `DataAdapter` и `TableAdapterManager` (рис. 8.11).

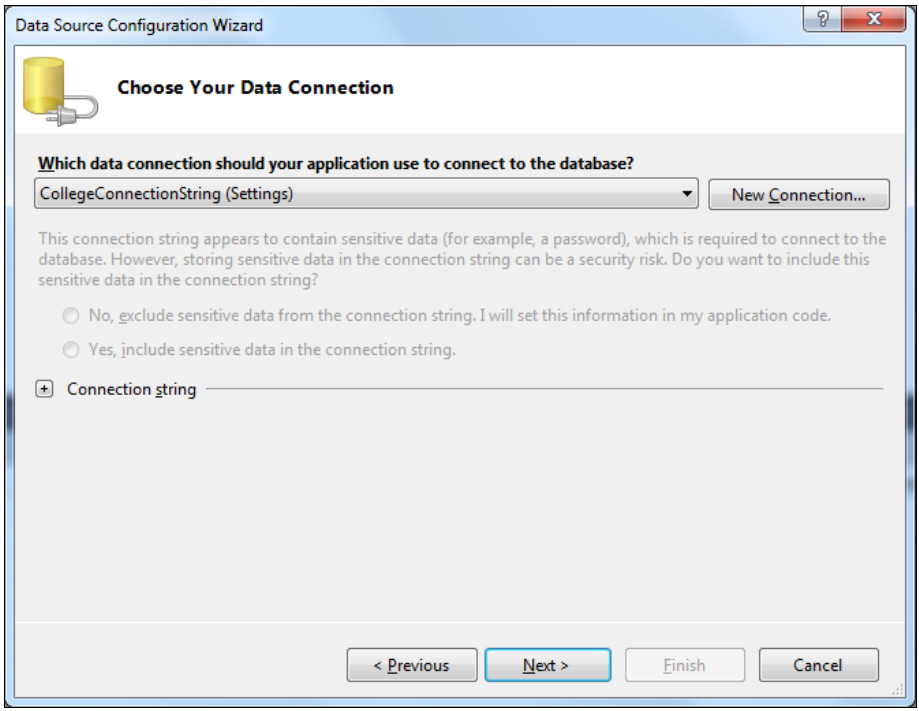


Рис. 8.7. Страница мастера для создания соединения с базой данных

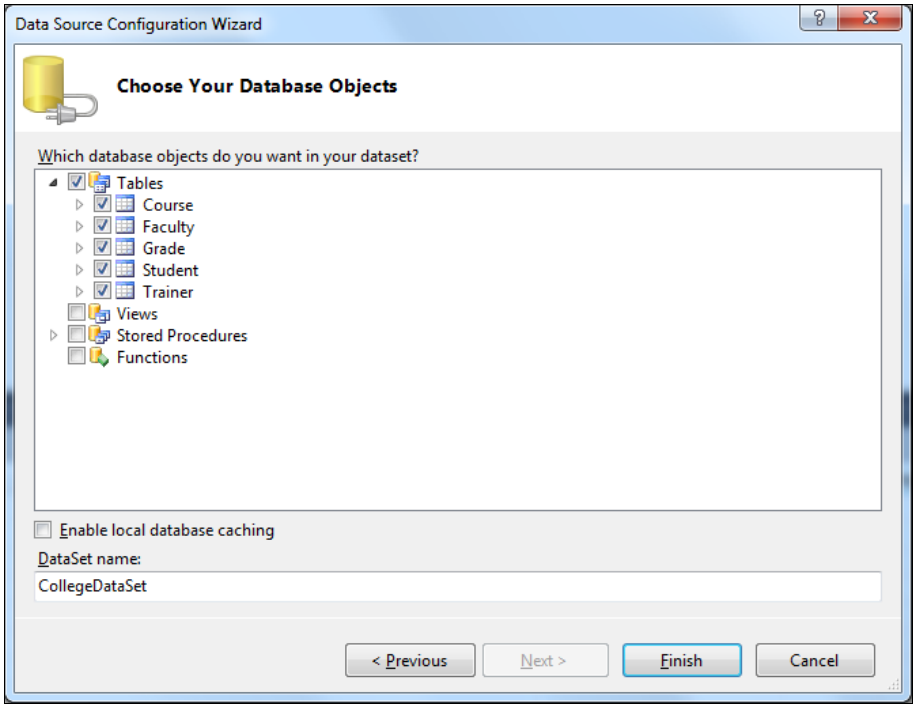


Рис. 8.8. Выбор объектов в базе данных `College`

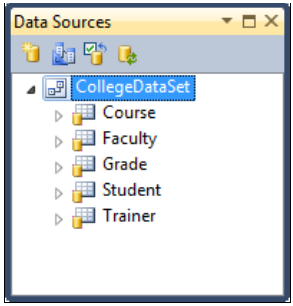


Рис. 8.9. Окно **Data Sources** с источником данных `College`

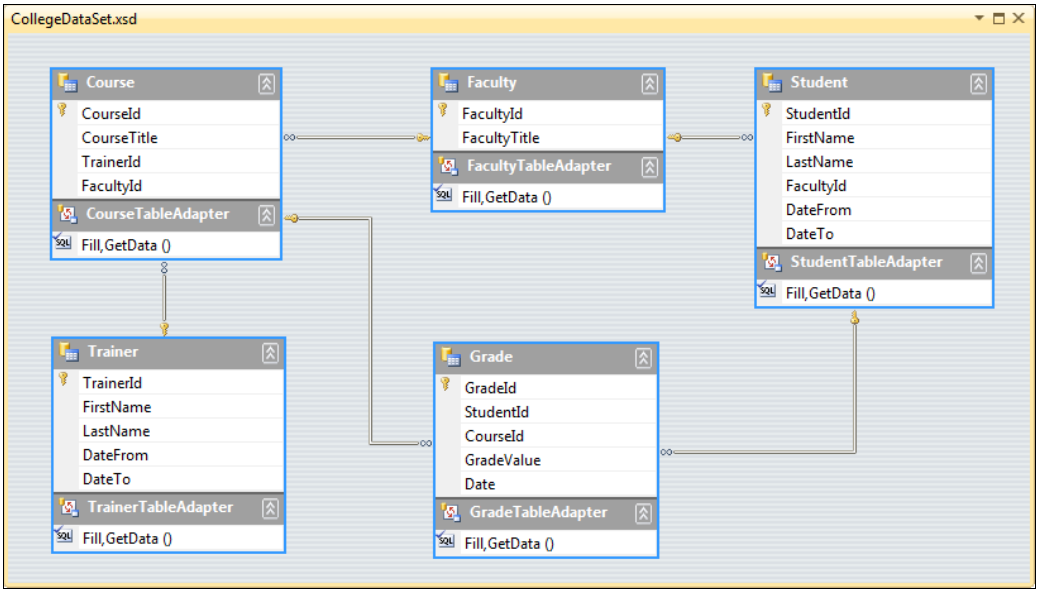


Рис. 8.10. Схема типизированного `DataSet` для базы данных `College`

Если открыть файл с классом формы `FMain`, мы увидим, что среда Visual Studio сгенерировала код, представленный в листинге 8.8.

Листинг 8.8. Код класса `FMain` главной формы приложения

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

namespace College
{
   public partial class FMain : Form
```

```
    {
        public FMain()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.studentTableAdapter.Fill(this.collegeDataSet.Student);
            this.studentTableAdapter.Fill(this.collegeDataSet.Student);
        }

        private void studentBindingNavigatorSaveItem_Click(object sender,
                EventArgs e)
        {
            this.Validate();
            this.studentBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.collegeDataSet);
        }
    }
}
```

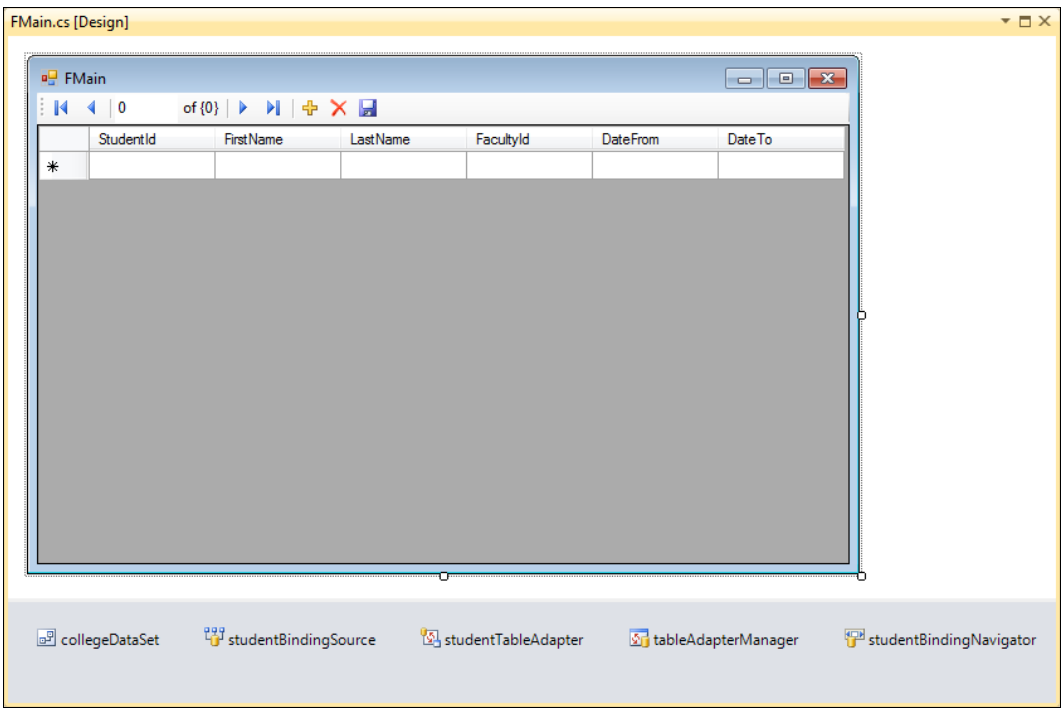
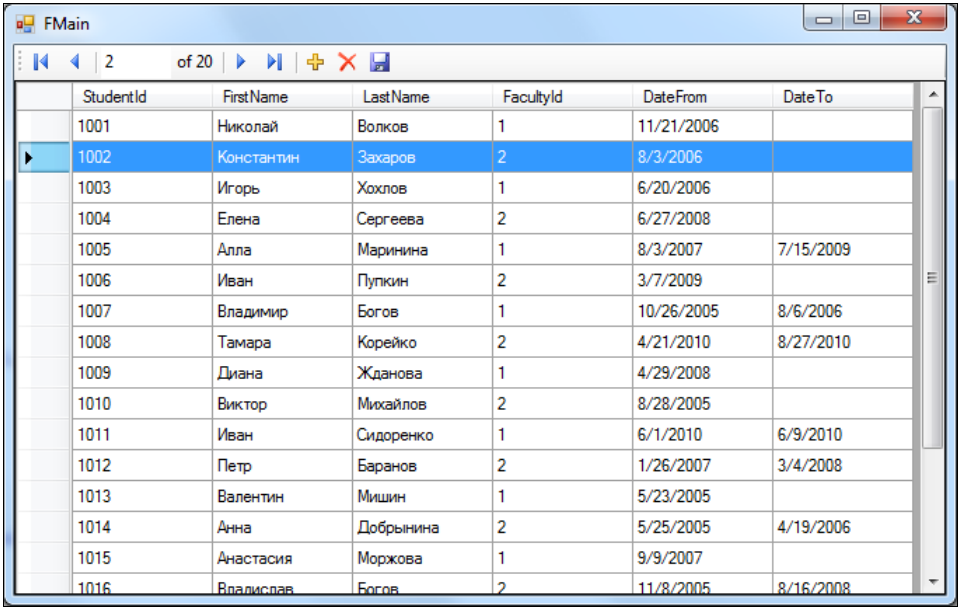


Рис. 8.11. Дизайн главной формы приложения

Запустив приложение, вы увидите, что оно отображает данные из таблицы `Student`, причем мы не написали ни строчки кода, код за нас был полностью сгенерирован средой разработки (рис. 8.12).



StudentId	FirstName	LastName	FacultyId	DateFrom	DateTo
1001	Николай	Волков	1	11/21/2006	
1002	Константин	Захаров	2	8/3/2006	
1003	Игорь	Хохлов	1	6/20/2006	
1004	Елена	Сергеева	2	6/27/2008	
1005	Алла	Маринина	1	8/3/2007	7/15/2009
1006	Иван	Пупкин	2	3/7/2009	
1007	Владимир	Богов	1	10/26/2005	8/6/2006
1008	Тамара	Корейко	2	4/21/2010	8/27/2010
1009	Диана	Жданова	1	4/29/2008	
1010	Виктор	Михайлов	2	8/28/2005	
1011	Иван	Сидоренко	1	6/1/2010	6/9/2010
1012	Петр	Баранов	2	1/26/2007	3/4/2008
1013	Валентин	Мишин	1	5/23/2005	
1014	Анна	Добрынина	2	5/25/2005	4/19/2006
1015	Анастасия	Моржова	1	9/9/2007	
1016	Владислав	Богов	2	11/8/2005	8/16/2008

Рис. 8.12. Приложение, отображающее данные таблицы `Student`

Помимо отображения данных, приложение даже позволяет вводить и обновлять данные. Конечно, это не значит, что таким способом вы сможете создавать серьезные промышленные приложения дла работы с данными, для сложных приложений, помимо этого, потребуется еще большая работа руками. Однако этот способ можно применять для быстрого создания простых приложений или программ для тестирования баз данных.

Модификация данных в *DataSet*

Данные, хранящиеся в объектах `DataSet`, можно модифицировать. Разрешается изменять значения существующих объектов `DataRow`, а также удалять и добавлять к `DataSet` новые объекты `DataRow`. Однако модификации не будут реплицированы в базу данных до тех пор, пока они не будут обновлены с помощью объекта `DataAdapter`.

Каждый экземпляр `DataSet` поддерживает две версии этого объекта: текущую, расположенную на клиентском компьютере и содержащую все модификации, и исходную, не менявшуюся со времени заполнения `DataSet`. При вызове метода `Update()` объекта `DataAdapter` на основе исходных значений генерируются команды `UPDATE`, `INSERT` и `DELETE`, необходимые для обновления базы данных.

Каждый объект `DataRow` поддерживает две версии состояния: исходную, содержащую первоначальные значения `DataRow`, и модифицированную. В любой момент

можно отменить (откатить) любые изменения, внесенные в объект DataRow, вызвав метод RejectChanges():

```
row.RejectChanges();
```

Чтобы внести некоторые изменения в объект DataRow, вызовите метод AcceptChanges(). В результате новая версия DataRow запишется поверх исходной.

```
row.AcceptChanges();
```

Если планируется реплицировать модификации в источник данных с помощью объекта DataAdapter, вызывать метод AcceptChanges() до вызова метода DataAdapter.Update() нельзя, иначе объекту DataAdapter не удастся получить доступ к исходной версии данных и сгенерировать корректную команду UPDATE.

Объекты DataTable и DataSet также поддерживают методы RejectChanges() и AcceptChanges(), позволяющие соответственно отменить или принять все изменения, внесенные в DataTable или DataSet.

Обновление базы данных

Обновить базу данных можно, вызвав метод Update() для объекта DataAdapter.

```
myDataAdapter.Update();
```

В результате изменения, внесенные в клиентскую копию данных, копируются в базу данных. При этом можно указать объект DataSet, DataTable или массив объектов DataRows, подлежащий обновлению:

```
myDataAdapter.Update(dataSet);
myDataAdapter.Update(dataTable);
myDataAdapter.Update(rows);
```

Реализация отдельного уровня данных

В промышленных приложениях, как правило, используется многоуровневая архитектура. Применение многоуровневой архитектуры, разделения пользовательского интерфейса, бизнес-логики и слоя доступа к данным позволяет совместное использование ресурсов и данных различными приложениями.

В этой и двух последующих главах мы создадим библиотеки, реализующие уровень данных для базы данных College с использованием различных технологий для доступа к данным, которые нам пригодятся при построении клиентских приложений с использованием ASP.NET, Windows Presentation Foundation, веб-сервисов и других технологий.

В этой главе мы создадим библиотеку доступа данных с использованием типизированного DataSet. Для создания библиотеки слоя данных создадим новый проект **Class Library** и назовем его College.Data (рис. 8.13).

> ***Примечание***
>
> Полный код приложения можно найти на диске, прилагаемом к книге, в каталоге Chapter08\College.Data.

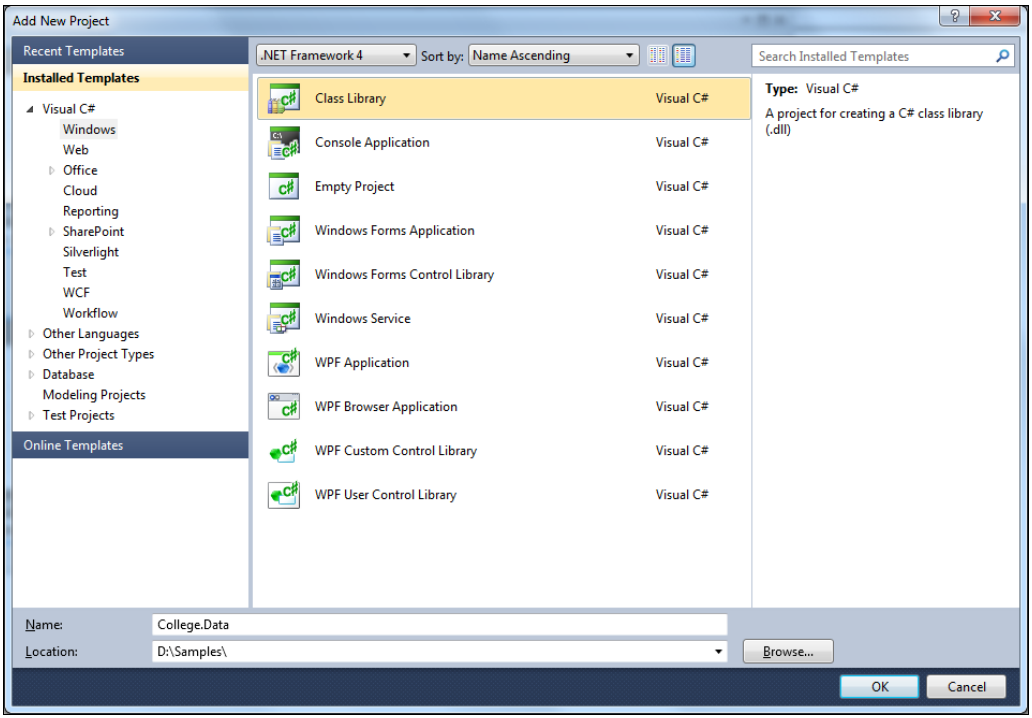


Рис. 8.13. Создание нового проекта **Class Library**

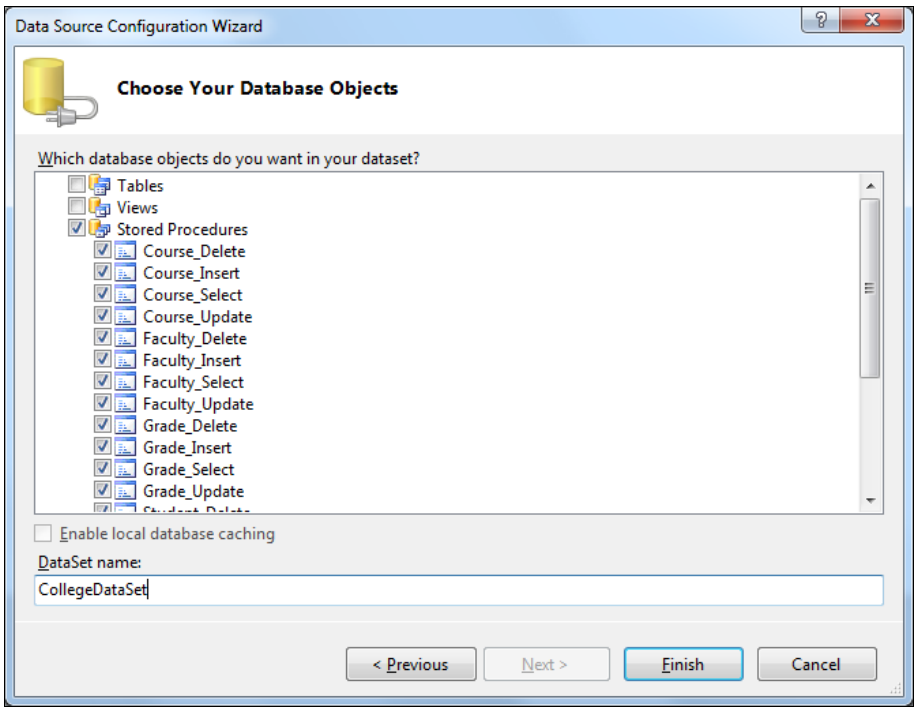


Рис. 8.14. Выбор хранимых процедур в качестве объектов источника данных

В окне **Solution Explorer** в дереве проекта удалите созданный по умолчанию файл Class1.cs. После этого откройте окно **Data Sources** и создайте новый источник данных на основе `DataSet`, как вы это сделали в предыдущем разделе, за исключением того, что в качестве объектов базы данных используйте набор хранимых процедур, а не таблицы, как в прошлый раз (рис. 8.14).

Сгенерированный мастером `DataSet` должен выглядеть в дизайнере так, как показано на рис. 8.15.

Далее мы напишем клиентские приложения, использующие созданную нами библиотеку для доступа к базе данных `College`.

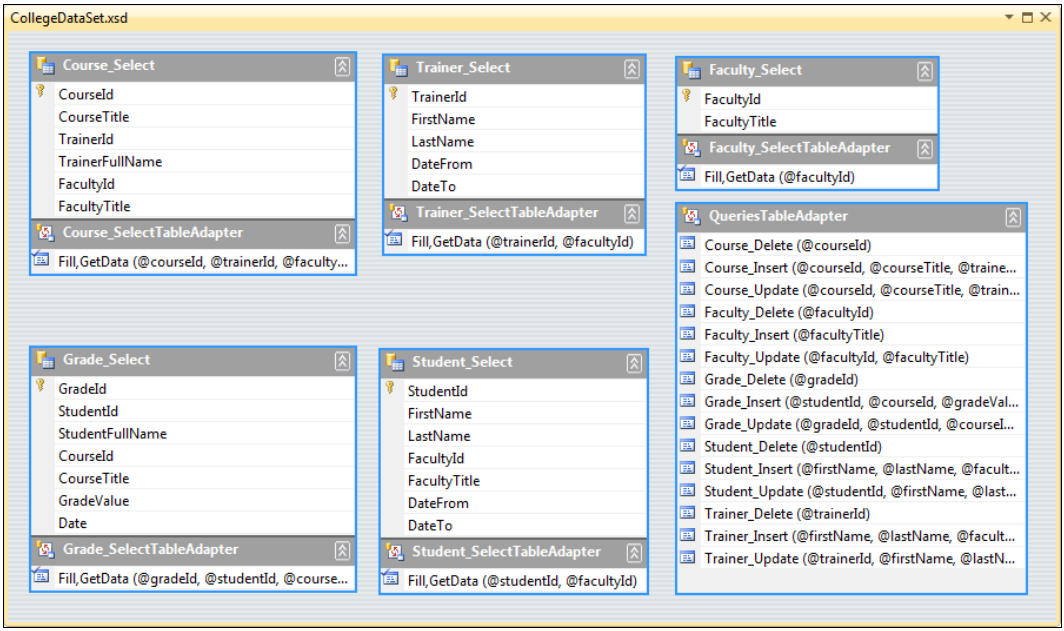


Рис. 8.15. XSD-файл сгенерированного `DataSet`

Динамическое связывание данных в период выполнения

Часто в период выполнения требуется модифицировать источник данных, с которым связан некоторый элемент управления. Или, наоборот, иногда необходимо связать элемент управления с источником данных, причем экземпляр этого элемента управления требуется создавать только во время работы программы. В таких случаях следует выполнять связывание данных программными средствами. Объекты `DataSet` также можно создавать динамически, в программном коде. Данные этого объекта разрешается связывать с элементами пользовательского интерфейса, а также модифицировать и применять для обновления информации в базе данных через объект `DataAdapter`.

Для примера создания динамического связывания данных создадим новый проект Windows Forms, который назовем `College.UI`, и реализуем динамическую привязку данных с элементами управления и `DataSet`. Мы уже создавали подобное приложение в начале главы. Для главной формы приложения вы можете использовать в качестве шаблона форму из этого проекта (см. рис. 8.2).

Примечание

Полный код приложения можно найти на диске, прилагаемом к книге, в каталоге Chapter08\College.UI.v1.

В форму внесем некоторые изменения: вместо элемента `DataGridView` в правой части поместим контейнерный элемент управления `TabControl`. У нас будет приложение типа TDI (Tabbed Document Interface), которое может содержать множество документов с закладками. Дизайн главной формы приложения представлен на рис. 8.16.

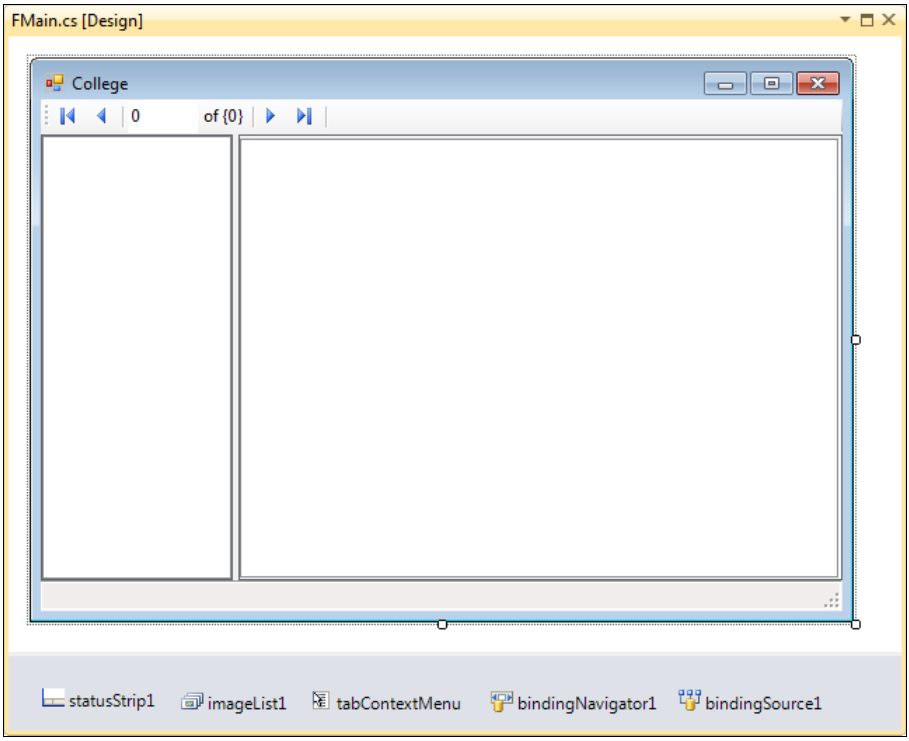


Рис. 8.16. Дизайн главной формы приложения

Для создания окна документа используем пользовательский элемент управления, наследуемый от класса `UserControl`. Элемент управления `UserControl` позволяет создавать элементы управления, которые можно многократно использовать в приложении. Добавим `UserControl` в проект и назовем его `UCGrid` (рис. 8.17).

На поверхности созданного пользовательского элемента управления разместите `DataGridView`.

Задайте для него свойства:

- `Name = dgvData;`
- `DockStyle = DockStyle.Fill;`
- `AllowUserToAddRow = false;`
- `AllowUserToDeleteRow = false;`
- `ReadOnly = true.`

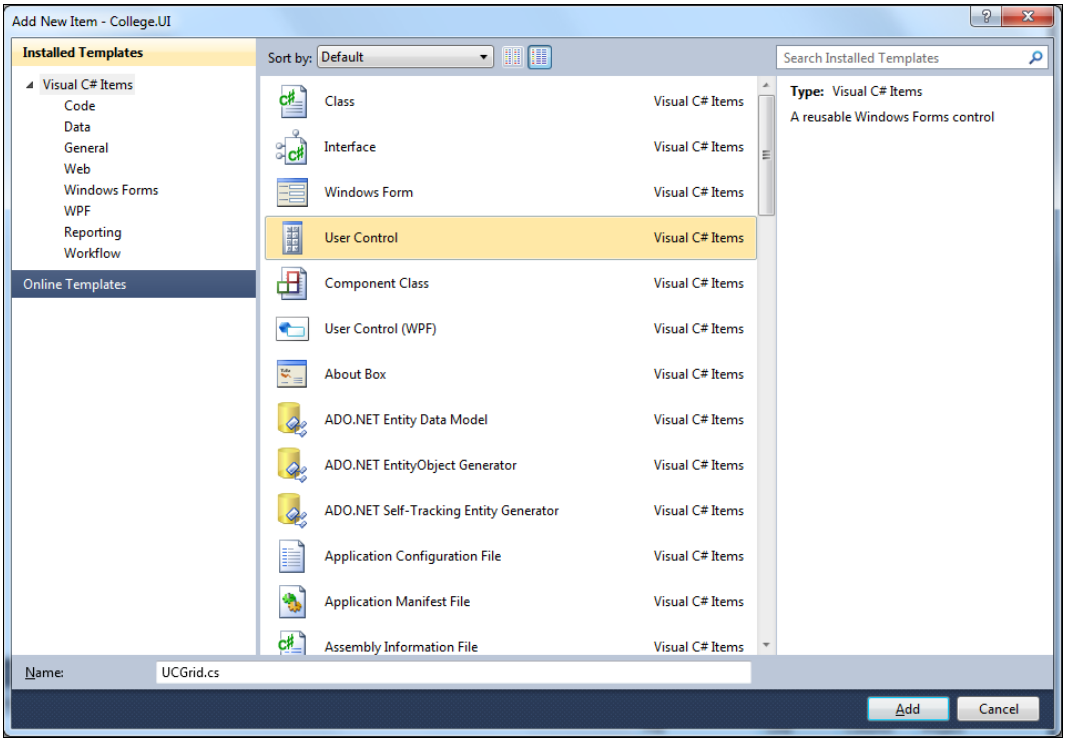


Рис. 8.17. Добавление `UserControl` в проект

В класс `UCGrid` добавьте свойство `DataSource` для доступа к одноименному свойству дочернего `DataGridView`. Код класса `UCGrid` показан в листинге 8.9.

Листинг 8.9. Код класса `UCGrid`

```
using System.Windows.Forms;

namespace College.UI
{
   public partial class UCGrid : UserControl
   {
      public UCGrid()
```

```
        {
            InitializeComponent();
        }

        public object DataSource
        {
            set { dgvData.DataSource = value; }
        }
    }
}
```

Теперь перейдем к главной форме приложения. В класс главной формы приложения добавьте закрытые переменные и константы, представленные в листинге 8.10.

Листинг 8.10. Закрытые константы и переменные класса `FMain`

```
private const string CourseName = "Course";
private const string FacultyName = "Faculty";
private const string GradeName = "Grade";
private const string StudentName = "Student";
private const string TrainerName = "Trainer";

private string currentEntity;
private List<string> displayTabPageNames;

private CollegeDataSet dsCollege;

private Course_SelectTableAdapter daCourse;
private Faculty_SelectTableAdapter daFaculty;
private Grade_SelectTableAdapter daGrade;
private Student_SelectTableAdapter daStudent;
private Trainer_SelectTableAdapter daTrainer;
```

В теле обработчика события `Load` формы напишите код инициализации объектов `DataSet`, адаптеров данных и дерева объектов базы данных, а также привязку навигатора к объекту `BindingSource`, как показано в листинге 8.11.

Листинг 8.11. Обработчик события `Load` формы

```
private void FMain_Load(object sender, EventArgs e)
{
    dsCollege = new CollegeDataSet();

    daCourse = new Course_SelectTableAdapter();
    daFaculty = new Faculty_SelectTableAdapter();
    daGrade = new Grade_SelectTableAdapter();
    daStudent = new Student_SelectTableAdapter();
    daTrainer = new Trainer_SelectTableAdapter();
```

```
    displayTabPageNames = new List<string>();

    treeView1.Nodes.Add("College", "College", 0, 0);

    treeView1.Nodes[0].Nodes.Add(CourseName, CourseName, 1, 1);
    treeView1.Nodes[0].Nodes.Add(FacultyName, FacultyName, 1, 1);
    treeView1.Nodes[0].Nodes.Add(TrainerName, TrainerName, 1, 1);
    treeView1.Nodes[0].Nodes.Add(GradeName, GradeName, 1, 1);
    treeView1.Nodes[0].Nodes.Add(StudentName, StudentName, 1, 1);

    treeView1.ExpandAll();

    tabControl1.ContextMenuStrip = tabContextMenu;
    bindingNavigator1.BindingSource = bindingSource1;
}
```

Основной код надо будет написать в обработчике события выбора узла в дереве объектов базы данных, который показан в листинге 8.12.

Листинг 8.12. Обработчик события выбора узла в дереве объектов

```
private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
{
    if (treeView1.SelectedNode.Parent != null)
    {
        currentEntity = treeView1.SelectedNode.Name;
        switch (currentEntity)
        {
            case CourseName:
                bindingSource1.DataSource = dsCollege.Course_Select;
                daCourse.Fill(dsCollege.Course_Select, null, null, null);
                break;
            case FacultyName:
                bindingSource1.DataSource = dsCollege.Faculty_Select;
                daFaculty.Fill(dsCollege.Faculty_Select, null);
                break;
            case TrainerName:
                bindingSource1.DataSource = dsCollege.Trainer_Select;
                daTrainer.Fill(dsCollege.Trainer_Select, null, null);
                break;
            case GradeName:
                bindingSource1.DataSource = dsCollege.Grade_Select;
                daGrade.Fill(dsCollege.Grade_Select, null, null, null);
                break;
            case StudentName:
                bindingSource1.DataSource = dsCollege.Student_Select;
                daStudent.Fill(dsCollege.Student_Select, null, null);
```

```
            break;
      }

      UCGrid grid;
      if (!displayTabPageNames.Contains(currentEntity))
      {
         displayTabPageNames.Add(currentEntity);
         TabPage page = new TabPage(currentEntity);
         page.Name = currentEntity;

         grid = new UCGrid();
         grid.Name = "dgv" + currentEntity;
         grid.Parent = page;
         grid.Dock = DockStyle.Fill;

         tabControl1.TabPages.Add(page);
      }
      else
      {
         grid = (UCGrid)tabControl1.TabPages[currentEntity].Controls[0];
      }
      grid.DataSource = bindingSource1;
      tabControl1.SelectedTab = tabControl1.TabPages[currentEntity];
   }
}
```

В обработчике события выбора закладки элемента управления `TabControl` напишите код, представленный в листинге 8.13.

Листинг 8.13. Обработчик события выбора закладки элемента управления `TabControl`

```
private void tabControl1_SelectedIndexChanged(object sender, EventArgs e)
{
   if (tabControl1.SelectedTab != null)
   {
      currentEntity = tabControl1.SelectedTab.Name;
      treeView1.Focus();
      treeView1.SelectedNode = treeView1.Nodes[0].Nodes[currentEntity];
   }
   else
   {
      currentEntity = "";
   }
}
```

В обработчике событий контекстного меню для управления закладками элемента управления `TabControl` напишите код, представленный в листингах 8.14—8.16.

Листинг 8.14. Обработчик события пункта меню *Close*

```
private void closeToolStripMenuItem1_Click(object sender, EventArgs e)
{
   displayTabPageNames.Remove(currentEntity);
   TabPage page = tabControl1.TabPages[currentEntity];
   tabControl1.TabPages.Remove(page);
}
```

Листинг 8.15. Обработчик события пункта меню *Close All But This*

```
private void closeAllButThisToolStripMenuItem_Click(object sender, EventArgs e)
{
   for (int i = displayTabPageNames.Count — 1; i >= 0; i--)
   {
      string name = displayTabPageNames[i];
      if (currentEntity != name)
      {
         tabControl1.TabPages.Remove(tabControl1.TabPages[name]);
         displayTabPageNames.Remove(name);
      }
   }
}
```

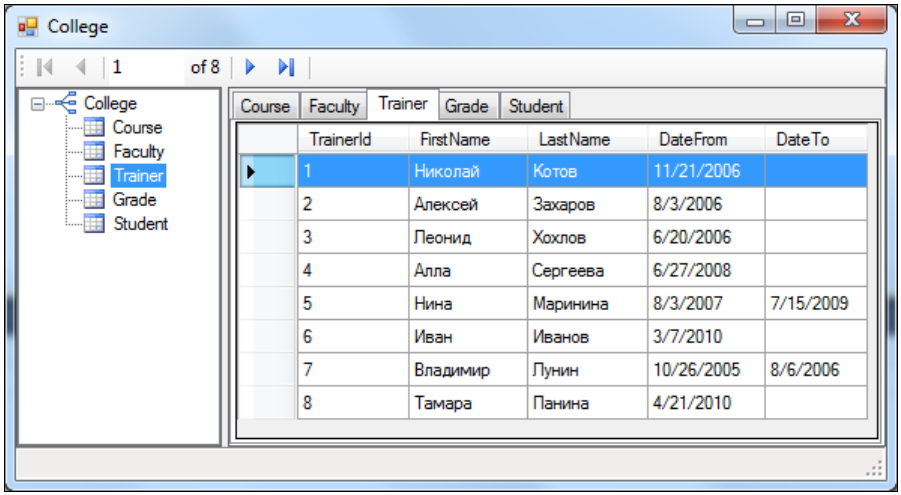


Рис. 8.18. Приложение для работы с данными

Листинг 8.16. Обработчик события пункта меню *Close All*

```
private void closeAllToolStripMenuItem_Click(object sender, EventArgs e)
{
   tabControl1.TabPages.Clear();
```

```
    displayTabPageNames.Clear();
    treeView1.SelectedNode = null;
    bindingSource1.DataSource = null;
}
```

В результате у нас получилось TDI-приложение, теперь можно управлять открытием и закрытием закладок с документами (почти как IDE Visual Studio!), отображающими информацию из базы данных `College` (рис. 8.18).

Сортировка и фильтрация данных

После заполнения объекта `DataSet` часто требуется работать с подмножеством его данных, загруженных в память. Такое подмножество выделяют при помощи объекта `DataView`. Объект `DataView` фактически выступает в роли фильтра для объекта `DataTable`, выбирающего из объекта `DataTable` некоторые данные и предоставляющего их элементам управления, связанным с этим объектом `DataTable`. Объект `DataView` поддерживает методы для сортировки и фильтрации данных, а также позволяет обновлять представляемый им объект `DataTable`.

Создание объекта *DataView*

Чтобы создать объект `DataView`, необходимо передать ссылку на объект `DataTable`, для которого он будет выполнять фильтрацию содержимого, например:

```
DataTable dataTable
...
DataView view = new DataView(dataTable);
```

Этот код создает объект `DataView`, представляющий данные объекта типа `DataTable`. Чтобы задать критерии фильтрации упорядочения данных, следует установить соответствующие свойства объекта `DataView`. Можно создать объект `DataView`, не связанный с объектом `DataTable`, но в этом случае `DataView` будет недоступен для связывания, пока не будет определено его свойство `Table`:

```
DataView view = new DataView();
view.Table = dataTable;
```

Объекты `DataView` позволяют выполнять сортировку и фильтрацию данных, которые они представляют, при этом критерии сортировки и фильтрации разрешается изменять в период выполнения.

Сортировка данных

Чтобы упорядочить данные, установите свойство `Sort` объекта `DataView`, записав в него строку, интерпретируемую как определение правил сортировки данных. Такая строка содержит имя поля, по которому выполняется сортировка:

```
view.Sort = "StudentId";
```

Чтобы задать упорядочение по нескольким полям, разделите их имена запятыми, например:

```
view.Sort = "LastName, FacultyTitle";
```

Чтобы упорядочить поле по убыванию, добавьте к нему ключевое слово `DESC`:

```
view.Sort = " LastName DESC";
```

При изменении критериев сортировки объекта `DataView` обновляются все связанные с ним элементы управления.

Давайте теперь добавим в приложение возможность сортировки данных. Создадим в проекте новую форму, назовем ее `FSort`, добавим на нее следующие элементы управления:

- `ListBox` для отображения списка полей объекта (`Name: lbFields`);
- `ListBox` для списка сортируемых полей (`Name: lbSort`);
- `Button` для наполнения списка `lbSort` выбранными полями (`Name: bAdd`);
- `Button` для удаления полей из списка `lbSort` (`Name: bDelete`);
- группу из двух элементов `RadioButton` для выбора порядка сортировки (`rbAsc` и `rbDesc`);
- две кнопки — **OK** и **Cancel** — для закрытия диалогового окна, которым зададим свойства `DialogResult`: `DialogResult.OK` и `DialogResult.Cancel` соответственно.

Дизайн формы показан на рис. 8.19.

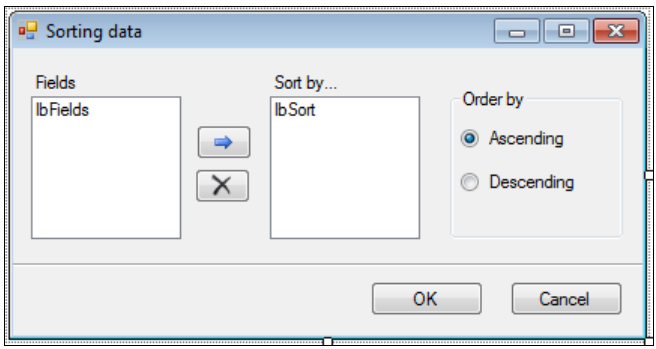


Рис. 8.19. Дизайн формы сортировки данных

Примечание

Полный код приложения можно найти на диске, прилагаемом к книге, в каталоге Chapter08\College.UI.v2.

Для формы создадим обработчик события `Load`. Также для кнопок добавления полей в список, удаления полей из списка и кнопки **OK** создадим обработчики событий `Click`. При нажатии кнопки **OK** в обработчике этого события формируется строковое выражение из имен полей в списке `lbSort`, разделенных запятыми, и, в зависимости от выбора переключателя, добавляется ключевое слово `ASC` или `DESC`.

Полный код класса формы для сортировки данных приведен в листинге 8.17.

Листинг 8.17. Код класса формы `FSort`

```
using System;
using System.Text;
using System.Data;
using System.Windows.Forms;

namespace College.UI
{
   public partial class FSort : Form
   {
      DataView view;

      public FSort(DataView dv)
      {
         InitializeComponent();
         view = dv;
      }

      private void FSort_Load(object sender, EventArgs e)
      {
         for (int i = 0; i < view.Table.Columns.Count; i++)
         {
            lbFields.Items.Add(view.Table.Columns[i].ColumnName);
         }
         lbFields.SelectedIndex = 0;
      }

      private void bAdd_Click(object sender, EventArgs e)
      {
         object item = lbFields.SelectedItem;
         if (!lbSort.Items.Contains(item))
         {
            lbSort.Items.Add(item);
            lbSort.SelectedItem = item;

            if (lbFields.SelectedIndex < lbFields.Items.Count — 1)
            {
               lbFields.SelectedIndex++;
            }
         }
      }

      private void bDelete_Click(object sender, EventArgs e)
      {
         lbSort.Items.Remove(lbSort.SelectedItem);
      }
```

```
      private void bOK_Click(object sender, EventArgs e)
      {
         StringBuilder sortBy = new StringBuilder();
         for (int i = 0; i < lbSort.Items.Count; i++)
         {
            sortBy.Append(lbSort.Items[i]);
            if (i < lbSort.Items.Count – 1)
            {
               sortBy.Append(", ");
            }
         }

         string orderBy = "ASC";
         if (rbDesc.Checked)
         {
            orderBy = "DESC";
         }
         view.Sort = String.Format("{0} {1}", sortBy, orderBy);
      }
   }
}
```

В класс `UCGrid` внесем дополнительные усовершенствования. Добавим свойство типа `DataView` для привязки объекта `DataView` из главной формы приложения, как показано в листинге 8.18.

Листинг 8.18. Свойство `DataView` класса `UCGrid`

```
public DataView DataView
{
   set { view = value; }
}
```

Для вызова формы сортировки данных добавьте контекстное меню `ContextMenuGrid`. В контекстном меню создайте пункт **Sort** и заведите для него обработчик события `Click`. В теле обработчика события `Click` пункта меню напишите код, представленный в листинге 8.19.

Листинг 8.19. Обработчик события `OnClick` пункта меню *Sort*

```
private void sortToolStripMenuItem_Click(object sender, EventArgs e)
{
   FSort sort = new FSort(view);
   sort.ShowDialog();
}
```

Скомпилируйте и запустите приложение. Теперь вы можете задавать различные условия сортировки для выбранного вами объекта базы данных `College`. В примере

на рис. 8.20 показана сортировка по возрастанию для двух полей: сначала по полю `StudentFullName`, а затем по полю `CourseName`.

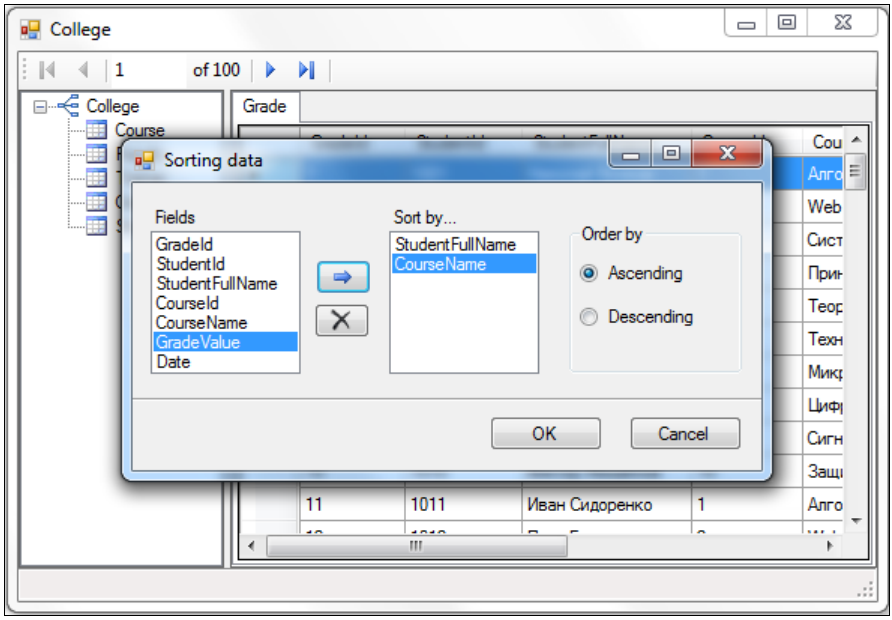


Рис. 8.20. Внешний вид приложения с сортировкой данных

Фильтрация данных

Чтобы задать критерий фильтрации данных, надо записать в свойство `RowFilter` объекта `DataView` строку, интерпретируемую как выражение, которое определяет подмножество записей. Например, можно выбрать только строки с заданным значением в некотором поле:

```
view.RowFilter = "FacultyId = 1";
```

Выражения, записываемые в `RowFilter`, должны подчиняться синтаксису SQL: строковые литералы необходимо заключать в одинарные кавычки, а даты — между символов "#". Для более сложных выражений применяют логические операторы `AND`, `OR` и `NOT`, `IN`, `LIKE`, арифметические операторы, конкатенацию, сравнение.

Фильтры часто жестко задают в программном коде для определенных заранее критериев фильтрации, однако можно создать обобщенный фильтр, позволяющий работать с полями любого типа и задавать различные критерии фильтрации.

Добавим в проект новую форму, назовем ее `FFilter`. Задайте для нее свойства:

- `Name: FFilter`
- `Text: "Custom Filter:"`
- `FormBorderStyle: FixedSingle`
- `StartPosition: CenterParent`

Добавьте на форму два элемента управления `ComboBox` с именами `cbCondition` — для выбора условия фильтрации и `cbData` — для выбора или написания выражения в фильтре. Также добавим две кнопки **OK** и **Cancel**, которым зададим свойства `DialogResult`: `DialogResult.OK` и `DialogResult.Cancel` соответственно. Дизайн формы показан на рис. 8.21.

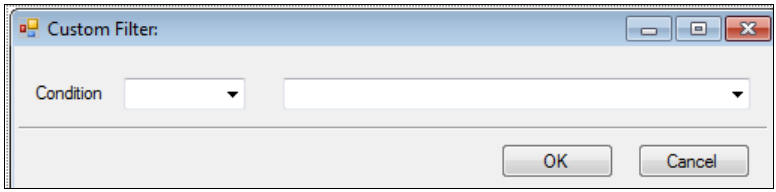


Рис. 8.21. Дизайн формы фильтра

Для списка `cbCondition` в свойстве `Items` создайте коллекцию условий фильтрации, как показано на рис. 8.22.

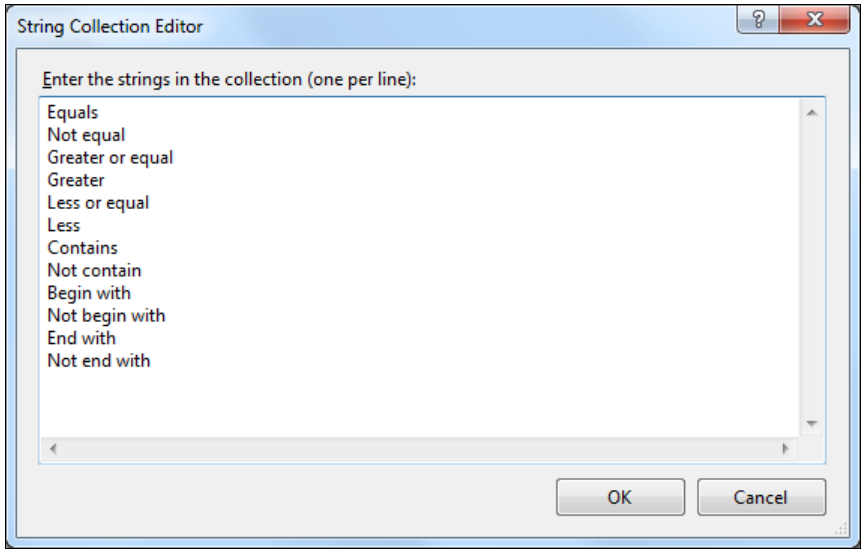


Рис. 8.22. Коллекция значений для выпадающего списка

Конструктор формы будет принимать в качестве параметра название столбца, по которому будет происходить фильтрация и экземпляр `DataView`. Код конструктора показан в листинге 8.20.

Листинг 8.20. Закрытые переменные и конструктор класса `FFilter`

```
DataView view;
string columnName;

public FFilter(string colName, DataView dv)
```

```
{
   InitializeComponent();
   view = dv;
   columnName = colName;

   cbData.DataSource = view;
   cbData.DisplayMember = columnName;
   cbData.SelectedIndex = -1;
   this.Text += columnName;
}
```

В теле обработчика события `Click` кнопки **OK** реализуем динамическое построение запроса к набору данных, находящихся в `DataView`. Запрос строится в зависимости от введенных пользователем условий фильтрации и аргументов.

Реализация обработчика события `Click` кнопки **OK** представлена в листинге 8.21.

Листинг 8.21. Реализация обработчика события `Click` кнопки *OK*

```
private void bOK_Click(object sender, EventArgs e)
{
   string exp = "";
   string arg = cbData.Text;
   switch (cbCondition.Text)
   {
      case "Equals":
         exp = String.Format("{0} = '{1}'", columnName, arg);
         break;
      case "Not equal":
         exp = String.Format("{0} <> '{1}'", columnName, arg);
         break;
      case "Greater or equal":
         exp = String.Format("{0} >= '{1}'", columnName, arg);
         break;
      case "Greater":
         exp = String.Format("{0} > '{1}'", columnName, arg);
         break;
      case "Less or equal":
         exp = String.Format("{0} <= '{1}'", columnName, arg);
         break;
      case "Less":
         exp = String.Format("{0} < '{1}'", columnName, arg);
         break;
      case "Contains":
         exp = String.Format("{0} LIKE '%{1}%'", columnName, arg);
         break;
```

```
      case "Not contain":
         exp = String.Format("{0} NOT LIKE '%{1}%'", columnName, arg);
         break;
      case "Begin with":
         exp = String.Format("{0} LIKE % '{1}%'", columnName, arg);
         break;
      case "Not begin with":
         exp = String.Format("{0} NOT LIKE '{1}%'", columnName, arg);
         break;
      case "End with":
         exp = String.Format("{0} LIKE '%{1}'", columnName, arg);
         break;
      case "Not end with":
         exp = String.Format("{0} LIKE '%{1}'", columnName, arg);
         break;
   }
   view.RowFilter = exp;
}
```

Для вызова формы фильтрации данных добавьте в контекстное меню `ContextMenuGrid` новый пункт **Filter** и создайте для него обработчик события `Click`. Однако, в отличие от вызова формы сортировки данных, нам надо для вызова формы фильтрации данных знать имя столбца, по которому будет задаваться фильтр. Поэтому в программе необходимо определить, на каком столбце элемента управления `DataGridView` пользователь щелкнул мышью.

Для этого в классе `DataGridView` определено событие `CellMouseUp`. Это событие происходит, когда пользователь щелкает кнопкой на одной из ячеек и отпускает любую кнопку мыши. С помощью этого события мы можем определить индекс столбца, который выбрал пользователь, и сохранить его в переменную, как представлено в листинге 8.22.

Листинг 8.22. Обработчик события `CellMouseUp` элемента управления `DataGridView`

```
// переменная уровня класса
int cIndex;
...
private void dgvData_CellMouseUp(object sender, DataGridViewCellMouseEventArgs e)
{
   cIndex = e.ColumnIndex;
}
```

Теперь переменную `cIndex` мы можем использовать для определения имени столбца и передать его в качестве параметра конструктору формы `FFilter`. В теле обработчика события `Click` пункта меню **Filter** напишите код, представленный в листинге 8.23.

Листинг 8.23. Обработчик события `OnClick` пункта меню *Filter*

```
private void filterToolStripMenuItem_Click(object sender, EventArgs e)
{
   FFilter filter = new FFilter(
         view.Table.Columns[cIndex].ColumnName, view);
   filter.ShowDialog();
}
```

Скомпилируйте и запустите приложение. Теперь вы можете задавать различные условия фильтрации или сортировки любого набора записей (рис. 8.23).

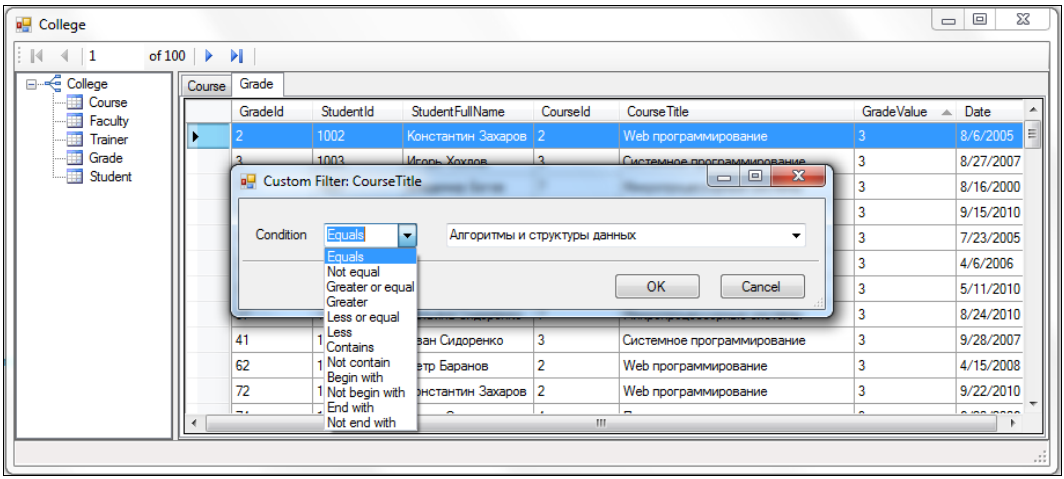


Рис. 8.23. Приложение с выборочной фильтрацией данных

Объект *DataSet* и XML

Автономные объекты данных `DataSet` и `DataTable` хорошо совместимы и обеспечивают удобную работу с XML. `DataSet` и `DataTable` допускают легкое преобразование в XML и обратно.

В библиотеке .NET Framework существует класс `XmlDataDocument`, который может хранить `DataSet` и его XML-представление, а также синхронизировать их. Он может хранить данные и выдавать их при необходимости как в виде `XmlDocument`, так и в виде `DataSet`.

Рассмотрим работу `XmlDataDocument` на примере приложения, которое сохраняет объект из базы данных в XML-файл, а также может его открывать и отображать в форматированном виде. Добавим в наше приложение еще один пользовательский элемент управления, аналогичный созданному ранее `UCGrid`. Назовем его `UCBrowser`, разместим на нем единственный элемент управления `WebBrowser` и зададим ему свойство `DockStyle`, равное `DockStyle.Fill`. Этот элемент будет выводить красивые XML-документы с подсветкой синтаксиса и раскрывающимися узлами.

Примечание

Полный код приложения можно найти на диске, прилагаемом к книге, в каталоге Chapter08\College.UI.v3.

В коде класса создайте единственное открытое свойство `Browser` для доступа к элементу управления `WebBrowser`. Код класса `UCBrowser` представлен в листинге 8.24.

Листинг 8.24. Код класса `UCBrowser`

```
using System;
using System.Windows.Forms;

namespace College
{
   public partial class UCBrowser : UserControl
   {
      public UCBrowser()
      {
         InitializeComponent();
      }

      public WebBrowser Browser
      {
         get { return webBrowser1; }
      }
   }
}
```

На главной форме приложения разместите компоненты `OpenFileDialog` и `SaveFileDialog` для открытия и сохранения XML-документов в файл. Добавьте также на панель инструментов кнопки для открытия и сохранения XML-документов. Внешний вид окна с новыми элементами управления в дизайнере форм показан на рис. 8.24.

Для диалогов открытия и сохранения файлов в обработчике события `Load` формы задайте следующие свойства, представленные в листинге 8.25. (Можете их установить и в дизайнере форм, если вам так удобнее.)

Листинг 8.25. Задание свойств диалогов открытия и сохранения XML-файлов

```
saveFileDialog1.Filter = "xml files (*.xml)|*.xml|All files (*.*)|*.*";
saveFileDialog1.FilterIndex = 1;
saveFileDialog1.RestoreDirectory = true;

openFileDialog1.Filter = "xml files (*.xml)|*.xml|All files (*.*)|*.*";
openFileDialog1.FilterIndex = 1;
openFileDialog1.RestoreDirectory = true;
```

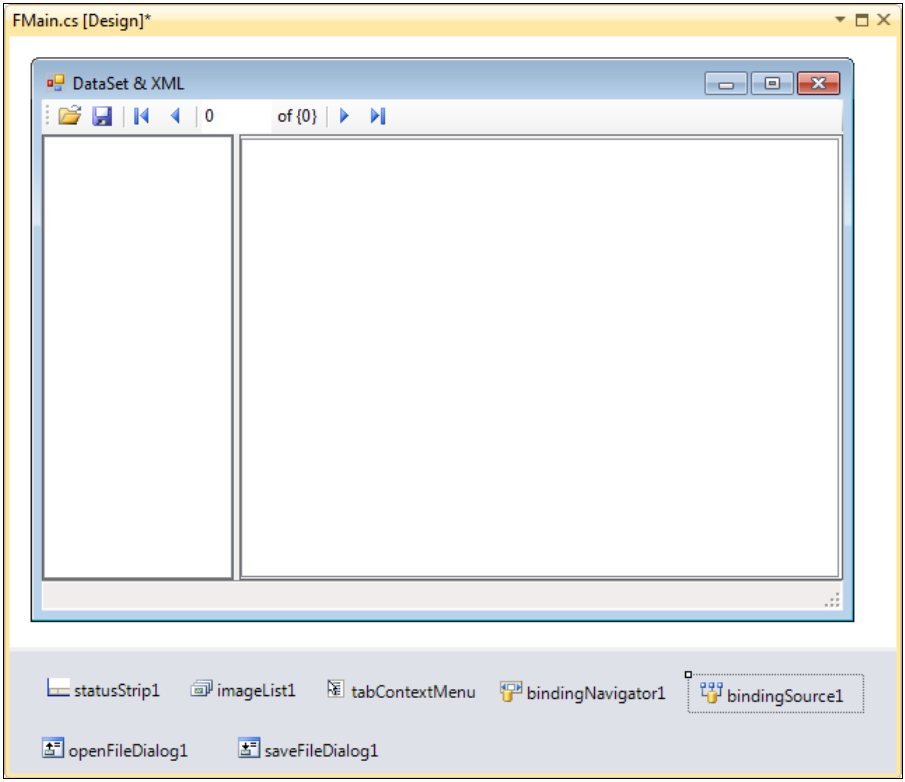


Рис. 8.24. Дизайн главной формы приложения

Для кнопок открытия и сохранения XML-документов создайте обработчики событий и напишите в них код, представленный в листингах 8.26 и 8.27.

Листинг 8.26. Обработчик события кнопки открытия XML-файла

```
private void openToolStripMenuItem_Click(object sender, EventArgs e)
{
   OpenFileDialog openDialog = new OpenFileDialog();
   openDialog.Filter = "xml files (*.xml)|*.xml|All files (*.*)|*.*";
   openDialog.FilterIndex = 1;
   openDialog.RestoreDirectory = true;

   if (openDialog.ShowDialog() == DialogResult.OK)
   {
      FileInfo fi = new FileInfo(openDialog.FileName);
      displayTabPageNames.Add(fi.Name);
      TabPage page = new TabPage(fi.Name);
      page.Name = fi.Name;

      UCBrowser browser = new UCBrowser();
      browser.Browser.Navigate(openDialog.FileName);
```

```
        browser.Dock = DockStyle.Fill;
        page.Controls.Add(browser);

        tabControl1.TabPages.Add(page);
        tabControl1.SelectedTab = tabControl1.TabPages[page.Name];
    }
}
```

Листинг 8.27. Обработчик события кнопки сохранения XML-файла

```
private void saveToolStripMenuItem_Click(object sender, EventArgs e)
{
    SaveFileDialog saveDialog = new SaveFileDialog();
    saveDialog.Filter = "xml files (*.xml)|*.xml|All files (*.*)|*.*";
    saveDialog.FilterIndex = 1;
    saveDialog.RestoreDirectory = true;

    UCGrid grid = (UCGrid)tabControl1.SelectedTab.Controls[0];
    DataTable dt = grid.DataView.ToTable();
    saveDialog.FileName = dt.TableName;

    if (saveDialog.ShowDialog() == DialogResult.OK)
    {
        dt.WriteXml(saveDialog.FileName);
    }
}
```

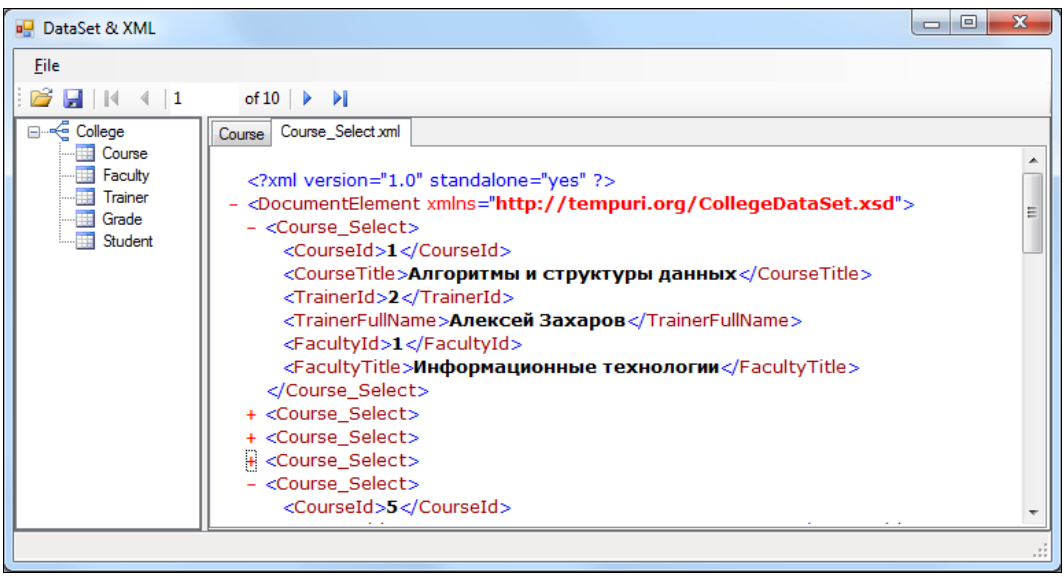


Рис. 8.25. Преобразование содержимого документа в XML

Скомпилируйте и запустите приложение. Вы можете задать различные условия фильтрации или сортировки любого набора данных `College` и сохранять их в файл. Откройте один из документов, например `Course`, и сохраните его в XML-файле, а затем откройте этот файл в программе. При этом появится новая вкладка, отображающая XML-документ. Вы должны увидеть данные, приведенные на рис. 8.25.

Резюме

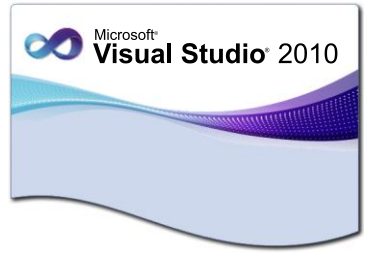
В этой главе мы изучили работу с автономными данными и использование строго типизированного объекта `DataSet`. Одно из главных различий между компонентом чтения данных `DataReader`, описанного в прошлой главе, и объекта `DataSet` заключается в том, что `DataReader` потребляет меньше памяти, чем `DataSet`. В зависимости от объема данных и доступной памяти, разница в производительности вследствие более экономного использования памяти может быть огромной.

Однако при решении вопроса о том, должно ли приложение использовать `DataReader` или объект `DataSet`, следует внимательно проанализировать функциональность, которая необходима для разрабатываемого приложения. Объект `DataSet` предназначен для эффективного выполнения следующих задач:

- локальное кэширование данных в приложении для последующей обработки. Если нужно только считывать результаты запроса, класс `DataReader` подходит наилучшим образом;
- удаленное взаимодействие с данными между уровнями или от веб-службы XML;
- динамическое взаимодействие с данными, например привязка к элементу управления Windows Forms или комбинирование и связывание данных из нескольких источников;
- выполнение интенсивной обработки, не требующей открытого соединения с источником данных, что освобождает соединение для использования другими клиентами.

Если широкая функциональность, которую предоставляет типизированный объект `DataSet`, не нужна для работы приложения, можно повысить его производительность, используя объект `DataReader` для возврата данных. Хотя класс `DataAdapter` использует класс `DataReader` для заполнения содержимого `DataSet`, путем применения объекта `DataReader` можно повысить производительность приложения, т. к. будет экономиться память, которую потреблял бы объект `DataSet`, и избежать обработки, необходимой для создания и заполнения содержимого `DataSet`.

Кроме того, в следующих главах мы рассмотрим более современные технологии доступа к данным, использование которых при создании приложений для работы с данными более эффективно, чем применение строго типизированных `DataSet`.



ГЛАВА 9

LINQ

LINQ (Language Integrated Query, язык интегрированных запросов) — это технология, которая позволяет разработчикам формировать в программном коде запросы, основанные на наборах, без использования дополнительного языка запросов. Можно писать запросы LINQ к различным перечислимым источникам данных, таким как хранимые в памяти структуры данных, XML-документы, базы данных SQL и объекты `DataSet`. Несмотря на то, что перечисленные источники данных реализованы различными способами, во всех них используется одинаковый синтаксис и языковые конструкции. Из-за того что запросы могут быть сформированы на языке программирования, нет необходимости использовать другой язык запросов, внедренный в виде строковых литералов, которые не могут быть проверены компилятором.

Встраивание запросов в язык программирования позволяет программистам, использующим среду Visual Studio, быть более продуктивными, предоставляя им проверку синтаксиса и соответствия типов во время компиляции и возможности технологии IntelliSense. Эти функции уменьшают затраты на отладку запросов и поиск ошибок.

Источники данных LINQ

LINQ ориентирован на запросы, возвращающие набор объектов, единственный объект или подмножество полей из объекта или множества объектов. В LINQ этот возвращенный набор объектов называется *последовательностью* (sequence). Большинство последовательностей LINQ имеют тип `IEnumerable<T>`, где `T` — тип данных объектов, находящихся в последовательности. Например, если у вас есть последовательность целых чисел, они должны храниться в переменной типа `IEnumerable<int>`.

Всего существует пять источников данных LINQ.

- *LINQ to Object* — позволяет выполнять запросы к массивам и находящимся в памяти коллекциям данных. Стандартные операции запросов — это статические методы класса `System.Linq.Enumerable`, которые вы используете для создания запросов LINQ to Objects.

- *LINQ to XML* — предназначен для работы с XML-документами. Microsoft не только добавила необходимые библиотеки XML для работы с LINQ, но также восполнила недостатки стандартной модели XML DOM, существенно облегчив работу с XML-документами.
- *LINQ to DataSet* — за последнее время было разработано множество приложений с доступом к данным на основе `DataSet`. Технология LINQ to DataSet предоставляет широкие оптимизированные возможности создания запросов через объект `DataSet` и модификации приложений, в которых были использованы типизированные `DataSet`.
- *LINQ to SQL* — позволяет через запросы LINQ работать с базой данных Microsoft SQL Server. Технология LINQ to SQL позволяет непосредственно запрашивать схемы баз данных SQL Server.
- *LINQ to Entities* — позволяет приложениям взаимодействовать с реляционными данными из баз данных как с объектами. LINQ to Entities изучается в следующей главе, где будет рассматриваться Entity Data Model.

Передача данных в объекты памяти часто бывает сложной и способствует совершению ошибок. Поставщик `LINQ`, реализованный в LINQ to DataSet и LINQ to SQL, преобразует исходные данные в данные, основанные на коллекции объектов, реализующих интерфейс `IEnumerable`. Программист всегда видит данные как коллекции `IEnumerable` как при запросе, так и при обновлении. Для написания запросов к этим коллекциям предоставлена полная поддержка технологии `IntelliSense`.

LINQ to Objects

Отчасти то, что делает LINQ настолько мощным и удобным в применении, заключается в его тесной интеграции с языком C#. Вы можете применять обычные коллекции и массивы, адаптированные к существующим классам. Это значит, что вы можете получить все преимущества запросов LINQ с минимальными модификациями существующего кода или вообще без них.

Одним из привлекательных для разработчиков средств LINQ является SQL-подобный синтаксис, доступный в LINQ-запросах. Выражения запросов позволяют запросам LINQ принимать форму, подобную SQL, лишь с несколькими небольшими отличиями.

Чтобы выполнить запрос LINQ, не обязательно использовать выражения запросов. Альтернативой является применение стандартной точечной нотации C# с вызовом методов на объектах и классах.

Основные операции запросов LINQ

Все операции запроса LINQ состоят из трех различных действий.

1. Получение источника данных.
2. Создание запроса.
3. Выполнение запроса.

Большинство стандартных операций запросов представляют собой расширяющие методы в статическом классе `System.Linq.Enumerable` и прототипированы `IEnumerable<T>` в качестве первого аргумента.

Методы стандартных операций запросов класса `System.Linq.Enumerable`, которые не являются расширяющими методами, — это просто статические методы, которые должны быть вызваны на классе `System.Linq.Enumerable`. Комбинация этих методов стандартных операций запросов дает вам возможность выполнять сложные запросы данных на последовательности `IEnumerable<T>`.

Запрос указывает, какую информацию нужно извлечь из источника или источников данных. При необходимости запрос также указывает способ сортировки, группировки и формирования этих сведений перед возвращением. Запрос хранится в переменной запроса и инициализируется выражением запроса.

Основные операции, выполняемые в запросе LINQ:

- получение источника данных;
- фильтрация;
- упорядочение;
- группировка;
- соединение;
- выбор (проецирование).

Хотя многие из стандартных операций запросов LINQ возвращают объект `IEnumerable<T>`, и мы воспринимаем `IEnumerable<T>` как последовательность, на самом деле операции запроса не возвращают последовательность в момент их вызова. Вместо этого операции возвращают объект, который при перечислении порождает очередной элемент последовательности. Во время перечисления возвращенного объекта запрос выполняется и порожденный элемент помещается в выходную последовательность.

Давайте для примера рассмотрим запрос LINQ к обычному массиву строк. Создадим небольшое приложение Windows Forms с единственной формой `FMain`, дизайн окна которого представлен на рис. 9.1.

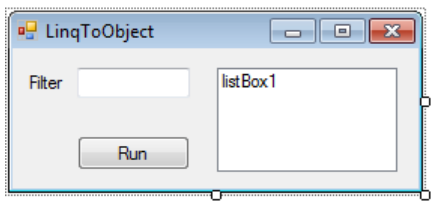


Рис. 9.1. Дизайн формы приложения

В текстовом поле **Filter** будем вводить критерий фильтрации массива. В классе формы `FMain` напишите код, представленный в листинге 9.1.

Примечание

Полный код приложения на прилагаемом к книге диске находится в каталоге Chapter09\LinqToObjects.v1.

Листинг 9.1. Программа с запросами к массиву строк

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

namespace LinqToObjects
{
   public partial class FMain: Form
   {
      // Массив данных
      private string[] names = new string[] {
         "Волков", "Захаров", "Хохлов", "Сергеева", "Маринина",
         "Пупкин", "Богов", "Корейко", "Жданова", "Михайлов",
         "Сидоренко", "Баранов", "Мишин", "Добрынина", "Моржова",
         "Антипов", "Симонова", "Антохин", "Агапова", "Бородин" };

      public FMain()
      {
         InitializeComponent();
      }

      // Обработчик события кнопки Run
      private void bRun_Click(object sender, EventArgs e)
      {
         listBox1.Items.Clear();

         var items = from s in names
                     where s.StartsWith(tbFilter.Text)
                     select s;

         foreach (var item in items)
         {
            listBox1.Items.Add(item);
         }
      }
   }
}
```

В этом примере источником данных является массив строк, который неявно поддерживает интерфейс `IEnumerable<T>`. Это значит, что к нему можно выполнять запросы с использованием LINQ. Запрос выполняется в операторе `foreach`, и оператору `foreach` требуется интерфейс `IEnumerable` или `IEnumerable<T>`. Типы, которые поддерживают `IEnumerable<T>` или производные от него интерфейсы, такие как универсальный интерфейс `IQueryable<T>`, называются *запрашиваемыми типами*.

Запрос, использующий операцию `where`, на самом деле не выполняется, когда выполняется строка, содержащая запрос. Вместо этого возвращается объект `items`. И только во время перечисления элементов возвращенного объекта в цикле `foreach` запрос `where` выполняется. Результат выполнения запроса представлен на рис. 9.2.

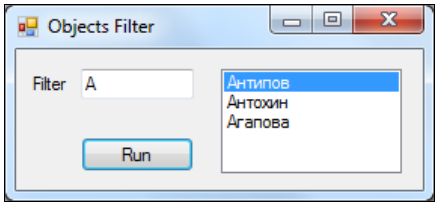


Рис. 9.2. Выполнение запроса с фильтром для массива строк

Этот пример представлял собой отложенный запрос. Однако есть способ получить результат выполнения запроса без цикла для перечисления элементов, например, преобразовав запрос в список типа `List<T>` с помощью метода `ToList()`. Попробуйте этот вариант, переписав код в обработчике события `bRun_Click()`, как показано в листинге 9.2.

Листинг 9.2. Обработчик события кнопки *Run*

```
private void bRun_Click(object sender, EventArgs e)
{
   var items = from s in names
               where s.StartsWith(tbFilter.Text)
               select s;
   List<string> listNames = items.ToList();
   // Заполняем список результатами запроса
   listBox1.DataSource = listNames;
}
```

Запросы к коллекциям

Аналогично запросам к статическому массиву простых типов осуществляется построение запроса к коллекциям. Давайте немного усложним предыдущий пример, использовав в качестве источника данных коллекцию `Dictionary<int, string>`. Код реализации программы с запросом представлен в листинге 9.3.

> ***Примечание***
>
> Полный код приложения на прилагаемом к книге диске находится в каталоге Chapter09\LinqToObjects.v2.

Листинг 9.3. Программа с реализацией запросов к коллекции

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

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

namespace LinqToObjects
{
   public partial class FMain : Form
   {
      private Dictionary<int, string> models;

      public FMain()
      {
         InitializeComponent();
         InitializeData();
      }

      private void InitializeData()
      {
         // Заполняем Dictionary данными
         models = new Dictionary<int, string>();
         models.Add(1, "Алгоритмы и структуры данных");
         models.Add(2, "Web-программирование");
         models.Add(3, "Системное программирование");
         models.Add(4, "Принципы языков программирования");
         models.Add(5, "Теория автоматического управления");
         models.Add(6, "Технологии цифровой связи");
         models.Add(7, "Микропроцессорные системы");
         models.Add(8, "Цифровая схемотехника");
         models.Add(9, "Сигналы и цепи");
         models.Add(10, "Защита данных");

      }
      private void bRun_Click(object sender, EventArgs e)
      {
         var items = from s in models
                     where s.Value.Contains(tbFilter.Text)
                     select s;

         // Заполняем список результатами запроса
         listBox1.DataSource = items.ToList();
      }
   }
}
```

Скомпилируйте и запустите приложение. Результат работы приложения и запроса с фильтром представлен на рис. 9.3.

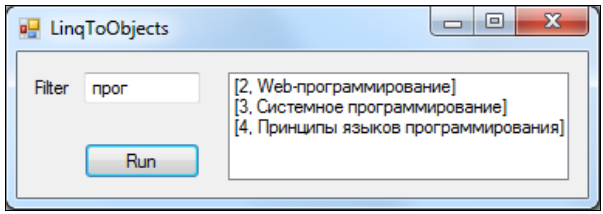


Рис. 9.3. Выполнение LINQ-запроса с фильтром для коллекции `Dictionary<int, string>`

Запросы к пользовательским классам

В качестве объектов запросов LINQ также возможно использовать и собственные классы, созданные вами. Для построения такого класса используем сущность `Trainer`, которую мы создали в нашей учебной базе данных. Присоедините к проекту новый класс (**Add | Class**) и назовите его `Trainer`. В классе создайте конструктор с параметрами и набор открытых свойств, такой же, как и поля в таблице `Trainer` базы данных `College`. Код класса представлен в листинге 9.4.

> ***Примечание***
>
> Полный код приложения на прилагаемом к книге диске находится в каталоге Chapter09\LinqToObjects.v3.

Листинг 9.4. Класс `Trainer`

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

namespace LinqToObjects
{
   public class Trainer
   {
      public Trainer(int trainerId, string firstName, string lastName,
         DateTime dateFrom, DateTime? dateTo)
      {
         TrainerId = trainerId;
         FirstName = firstName;
         LastName = lastName;
         DateFrom = dateFrom;
         DateTo = dateTo;
      }

      public int TrainerId { get; set; }
      public string FirstName { get; set; }
      public string LastName { get; set; }
```

```
        public DateTime DateFrom { get; set; }
        public DateTime? DateTo { get; set; }
    }
}
```

В классе формы инициализируем список объектов `List<Trainer>` и напишем запрос, возвращающий преподавателей, работающих в данный момент в колледже. Полный код класса формы приложения представлен в листинге 9.5.

Листинг 9.5. Программа с запросом к пользовательскому классу `Trainer`

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

namespace LinqToObjects
{
   public partial class FMain : Form
   {
      private List<Trainer> trainers;

      public Form1()
      {
         InitializeComponent();
         InitData();
      }

      private void InitData()
      {
         trainers = new List<Trainer>();

         trainers.Add(new Trainer(1001, "Николай", "Котов",
            new DateTime(2006, 11, 21), null));
         trainers.Add(new Trainer(1002, "Константин", "Захаров",
            new DateTime(2006, 08, 03), null));
         trainers.Add(new Trainer(1003, "Игорь", "Хохлов",
            new DateTime(2006, 06, 20), null));
         trainers.Add(new Trainer(1004, "Елена", "Сергеева",
            new DateTime(2008, 06, 27), null));
         trainers.Add(new Trainer(1005, "Алиса", "Маринина",
            new DateTime(2007, 08, 03), new DateTime(2009, 07, 15)));
         trainers.Add(new Trainer(1006, "Иван", "Иванов",
            new DateTime(2010, 03, 07), null));
         trainers.Add(new Trainer(1007, "Владимир", "Лунин",
            new DateTime(2005, 10, 26), null));
```

```
            trainers.Add(new Trainer(1008, "Тамара", "Панина",
               new DateTime(2010, 04, 21), new DateTime(2010, 08, 06)));
        }

        private void bRun_Click(object sender, EventArgs e)
        {
            var items = from s in trainers
                        where s.DateTo == null
                        select s.FirstName + ' ' + s.LastName;
            listBox1.DataSource = items.ToList();
        }
    }
}
```

Скомпилируйте и запустите приложение. Результат работы приложения и запроса с фильтром к пользовательскому классу представлен на рис. 9.4.

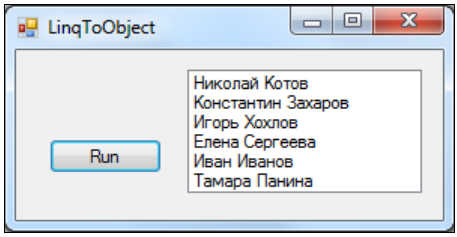


Рис. 9.4. Выполнение LINQ-запроса с фильтром для пользовательского класса

Как мы видим из всех вышеприведенных примеров запросов LINQ to Object, строить запросы LINQ одинаково легко как к простым объектам, вроде массивов и коллекций, так и к пользовательским классам.

LINQ to XML

В LINQ to XML реализован современный пересмотренный подход к программированию средствами XML. Кроме того, реализованы встроенные в память возможности модификации документов модели DOM, поддерживаются выражения запросов LINQ. Хотя в синтаксическом отношении эти выражения запросов отличаются от XPath, они предоставляют аналогичные функциональные возможности.

Используя LINQ to XML, можно сократить объемы необходимого кода и сделать его более выразительным и компактным. LINQ to XML — это оснащенный средствами LINQ и встроенный в память программный интерфейс XML, позволяющий работать с XML-файлами внутри языков программирования .NET Framework.

LINQ to XML подобен модели DOM в том отношении, что загружает XML-документ в память. К такому документу можно направить запрос, его можно изменить, а после изменения его можно сохранить в файле или сериализовать и передать через Интернет. Однако между интерфейсом LINQ to XML и моделью DOM

существуют отличия: интерфейс реализует более легкую и простую в работе модель объектов. Кроме того, в нем используются преимущества, реализованные в Visual C# 2010.

XElement

Класс `XElement` представляет элемент XML и является основным классом в LINQ to XML. Этот класс можно использовать для создания элементов, изменения содержимого элемента, добавления, изменения или удаления дочерних элементов, добавления к элементам атрибутов или сериализации элемента.

Сначала рассмотрим применение класса `XElement` для создания дерева объектов. Потом мы покажем примеры запросов LINQ к созданному нами XML-дереву.

Создайте новый проект Windows Forms. На форме разместите панель инструментов с двумя кнопками: для создания XML-дерева (`bNew`) и для выполнения запроса к этому дереву (`bFilter`). Значки для кнопок можно взять из VS2010 Image Library. На форму также поместите элемент управления `WebBrowser` для отображения XML-документа. Дизайн формы приложения представлен на рис. 9.5.

> ***Примечание***
>
> Полный код приложения на прилагаемом к книге диске находится в каталоге Chapter09\LinqToXml.

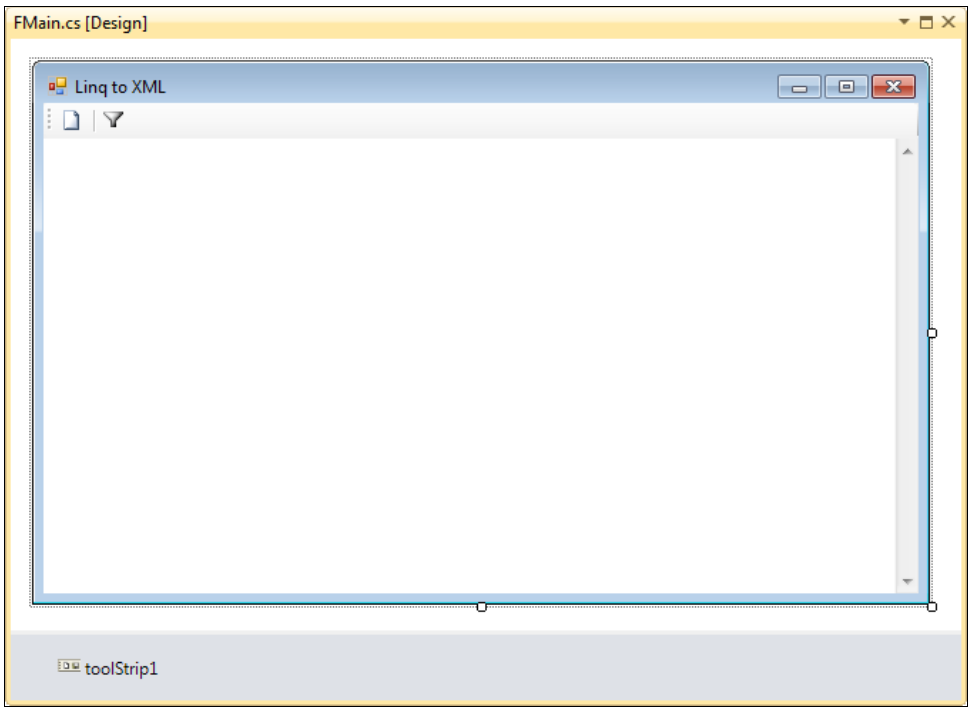


Рис. 9.5. Дизайн формы приложения для запросов к XML-деревьям

В классе `FMain` объявим закрытые переменные, как показано в листинге 9.6.

Листинг 9.6. Закрытые переменные класса `FMain`

```
// Название корневого элемента XML-дерева
private const string Root = "DocumentElement";

// Название файла для сохранения XML-дерева
private const string FileName = "D:\\tmp.xml";

// Экземпляр XML-дерева
private XElement xmlDoc;

// Пространство имен XML-дерева
private XNamespace ns = "http://tempuri.org/CollegeDataSet.xsd";
```

Для кнопки **New** создайте обработчик события нажатия кнопки `bNew_Click`. Напишите в теле обработчика события код для динамического создания XML-дерева, как показано в листинге 9.7.

Листниг 9.7. Обработчик события кнопки *New*

```
private void bNew_Click(object sender, EventArgs e)
{
   xmlDoc = new XElement(ns + Root,
      new XElement(ns + "Trainer",
         new XElement(ns + "TrainerId", 1),
         new XElement(ns + "FirstName", "Николай"),
         new XElement(ns + "LastName", "Котов"),
         new XElement(ns + "DateFrom", new DateTime(2006, 11, 21)),
         new XElement(ns + "DateTo", null)),
      new XElement(ns + "Trainer",
         new XElement(ns + "TrainerId", 2),
         new XElement(ns + "FirstName", "Алексей"),
         new XElement(ns + "LastName", "Захаров"),
         new XElement(ns + "DateFrom", new DateTime(2006, 08, 03)),
         new XElement(ns + "DateTo", null)),
      new XElement(ns + "Trainer",
         new XElement(ns + "TrainerId", 3),
         new XElement(ns + "FirstName", "Леонид"),
         new XElement(ns + "LastName", "Хохлов"),
         new XElement(ns + "DateFrom", new DateTime(2006, 06, 20)),
         new XElement(ns + "DateTo", null)),
      new XElement(ns + "Trainer",
         new XElement(ns + "TrainerId", 4),
         new XElement(ns + "FirstName", "Алла"),
         new XElement(ns + "LastName", "Сергеева"),
```

```
            new XElement(ns + "DateFrom", new DateTime(2008, 6, 27)),
            new XElement(ns + "DateTo", new DateTime(2009, 7, 15))),
        new XElement(ns + "Trainer",
            new XElement(ns + "TrainerId", 5),
            new XElement(ns + "FirstName", "Нина"),
            new XElement(ns + "LastName", "Маринина"),
            new XElement(ns + "DateFrom", new DateTime(2006, 11, 21)),
            new XElement(ns + "DateTo", new DateTime(2009, 07, 15))),
        new XElement(ns + "Trainer",
            new XElement(ns + "TrainerId", 6),
            new XElement(ns + "FirstName", "Иван"),
            new XElement(ns + "LastName", "Иванов"),
            new XElement(ns + "DateFrom", new DateTime(2006, 6, 20)),
            new XElement(ns + "DateTo", null)),
        new XElement(ns + "Trainer",
            new XElement(ns + "TrainerId", 7),
            new XElement(ns + "FirstName", "Владимир"),
            new XElement(ns + "LastName", "Лунин"),
            new XElement(ns + "DateFrom", new DateTime(2005, 6, 27)),
            new XElement(ns + "DateTo", null)),
        new XElement(ns + "Trainer",
            new XElement(ns + "TrainerId", 8),
            new XElement(ns + "FirstName", "Тамара"),
            new XElement(ns + "LastName", "Панина"),
            new XElement(ns + "DateFrom", new DateTime(2010, 4, 21)),
            new XElement(ns + "DateTo", new DateTime(2010, 9, 21))));
    xmlDoc.Save(FileName);

    webBrowser1.Navigate(FileName);
}
```

Скомпилируйте и запустите приложение. При нажатии кнопки **New** будет создано XML-дерево, которое отобразится в окне приложения. Результат работы приложения представлен на рис. 9.6.

Большим преимуществом LINQ to XML является возможность использования результатов запросов в качестве параметров конструкторов объектов `XElement` и `XAttribute`, которые можно использовать для программного создания XML-деревьев. Этот подход, именуемый *функциональным построением*, дает разработчикам возможность легко создавать XML-деревья и работать с ними в коде приложения.

Выполнение запросов LINQ to XML

Можно написать LINQ-запросы, по которым из созданного нами XML-дерева будут извлекаться данные. Этот запрос вернет список преподавателей, на данный момент времени работающих в колледже. Для кнопки **Filter** создайте обработчик

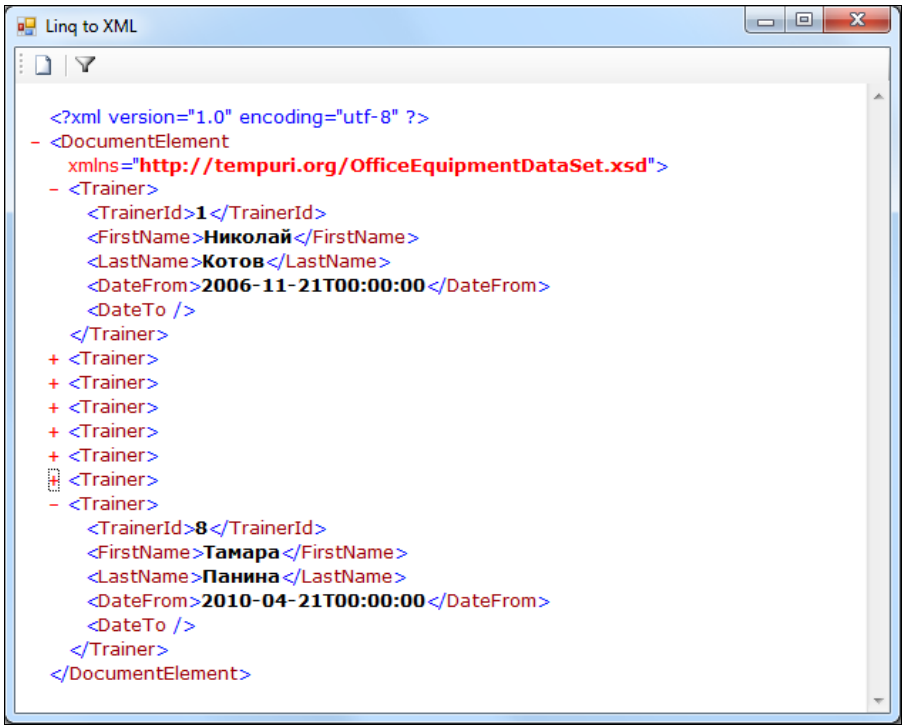


Рис. 9.6. Приложение, отображающее сгенерированное XML-дерево

события нажатия кнопки `bFilter _Click` и напишите в теле обработчика события код, представленный в листинге 9.8.

Листинг 9.8. Реализация обработчика события `bFilter_Click`

```
private void bFilter_Click(object sender, EventArgs e)
{
   xmlDoc = new XElement(ns + Root,
         from item in xmlDoc.Elements()
         where (item.Element(ns + "DateTo").IsEmpty)
         select item);

   xmlDoc.Save(FileName);
   webBrowser1.Navigate(FileName);
}
```

Теперь скомпилируйте и запустите приложение. Создайте новое XML-дерево, которое отобразится в окне приложения. Затем кнопкой **Filter** выполните запрос LINQ к созданному XML-дереву. Результат работы приложения представлен на рис. 9.7.

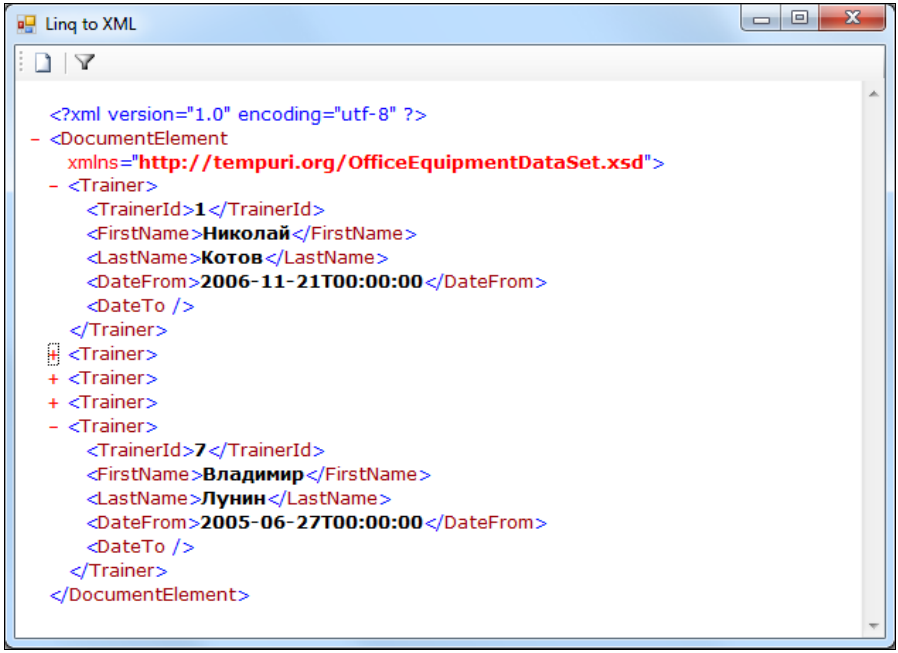


Рис. 9.7. Результат запроса к созданному XML-дереву

LINQ to DataSet

Объект `DataSet`, который мы рассматривали в прошлой главе, имеет довольно ограниченные возможности для создания запросов. Например, метод `Select()` можно использовать для фильтрации и сортировки, а методы `GetChildRows()` и `GetParentRow()` — для навигации по иерархии. Для более сложных операций необходимо писать пользовательские запросы. Это может снизить производительность разрабатываемых приложений и, кроме того, создать проблемы при сопровождении или модифицировании этих приложений.

Использование LINQ to DataSet упрощает и ускоряет запросы к данным, кэшированным в объекте `DataSet`. Эти запросы выражены на языке программирования, а не в виде строк, внедренных в код приложения, что повышает эффективность разработки в среде Visual Studio, поскольку обеспечивается проверка синтаксиса во время компиляции, статическая типизация и поддержка технологии IntelliSense для LINQ. Технология LINQ to DataSet также может использоваться для запросов к данным, находящимся в одном или нескольких источниках.

Хотя технология LINQ to DataSet построена на основе архитектуры ADO.NET и использует ее, но не заменяет ADO.NET в коде приложения. Существующий код ADO.NET будет по-прежнему работать в приложении LINQ to DataSet. Таким образом, технологию LINQ to DataSet выгодно использовать при модификации ранее созданных приложений, в которых работа с данными осуществлялась с использованием типизированных объектов `DataSet`.

Запросы к наборам данных с помощью LINQ to DataSet

Перед выполнением запроса к объекту `DataSet` с помощью LINQ to DataSet необходимо поместить в объект `DataSet`. После того как в объекте `DataSet` появятся данные, к нему можно выполнять запросы. Подготовка запросов с помощью LINQ to DataSet похожа на использование LINQ с другими источниками данных, поддерживающих LINQ, например LINQ to Objects, рассмотренные ранее в этой главе. Запросы LINQ могут выполняться к одиночным таблицам `DataSet` или к нескольким таблицам с помощью стандартных операторов запроса `Join` и `GroupJoin`.

Запросы LINQ поддерживаются к типизированным и к нетипизированным объектам `DataSet`. Если схема объекта `DataSet` известна во время разработки приложения, то рекомендуется использование типизированного `DataSet`. Типизированный `DataSet` делает запросы на LINQ более простыми и читабельными.

Поскольку `DataTable` и `DataSet` не поддерживают интерфейс `IEnumerable<T>`, при выполнении запроса необходимо их явное приведение к интерфейсу `IEnumerable<T>`, используя метод `AsEnumerable()`. Например, запрос с немедленным выполнением для получения данных из таблицы `Grade` всех оценок, равных 5, будет выглядеть следующим образом:

```
IEnumerable<DataRow> query =
                from item in dtGrade.AsEnumerable()
                where item.Field<int>("GradeValue") == 5
                select item;
```

Создание запросов LINQ to DataSet в приложениях

В качестве примера приложения, реализующего запросы LINQ to DataSet, создадим новый проект Windows Forms. Кстати, для формы приложения вы можете использовать формы из созданных в предыдущей главе проектов. Обратите внимание, что у наших проектов, которые работают с базой данных `College`, применяется одинаковое пространство имен и главные формы также имеют одинаковое название. Вы можете их использовать при создании новых проектов, внося лишь небольшие изменения в пользовательский интерфейс и удаляя ненужные элементы управления и обработчики событий.

В этом проекте у нас будет форма с элементом управления `DataGridView` в клиентской области окна формы и панелью инструментов, на которой будут располагаться кнопки навигатора и кнопка для выполнения запроса (при желании можете не использовать навигатор). Дизайн формы приложения для запросов LINQ to DataSet представлен на рис. 9.8.

Примечание

Полный код приложения на прилагаемом к книге диске находится в каталоге Chapter09\LinqToDataSet.

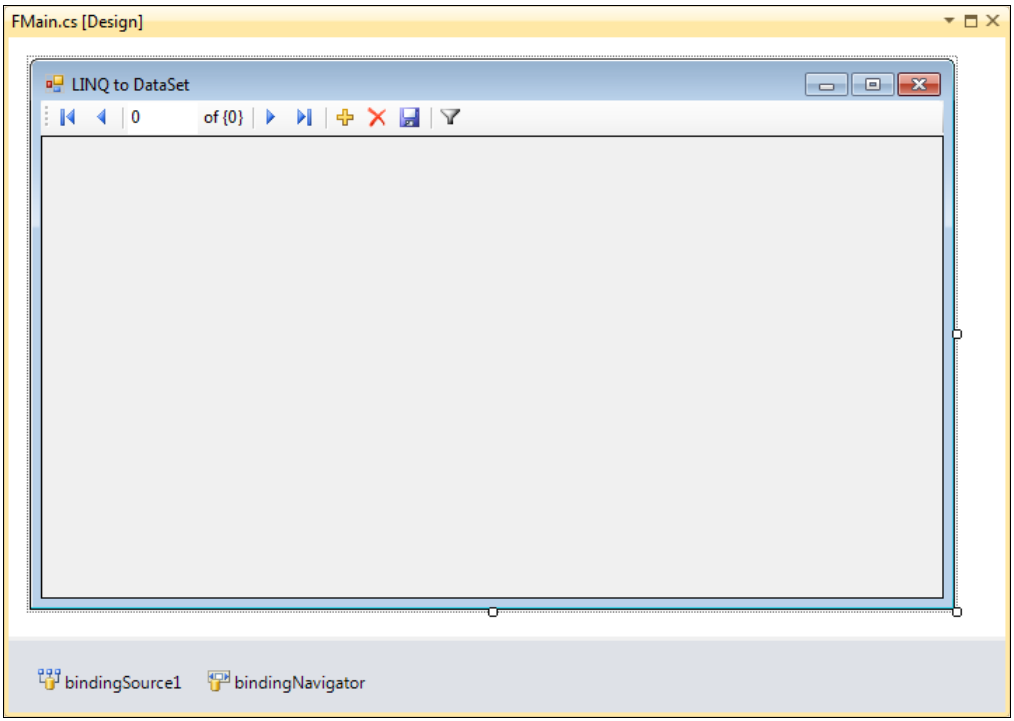


Рис. 9.8. Дизайн формы приложения для запросов LINQ to DataSet

В качестве `DataSet` для источника данных подключите библиотеку `College.Data`, созданную нами в предыдущей главе. Для этого на нее надо будет добавить ссылку в проект. Если ваш новый проект и библиотека находятся в разных решениях, для добавления ссылки на библиотеку в диалоговом окне **Add Reference** используйте вкладку **Browse** и укажите путь до этой библиотеки (рис. 9.9).

Создайте обработчики события `Load` для формы и `Click` для кнопки. В коде класса `FMain` в обработчике `FMain_Load` будет код для заполнения `DataSet` данными из таблицы `Grade`. В теле обработчика события `bFilter_Click` создадим запрос, возвращающий список студентов, имеющих оценку 5. Полный код класса формы `FMain` представлен в листинге 9.9.

Листинг 9.9. Код класса формы `FMain`

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Windows.Forms;

using College.Data;
using College.Data.CollegeDataSetTableAdapters;
```

```
namespace College.UI
{
   public partial class FMain : Form
   {
      private Grade_SelectTableAdapter daGrade;
      private CollegeDataSet dsCollege;

      public FMain()
      {
         InitializeComponent();
         dsCollege = new CollegeDataSet();
         daGrade = new Grade_SelectTableAdapter();
      }

      private void FMain_Load(object sender, EventArgs e)
      {
         daGrade.Fill(dsCollege.Grade_Select, null, null, null);
         bindingSource1.DataSource = dsCollege.Grade_Select;
         dgvData.DataSource = bindingSource1;
      }

      private void bFilter_Click(object sender, EventArgs e)
      {
         DataTable dtGrade = dsCollege.Grade_Select;
         IEnumerable<DataRow> query =
               from item in dtGrade.AsEnumerable()
               where item.Field<int>("GradeValue") == 5
               select item;

         dtGrade = query.CopyToDataTable();
         bindingSource1.DataSource = dtGrade;
      }
   }
}
```

Скомпилируйте и запустите приложение. Приложение должно отобразить информацию о всех сданных экзаменах по предметам, фамилии и имена студентов. Затем кнопкой **Filter** выполните запрос LINQ к созданному XML-дереву. Результат работы приложения представлен на рис. 9.10.

> ***Примечание***
>
> В этих примерах мы использовали простые запросы LINQ в коде приложения с фиксированными параметрами запросов. Попробуйте усовершенствовать приложения и создать построитель запросов, примерно такой, как мы создали в предыдущей главе для `DataSet`, но используйте для этого LINQ to DataSet.

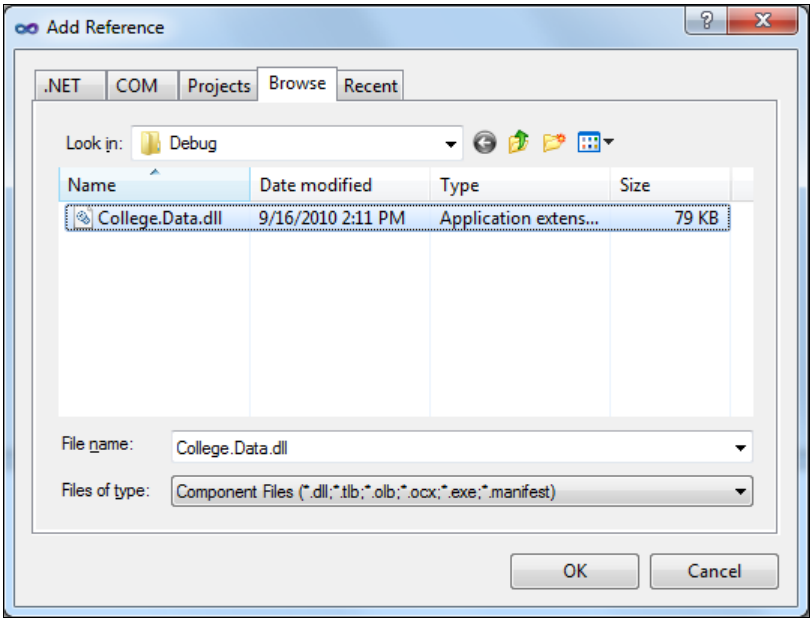
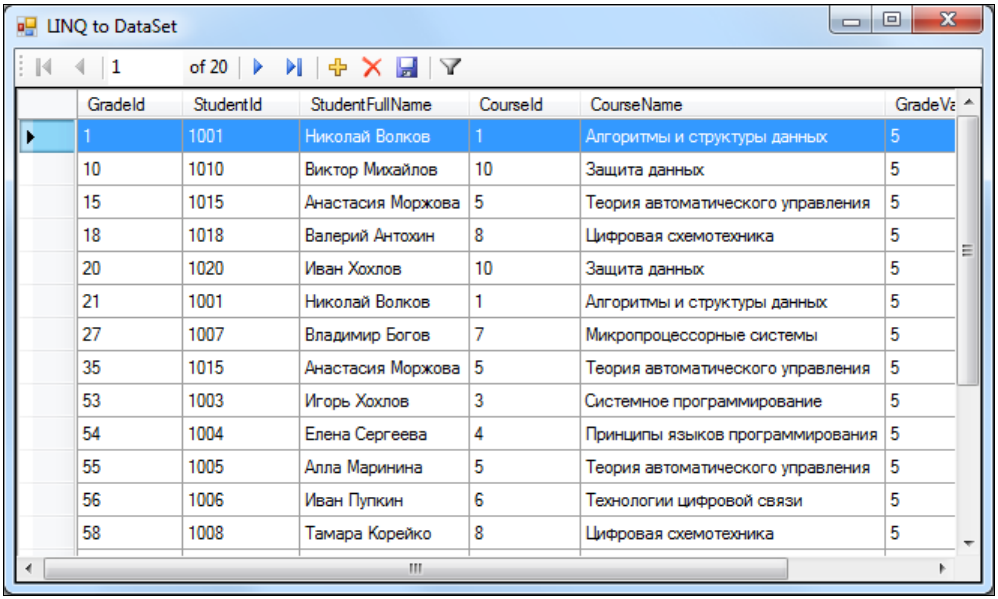


Рис. 9.9. Добавление ссылки на библиотеку `College.Data`



GradeId	StudentId	StudentFullName	CourseId	CourseName	GradeVa
1	1001	Николай Волков	1	Алгоритмы и структуры данных	5
10	1010	Виктор Михайлов	10	Защита данных	5
15	1015	Анастасия Моржова	5	Теория автоматического управления	5
18	1018	Валерий Антохин	8	Цифровая схемотехника	5
20	1020	Иван Хохлов	10	Защита данных	5
21	1001	Николай Волков	1	Алгоритмы и структуры данных	5
27	1007	Владимир Богов	7	Микропроцессорные системы	5
35	1015	Анастасия Моржова	5	Теория автоматического управления	5
53	1003	Игорь Хохлов	3	Системное программирование	5
54	1004	Елена Сергеева	4	Принципы языков программирования	5
55	1005	Алла Маринина	5	Теория автоматического управления	5
56	1006	Иван Пупкин	6	Технологии цифровой связи	5
58	1008	Тамара Корейко	8	Цифровая схемотехника	5

Рис. 9.10. Пример выполнения запросов LINQ to DataSet в приложении

LINQ to SQL

Технология LINQ to SQL — это удобный и эффективный инструмент при разработке приложений, предназначенных для работы с базами данных. Применяя LINQ to SQL, можно использовать программную модель LINQ для уже существующей схемы базы данных. LINQ to SQL позволяет создавать типизированные объекты, которые отображают данные. Вместо сопоставления с концептуальной моделью эти созданные классы отображаются напрямую на таблицы, представления, хранимые процедуры и пользовательские функции базы данных.

При работе приложения технология LINQ to SQL преобразует запросы LINQ из объектной модели в язык SQL и отправляет их в базу данных для выполнения. Когда база данных возвращает результаты запроса, LINQ to SQL преобразует их обратно в типизированные объекты, с которыми можно работать непосредственно в коде приложения.

Создание объектной модели LINQ to SQL

Объектную модель можно создать из существующей базы данных и использовать ее в заданном по умолчанию состоянии. Кроме того, при необходимости можно настроить дополнительные свойства модели и ее поведение.

При работе в интегрированной среде разработки Visual Studio для создания и настройки объектной модели базы данных используется инструмент **Object Relational Designer** (Реляционный конструктор объектов).

В этом разделе мы разработаем библиотеку для доступа к нашей базе данных `College`, аналогичную созданной ранее в *главе 8*, но с использованием технологии LINQ to SQL вместо типизированного объекта `DataSet`.

Создадим новый проект **Class Library** и назовем его `College.Data`, как и библиотеку из *главы 8*. Из библиотеки удалим файл класса Class1.cs, созданный по умолчанию, и добавим **LINQ to SQL Classes** (рис. 9.11).

> ***Примечание***
>
> Полный код приложения на прилагаемом к книге диске находится в каталоге Chapter09\ College.Data.

После добавления **LINQ to SQL Classes** в окне **Solution Explorer** (рис. 9.12) появится дополнительный узел **College.dbml**, состоящий из двух файлов:

- **College.dbml.layout** — представление модели объектов в виде XML;
- **College.designer.cs** — автоматически сгенерированный код классов C#, представляющих объектно-ориентированную модель базы данных.

Однако мы еще не подсоединяли базу данных, и эти файлы пока не содержат нужной нам информации.

Если два раза щелкнуть мышью на файле College.dbml, откроется пустое окно **Object Relational Designer**, показанное на рис. 9.13.

Перенесите на поверхность окна **Object Relational Designer** из **Server Explorer** все таблицы из нашей базы данных `College`. Конструктор отобразит их вместе со связями (рис. 9.14).

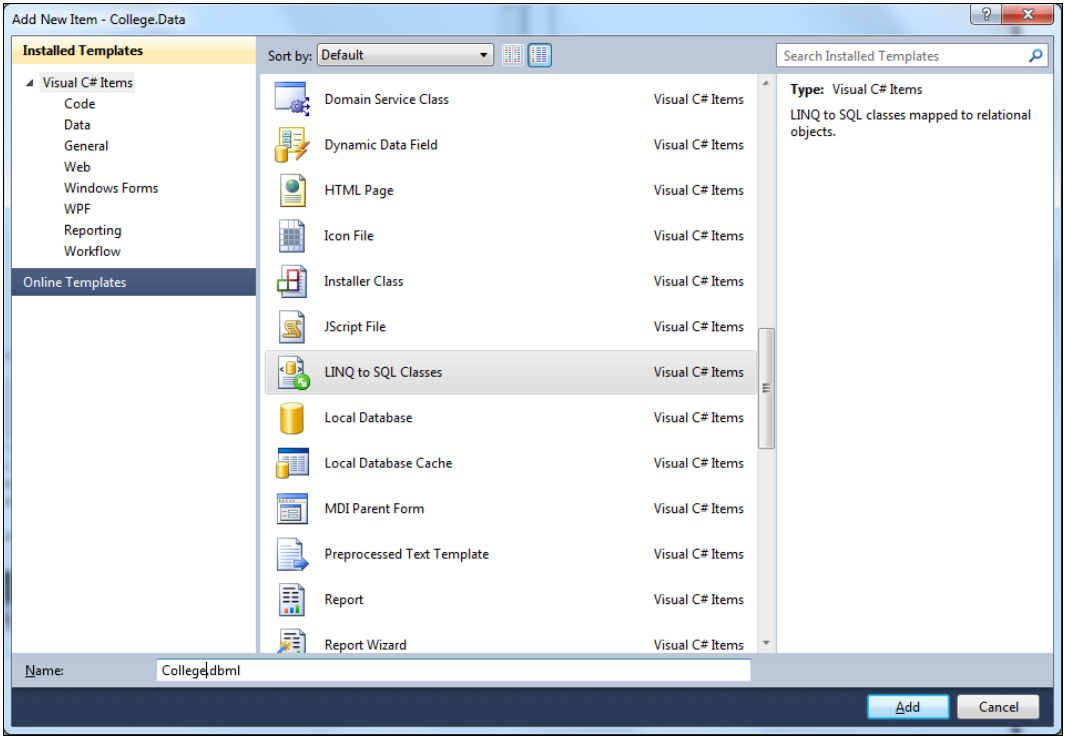


Рис. 9.11. Добавление в проект **LINQ to SQL Classes**

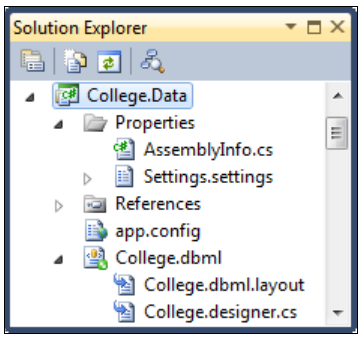


Рис. 9.12. Структура проекта `College.Data`

В файле College.designer.cs конструктор **Object Relational Designer** автоматически сгенерирует оболочку для данных, представляющую собой набор классов, соответствующих объектам в базе данных `College`. Вы можете посмотреть сгенерированные классы, добавив к проекту диаграмму классов (пункт **View Class Diagram** в контекстном меню проекта `College.Data` или соответствующий значок на панели

инстументов окна **Solution Explorer**). Диаграмма классов реляционной модели нашей базы данных `College` показана на рис. 9.15.

Как видно из диаграммы классов, кроме классов, представляющих объекты из базы данных `College`, был сгенерирован класс `CollegeDataContext`. Для модели данных класс `DataContext` (в нашем случае он называется `CollegeDataContext`) пред-

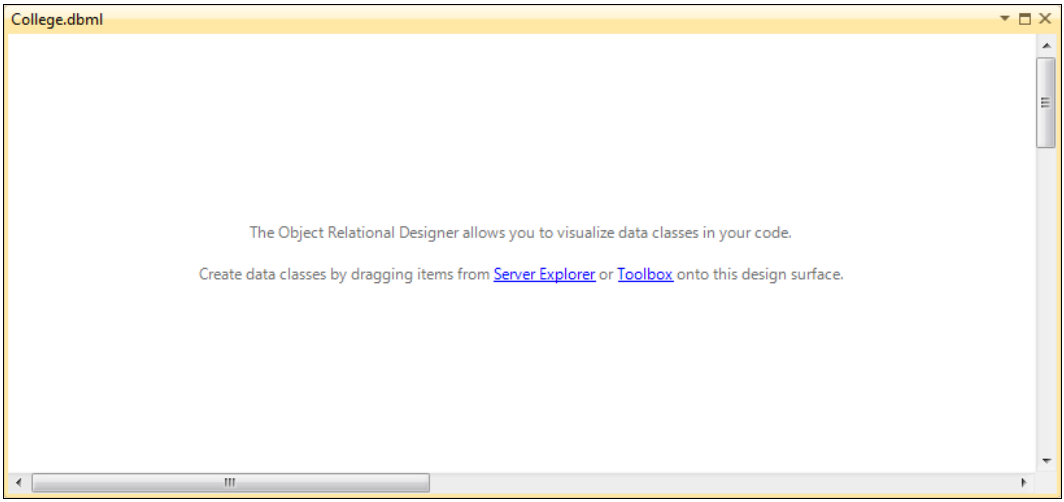


Рис. 9.13. Окно **Object Relational Designer**

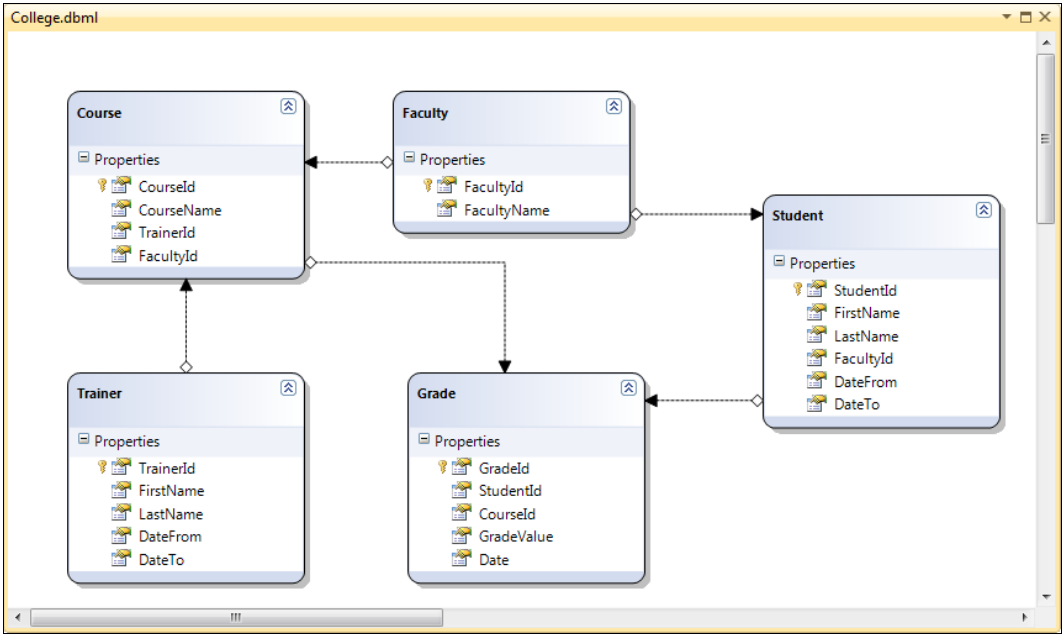


Рис. 9.14. Модель базы данных `College` в окне **Object Relational Designer**

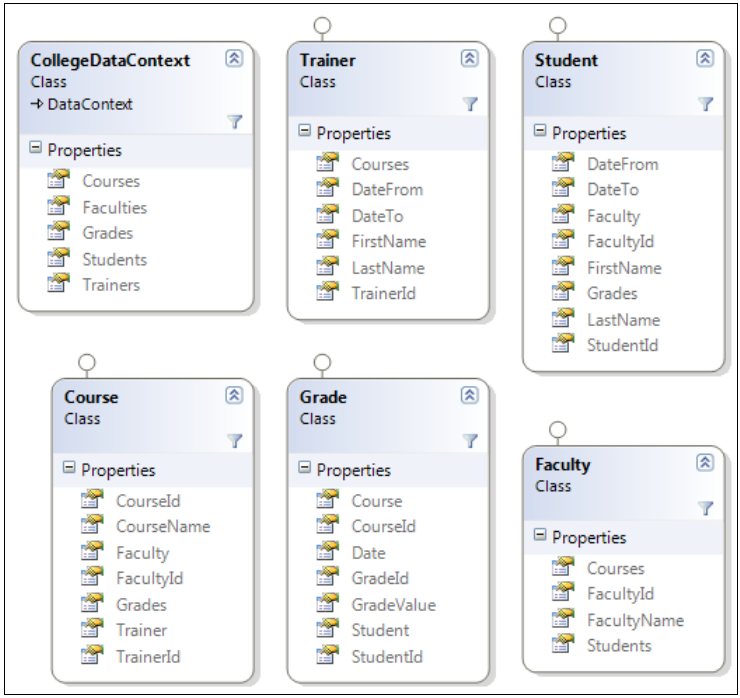


Рис. 9.15. Диаграмма классов реляционной модели базы данных `College`

ставляет собой класс LINQ to SQL, который действует как посредник между базой данных SQL Server и классами сущностей LINQ to SQL, сопоставленных этой базе данных. Класс `DataContext` содержит данные строки соединения и методы для подключения к базе данных и работы с данными в базе данных.

По умолчанию класс `DataContext` содержит несколько открытых методов, доступных для вызова из вашего кода, например, `SubmitChanges()`, который отправляет обновленные данные из классов LINQ to SQL в базу данных. Можно также создать дополнительные методы в класс `DataContext` точно так, как добавлялись бы методы, чтобы расширить любой класс.

Классы, представляющие объекты базы данных, тоже позволяют расширение. Например, для отображения имени и фамилии в одном поле можно создать дополнительные свойства в классах `Student` и `Trainer`. Расширения для существующих классов рекомендуется создавать в отдельных файлах (как `partial`-класс), т. к., добавив их в файл College.designer.cs, можно их потерять при обновлении модели. Для нашей модели можно создать отдельные файлы Student.cs и Trainer.cs и добавить в них дополнительные свойства `StudentFullName` и `TrainerFullName`, как показано в листингах 9.10 и 9.11.

Листинг 9.10. Расширение класса `Student`

```
namespace College.Data
{
   public partial class Student
```

```
    {
        public string StudentFullName
        {
            get { return _FirstName + " " + _LastName; }
        }
    }
}
```

Листинг 9.11. Расширение класса `Trainer`

```
namespace College.Data
{
    public partial class Trainer
    {
        public string TrainerFullName
        {
            get { return _FirstName + " " + _LastName; }
        }
    }
}
```

Создание приложения для работы с данными

Для создания приложения, отображающего данные, возьмем за основу созданное нами приложение из *главы 8, разд. "Динамическое связывание данных в период выполнения"*. В *главе 8* мы создали удобный пользовательский интерфейс для работы с данными, который мы можем использовать в этом примере, а также в наших будущих примерах приложений для работы с данными. Взяв его за основу, в это приложение мы внесем некоторые изменения.

> ***Примечание***
>
> Полный код приложения на прилагаемом к книге диске находится в каталоге Chapter09\LinqToSql.

Для начала удалим ссылку на старую библиотеку `College.Data`, использующую `DataSet`, и добавим ссылку на библиотеку, созданную в этой главе. Затем нам потребуется ссылка на .NET-сборку `System.Data.Linq` (рис. 9.16).

В классе `FMain` формы также внесем некоторые изменения. Объявим закрытые переменные и строковые константы, представляющие имена сущностей нашей базы данных, как показано в листинге 9.12, добавив переменную, представляющую модель базы данных `College`.

Листинг 9.12. Закрытые переменные класса `FMain`

```
private const string CourseEntity = "Course";
private const string FacultyEntity = "Faculty";
```

```
private const string GradeEntity = "Grade";
private const string StudentEntity = "Student";
private const string TrainerEntity = "Trainer";

private string currentEntity;
private List<string> displayTabPageNames;

// Переменная, представляющая модель базы данных College
private CollegeDataContext collegeContext;
```

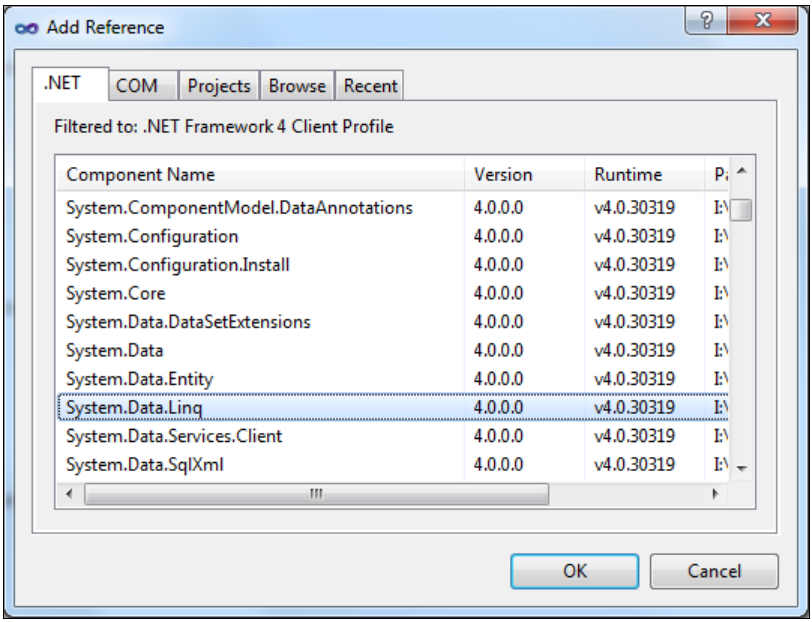


Рис. 9.16. Подключение ссылки на сборку `System.Data.Linq`

Код обработчика события `Load` формы представлен в листинге 9.13.

Листинг 9.13. Обработчик события `Load` формы `FMain`

```
private void FMain_Load(object sender, EventArgs e)
{
   // Создаем новый экземпляр модели базы данных
   collegeContext = new CollegeDataContext();

   displayTabPageNames = new List<string>();

   treeView1.Nodes.Add("College", "College", 0, 0);

   treeView1.Nodes[0].Nodes.Add(CourseEntity, CourseEntity, 1, 1);
   treeView1.Nodes[0].Nodes.Add(FacultyEntity, FacultyEntity, 1, 1);
   treeView1.Nodes[0].Nodes.Add(TrainerEntity, TrainerEntity, 1, 1);
```

```
    treeView1.Nodes[0].Nodes.Add(GradeEntity, GradeEntity, 1, 1);
    treeView1.Nodes[0].Nodes.Add(StudentEntity, StudentEntity, 1, 1);

    treeView1.ExpandAll();

    tabControl1.ContextMenuStrip = tabContextMenu;
    bindingNavigator1.BindingSource = bindingSource1;
}
```

В теле обработчика события `AfterSelect` будут самые большие изменения. Для привязки данных к объекту класса `BindingSource` мы используем запросы LINQ. Код обработчика события `AfterSelect` представлен в листинге 9.14.

Листинг 9.14. Обработчик события `AfterSelect` элемента управления `TreeView`

```
private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
{
    if (treeView1.SelectedNode.Parent != null)
    {
        currentEntity = treeView1.SelectedNode.Name;
        switch (currentEntity)
        {
            case CourseEntity:
                bindingSource1.DataSource =
                    from item in collegeContext.Courses
                    select new
                    {
                        item.CourseId,
                        item.CourseName,
                        item.Trainer.FirstName,
                        item.Trainer.LastName
                    };
                break;
            case FacultyEntity:
                bindingSource1.DataSource =
                    from item in collegeContext.Faculties
                    select new
                    {
                        item.FacultyId,
                        item.FacultyName,
                    };
                break;
            case GradeEntity:
                bindingSource1.DataSource =
                    from item in collegeContext.Grades
                    select new
                    {
                        item.GradeId,
```

```
                item.Student.FirstName,
                item.Student.LastName,
                item.Course.CourseName,
                item.GradeValue,
                item.Date
            };
         break;
      case StudentEntity:
         bindingSource1.DataSource =
            from item in collegeContext.Students
            select new
            {
                item.StudentId,
                item.FirstName,
                item.LastName,
                item.Faculty.FacultyName,
                item.DateFrom,
                item.DateTo
            };
         break;
      case TrainerEntity:
         bindingSource1.DataSource =
            from item in collegeContext.Trainers
            select new
            {
                item.TrainerId,
                item.FirstName,
                item.LastName,
                item.DateFrom,
                item.DateTo
            };
         break;
   }

   UCGrid grid;
   if (!displayTabPageNames.Contains(currentEntity))
   {
      displayTabPageNames.Add(currentEntity);
      TabPage page = new TabPage(currentEntity);
      page.Name = currentEntity;

      grid = new UCGrid();
      grid.Name = "dgv" + currentEntity;
      grid.Parent = page;
      grid.Dock = DockStyle.Fill;
      grid.DataSource = bindingSource1;
      tabControl1.TabPages.Add(page);
   }
```

```
        else
        {
            grid = (UCGrid)tabControl1.TabPages[currentEntity].Controls[0];
        }

        tabControl1.SelectedTab = tabControl1.TabPages[currentEntity];
    }
}
```

Скомпилируйте и запустите приложение. Результат работы приложения представлен на рис. 9.17.

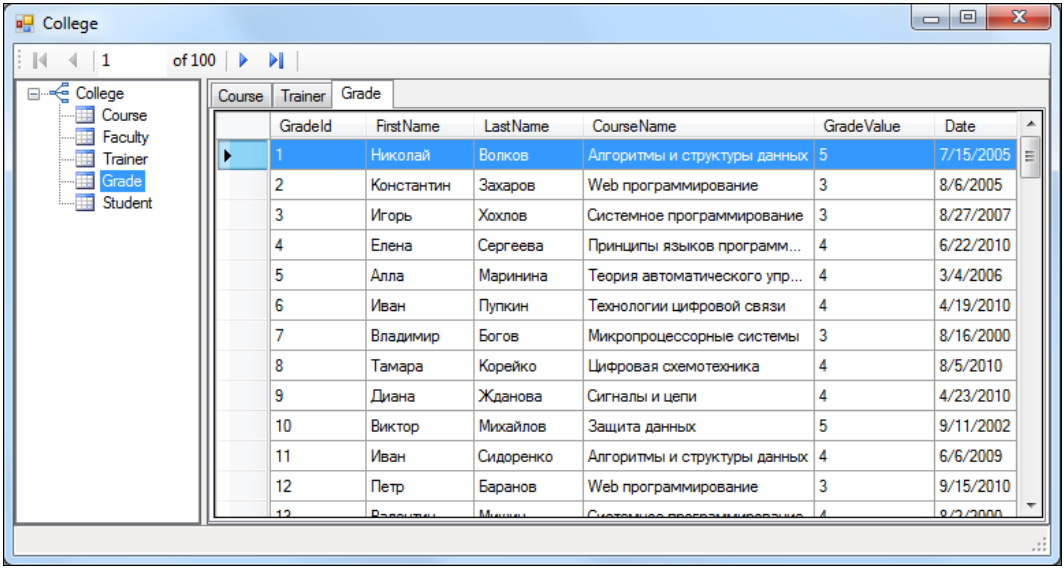
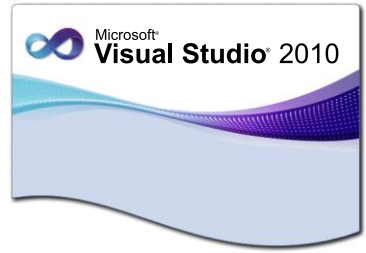


Рис. 9.17. Приложение для работы с данными, построенное на LINQ to SQL

Резюме

В этой главе вы получили краткую информацию о технологии LINQ для создания и управления данными из различных источников. Мы рассмотрели имеющийся в интегрированной среде разработки Visual Studio набор инструментов для обеспечения возможности написания запросов LINQ к объектам различного типа и генерирования модели базы данных в виде типизированных объектов с использованием технологии LINQ to SQL.

Технологии LINQ to SQL, а также Entity Framework, которую мы будем рассматривать в следующей главе, являются более эффективными технологиями, используемыми при разработке приложений для работы с данными, чем применение типизированных объектов `DataSet`.



ГЛАВА 10

Entity Framework

Платформа ADO.NET Entity Framework позволяет разработчикам создавать приложения для доступа к данным, работающие с концептуальной моделью приложения. Entity Framework предоставляет приложениям для работы с данными возможность чтения и изменения данных, представленных в виде сущностей и связей в концептуальной модели.

Платформа Entity Framework также представляет логическую структуру базы данных, состоящую из следующих слоев:

- концептуальный слой;
- слой сопоставления;
- логический слой.

Эти три слоя позволяют сопоставить реляционную базу данных и ее объектно-ориентированную модель.

Работа с данными в Entity Framework

При создании приложений на платформе Entity Framework в программном коде предоставляется возможность работы с данными, представленными в форме объектов и свойств, без необходимости прямого обращения к таблицам и столбцам реляционной базы данных.

Платформа Entity Framework компилирует набор концептуальных схем и схем хранения вместе с заданными для них сопоставлениями, создавая двунаправленные пары инструкций Entity SQL, называемые *клиентскими представлениями*. Эти представления управляют обработкой запросов и обновлений в среде выполнения. Создающий представления компилятор сопоставлений может быть вызван как во время разработки, так и во время выполнения, в момент выполнения первого запроса к схеме модели Entity Data Model.

Платформа Entity Framework строится на основе поставщиков данных ADO.NET, специфичных для типа хранилища данных, например, SQL Server. Для этого базовому поставщику данных и реляционной базе данных передается объект `EntityConnection`.

Во время выполнения запрос проходит синтаксический анализ и преобразуется в каноническое дерево команд, которое является представлением запроса в модели объектов. *Канонические деревья команд* представляют операции выбора, обновления, вставки и удаления. Вся дальнейшая обработка выполняется над деревом команд, которое является средством взаимодействия между поставщиком `System.Data.EntityClient` и базовым поставщиком данных .NET Framework, например `System.Data.SqlClient`.

Entity Data Model

Для определения данных, используемых приложениями, построенными на основе платформы Entity Framework, используется спецификация — сущностная модель данных (Entity Data Model, модель EDM).

Приложения используют определенные Entity Data Model сущности и связи в домене приложения в схеме макета. *Схема макета* применяется для создания программируемых классов, используемых кодом приложения.

Структуры хранилища, материализующие данные для приложений в этой модели, представлены в другой схеме, называемой *схемой хранилища*. Спецификация сопоставления соединяет схему макета и схему хранилища данных.

Создание Entity Data Model в Visual Studio

В Entity Framework есть инструментальные средства для определения сущностной модели данных при помощи XML-файлов EDMX. С помощью этих инструментов генерируются наборы классов на основе существующей концептуальной модели базы данных. Эти классы наследуются от `ObjectContext`, который представляет контейнер сущностей в концептуальной модели. Класс `ObjectContext` представляет также функциональность для записи результатов вставки, обновления и удаления данных в базу данных.

Для создания Entity Data Model в среде разработки Visual Studio 2010 используется мастер **Entity Data Model Vizard**. Мастер **Entity Data Model Vizard** служит для формирования EDMX-файла и позволяет создать модель на основе существующей базы данных.

Для примера снова создадим новый проект **Class Library** — библиотеку для доступа к данным с именем `College.Data`, добавим в нее (выбрав пункт **Add | New Item** в контекстном меню проекта) шаблон **ADO.NET Entity Data Model** с именем College.edmx (рис. 10.1).

> ***Примечание***
>
> Полный код приложения на прилагаемом к книге диске находится в каталоге Chapter10\ College.Data.

После нажатия кнопки **Add** запустится мастер **Entity Data Model Vizard**. В открывшемся диалоговом окне **Choose Model Contents** (Выбор содержимого модели) выберите **Generate from database** (Создать из базы данных), а затем нажмите кнопку **Next** (рис. 10.2).

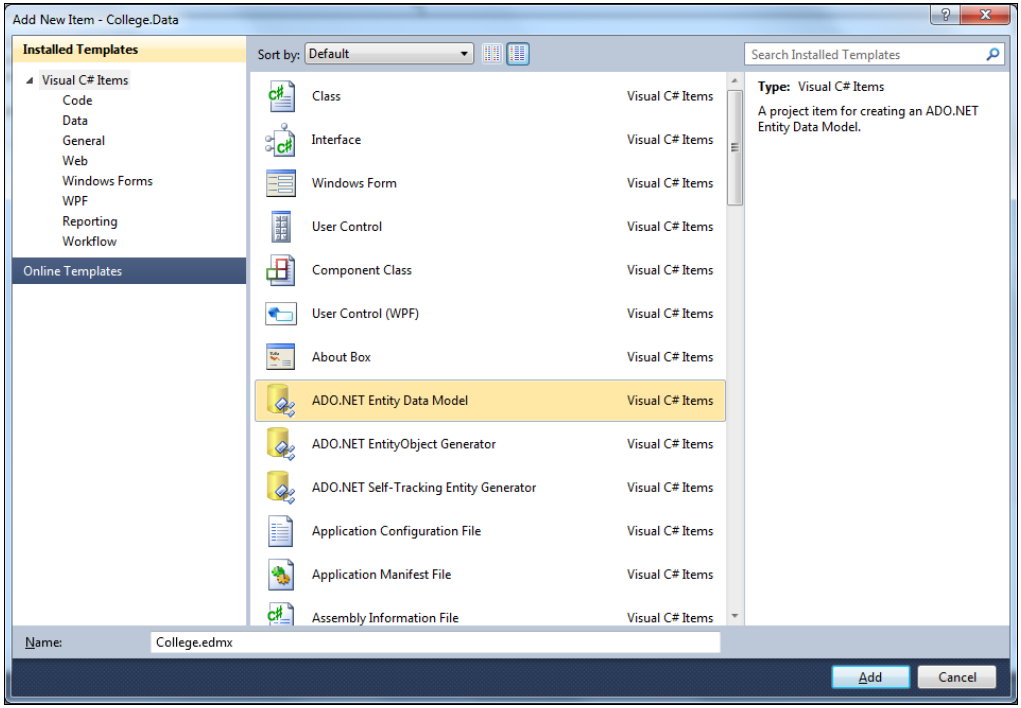


Рис. 10.1. Создание Entity Data Model

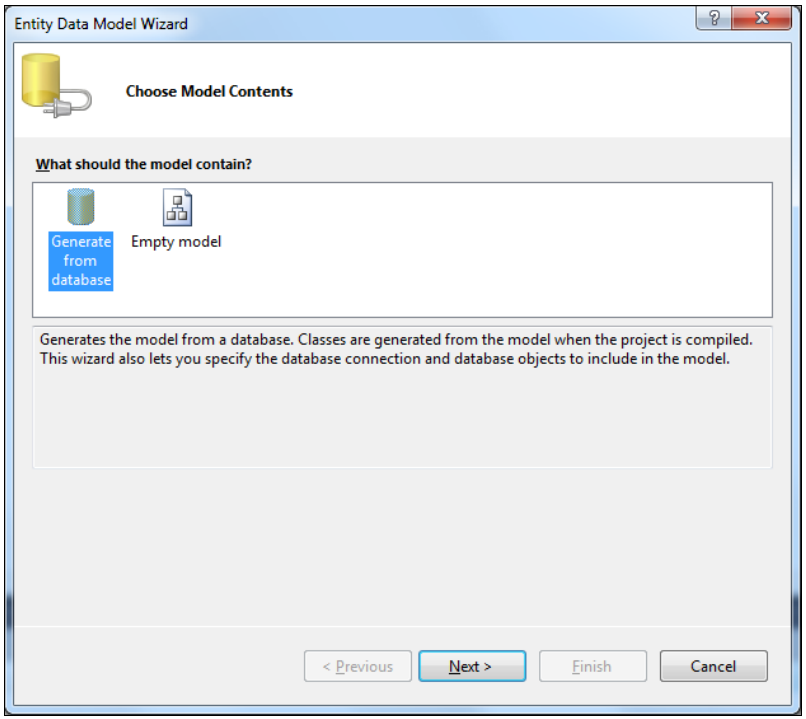


Рис. 10.2. Начальная страница мастера **Entity Data Model Wizard**

В следующем диалоговом окне мастера **Choose Your Data Connection** выберите ваше соединение с базой данных `College` (рис. 10.3) и нажмите кнопку **Next**.

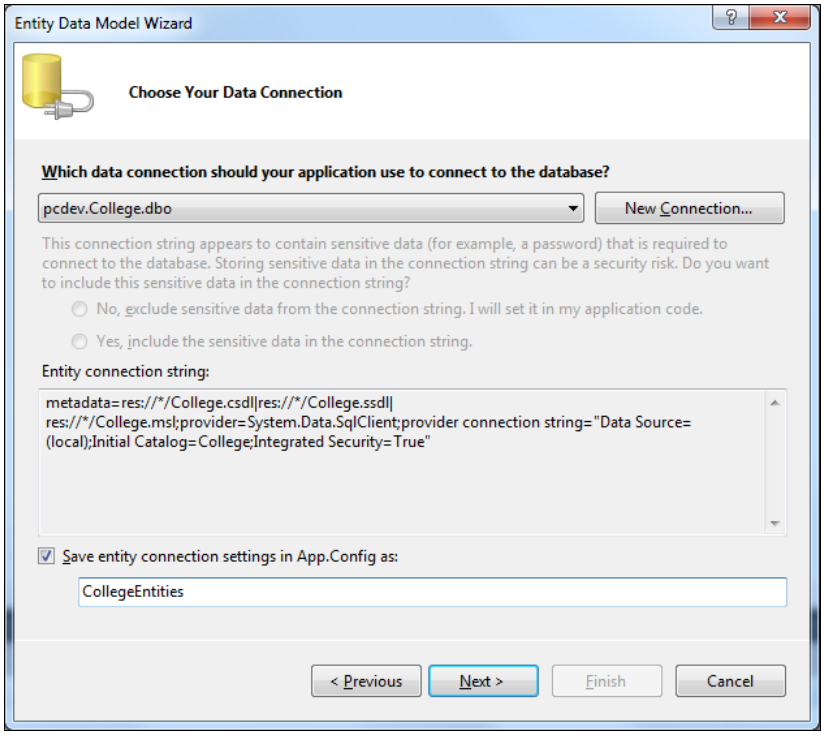


Рис. 10.3. Выбор подключения к базе данных

Появится диалоговое окно **Choose Your Database Objects** (Выбор объектов базы данных). Это окно похоже на окно мастера создания типизированного `DataSet` из *главы 8*. По умолчанию объекты в базе данных не выбираются для включения в файл модели EDMX. Выберите все таблицы для нашей базы данных, кроме `sysdiagrams` (рис. 10.4), и нажмите кнопку **Finish**.

По окончании создания EDMX-файла мастер **Entity Data Model Vizard** запускает конструктор моделей EDM ADO.NET или конструктор сущностей, который представлен на рис. 10.5.

Мастер **Entity Data Model Vizard** добавил в проект ссылки на необходимые сборки, создал файл College.edmx, который инкапсулирует модель хранения, концептуальную модель и сопоставления, и создал файл College.designer.cs с исходным кодом, содержащий классы, сформированные из концептуальной модели. Созданный файл с исходным кодом можно просмотреть, развернув узел **College.edmx** в окне **Solution Explorer** (рис. 10.6).

Кроме того, мастер **Entity Data Model Vizard** добавляет в проект файл конфигурации (если он еще не был создан) и создает в нем строку подключения к базе данных. Эта строка отличается от созданных нами подключений к базе данных. Код файла конфигурации с созданной строкой подключения представлен в листинге 10.1.

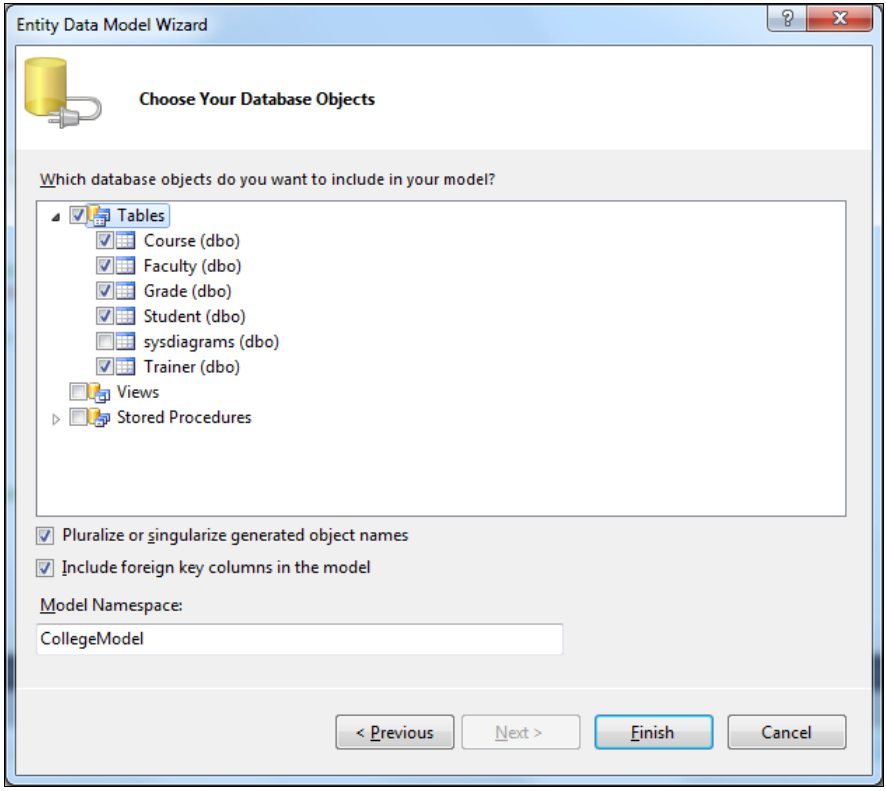


Рис. 10.4. Выбор объектов базы данных `College`

Листинг 10.1. Файл конфигурации и строка подключения к базе данных

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
  <add name="CollegeEntities" connectionString="metadata=
     res://*/College.csdl|res://*/College.ssdl|res://*/College.msl;
     provider=System.Data.SqlClient;
     provider connection string=&quot;Data Source=(local);
     Initial Catalog=College;Integrated Security=True;
     MultipleActiveResultSets=True&quot;"
providerName="System.Data.EntityClient" />
</connectionStrings>
</configuration>
```

Строка подключения содержит ключевые слова, которые определяют параметры подключения к базе данных. Вот основные ключевые слова:

- `provider` — имя поставщика, которое используется для получения объекта `DbProviderFactory`, относящегося к базовому поставщику;

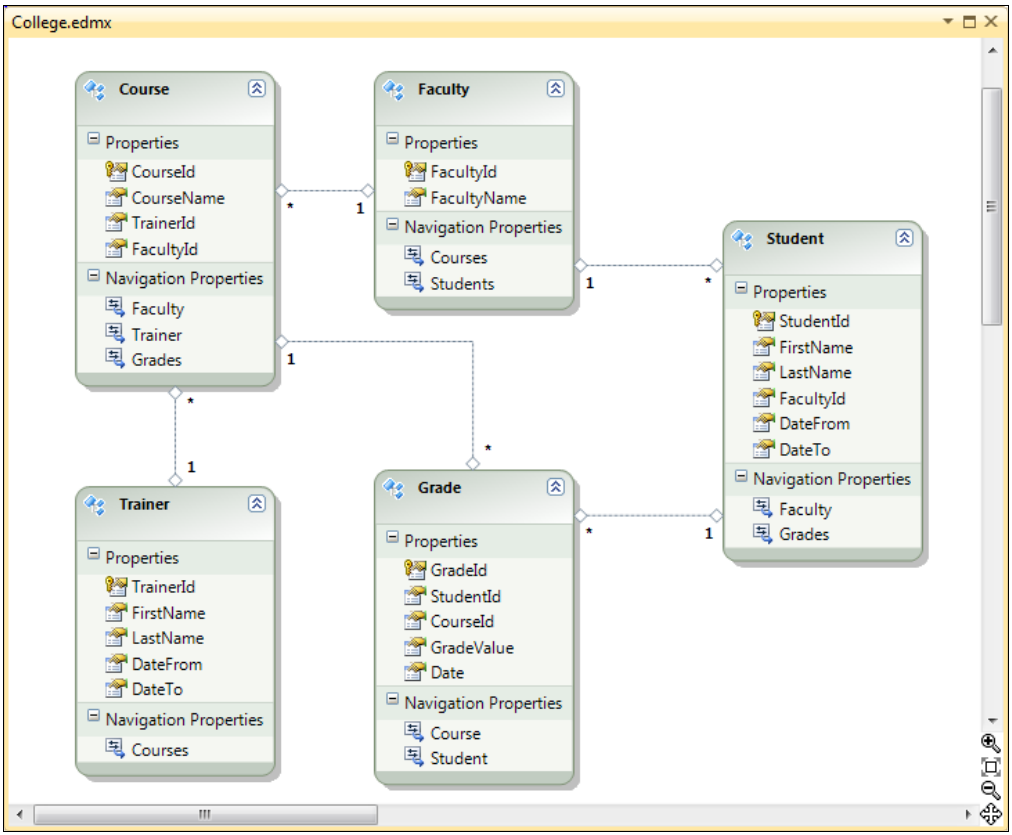


Рис. 10.5. Модель данных EDMX

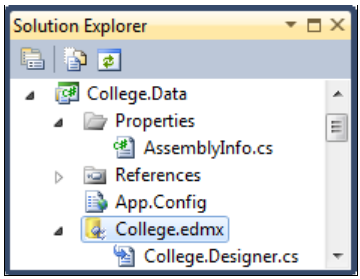


Рис. 10.6. Дерево проекта `College.Data` в **Solution Explorer**

- `provider Connection String` — указывает зависящую от поставщика строку соединения, которая передается в базовый источник данных. Эта строка соединения должна быть выражена с помощью пар "ключ-значение", допустимых для этого поставщика данных;
- `metadata` — список каталогов, файлов и расположений ресурсов, в которых будет выполняться поиск сведений о метаданных и сопоставлениях.

Вы можете посмотреть сгенерированные мастером классы, добавив к проекту диаграмму классов. Диаграмма классов реляционной модели нашей базы данных `College` показана на рис. 10.7.

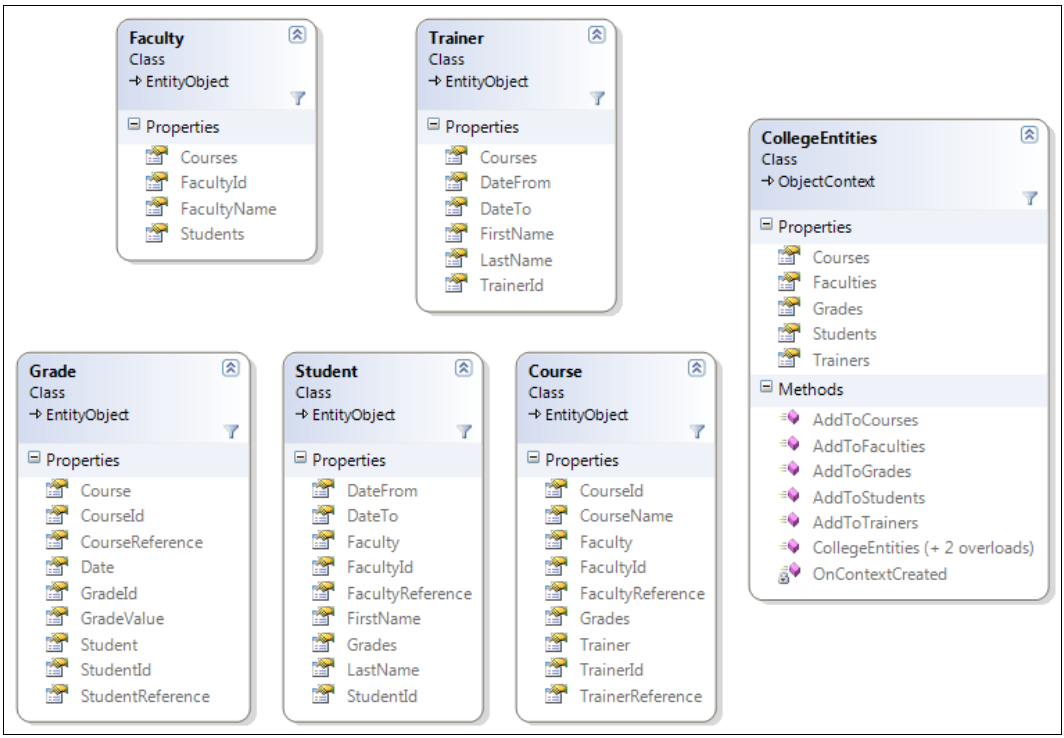


Рис. 10.7. Набор классов, сгенерированных для базы данных `College`

Обратите внимание, что сгенерированный набор классов и их интерфейс для базы данных `College` несколько отличается от классов LINQ to SQL в *главе 9*. Класс `CollegeEntities` наследуется от класса `ObjectContext`. Как уже упоминалось ранее, класс `ObjectContext` — это базовый класс для взаимодействия с данными как с объектами, которые являются экземплярами типов сущностей, определенных в концептуальной модели данных. Класс `ObjectContext` содержит функциональность для подключения к базе данных в виде набора методов и метаданные, описывающие модель.

Для работы с Entity Data Model предусмотрено окно **Model Browser** (Обозреватель моделей). Это окно содержит древовидное представление концептуальной модели и режима хранения, определенных в EDMX-файле (рис. 10.8).

Обозреватель **Model Browser** группирует информацию в два узла. Узел **CollegeModel** отображает концептуальную модель базы данных. Путем развертывания дочерних узлов можно просматривать все типы сущностей и взаимосвязей в модели.

Узел **CollegeModel.Store** показывает целевую модель базы данных в режиме хранения. Путем развертывания дочерних узлов можно увидеть, какие части таблиц базы данных, представлений и хранимых процедур были импортированы в модель.

Model Browser позволяет выполнять следующие действия:

- изменять свойства и сопоставления;
- импортировать хранимые процедуры из базы данных;

- обновлять режим хранения после изменения базы данных;
- удалять таблицы, представления и хранимые процедуры из режима хранения.

Обозреватель моделей открывается при открытии конструктора сущностей. Если обозреватель моделей не отображается, щелкните правой кнопкой мыши главную область конструктора и выберите пункт **Model Browser**.

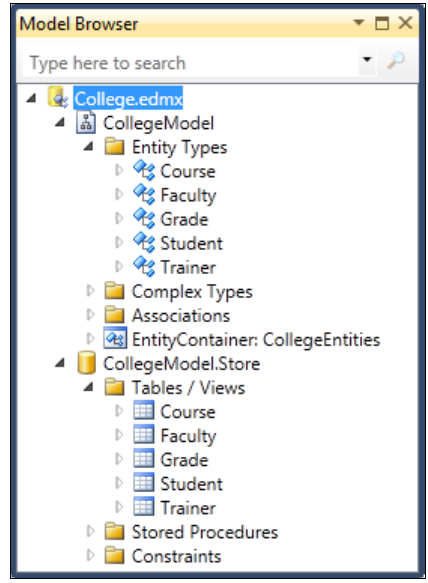


Рис. 10.8. Окно **Model Browser**

Создание приложения для работы с данными

Для отображения данных мы создадим приложение, взяв за основу приложение с использованием технологии LINQ to SQL из предыдущей главы. Для работы с EDM в приложении нам потребуется создать в проекте ссылку на .NET-сборку `System.Data.Entity` (рис. 10.9).

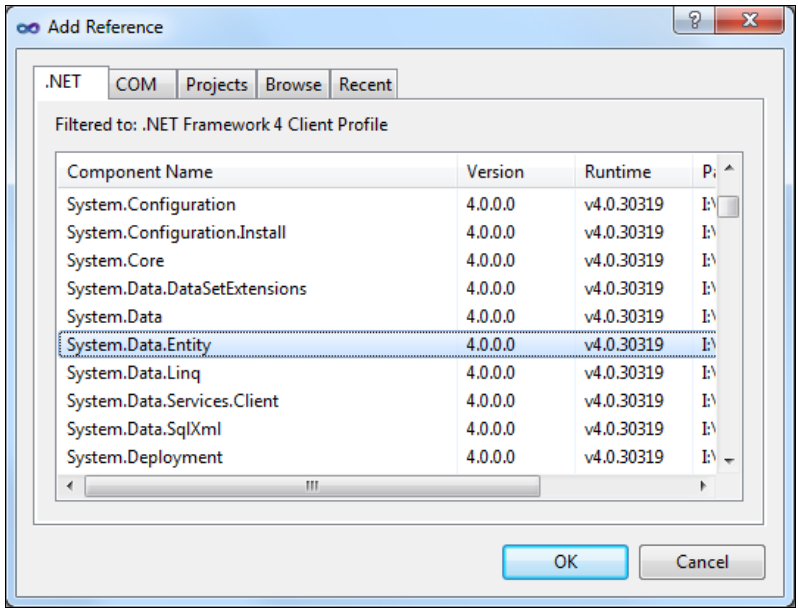


Рис. 10.9. Подключение ссылки на сборку `System.Data.Entity`

Для использования EDM в коде приложения для чтения данных из БД `College` надо будет внести только некоторые небольшие изменения в код класса главной формы приложения, как показано в листинге 10.2.

> ***Примечание***
>
> Полный код приложения на прилагаемом к книге диске находится в каталоге Chapter10\College.UI.v1.

Листинг 10.2. Изменения в коде приложения для чтения данных из БД `College`

```
private CollegeEntities collegeContext;
...
private void FMain_Load(object sender, EventArgs e)
{
   collegeContext = new CollegeEntities();
   ...
}
```

В обработчике события `AfterSelect` элемента управления `TreeView` для выбора объектов из базы данных даже не надо производить никаких изменений, мы можем использовать те же запросы LINQ, что и в приложении из *главы 9*.

Скомпилируйте и запустите приложение. Результат работы приложения представлен на рис. 10.10.

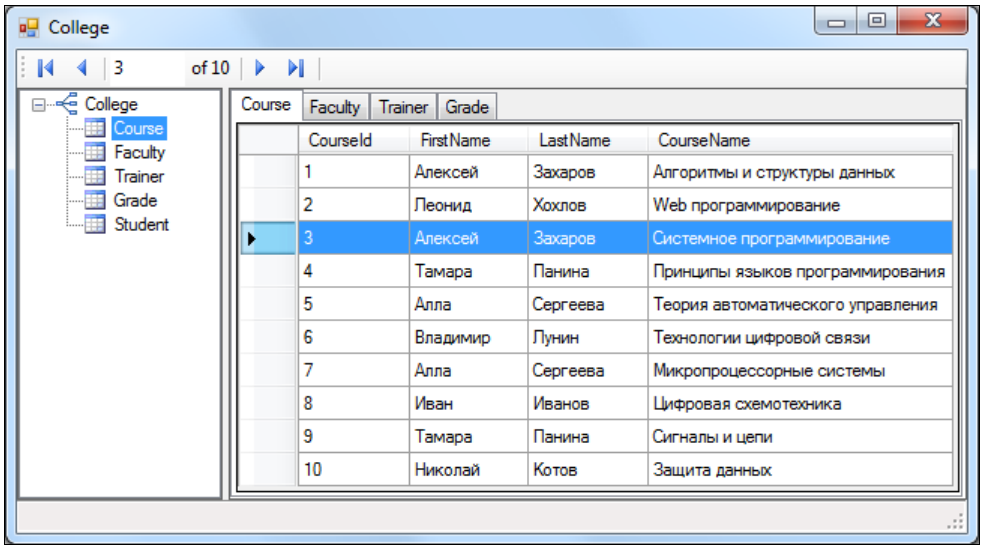


Рис. 10.10. Готовое приложение, использующее Entity Data Model

Поэкспериментируйте с приложением, создав функциональность для вставки, модификации и удаления данных, аналогично той, которую мы создавали в предыдущих главах, посвященных работе с данными.

Комплексные типы

В EDM есть структуры данных, называющиеся *комплексными типами*, которые служат для представления набора свойств, тесно связанных между собой. Например, можно создать типы `StudentFullName` и `TrainerFullName`. С помощью сложных типов осуществляется логическая группировка похожих скалярных свойств, что упрощает поиск тесно связанных свойств сущности и сохраняет эту логическую группировку в EDM. Как и сущности, комплексные типы содержат скалярные свойства, однако в отличие от свойства у них нет идентификатора (значений ключа) и их нельзя сохранять в базу данных через класс, производный от `ObjectContext`. Комплексные типы всего лишь стороны сущностей, но не сами сущности.

Комплексные типы также требуются при использовании хранимых процедур из базы данных. Например, в нашей базе данных `College` все хранимые процедуры `*Select` возвращают не только данные из конкретной таблицы, но и данные из связанных таблиц. Далее мы рассмотрим особенности работы с хранимыми процедурами более подробно.

Импорт хранимых процедур из базы данных

Хранимая процедура, добавленная в концептуальную модель, называется *импортом функции*. Добавление импорта функции позволяет вызывать соответствующую хранимую процедуру из кода приложения. Импорт функции может возвращать коллекции простых типов, типов, определенных EDM или комплексными типами, либо не возвращать значение (`void`).

Обновление модели хранения данных

Для импорта хранимых процедур надо изменить EDMX-файл нашей базы данных. Для обновления EDMX-файла используется мастер обновления моделей — **Update Wizard**. Чтобы вызвать мастер, перейдите в окно **Model Browser** (Браузер моделей), выделите корневой узел модели нашей базы данных **College.edmx**, откройте мышью контекстное меню и выберите пункт **Update Model from Database** (Обновить модель из базы данных).

В открывшемся диалоговом окне **Update Wizard** (рис. 10.11) пометьте все пользовательские процедуры в базе данных `College` (мастер **Update Wizard** отображает также и системные процедуры).

Выбранные нами хранимые процедуры появятся в узле **CollegeModel.Store** в каталоге **Stored Procedures**. Однако это еще не все. Для хранимых процедур, возвращающих набор данных, необходимо определить комплексные типы для возвращаемых значений.

Для этого выделите нужную хранимую процедуру в каталоге **Stored Procedures**. Щелкните правой кнопкой мыши и в контекстном меню выберите пункт **Add Function Import** (Добавить импорт функции). Откроется диалоговое окно **Add Function Import** (рис. 10.12) с именем выбранной хранимой процедуры в списке

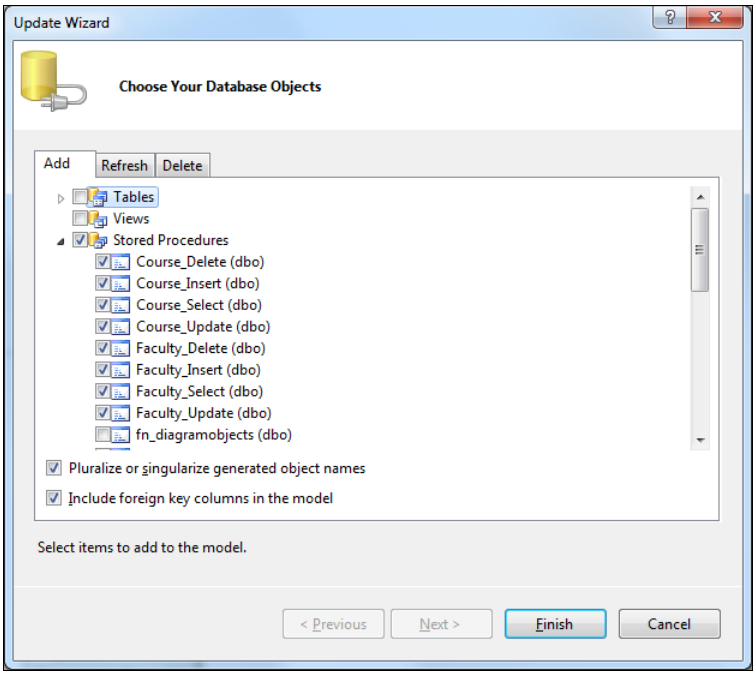


Рис. 10.11. Мастер обновления модели EDMX

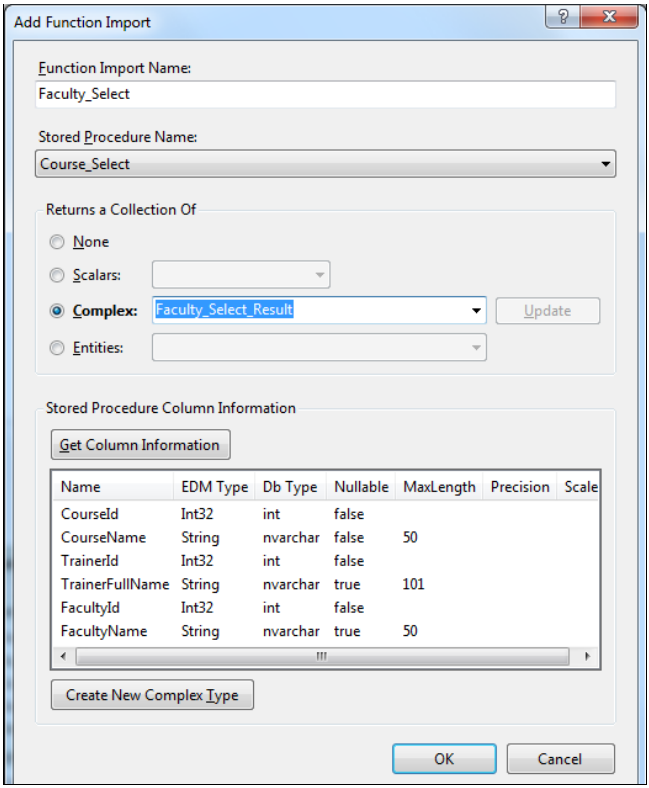


Рис. 10.12. Окно **Add Function Import**

Stored Procedure Name и импортируемым именем по умолчанию (**Function Import Name**), которое при желании можно изменить.

В группе **Returns a Collection Of** укажите возвращаемый тип **Complex**. Далее нажмите кнопку **Get Column Information**, чтобы получить информацию о типах в возвращаемом наборе данных. После этого нажмите кнопку **Create New Complex Type**, и мастер импорта функций создаст новый комплексный тип, название которого отобразится в поле около переключателя **Complex**.

После нажатия кнопки **OK** новый сложный тип добавляется в концептуальную модель, а возвращаемый тип импорта функции устанавливается в созданный новый комплексный тип. Все комплексные типы будут добавлены в узел **CollegeModel** в каталог **Complex Types** (рис. 10.13).

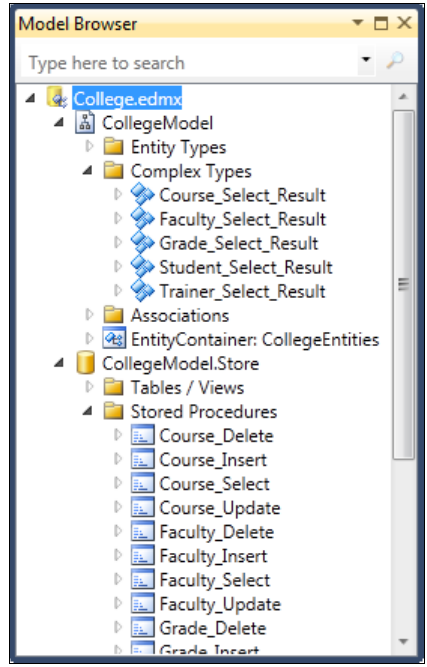


Рис. 10.13. Хранимые процедуры и новые комплексные типы в **Model Explorer**

Вызов хранимых процедур

Вызов хранимых процедур в коде приложения аналогичен вызовам хранимых процедур при использовании других технологий, рассмотренных нами в прошлых главах. Комплексный тип, возвращаемый хранимой процедурой, можно передавать в качестве источника данных объекту `BindingSource` для связывания с элементами отображения данных. Например:

```
bindingSource1.DataSource = collegeContext.Course_Select(null, null, null);
```

Мы можем переделать приложение, созданное нами в предыдущем разделе, на вызовы хранимых процедур, внеся изменения в теле обработчика события в переключателе `switch`, как показано в листинге 10.3.

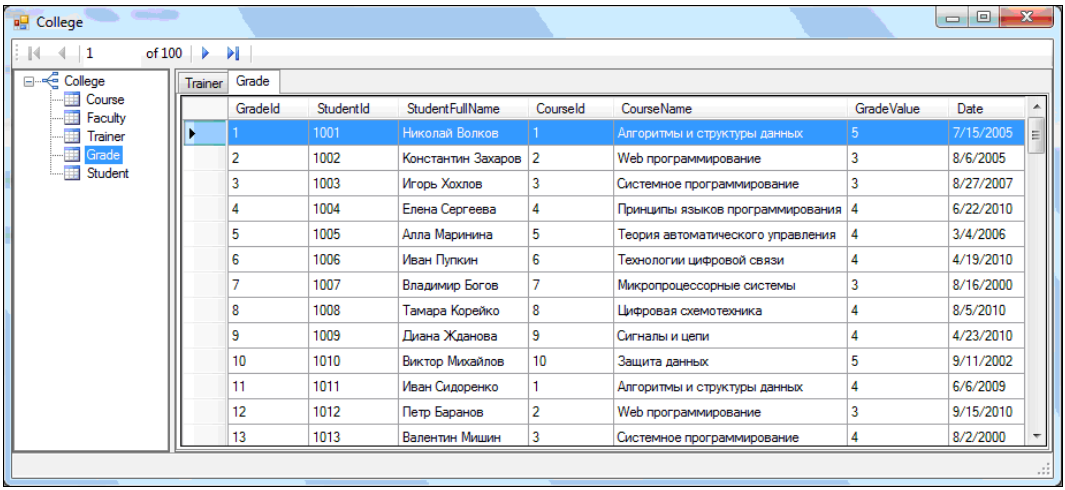
Примечание

Полный код приложения на прилагаемом к книге диске находится в каталоге Chapter10\College.UI.v2.

Листинг 10.3. Вызов хранимых процедур из EDM в коде приложения

```
switch (currentEntity)
{
   case CourseEntity:
     bindingSource1.DataSource = collegeContext.Course_Select(
        null, null, null);
      break;
   case FacultyEntity:
      bindingSource1.DataSource = collegeContext.Faculty_Select(null);
      break;
   case GradeEntity:
      bindingSource1.DataSource = collegeContext.Grade_Select(
         null, null, null);
      break;
   case StudentEntity:
      bindingSource1.DataSource = collegeContext.Student_Select(null, null);
      break;
   case TrainerEntity:
      bindingSource1.DataSource = collegeContext.Trainer_Select(null);
      break;
}
```

Скомпилируйте и запустите приложение. Результат работы приложения представлен на рис. 10.14.



GradeId	StudentId	StudentFullName	CourseId	CourseName	GradeValue	Date
1	1001	Николай Волков	1	Алгоритмы и структуры данных	5	7/15/2005
2	1002	Константин Захаров	2	Web программирование	3	8/6/2005
3	1003	Игорь Хохлов	3	Системное программирование	3	8/27/2007
4	1004	Елена Сергеева	4	Принципы языков программирования	4	6/22/2010
5	1005	Алла Маринина	5	Теория автоматического управления	4	3/4/2006
6	1006	Иван Пупкин	6	Технологии цифровой связи	4	4/19/2010
7	1007	Владимир Богов	7	Микропроцессорные системы	3	8/16/2000
8	1008	Тамара Корейко	8	Цифровая схемотехника	4	8/5/2010
9	1009	Диана Жданова	9	Сигналы и цепи	4	4/23/2010
10	1010	Виктор Михайлов	10	Защита данных	5	9/11/2002
11	1011	Иван Сидоренко	1	Алгоритмы и структуры данных	4	6/6/2009
12	1012	Петр Баранов	2	Web программирование	3	9/15/2010
13	1013	Валентин Мишин	3	Системное программирование	4	8/2/2000

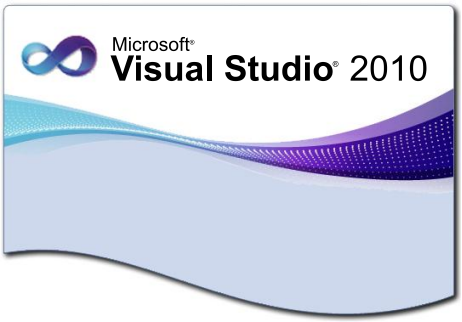
Рис. 10.14. Приложение с использованием вызовов хранимых процедур

Резюме

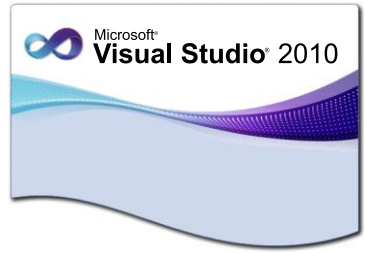
Платформа Entity Framework еще относительно новая технология для доступа к данным, в сравнении с LINQ to SQL из *главы 9* или "классической" ADO.NET. Кроме того, для доступа к данным в .NET существует достаточно много разработанных библиотек от сторонних производителей, например, уже давно используется библиотека NHibernate, портированная на .NET из Java.

Выбор из всех технологий доступа к данным, перечисленных в *главах 7—10*, является, конечно, личным делом разработчика программного обеспечения, в зависимости от его опыта и личных предпочтений.

ЧАСТЬ III



Создание уровня презентаций



ГЛАВА 11

Веб-формы ASP.NET

ASP.NET — это платформа для создания веб-приложений и веб-сервисов, работающих под управлением IIS.

ASP.NET является единой моделью для разработки веб-приложений с применением минимума кода, которая содержит службы, необходимые для построения веб-приложений для предприятий. ASP.NET является частью платформы .NET Framework, а потому обеспечивает доступ к классам этой платформы.

Одним главным отличием платформы ASP.NET является обеспечение полноценной объектной модели на стороне сервера во время выполнения. Платформа ASP.NET предоставляет доступ ко всем элементам управления страницы как к объектам, а элементы управления, размещаемые на веб-страницах, предлагают богатый набор функциональности.

Веб-страницы ASP.NET являются полностью объектно-ориентированными. Со страницами ASP.NET можно работать так же, как и с Windows Forms, используя свойства, методы и события. Структура страниц ASP.NET предоставляет единую модель отклика на клиентские события в коде, выполняемом на сервере, поэтому реализация разделения клиента и сервера, используемая в веб-приложениях, не нужна. Среда выполнения ASP.NET автоматически обрабатывает состояния страницы и ее элементов управления во время цикла обработки страницы.

Жизненный цикл веб-страниц ASP.NET

При выполнении страницы ASP.NET выполняется ряд шагов обработки: инициализация, определение элементов управления, восстановление и поддержание рабочего состояния, выполнение кода обработчика событий, а также отрисовка. Важно ознакомиться с жизненным циклом страницы, чтобы иметь возможность записывать код в нужный шаг жизненного цикла с необходимым результатом.

Основные события веб-страницы

На каждом этапе жизненного цикла страницы генерируются события, которые могут быть использованы для управления логикой работы веб-страницы.

Веб-страницы также поддерживают автоматическую обработку событий. ASP.NET производит поиск методов с соответствующими именами и автоматически выполняет эти методы при возникновении определенных событий. Если атрибут `AutoEventWireup` директивы `@Page` (о структуре веб-страницы, в том числе и о директиве `@page` будет рассказано далее в этой главе) установлен в значение `true`, то события страницы автоматически привязываются к методам, которые используют правила именования `Page_`*событие*, например, `Page_Load` и `Page_Init`.

Далее представлены события жизненного цикла страницы, которые используются наиболее часто.

- `Page_PreInit` — происходит после завершения начального этапа и до начала этапа инициализации.
- `Page_Init` — происходит после инициализации всех элементов управления и применения параметров обложки. Это событие можно использовать для инициализации элементов управления, расположенных на веб-странице.
- `Page_InitComplete` — происходит в конце этапа инициализации страницы, после события `Page_Init`. Между событиями `Page_Init` и `Page_InitComplete` включается отслеживание изменений состояния просмотра. Пока отслеживание состояния просмотра не будет включено, все значения, добавленные в состояние просмотра, теряются во время обратных передач. Это событие используется для внесения изменений в состояние просмотра, которые требуется сохранить после следующей обратной передачи.
- `Page_PreLoad` — происходит после загрузки страницей состояния просмотра и обработки данных обратной передачи, которые были включенны в экземпляр класса `Request`.
- `Page_Load` — происходит после события `Page_PreLoad`. При этом объект `Page` вызывает метод `OnLoad()` для объекта `Page`, а затем рекурсивно повторяет те же действия для каждого дочернего элемента управления, пока не будет загружена сама веб-страница, а также все элементы управления, находящиеся на этой странице. Событие `Load` отдельных элементов управления происходит после события `Page_Load` страницы. Метод события `OnLoad()` используется для задания свойств элементов управления и создания подключений к базам данных.
- `Page_LoadComplete` — происходит в конце этапа обработки события `Page_Load`. Это событие используется для обработки заданий, требующих полной загрузки всех других элементов управления страницы.
- `Page_PreRender` — происходит после создания объектом `Page` всех необходимых элементов управления для отображения страницы, включая дочерние элементы управления составных элементов управления. Для этого объект `Page` вызывает метод `EnsureChildControls()` для каждого элемента управления и для страницы. Объект `Page` вызывает событие `PreRender` в объекте `Page`, а затем рекурсивно повторяет те же действия для каждого дочернего элемента управления, пока не будут загружены страница и все элементы управления. Событие `PreRender` отдельных элементов управления происходит после события `PreRender` страницы. Это

событие используется для внесения окончательных изменений в содержимое страницы или в ее элементы управления до начала визуализации этой страницы.

- `Page_PreRenderComplete` — происходит после того, как каждый элемент управления с привязкой данных, свойство которого `DataSourceID` установлено, вызывает свой метод `DataBind`.
- `Page_SaveStateComplete` — происходит после сохранения состояния просмотра и состояния элемента управления для страницы и для всех элементов управления. Любые изменения страницы или элементов управления в этой точке влияют на визуализацию, однако эти изменения не извлекаются при следующем обратном вызове.
- `Page_Unload` — возникает для каждого элемента управления, а затем для страницы. При работе с элементами управления это событие используется для выполнения окончательной очистки определенных элементов управления, например, для закрытия подключений отдельных элементов управления к базам данных. Относительно страницы это событие используется для выполнения окончательных действий, например, для закрытия открытых файлов и подключений к базам данных, а также закрытия пользовательских сеансов или других задач, связанных с запросами.

Обратная отсылка на сервер

В ASP.NET действия клиента происходят на стороне клиента, а серверная обработка осуществляется на веб-сервере. Основным способом отправки содержимого веб-страницы на сервер является щелчок мышью на кнопке `Submit`. Если используются стандартные серверные элементы управления HTML, это является единственной возможностью для отправки содержимого веб-страницы на сервер.

ASP.NET включает две группы элементов управления: простые серверные элементы управления HTML и более полнофункциональные элементы управления Web. Элементы управления Web ASP.NET, в отличие от обычных элементов управления HTML, дополняют эту модель средством автоматической обратной отсылки. При автоматической обратной отсылке элементы управления вводом могут генерировать различные события, на которые сервер может немедленно отвечать. Например, когда веб-страница отсылается обратно, среда ASP.NET может также генерировать дополнительные события при изменении значений в элементах управления вводом.

Для автоматической обратной отсылки на сервер необходимо установить свойство `AutoPostBack` веб-элемента управления в `true` (значением `AutoPostBack` по умолчанию является `false`).

Состояние вида

В ASP.NET также реализован механизм состояния вида. Когда веб-страница отправляется на сервер, вы получаете всю информацию, введенную пользователем

в любом элементе управления <input> дескриптора <form>. Затем ASP.NET загружает веб-страницу в ее исходном состоянии. При этом получается, что информация будет теряться после каждой обратной отсылки.

Чтобы потери информации не происходило, в ASP.NET используется преобразование состояний в последовательную форму. Как только выполнение кода страницы завершается перед генерацией и отправкой клиенту кода HTML, ASP.NET исследует свойства всех элементов управления вашей страницы. Если любое из свойств изменялось, ASP.NET сохраняет эту информацию в коллекции *имя-значение*. Окончательная строка вставляется в раздел страницы <form> как новое скрытое поле.

При обратной отсылке страницы клиенту среда ASP.NET последовательно выполняет следующие действия.

1. Повторно создает страницу и объекты элементов управления на основе правил по умолчанию. Поэтому страница находится в том же состоянии, в котором была при первом запросе.
2. Выполняет десериализацию информации о состоянии и обновляет все элементы управления. В результате страница возвращается в состояние, в котором находилась перед последней отправкой клиенту.
3. Настраивает веб-страницу в соответствии с отправленными данными формы.
4. Генерирует нужные события, и ваш код может реагировать изменением страницы, перемещением на новую страницу или же выполнением какой-то другой операции.

При использовании состояния вида серверные ресурсы могут освобождаться после каждого запроса, обрабатывая множество запросов без замедления работы сервера. Однако за все приходится платить. Хранение состояния вида на странице увеличивает размеры страницы, т. к. клиенту требуется не только получить страницу больших размеров, но и вернуть скрытые данные о состоянии вида серверу со следующей обратной отправкой. Поэтому много времени уходит на получение и отправку страницы. Для простых страниц эти издержки незначительны, но при настройке сложных элементов управления информация о состоянии вида может существенно увеличиться.

Примечание

ASP.NET применяет состояние вида только к свойствам веб-страниц и элементов управления.

Веб-приложения и веб-сайты

Visual Studio включает несколько шаблонов проектов, которые помогут начать работу при создании нового веб-проекта. Можно создавать проекты веб-приложений или веб-сайтов (рис. 11.1 и 11.2).

Прежде чем приступить к разработке приложения ASP.NET, следует обдумать, какой тип проекта будет наиболее подходящим для данной задачи. Каждый тип

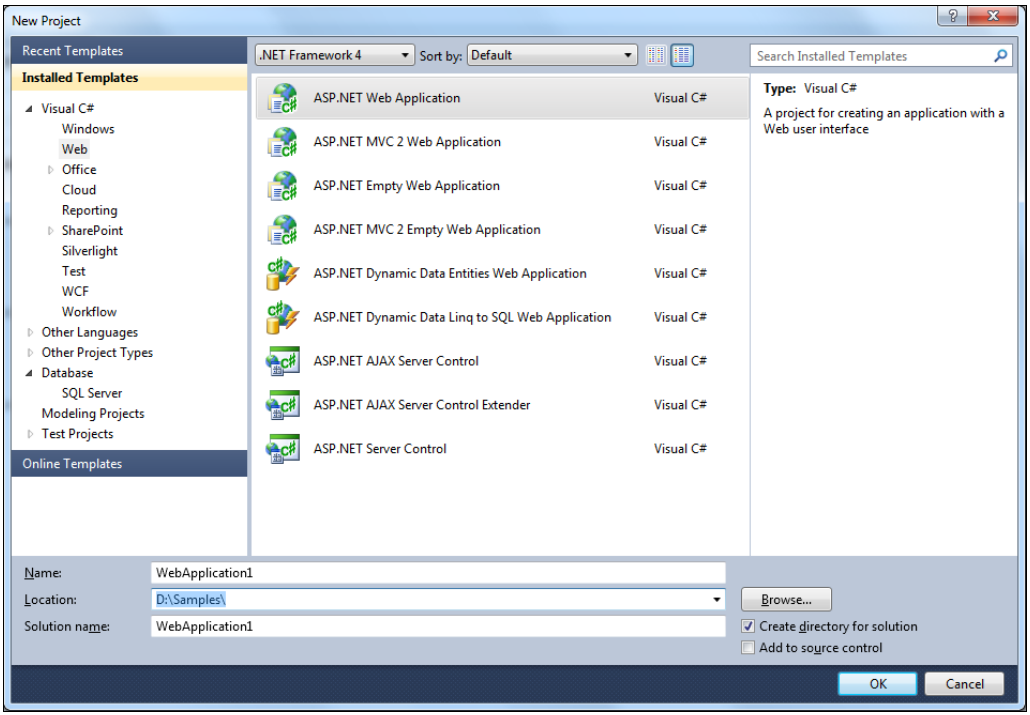


Рис. 11.1. Шаблоны веб-приложений

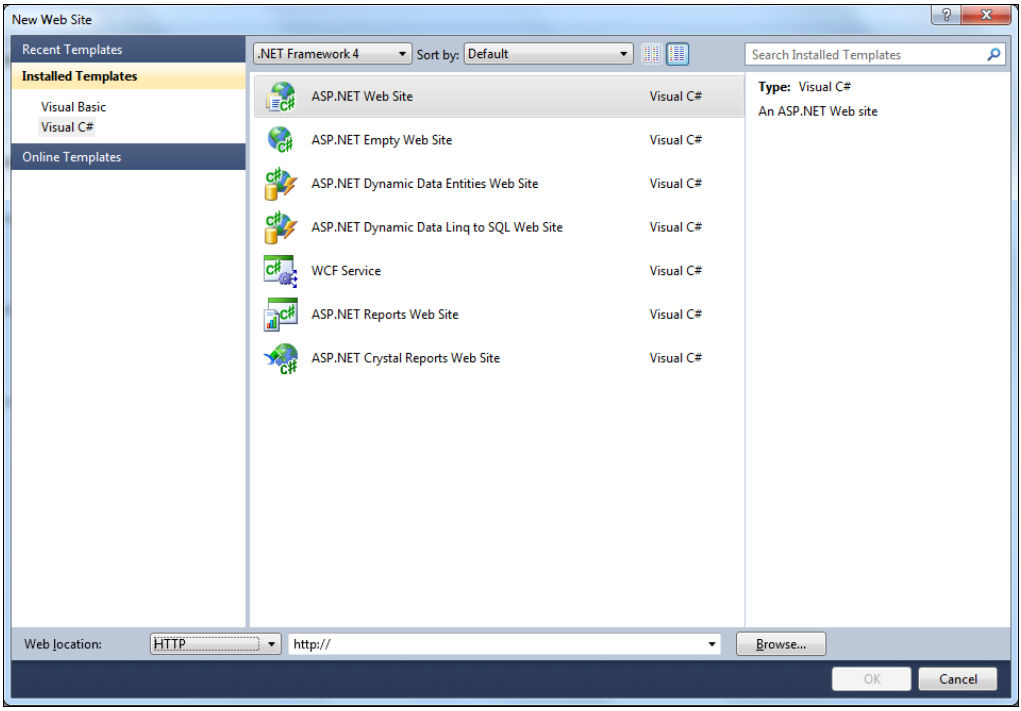


Рис. 11.2. Шаблоны веб-сайтов

имеет свои преимущества и недостатки, поэтому для выбора оптимального варианта необходимо понимать различия между ними. Важно выбрать соответствующий тип проекта до его создания, поскольку преобразование одного типа проекта в другой — сложная задача.

У веб-приложений файл проекта Visual Studio с расширением CSPROJ хранит сведения о проекте, такие как список файлов, входящих в его состав, а также все ссылки между проектами. Проекты веб-приложений используют файлы проектов Visual Studio с расширением CSPROJ для отслеживания сведений о проекте. Помимо других задач, это позволяет указать файлы, добавляемые в проект или исключаемые из него, тем самым определяя файлы, компилируемые во время построения.

Использование файлов проектов веб-приложений имеет определенные преимущества.

- Можно с легкостью временно удалить файлы из сайта и сохранить их отслеживание, поскольку они остаются в структуре папок. Например, если страница не готова для развертывания, можно временно исключить ее из построения, не удаляя из структуры папок. Скомпилированную сборку можно развернуть, а затем снова добавить файл в проект. Это особенно важно при работе с репозиторием системы управления версиями.
- Для веб-сайта файл проекта CSPROJ отсутствует. Все файлы в структуре папок автоматически добавляются в каталог веб-сайта. Если из компиляции требуется исключить какой-либо файл, его необходимо удалить из папки проекта веб-сайта или изменить расширение имени файла на расширение, которое не поддерживается при компиляции и не обслуживается в IIS.

Если использовать структуру папок без файлов проектов веб-сайтов, то управлять структурой проекта исключительно в среде Visual Studio не требуется. Например, можно копировать файлы в каталог веб-сайта или удалять их из каталога.

У веб-приложений исходный код компилируется явным образом. По умолчанию компиляция файлов кода, за исключением файлов ASPX и ASCX, обеспечивает создание единой сборки. Проекты веб-приложений обычно создаются в среде разработки Visual Studio или с помощью пакетного компилятора ASP.NET на компьютере, не являющимся рабочим сервером IIS. Все файлы классов с выделенным кодом и отдельные файлы классов в проекте компилируются в единую сборку, которая затем помещается в папку Bin проекта веб-приложения. Файлы ASPX и ASCX динамически компилируются способом, аналогичным процедуре для проектов веб-сайтов.

Предварительная компиляция для проектов веб-приложений имеет следующие преимущества.

- Предварительная компиляция гарантирует, что пользователям не придется ожидать окончания компиляции сайта на рабочем сервере. Однако необходимо учитывать, что в случае большого размера сайта динамическая компиляция проекта веб-сайта может занять длительное время. Динамическая компиляция выполняется, если запрос о ресурсе сайта получен после обновления сайта. Запрос, запускающий компиляцию, может быть отложен до тех пор, пока требуемые ресурсы не будут скомпилированы. Если задержка недопустима, можно выпол-

нить предварительную компиляцию сайта. Однако в этом случае потеряются все преимущества динамической компиляции веб-сайта.

- Пользователь имеет полный контроль над местом размещения файлов кода в структуре папок проекта, а также каким образом классы в проекте ссылаются друг на друга. Для динамической компиляции требуется, чтобы исходный код для любых классов, использующихся на сайте, хранился в папке App_Code. Нельзя ссылаться на класс страницы или пользовательского элемента управления из класса в папке App_Code.
- Проекты веб-сайтов компилировать вручную не требуется. В большинстве случаев проекты веб-сайтов компилируются динамически при первом получении запроса после установки или обновления сайта средой ASP.NET на сервере IIS. Также веб-сайт можно предварительно компилировать на компьютере разработчика перед развертыванием на сервере IIS.

Различия в производительности между проектами веб-сайтов и веб-приложений нет. Для первого запроса, отправляемого на веб-сайт, может требоваться его компиляция, что, безусловно, может привести к некоторой задержке выполнения запроса.

Выбор места расположения веб-сайта

В окне **New Web Site** можно указать место расположения веб-сайта (выпадающий список **Web location**), которое определяет место хранения файлов веб-сайта. Этот список содержит следующие значения:

- **File System** (Файловая система);
- **Local IIS** (Локальный IIS);
- **FTP Site** (FTP-сайт);
- **Remote Site** (Удаленный сайт).

Место расположения веб-сайта **File System** является самым простым выбором места для веб-сайта (рис. 11.3).

Если вы выбираете для локальной разработки веб-сайта место расположения **File System**, то вам не нужно иметь на вашем компьютере сервер IIS. При этом Visual Studio распознает содержимое каталога файловой системы как веб-сайт и запускает локальный экземпляр тестового сервера ASP.NET Development Server. Этот сервер имитирует IIS и его при необходимости можно запускать и останавливать.

Место расположения веб-сайта **Local IIS** позволяет создать более высокую степень управления конфигурацией веб-сайта (рис. 11.4).

При выборе места расположения **FTP Site** перед подключением необходимо будет ввести всю информацию о соединении, включая FTP-сайт, порт, каталог, имя пользователя и пароль (рис. 11.5).

Место расположения веб-сайта **Remote Site** предоставляет доступ к веб-сайту по определенному URL с использованием протокола HTTP (рис. 11.6).

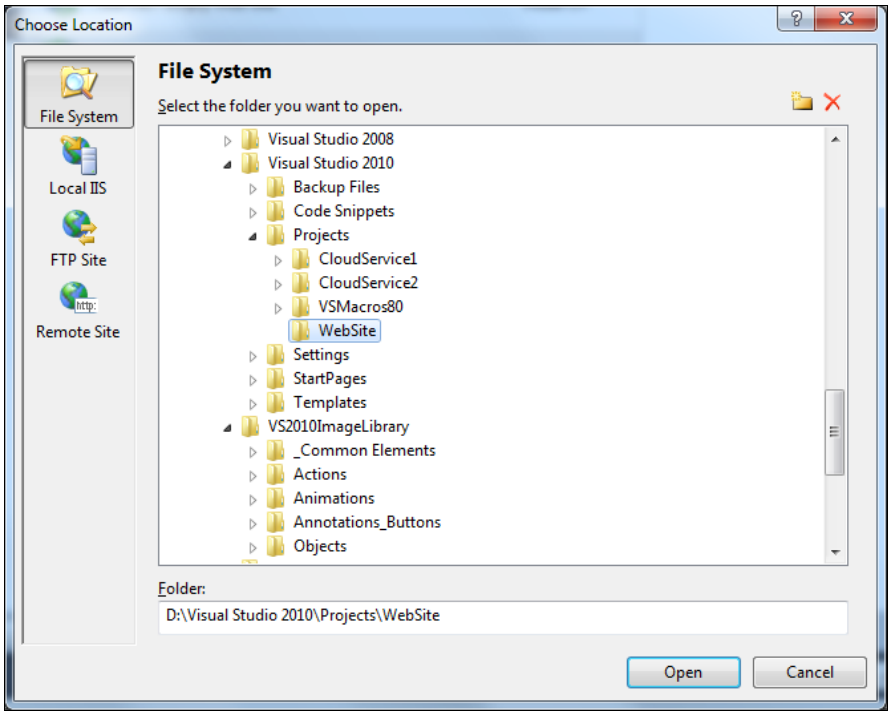


Рис. 11.3. Выбор файловой системы для размещения веб-сайта

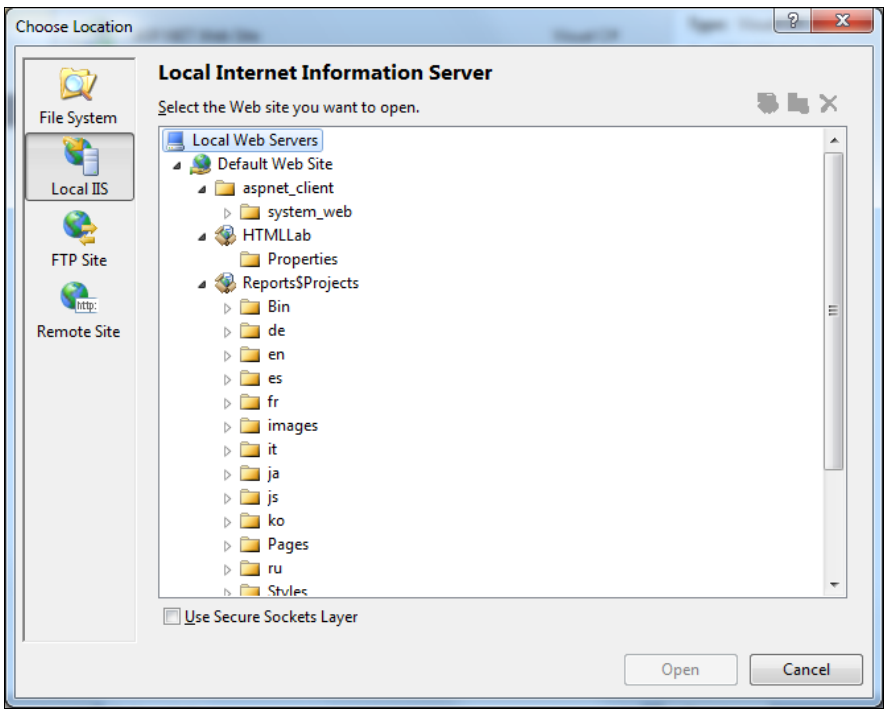


Рис. 11.4. Выбор IIS для размещения веб-сайта

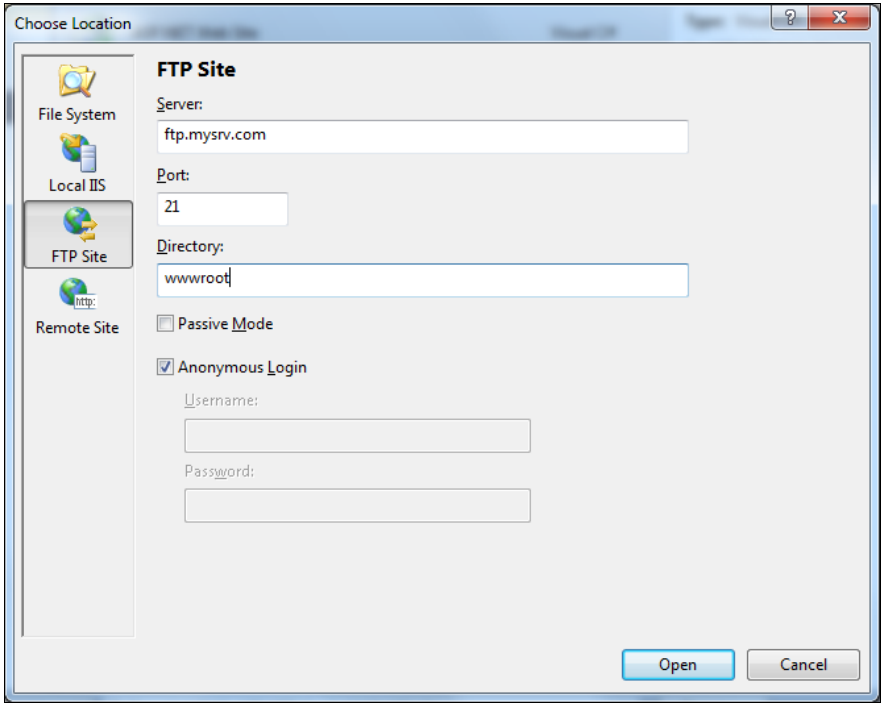


Рис. 11.5. Выбор FTP-сайта для размещения веб-сайта

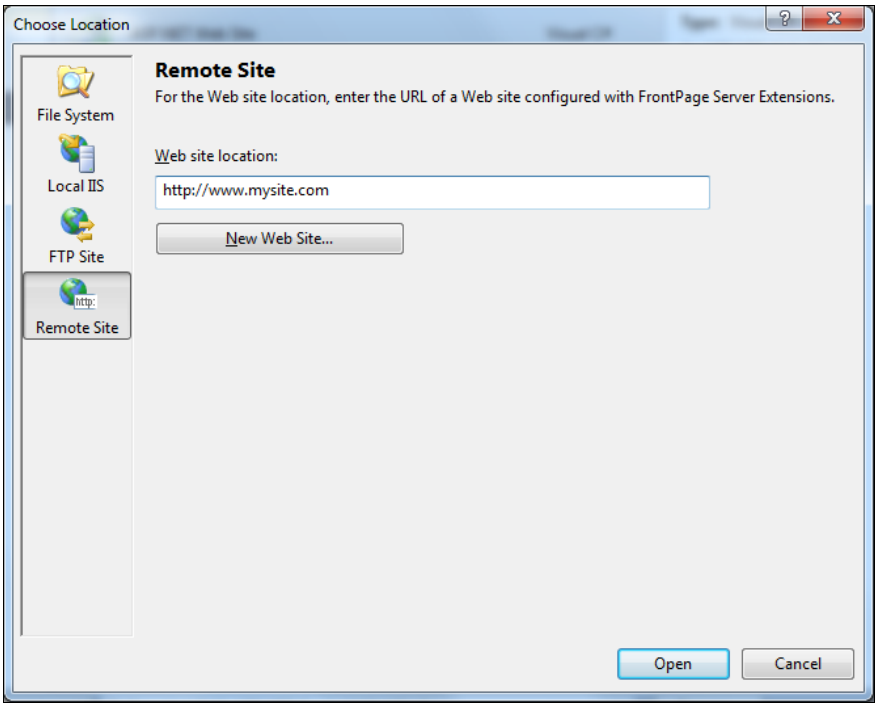


Рис. 11.6. Выбор **Remote Site** для размещения веб-сайта

Веб-формы

Веб-формы — это объекты веб-приложения, определяющие его пользовательский интерфейс. В целом веб-формы аналогичны Windows-формам. Однако, в отличие от Windows-приложений, веб-приложения работают на сервере, обмениваясь данными с клиентами через Интернет. Поэтому в Visual Studio создание проекта веб-приложения и управление его файлами существенно отличается от создания проектов для Windows-приложений.

Давайте создадим наш первый проект ASP.NET и рассмотрим принципы разработки веб-форм. В главном меню выберем пункт **File | New | Web Site** и в открывшемся диалоговом окне **New Web Site** выберем шаблон **Empty Web Site** и размещение **File System**. Создав каталог для сайта с именем WebSite, нажмем кнопку **OK**. Новый веб-сайт, созданный средой Visual Studio, не содержит веб-страниц, в нем есть только файл конфигурации web.config.

Теперь добавим веб-форму к созданному сайту (рис. 11.7). Оставим ей имя по умолчанию (Default.aspx).

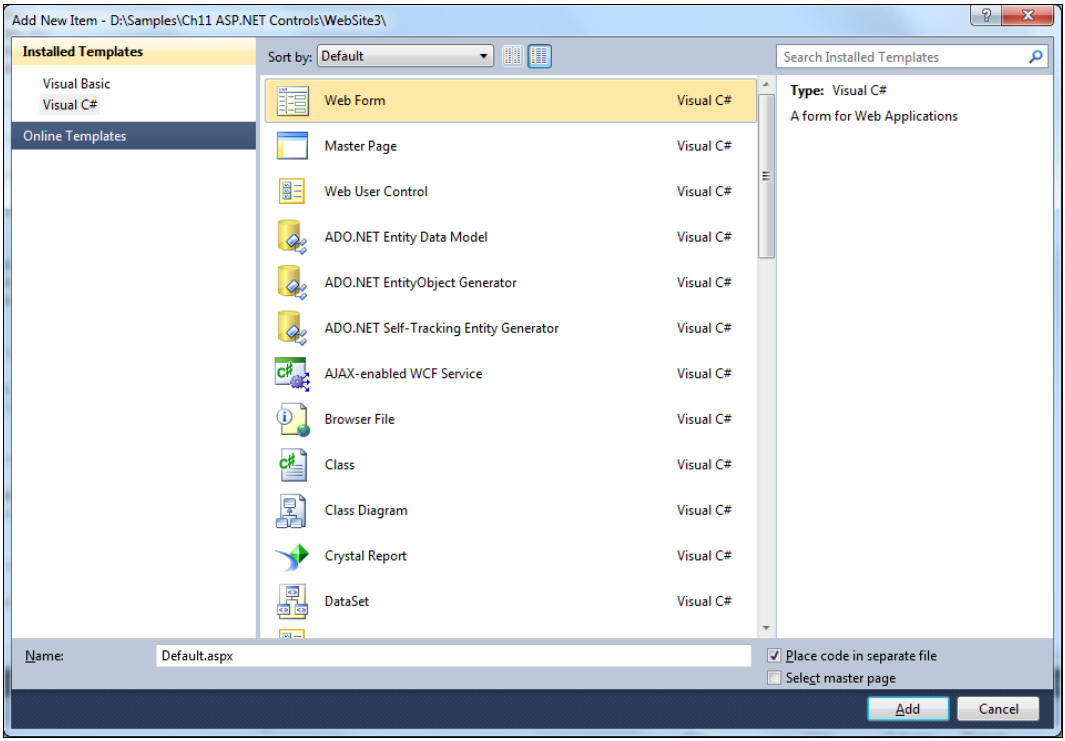


Рис. 11.7. Добавление веб-формы к сайту

Если дважды щелкнуть мышью на файле Default.aspx, откроется дизайнер веб-форм. Он имеет в нижней части три закладки: **Design**, **Split** и **Source**. Переключимся на закладку **Source** и рассмотрим код для пустой формы, сгенерированный средой разработки.

В ASP.NET все элементы управления помещены в отдельном дескрипторе `<form>`. Этот дескриптор помечен атрибутом `runat="server"`, который позволяет ему работать на сервере. ASP.NET не допускает создание веб-форм, содержащих более одного серверного дескриптора `<form>`.

Код пустой веб-формы представлен в листинге 11.1.

Листинг 11.1. Код пустой формы Default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!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></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

Помимо файла Default.aspx, среда разработки создает файл C#-кода Default.aspx.cs, код которого приведен в листинге 11.2.

Листинг 11.2. Файл Default.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
}
```

Обработчик события загрузки веб-страницы `Page_Load()` в листинге 11.2 может быть использован для добавления любого необходимого в момент загрузки страницы кода. По мере добавления новых обработчиков событий этот класс будет наполняться кодом. Обратите внимание, что здесь нет кода, который связывает обработчик событий со страницей — это делает исполняющая система ASP.NET. Такое поведение устанавливается атрибутом `AutoEventWireUp=true` (см. листинг 11.1) — присваивание ему значения `false` потребовало бы ассоциировать обработчики с событиями вручную.

Внутри дескриптора `<form>` на веб-странице можно размещать дескрипторы для различных элементов управления: кнопок, текстовых полей, окон списков и переключателей.

При отправке введенных на форме данных на сервер браузер извлекает текущие значения каждого элемента управления и формирует из них длинную строку. Затем эта строка отправляется странице, указанной в дескрипторе `<form>`, например, с использованием HTTP-операции `POST`. Данные со страницы всегда отправляются в виде последовательности пар *`имя-значение`*, разделенных знаком амперсанда `&`. Каждая пара *`имя-значение`* отделяется знаком равенства `=`.

Среда выполнения ASP.NET позволяет извлекать значение элемента управления формы и сохраняет его в коллекции `Form` объекта `Request`. В ASP.NET можно искать значения по имени в коллекции `Form`, например, так:

```
string firstName = Request.Form["FirstName"];
```

Когда страница отправляется обратно на сервер, она также извлекает значения, заполняет коллекцию `Form` и конфигурирует соответствующие объекты элементов управления. Это означает, что для извлечения информации можно также использовать следующий намного более содержательный синтаксис:

```
string firstName = txtFirstName.Text;
```

Данный код также обладает преимуществом безопасности типов. В результате разработчики веб-страниц оказываются изолированными от индивидуальных особенностей синтаксиса HTML.

Серверные элементы управления

Серверные элементы управления представляют собой классы в среде .NET Framework, соответствующие визуальным элементам веб-формы. Некоторые из этих классов являются относительно простыми и отображаются на определенные HTML-дескрипторы. Другие элементы управления являются более абстрактными и реализуют более сложное представление нескольких элементов HTML.

ASP.NET предлагает множество серверных элементов управления, которые разделены на несколько групп.

Все серверные элементы управления наследуются от базового класса `Control` из пространства имен `System.Web.UI`. Далее перечислены наиболее часто используемые методы класса `Control`.

- `ClientID` — идентификатор элемента, который является уникальным именем, созданным ASP.NET при создании экземпляра страницы.

- `Controls` — коллекция дочерних элементов управления страницы.
- `EnableViewState` — указывает на то, должен ли элемент управления поддерживать свое состояние с помощью обратных отсылок своей родительской страницы.
- `ID` — идентификатор элемента управления. Это имя, с помощью которого вы можете получить доступ к элементу управления из программного кода.
- `PageParent` — возвращает ссылку на объект родительской страницы.
- `Visible` — определяет видимость элемента управления.
- `DataBind()` — связывает элемент управления с источником данных.
- `FindControl()` — ищет дочерний элемент управления с определенным именем в текущем элементе управления и всех содержащихся в нем элементах. Если дочерний элемент найден, метод возвращает ссылку на общий тип `Control`, который затем можно привести к соответствующему типу.
- `HasControls()` — указывает на наличие у элемента управления дочерних элементов управления. Для того чтобы содержать дочерние элементы, элемент управления должен быть дескриптором контейнера, например, `<div>`.
- `RenderControl()` — записывает вывод HTML для элемента управления на основании его текущего состояния. Этот метод не вызывается напрямую, его вызывает ASP.NET при генерации страницы.

Элементы управления HTML

Серверные элементы управления HTML определены в пространстве имен `System.Web.UI.HtmlControls`. Они разбиты на отдельные категории на основании того, являются ли они элементами управления вводом и унаследованы от класса `HtmlInputControl` или могут содержать другие элементы управления и являются производными от класса `HtmlContainerControl`. На рис. 11.8 показана иерархия наследования.

Класс `Control` мы уже рассмотрели в предыдущем разделе. Все серверные элементы управления HTML являются производными от базового класса `HtmlControl`, наследующего от класса `Control`. Основные свойства, определяемые в классе `HtmlControl`:

- `Attributes` — коллекция атрибутов дескриптора элемента управления;
- `Disabled` — состояние доступности элемента управления;
- `Style` — коллекция атрибутов CSS, применяемых к элементу управления. На веб-странице свойство `Style` устанавливается как список атрибутов `style:`, разделенных точками с запятой (работа со стилями будет рассматриваться в следующей главе);
- `TagName` — имя дескриптора элемента управления.

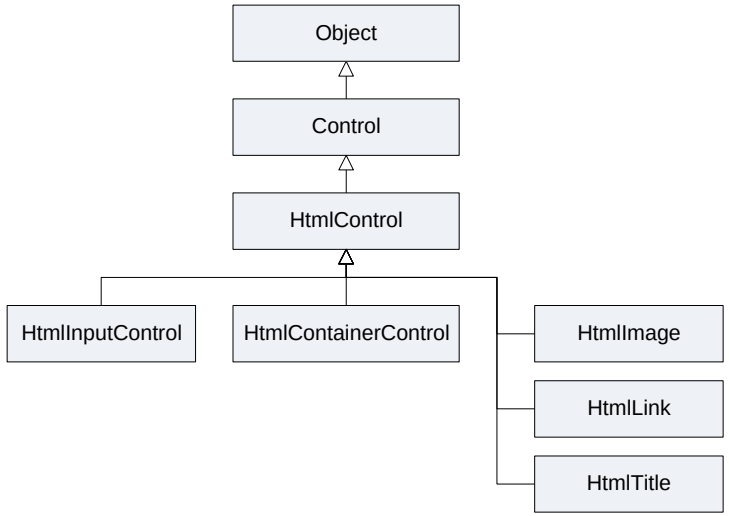


Рис. 11.8. Серверные элементы управления HTML

Элементы управления вводом HTML допускают взаимодействие с пользователем. Сюда входят знакомые графические элементы управления — флажки, текстовые поля, кнопки и окна со списком. Все эти элементы управления используют тег `<input>`.

Элементы управления вводом HTML наследуются от класса `HtmlInputControl`, который добавляет новые свойства к элементам управления:

- `Type` — предоставляет тип `HtmlInputControl`. Например, если это свойство установлено в `text` (текст), `HtmllnputControl` является текстовым полем для ввода данных;
- `Value` — предоставляет либо устанавливает значение, связанное с элементом управления вводом. Значение, связанное с элементом управления, зависит от типа элемента управления. Например, в текстовом поле это свойство хранит текст, введенный в элементе управления. Для кнопок оно определяет текст кнопки.

Указание атрибута `runat="server"` позволяет ASP.NET обрабатывать их и транслировать в экземпляры соответствующего класса .NET.

Серверные элементы управления HTML реализуют модель нечастых событий, включающую только два возможных события:

- `ServerClick()` — это событие, реагирующее на щелчок мышью на элементе управления, обрабатываемое на стороне сервера. Событие `ServerClick()` предоставляется большинством кнопок;
- `ServerChange()` — это событие генерируется при внесении изменений в элемент ввода текста или при выборе пользователем опции в элементах-списках. Это событие не происходит до обратной отсылки страницы (например, после щелчка пользователя на кнопке `Submit`). При отправке страницы на сервер для всех из-

мененных элементов управления генерируется событие `ServerChange`, за которым следует соответствующее событие `ServerClick`.

Чтобы изучить работу с серверными элементами управления HTML на практике, давайте создадим веб-страницу с этими элементами. Добавим в наш веб-сайт новую страницу с именем HtmlControls.aspx. Эта страница будет имитировать форму регистрации нового абитуриента на веб-сайте колледжа (в следующих главах мы будем развивать этот веб-сайт, добавляя к нему новую функциональность) и будет содержать элементы для ввода текста, элементы-списки, переключатели и кнопку `Submit` для отправки данных на сервер. Кроме того, мы рассмотрим обработку событий элементов управления HTML.

В файле разметки веб-страницы HtmlControls.aspx напишите код, представленный в листинге 11.3.

Примечание

Полный код веб-страницы на прилагаемом к книге диске можно найти в каталоге Chapter11\Controls\HtmlControls.aspx.

Листинг 11.3. Пример веб-страницы с элементами управления HTML

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="HtmlControls.aspx.cs"
Inherits="HtmlControls" %>

<!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 id="Head1" runat="server">
  <title>HTML Controls</title>
  <style type="text/css">
    .column0{width: 150px;}
    .column1{width: 350px;}
  </style>
</head>
<body>
<div><h2>Регистрация абитуриента</h2></div>
  <form id="form1" runat="server">
  <div>
    <table>
      <tr>
        <td class="column0">Имя</td>
        <td class="column1">
         <input type="text" runat="server" id="TextFirstName"
           onserverchange="InputControls_ServerChange"/>
        </td>
      </tr>
      <tr>
```

```
  <td class="column0">Фамилия</td>
  <td class="column1">
    <input type="text" runat="server" id="TextLastName"
      onserverchange="InputControls_ServerChange"/>
  </td>
</tr>
<tr>
  <td class="column0">Год рождения</td>
  <td class="column1">
   <select runat="server" id="ListYearBirth"
     onserverchange="Select_ServerChange" ></select>
  </td>
</tr>
<tr>
  <td class="column0">E-mail</td>
  <td class="column1">
   <input type="text" runat="server" id="TextEMail"
     onserverchange="InputControls_ServerChange"/>
  </td>
</tr>
<tr>
  <td class="column0">Факультет</td>
  <td class="column1">
    <select runat="server" id="ListFaculties"
      onserverchange="Select_ServerChange" >
    </select>
  </td>
</tr>
<tr>
  <td class="column0">Форма обучения</td>
  <td class="column1">
    <p><input type="radio" runat="server" id="RadioFullTime"
         checked="true" name="RadioGroup" value="V1"
         onserverchange="RadioButton_ServerChange"/>Дневная</p>
    <p><input type="radio" runat="server" id="RadioPartTime"
         name="RadioGroup" value="V2"
         onserverchange="RadioButton_ServerChange"/>Вечерняя</p>
   </td>
</tr>
<tr>
  <td class="column0">Дополнительная информация о себе</td>
  <td class="column1">
    <textarea id="TextAreaInfo" runat="server" rows="5" cols="20"
      name="S1" onserverchange="TextArea_ServerChange">
    </textarea>
  </td>
</tr>
```

```
      </table>
  </div>
  <br />
  <div>
   <input type="submit" runat="server" id="SubmitData" value="Отправить"
     onserverclick="SubmitData_ServerClick" />
  </div>
  <br />
   <textarea id="TextAreaOutput" runat="server" rows="5" cols="60" name="S1"
     visible="false">
   </textarea>
 </form>
</body>
</html>
```

В результате должно получиться расположение элементов управления, как показано на рис. 11.9.

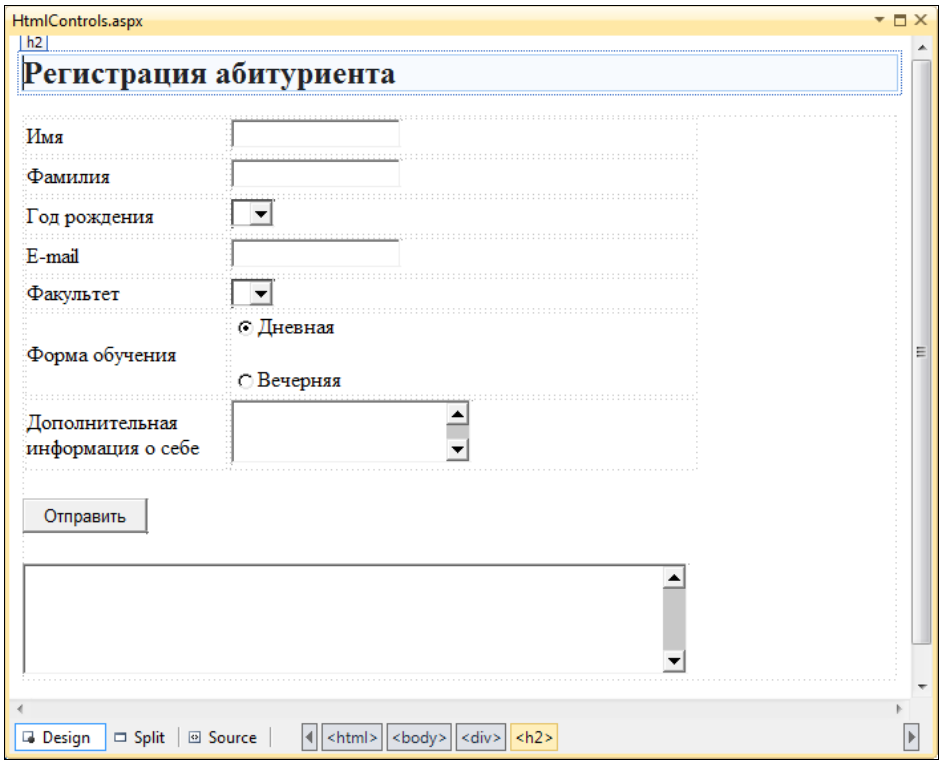


Рис. 11.9. Страница регистрации с серверными элементами управления HTML

Затем в программной части класса `HtmlControls` мы напишем обработчики событий для элементов управления, причем создадим общие обработчики событий для групп элементов. Для разных групп элементов управления тело обработчиков со-

бытий будет различаться. Кроме обработчика загрузки страницы `Page_Load()` будет еще 5 обработчиков событий:

- `InputControls_ServerChange()` — для элементов `<input type="text">`: `TextFirstName`, `TextLastname`, `TextEMail`;
- `Select_ServerChange()` — для элементов `<select>`: `ListYearBirth` и `ListFaculties`;
- `TextArea_ServerChange()` — для элемента `<textarea>`: `TextAreaInfo`;
- `RadioButton_ServerChange()` — для элементов `<input type="radio">`: `RadioFullTime` и `RadioPartTime`;
- `SubmitData_ServerClick()` — для кнопки `Submit`.

Код программной части класса `HtmlControls` представлен в листинге 11.4.

Примечание

Полный код программной части класса `HtmlControls` на прилагаемом к книге диске можно найти в каталоге Chapter11\Controls\HtmlControls.aspx.cs.

Листинг 11.4. Файл HtmlControls.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Text;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;

public partial class HtmlControls : System.Web.UI.Page
{
    // переменная для вывода суммарной информации на страницу
    StringBuilder output;

    protected void Page_Load(object sender, EventArgs e)
    {
       output = new StringBuilder();
       // заполняем список ListYearBirth
       for (int i = DateTime.Today.Year; i >= 1900; i--)
       {
          ListYearBirth.Items.Add(i.ToString());
       }
       // заполняем список ListFaculties
       ListFaculties.Items.Add("Информационные технологии");
       ListFaculties.Items.Add("Электроника и телекоммуникации");
    }
```

```
    protected void InputControls_ServerChange(object sender, EventArgs e)
    {
      output.AppendLine(String.Format("ServerChange for {0}, value={1}",
        ((HtmlInputControl)sender).ID, ((HtmlInputControl)sender).Value));
    }

    protected void Select_ServerChange(object sender, EventArgs e)
    {
       output.AppendLine(String.Format("ServerChange for {0}, value={1}",
         ((HtmlSelect)sender).ID, ((HtmlSelect)sender).Value));
    }

    protected void TextArea_ServerChange(object sender, EventArgs e)
    {
       output.AppendLine(String.Format("ServerChange for {0}, value={1}",
         ((HtmlTextArea)sender).ID, ((HtmlTextArea)sender).Value));
    }

    protected void RadioButton_ServerChange(object sender, EventArgs e)
    {
       output.AppendLine(String.Format("ServerChange for {0}, value={1}",
         ((HtmlInputRadioButton)sender).ID,
         ((HtmlInputRadioButton)sender).Checked ? "true" : "false"));
    }

    protected void SubmitData_ServerClick(object sender, EventArgs e)
    {
       TextAreaOutput.Value = output.ToString();
       TextAreaOutput.Visible = true;
    }
}
```

Запустите страницу в браузере, выберите несколько элементов в окне списка, введите информацию в поля и щелкните на кнопке **Отправить** для генерации обратной отсылки. В поле `TextAreaOutput` отобразится информация, созданная в обработчиках событий. Результат показан на рис. 11.10.

Следует отметить, что нельзя полагаться на то, что события произойдут в каком-то фиксированном порядке. Однако вы наверняка заметите, что события запускаются в порядке объявления элементов управления. Гарантируется лишь то, что все события `ServerChange` генерируются перед событием `ServerClick`, инициирующим обратную отсылку.

Элементы управления HTML имеют определенные ограничения, например, свойства стилей должны устанавливаться с применением синтаксиса CSS (более сложного, чем установка прямого свойства), а события изменения не могут генерироваться до тех пор, пока страница не будет отправлена в ответ на другое действие.

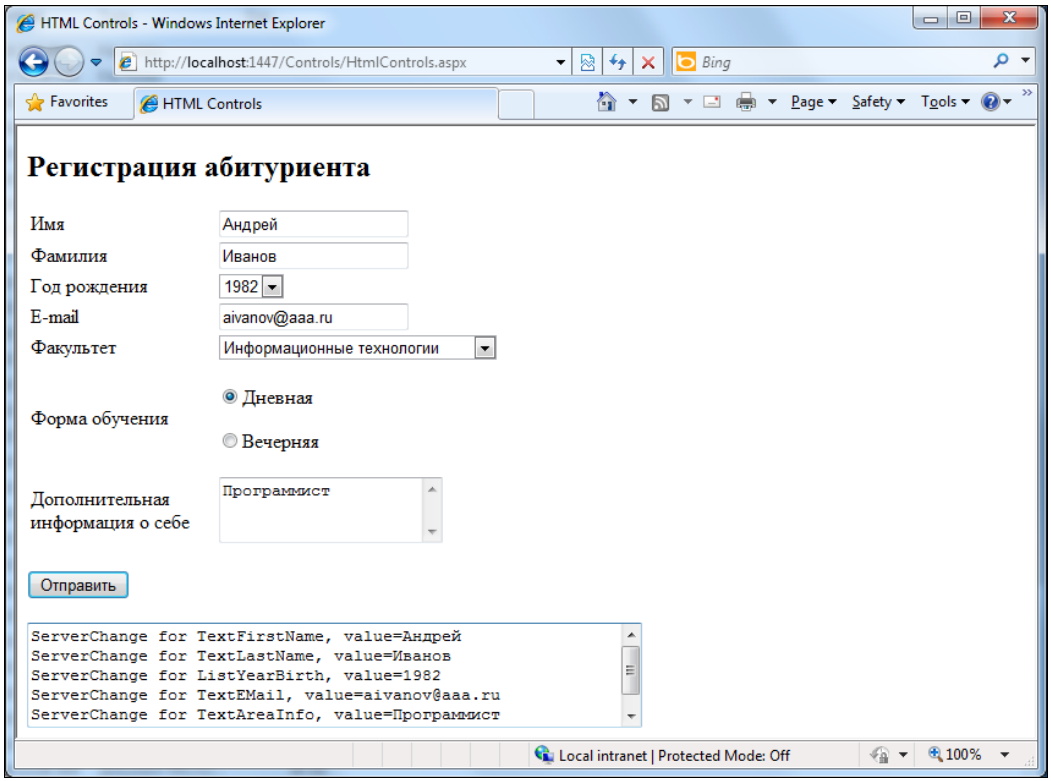


Рис. 11.10. Определение и вывод серверных событий

Элементы управления Web

Для решения проблем, связанных с элементами управления HTML, ASP.NET предоставляет высокоуровневую модель элементов управления Web. Все элементы управления Web определены в пространстве имен `System.Web.UI.WebControl` и являются производными от базового класса `WebControl`, реализующего более абстрактную и непротиворечивую модель, чем модель серверных элементов управления HTML. Элементы управления Web также поддерживают дополнительные свойства наподобие автоматической обратной отсылки. Но действительно примечательным является то, что многие усовершенствованные элементы управления не просто отображаются на отдельный HTML-дескриптор, но генерируют более сложный вывод, состоящий из нескольких HTML-дескрипторов и JavaScript-кода. В качестве примеров можно привести списки флажков, переключатели, календари, редактируемые сетки и т. д.

На рис. 11.11 показана часть иерархии наследования элементов управления Web.

Все элементы управления Web унаследованы от класса `WebControl`. В свою очередь класс `WebControl` наследуется от `Control`. Поэтому многие его свойства и методы, например `Controls()`, `Visible()`, `FindControl()`, похожи на свойства и методы серверных элементов управления HTML.

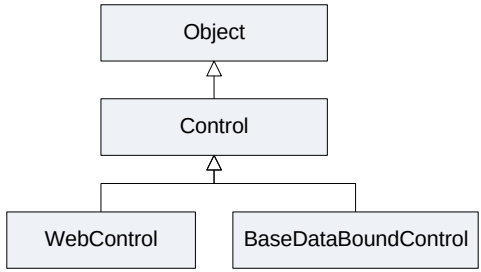


Рис. 11.11. Иерархия классов элементов управления Web

Однако в классе `WebControl` добавлены еще и свойства, которые заключают в себе атрибуты стиля CSS наподобие цвета изображения, фонового цвета, шрифта, высоты, ширины и прочего. Эти свойства намного упрощают настройку макета веб-элемента управления и уменьшают вероятность ошибок.

ASP.NET содержит элементы управления Web, дублирующие все серверные элементы управления HTML и предлагающие такую же функциональность. Так как элементы управления Web унаследованы от класса `WebControl`, они имеют и дополнительные свойства и события.

Элементы управления Web всегда объявляются на странице с использованием синтаксиса `asp:`, например, так можно объявить элемент управления `TextBox`:

```
<asp:TextBox runat="server" ID="TextBox1" Text="This is a test"
   ForeColor="red"  BackColor="lightyellow" Width="250px"
   Font-Name="Verdana"  Font-Bold="True" Font-Size="20" />
```

Все элементы управления Web должны объявляться внутри дескриптора серверной формы (и может существовать только одна серверная форма на страницу), даже если они не вызывают обратную отсылку.

В отличие от серверных элементов управления HTML, каждый элемент управления Web предоставляет метод `Focus()`. Фокус оказывает влияние только на элементы управления вводом. При загрузке страницы в браузер пользователь начинает работу с элемента управления, имеющего фокус.

Вместо программного вызова метода `Focus()` вы можете установить элемент управления, который должен всегда иметь фокус. Это делается с помощью свойства `DefaultFocus` следующим образом:

```
<form id="Form1" DefaultFocus="TextBox1" runat="server">
```

Иногда может потребоваться сделать кнопку `Submit` формы кнопкой по умолчанию, чтобы при нажатии пользователем клавиши <Enter> страница отправлялась на сервер. Для создания кнопки по умолчанию необходимо указать в свойстве `DefaultButton` формы идентификатор соответствующего элемента управления:

```
<form id="Form1" DefaultButton="cmdSubmit" runat="server">
```

Кнопка по умолчанию должна быть элементом управления, реализующим интерфейс `IButtonControl`, который реализуют классы `Button`, `LinkButton` и `ImageButton`.

Серверные события работают так же, как и серверные события элементов управления HTML, только элементы управления Web реализуют более богатый набор событий.

Элементы управления Web, в отличие от элементов управления HTML, поддерживают свойство `AutoPostBack`, которое использует `JavaScript` для захвата клиентского события и запуска обратной отсылки. Сервер при получении отправленной страницы сразу запускает соответствующее серверное событие.

В этой главе мы рассмотрим основные группы элементов управления:

- стандартные элементы управления Web;
- элементы управления для проверки вводимых пользователем данных;
- элементы управления для работы с источниками данных.

В следующих главах мы рассмотрим и другие группы элементов управления: элементы управления навигацией, расширения AJAX, элементы управления динамическими данными. Конечно, для досконального изучения всего множества веб-элементов управления в книге не хватит места, поэтому мы рассмотрим только основные элементы управления из этих групп.

Базовые элементы управления Web

Базовые элементы управления Web включают в себя такие элементы, как `DropDownList`, `ListBox`, `CheckBoxList` и `RadioButtonList`. Они позволяют выбирать один или несколько элементов в списке. Когда страница отправляется обратно, можно проверить, какие элементы были выбраны.

Элементы управления `RadioButtonList` и `CheckBoxList` создаются из нескольких кнопок выбора вариантов или флажков. Для включения элемента в `RadioButtonList` или `CheckBoxList` необходимо установить свойство `Listltem.Disabled` в `true`. Элемент `Listltem` появится на странице, но будет неактивным, и выбрать его будет нельзя.

Для примера создадим веб-страницу с серверными элементами Web, аналогичную созданной ранее странице с элементами HTML. Добавим в наш веб-сайт новую страницу с именем Registration_v1.aspx (далее будут еще версии этой страницы). Код этой веб-страницы приведен в листинге 11.5.

> ***Примечание***
>
> Полный код примера на прилагаемом к книге диске можно найти в каталоге Chapter11\Controls\Registration_v1.aspx.

Листинг 11.5. Страница Registration_v1.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Registration_v1.aspx.cs" Inherits="Registration_v1" %>

<!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>Web controls</title>
</head>
  <style type="text/css">
    .column0{width: 200px;}
    .column1{width: 350px;}
  </style>
<body>
<div><h2>Регистрация абитуриента</h2></div>
  <form id="form1" runat="server">
  <div>
    <table>
      <tr>
        <td class="column0">Имя</td>
        <td class="column1">
         <asp:TextBox ID="TextFirstName" runat="server"></asp:TextBox>
        </td>
      </tr>
      <tr>
        <td class="column0">Фамилия</td>
        <td class="column1"">
         <asp:TextBox ID="TextLastName" runat="server"></asp:TextBox>
         </td>
      </tr>
      <tr>
        <td class="column0">Год рождения </td>
        <td class="column1"">
          <asp:DropDownList ID="ListYearBirth" runat="server">
          </asp:DropDownList>
        </td>
      </tr>
      <tr>
        <td class="column0">E-mail</td>
        <td class="column1"">
          <asp:TextBox ID="TextEMail" runat="server"></asp:TextBox>
         </td>
      </tr>
      <tr>
        <td class="column0">Факультет</td>
        <td class="column1"">
          <asp:DropDownList ID="ListFaculties" runat="server">
          </asp:DropDownList>
         </td>
      </tr>
      <tr>
        <td class="column0">Форма обучения</td>
```

```
        <td class="column1"">
          <asp:RadioButtonList ID="RadioButtonList1" runat="server"
            onselectedindexchanged="RadioGroup_SelectedIndexChanged">
            <asp:ListItem Selected="True">Дневная</asp:ListItem>
            <asp:ListItem>Вечерняя</asp:ListItem>
          </asp:RadioButtonList>
          </td>
      </tr>
      <tr>
        <td class="column0">Дополнительная информация о себе</td>
        <td class="column1"">
         <asp:TextBox ID="TextInfo" runat="server" Height="49px"
           TextMode="MultiLine"></asp:TextBox>
        </td>
      </tr>
      </table>
   <br /></div>
   <div>
     <asp:Button ID="bSubmit" runat="server" onclick="Submit_Click"
Text="Отправить" CssClass="button" />
   </div>
   <br /><div>
     <asp:TextBox ID="TextOutput" runat="server" Height="85px"
     TextMode="MultiLine" Visible="False" Width="385px"></asp:TextBox>
    </div>
  </form>
</body>
</html>
```

Код программной части класса с обработчиками событий `Page_Load()`, `RadioGroup_SelectedIndexChanged()` для переключателей и `Submit_Click()` для кнопки отправки данных на сервер представлен в листинге 11.6.

Примечание

Полный код примера на прилагаемом к книге диске можно найти в каталоге Chapter11Controls\Registration_v1.aspx.cs.

Листинг 11.6. Код программной части веб-страницы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class Registration_v1 : System.Web.UI.Page
{
   bool isFullTime = true;
```

```
    protected void Page_Load(object sender, EventArgs e)
    {
       for (int i = DateTime.Today.Year; i >= 1900; i--)
       {
          ListYearBirth.Items.Add(i.ToString());
       }

       ListFaculties.Items.Add("Информационные технологии");
       ListFaculties.Items.Add("Электроника и телекоммуникации");
    }

    protected void RadioGroup_SelectedIndexChanged(
       object sender, EventArgs e)
    {
       isFullTime = RadioButtonList1.Items[0].Selected;
    }

    protected void Submit_Click(object sender, EventArgs e)
    {
       TextOutput.Text = String.Format("Абитуриент: {0} {1}, {2}г.р.\n" +
         "EMail: {3}\nФакультет: {4}\n" +
         "Форма обучения: {5}\nДоп. информация: {6}",
         TextFirstName.Text,
         TextLastName.Text,
         ListYearBirth.Text,
         TextEMail.Text,
         ListFaculties.Text,
         isFullTime ? "Дневная" : "Вечерняя",
         TextInfo.Text);
       TextOutput.Visible = true;
    }
}
```

Загрузите страницу в браузере, заполните все поля и нажмите кнопку `Submit`. После отправки данных на сервер в поле `TextOutput` должна отобразиться суммарная информация о введенных данных (рис. 11.12).

Элементы валидации данных

После ввода пользователем данных необходимо проверить их на правильность. В принципе проверка достоверности пользовательского ввода должна выполняться на стороне клиента, чтобы пользователь был немедленно информирован о проблемах с вводом перед отправкой формы на сервер. Это сохраняет серверные ресурсы и ускоряет обратную связь с пользователем. Однако, независимо от того, выполняется ли клиентская проверка достоверности, следует также проверять достоверность введенных данных и на стороне сервера.

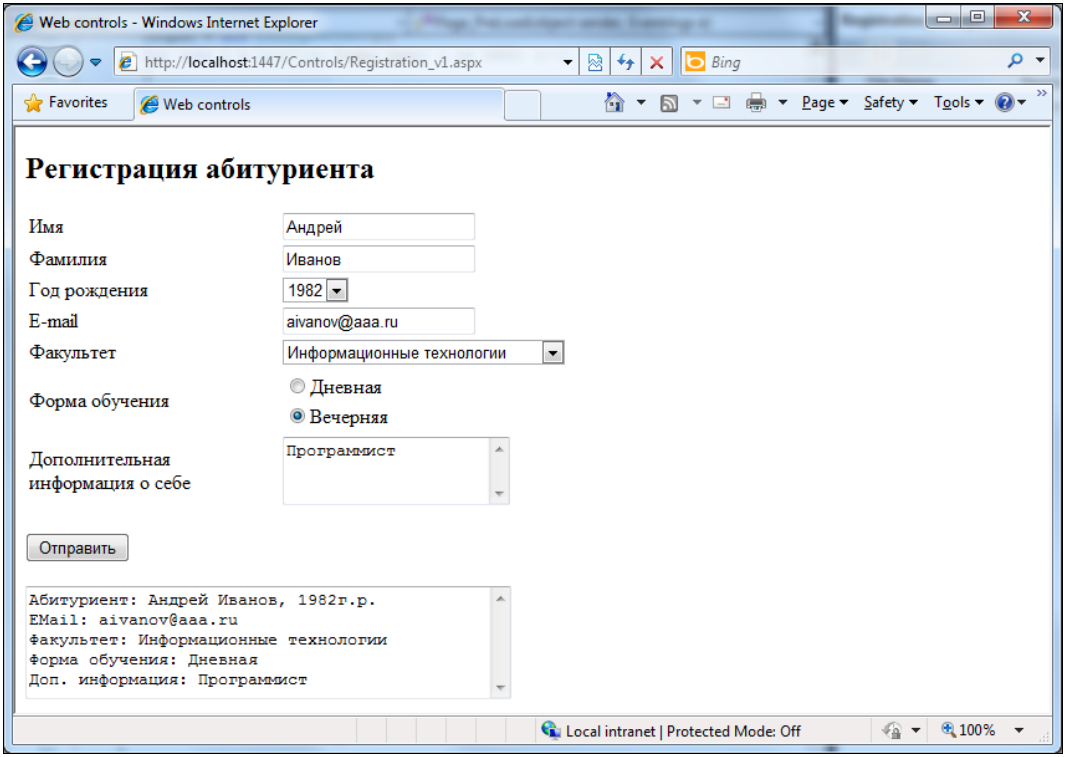


Рис. 11.12. Веб-страница со стандартными элементами управления

Всего существует шесть элементов валидации данных:

- `RequiredFieldValidator`
- `CompareValidator`
- `RangeValidator`
- `RegularExpressionValidator`
- `CustomValidator`
- `ValidationSumary`

Элементы управления проверкой достоверности можно использовать для автоматической проверки страницы при ее отправке пользователем. При автоматической проверке достоверности пользователь получает обычную страницу и начинает заполнять элементы управления вводом. По окончании работы пользователь щелкает на кнопке отправки страницы. Каждая кнопка имеет булево свойство `CausesValidation`, которое можно установить в `true` или `false`.

- Если значение `CausesValidation` равно `false`, ASP.NET игнорирует элементы управления проверкой достоверности, страница будет отправлена, и ваш код обработки событий будет нормально работать.
- Если значение `CausesValidation` равно `true`, которое установлено по умолчанию, ASP.NET автоматически проверит страницу при щелчке пользователя на кнопке,

выполняя проверку каждого элемента управления на странице. Если какой-либо элемент управления не проходит проверку, ASP.NET возвращает страницу с сообщениями об ошибке.

Элементы валидации данных, кроме `ValidationSumary`, имеют свойство `ControlToValidate`, которое определяет идентификатор проверяемого элемента, и свойство `ErrorMessage` для вывода сообщения об ошибке. Рассмотрим элементы валидации данных более подробно.

RequiredFieldValidator

Элемент управления `RequiredFieldValidator` проверяет текстовое поле на наличие в нем текста. Также можно указать значение по умолчанию с использованием свойства `InitialValue`. В этом случае данные считаются неправильными, если содержимое элемента управления совпадает со значением `InitialValue`, т. е. пользователь не внес никаких изменений в содержимое элемента.

Далее показан пример использования элемента `RequiredFieldValidator`:

```
<asp:RequiredFieldValidator ID="ValFirstName" runat="server"
   ControlToValidate="TextFirstName" Display="Dynamic"
   ErrorMessage="Введите имя!">
</asp:RequiredFieldValidator>
```

Текст сообщения об ошибке, содержащийся в `ErrorMessage`, появляется, когда пользователь пытается отправить форму на сервер, оставив пустое поле ввода.

RangeValidator

Элемент управления `RangeValidator` проверяет попадание введенного значения в определенный диапазон. Он имеет три специальных свойства:

- `Type`
- `MinimumValue`
- `MaximumValue`

Свойство `Type` устанавливает допустимый тип вводимых пользователем данных, например `Integer`, `Double`, `String`, `Date` и т. д. Свойства `MinimumValue` и `MaximumValue` элемента управления `RangeValidator` определяют диапазон допустимых значений, вводимых пользователем.

Например, так может осуществляться проверка нахождения введенного года рождения в диапазоне между 1950 и 1995 гг.

```
<asp:RangeValidator ID="ValYearOfBirth" runat="server"
   ControlToValidate="ListYearBirth" Display="Dynamic"
   Type="Integer"
   MinimumValue="1950"
   MaximumValue="1995"
   ErrorMessage="Ваш возраст не подходит для поступления в колледж!">
</asp:RangeValidator>
```

CompareValidator

Элемент управления `CompareValidator` сравнивает значение в одном элементе управления с фиксированным значением или со значением в другом элементе управления. Например, это позволяет проверить, содержат ли два текстовых поля одни и те же данные, или то, что значение одного текстового поля не превышает максимальное значение, заданное в другом текстовом поле.

В отличие от элемента управления `RangeValidator`, элемент управления `CompareValidator` кроме свойства `Type`, определяющего тип сравниваемых данных, также предлагает свойства `ValueToCompare` и `ControlToCompare`, позволяющие сравнивать значение элемента управления вводом, соответственно, с константой или значением другого элемента управления ввода. При проверке данных используется только одно из этих свойств.

Свойство `Operator` позволяет указать желаемый тип сравнения. Доступные значения: `Equal`, `NotEqual`, `GreaterThan`, `GreaterThanEqual`, `LessThan`, `LessThanEqual` и `DataTypeCheck`.

Установка свойства `Operator` в значение `DataTypeCheck` вынуждает элемент управления проверкой достоверности убедиться, что ввод имеет необходимый тип данных, определенный в свойстве `Type`, без выполнения какого-либо дополнительного сравнения.

В следующем примере показана проверка, является ли введенный год рождения равным или меньшим 1995 г.

```
<asp:CompareValidator ID="ValYearOfBirth" runat="server"
   ControlToValidate="ListYearBirth" Display="Dynamic"
   Type="Integer"
   ValueToCompare="1995"
   Operator="LessThanEqual"
   ErrorMessage="Ваш возраст не подходит для поступления в колледж!">
</asp:CompareValidator>
```

RegularExpressionValidator

Элемент управления `RegularExpressionValidator` позволяет проверить достоверность вводимого пользователем текста, сравнивая его с шаблоном, определенным в регулярном выражении. Регулярное выражение определяется в свойстве `ValidationExpression`.

Например, следующий элемент управления проверяет, является ли введенная в текстовом окне информация допустимым адресом электронной почты:

```
<asp:RegularExpressionValidator ID="ValEMail" runat="server"
   ControlToValidate="TextEMail" Display="Dynamic"
   ValidationExpression=".*@.{2,}\..{2,}"
   ErrorMessage="Неправильный E-Mail!" >
</asp:RegularExpressionValidator>
```

CustomValidator

Элемент управления `CustomValidator` позволяет выполнять пользовательские процедуры проверки достоверности на стороне клиента и сервера. Если проверка достоверности завершается неудачей, свойство `Page.IsValid` устанавливается в `false`, что характерно для любого другого элемента управления проверкой достоверности.

Пример веб-страницы с валидацией введенных данных

Теперь используем элементы для валидации введенных данных на нашей веб-странице. Для элементов ввода текста `TextFirstName`, `TextLastName` используем элемент валидации `RequiredFieldValidator`, для списка `ListYearBirth` — элемент `RangeValidator`, а для текстового поля `TextEMail` — элемент `RegularExpressionValidator`.

Код веб-страницы с валидацией данных представлен в листинге 11.7.

Примечание

Полный код примера веб-страницы с валидацией данных на прилагаемом к книге диске находится в каталоге Chapter11\Controls\Registration_v3.aspx.

Листинг 11.7. Веб-страница с валидацией введенных данных

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Registration_v2.aspx.cs" Inherits="Registration_v3" %>

<!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 id="Head1" runat="server">
  <title>Web controls</title>
</head>
  <style type="text/css">
   .column0{width: 200px;}
   .column1{width: 350px;}
  </style>
<body>
<div class="header"><h2>Регистрация абитуриента</h2></div>
  <form id="form1" runat="server">
    <div>
      <table>
        <tr>
          <td class="column0">Имя</td>
          <td class="column1">
            <asp:TextBox ID="TextFirstName" runat="server"></asp:TextBox>
```

```
      <asp:RequiredFieldValidator ID="ValFirstName" runat="server"
        ControlToValidate="TextFirstName"
        ErrorMessage="Введите имя!"
        Display="Dynamic">
      </asp:RequiredFieldValidator>
    </td>
  </tr>
  <tr>
    <td class="column0">Фамилия</td>
    <td class="column1"">
      <asp:TextBox ID="TextLastName" runat="server"></asp:TextBox>
      <asp:RequiredFieldValidator ID="ValLastName" runat="server"
        ControlToValidate="TextLastName"
        ErrorMessage="Введите фамилию!"
        Display="Dynamic">
      </asp:RequiredFieldValidator>
    </td>
  </tr>
  <tr>
    <td class="column0">Год рождения</td>
    <td class="column1"">
      <asp:DropDownList ID="ListYearBirth" runat="server">
      </asp:DropDownList>
      <asp:CompareValidator ID="ValYearOfBirth" runat="server"
        ControlToValidate="ListYearBirth" Type="Integer"
        ValueToCompare="1995"
        Operator="LessThanEqual"
        ErrorMessage=
          "Ваш возраст не подходит для поступления в колледж!"
        Display="Dynamic">
      </asp:CompareValidator>
    </td>
  </tr>
  <tr>
    <td class="column0">E-mail</td>
    <td class="column1"">
      <asp:TextBox ID="TextEMail" runat="server"></asp:TextBox>
      <asp:RegularExpressionValidator ID="ValEMail" runat="server"
        ControlToValidate="TextEMail"
        ValidationExpression=".*@.{2,}\..{2,}"
        ErrorMessage="Неправильный E-Mail!" Display="Dynamic">
      </asp:RegularExpressionValidator>
    </td>
  </tr>
  <tr>
    <td class="column0">Факультет</td>
    <td class="column1"">
```

```
            <asp:DropDownList ID="ListFaculties" runat="server"
              CausesValidation="True"
              DataTextField="FacultyName"
              DataValueField="FacultyId">
            </asp:DropDownList>
        </td>
      </tr>
      <tr>
        <td class="column0">Форма обучения</td>
        <td class="column1"">
          <asp:RadioButtonList ID="RadioButtonList1" runat="server">
          <asp:ListItem Selected="True">Дневная</asp:ListItem>
          <asp:ListItem>Вечерняя</asp:ListItem>
          </asp:RadioButtonList>
        </td>
      </tr>
      <tr>
        <td class="column0">Дополнительная информация о себе</td>
        <td class="column1"">
          <asp:TextBox ID="TextInfo" runat="server" Height="49px"
            TextMode="MultiLine">
          </asp:TextBox>
        </td>
      </tr>
    </table>
  </div>
  <br />
  <div>
    <asp:Button ID="bSubmit" runat="server" onclick="bSubmit_Click"
      Text="Отправить" CssClass="button" />
  </div>
  </form>
</body>
</html>
```

На рис. 11.13 показана веб-страница, отображающая сообщения об ошибке около каждого проверяемого поля ввода.

ValidationSummary

В наборе элементов валидации данных еще присутсвует `ValidationSummary`. Этот элемент не выполняет проверку достоверности, а предназначен для отображения итоговой информации по всем ошибкам на странице. В итоговой информации выводится значение `ErrorMessage` каждого элемента управления проверкой достоверности, для которого эта проверка завершилась неудачей. Итоговая информация может отображаться на странице только в том случае, если свойство `ShowSummary`

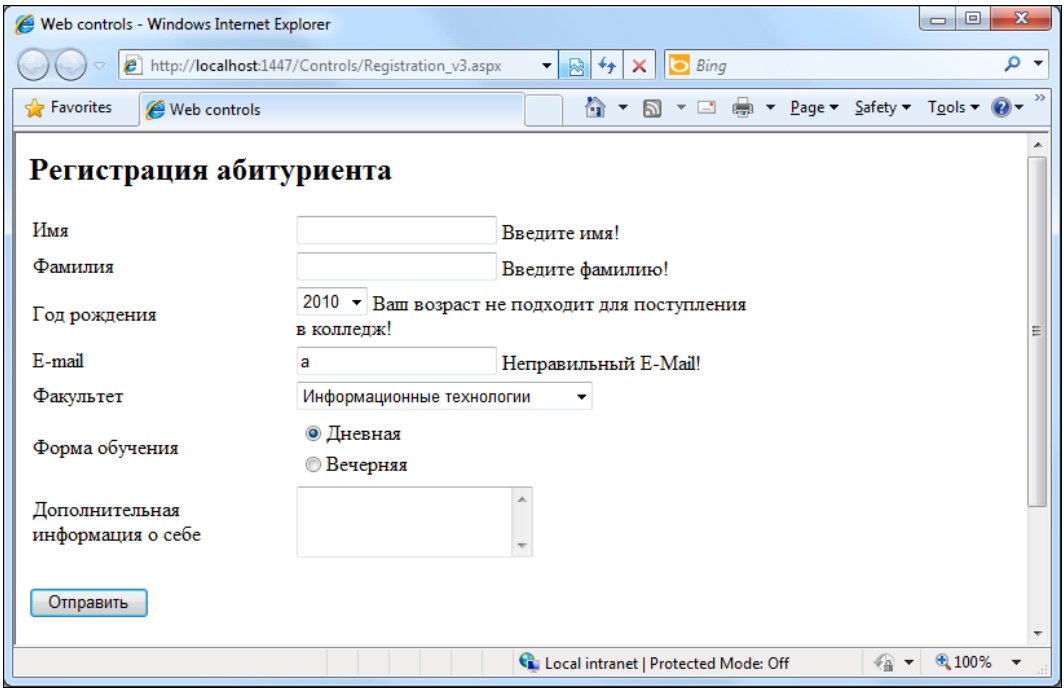


Рис. 11.13. Веб-страница с валидацией введенных данных

установлено в `true`. С помощью свойства `HeaderText` можно задать заголовок для итоговой информации.

Создание элемента управления `ValidationSummary` на веб-странице является достаточно простым делом. Добавьте на нашу веб-страницу код, показанный в листинге 11.8 перед закрывающим тегом `</form>`.

Листинг 11.8. Применение элемента управления `ValidationSummary`

```
...
  <div>
    <asp:Button ID="bSubmit" runat="server" onclick="bSubmit_Click"
      Text="Отправить" CssClass="button" />
  </div>
  <br />
  <asp:ValidationSummary runat="server" ID="ValSummary"
    ShowSummary="true" ShowMessageBox="true"
    HeaderText="Пожалуйста, устраните следующие ошибки:" />
</form>
...
```

Теперь при отправке на сервер веб-страницы с неправильно введенными данными внизу страницы будет отображаться суммарная информация об ошибках (рис. 11.14).

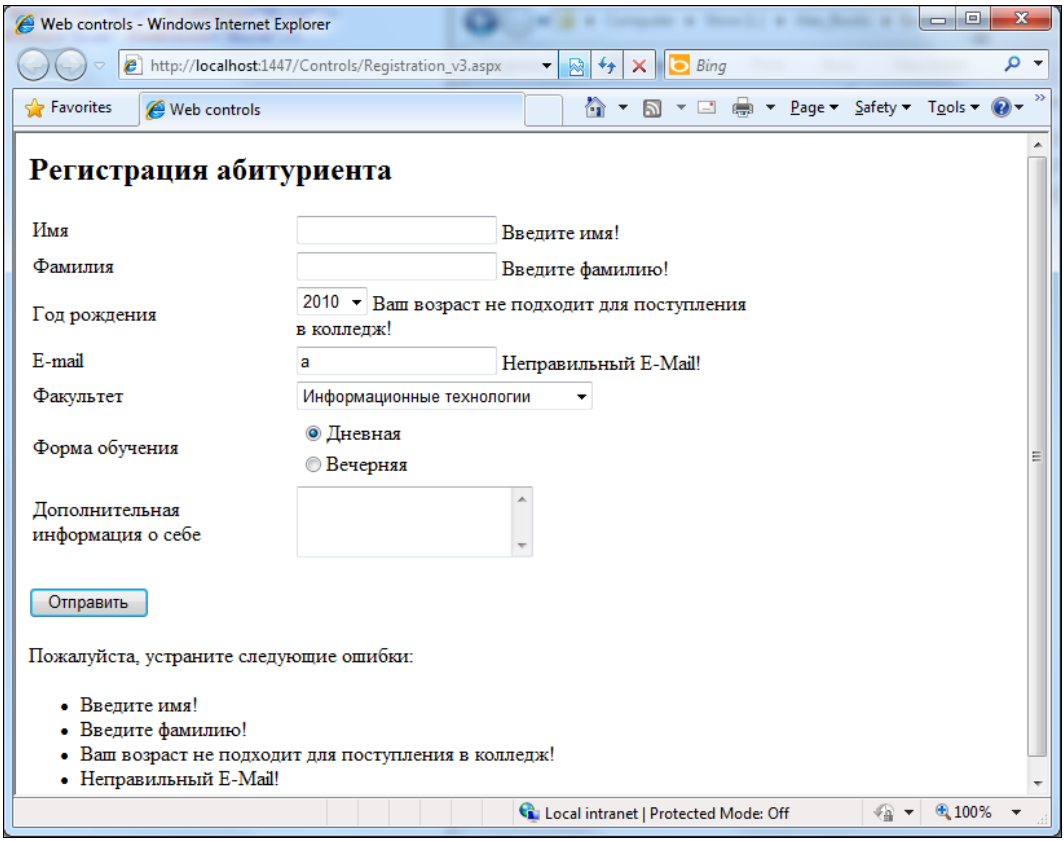


Рис. 11.14. Веб-страница с элементом `ValidationSummary`

Вывод сообщений об ошибках можно производить и в клиентском окне сообщений JavaScript, если свойство `ShowMessageBox` элемента `ValidationSummary` установить в `true`, как показано в листинге 11.9.

Листинг 11.9. Код для вывода ошибок в клиентское окно сообщений

```
<asp:ValidationSummary runat="server"  ID="ValSummary"
   ShowMessageBox="true"
   HeaderText="Пожалуйста, устраните следующие ошибки:" />
```

Запустите страницу в браузере. При отправке на сервер веб-страницы с неправильно введенными данными внизу страницы будет отображаться вывод сообщений об ошибках в клиентском окне JavaScript (рис. 11.15).

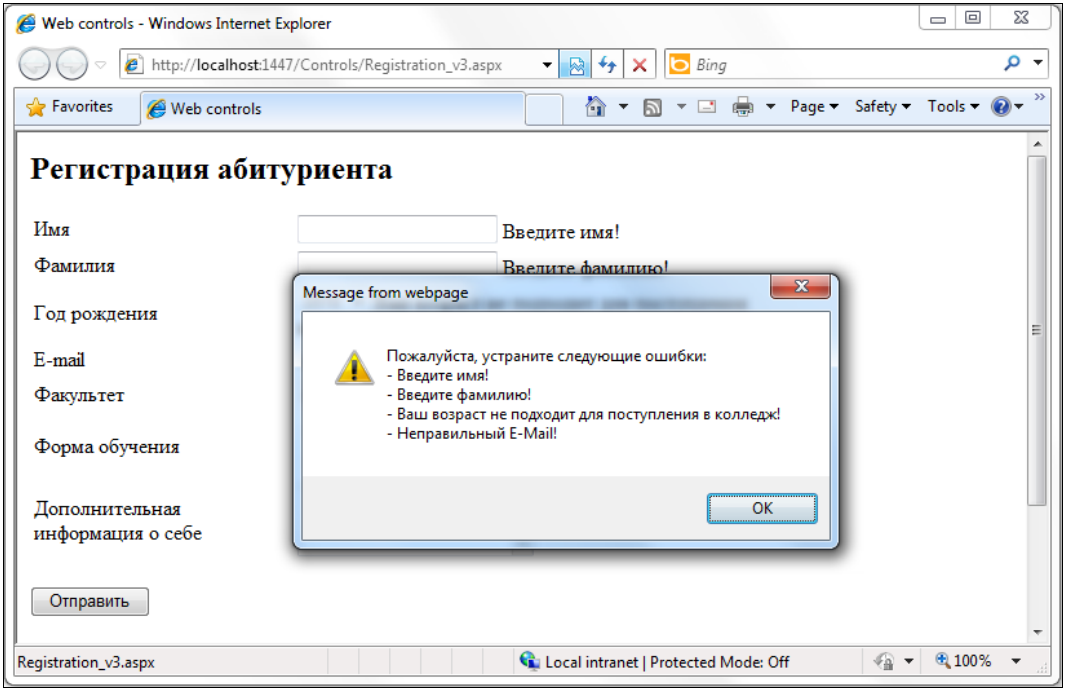


Рис. 11.15. Вывод сообщений об ошибках в клиентское окно сообщений

Привязка данных к элементам управления Web

Несмотря на то, что общие концепции сохраняются, привязка данных к веб-элементам управления несколько отличается от привязки данных в Windows Forms. Источники данных для веб-форм реализованы элементами управления источником данных в пространстве имен `System.Web.UI.WebControls`, и для веб-приложений не существует концепции окна **Data Sources**. Поэтому вместо того, чтобы начинать с источника данных, вам придется начинать с элемента управления данными, а затем прикреплять этот элемент управления к источнику данных.

Использование элемента управления *GridView*

Сначала в открытом веб-проекте перетащите мышью элемент управления `GridView` из панели **Toolbox** на пустую веб-страницу. Откройте смарт-тег и выберите в списке **Choose Data Source** пункт **New data source** (рис. 11.16).

Откроется уже знакомое вам по *главе 8* окно мастера **Data Source Configuration Wizard**. Однако сейчас мы будем выбирать тип источника данных **Entity** (рис. 11.17).

При помощи меню смарт-тега `GridView` вы можете выбрать опцию **Auto Format** (рис. 11.18) для применения нескольких видов украшений (рис. 11.19).

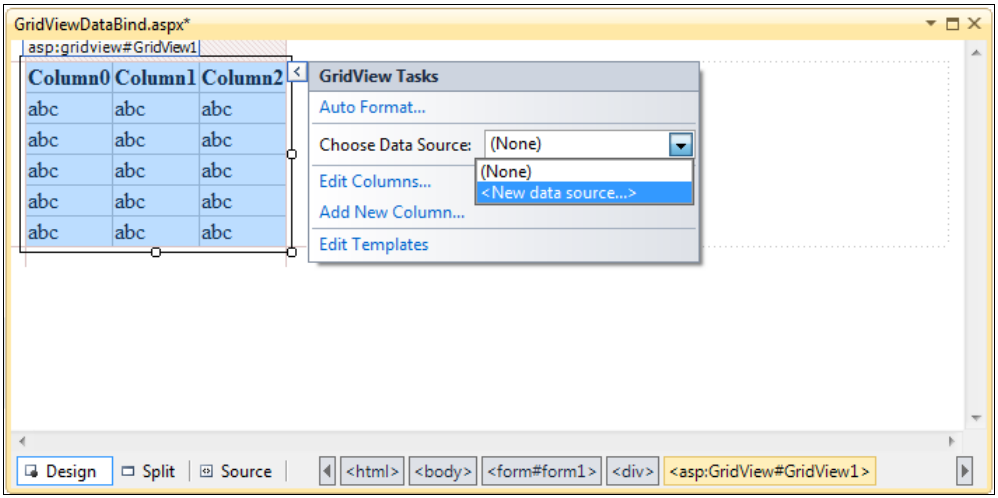


Рис. 11.16. Выбор источника данных для `GridView`

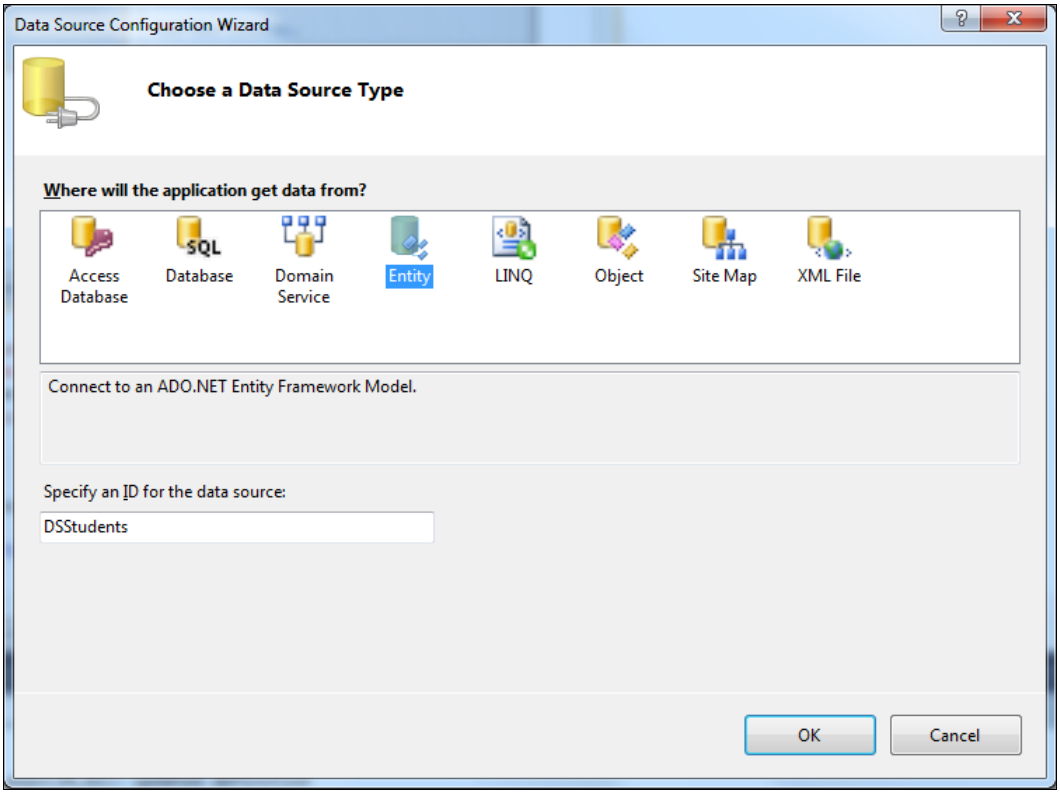


Рис. 11.17. Первая страница мастера **Data Source Configuration Wizard**

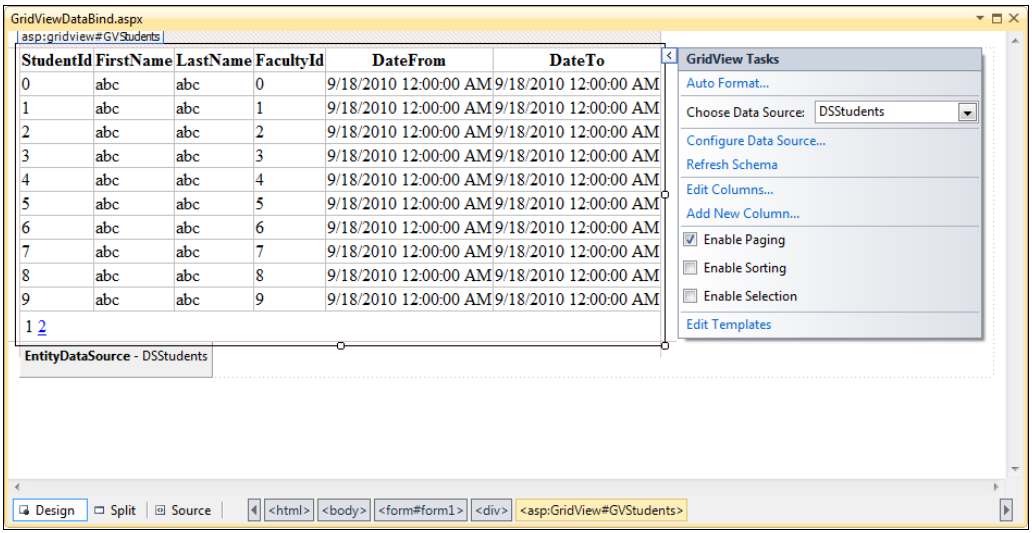


Рис. 11.18. Выбор источника данных для элемента управления `GridView`

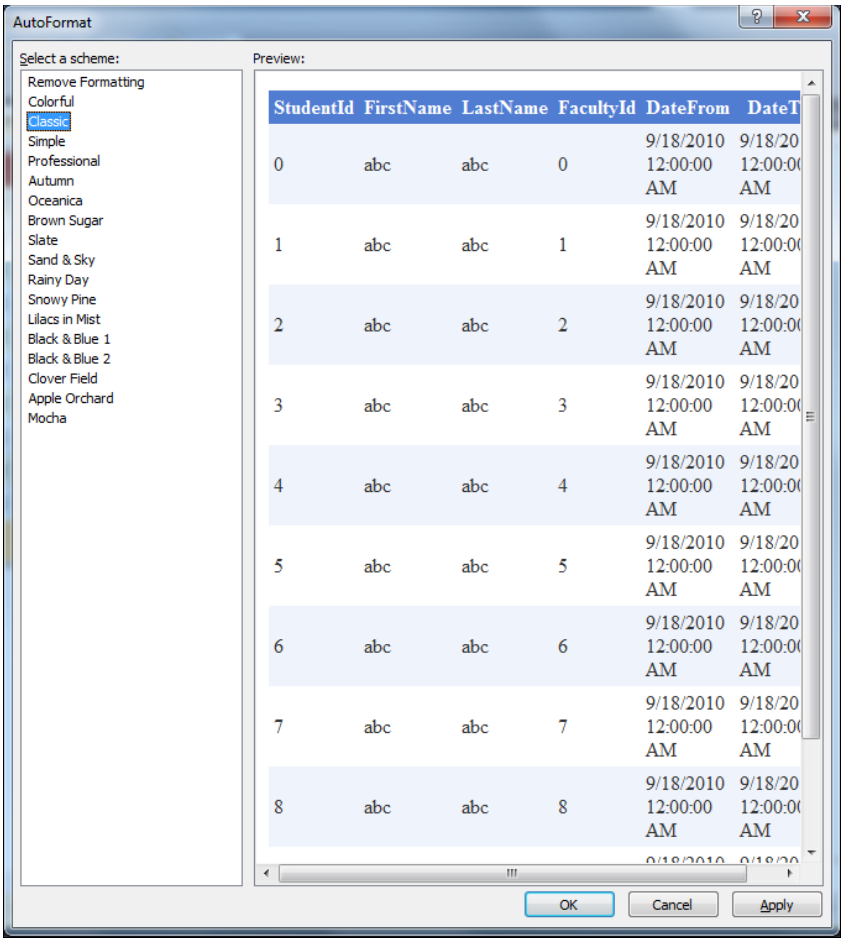
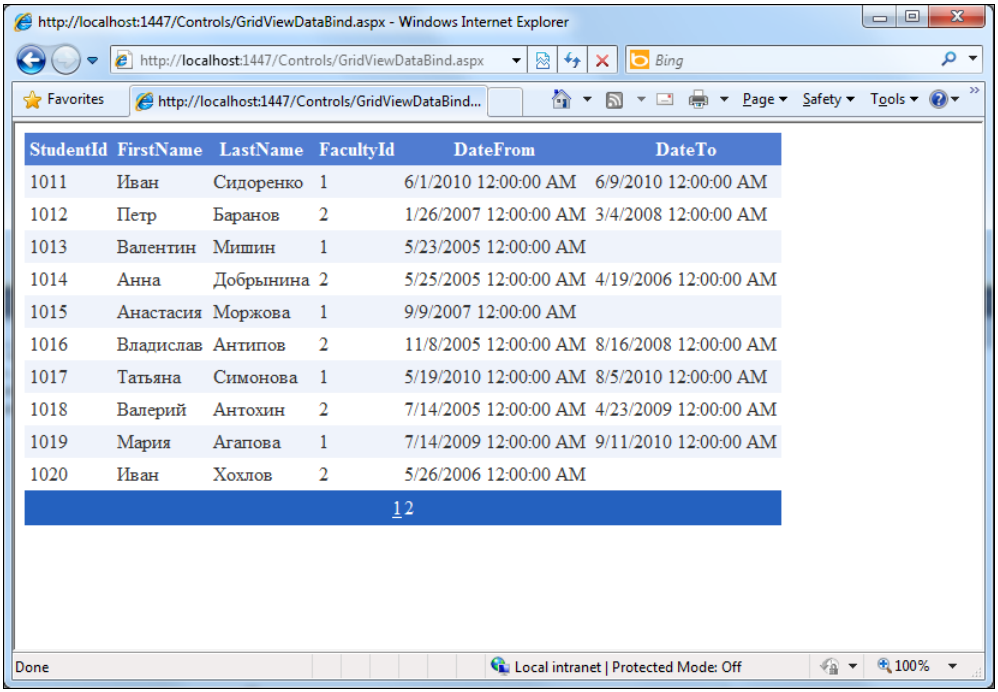


Рис. 11.19. Варианты форматирования для элемента управления `GridView`

На рис. 11.20 показана веб-страница с данными о преподавателях колледжа.

Вы также можете изменить вид страницы посредством использования таблицы стилей, но этот вопрос мы будем рассматривать в следующей главе.



StudentId	FirstName	LastName	FacultyId	DateFrom	DateTo
1011	Иван	Сидоренко	1	6/1/2010 12:00:00 AM	6/9/2010 12:00:00 AM
1012	Петр	Баранов	2	1/26/2007 12:00:00 AM	3/4/2008 12:00:00 AM
1013	Валентин	Мишин	1	5/23/2005 12:00:00 AM	
1014	Анна	Добрынина	2	5/25/2005 12:00:00 AM	4/19/2006 12:00:00 AM
1015	Анастасия	Моржова	1	9/9/2007 12:00:00 AM	
1016	Владислав	Антипов	2	11/8/2005 12:00:00 AM	8/16/2008 12:00:00 AM
1017	Татьяна	Симонова	1	5/19/2010 12:00:00 AM	8/5/2010 12:00:00 AM
1018	Валерий	Антохин	2	7/14/2005 12:00:00 AM	4/23/2009 12:00:00 AM
1019	Мария	Агапова	1	7/14/2009 12:00:00 AM	9/11/2010 12:00:00 AM
1020	Иван	Хохлов	2	5/26/2006 12:00:00 AM	

1 2

Рис. 11.20. Информация о студентах в `GridView` с постраничным выводом данных

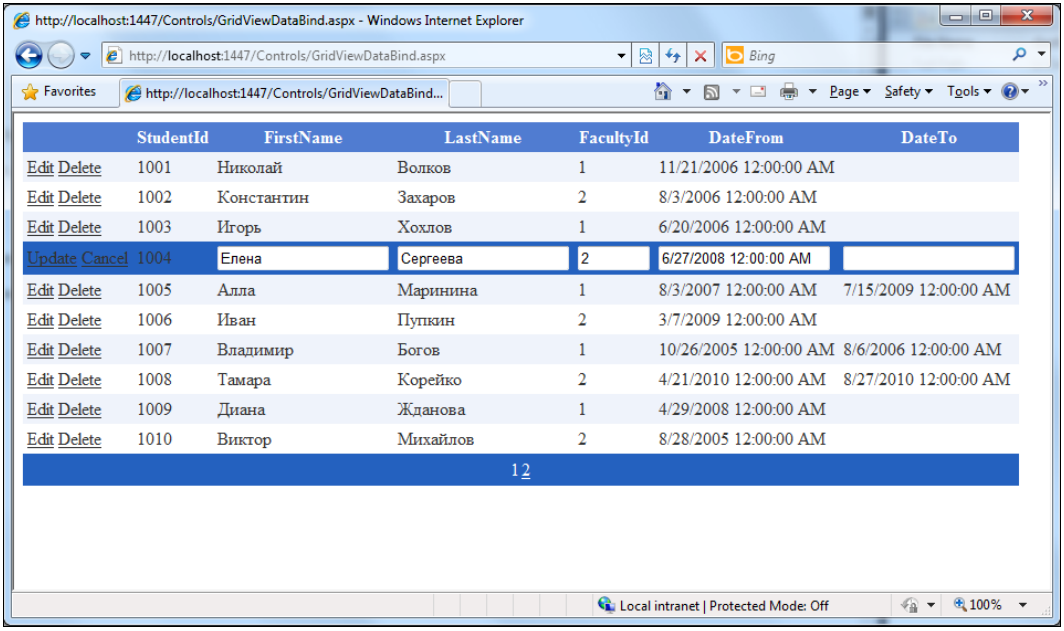
Обновление данных в *GridView*

Если требуется редактировать данные внутри сетки, в элементе `DataGrid` существует набор свойств `GridView`:

- `AutoGenerateEditButton`
- `AutoGenerateDeleteButton`

Когда вы устанавливаете эти свойства в значение `true`, элемент управления `GridView` автоматически добавит ссылки **Edit** и **Delete**, которые вы можете использовать для модификации данных в записях (рис. 11.21).

После изменения данных в одном или нескольких столбцах вы можете нажать ссылку **Update** для отправки данных обратно в базу данных. Однако для того чтобы обновление работало, вам нужно явно указать элементу управления источником данных `DSStudents` тот запрос, который должен использоваться для обработки обновлений. Это делается при помощи свойства `DSStudents.UpdateQuery`. Указав в этом свойстве параметризированный запрос `Update`, вы полностью проинформировали источник данных о том, как нужно обрабатывать обновления.



	StudentId	FirstName	LastName	FacultyId	DateFrom	DateTo
Edit Delete	1001	Николай	Волков	1	11/21/2006 12:00:00 AM	
Edit Delete	1002	Константин	Захаров	2	8/3/2006 12:00:00 AM	
Edit Delete	1003	Игорь	Хохлов	1	6/20/2006 12:00:00 AM	
Update Cancel	1004	Елена	Сергеева	2	6/27/2008 12:00:00 AM	
Edit Delete	1005	Алла	Маринина	1	8/3/2007 12:00:00 AM	7/15/2009 12:00:00 AM
Edit Delete	1006	Иван	Пупкин	2	3/7/2009 12:00:00 AM	
Edit Delete	1007	Владимир	Богов	1	10/26/2005 12:00:00 AM	8/6/2006 12:00:00 AM
Edit Delete	1008	Тамара	Корейко	2	4/21/2010 12:00:00 AM	8/27/2010 12:00:00 AM
Edit Delete	1009	Диана	Жданова	1	4/29/2008 12:00:00 AM	
Edit Delete	1010	Виктор	Михайлов	2	8/28/2005 12:00:00 AM	

1 2

Рис. 11.21. Редактирование записи в `GridView`

Привязка данных к спискам

Привязка данных к спискам аналогична привязке к `GridView`, однако есть некоторые особенности. Разместим на форме элемент типа `EntityDataSource` с привязкой к сущности `Faculty` и назовем его `DSFaculty`, как показано на рис. 11.22.

Для привязки данных к элементу `DropDownList` используются свойство `DataTextField`, которому присваивается значение `FacultyName`, и свойство `DataValueField`, которому присваивается значение `FacultyId`, как показано на рис. 11.23.

Код, сгенерированный при привязке данных к элементу управления `DropDownList`, представлен в листинге 11.10.

Листинг 11.10. Привязка данных к элементу управления `DropDownList`

```
<asp:DropDownList ID="ListFaculties" runat="server"
   CausesValidation="True"
   DataTextField="FacultyName"
   DataValueField="FacultyId"
   DataSourceID="DSFaculty">
</asp:DropDownList>
```

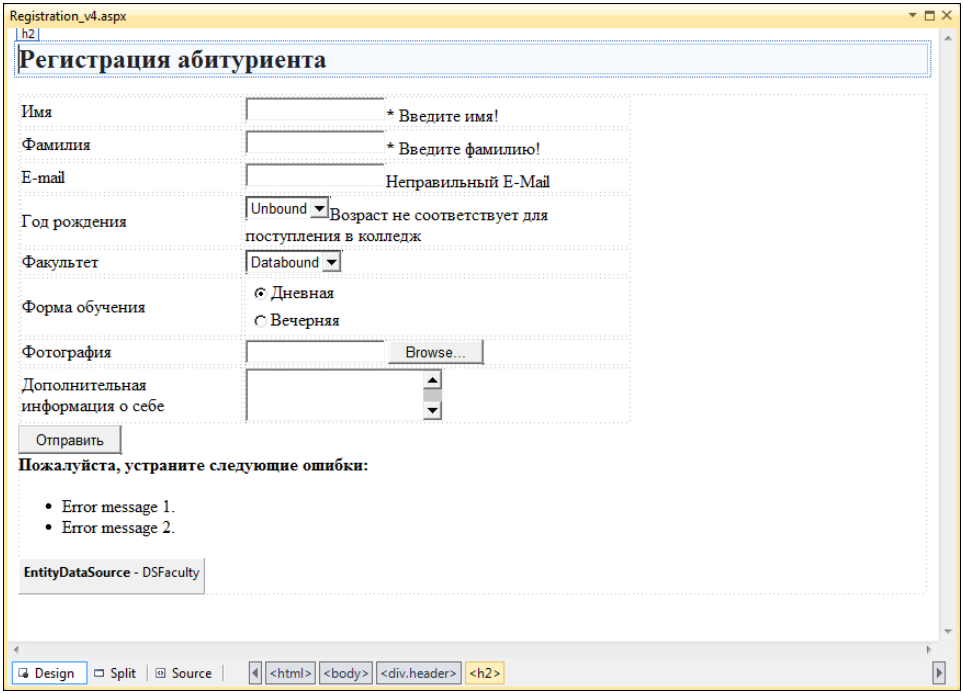


Рис. 11.22. Разметка для веб-страницы регистрации

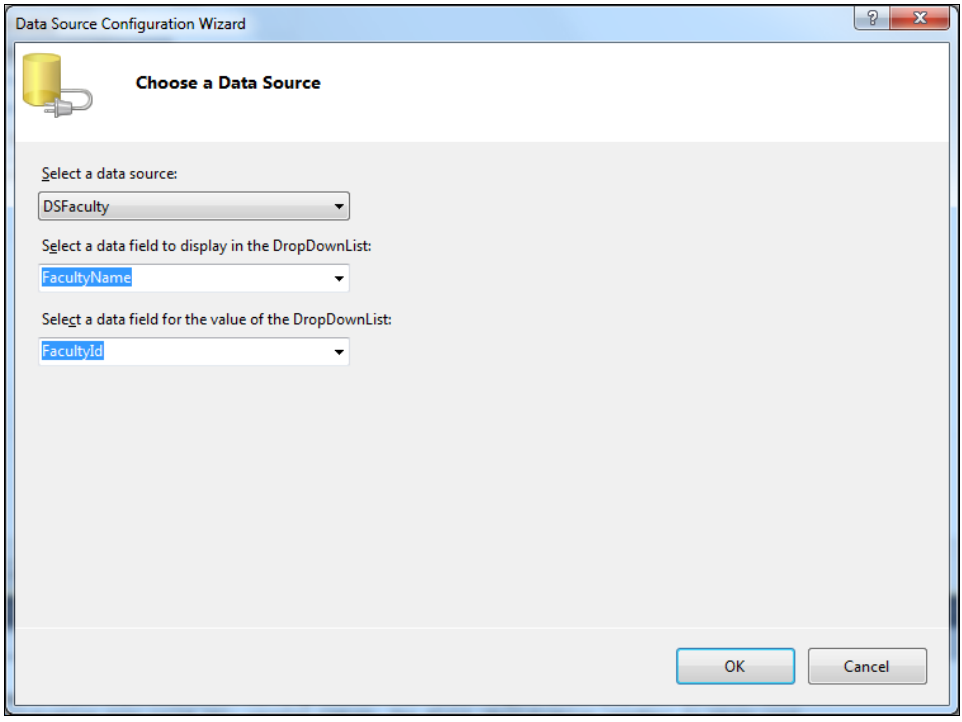


Рис. 11.23. Привязка полей списка к источнику данных

Динамическая привязка данных

Привязку данных можно осуществлять динамически, в программной части веб-страницы. Давайте создадим веб-страницу, на которой пользователь посредством выбора названия факультета в элементе управления `DropDownList` может выводить в элемент `GridView` список курсов, относящихся к выбранному факультету.

Дизайн новой веб-страницы представлен на рис. 11.24.

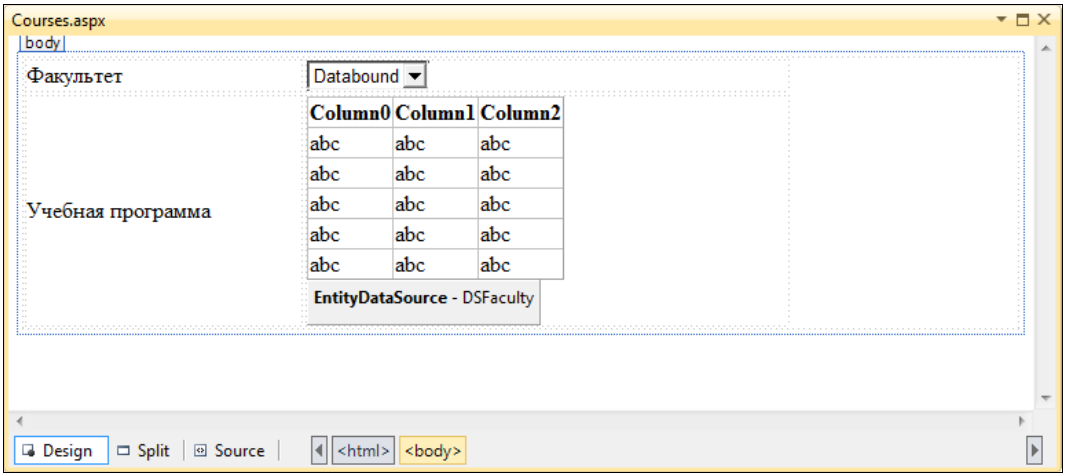


Рис. 11.24. Дизайн веб-страницы Courses.aspx

Код разметки страницы с динамической привязкой данных к элементу управления `GridView` представлен в листинге 11.11.

> ***Примечание***
>
> Полный код примера данной веб-страницы на прилагаемом к книге диске находится в каталоге Chapter11\Controls\GridViewDataBind.aspx.

Листинг 11.11. Файл Courses.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Courses.aspx.cs"
   Inherits="Courses" %>

<!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 id="Head1" runat="server">
    <title></title>
    <style type="text/css">
        .style1{width: 200px;}
    </style>
</head>
```

```
<body>
    <form id="form1" runat="server">
    <table>
       <tr>
          <td class="style1">
             Факультет</td>
          <td>
             <asp:DropDownList ID="ListFaculties" runat="server"
                AutoPostBack="True"
                DataSourceID="DSFaculties"
                DataTextField="FacultyName"
                DataValueField="FacultyId"
                onselectedindexchanged="ListFaculties_SelectedIndexChanged">
             </asp:DropDownList>
          </td>
       </tr>
       <tr>
          <td class="style1">
             Учебная программа</td>
          <td>
             <asp:GridView ID="DataGridCourse" runat="server"
                   CellPadding="4"
                   ForeColor="#333333" GridLines="None" >
                <AlternatingRowStyle BackColor="White" />
                <EditRowStyle BackColor="#2461BF" />
                <FooterStyle BackColor="#507CD1" Font-Bold="True"
                   ForeColor="White" />
                <HeaderStyle BackColor="#507CD1" Font-Bold="True"
                   ForeColor="White" />
                <PagerStyle BackColor="#2461BF" ForeColor="White"
                   HorizontalAlign="Center" />
                <RowStyle BackColor="#EFF3FB" />
                <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True"
                   ForeColor="#333333" />
                <SortedAscendingCellStyle BackColor="#F5F7FB" />
                <SortedAscendingHeaderStyle BackColor="#6D95E1" />
                <SortedDescendingCellStyle BackColor="#E9EBEF" />
                <SortedDescendingHeaderStyle BackColor="#4870BE" />
             </asp:GridView>
          </td>
       </tr>
    </table>
    <asp:EntityDataSource ID="DSFaculties" runat="server"
       ConnectionString="name=CollegeEntities"
          DefaultContainerName="CollegeEntities"
       EnableFlattening="False" EntitySetName="Faculties">
    </asp:EntityDataSource>
```

```
    </form>
</body>
</html>
```

В коде программной части класса `Courses` в обработчике событий `ListFaculties_SelectedIndexChanged()` реализован запрос LINQ к сущности `Faculty`, результат выполнения которого отображается в элементе управления `DataGrid`.

Код программной части класса `Courses` представлен в листинге 11.12.

Листинг 11.12. Файл Courses.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using College.Data;

public partial class Courses : System.Web.UI.Page
{
   private CollegeEntities collegeEnt;

    protected void Page_Load(object sender, EventArgs e)
    {
       collegeEnt = new CollegeEntities();
    }

    protected void ListFaculties_SelectedIndexChanged(
          object sender, EventArgs e)
    {
       int id = Convert.ToInt32(ListFaculties.SelectedValue);

       var courseInfo = from c in collegeEnt.Courses
                        where c.FacultyId == id
                        select new
                        {
                           c.CourseId,
                           c.CourseName,
                           c.Trainer.FirstName,
                           c.Trainer.LastName
                        };

       DataGridCourse.DataSource = courseInfo;
       DataGridCourse.DataBind();
    }
```

```
    protected void Page_PreRenderComplete(object sender, EventArgs e)
    {
        ListFaculties_SelectedIndexChanged(this.Page, new EventArgs());
    }
}
```

Запустите страницу в браузере и протестируйте ее, выбирая различные факультеты из списка. Внешний вид этой страницы представлен на рис. 11.25.

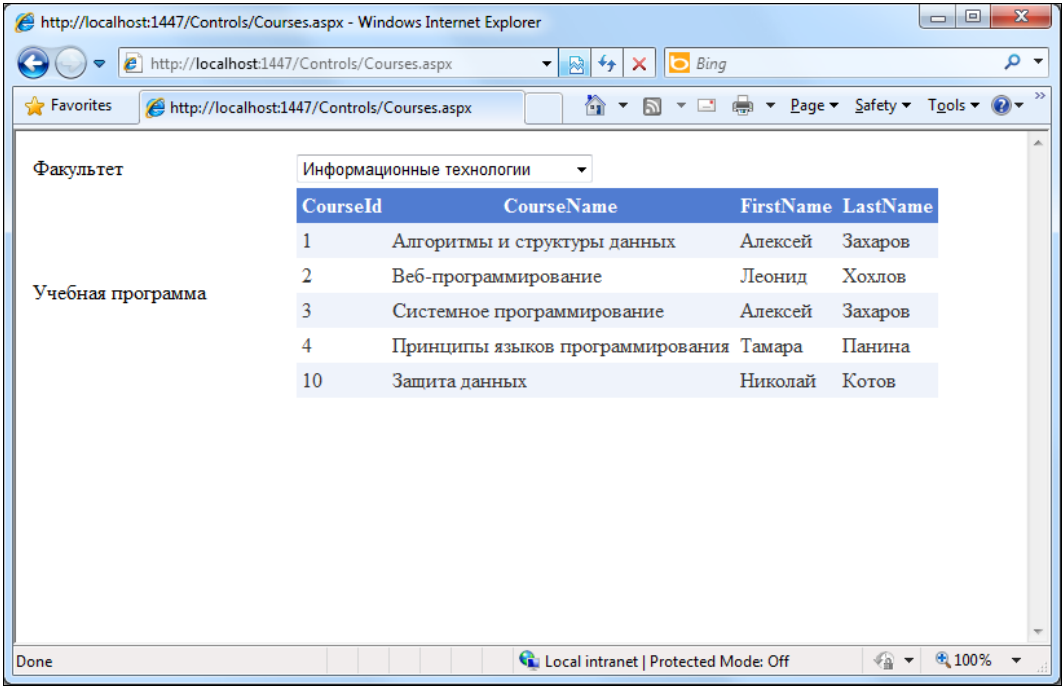
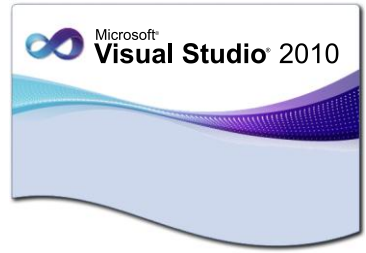


Рис. 11.25. Страница с динамической привязкой данных

Резюме

В этой главе вы познакомились с основами формирования веб-страниц и созданием серверных элементов управления ASP.NET. В следующей главе будут рассматриваться вопросы форматирования страниц и объединения страниц для построения полноценных веб-приложений.



ГЛАВА 12

Стили и темы

Чтобы создать профессиональный и визуально привлекательный веб-сайт, необходимо разработать для него собственный стиль. Среда Visual Studio 2010 предоставляет широкую поддержку для создания стилей, управления ими, а также применения стилей как для отдельных веб-страниц, так и для всего сайта в целом.

Структура страниц и элементов управления ASP.NET также предоставляет возможности управления отображением и поведением веб-узла с помощью тем и обложек. Можно определить темы и затем применить их на уровне страницы или элемента управления.

Стили

Стили и таблицы стилей определяют общий дизайн веб-сайта. При необходимости, достаточно изменить стиль в одном месте — и изменения произойдут везде, где используется этот стиль.

Стили можно определять различными способами:

- создать стили для отдельного элемента управления;
- определить стили на уровне веб-страницы;
- вынести стили в отдельный файл — каскадную таблицу стилей (Cascading Style Sheet, CSS).

Далее мы рассмотрим каждый способ определения стилей более подробно.

Стили элементов

Стиль определяет вид данного элемента. Можно создать встроенные стили, которые используют атрибут стиля данного элемента формы для указания того, как этот элемент выглядит. Например, так можно определить встроенный стиль, который настраивает для элемента управления `TextBox` шрифт, размер шрифта и цвет:

```
<asp:TextBox ID="text1" runat="server"
  style="font-family: Arial; font-size: large; color: Blue">
</asp:Label>
```

Встроенные стили нельзя повторно использовать, и приходится определять стиль для каждого элемента на странице. Лучше отделить определение стиля от элемента управления. Для этого необходимо поместить определение стиля в стилевой класс, как будет показано в следующем разделе.

Стили страниц

Класс стиля может быть определен на уровне веб-страницы. Если вы определяете стилевой класс в самой странице, то он применим только к этой странице. Стили, определенные на уровне страницы, объявляются в разделе `<head>` кода страницы. Например, таким образом:

```
<head runat="server">
  <title>Edit Customer</title>
  <style type="text/css">
  .titleText
  {
    font-family: Arial;
    font-size: larger-
    color: Blue;
  }
  </style>
</head>
```

Для применения этого стиля к элементу формы вы настраиваете его свойство `Class` или `CssClass` так, как показано здесь:

```
<asp:Label ID="Labell" runat="server" Text="Edit Customer"
CssClass="titleTextMx/asp:Label>
```

Каскадные таблицы стилей

Таблица стилей представляет собой файл, содержащий набор стилей. Внутри этого файла можно добавлять стили для элементов и определять собственные классы стилей.

Для создания внешнего файла стилей на уровне веб-сайта откройте окно **Add New Item** и выберите шаблон **Style Sheet** (рис. 12.1).

Создав файл стилей, вы можете писать код стилей непосредственно в редакторе кода или использовать различные окна и панели, предоставляемые средой разработки Visual Studio 2010 для создания и управления стилями.

После определения таблицы стилей вы связываете страницу с таблицей стилей. Вы можете сделать это, перетащив таблицу стилей на визуальный конструктор форм или добавив следующий код в раздел `<head>` страницы:

```
<head runat="server">
<link href="StyleSheetMaster.ess" rel="stylesheet" type="text/css" />
<title>Edit Customer</title>
</head>
```

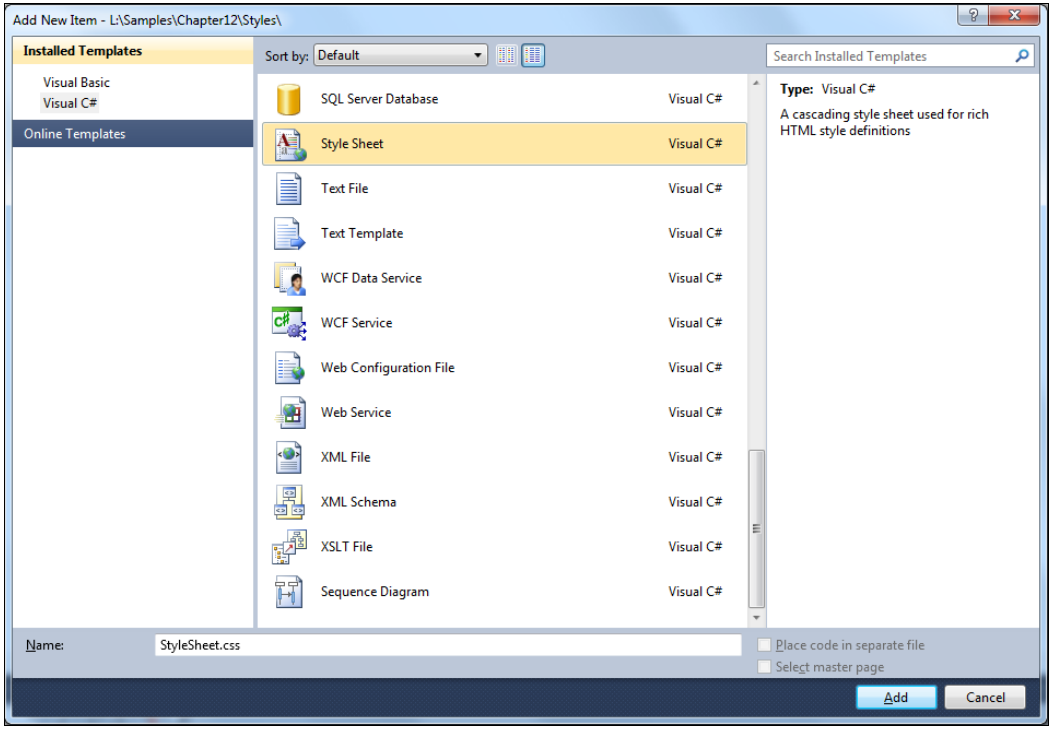


Рис. 12.1. Добавление файла стилей в проект

Визуальный конструктор Visual Studio при конструировании будет показывать применимые стили. Та есть во время конструирования вы можете видеть, как будет выглядеть ваше приложение.

Создание и управление стилями

Для создания и управления стилями среда разработки Visual Studio предоставляет панель инструментов **Style Sheet** и меню **Styles**. Панель инструментов **Style Sheet** имеет несколько кнопок. Первая кнопка слева запускает диалоговое окно **Add Style Rule**. Вторая кнопка — **Build Style** — открывает диалоговое окно **Modify Style**. Это диалоговое окно генерирует код для стиля.

После создания файла стилей вы можете открыть таблицу стилей в редакторе кода Visual Studio. Панель инструментов **Style Sheet** имеет несколько кнопок. Первая слева запускает диалоговое окно **Add Style Rule**. Вторая — **Build Style** — открывает диалоговое окно **Modify Style**. Этот инструмент генерирует код для вашего стиля.

Диалоговое окно **Modify Style**, показанное на рис. 12.2, имеет много различных опций для форматирования стилей. Внешний вид стиля отображается в поле **Preview**, а описание стиля — в поле **Description** в нижней части диалогового окна.

Правило стиля (**Style Rule**) определяет стиль для элемента или класса. Для создания нового правила стиля можно использовать диалоговое окно **Add Style Rule** (Добавить правило стиля).

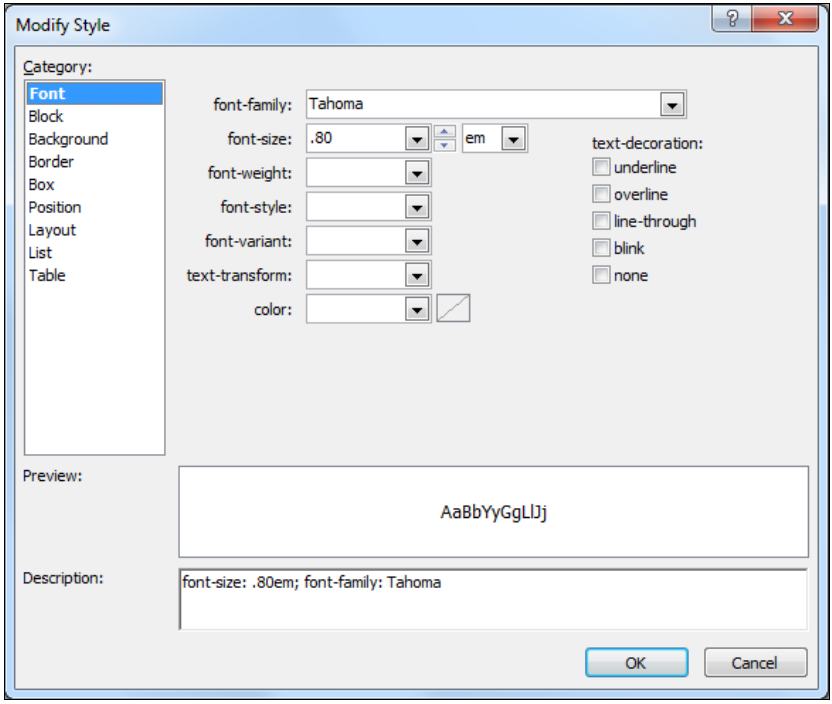


Рис. 12.2. Диалоговое окно **Modify Style**

Для вызова окна **Add Style Rule** необходимо щелкнуть правой кнопкой мыши на панели редактора таблиц стилей и в контекстном меню выбрать пункт **Add Style Rule** (или выбрать пункт **Add Style Rule** в разделе **Style** главного меню). В окне **Add Style Rule** имеется список различных HTML-элементов, для которых можно определять стили. Это окно также позволяет определять иерархию стилей. Диалоговое окно **Add Style Rule** показано на рис. 12.3.

После нажатия кнопки **OK** в окне стиль сохраняется в таблице стилей. Затем, при необходимости, этот стиль можно модифицировать.

Для отображения и управления стилями на веб-странице предназначена панель **Manage Styles** (Управление стилями). Панель **Manage Styles** открывается из меню **View | Manage Styles**, если открыт дизайнер веб-страниц (рис. 12.4).

Панель **Manage Styles** показывает все стили для текущей веб-страницы, а также те стили, которые связаны с данной веб-страницей через каскадную таблицу стилей. На панели **Manage Styles** можно с помощью мыши перетаскивать элементы стиля, которые определены на текущей странице, в таблицу стилей и обратно.

Панель имеет две кнопки в левом верхнем углу: **New Style** и **Attach Style Sheet**. Кнопка **New Style** открывает диалоговое окно **New Style** (рис. 12.5), похожее на диалоговое окно **Modify Style**.

Кнопка **Attach Style Sheet** позволяет выбрать существующую таблицу стилей и прикрепить ее к данной веб-странице. Кнопка с выпадающим списком **Options** позволяет группировать стили в окне по элементам, типам и порядку.

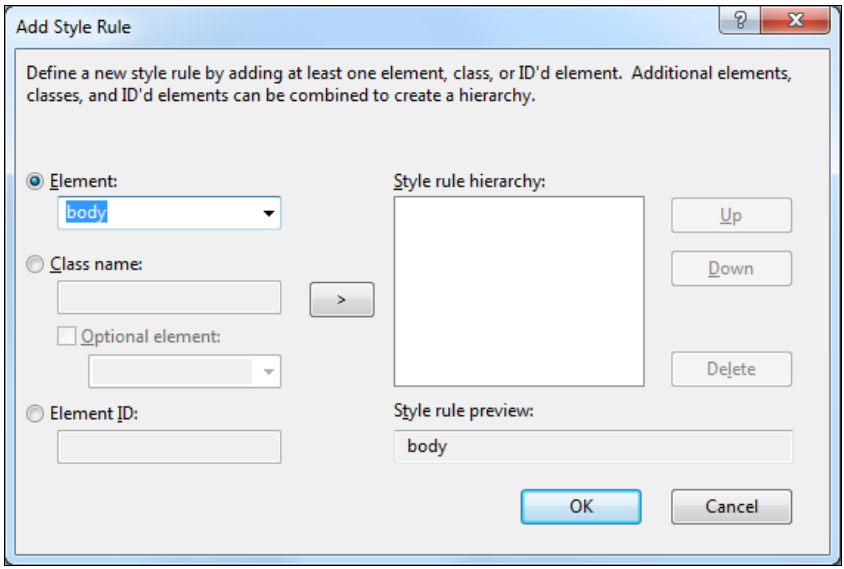


Рис. 12.3. Диалоговое окно **Add Style Rule**

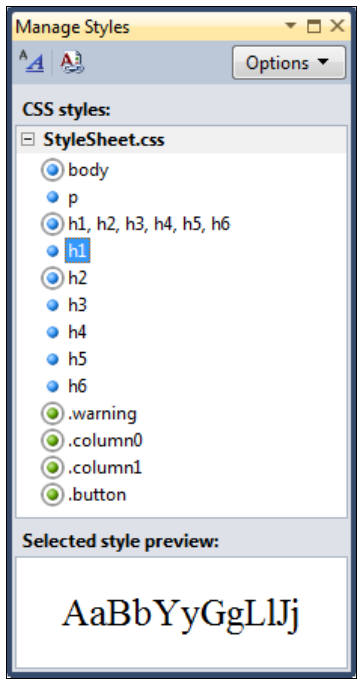


Рис. 12.4. Панель **Manage Styles**

Для применения стилей также используется панель **Apply Styles**, которую можно открыть из меню **View | Apply Styles**. Панель **Apply Styles** ведет себя аналогично **Style Manager**, но может отображать визуальное представление стиля в самом его названии, как показано на рис. 12.6.

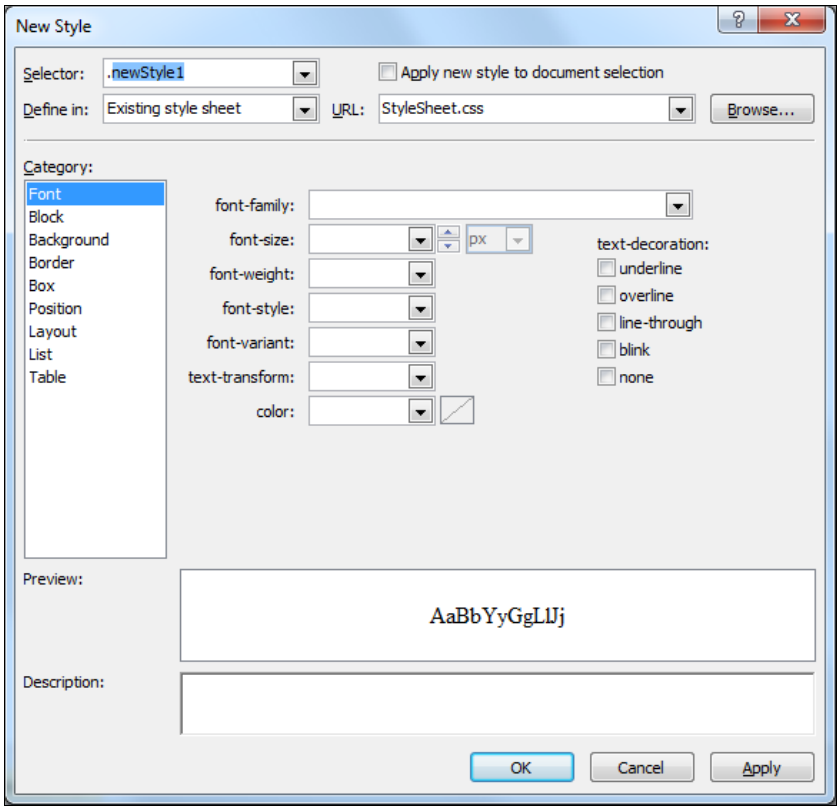


Рис. 12.5. Диалоговое окно **New Style**

Также можно настраивать стили элемента управления внутри окна свойств **Property**. Свойство **CssClass** в окне **Property** предоставляет выпадающий список стилей, доступных для данного элемента управления.

Редактировать стили можно различными способами. Например, в контекстном меню редактора кода есть пункт **Build Style** для того, чтобы открыть диалоговое окно **Modify Style**. Кроме того, для редактирования стилей в Visual Studio 2010 имеется панель **CSS Properties**, которую можно открыть, выбрав в главном меню пункт **View | CSS Properties**. Панель **CSS Properties** отображает все свойства данного стиля в редакторе свойств (рис. 12.7).

Теперь, после изучения инструментов для работы со стилями, предоставляемых средой Visual Studio, создадим файл стилей для нашего веб-сайта. Готовый файл StyleSheet.css приведен в листинге 12.1. Вы можете также поэкспериментировать с приведенными ранее в этом разделе окнами управления стилей для редактирования и добавления своих стилей в файл StyleSheet.css.

Примечание

Пример веб-сайта с подключенными стилями находится на прилагаемом к книге диске в каталоге Chapter12\Styles.

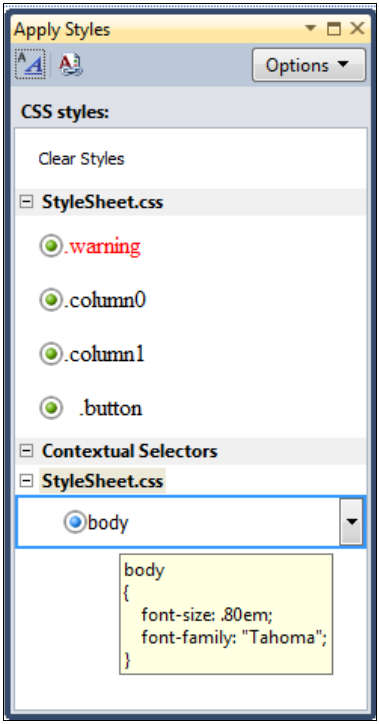


Рис. 12.6. Панель **Apply Styles**

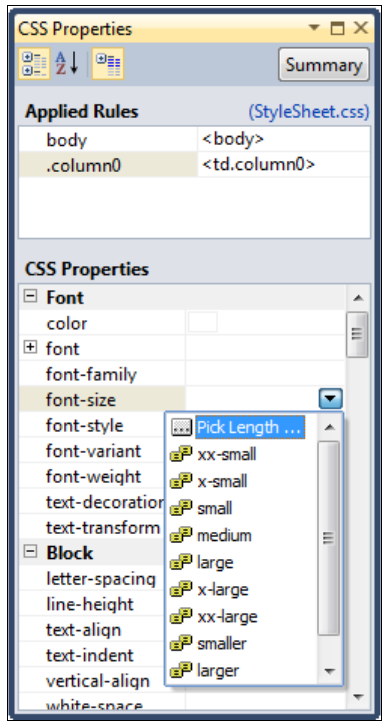


Рис. 12.7. Панель **CSS Properties**

Листинг 12.1. Файл StyleSheet.css

```
body
{
    font-size: .80em;
    font-family: "Tahoma";
}
p
{
    margin-bottom: 10px;
    line-height: 1.6em;
}
h1, h2, h3, h4, h5, h6
{
    font-variant: small-caps;
    text-transform: none;
    font-weight: 200;
    margin-bottom: 0px;
}
h1
{
    font-size: 1.6em;
```

```
    padding-bottom: 0px;
    margin-bottom: 0px;
}

h2
{
    font-size: 1.5em;
    font-weight: 600;
}

h3
{
    font-size: 1.2em;
}

h4
{
    font-size: 1.1em;
}

h5
{
    font-size: 1em;
}
h6
{
   font-size: 0.9em;
}
.warning
{
   color:Red;
}
.column0
{
   width:143px;
}
.column1
{
   width:auto;
}
.button
{
    margin: 10px;
}
```

Для добавления внешнего файла стилей на веб-страницу используется тег `<link>`. Например, чтобы подключить файл стилей для нашей страницы регистрации, надо добавить в раздел `<head>` код, показанный в листинге 12.2.

Листинг 12.2. Добавление внешнего файла стилей на веб-страницу

```
<head runat="server">
    <title>Registrations</title>
    <link href="~/Styles/StyleSheet.css" rel="stylesheet"
        type="text/css" />
</head>
```

После подключения файла стиля наша страница регистрации, созданная в предыдущей главе, приобретет более приятный вид (рис. 12.8).

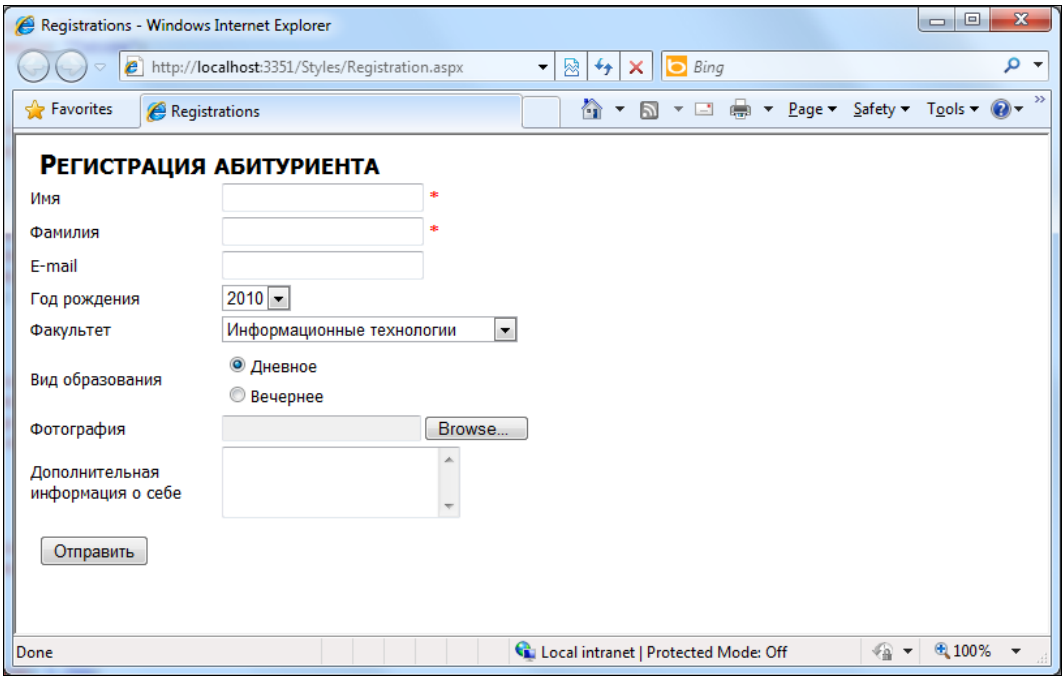


Рис. 12.8. Внешний вид приложения со стилями

Темы

Темы позволяют создавать набор атрибутов стилей, которые применяют к элементам управления веб-страниц один или несколько специфических внешних обликов для элементов управления, составляющих ваше приложение. Вы можете переключаться между разными темами, меняя внешний вид сайта.

Темы используют таблицы стилей. Каждая тема может иметь связанную с ней таблицу стилей. Темы также могут использовать графические файлы. Таблицы стилей этого делать не могут.

Создание темы

Темы создаются внутри каталога AppTheme и применяются на стороне сервера. Каждая тема имеет свой подкаталог, и название этого подкаталога является именем темы.

Для создания темы нужно создать каталог AppTheme при помощи контекстного меню, выбрав пункт **Add | ASP.NET Folder | Theme**. После создания папки AppTheme надо добавить подкаталог для темы. Вам нужно назвать этот каталог именем вашей темы. Затем вы добавляете сам файл темы. Эти файлы обычно содержат таблицу стилей, изображения и ресурсы.

В листинге 12.3 приведен пример простого файла темы, который окрашивает фон и текст определенных типов элементов управления в заданный цвет (в данном случае цвет фона будет синий, цвет текста — белый).

Примечание

Пример проекта веб-сайта из этого раздела находится на прилагаемом к книге диске в каталоге Chapter12\Themes.

Листинг 12.3. Файл темы Custom.skin

```
<asp:Button runat="server"
    BackColor="Blue" Font-Bold="true" ForeColor="White"/>
<asp:FileUpload runat="server" BackColor="Blue" ForeColor="White"/>
<asp:GridView runat="server" BackColor="Blue" ForeColor="White"/>
<asp:DropDownList runat="server" BackColor="Blue" ForeColor="White"/>
<asp:TextBox  runat="server" BackColor="Blue" ForeColor="White"/>
```

Подключение темы

Подключить тему можно как для всего сайта, так и для отдельной веб-страницы. Возможно также применить одну тему к отдельному элементу управления на веб-странице.

Тему на уровне страницы можно применить с помощью директивы `@ Page` внутри кода разметки страницы. При этом возможно несколько вариантов применения темы. Вы можете решить, что ваша тема должна всегда подавлять все настройки элементов управления. То есть если разработчик явным образом устанавливает значение элемента управления, и это значение переопределяется темой, то тема берет верх. Такого рода способ подключения темы с именем Theme1 может выглядеть следующим образом:

```
<%@ Page Theme="Theme1 %>
```

Такой способ подключения темы, например для нашей веб-страницы, определяется так, как показано в листинге 12.4.

Листинг 12.4. Подключение темы для страницы Registration.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
  CodeFile="Registration.aspx.cs" Inherits="Registration"
  Theme="Custom" %>
```

Внесите изменения в файл Registration.aspx, чтобы подключить тему к нашей странице регистрации абитуриента. Загрузите модифицированную страницу в браузер, и вы увидите, что текстовые поля и списки на странице поменяли цвет фона на синий, а цвет текста в текстовых полях и кнопке стал белым (рис. 12.9).

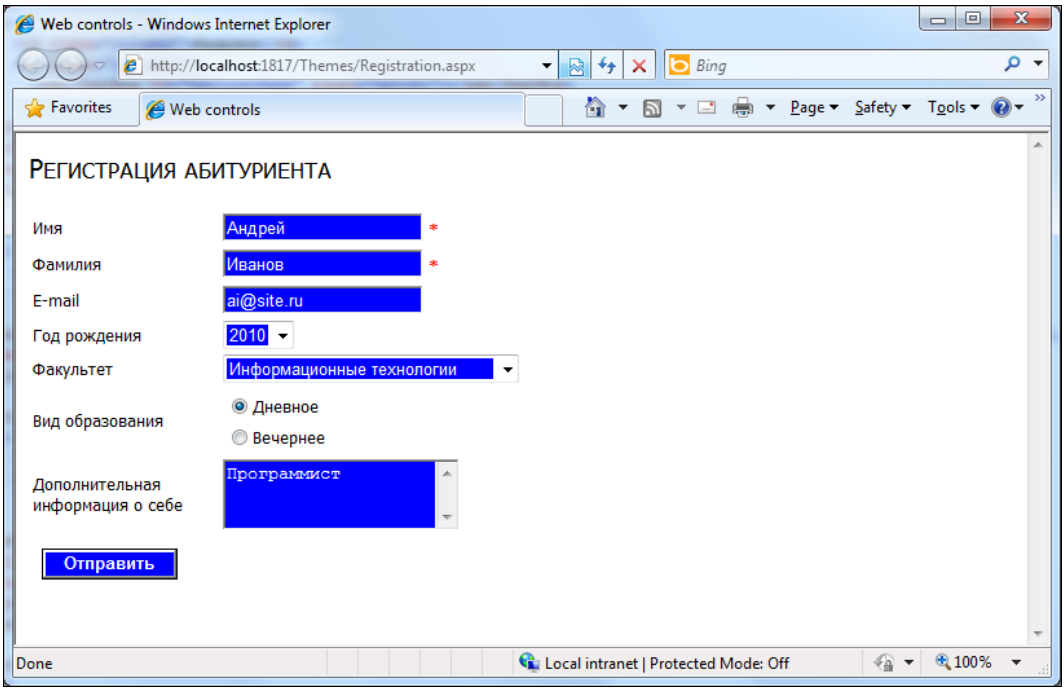


Рис. 12.9. Страница регистрации с подключенной темой

Можете для закрепления навыков работы с темами поэкспериментировать с файлом темы, подключить его для второй веб-страницы (Courses.aspx), придумать какое-нибудь оригинальное оформление для нашего сайта.

При таком объявлении, как в листинге 12.4, тема переопределит локальные настройки элементов управления. Если вы хотите, чтобы тема не меняла локальные настройки элементов управления, можно определить для веб-страницы в директиве `Page` атрибут `StyleSheetTheme`. Атрибут `StyleSheetTheme` указывает, что тема применима к элементам управления на данной веб-странице только там, где элементы управления не имеют явно заданных значений.

При помощи атрибута `StyleSheetTheme` можно задать применение темы только к тем настройкам элементов управления, которые не установлены явным образом. То есть если элемент управления имеет значение для некого атрибута, то будет ис-

пользоваться именно это значение. Если данный элемент управления его не имеет, то будет использоваться значение из темы. Такой тип темы для страницы определяется следующим образом:

```
<%@ Page StylesheetTheme="MyPageTheme" %>
```

Определение темы в файле конфигурации

В файле конфигурации сайта можно определить тему для всего веб-сайта, что дает возможность переключать темы без повторной компиляции веб-сайта. Для этого надо добавить атрибут `Theme` или `StyleSheetTheme` в элемент `pages` внутри тега `<system.web>`, например, так, как показано в листинге 12.5.

Листинг 12.5. Подключение темы для страницы в файле web.config

```
<system.web>
  ...
  <pages theme="Custom" />
  ...
</system.web>
```

Если определить тему в файле конфигурации web.config, данная тема будет применена для всех страниц веб-сайта. Однако, если веб-страница имеет атрибут `Theme` (как в листинге 12.4), тема, определенная на веб-странице, будет иметь приоритет перед темой, определенной в файле web.config.

Вы можете также динамически подключить и настроить тему в программном коде веб-страницы.

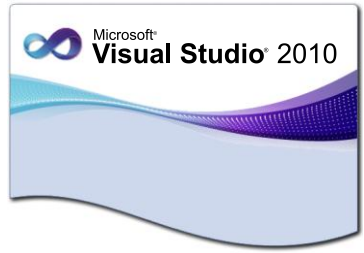
> ***Примечание***
>
> При внесении изменений в файлы тем повторная компиляция веб-сайта не требуется. Эти изменения будут просто применены при следующем обновлении в браузере.

Резюме

Использование стилей позволяет стандартизировать оформление веб-страниц. Кроме того, использование внешнего файла стилей позволяет централизованно и без особых усилий модифицировать внешний вид страниц вашего веб-сайта.

Также как и каскадные таблицы стилей, темы позволяют определять набор атрибутов стилей, которые можно применять к элементам управления на многих страницах. Хотя темы не заменяют стили, они дополняют дизайн веб-сайта возможностями, которых нет в CSS.

Кроме тем для оформления сайта можно создавать мастер-страницы, которые мы будем рассматривать в следующей главе. Мастер-страницы, позволяющие создать макет страницы, который можно будет использовать для всех страниц в приложении. Одна главная страница определяет макет и стандартное поведение, которые можно использовать для всех страниц (или группы страниц) в приложении. Затем можно создать отдельные страницы содержимого, включающие содержимое, связанное со страницей, которое следует отображать.



ГЛАВА 13

Мастер-страницы и управление навигацией

Чтобы построить профессиональный веб-сайт, используют мастер-страницы и элементы, управляющие навигацией по веб-сайту. Мастер-страница представляет собой шаблон страницы, которая позволяет ее повторное использование на всех страницах. Использование мастер-страниц при разработке веб-сайта позволяет создать общий дизайн сайта.

Кроме того, для любого веб-сайта, который содержит множество страниц, необходима возможность навигации по его страницам. С применением мастер-страниц создание навигации для веб-сайта становится намного проще.

Мастер-страницы

Мастер-страницы используются для стандартизации компоновки веб-страницы. Мастер-страницы представляют собой специальные шаблоны веб-страниц, в которых определяется место для фиксированного содержимого и место для специального содержимого, например скриптов. При использовании мастер-страницы на веб-сайте вы получаете одинаковую компоновку для всех веб-страниц этого сайта. Если поменять дизайн мастер-страницы, то все веб-страницы, использующие ее, также поменяют свой дизайн.

Создание мастер-страницы

Мастер-страница создается и добавляется в проект при помощи диалогового окна **Add New Item**. В этом диалоговом окне выберите шаблон **Master Page** и нажмите кнопку **Add** (рис. 13.1).

В проект будет добавлена мастер-страница с именем по умолчанию MasterPage.master. Код разметки шаблона мастер-страницы, сгенерированного средой Visual Studio, показан в листинге 13.1.

Листинг 13.1. Разметка шаблона мастер-страницы

```
<%@ Master Language="C#" AutoEventWireup="true"
  CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
```

```
<!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></title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">

        </asp:ContentPlaceHolder>
    </div>
    </form>
</body>
</html>
```

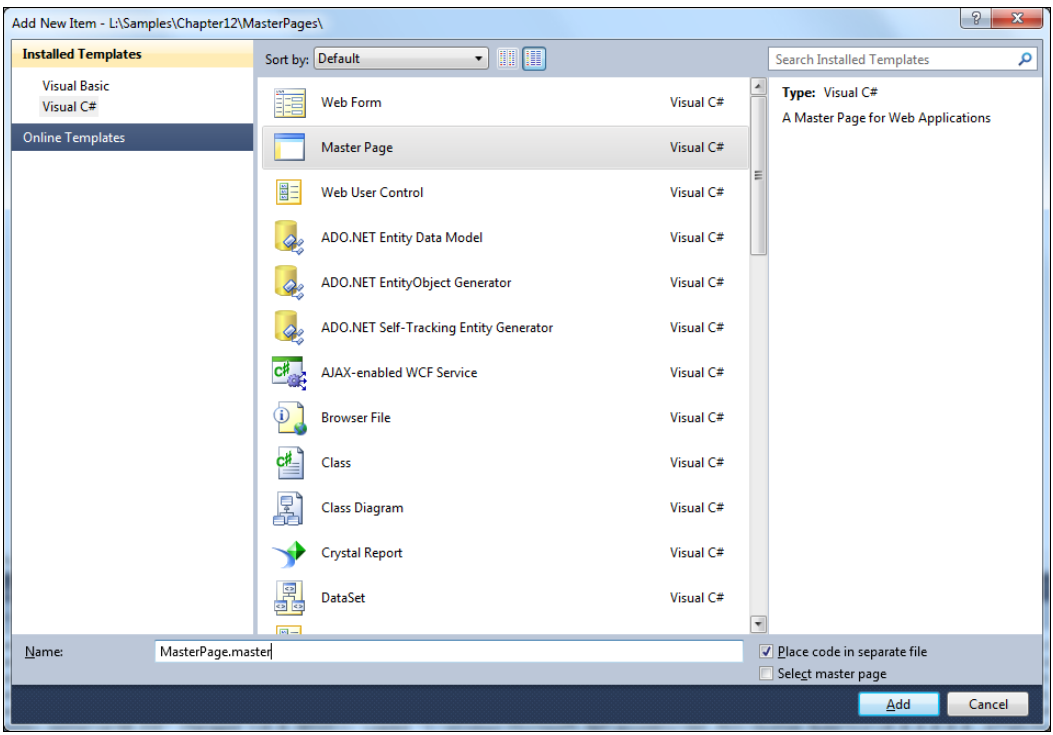


Рис. 13.1. Создание мастер-страницы

Мастер-страница подобна обычной веб-странице ASP.NET. Отличие мастер-страницы от обычной веб-страницы заключается в том, что обычная веб-страница

начинается с директивы `Page`, а мастер-страница начинается с директивы `Master`, например:

```
<%@ Master Language="C#" AutoEventWireup="true"
  CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
```

Кроме того, мастер-страницы, в отличие от простых страниц, содержат элементы управления `ContentPlaceHolder`. Элемент управления `ContentPlaceHolder` является частью страницы, в которую обычная страница может вставлять содержимое (рис. 13.2). Веб-страницы, использующие мастер-страницу в качестве контейнера, называются *страницами содержимого*.

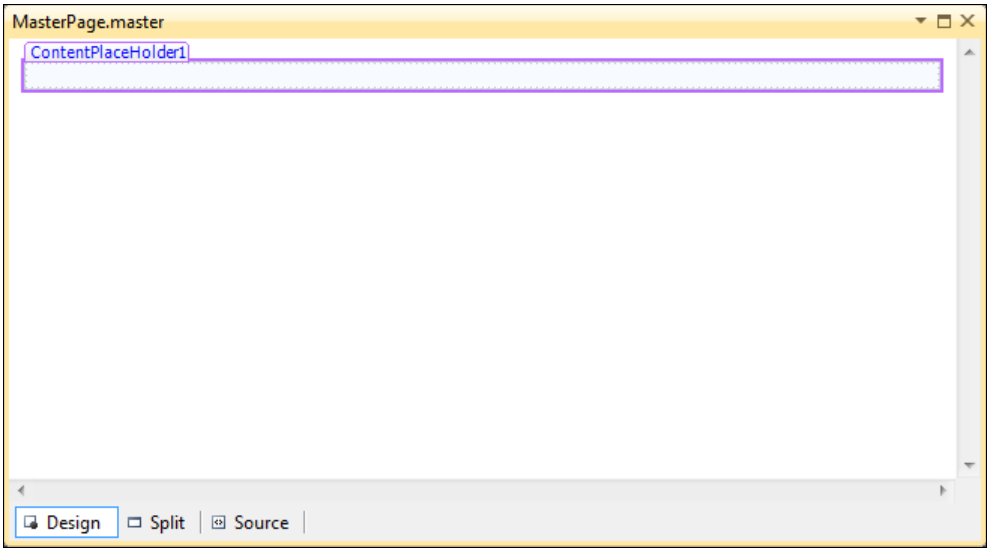


Рис. 13.2. Внешний вид мастер-страницы в дизайнере веб-форм

В шаблоне мастер-страницы из листинга 13.1, созданной средой разработки, уже добавлено два элемента управления `ContentPlaceHolder`. Один из элементов `ContentPlaceHolder` находится в разделе `<head>`, чтобы страницы содержимого могли добавлять свои метаданные. Другой элемент управления `ContentPlaceHolder` определен в разделе `<body>` и предназначен для отображения содержимого страницы.

Для веб-сайта может быть создано несколько мастер-страниц. Эта возможность особенно полезна в том случае, когда веб-сайт имеет несколько компоновок по умолчанию (для разных областей сайта). Обычно мастер-страница содержит общую навигацию, общую графику и общий нижний колонтитул. Вы можете также вкладывать мастер-страницы одна в другую (подробнее об этом далее).

Мастер-страница определяет основной код HTML для страницы. На ней находятся открывающие и закрывающие HTML-теги, заголовок, тело и форма. Внутри мастер-страницы есть один или несколько элементов управления `ContentPlaceHolder`. Эти элементы управления как раз и указывают те области, где будут размещаться веб-страницы.

К мастер-странице также можно подсоединить таблицу стилей. Таким образом, все страницы, которые используют мастер-страницу, будут также связаны с таблицей стилей. Мастер-страницы имеют также собственный файл выделенного кода. Этот файл должен содержать весь код, который относится к функционированию самой мастер-страницы, то есть мастер-страница имеет свой набор событий так же, как и обычная веб-страница ASPX.

Шаблон, который предоставляет среда разработки для мастер-страницы, очень простой и использует потоковую компоновку. Это значит, что по мере добавления содержимого страница реорганизуется, приводя к растягиванию остального содержимого. Такая компоновка не даст возможности получить желаемый результат при использовании мастер-страниц. Поэтому мастер-страницы будут применять либо в большинстве случаев уже используют таблицы или позиционирование CSS для управления компоновкой.

При позиционировании CSS содержимое разбивается на дескрипторы `<div>`, которые затем либо позиционируются по абсолютным координатам, либо располагаются вдоль одной из сторон страницы. После этого в дескриптор `<div>` помещается элемент `ContentPlaceHolder`.

При использовании таблиц вся страница или ее часть разбивается на столбцы и строки, а элементы `ContentPlaceHolder` добавляют в отдельные ячейки. Мы можем создать такую страницу и использовать ее для нашего веб-сайта.

Приведем простой пример создания мастер-страницы с табличной компоновкой. Разметка для мастер-страницы с верхним и нижним колонтитулами, каждый из которых размещен в ячейках таблицы, приведена на рис. 13.3.

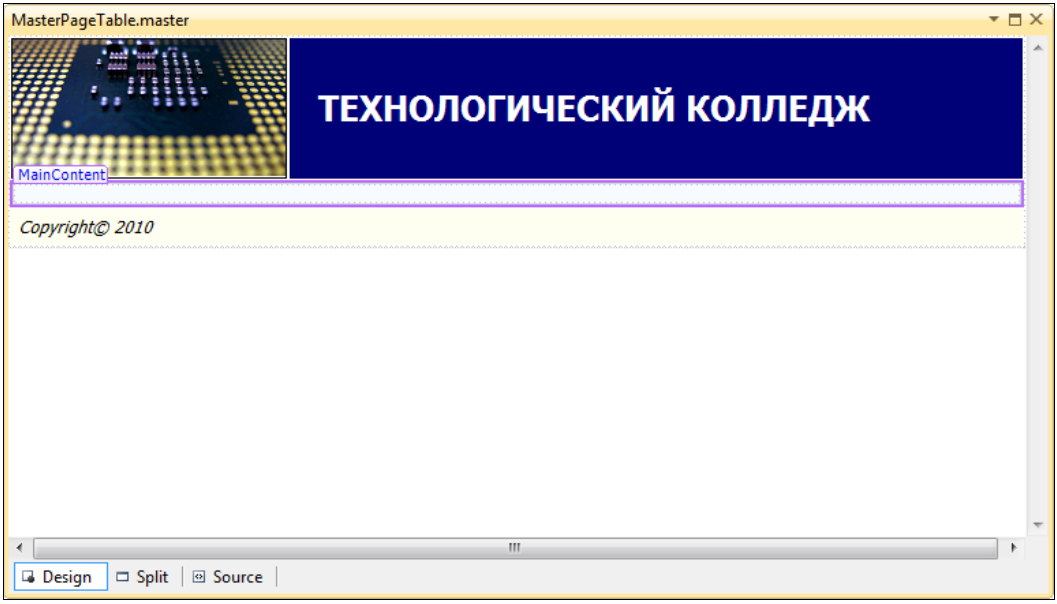


Рис. 13.3. Разметка мастер-страницы

Ячейка в центральной строке (помеченная как `MainContent` на рис. 13.3) содержит элемент управления `ContentPlaceHolder`. Это единственное место, где страницы содержимого (использующие мастер-страницу) могут добавлять свой контент. Код разметки такой мастер-страницы приведен в листинге 13.2.

Примечание

Пример проекта веб-сайта с мастер-страницей находится на прилагаемом к книге диске в каталоге Chapter13\MasterPages.

Листинг 13.2. Мастер-страница с табличной разметкой содержимого

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterPageTable.master.cs" Inherits="MasterPageTable" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
    <link href="~/Styles/StyleSheet.css" rel="stylesheet" type="text/css" />
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <table width="100%">
      <tr>
      <td class="columnImage">
         <asp:Image ID="Image1" runat="server" ImageUrl="~/cpu.png" />
      </td>
        <td style="background:#000077;">
           <h1>Технологический колледж</h1>
        </td>
      </tr>
      <tr>
         <td colspan="2">
            <asp:contentplaceholder id="MainContent" runat="server">
            </asp:contentplaceholder>
         </td>
     </tr>
     <tr><td colspan="2"
          style="border: thin #008000;
          background: #FFFFF1;padding: 5px;">
          <em>Copyright© 2010</em>
         </td>
      </tr>
```

```
    </table>
    </div>
    </form>
</body>
</html>
```

Создание страницы содержимого

Теперь создадим страницу содержимого. Выберите команду **Add New Item** (Добавить новый элемент), выберите шаблон **Web Form**, оставьте название веб-страницы, предлагаемое по умолчанию — Default.aspx, установите флажок **Select master page** и щелкните мышью на кнопке **Add** (рис. 13.4).

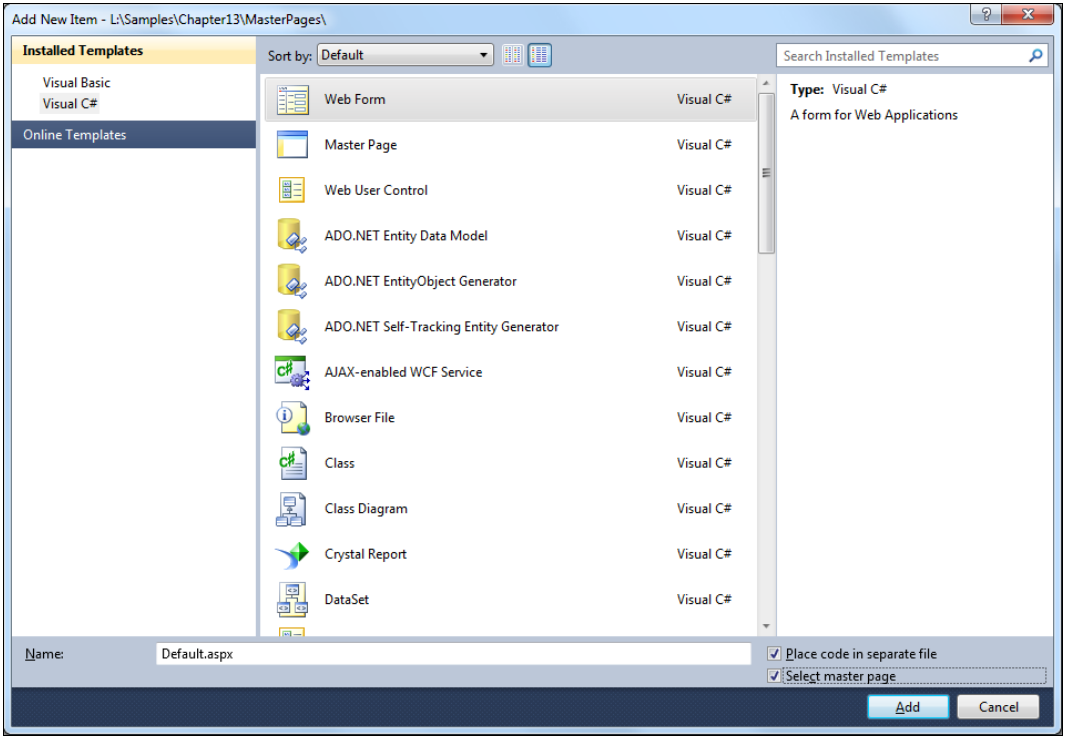


Рис. 13.4. Добавление веб-формы с привязкой к мастер-странице

Среда Visual Studio предложит вам выбрать файл мастер-страницы из текущего веб-проекта (рис. 13.5).

Как только вы сделаете это, среда Visual Studio создаст элемент управления `Content` для каждого элемента управления `ContentPlaceHolder` на мастер-странице.

Таким образом, чтобы создать полную страницу содержимого, которая будет использовать мастер-страницу `MasterPage.master`, необходимо заполнить содержимое элемента управления `ContentPlaceHolder` с `ID`, равным `ContentPlaceHolder1`. Код страницы содержимого Default.aspx показан в листинге 13.3.

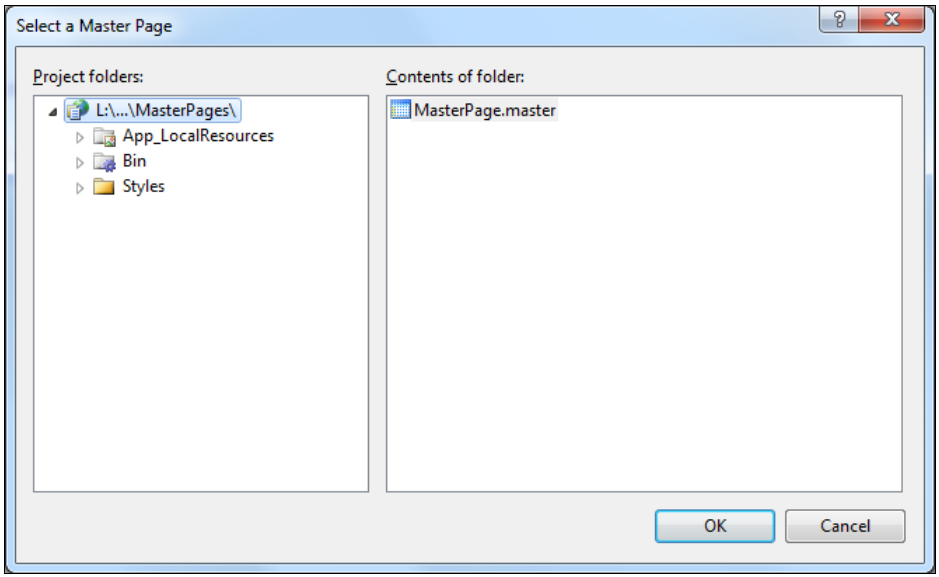


Рис. 13.5. Выбор файла мастер-страницы

Листинг 13.3. Страница содержимого Default.aspx

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPage.master"
   AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>

<asp:Content ID="Content1"
   ContentPlaceHolderID="MainContent" Runat="Server">

</asp:Content>
```

Рассмотрим код, созданный средой разработки для страницы содержимого. Чтобы использовать мастер-страницу в другой веб-странице, необходимо добавить атрибут `MasterPageFile` в директиву `Page`. Этот атрибут показывает имя файла требуемой мастер-таблицы:

```
<%@ Master Language="C#" AutoEventWireup="true"
  CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
```

Теперь внешняя оболочка предоставлена мастер-страницей, то есть элементы `<html>`, `<head>`, `<body>` и `<form>` уже определены на мастер-странице, и на странице содержимого они не используются.

Чтобы предоставить содержимое для элемента управления `ContentPlaceHolder`, используется другой специализированный элемент управления, называемый `Content`. Для каждого элемента управления `ContentPlaceHolder` на мастер-странице страница содержимого предлагает соответствующий элемент управления `Content`:

```
<asp:Content ID="Content1"
   ContentPlaceHolderID="MainContent" Runat="Server">

</asp:Content>
```

Среда ASP.NET связывает элемент управления `Content` с соответствующим элементом управления `ContentPlaceHolder`, сопоставляя идентификатор элемента `ContentPlaceHolder` со свойством `ContentPlaceHolderID` соответствующего элемента управления `Content`.

Можно переделать уже существующие страницы на использование в них мастер-страниц. Для этого надо просто взять код разметки, находящийся внутри элемента `<form>`, и вставить его внутрь элемента `Content`. Сделайте это для нашей страницы регистрации.

Если открыть страницу регистрации в дизайнере веб-форм, вы увидите на заднем плане содержимое мастер-страницы. На переднем плане будут находиться элементы управления `ContentPlaceHolder` (рис. 13.6).

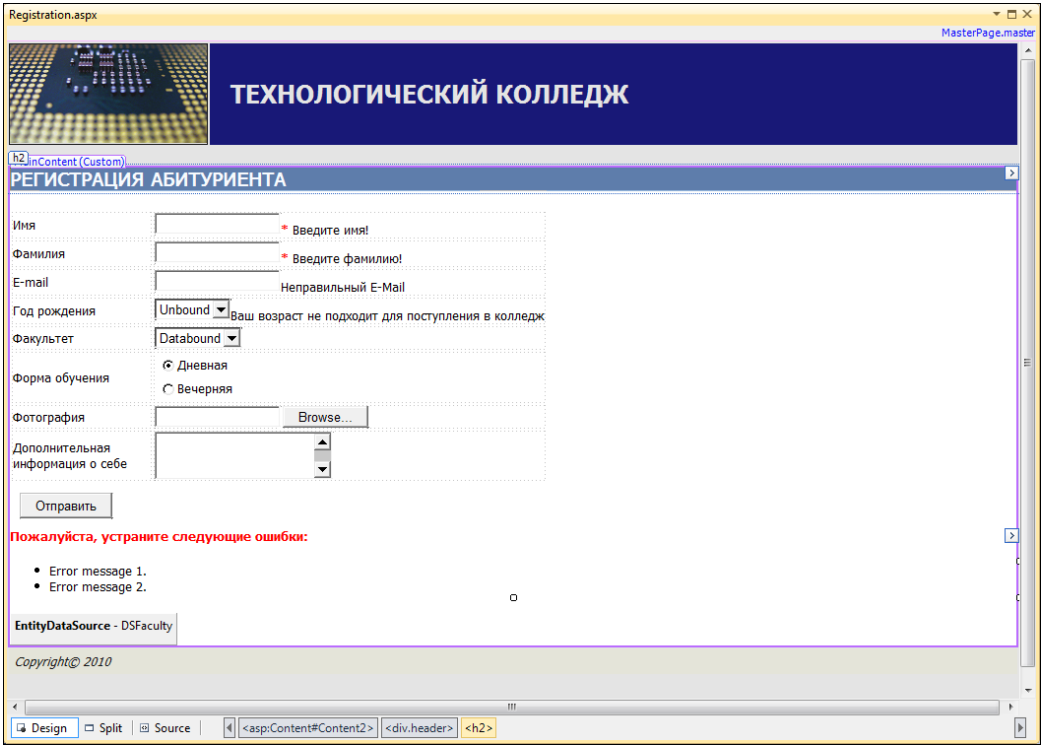


Рис. 13.6. Страница `Registration`, загруженная в мастер-страницу

Именно сюда надо добавлять элементы управления. Таким образом обеспечивается разделение функциональности страницей содержимого и мастер-страницей. Работать со страницей можно точно так же, как и с любой другой веб-страницей ASP.NET. Добавляйте элементы управления на форму и пишите код событий в файле выделенного кода.

Запустите приложение в браузере. Внешний вид нашего веб-сайта с загруженной страницей регистрации абитуриента представлен на рис. 13.7.

В качестве тренировки переделайте остальные веб-страницы, созданные нами в предыдущей главе, в страницы содержимого и добавьте их на наш веб-сайт. Они

понадобятся нам в следующем разделе, когда мы будем изучать элементы управления навигацией и встраивать их в наш веб-сайт.

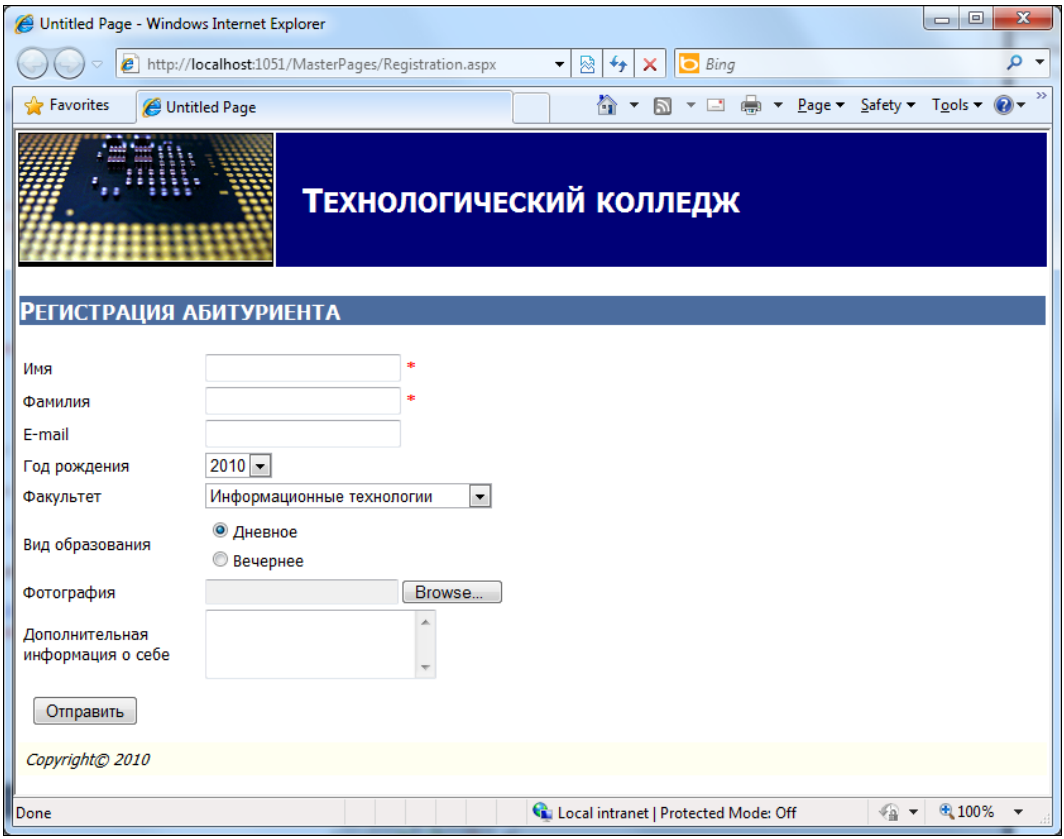


Рис. 13.7. Страница регистрации, вложенная в мастер-страницу

Навигация

Навигация является неотъемлемой частью любого достаточно сложного веб-сайта. В этом разделе мы рассмотрим создание и применение навигационных веб-серверных элементов управления в комбинации с ведущими страницами для обеспечения согласованной навигации в пределах веб-сайта. Для управления навигацией в ASP.NET применяются элементы управления `TreeView` и `Menu`, обладающие широкими функциональными возможностями.

Элемент управления *TreeView*

Элемент управления `TreeView` предназначен для управления навигацией в виде дерева веб-страниц. Он не только позволяет генерировать полнофункциональные

представления деревьев, но и поддерживает заполнение частей дерева по запросу без обновления страницы.

Элемент управления `TreeView` поддерживает широкий диапазон стилей, которые могут преобразовывать его внешний вид. Задавая всего лишь несколько основных свойств, можно добиться того, что элемент управления `TreeView` будет представлять не индекс справочного раздела, а перечень файлов и папок каталога. В действительности, элемент управления `TreeView` вообще не нужно генерировать в виде дерева, так как он может работать также с иерархическими данными, не предназначенными для него — например, с таблицей содержимого — за счет изменения всего лишь некоторых настроек стиля.

Объекты *TreeNode*

Каждый узел в дереве представляется объектом `TreeNode`. Как вы уже знаете, каждый объект `TreeNode` имеет связанную порцию текста, отображаемого в дереве. Объект `TreeNode` предлагает также свойства навигации, такие как `ChildNodes` (коллекция узлов, которые он содержит) и `Parent` (контейнерный узел, на уровень выше в дереве). Наряду с этим объект `TreeNode` содержит специфические свойства:

- `Text` — текст, отображаемый в дереве для данного узла;
- `ToolTip` — текст контекстной подсказки, который появляется при наведении указателя мыши на текст узла;
- `Value` — значение для идентификации этого узла;
- `NavigateUrl` — задает URL веб-страницы, на которую будет осуществлен переход;
- `ImageUrl` — задает URL изображения для узла.

Элемент управления `TreeNode` может использоваться в одном из двух режимов.

- В *режиме выбора* щелчок на узле приводит к обратной отправке страницы и генерации события `TreeView.SelectedNodeChanged()`. Это режим по умолчанию для всех узлов.
- В *режиме навигации* щелчок на узле приводит к переходу на новую страницу, без возникновения события `TreeView.SelectedNodeChanged()`. Объект `TreeNode` перейдет в режим навигации, если свойству `NavigateUrl` присвоить значение, отличное от пустой строки. Объект `TreeNode`, привязанный к данным карты сайта, работает в режиме навигации, поскольку каждый узел карты сайта предоставляет информацию об URL.

Применение стилей к типам узлов

Элемент управления `TreeView` позволяет индивидуально настраивать стили узлов. Для этого в классе `TreeView` существует группа свойств, с помощью которых можно определять стили корневых и дочерних узлов, а также стили для выделенного узла.

Далее перечислены основные свойства стилей.

- `NodeStyle` — стиль применяется ко всем узлам.
- `RootNodeStyle` — стиль применяется только к корневым узлам.
- `ParentNodeStyle` — стиль применяется к узлу, содержащему дочерние узлы, кроме корневых узлов.
- `LeafNodeStyle` — стиль применяется к узлу, не содержащему дочерних узлов и не являющемуся корневым узлом.
- `SelectedNodeStyle` — стиль применяется к выделенному пользователем узлу.
- `HoverNodeStyle` — стиль применяется к узлу, над которым находится указатель мыши.

Элемент управления `TreeView` также обладает встроенной коллекцией изображений для узлов. Чтобы получить доступ к этим изображениям, нужно использовать свойство `ImageSet`, которое принимает одно из 16 значений из перечисления `TreeViewImageSet`. Каждое значение перечисления `TreeViewImageSet` содержит набор изображения для свернутого и развернутого узла, а также для узла, не имеющего дочерних узлов. Если вы будете работать со свойством `ImageSet`, вам не нужно будет использовать какое-либо из свойств, связанных с изображениями.

Разобравшись с особенностями использования элемента `TreeView`, применим его на нашем веб-сайте, добавив на мастер-страницу. Для этого во вторую строку таблицы HTML, которая является разметкой для мастер-страницы, в левую ячейку добавьте код, объявляющий элемент `TreeView` с тремя узлами — `"Учебные программы"`, `"Абитуриентам"`, `"Контакты"`, как показано в листинге 13.4.

Примечание

Пример проекта веб-сайта с управлением навигацией находится на прилагаемом к книге диске в каталоге Chapter13\Navigation.

Листинг 13.4. Код элемента `TreeView` в разметке мастер-страницы

```
<asp:treeview ID="Treeview1" runat="server"
  BorderColor="#6600CC" Font-Bold="True">
  <Nodes>
    <asp:TreeNode Text="Главная страница" NavigateUrl="~/Default.aspx">
      <asp:TreeNode Text="Учебные программы"
        NavigateUrl="~/Courses.aspx"></asp:TreeNode>
      <asp:TreeNode Text="Абитуриентам"
        NavigateUrl="~/Registration.aspx"></asp:TreeNode>
      <asp:TreeNode Text="Контакты"
        NavigateUrl="~/Contacts.aspx"></asp:TreeNode>
    </asp:TreeNode>
  </Nodes>
</asp:treeview>
```

Наш сайт с деревом навигации на левой панели в дизайнере веб-форм должен выглядеть так, как представлено на рис. 13.8.

Запустите наш сайт в браузере. Теперь сайт имеет панель навигации, и вы можете переключаться между веб-страницами (рис. 13.9).

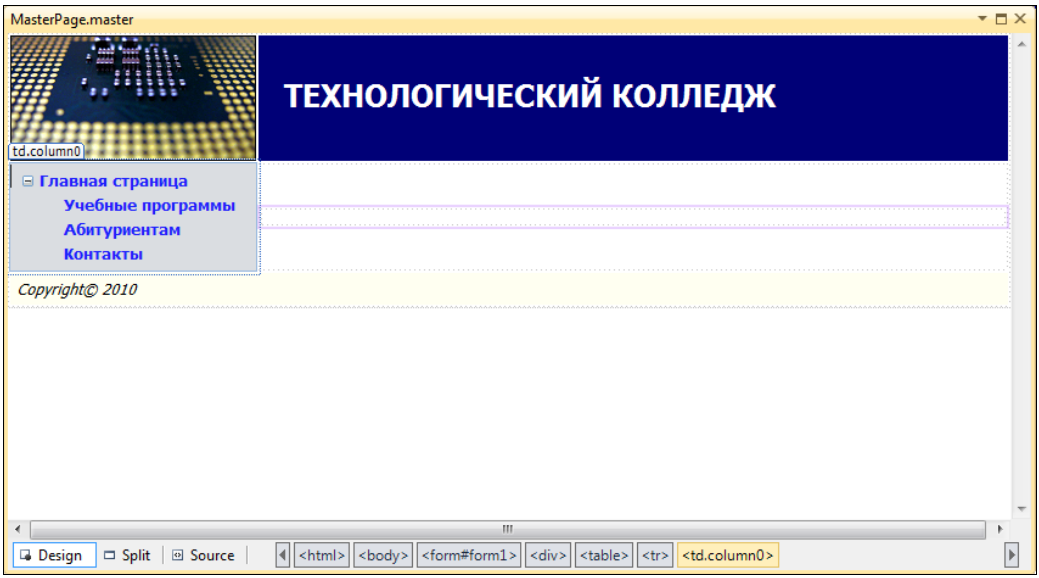


Рис. 13.8. Добавление навигации для сайта

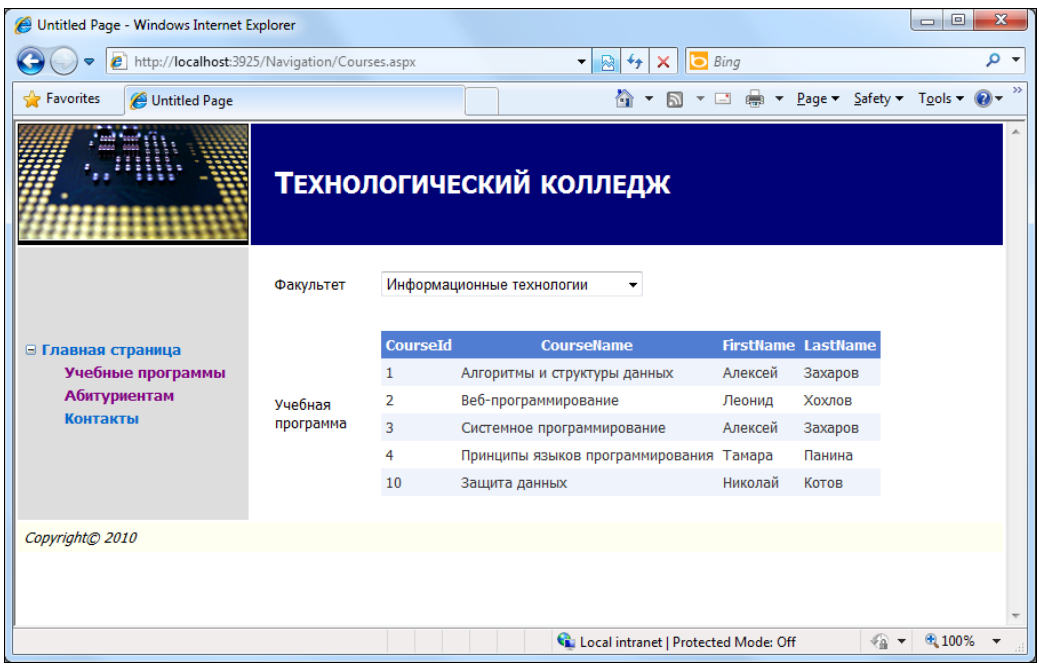


Рис. 13.9. Веб-сайт с панелью навигации

Элемент управления *Menu*

Среда разработки ASP.NET предлагает еще один полнофункциональный элемент управления, поддерживающий иерархические данные, — `Menu`. Этот элемент является контейнером для элементов управления `MenuItem`.

Класс `MenuItem` не является таким же полнофункциональным, как класс `TreeNode` — например, объекты `MenuItem` не поддерживают флажки или программную установку развернутого/свернутого состояния. Тем не менее у них есть много похожих свойств, среди которых есть свойства, отвечающие за настройку изображений, а также свойства, определяющие URL для перехода на заданную вебстраницу, например, `Text`, `ToolTip`, `Value`, `NavigateUrl`, `ImageUrl`. Эти свойства мы уже рассматривали в предыдущем разделе.

Кроме перечисленных свойств, у элемента управления `MenuItem` есть свои специфические свойства, основные из которых перечислены далее.

- `Selectable` — свойство с булевым значением. Если это свойство будет иметь значение `false`, то элемент нельзя будет выделить. Обычно значение `false` присваивается этому свойству только тогда, когда элемент является подзаголовком, содержащим выбираемые дочерние элементы.
- `PopOutImageUrl` — определяет изображение, которое отображается справа от элемента меню, если этот элемент имеет вложенные элементы меню. По умолчанию изображением является стрелка.
- `SeparatorImageUrl` — определяет URL изображения, которое отображается непосредственно под данным элементом меню и отделяет его от остальных элементов меню.

В программном коде вы можете работать со структурой элемента управления `Menu` точно так же, как и с объектом `TreeView`. Элемент управления `Menu` содержит коллекцию объектов `MenuItem` в свойстве `Items`, а каждый объект `MenuItem` имеет коллекцию `ChildItems`, содержащую вложенные элементы.

Стили элемента управления *Menu*

Элемент управления `Menu` предлагает большое количество стилей. Подобно элементу управления `TreeView`, `Menu` является специальным классом, порожденным от базового класса `Style` — в действительности, классов два — `MenuStyle` и `MenuItemStyle`. В результате этого доступны свойства интервалов:

- `ItemSpacing`
- `HorizontalPadding`
- `VerticalPadding`

Однако вы не можете задавать изображения элемента меню посредством стиля, поскольку у него нет свойства `ImageUrl`.

Подобно `TreeView`, элемент управления `Menu` поддерживает определение стилей меню для различных уровней меню. С помощью этих стилей вы можете задавать

стили, специфические для уровней, чтобы разнообразить внешний вид каждого уровня меню и подменю. Это можно сделать с помощью трех коллекций стилей, определяемых в классе `Menu`:

- `LevelMenuItemStyles` — для обычных меню;
- `LevelSubMenuStyles` — для меню, содержащего дочерние пункты;
- `LevelSelectedStyles` — для выделенных пунктов меню.

Элемент управления `Menu` позволяет выбирать количество статических уровней. По умолчанию существует только один статический уровень, а все остальное отображается как плавающие меню, когда пользователь наводит указатель мыши на соответствующий родительский пункт меню. Можно изменить такое поведение с помощью свойства `StaticDisplayLevels`. Например, если этому свойству присвоить значение 2, то первые два уровня меню будут генерироваться на странице с использованием статических стилей. Отступ каждого уровня меню можно устанавливать с помощью свойства `StaticSubMenuIdent`.

Теперь вернемся к нашему веб-сайту и разместим на мастер-странице меню. Для этого добавьте новую строку с размещенным в ней элементом управления `Menu` в разметку мастер-страницы между первой и второй строкой HTML-таблицы, определенной в коде разметки.

Код, который необходимо добавить на веб-страницу, представлен в листинге 13.5.

Листинг 13.5. Код мастер-страницы с элементами управления навигацией

```
<tr>
  <td colspan="2"">
    <asp:Menu ID="NavigationMenu" runat="server" CssClass="menu"
      EnableViewState="false"
      IncludeStyleBlock="false"
      Orientation="Horizontal">
      <Items>
        <asp:MenuItem Text="Главная страница"
          NavigateUrl="~/Default.aspx"/>
        <asp:MenuItem Text="Учебные программы"
          NavigateUrl="~/Courses.aspx"/>
        <asp:MenuItem Text="Абитуриентам"
          NavigateUrl="~/Registration.aspx"/>
        <asp:MenuItem Text="Контакты"
          NavigateUrl="~/Contacts.aspx"/>
      </Items>
    </asp:Menu>
  </td>
</tr>
```

Теперь внешний вид мастер-страницы с добавленным элементом `Menu` должен стать таким, как изображено на рис. 13.10.

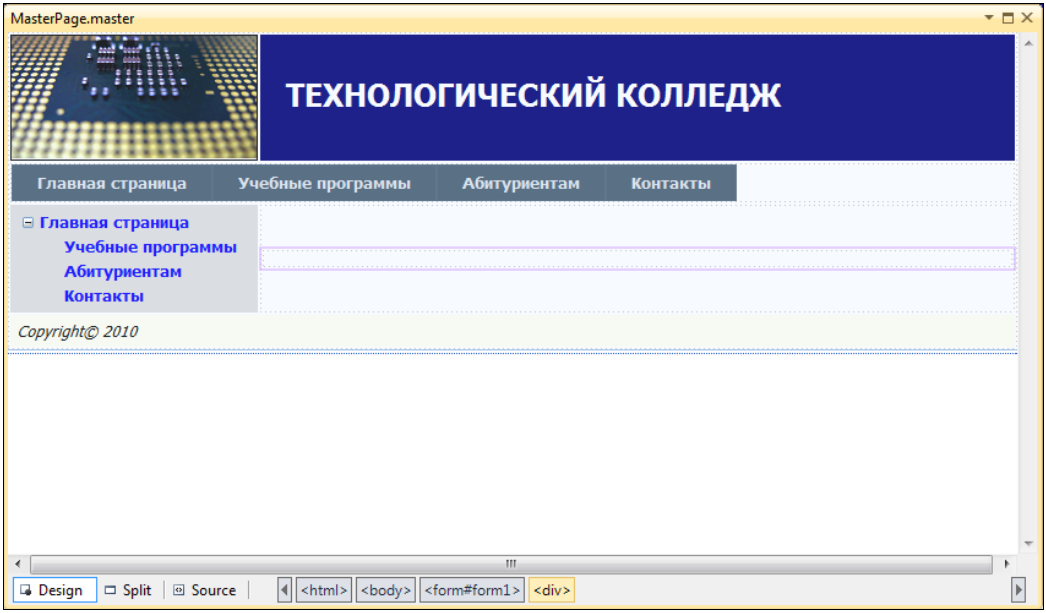


Рис. 13.10. Мастер-страница с добавленным элементом `Menu`

Запустите сайт в браузере. Как мы видим, теперь наш веб-сайт приобрел еще один компонент — меню, для управления навигацией по сайту, дублирующий элемент управления `TreeView` (рис. 13.11).

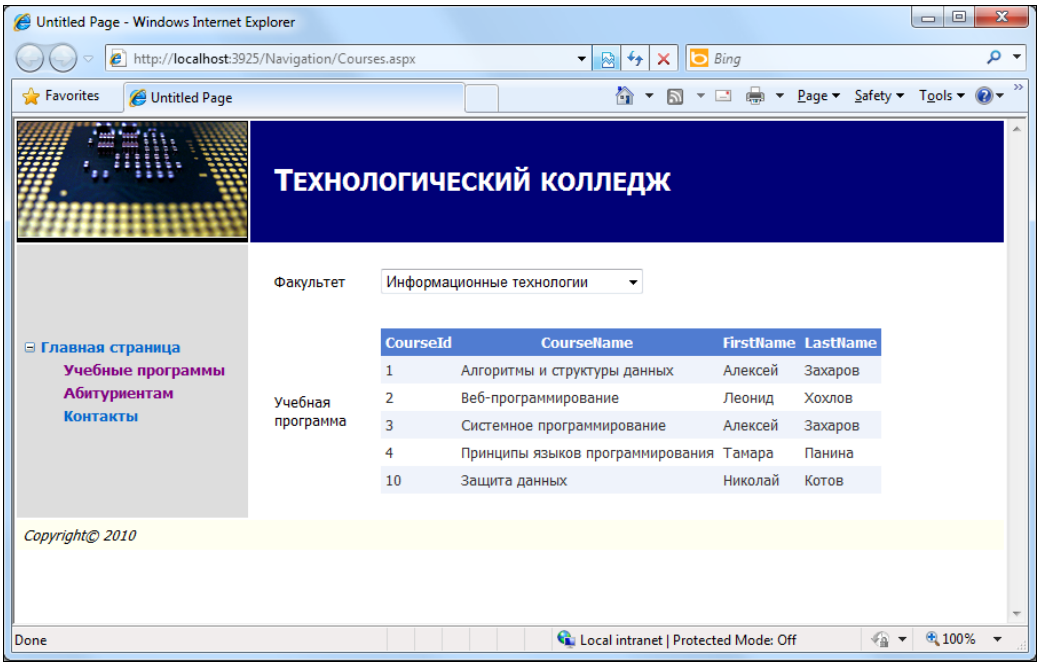
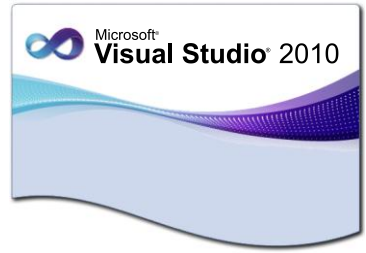


Рис. 13.11. Веб-сайт с элементами управления навигацией `TreeView` и `Menu`

Резюме

В этой главе мы изучили создание и использование мастер-страниц и элементов управления навигацией веб-сайта. Используя эти средства в своих практических разработках, вы можете создавать профессиональные веб-приложения, которые будут иметь унифицированный дизайн и компоновку страниц.



ГЛАВА 14

ASP.NET AJAX

AJAX позволяет усовершенствовать пользовательский интерфейс веб-приложений за счет асинхронного обмена и динамических манипуляций веб-страницей на стороне клиента. Пользователь может взаимодействовать с серверными функциями и данными без необходимости полного обновления страниц. В результате для работы с веб-приложением требуется меньше сетевого трафика, что, в свою очередь, ведет к более быстрой реакции со стороны веб-приложений.

Архитектура AJAX

Библиотека Microsoft AJAX позволяет приложениям, использующим технологию AJAX, выполнять всю обработку на стороне клиента. Клиентское и серверное решение использует как библиотеку Microsoft AJAX, так и серверные элементы управления ASP.NET.

Веб-приложение, использующее технологию Microsoft AJAX, состоит либо из клиентского решения, либо одновременно из клиентского и серверного решения. Только клиентское решение использует библиотеку Microsoft AJAX, но не использует какие-либо серверные элементы управления ASP.NET. Код HTML может включать теги `<script>`, которые ссылаются на файлы JavaScript библиотеки Microsoft AJAX.

Клиентская архитектура включает библиотеки для поддержки компонентов, совместима с обозревателями, имеет сетевые средства и базовые службы.

С помощью клиентских компонентов реализуется пользовательский интерфейс в обозревателе без операций обратной передачи данных на сервер. Клиентские компоненты делятся на три категории:

1. Компоненты, не являющиеся визуальными объектами, которые инкапсулируют код.
2. Компоненты для расширения функциональности, которые расширяют поведение существующих элементов DOM.
3. Элементы управления, представляющие новый элемент DOM, обладающий пользовательским поведением.

Тип используемого компонента зависит от требуемого типа клиентского поведения. Например, водяной знак в существующем текстовом поле можно создать, используя поведение, связанное с текстовым полем.

Серверные составляющие, поддерживающие разработку с использованием технологии AJAX, состоят из серверных веб-элементов управления ASP.NET и компонентов, управляющих пользовательским интерфейсом и потоком выполнения приложения. Эти серверные составляющие также управляют сериализацией, проверкой и расширяемостью элементов управления. Существуют также веб-службы ASP.NET, позволяющие осуществлять доступ к службам приложения ASP.NET, используемым для проверки подлинности на основе форм, поддержки ролей и пользовательских профилей.

Элементы управления AJAX в ASP.NET

Элементы управления AJAX в ASP.NET — это набор элементов управления, которые предназначены для частичного обновления страниц. Частичные обновления страниц позволяют создать для пользователя более комфортную среду, т. к. по каждому запросу пользователя не нужно выполнять обновление всей страницы целиком.

Вместо этого данные элементы управления работают совместно для того, чтобы части страницы могли отправляться и обновляться независимо. Элементы управления ASP.NET управляют этим процессом частичного обновления.

Вы можете использовать элементы управления ASP.NET AJAX при создании веб-страницы в визуальном конструкторе веб-форм. Вы можете перетащить их из панели **Toolbox** на вашу форму и работать с ними точно так же, как и с другими элементами управления. На рис. 14.1 показан список элементов управления AJAX, имеющихся на панели инструментов **Toolbox** в Visual Studio.

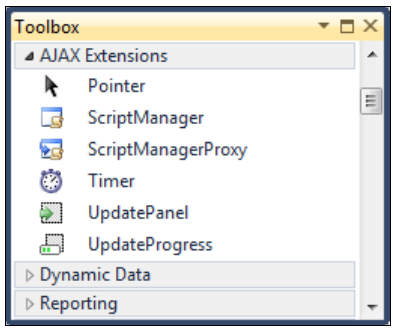


Рис. 14.1. Элементы управления AJAX

Этих элементов всего пять. Вот перечень элементов управления AJAX из группы **AJAX Extensions** панели инструментов.

- `ScriptManager` — управляет библиотекой сценариев AJAX для ASP.NET и файлами сценариев, частичной визуализацией страниц и созданием прокси-класса клиента для веб-служб и служб приложений. Всем страницам, использующим AJAX, обязательно требуется один элемент управления `ScriptManager` на каждой

странице. Он используется ASP.NET для управления другими элементами управления на странице, а также для обработки визуализации страниц по частям, глобализации, локализации и т. д.

- `ScriptManagerProxy` — позволяет таким вложенным компонентам, как страница с содержимым и пользовательские элементы управления, добавлять в страницы ссылки на сценарий и служебные ссылки, если элемент управления `ScriptManager` определен в родительском элементе. Страница может содержать только один элемент управления `ScriptManager`. Элемент управления `ScriptManagerProxy` может использоваться и на пользовательских элементах управления.
- `UpdatePanel` — разрешает частично воспроизводить разделы страницы без обратной передачи. Элементы управления внутри `UpdatePanel`, которые производят обратную передачу на сервер, делают это только для панели, а не для всей страницы. Таким образом, вы получаете легкую модель для обновления только некоторых частей вашей страницы с сервера без полного обновления страницы в браузере.
- `UpdateProgress` — обеспечивает обратную визуальную связь в веб-обозревателе при обновлении содержимого одного или нескольких элементов управления `UpdatePanel`. Элемент управления `UpdateProgress` используется вместе с элементом управления `UpdatePanel` при наличии длительных процессов обновления (чтобы страница при отправке не "зависала"). При обратной передачи части страницы на сервер отображается индикатор прогресса.
- `Timer` — предоставляет таймер, который позволяет выполнять асинхронные или синхронные обратные передачи дочерних элементов управления, находящихся в элементе управления `UpdatePanel` через установленные интервалы времени.

Создание страницы AJAX

Процесс создания страниц с компонентами AJAX очень похож на создание стандартных веб-страниц. Частичное обновление страницы позволяет вам отправить на сервер только часть страницы, обработать ее и обновить в браузере пользователя только эту часть страницы. Это сокращает накладные расходы и обработку на сервере, а пользователи получают улучшенную среду. Обновляется только та часть страницы, с которой они работают.

В этом примере мы усовершенствуем веб-страницу, отображающую информацию об учебных программах, которая позволяет пользователям выполнять поиск заказов клиентов. Эта страница будет реализована как страница с частичным обновлением.

Сначала надо добавить на страницу `ScriptManager`. Вы можете считать, что `ScriptManager` требуется элементу управления `UpdatePanel` для обработки частичного обновления страницы.

Затем добавьте в ячейку таблицы, где расположен `DataGrid`, отображающий список доступных курсов, элемент управления `UpdatePanel`. Этот элемент управления

будет содержать ту часть страницы, которую вы хотите частично обновить. Переместите содержимое `DataGrid` внуть элемента `UpdatePanel`. Помещенные в `UpdatePanel` элементы будут переданы обратно на сервер независимо от остальной части страницы и, соответственно, обновлены без обновления всей страницы. Пользователь не увидит полного обновления страницы.

Вид веб-страницы в дизайнере должен соответствовать приведенному на рис. 14.2.

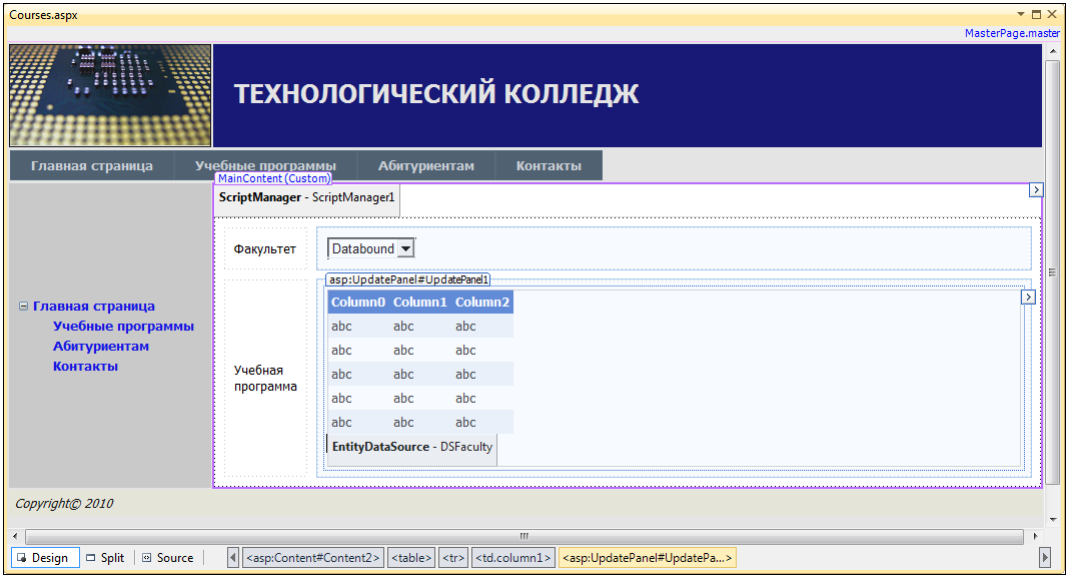


Рис. 14.2. Дизайн веб-страницы с частичным обновлением

Полный код веб-страницы с учебными программами с добавленными элементами управления AJAX представлен в листинге 14.1.

> ***Примечание***
>
> Полный код веб-страницы на прилагаемом к книге диске можно найти в каталоге Chapter14\AjaxControls\Registration.aspx.

Листинг 14.1. Код страницы Registration.aspx с элементами управления AJAX

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Courses.aspx.cs"
  Inherits="Data" MasterPageFile="~/MasterPage.master"%>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
  Runat="Server">
  <asp:ScriptManager
    ID="ScriptManager1" runat="server">
  </asp:ScriptManager>
  <table cellpadding="10" cellspacing="10" style="width:100%;">
```

```
    <tr>
      <td class="column0">
        Факультет</td>
      <td class="column1">
        <asp:DropDownList ID="ListFaculties" runat="server"
          AutoPostBack="True"
          DataSourceID="DSFaculty" DataTextField="FacultyName"
          DataValueField="FacultyId"
          onselectedindexchanged="ListFaculties_SelectedIndexChanged">
        </asp:DropDownList>
      </td>
    </tr>
    <tr>
      <td class="column0">
        Учебная программа</td>
      <td class="column1">
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
          <ContentTemplate>
            <asp:GridView ID="DataGridCourse" runat="server"
                CellPadding="4"
                ForeColor="#333333" GridLines="None">
              <AlternatingRowStyle BackColor="White" />
              <EditRowStyle BackColor="#2461BF" />
              <FooterStyle BackColor="#507CD1" Font-Bold="True"
                ForeColor="White" />
              <HeaderStyle BackColor="#507CD1" Font-Bold="True"
                ForeColor="White" />
              <PagerStyle BackColor="#2461BF" ForeColor="White"
                HorizontalAlign="Center" />
              <RowStyle BackColor="#EFF3FB" />
              <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True"
                ForeColor="#333333" />
              <SortedAscendingCellStyle BackColor="#F5F7FB" />
              <SortedAscendingHeaderStyle BackColor="#6D95E1" />
              <SortedDescendingCellStyle BackColor="#E9EBEF" />
              <SortedDescendingHeaderStyle BackColor="#4870BE" />
            </asp:GridView>
          <asp:EntityDataSource ID="DSFaculty" runat="server"
            ConnectionString="name=CollegeEntities"
              DefaultContainerName="CollegeEntities"
            EnableFlattening="False" EntitySetName="Faculties">
          </asp:EntityDataSource>
        </ContentTemplate>
       </asp:UpdatePanel>
     </td>
    </tr>
  </table>
</asp:Content>
```

Скомпилируйте и запустите веб-сайт. Перейдите на страницу **Учебные программы**. Теперь, когда пользователь выбирает название факультета в элементе `DropDownList`, происходит обновление только таблицы с учебными программами, а не всей страницы (рис. 14.3).

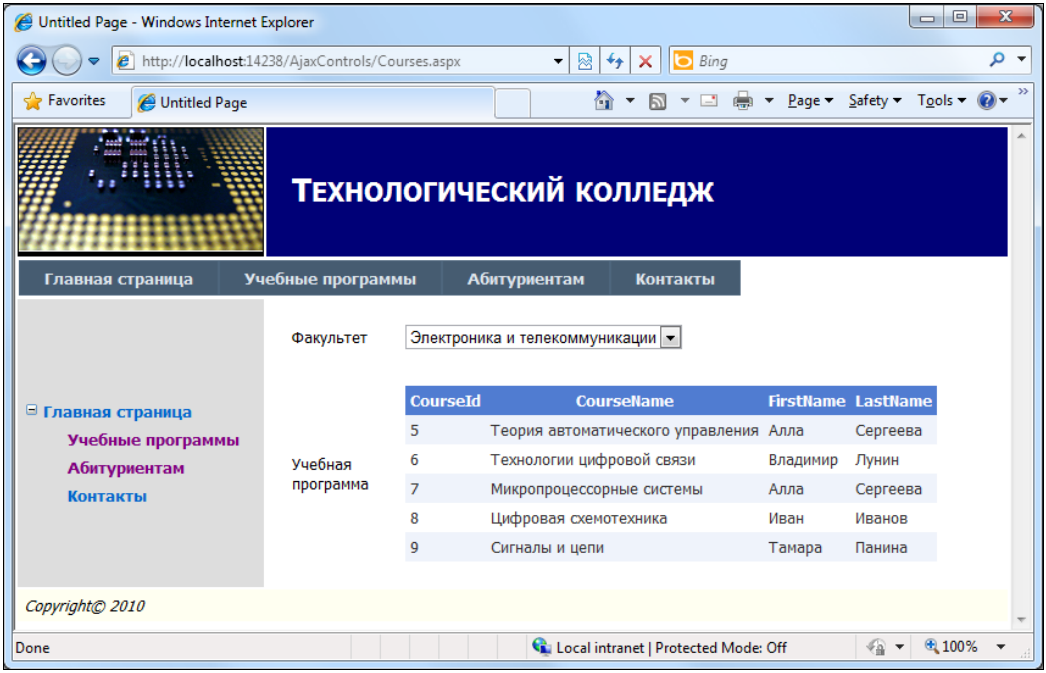


Рис. 14.3. Веб-страница с частичным обновлением

Библиотека AJAX Control Toolkit

Библиотека AJAX Control Toolkit предоставляет множество элементов управления для разработчиков, позволяя вам не писать код на стороне клиента.

Недавно AJAX Control Toolkit стал частью библиотеки ASP.NET AJAX. Новая библиотека ASP.NET AJAX объединяет ASP.NET AJAX и AJAX Control Toolkit в один проект с открытым исходным кодом.

Внутри этого набора находятся нестандартные элементы управления AJAX, которые обеспечивают большое количество динамических действий пользователя внутри браузера. Там имеются элементы управления для всевозможных функций, в том числе автоматическое дописывание для пользователя, поля редактирования текста с шаблонами, проверка силы пароля, модальные окна диалогов и многое другое. Вы можете скачать эту коллекцию элементов управления в виде исходного кода или в виде двоичного файла.

Затем вы можете использовать эти элементы управления в ваших приложениях.

Исходный код пакета AJAX Control Toolkit и двоичные версии имеются по адресу **http://www.codeplex.com**. Там есть варианты под различные версии .NET

Framework. Последние стабильные версии предназначены для .NET Framework 3.5 и 4.0. Вы можете скачать библиотеку вместе с исходным кодом или выбрать версию без исходного кода, содержащую только скомпилированные библиотеки.

После скачивания и распаковывания ZIP-файла вы получаете инструментальный набор AJAX. Вы найдете его в каталоге SampleWebSite\Bin. Библиотека представляет собой библиотеку DLL, в которой содержатся элементы управления AJAX и архив веб-сайта AjaxControlToolkitSampleSite.zip. Этот архив является работающим веб-сайтом и содержит примеры использования и документацию на каждый элемент управления библиотеки AJAX.

Подключение набора элементов AJAX Control Toolkit к панели *Toolbox*

Для подключения набора элементов AJAX Control Toolkit сначала создайте новый сайт или приложение ASP.NET, с которым вы будете работать, или откройте любой созданный ранее проект ASP.NET.

Теперь нужно добавить элементы управления на панель **Toolbox** для того, чтобы их можно было использовать при разработке веб-страниц. Сначала надо создать новую вкладку для группы элементов управления AJAX. Для этого щелкните правой кнопкой мыши на панели инструментов и в контекстном меню выберите опцию **Add Tab**. Назовите эту вкладку, например, **AJAX Control Toolkit**

Затем щелкните правой кнопкой мыши внутри новой вкладки и в контекстном меню выберите опцию **Choose Items**. Откроется диалоговое окно **Choose Toolbox Items** (рис. 14.4)

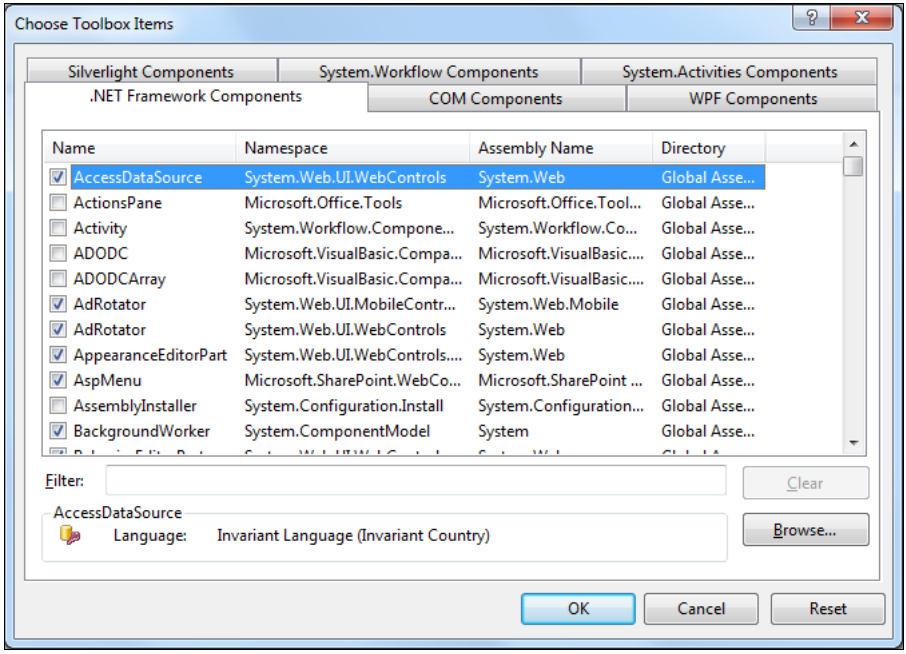


Рис. 14.4. Окно **Choose Toolbox Items**

Это окно позволяет добавлять компоненты на панель **Toolbox**. В окне **Choose Toolbox Items** нажмите кнопку **Browse**. При помощи диалога **Open** перейдите в тот каталог, куда вы развернули архив с инструментальным набором. Здесь вы увидите библиотечный файл AjaxControlToolkit.dll. Выберите его и нажмите кнопку **Open**, а затем кнопку **OK**. После этого элементы управления AJAX появятся на панели инструментов **Toolbox.**

Элементы управления из библиотеки AJAX теперь готовы для использования. На рис. 14.5 показаны элементы управления группы **AJAX Control Toolkit** внутри панели **Toolbox.**

Обратите также внимание на то, что большая часть элементов управления данной таблицы имеет суффикс "Extender". Это означает, что данный элемент управления расширяет уже существующий элемент управления ASP.NET, т. е. данный элемент управления для обеспечения своей функциональности работает с другим элементом управления ASP.NET.

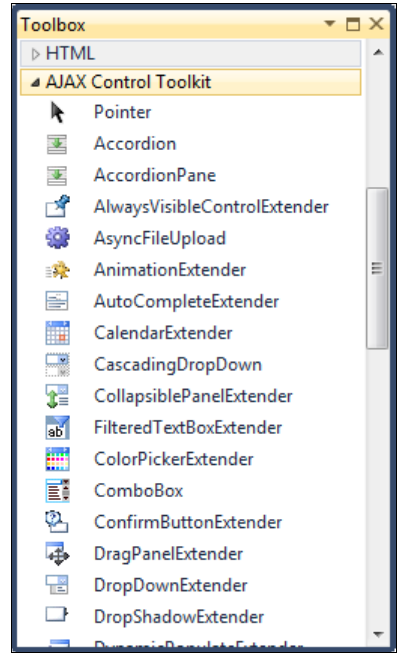


Рис. 14.5. Инструментальный набор **AJAX Control Toolkit** на панели инструментов **Toolbox**

Например, элемент управления AJAX `CalendarExtender` работает с элементом управления ASP.NET `TextBox`. Элементы управления в библиотеке AJAX, которые не имеют суффикса "Extender", используются как самостоятельные элементы управления.

Применение элементов управления AJAX

Набор **AJAX Control Toolkit** значительно облегчает работу со стандартными веб-элементами управления, а тем более с расширяющими элементами управления. Как вы уже видели, в наборе AJAX большинство элементов имеют суффикс "Extender".

Некоторые из них представляют собой довольно сложные элементы управления, например, `Edit` является полнофункциональным текстовым редактором с возможностями форматирования вводимого текста и панелями инструментов для выбора опций форматирования, подобен WordPad в Windows.

Мы можем попробовать этот элемент управления в действии. Для этого создадим пустую страницу ASPX и перенесем на нее элемент управления `ScriptManager` (этот элемент всегда должен присутствовать на страницах, если на них используются элементы AJAX) и элемент управления `Edit` (рис. 14.6).

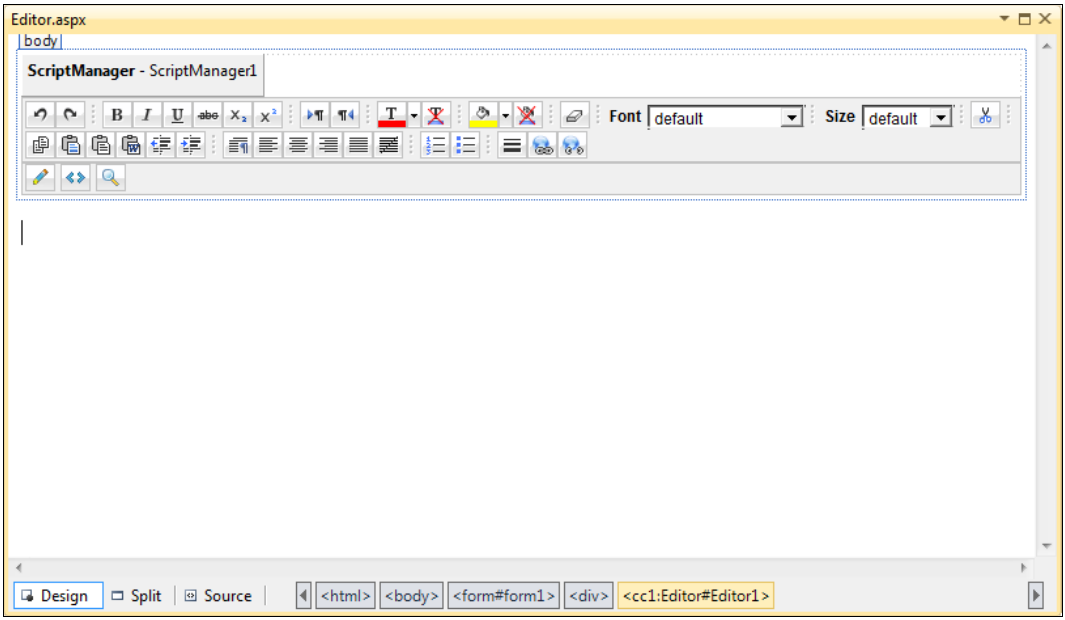


Рис. 14.6. Добавление на страницу элементов управления `ScriptManager` и `Edit`

Код разметки веб-страницы с установленными элементами `ScriptManager` и `Edit` представлен в листинге 14.2.

Листинг 14.2. Разметка страницы Edit.aspx

```
<body>
  <form id="form1" runat="server">
    <div>
      <asp:ScriptManager
        ID="ScriptManager1" runat="server">
      </asp:ScriptManager>
      <cc1:Editor ID="Editor1" runat="server" />
    </div>
  </form>
</body>
```

Скомпилируйте приложение и запустите страницу в браузере. Внешний вид и работа редактора представлены на рис. 14.7.

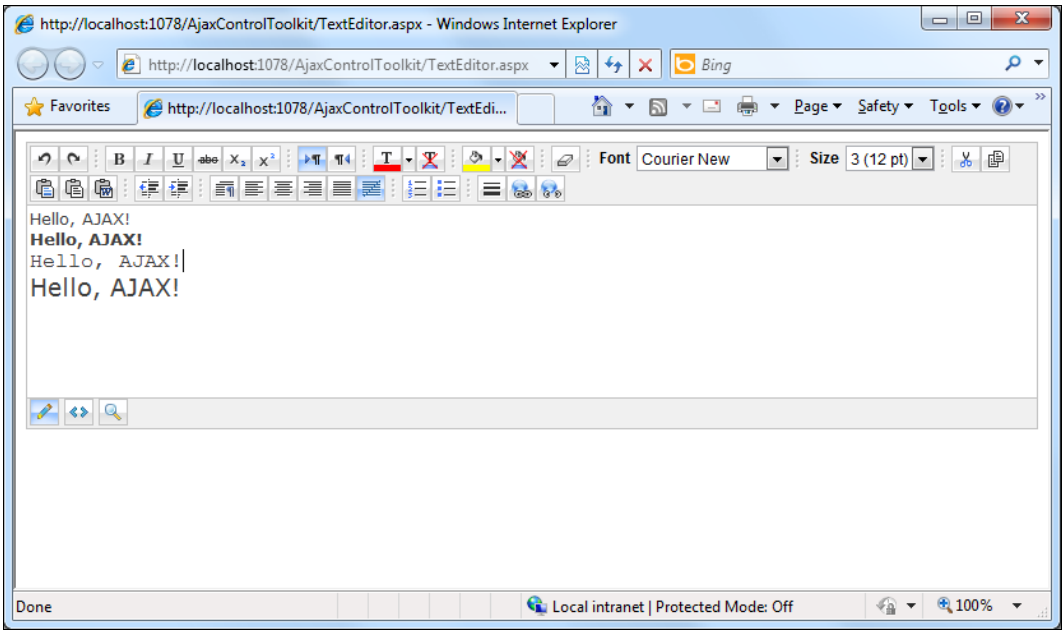


Рис. 14.7. Элемент управления `Edit` в браузере

Расширения для элементов управления

Чтобы получить расширение для существующего элемента управления ASP.NET, можете использовать смарт-тег для данного элемента управления. Для примера такого расширения давайте создадим на нашей странице регистрации абитуриента диалог подтверждения отправки данных на сервер, появляющийся при нажатии кнопки **Отправить**.

Откройте страницу Registration.aspx в дизайнере веб-форм, выберите кнопку `BSubmit` и отобразите ее смарт-тег. В нем появились пункты **Add Extender** и **Remove Extender**. Эти пункты были добавлены средой разработки Visual Studio при подключении набора **AJAX Control Toolkit**. Выберите пункт **Add Extender**, как показано на рис. 14.8.

При этом отобразится диалоговое окно **Choose an Extender** (рис. 14.9). Здесь вы увидите список всех расширяющих элементов управления. Выберите `ConfirmButtonExtender`. Обратите внимание, что `ID` для элемента управления устанавливается автоматически (по расширяемому элементу управления). Нажмите кнопку **OK** для продолжения.

Элемент управления `ConfirmButtonExtender` не имеет визуального представления, поскольку он просто расширяет функциональность элемента управления `Button`. Выделите его в дизайнере веб-форм и перейдите в окно **Properties** этого элемента. Напишите в свойстве `ConfirmText` элемента управления `ConfirmButtonExtender` сообщение для подтверждения операции пользователя, например: "Вы подтверждаете введенную информацию?"

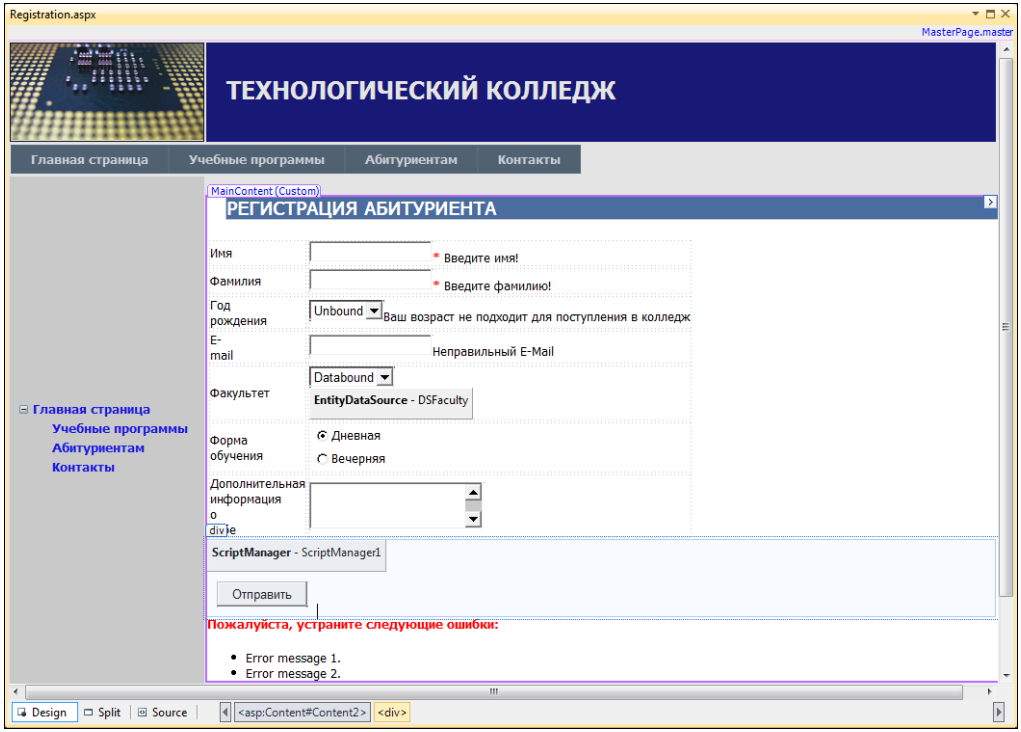


Рис. 14.8. Смарт-тег **Add Extender** для кнопки

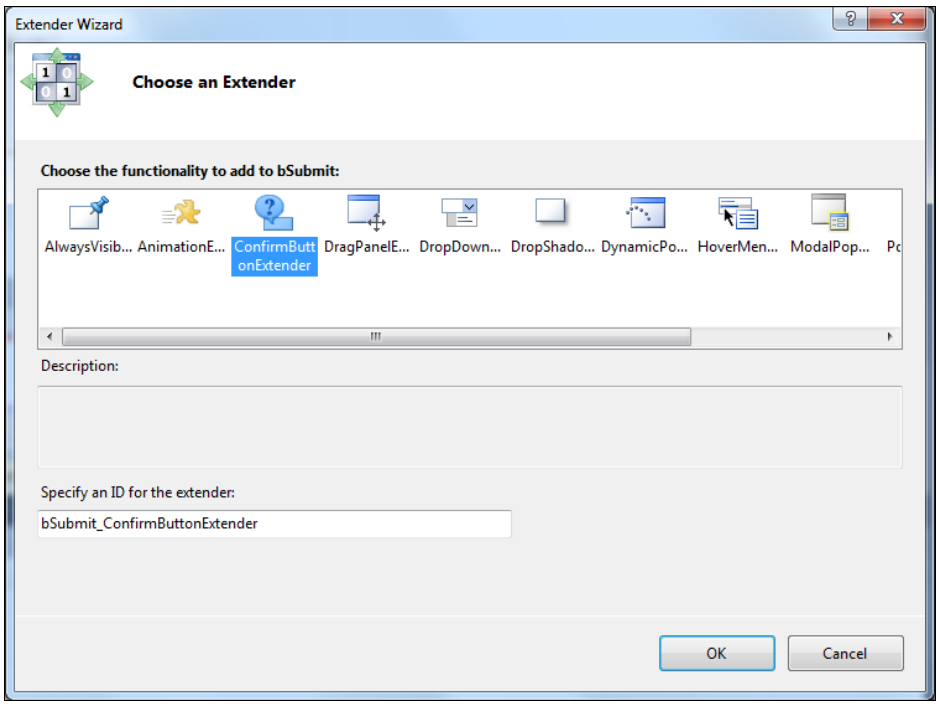


Рис. 14.9. Диалоговое окно **Choose an Extender**

Обратите также внимание, что для свойства `TargetControlID` расширяющего элемента управления было автоматически установлено значение `ID` расширяемого элемента управления (кнопки `BSubmit`). В блоке `<div>` страницы регистрации, где ранее была размещена кнопка **Отправить**, средой разработки должен быть сгенерирован код, представленный в листинге 14.3.

Листинг 14.3. Код расширения для элемента управления `Button`

```
<div>
  <asp:ScriptManager
    ID="ScriptManager1" runat="server">
  </asp:ScriptManager>
  <asp:Button ID="BSubmit" runat="server"
    onclick="bSubmit_Click" Text="Отправить"
    CssClass="button" Width="94px" />
  <asp:ConfirmButtonExtender runat="server"
    ID="BSubmit_ConfirmButtonExtender"
    ConfirmOnFormSubmit="True"
    ConfirmText="Вы подтверждаете введенную информацию?"
    Enabled="True" TargetControlID="BSubmit">
  </asp:ConfirmButtonExtender>
</div>
```

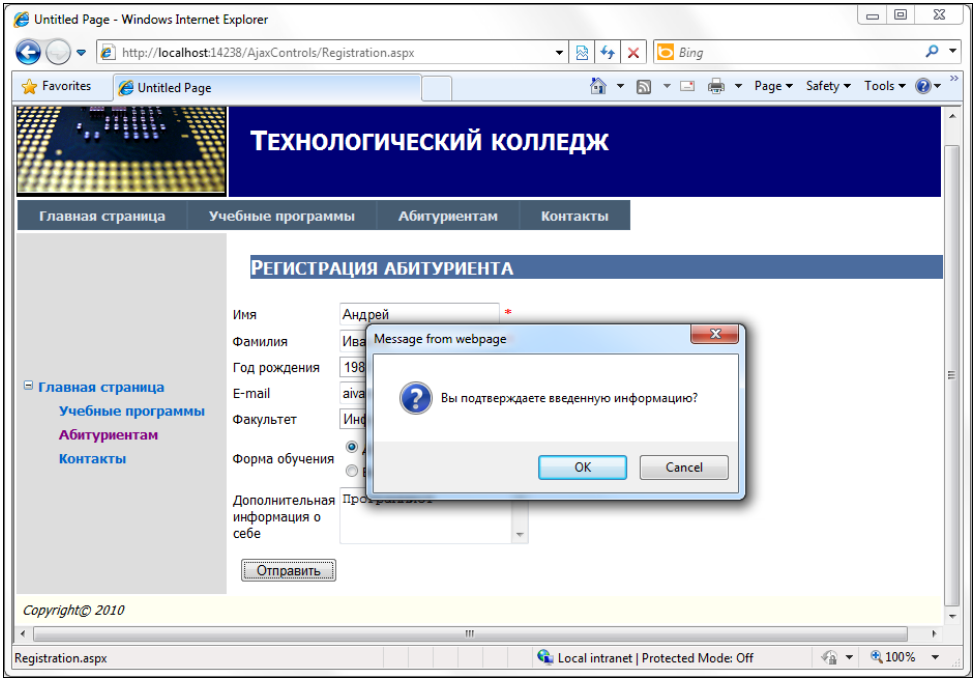


Рис. 14.10. Элемент управления `ConfirmButtonExtender`, связанный с кнопкой `Submit`

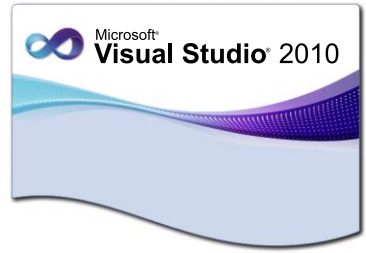
Скомпилируйте и запустите приложение и перейдите на страницу регистрации абитуриента. Нажмите кнопку **Отправить**, и вы получите диалог подтверждения с кнопками подтверждения и отмены (рис. 14.10).

Нажатие кнопки **OK** в диалоговом окне приведет к завершению передачи введенной информации на сервер. При нажатии кнопки **Cancel** произойдет отмена передачи на сервер, и обработчик события, определенный для кнопки в программной части страницы Courses.aspx.cs, не сработает.

Резюме

Использование AJAX позволяет значительно сократить сетевой трафик и снизить нагрузку на сервер при работе с веб-приложением, т. к. вместо загрузки всей страницы достаточно загрузить только часть страницы.

Кроме того, библиотека Microsoft AJAX обертывает код, написанный на JavaScript, в удобную для использования объектно-ориентированную структуру, которую можно легко встраивать в уже существующие веб-страницы.



ГЛАВА 15

Библиотека jQuery

Один из недостатков языка JavaScript состоит в том, что это интерпретируемый, а не компилируемый язык. Не все браузеры одинаково обрабатывают код JavaScript. Библиотека jQuery предоставляет вспомогательные функции, радикально повышающие производительность и облегчающие жизнь.

Принцип работы скриптов библиотеки jQuery заключается в том, чтобы отбирать элементы HTML-страниц и выполнять над ними некоторые операции.

При использовании библиотеки jQuery значительно облегчается написание кода на JavaScript. Библиотека jQuery предоставляет различные вспомогательные функции, повышающие производительность и облегчающие программистам создание веб-приложений.

Библиотека jQuery обеспечивает функциональность для выбора, обхода и управления элементами Web-страниц. Кроме того, jQuery предоставляет всевозможные специальные эффекты. Библиотека jQuery позволяет легко отделять код от дизайна. В итоге написанный код проще читается и более надежен.

Весь набор функциональности, предоставляемой библиотекой jQuery, можно разделить на четыре части:

- работа с моделью DOM;
- визуальные эффекты и анимация;
- возможность интеграции с AJAX;
- функции для работы с данными, фильтрации данных.

Подключение библиотеки jQuery

Библиотека jQuery включена в состав среды разработки Visual Studio 2010 и автоматически подключается при создании некоторых шаблонов ASP.NET проектов и веб-сайтов. Например, можно создать проект с помощью шаблона ASP.NET Web Site (рис. 15.1).

В структуре этого проекта, представленной на рис. 15.2, есть каталог Scripts, который содержит 3 файла:

- jquery-1.4.1.vsdoc.js
- jquery-1.4.1.js
- jquery-1.4.1.min.js

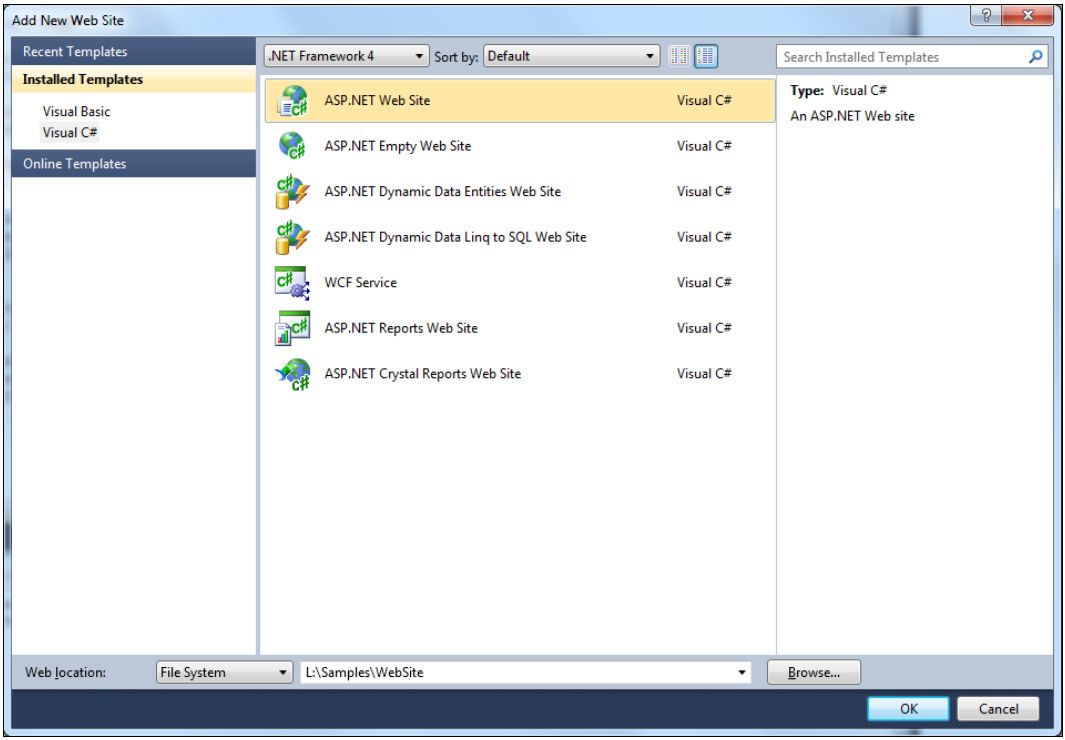


Рис. 15.1. Шаблон ASP.NET Web Site

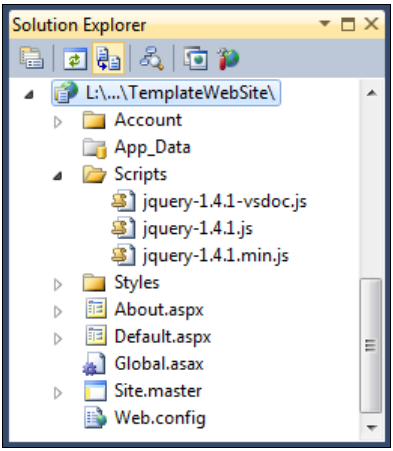


Рис. 15.2. Структура шаблона веб-сайта

Файл jquery-1.4.1.vsdoc.js необходим для полноценной поддержки технологии IntelliSense при написании кода на jQuery в редакторе Visual Studio.

Файл jquery-1.4.1.js предназначен для разработчиков. Это несжатая версия библиотеки jQuery. Она снабжена комментариями в коде и ее используют в основном при отладке веб-приложения.

Файл jquery-1.4.1.min.js — минимизированная версия библиотеки, предназначенная для работы на клиенте. Это сжатая версия, ее размер в несколько раз меньше версии jquery-1.4.1.js, т. к. из ее исходного кода удалены все дополнительные символы и комментарии.

> ***Примечание***
>
> Visual Studio 2010 содержит версию jQuery 1.4.1. Последнюю версию jQuery можно скачать на сайте проекта по адресу **http://jquery.com**.

Шаблон проекта веб-сайта, сгенерированного средой разработки Visual Studio 2010, представляет собой готовое веб-приложение с мастер-страницей, двумя страницами содержимого: **Home** и **About**, а также с элементами аутентификации для входа в систему. В данной главе мы будем использовать его лишь для изучения работы с библиотекой jQuery, но в будущем я советую вам познакомиться с этим шаблоном подробнее, он пригодится вам при создании полноценных веб-приложений. Запущенный в браузере шаблон веб-сайта представлен на рис. 15.3.

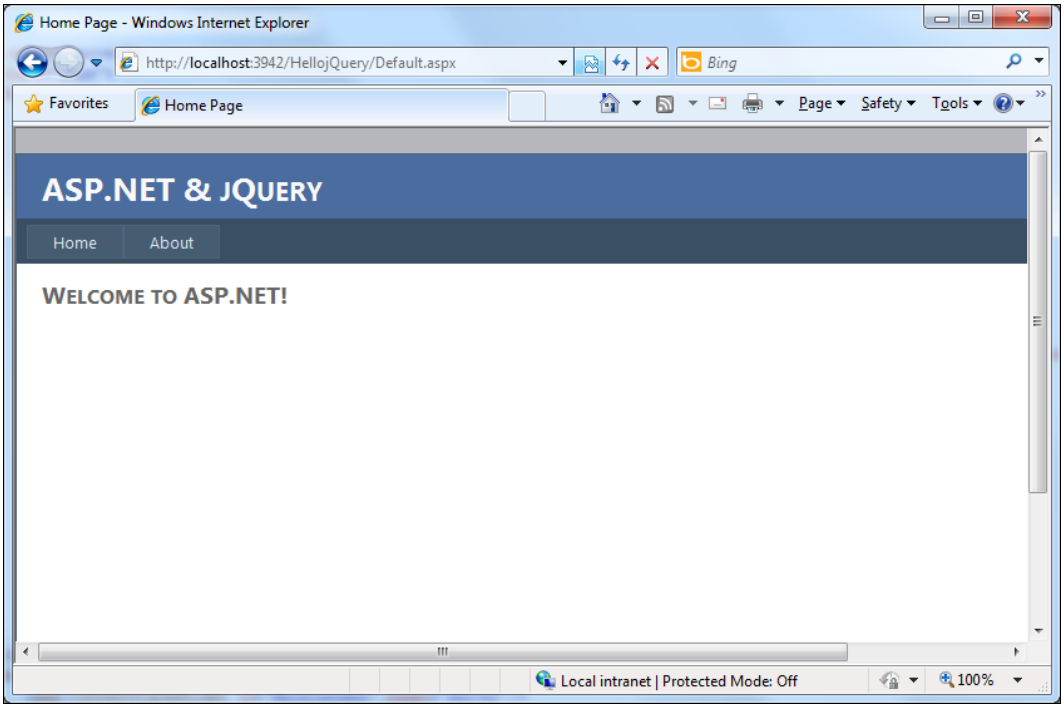


Рис. 15.3. Шаблон ASP.NET Web Site в браузере

Объекты библиотеки jQuery

Библиотека jQuery предоставляет интерфейс для выбора элементов модели DOM. Например, библиотека jQuery позволяет отбирать элементы HTML-страниц и выполнять над ними некоторые операции. В *главе 12* вы уже познакомились с применением CSS в разработке стилей для веб-сайта. Например, при использовании CSS на веб-странице можно с помощью jQuery выбрать все элементы, которые имеют один и тот же класс CSS или которые обладают определенными атрибутами. Можно также добавлять условия фильтра и связывать все эти возможности запроса вместе, как при запросе данных на SQL.

Основой библиотеки jQuery является функция `jQuery`, определенная следующим образом:

```
var jQuery = window.jQuery = window.$ = function( selector, context )
{
   return new jQuery.fn.init( selector, context );
};
```

Функция `$` является *псевдонимом* для функции `jQuery`. При создании объекта `jQuery` передаются *селектор* и *контекст*. Селектор указывает выражение запроса, а контекст указывает, на какой части модели DOM следует выполнить запрос. Если контекст не указан, функция jQuery ищет элементы модели DOM в пределах модели DOM всей страницы.

Функция `$()` возвращает специальный объект JavaScript, который содержит массив элементов DOM, соответствующих указанному селектору. У этого объекта много удобных предопределенных методов, способных воздействовать на группу элементов.

Функция `jQuery`, так же как и ее псевдоним `$`, выполняет запрос и возвращает новый объект `jQuery`, который содержит результаты выполнения запроса. К этому вновь созданному объекту `jQuery`, в свою очередь, может быть выполнен новый запрос или же он может быть отфильтрован в новом операторе или в их цепи.

Вот пример использования функции `$` для выборки массива элементов HTML с именем `Table1`:

```
var elem = $("#Table1");
```

В этом примере кода функция `$` извлекает все элементы модели DOM, свойство идентификатора которых совпадает с указанным выражением. Символ `#` не принадлежит строке `ID`, а является префиксом функции `$`, с помощью которого различаются строки идентификаторов, классы CSS и имена тегов HTML. Символ `#` является частью стандартного синтаксиса CSS для выбора идентификатора. Если при изучении CSS вы на это не обратили внимание, вернитесь к *главе 12* и посмотрите заново синтаксис CSS.

На этом заканчивается сходство между методами модели DOM и функцией `$`. Однако возможностей для использования функции `$` библиотеки jQuery гораздо больше, чем у методов DOM.

Селекторы

Давайте теперь познакомимся поближе с селекторами библиотеки jQuery и их синтаксисом на основе CSS.

Выражение селектора управляется синтаксисом CSS 3.0 и может быть достаточно сложным. В иерархии селекторов предыдущий элемент также может быть любым допустимым селектором, а не только элементом HTML.

Библиотека jQuery поддерживает множество способов выбора одного элемента или группы элементов, например, следующим образом:

```
// Выбор элемента div с ID div1
$("#div1")

// Выбор всех элементов div
$("div")

// Выбор всех div с классом standardDiv
$("div.colorDiv")

// Выбор всех элементов с классом standardDiv
$(".colorDiv ")
```

В дополнение к предыдущим селекторам существует набор более сложных селекторов, примеры которых показаны здесь:

```
// Выбирает первый элемент div
$("div:first");

// Выбирает последний элемент div
$("div:last");

// Выбирает четные элементы div
$("div:even");

// Выбирает нечетные элементы div
$("div:odd");

// Выбирает первый дочерний элемент div
$("div:first-child");

// Выбирает последний дочерний элемент div
$("div:last-child");

// Выбирает третий дочерний элемент div
$("div:nth-child(3)");

// Выбирает все помеченные флажки
$(":checked");
```

```
// Выбирает любую гиперссылку, начинающуюся с текста http://
$("a[hrefA=http://]");

// Выбирает все неактивные элементы на странице
$(":disabled");

// Выбирает все активные элементы
$(":enabled");
```

Библиотека jQuery также предоставляет собственные встроенные селекторы для выборки элементов, имеющих специфические свойства, например:

```
// Выбирает скрытые элементы
$(":hidden")

// Выбирает выбранные элементы
$(":selected")

// Выбирает видимые элементы
$(":visible")

// Выбирает элементы, которые не имеют класса standardDiv
$("div:not(.standardDiv)")
```

Использование селекторов и фильтров

Приступим теперь к практическому использованию селекторов jQuery. Создайте новый проект ASP.NET Web Site (или используйте созданный в предыдущем разделе) и добавьте в проект новую страницу содержимого, по имени selectors.htm (листинг 15.1).

Примечание

Полный код мастер-страницы на прилагаемом к книге диске можно найти в каталоге Chapter15\jQuery\Site.master.

Листинг 15.1. Фрагмент с элементом `Menu` мастер-страницы `Site.master`

```
<div class="clear hideSkiplink">
    <asp:Menu ID="NavigationMenu" runat="server" CssClass="menu"
        EnableViewState="false" IncludeStyleBlock="false"
        Orientation="Horizontal">
        <Items>
            <asp:MenuItem NavigateUrl="~/Default.aspx" Text="Home"/>
            <asp:MenuItem NavigateUrl="~/Selectors.aspx"
               Text="Selectors"/>
            <asp:MenuItem NavigateUrl="~/Accordion.aspx"
               Text="Accordion"/>
            <asp:MenuItem NavigateUrl="~/Effects.aspx" Text="Effects"/>
            <asp:MenuItem NavigateUrl="~/About.aspx" Text="About"/>
```

```
        </Items>
    </asp:Menu>
</div>
```

Для использования библиотеки jQuery на веб-странице необходимо установить ссылку на эту библиотеку в коде веб-страницы. Для этого перетащите мышью файл jQuery из панели **Solution Explorer** в окно редактора кода или добавьте внутри элемента `<asp:content>` с атрибутом `ContentPlaceHolderID`, равным `HeadContent`, веб-страницы следующий код:

```
<script src="Scripts/jquery-1.4.1-vsdoc.js" type="text/javascript">
</script>
```

После этого добавьте несколько встроенных стилей, чтобы продемонстрировать функциональность jQuery. Например, добавьте следующие стили в дескриптор `header`:

```
<style type="text/css">
  .color
  {
    background:#00FFFF;
  }
</style>
```

Далее добавим блок сценария, содержащий две функции: `filter1` и `filter2`, первая из которых будет подсвечивать красным все нечетные ячейки, а вторая будет искать в таблице строки, в которых хотя бы одна ячейка имеет атрибут `align=center`, и менять стиль текста в этих ячейках на `bold`:

```
<script type="text/javascript">
    function filter1() {
      $("#Table1 td:odd").css("background", "#FF0000");
    }
    function filter2() {
      $("#Table1 tr:has(td[align=center])").css("color", "#0000FF");
    }
</script>
```

Для вызова этих функций понадобятся две кнопки и, конечно, сама HTML-таблица с произвольным содержимым. Полный код веб-страницы Selectors.aspx представлен в листинге 15.2.

> ***Примечание***
>
> Полный код веб-страницы на прилагаемом к книге диске можно найти в каталоге Chapter15\jQuery\Selectors.aspx.

Листинг 15.2. Веб-страница Selectors.aspx для работы с фильтрами и селекторами jQuery

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.master"
AutoEventWireup="true" CodeFile="Selectors.aspx.cs" Inherits="Selectors" %>
```

```
<asp:Content ID="Content1" ContentPlaceHolderID="HeadContent" Runat="Server">
  <script src="Scripts/jquery-1.4.1.js" type="text/javascript"></script>

  <script type="text/javascript">
    function filter1() {
      $("#Table1 td:odd").css("background", "#FF0000");
    }

    function filter2() {
      $("#Table1 tr:has(td[align=center])").css("color", "#0000FF");
    }
  </script>

   <style type="text/css">
   .color
   {
    background:#00FFFF;
   }
   </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" Runat="Server">
<table border="1px" cellspacing="5px" id="Table1">
   <tr>
     <td>Column 1</td>
     <td>Column 2</td>
     <td>Column 3</td>
   </tr>
   <tr>
     <td>1</td>
     <td align="center">A</td>
     <td>A</td>
   </tr>
   <tr>
     <td>2</td>
     <td align="center">B</td>
     <td>A</td>
   </tr>
   <tr>
     <td>3</td>
     <td>C</td>
     <td>A</td>
   </tr>
   <tr>
     <td>4</td>
     <td>D</td>
     <td>D</td>
   </tr>
```

```
    <tr>
      <td>5</td>
      <td>E</td>
      <td>E</td>
    </tr>
</table>
<br />

<div>
  <input type="button" value="Filter1" onclick="javascript:filter1();"/>
  <input type="button" value="Filter2" onclick="javascript:filter2();"/>
  </div>
</asp:Content>
```

Внешний вид страницы Selectors.aspx в дизайнере форм, содержащий таблицу и кнопки, показан на рис. 15.4.

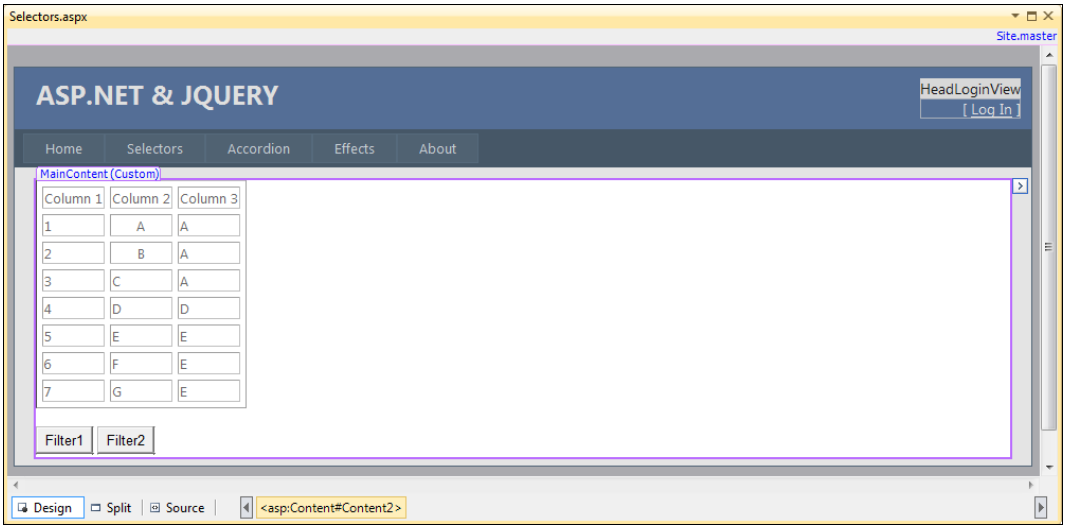


Рис. 15.4. Страница Selectors.aspx в дизайнере форм

Запустите ваше веб-приложение в браузере, перейдите на страницу **Selectors** и нажмите на этой странице кнопку **Filter1**. Все ячейки таблицы с нечетными значениями индекса окрасятся в красный цвет (рис. 15.5).

Обратите внимание на шахматную раскраску ячеек: в таблицах HTML индексы ячеек имеют сквозную нумерацию и не привязаны к строкам таблицы.

При нажатии кнопки **Filter2** скрипт будет искать в таблице строки, в которых ячейки (хотя бы одна) имеют атрибут `align="center"` и меняют стиль текста в этих ячейках на `bold`, как показано на рис. 15.6.

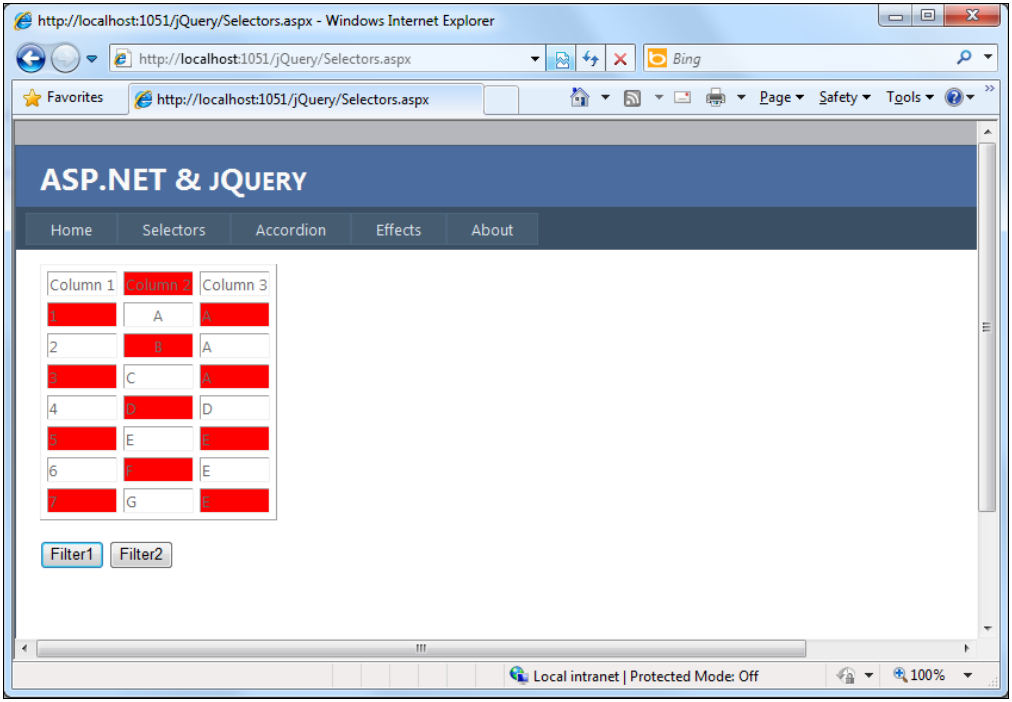


Рис. 15.5. Работа фильтра jQuery для выделения цветом нечетных ячеек таблицы

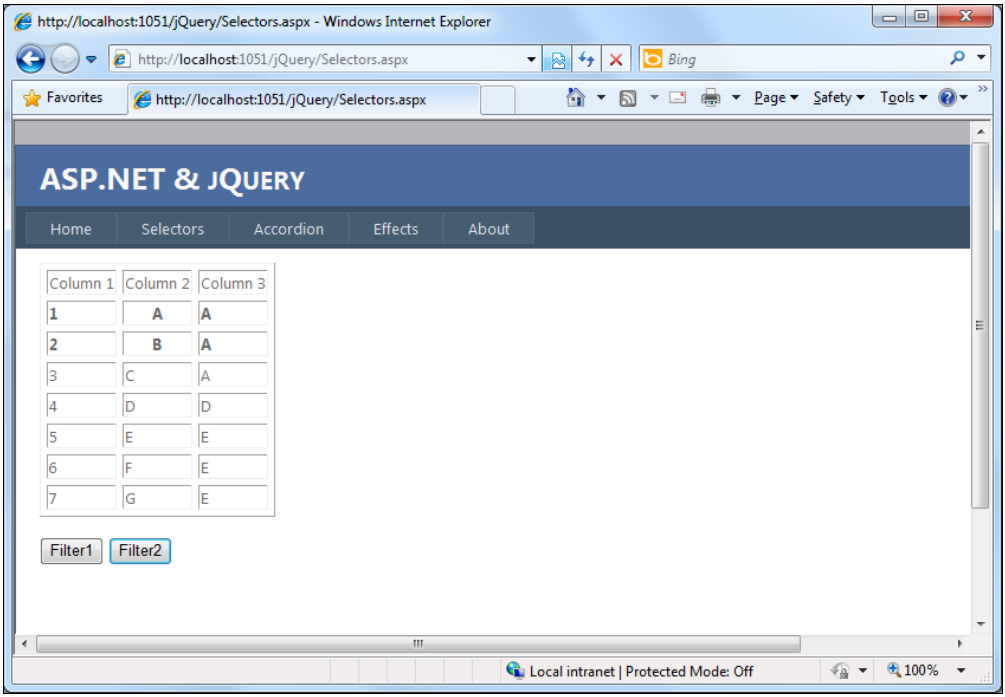


Рис. 15.6. Форматирование текста в строках, в которых хотя бы одна ячейка имеет свойство `align="center"`

Фильтры и эффекты

В библиотеке jQuery есть много интересных графических эффектов. Вот пример одного из эффектов:

```
$("#div1").fadeOut();
```

При вызове функции `fadeOut()` элемент `div` должен постепенно исчезнуть.

Большинство функций графических эффектов в библиотеке jQuery позволяют использовать входные параметры различного типа, например:

```
// задание времени выполнения эффекта в миллисекундах
$ ("#div1").fadeOut (3000);

// задание скорости выполнения эффекта:
$ ("#div1").fadeOut ("slow");
```

При необходимости можно также определить функцию обратного вызова, которая будет реализовывать какую-либо функциональность после завершения графического эффекта:

```
function hellojQuery()
{
  $("#div1").fadeOut('slow',funtionToCall);
}
Function functionToCall()
{
  // какое-либо действие
}
```

Далее показаны примеры вызова функций библиотеки jQuery, которые вы можете использовать для отображения различных эффектов и анимации элементов на веб-странице:

```
// Скрытие элемента
$("#div1").hide();

// Отображение элемента
$("#div1").show();

// Постепенное исчезновение элемента
$("#div1").fadeOut();

// Постепенное появление элемента
$("#div1").fadeIn();

// Переключатель видимости элемента
$("#div1").toggle();

// Анимация элемента в течение времени, указанного в миллисекундах
$("#div1").animate({width: "100%", fontSize: "100px" }, 1500 );
```

```
// Остановка любой анимации или эффекты в прогрессе
$("#div1").stop();
```

Кроме перечисленных ранее функций, в библиотеке jQuery содержится ряд дополнительных функций для создания графических эффектов:

- `fold()` — свертывание;
- `pulsate()` — пульсация;
- `puff()` — раздувание;
- `bounce()` — подпрыгивание;
- `explode()` — взрыв.

Попробуем теперь работу с фильтрами и эффектами на их основе на практике. Создайте новую веб-страницу Effects.aspx, на которую поместите код, показанный в листинге 15.3.

> ***Примечание***
>
> Полный код веб-страницы на прилагаемом к книге диске можно найти в каталоге Chapter15\jQuery\Effects.aspx.

Листинг 15.3. Веб-страница Effects.aspx

```
<%@ Page Title="Page 1" Language="C#"
   MasterPageFile="~/Site.master"
   AutoEventWireup="true"
   CodeFile="Effects.aspx.cs" Inherits="Effects" %>

<asp:Content ID="HeaderContent" runat="server"
   ContentPlaceHolderID="HeadContent">
   <script src="Scripts/jquery-1.4.1.js" type="text/javascript"></script>
   <style type="text/css">
   .colorDiv
   {
   background:#00FFFF;
   width:200px;
   height:50px;
   color:Black;
   }

   .column0
   {
     width:100px;
   }

   td
   {
     padding:5px;
   }
   </style>
```

```
   <script type="text/javascript">
    function textOut() {
      $("#div1").html("Some text. Some text. Some text. Some text.");
    }

    function toggleDiv() {
      $("#div2").toggle(3000);
    }

    function hideDiv() {
      $("#div3").hide();
    }

    function showDiv() {
      $("#div3").show();
    }

    function fadeOutDiv() {
      $("#div4").fadeOut(3000);
    }

    function fadeInDiv() {
      $("#div4").fadeIn(3000);
    }

    function animateDiv() {
      $("#div5").animate({ width: "100%", fontSize: "40px" }, 1500);
    }

    function slideDownDiv() {
      $("#div6").slideDown();
    }

    function slideUpDiv() {
      $("#div6").slideUp();
    }
</script>

</asp:Content>
<asp:Content ID="BodyContent" runat="server"
    ContentPlaceHolderID="MainContent">
  <table style="width:100%;">
    <tr>
      <td>
        <input type="button" value="Text out"
           onclick="javascript:textOut();"/>
      </td>
```

```
    <td>
      <div id="div1" class="colorDiv"></div>
    </td>
    <td>
      <input type="button" value="Toggle Div" on
         click="javascript:toggleDiv();"/>
    </td>
    <td>
      <div id="div2" class="colorDiv">Some text</div>
    </td>
  </tr>
  <tr>
    <td>
      <input type="button" value="Hide"
        onclick="javascript:hideDiv();"/>
      <input type="button" value="Show"
        onclick="javascript:showDiv();"/>
    </td>
    <td>
    <div id="div3" class="colorDiv">Some text</div>
    </td>
    <td>
      <input type="button" value="Fade Out"
        onclick="javascript:fadeOutDiv();"/>
      <input type="button" value="Fade In"
        onclick="javascript:fadeInDiv();"/>
    </td>
    <td>
      <div id="div4" class="colorDiv">Some text</div>
    </td>
  </tr>
  <tr>
    <td>
      <input type="button" value="Animate"
        onclick="javascript:animateDiv();"/>
    </td>
    <td>
     <div id="div5" class="colorDiv">Some text</div>
    </td>
    <td>
      <input type="button" value="Slide Up"
        onclick="javascript:slideUpDiv();"/>
      <input type="button" value="Slide Down"
        onclick="javascript:slideDownDiv();"/>
    </td>
    <td>
      <div id="div6" class="colorDiv">Some text</div>
    </td>
```

```
    </tr>
  </table>
</asp:Content>
```

Запустите веб-страницу в браузере и посмотрите различные интересные эффекты, предоставляемые функциями библиотеки jQuery в действии (рис. 15.7).

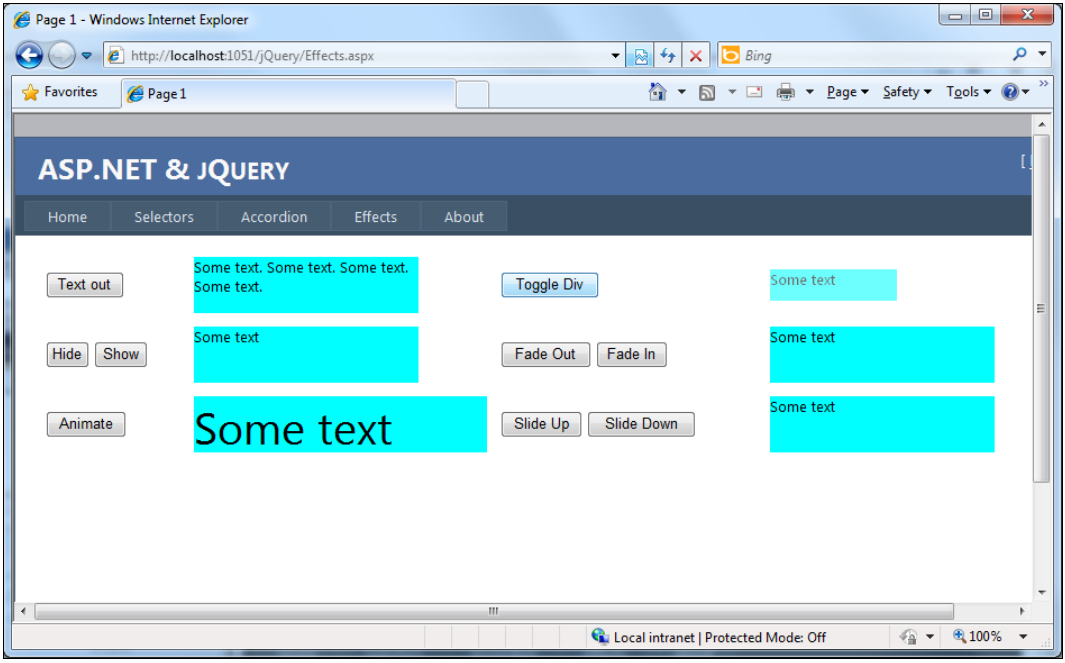


Рис. 15.7. Внешний вид страницы
с набором различных графических эффектов и анимации

Как видите, графические эффекты, предоставляемые библиотекой jQuery, очень легко встроить в уже готовые веб-страницы и тем самым разнообразить дизайн вашего веб-сайта.

Операции над множествами

В библиотеке jQuery есть много разных методов для работы над множествами элементов. Например, таким способом можно получить первый элемент из множества элементов `<div>`:

```
$("div").get(0);
$("div")[0];
```

Если необходимо получить все элементы `<div>`, надо вызвать функцию `get()` без параметров:

```
$("div").get();
```

В библиотеке jQuery есть фукция `.each()`, которая работает подобно оператору `foreach` в C# .NET и позволяет выполнять итерации по набору элементов. Следующий пример выполняет цикл по всему набору элементов `<div>`:

```
function showDivNumber() {
  $("div").each(function(index) {
    ...
  });
}
```

Создание элемента *Accordion*

С помощью функций библиотеки jQuery можно создавать разнообразные интересные элементы управления. Примером использования функций библиотеки jQuery может служить создание компонента `Accordion`, который помещает набор элементов DOM в заданную область и расширяет их по одному. Компонент такого типа обычно используют на веб-странице в качестве ленты новостей. Щелчок по заголовку скрывает или, наоборот, отображает содержимое, разбитое на отдельные секции. При отображении содержимого одной секции открытая ранее секция обязательно закрывается.

Сейчас мы создадим простой компонент `Accordion` с использованием библиотеки jQuery. Нам потребуется родительский элемент DOM; каждый элемент, дочерний по отношению к нему, становится отдельной панелью в составе области. Для каждой панели мы определяем элемент `<h3>` для заголовка панели и `<p>` для содержимого панели.

Создайте новую страницу Accordion.aspx и подключите для нее ссылку на мастер-странице Site.master, создав новый элемент меню. Полный код страницы Accordion.aspx представлен в листинге 15.4.

> ***Примечание***
>
> Полный код веб-страницы на прилагаемом к книге диске можно найти в каталоге Chapter15\jQuery\Accordion.aspx.

Листинг 15.4. Страница Accordion.aspx

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.master"
  AutoEventWireup="true" CodeFile="Accordion.aspx.cs"
  Inherits="Accordion" %>

<asp:Content ID="Content1" ContentPlaceHolderID="HeadContent"
  Runat="Server">
<script src="Scripts/jquery-1.4.1.js" type="text/javascript"></script>

<script type="text/javascript">
  $(document).ready(function () {
    $(".accordion h3:first").addClass("active");
    $(".accordion p:not(:first)").hide();
```

```
    $(".accordion h3").click(function () {
      $(this).next("p").slideToggle("slow").siblings("p:visible")
          .slideUp("slow");
      $(this).toggleClass("active");
      $(this).siblings("h3").removeClass("active");
    });
  });
</script>

<style type="text/css">
.accordion {
  width: 500px;
  border-bottom: solid 1px #C0C0C0;
}
.accordion h3 {
  padding: 8px 16px;
  margin: 0;
  font: bold 120%/100% Arial, Helvetica, sans-serif;
  border: solid 1px #C0C0C0;
  border-bottom: none;
  cursor: pointer;
}
.accordion h3.active {
  background-position: right 5px;
}
.accordion h3:hover {
  background-color: #D3D3D3;
}
.accordion p {
  background: #F0F0F0;
  margin: 0;
  padding:  10px 10px 10px;
  border-left: solid 1px #C5C5C5;
  border-right: solid 1px #C5C5C5;
}
</style>
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
    Runat="Server">
<div class="accordion">
  <h3>News 1</h3>
  <p>Some text. Some text. Some text. Some text. Some text. Some text.
     Some text. Some text. Some text. Some text. Some text. Some text.
     Some text. Some text. Some text. Some text. Some text. Some text.
  </p>
  <h3>News 2</h3>
  <p>Some text. Some text. Some text. Some text. Some text. Some text.
```

```
      Some text. Some text. Some text. Some text. Some text. Some text.
      Some text. Some text. Some text. Some text. Some text. Some text.
   </p>
   <h3>News 3</h3>
   <p>Some text. Some text. Some text. Some text. Some text. Some text.
      Some text. Some text. Some text. Some text. Some text. Some text.
      Some text. Some text. Some text. Some text. Some text. Some text.
   </p>
   <h3>News 4</h3>
   <p>Some text. Some text. Some text. Some text. Some text. Some text.
      Some text. Some text. Some text. Some text. Some text. Some text.
      Some text. Some text. Some text. Some text. Some text. Some text.
   </p>
   <h3>News 5</h3>
   <p>Some text. Some text. Some text. Some text. Some text. Some text.
      Some text. Some text. Some text. Some text. Some text. Some text.
      Some text. Some text. Some text. Some text. Some text. Some text.
   </p>
   <h3>News 6</h3>
   <p>Some text. Some text. Some text. Some text. Some text. Some text.
      Some text. Some text. Some text. Some text. Some text. Some text.
      Some text. Some text. Some text. Some text. Some text. Some text.</p>
</div>
</asp:Content>
```

Внешний вид страницы Accordion.aspx в дизайнере форм, содержащий таблицу и кнопки, показан на рис. 15.8.

Попробуйте компонент в действии. Запустите приложение в браузере. При выборе одного из заголовков будет отображено содержимое этой секции. При отображении содержимого одной секции открытая ранее секция обязательно закрывается. Внешний вид веб-страницы показан на рис. 15.9.

Резюме

Для полного изучения возможностей библиотеки jQuery в рамках этой главы было невозможно показать все многообразие способов использования данного интерфейса в страницах. Приведенные в этой главе примеры веб-страниц с внедрением скриптов jQuery были созданы с расчетом показать, как можно решать различные задачи, с которыми ежедневно приходится сталкиваться при разработке страниц веб-приложений.

Библиотека jQuery будет полезна для любой страницы, которая должна выполнять различные действия и позволит авторам страниц применять методику ненавязчивого JavaScript. При таком подходе поведение элементов отделяется от структуры документа так же, как CSS позволяет отделить информацию о представлении от структуры, за счет чего улучшается организация страниц и увеличивается гибкость программного кода.

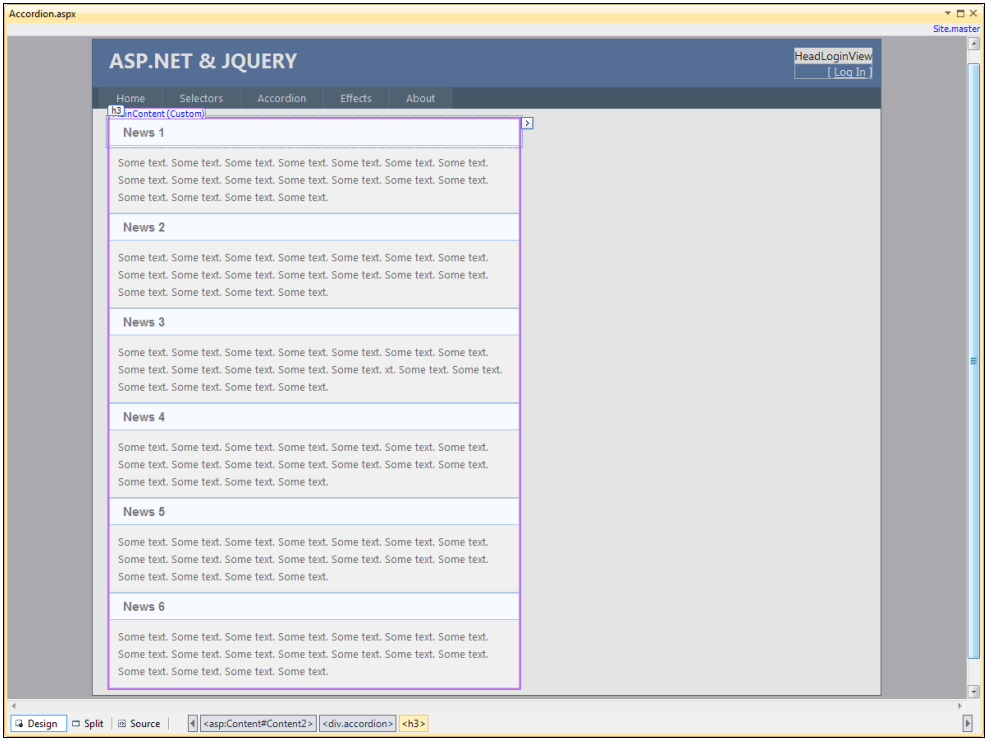


Рис. 15.8. Внешний вид страницы в дизайнере веб-форм

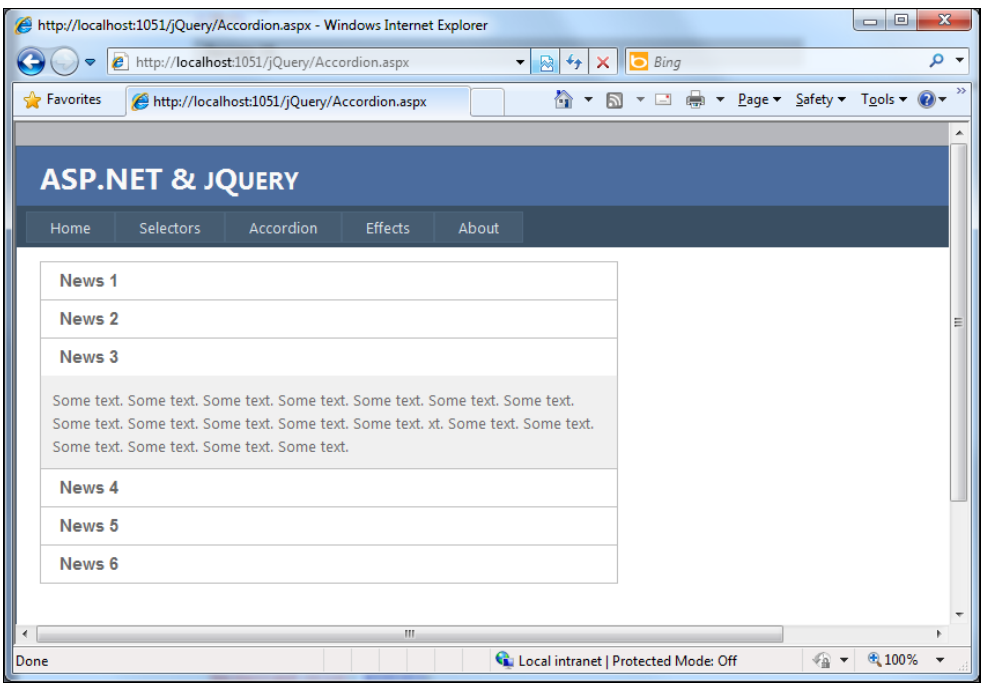
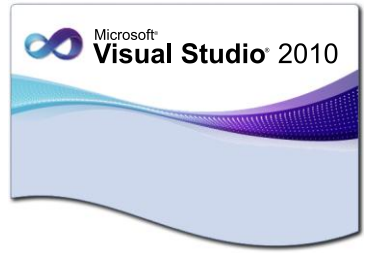


Рис. 15.9. Внешний вид страницы с лентой новостей в браузере

ГЛАВА 16



ASP.NET Dynamic Data

ASP.NET Dynamic Data — платформа, позволяющая создавать веб-приложения ASP.NET, предназначенные для работы с данными. ASP.NET Dynamic Data способна автоматически определять метаданные модели данных во время выполнения и создавать на основе этой модели пользовательский интерфейс. Dynamic Data формирует шаблоны, которые составляют основу веб-приложения для просмотра и изменения данных.

Эти шаблоны, в зависимости от требований к веб-приложению, можно настраивать, изменяя их или создавая новые. Также шаблоны, создаваемые с помощью платформы Dynamic Data, можно встраивать в уже существующие веб-приложения. Платформа ASP.NET Dynamic Data также поддерживает формирование шаблонов, предоставляющее способ автоматического создания веб-страниц для каждой таблицы в базе данных.

Архитектура платформы Dynamic Data

На рис. 16.1 показана архитектура платформы ASP.NET Dynamic Data.

Как видно из рисунка, архитектуру платформы Dynamic Data составляют приведенные далее уровни.

- *Уровень представления данных* — содержит элементы, которые используются для создания пользовательского интерфейса в целях отображения и правки сущностей данных.
- *Уровень управления данными* — содержит функциональность для редактирования существующих данных, вставки новых данных и удаления данных.
- *Уровень сопоставления источников данных* — содержит элементы, являющиеся частью среды CLR, но используемые платформой Dynamic Data. Эти элементы технологии, такие как LINQ to SQL и платформа ADO.NET Entity Framework, которые используются для создания моделей данных.
- *Уровень данных (хранилище)* — содержит модель данных, представляющую сущности базы данных как типы среды CLR.

Платформа Dynamic Data получает сведения схемы данных или метаданные из модели данных, представляющие сущности базы данных как типы среды CLR.

Подробнее элементы уровня представления данных мы рассмотрим при создании проекта ASP.NET Dynamic Data в следующем разделе.

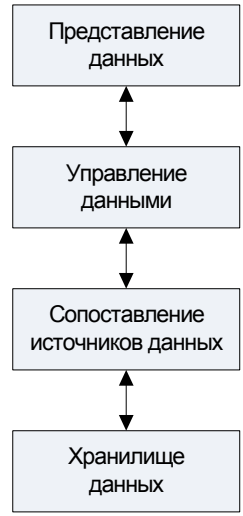


Рис. 16.1. Архитектура платформы ASP.NET Dynamic Data

Создание веб-сайта с использованием ASP.NET Dynamic Data

При создании нового проекта веб-сайта с использованием платформы Dynamic Data в Visual Studio 2010 нужно выбрать шаблон **ASP.NET Dynamic Data Entities Web Site** (при использовании ADO.NET Entity Data Model) или **ASP.NET Dynamic Data Linq to SQL Web Site**. Выбранный шаблон определяет тип модели, используемой в проекте: модель Entity Framework или модель LINQ to SQL. Механизм формирования шаблонов платформы Dynamic Data способен поддерживать только один из типов моделей данных в одном и том же проекте.

В *главах 9* и *10* мы уже создавали библиотеки для доступа к данным с использованием технологий Linq to SQL и Entity Framework, таким образом, у нас уже есть реализованные уровни сопоставления источников данных и уровень данных. Для нашего примера мы используем библиотеку Entity Framework для базы данных `College` из *главы 10* и выберем шаблон проекта **ASP.NET Dynamic Data Entities Web Site** (рис. 16.2).

Среда Visual Studio создаст папку и структуру веб-сайта (рис. 16.3).

Рассмотрим структуру созданного шаблона веб-сайта подробнее. Файлы Default.aspx, Global.asax, Site.css, Site.master, Web.config вам уже известны по предыдущим главам книги. Поэтому наибольший интерес для нас будет представлять каталог DynamicData. Этот каталог содержит подкаталоги для пользовательских элементов управления и для страниц, отображающих данные:

- Content — содержит каталог Images и файл GridViewPager.ascx. Каталог Images содержит графические файлы, используемые в качестве значков в элементе управления страничного навигатора. Файл GridViewPager.ascx представляет собой пользовательский элемент управления, производный от класса `WebControl`. Он используется для расширения возможностей страничного просмотра при на-

личии более одной страницы данных в таблице. Пользовательский элемент управления GridViewPager.ascx также используется в шаблонах страниц List.aspx и ListDetails.aspx.

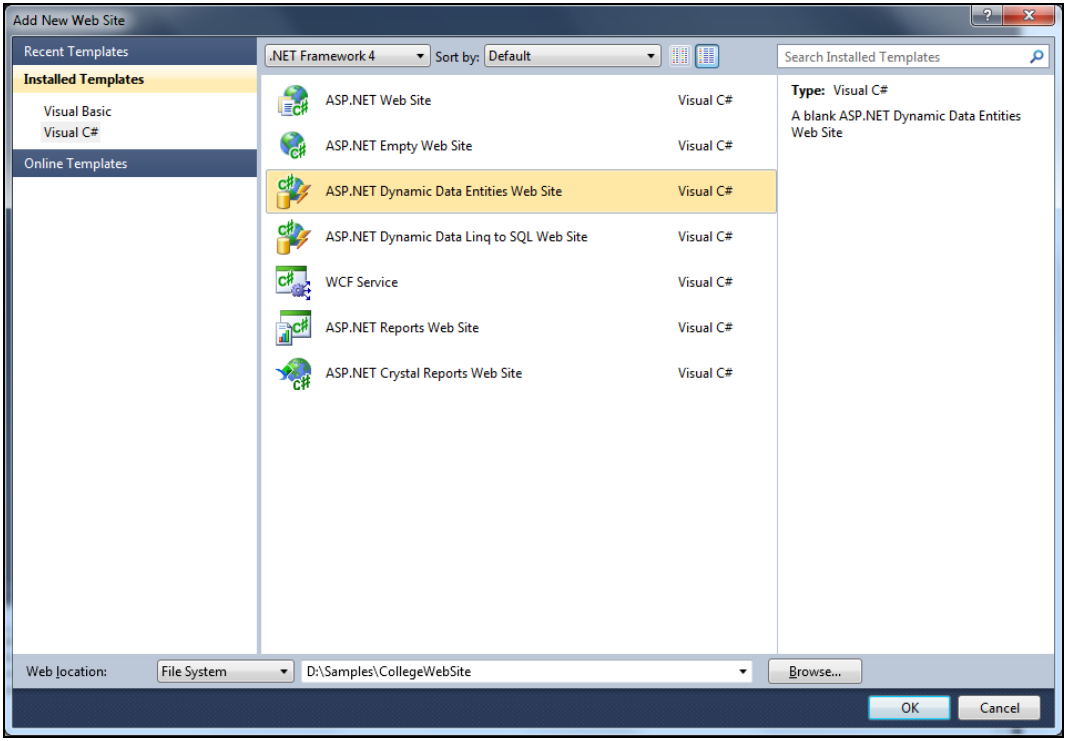


Рис. 16.2. Создание нового проекта ASP.NET Dynamic Data

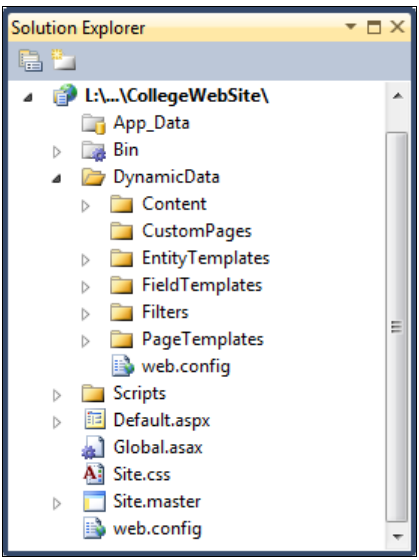


Рис. 16.3. Структура проекта

- PageTemplates — содержит шаблоны страниц, которые генерируют пользовательский интерфейс для просмотра и редактирования данных.
- CustomPages — содержит пользовательские шаблоны страниц. Бывают ситуации, когда стандартные шаблоны страниц не могут обеспечить достаточную функциональность для работы с данными. Пользовательские шаблоны страниц используются для переопределения шаблонов страниц, определенных в папке PageTemplates.
- EntityTemplates — содержит шаблоны сущностей, которые генерируют табличный интерфейс для просмотра и редактирования данных.
- FieldTemplates — содержит пользовательские элементы управления, которые генерируют пользовательский интерфейс для просмотра и редактирования полей данных.
- FilterTemplates — содержит пользовательские элементы управления, которые генерируют пользовательский интерфейс для фильтрации строк данных.

Элементы уровня представления данных можно разделить по своему функциональному назначению на несколько групп:

- шаблоны страниц;
- шаблоны сущностей;
- шаблоны полей;
- шаблоны фильтров.

Рассмотрим теперь эти группы более подробно.

Шаблоны страниц

Шаблоны страниц располагаются в каталоге PageTemplates — это веб-страницы, отображающие данные из любой таблицы базы данных. Платформа ASP.NET Dynamic Data предусматривает шаблоны страниц для создания различных представлений данных:

- представление `List` — для вывода данных в табличном виде;
- представление `Details` — для вывода одной записи из таблицы;
- представление `Edit` — для редактирования одной записи из таблицы.

По умолчанию платформа ASP.NET Dynamic Data использует только шаблон страниц с представлением списка (т. е. представление `List`). При необходимости разработчик может добавить представления `Details` и `Edit`, изменить шаблоны по умолчанию, а также создать новые шаблоны для задания нестандартного представления данных.

Шаблоны сущностей

Каталог EntityTemplates содержит шаблоны сущностей по умолчанию. С помощью шаблонов сущностей можно настроить пользовательский интерфейс для всех сущностей базы данных.

Шаблоны сущностей можно создавать для операций выборки, модификации и вставки данных. Шаблоны сущностей по умолчанию отображают данные в двух столбцах с использованием метки для имени поля и соответствующего элемента управления для значения поля. Разработчик может изменять шаблоны сущностей по умолчанию, чтобы изменить внешний вид и поведение строк данных для всего сайта.

Шаблоны сущностей включаются в шаблоны страниц Details.aspx, Insert.aspx и Update.aspx, а также в шаблоны любых пользовательских страниц CustomPages. В свою очередь шаблоны сущностей используют шаблоны полей для отображения фактических данных.

Шаблоны полей

Шаблоны полей по умолчанию располагаются в каталоге FieldTemplates. Шаблоны полей представляют собой пользовательские элементы управления, которые отображают пользовательский интерфейс для отдельных полей данных. По умолчанию платформа Dynamic Data выбирает шаблон поля на основе типа данных отображаемого поля из базы данных. Например, для отображения целочисленных данных платформа Dynamic Data использует шаблон целочисленных полей, для отображения текстовых данных — шаблон текстовых полей и т. д.

При необходимости программист может изменить шаблоны полей по умолчанию или создать новые шаблоны для задания способа отображения определенных специфических полей данных.

Шаблоны фильтров

Шаблоны фильтров располагаются в каталоге FilterTemplates. Шаблоны фильтров являются пользовательскими элементами управления, которые отображают пользовательский интерфейс для фильтрации данных, и предоставляют пользователю возможность выбирать строки таблицы для отображения на основе значения столбца. Платформа Dynamic Data предоставляет несколько типов шаблонов фильтров:

- логические шаблоны;
- шаблоны внешнего ключа;
- шаблоны фильтрации перечисления.

Платформа Dynamic Data также позволяет использовать разметку веб-страницы для создания пользовательского интерфейса для фильтрации данных в столбце. Если шаблон фильтра для этого типа столбца не существует, то можно создать пользовательский тип и указать, что платформа Dynamic Data должна его использовать при создании пользовательского интерфейса.

Элементы уровня данных

Как уже говорилось ранее, платформа ASP.NET Dynamic Data поддерживает модели данных LINQ to SQL и ADO.NET Entity Framework. Платформа Dynamic

Data использует эти типы для отправки запросов в базу данных, а также для создания, чтения, обновления и удаления данных. Модели данных обеспечивают простой способ интеграции пользовательской проверки данных и правил бизнес-логики.

Во время выполнения платформа Dynamic Data автоматически извлекает из модели данных метаданные, такие как свойства данных. По этим сведениям она определяет, как создавать пользовательский интерфейс для отображения и редактирования данных.

Платформа ASP.NET Dynamic Data использует метаданные модели для автоматического определения того, какие шаблоны полей использовать для отрисовки пользовательского интерфейса отображения и изменения полей данных.

Платформа Dynamic Data позволяет настраивать пользовательский интерфейс для отображения и редактирования полей данных. Возможны следующие способы настройки пользовательского интерфейса:

- добавление к полям данных пользовательских метаданных;
- добавление к типам модели данных типов, не являющихся встроенными типами данных.

Во время выполнения платформа Dynamic Data автоматически извлекает сведения о модели данных. По этим сведениям она определяет, как создавать пользовательский интерфейс для отображения и редактирования данных. Для отрисовки пользовательского интерфейса используются шаблоны полей.

Платформа Dynamic Data также может использовать информацию из базы данных о связях между таблицами модели данных для отображения столбцов внешнего ключа и обеспечения переходов между таблицами.

Регистрация модели данных

После создания шаблона проекта Dynamic Data следующим этапом разработки является регистрация модели данных для использования платформы Dynamic Data. Для регистрации модели данных откройте файл Global.asax и измените в теле метода `RegisterRoutes()` вызов метода `DefaultModel.RegisterContext()` так, как приведено в листинге 16.1.

Листинг 16.1. Код метода `RegisterRoutes()`

```
public static void RegisterRoutes(RouteCollection routes) {
. . .
DefaultModel.RegisterContext(
        typeof(College.Data.CollegeEntities),
        new ContextConfiguration() { ScaffoldAllTables = true });
. . .
}
```

Метод `RegisterContext()` регистрирует экземпляр контекста данных, используя заданную конфигурацию контекста. Этот код означает, что будет зарегистрирован

контекст данных Entity Framework, созданный для нашей базы данных `College` (находящийся во внешней библиотеке College.Data). Этот контекст данных будет использоваться платформой ASP.NET Dynamic Data, которая автоматически сформирует шаблоны модели данных для базы данных `College`.

Скомпилируйте и запустите веб-сайт в браузере. На странице отобразится список таблиц, добавленных в модель данных (рис. 16.4).

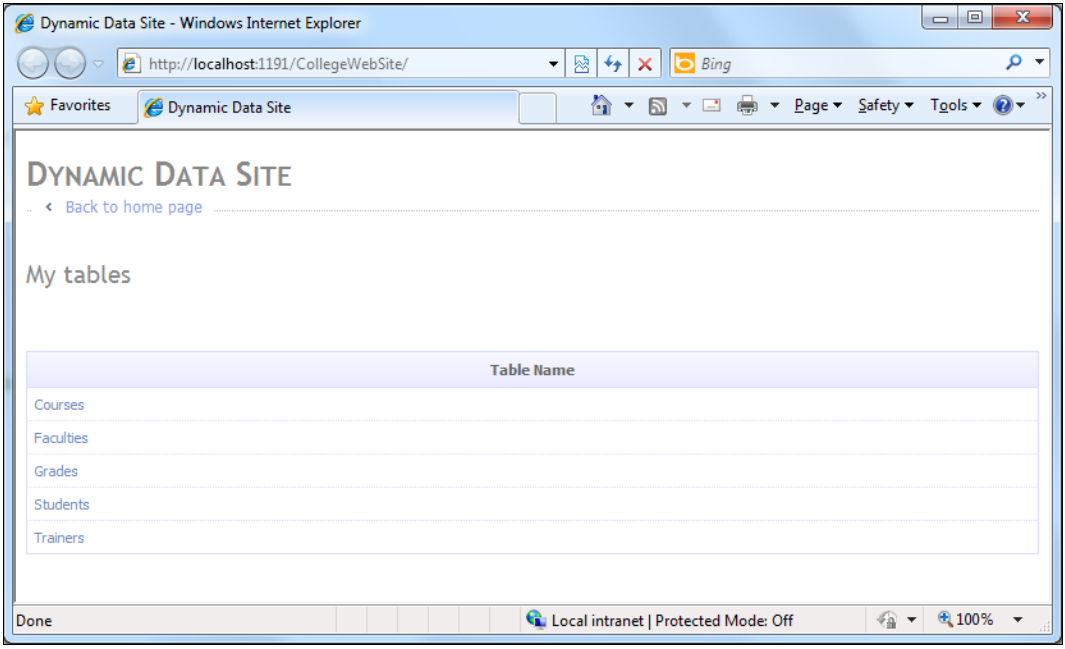


Рис. 16.4. Домашняя страница приложения

Выберите одну из таблиц. Например, с информацией о преподавателях (`Trainers`). Появится страница (представление типа `List`), содержащая данные из таблицы `Trainers`, внешний вид которой показан на рис. 16.5.

Для таблиц, содержащих поля внешних ключей, приводится ссылка на страницу сведений о связанной таблице. Если таблица является родительской таблицей в связи "один-ко-многим", приводится ссылка на страницу списка дочерней таблицы. В нашем случае каждый преподаватель может вести один или несколько курсов, поэтому в таблице присутствует ссылка на страницу **View Courses** (см. рис. 16.5).

В левой части страницы расположены ссылки **Edit**, **Delete**, **Details**. При выборе пользователем опции **Details** откроется детализированное представление для выбранной записи в режиме только для чтения данных, без возможности их модификации (рис. 16.6).

Если нажать ссылку **Edit** для изменения записи на странице `Trainers`, откроется представление `Edit`, как показано на рис. 16.7.

Аналогично выглядит веб-страница с представлением для создания новой записи, которая показана на рис. 16.8. Для его открытия служит ссылка **Insert new item** в нижней левой части представления `List` (см. рис. 16.5).

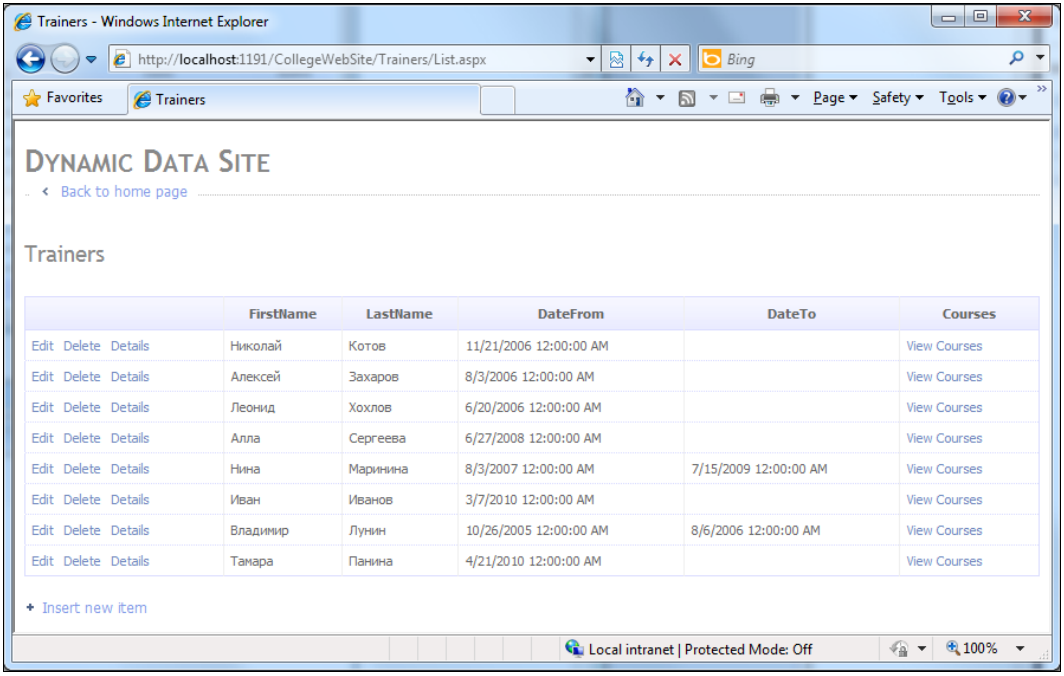


Рис. 16.5. Данные о преподавателях

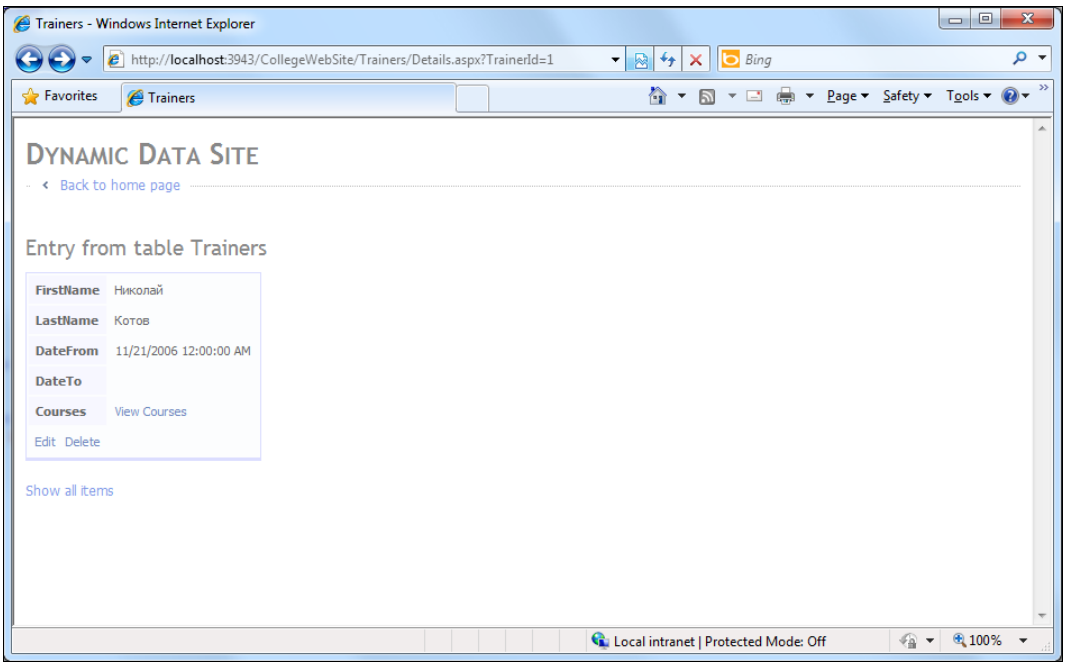


Рис. 16.6. Страница для отображения одной записи

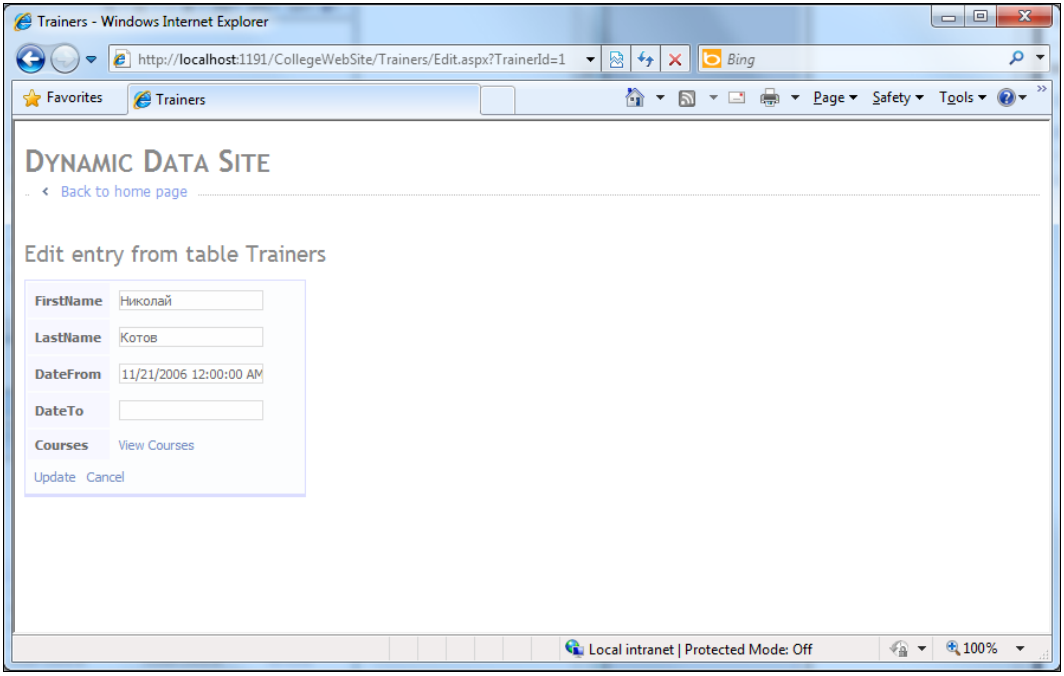


Рис. 16.7. Страница редактирования записи

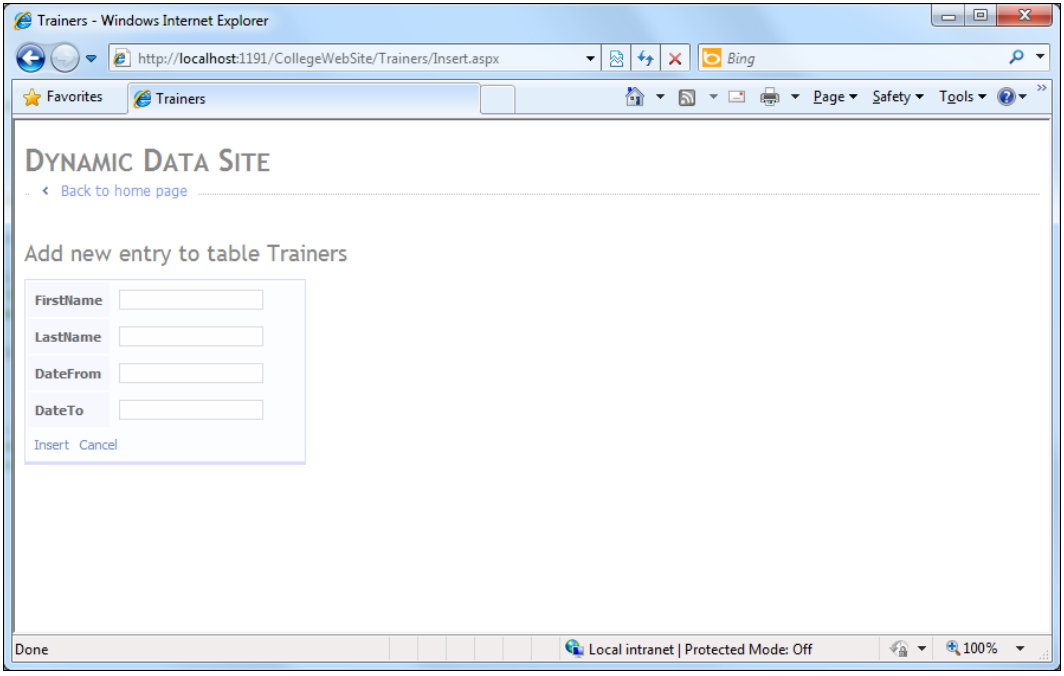
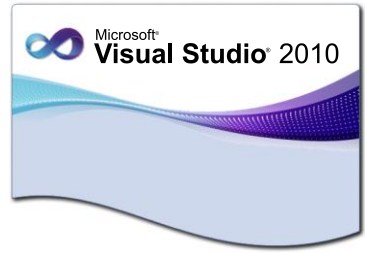


Рис. 16.8. Страница добавления записи

Резюме

Платформа ASP.NET Dynamic Data позволяет быстро строить веб-приложения для работы с данными. При использовании ASP.NET Dynamic Data количество функциональных возможностей, которые можно получить, написав несколько строк кода, несоизмеримо по усилиям, затрачиваемым при ручной привязке данных к элементам отображения данных, используемых при создании классических технологий ASP.NET.



ГЛАВА 17

ASP.NET MVC

Model-View-Controller (MVC) — это архитектурный принцип, согласно которому веб-приложение делится на компоненты. Разделение веб-приложения на компоненты упрощает его разработку, тестирование и сопровождение. Платформа ASP.NET MVC представляет собой альтернативное направление для разработки веб-приложений, но в то же время не является полной заменой программированию с помощью веб-форм ASP.NET.

Архитектура MVC

В состав платформы MVC входят три компонента.

- *Модель* — реализует логику работы для домена данных приложения. Объекты моделей получают и сохраняют состояние модели в базе данных.
- *Представление* — отображает пользовательский интерфейс приложения. Пользовательский интерфейс обычно создается на основе данных модели.
- *Контроллер* — осуществляет взаимодействие с пользователем, работу с моделью, а также выбор представления, отображающего пользовательский интерфейс. Контроллер обрабатывает вводимые данные и отвечает на действия пользователя.

Архитектура Model-View-Controller позволяет создавать приложения, в которых логика ввода, бизнес-логика и логика интерфейса разделены, но достаточно тесно взаимодействуют друг с другом. Пользовательский интерфейс располагается в представлении, логика ввода — в контроллере. Бизнес-логика находится в модели.

Связь между тремя основными компонентами приложения Model-View-Controller также облегчает параллельную разработку веб-приложения в команде. Например, один разработчик может создавать представление, другой — логику контроллера, а третий — бизнес-логику модели.

Платформа ASP.NET MVC сопоставляет URL-адреса с кодом сервера способом, который несколько отличается от сопоставления страниц URL-адресов веб-форм

ASP.NET. Вместо сопоставления URL-адресов со страницами или обработчиками платформа ASP.NET MVC сопоставляет их с классами контроллеров. Классы контроллера обрабатывают входящие запросы, например ввод информации пользователем и его действия, а также реализуют соответствующую логику приложений и данных на основании введенной информации. Класс контроллера обычно вызывает отдельное представление, которое генерирует веб-страницу.

В платформе ASP. NET MVC компоненты модели, представления и контроллера разделены. Модель представляет бизнес-логику или логику домена приложения для работы с данными. Представление выбирается контроллером и создает соответствующий пользовательский интерфейс.

При разработке приложений на платформе MVC используются обычные веб-страницы, мастер-страницы и пользовательские элементы управления. Контроллер выполняет поиск соответствующего метода действия, получает значения для использования в качестве аргументов метода, а затем обрабатывает ошибки, возникающие при выполнении метода. После этого выполняется отображение требуемого представления. По умолчанию наборы компонентов хранятся в отдельных папках проекта веб-приложения MVC.

Одно из отличий платформы MVC от веб-форм состоит в том, что платформа ASP.NET MVC не использует модель обратной передачи веб-форм ASP.NET для взаимодействия с сервером. Все взаимодействия с пользователем передаются в класс контроллера. Это обеспечивает отдельное создание логики пользовательского интерфейса и бизнес-логики и, кроме того, облегчает тестирование веб-приложения. Таким образом, события состояния просмотра ASP.NET и жизненного цикла веб-страниц ASP.NET не связаны с представлениями на основе MVC.

Создание приложения MVC

В состав платформы ASP.NET MVC входит шаблон проекта Visual Studio, который позволяет создавать веб-приложения с соответствующей шаблону MVC структурой. Этот шаблон создает новое веб-приложение MVC, конфигурация которого предусматривает все необходимые папки, шаблоны элементов и записи файла конфигурации.

Шаблоны проекта веб-приложения ASP.NET MVC основаны на шаблоне проекта веб-приложения ASP.NET. Новый проект ASP.NET MVC создается путем выбора пункта **New Project** из меню **File** (рис. 17.1).

При создании нового веб-приложения MVC Visual Studio предоставляет возможность создания двух проектов одновременно. Первый из них является веб-проектом, в котором реализуется само приложение MVC. Второй проект представляет собой проект модульного теста, в котором возможно создание модульных тестов для компонентов MVC первого проекта (рис. 17.2).

При создании проекта веб-приложения ASP.NET MVC среда Visual Studio создает проект, в котором компоненты MVC внутри каталога проекта разделяются по специальным подкаталогам. Этот проект представляет собой готовый шаблон веб-

приложения с базовой функциональностью, необходимой для стандартного веб-сайта. Структура шаблона проекта веб-приложения на ASP.NET MVC 2 Web Application представлена на рис. 17.3.

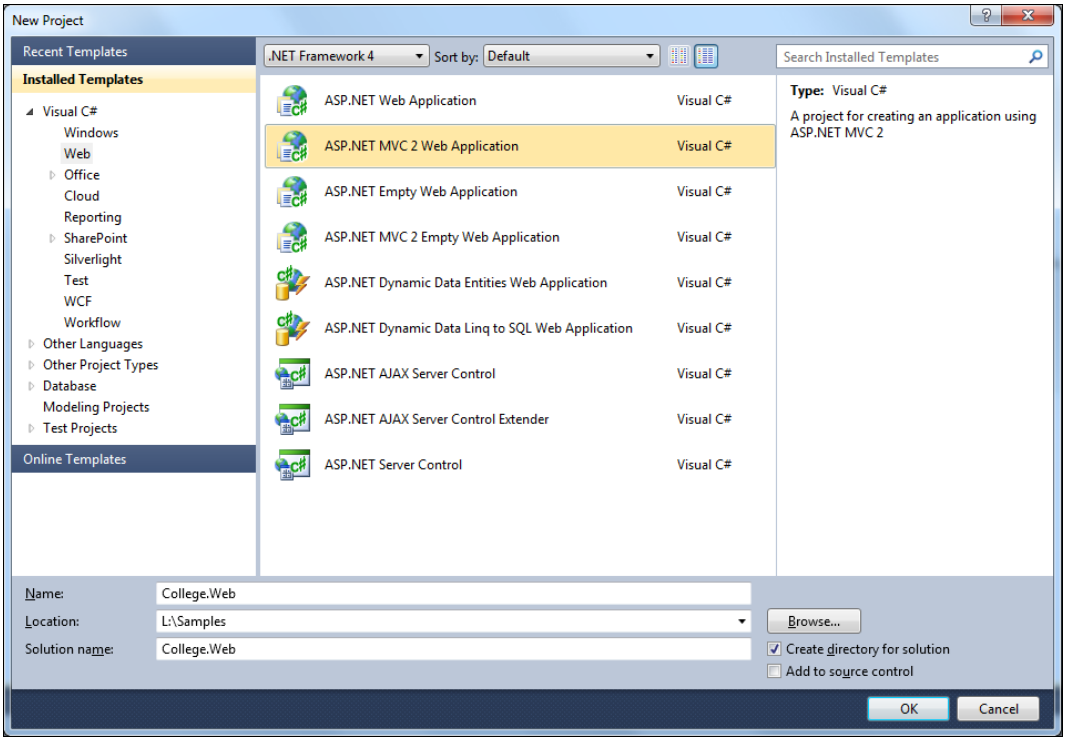


Рис. 17.1. Создание нового проекта **ASP.NET MVC 2 Web Application**

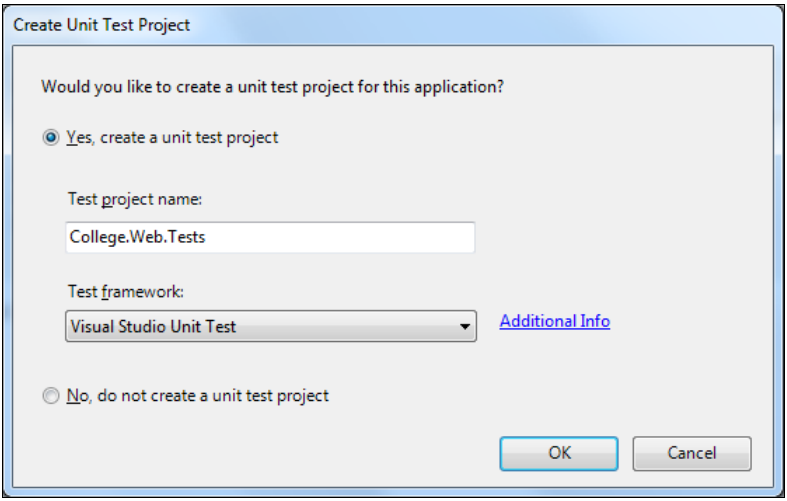


Рис. 17.2. Окно **Create Unit Test Project**

Как видно из рис. 17.3, в шаблоне присутствуют специфические каталоги, которых нет при создании классического проекта на ASP.NET. Единственное исключение — это каталог App_Data, который является физическим хранилищем данных. Этот каталог выполняет те же функции, что и для веб-сайтов ASP.NET.

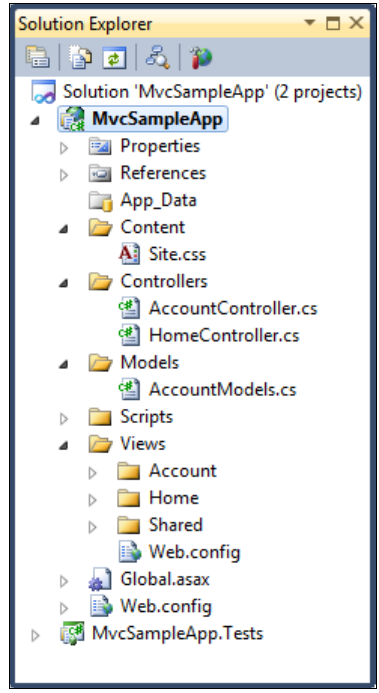


Рис. 17.3. Структура проекта приложения MVC

Остальные каталоги имеют следующее назначение.

- Content — предназначен для вспомогательных файлов содержимого. Эта папка содержит каскадную таблицу стилей (CSS-файл) приложения. В общем случае папка Content предназначена для статических файлов.
- Controllers — предназначен для файлов контроллера. В этом каталоге расположены образцы контроллеров приложения с именами `AccountController` и `HomeController`. Класс `AccountController` содержит реализацию входа в приложение. Класс `HomeController` содержит реализацию запуска приложения. Имена контроллеров в платформе MVC должны иметь суффикс `Controller`, например, в случае с нашей базой данных: `StudentController`, `FacultyController` или `ProductController`.
- Models — предназначен для файлов модели данных, например файлов преобразования из LINQ в SQL, т. е. DBML-файлов, или файлов сущностей данных. Этот каталог обычно содержит код, который определяет объекты и логику взаимодействия с хранилищем данных. Сами объекты модели обычно располагаются в отдельных библиотеках классов.
- Scripts — предназначен для файлов скриптов, поддерживающих приложение. Эта папка по умолчанию содержит файлы платформы ASP.NET AJAX и библиотеку jQuery.
- Views — предназначен для файлов страниц представления. Каталог Views содержит три подкаталога:
  - Account — здесь находятся представления, используемые в качестве пользовательского интерфейса при входе и смене пароля;

- Home — здесь находятся представление `Index`, являющееся стартовой страницей приложения по умолчанию, и представление страницы `About`.
- Shared — предназначен для представлений, которые могут использоваться в различных контроллерах. Например, мастер-страница веб-приложения по умолчанию расположена в папке Shared.

В каталоге Views в процессе разработки помещают подкаталоги для всех создаваемых контроллеров. Название подкаталога должно состоять из префикса имени контроллера. Например, если существует контроллер с именем `HomeController`, то в каталоге Views будет вложенный каталог с именем Home. При загрузке представления средой выполнения ASP.NET в подкаталоге Views*имя_контроллера* по умолчанию выполняется поиск файла ASPX, который имеет имя требуемого представления.

Новый проект MVC, созданный средой Visual Studio, — это полноценное приложение, которое можно скомпилировать и запустить без изменений. Внешний вид этого приложения в браузере показан на рис. 17.4.

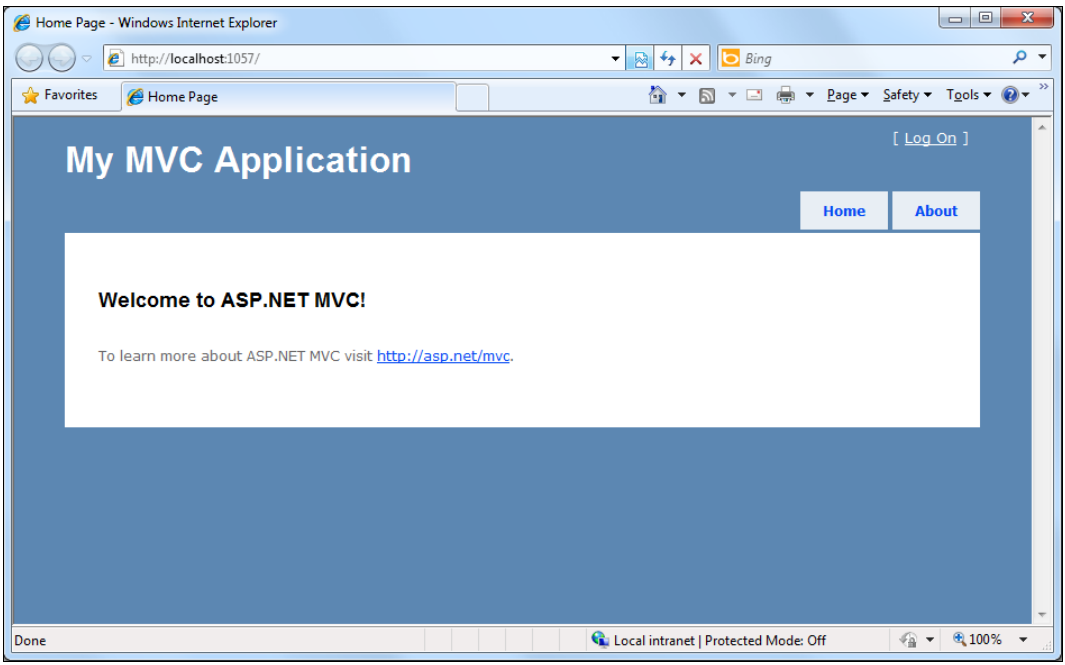


Рис. 17.4. Шаблон приложения MVC в браузере

Выполнение запросов в MVC

Запросы к веб-приложениям ASP.NET, созданным на платформе MVC, сначала проходят через объект `UrlRoutingModule`, который представляет собой HTTP-модуль. Объект `UrlRoutingModule` анализирует запрос и выбирает маршрут. Объект

UrlRoutingModule выбирает первый объект маршрута, соответствующий текущему запросу. Объект UrlRoutingModule — это класс, реализующий тип RouteBase, который является базовым для всех классов, представляющих маршрут ASP.NET. В том случае, если подходящих маршрутов нет, объект UrlRoutingModule не производит никаких действий и передает запрос обратно на обычную обработку средствами ASP.NET или IIS.

В выбранном объекте Route объект UrlRoutingModule получает объект, реализующий интерфейс IRouteHandler и связанный с объектом Route. Обычно в MVC-приложении это экземпляр класса MvcRouteHandler. Экземпляр MvcRouteHandler создает объект MvcHandler, реализующий интерфейс IHttpHandler. Объект MvcHandler затем выбирает контроллер, который и будет обрабатывать запрос.

Классы UrlRoutingModule и MvcRouteHandler служат входными точками для платформы MVC в ASP.NET. Они отвечают за следующее:

- выбор нужного контроллера в веб-приложении MVC;
- получение отдельного экземпляра контроллера;
- вызов метода Execute контроллера.

Рассмотрим теперь выполнение запроса более подробно. Этапы выполнения запроса в MVC наглядно представлены на рис. 17.5.

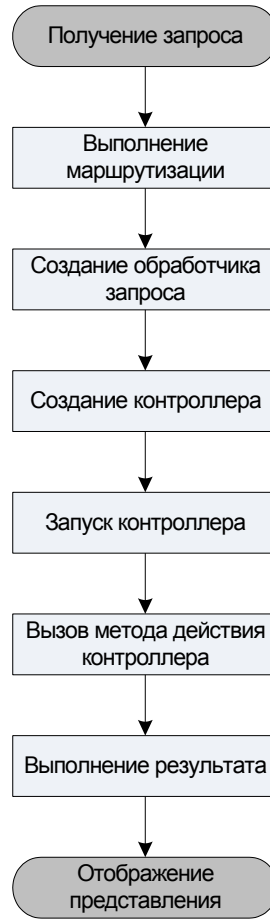


Рис. 17.5. Этапы выполнения запроса в MVC

Во время выполнения запроса в приложении MVC происходит следующая последовательность действий:

1. Получение первого запроса к приложению. При этом в файле Global.asax объекты Route добавляются к объекту RouteTable.
2. Выполнение маршрутизации. Модуль UrlRoutingModule использует первый подходящий объект Route в коллекции RouteTable для создания объекта RouteData. Объект RouteData далее используется для создания объекта RequestContext.
3. Создание обработчика запроса MVC. Объект MvcRouteHandler создает экземпляр класса MvcHandler и передает этому обработчику экземпляр RequestContext.

4. Создание контроллера. Объект `MvcHandler` использует экземпляр `RequestContext` для идентификации объекта типа `IControllerFactory`, в котором определены методы, требующиеся для создания контроллера Обычно это экземпляр класса `DefaultControllerFactory`, с помощью которого будет создан экземпляр контроллера.
5. Запуск контроллера. Экземпляр класса `MvcHandler` вызывает метод `Execute` контроллера.
6. Вызов действия. Объект типа `ControllerActionInvoker` (ответственный за вызов методов действия контроллера), связанный с данным контроллером, определяет, какой метод действия контроллера нужно вызвать, и вызывает его.
7. Выполнение результата. Метод действия получает введенные пользователем данные, готовит соответствующие данные ответа и выполняет результат, возвращая тип результата. Возвращаемые типы результата выполнения могут быть следующие:
   - `ViewResult`;
   - `RedirectToRouteResult`;
   - `RedirectResult`;
   - `ContentResult`;
   - `JsonResult`;
   - `FileResult`;
   - `EmptyResult`.

Маршрутизация URL-адресов

Платформа ASP.NET MVC использует механизм маршрутизации ASP.NET, который обеспечивает гибкость при сопоставлении URL-адресов с классами контроллера. Пользователь также может определять правила маршрутизации, которые используются платформой ASP.NET MVC для оценки входящих URL-адресов и выбора соответствующего контроллера. Пользователь также может настроить механизм маршрутизации на автоматический анализ переменных, определенных в URL-адресе, с последующей передачей значений в качестве аргументов параметра из платформы ASP.NET MVC в контроллер.

Приложение MVC использует код в файле Global.asax для установки глобальных параметров маршрутизации URL-адресов по умолчанию, а также использует файл Web.config для настройки приложения.

Маршруты URL-адресов инициализируются в методе `Application_Start()` файла Global.asax.cs. В листинге 17.1 показан файл Global.asax.cs, в котором с помощью метода `MapRoute()` класса `RouteCollection` реализована логика маршрутизации по умолчанию.

Примечание

Полный код примера MVC веб-сайта находится в каталоге Chapter17\College.web на прилагаемом к книге диске.

Листинг 17.1. Код файла Global.asax.cs с логикой маршрутизации по умолчанию

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace College.web
{
   public class MvcApplication : System.Web.HttpApplication
   {
      public static void RegisterRoutes(RouteCollection routes)
      {
         routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

         // Логика маршрутизации по умолчанию
         routes.MapRoute(
             "Default",
             "{controller}/{action}/{id}",
             new {
                controller = "Home",
                action = "Index",
                id = UrlParameter.Optional }
         );

      }

      protected void Application_Start()
      {
         AreaRegistration.RegisterAllAreas();

         RegisterRoutes(RouteTable.Routes);
      }
   }
}
```

Контроллеры и методы действий

Платформа MVC для ASP.NET сопоставляет URL-адреса с контроллерами. Контроллеры обрабатывают входящие запросы, например, вводимые пользователями данные и их действия, а также реализуют необходимую логику работы приложения. В классе контроллера происходит вызов представления, которое генерирует HTML-разметку веб-страницы, отображающуюся в браузере.

Код класса контроллера `HomeController`, созданный по умолчанию средой Visual Studio, представлен в листинге 17.2.

Листинг 17.2. Код класса контроллера `HomeController`

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

namespace College.web.Controllers
{
   [HandleError]
   public class HomeController : Controller
   {
      public ActionResult Index()
      {
         ViewData["Message"] =
           "Добро пожаловать на сайт технологического колледжа!";
         return View();
      }

      public ActionResult About()
      {
         return View();
      }

      public ActionResult Registration()
      {
         return View();
      }
   }
}
```

Базовый класс для всех контроллеров в MVC — это класс `ControllerBase`, реализующий общую обработку MVC. Класс `Controller` наследует от класса `ControllerBase` и является реализацией контроллера по умолчанию. Класс `Controller` отвечает за такие этапы обработки, как:

- поиск метода действия, который необходимо вызвать, и проверка допустимости его вызова;
- получение значений, используемых как аргументы метода действия;
- обработка всех ошибок, которые могут возникнуть при выполнении метода действия;
- предоставление используемого по умолчанию класса `WebFormViewEngine` для отображения различных типов страниц ASP.NET (представлений).

В имени всех классов контроллеров, которые будут добавляться в проект MVC, должен быть суффикс `Controller`. В следующем примере показан образец класса контроллера с именем `HomeController`. Этот класс содержит методы действий, отображающие страницы представлений.

Методы действий

В приложениях ASP.NET, не использующих платформу MVC, взаимодействие с пользователем организуется на основе страниц, создания и обработки событий на странице и элементов управления на странице. Напротив, взаимодействие с пользователем в MVC-приложениях ASP.NET основано на контроллерах и методах действий. Методы действий определяются в контроллере. Контроллеры могут содержать любое число методов действий.

Методы действий обычно однозначно сопоставляются с различными формами взаимодействия с пользователем. Примером может служить ввод URL-адреса в браузере, щелчок ссылки или отправка формы. Любое из таких действий приводит к отправке запроса на сервер. В каждом случае URL-адрес запроса включает сведения, которые платформа MVC использует для вызова нужного метода действия.

Когда пользователь вводит в браузере URL-адрес, MVC-приложение использует правила маршрутизации, определенные в файле Global.asax, для анализа URL-адреса и определения пути контроллера. Затем контроллер определяет, какой метод действия нужен для обработки запроса. По умолчанию URL-адрес запроса рассматривается как подпуть, включающий имя контроллера, за которым идет имя действия.

Возвращаемый тип *ActionResult*

Большинство методов действий возвращают экземпляр класса-наследника `ActionResult`. Класс `ActionResult` является базовым для всех результатов действий. Однако существуют различные типы результатов действий, зависящие от задач, выполняемых методом действия. Например, наиболее часто используемое действие — это вызов метода `View()`. Метод `View()` возвращает экземпляр класса `ViewResult`, который является наследником `ActionResult`.

Можно создавать методы действий, возвращающие объекты любых типов, например строки, целые или логические значения. Такие возвращаемые типы обертываются в подходящий тип `ActionResult` перед преобразованием в поток ответа.

Далее приведены встроенные типы результатов действий и возвращающие их вспомогательные методы действий.

- `ViewResult View()` — отображает представление в качестве веб-страницы.
- `PartialViewResult PartialView()` — отображает разделяемое представление, в котором определяется часть представления, которая может отображаться внутри другого представления.
- `RedirectResult Redirect()` — перенаправляет к другому методу действия, используя его URL-адрес.

- `RedirectToRouteResult RedirectToAction()` — перенаправляет к другому методу действия.
- `RedirectToRouteResult RedirectToRoute ()` — перенаправляет к другому методу действия.
- `ContentResult Content()` — возвращает пользовательский тип содержимого.
- `JsonResult Json()` — возвращает сериализованный объект JSON.
- `JavaScriptResult JavaScript()` — возвращает скрипт, который может выполняться на клиенте.
- `FileResult File()` — возвращает двоичные выходные данные, подлежащие записи в ответ.
- `void EmptyResult()` — используется в случае, если метод действия должен вернуть результат `null`.

По умолчанию платформа MVC рассматривает все открытые методы класса контроллера как методы действий. Если в классе контроллера имеется открытый метод, который не следует воспринимать как метод действия, его необходимо пометить атрибутом `[NonAction]`, например, так:

```
[NonAction]
private void NonActionMethod()
{
   ...
}
```

Параметры методов действий

Значения параметров методов действий извлекаются из коллекции данных запроса. Коллекция данных включает пары "имя-значение", содержащее данные форм, значения из строки запросов и значения из файлов Cookies.

Класс контроллера определяет метод действия и все значения параметров для него на основании экземпляра `RouteData` и по данным форм. Если значение параметра не удается проанализировать, и тип параметра является ссылочным или поддерживает значение `null`, то в качестве значения параметра передается `null`. В противном случае генерируется исключение.

В методах действий классов контроллеров существует несколько способов доступа к значениям параметров URL-адресов. Класс `Controller` предоставляет свойства `Request` и `Response`, которые можно использовать в методе действия. Эти свойства аналогичны `HttpRequest` и `HttpResponse`, входящим в состав среды ASP.NET. Однако, в отличие от объектов `HttpRequest` и `HttpResponse`, объекты `Request` и `Response` класса `Controller` принимают объекты, реализующие абстрактные классы `HttpRequestBase` и `HttpResponseBase`, и не являются `sealed`-классами. Эти базовые классы упрощают создание макетов объектов, соответственно, облегчают разработку модульных тестов для классов контроллеров.

Например, так можно использовать объект `Request` для извлечения значения `id` из строки запроса:

```
public void Detail()
{
    int id = Convert.ToInt32(Request["id"]);
}
```

Платформа ASP.NET MVC способна *автоматически* сопоставлять значения параметров URL-адресов со значениями параметров методов действий. По умолчанию в случае, когда метод действия принимает какой-либо параметр, платформа MVC проверяет данные поступившего запроса и определяет, есть ли в запросе значение HTTP-запроса с аналогичным именем. Если есть, то значение запроса автоматически передается в метод действия примерно так:

```
public ActionResult Detail(int id)
{
    ViewData["DetailInfo"] = id;
    return View();
}
```

Значения параметров также можно включать в состав URL-адреса, а не в значения строки запроса. Например, вместо URL-адреса со строкой запроса вида `/Products/Detail?id=3` можно использовать URL-адрес вида `/Products/Detail/3`. Правило сопоставления маршрутизации по умолчанию имеет следующий формат:

```
/{controller}/{action}/{id}
```

При наличии в URL-адресе дополнительного пути после имени контроллера и действия он рассматривается как параметр с именем `id` и автоматически передается методу действия в качестве значения параметра.

Добавление нового контроллера

Теперь используем наш проект для работы с базой данных `College`. Для начала необходимо подключить ссылку на библиотеку `College.Data`, созданную нами в *главе 10* (см. на диске каталог Chapter10\College.Data).

Добавьте также в файл Web.config строку подключения к базе данных, как показано в листинге 17.3.

Листинг 17.3. Добавление строки подключения к БД в файл Web.config

```
<configuration>
  <connectionStrings>
    <add name="CollegeEntities"
      connectionString=
    "metadata=res://*/College.csdl|res://*/College.ssdl|res://*/College.msl;
        provider=System.Data.SqlClient;provider connection string=&amp;quot;
        Data Source=(local);Initial Catalog=College;
        Integrated Security=True;
        MultipleActiveResultSets=True&amp;quot;"
        providerName="System.Data.EntityClient" />
  </connectionStrings>
...
```

Теперь необходимо добавить контроллер, реализующий загрузку курсов из нашей базы данных `College`. Для добавления контроллера в проект MVC в окне **Solution Explorer** щелкните правой кнопкой мыши папку Controllers и в контекстном меню последовательно выберите пункты **Add | Controller**. При этом появится диалоговое окно **Add Controller**, представленное на рис. 17.6.

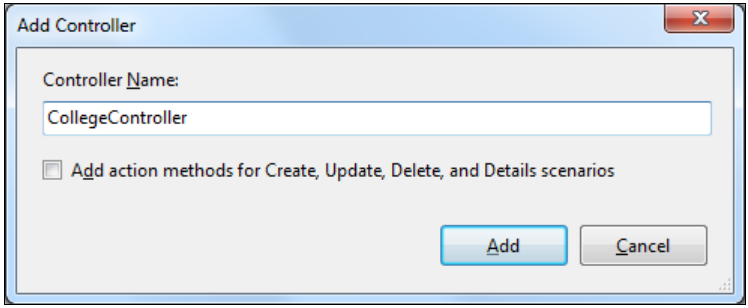


Рис. 17.6. Окно добавления нового контроллера

В поле **Controller Name** введите `CollegeController` и снимите флажок **Add action methods for Create, Update, Delete, and Details scenarios**. Нажмите кнопку **Add**. Среда Visual Studio добавит в проект новый класс контроллера `CollegeController` и откроет его в редакторе кода.

> ***Примечание***
>
> В платформе MVC ASP.NET у имен контроллеров должно быть окончание `Controller`, например `HomeController`, или, в нашем конкретном случае, `CollegeController`.

В классе `CollegeController` замените код контроллера, сгенерированный средой Visual Studio, на код, отображающий представление `Course`, как показано в листинге 17.4.

Листинг 17.4. Код класса `CollegeController`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using College.Data;

namespace College.web.Controllers
{
   [HandleError]
    public class CollegeController : Controller
    {
        // GET: /College/
```

```
        public ActionResult Course()
        {
           CollegeEntities ent = new CollegeEntities();
            return View(ent.Course_Select(null, null, null));
        }

    }
}
```

Добавление представления

Далее нам надо добавить представление `Course`. Для начала создадим новую папку в каталоге `Views` и назовем ее `College`. В этой папке мы будем размещать все представления для работы с базой данных `College`.

Откройте файл CollegeController.cs, щелкните правой кнопкой мыши внутри метода действия `Course()` и выберите в контекстном меню пункт **Add View**. При этом отобразится диалоговое окно **Add View**, представленное на рис. 17.7.

Примечание

Представление можно также создать, щелкнув правой кнопкой мыши на папке College и выбрав пункт контекстного меню **Add View**.

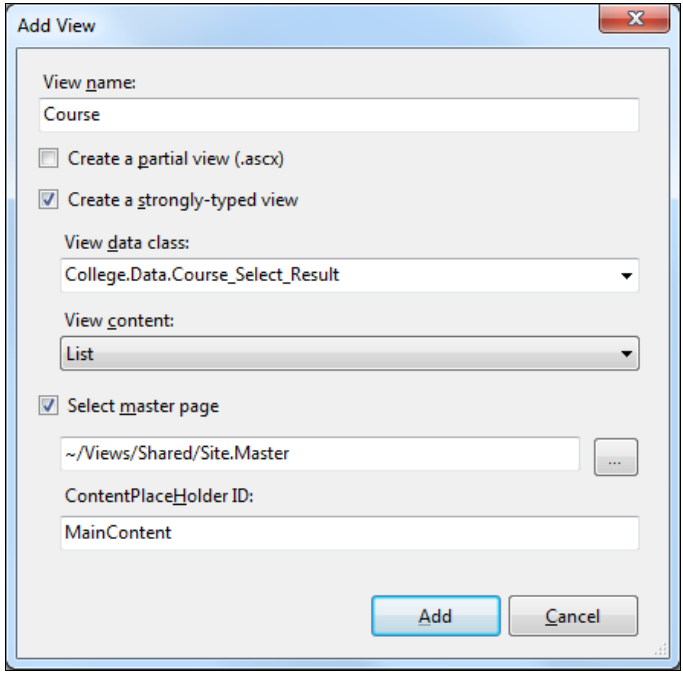


Рис. 17.7. Окно для добавления представления

В поле **View name** введите имя представления: `Course`. Снимите флажок **Create a partial view (.ascx)** и установите флажок **Create a strongly-typed view**. В раскрывающемся списке **View data class** выберите **College.Data.Course_Select_Result** из нашей библиотеки `College.Data`, который возвращает сущность `Course`. В раскрывающемся списке **View content** выберите тип представления данных **List**.

Установите также флажок **Select master page** и укажите мастер-страницу **~/Views/Shared/Site.Master**. В поле **ContentPlaceHolder ID** задайте значение `MainContent` и нажмите кнопку **Add**.

Новое представление с именем Course.aspx будет создано и добавлено в проект в папку College. Поскольку мы создавали строго типизированное представление, среда Visual Studio автоматически добавит весь код в файл разметки Course.aspx. Содержимое файла Course.aspx представлено в листинге 17.5.

Листинг 17.5. Код разметки представления Course.aspx

```
<%@ Page Title="" Language="C#"
 MasterPageFile="~/Views/Shared/Site.Master"
 Inherits=
"System.Web.Mvc.ViewPage<IEnumerable<College.Data.Course_Select_Result>>" %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
  runat="server">
    Course
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
    <h2>Course</h2>
    <table>
        <tr>
            <th></th>
            <th>
                CourseId
            </th>
            <th>
                CourseName
            </th>
            <th>
                TrainerId
            </th>
            <th>
                TrainerFullName
            </th>
            <th>
                FacultyId
            </th>
```

```
            <th>
                FacultyName
            </th>
        </tr>
    <% foreach (var item in Model) { %>
        <tr>
            <td>
                <%: Html.ActionLink("Edit", "Edit", new { /*
                  id=item.PrimaryKey */ }) %> |
                <%: Html.ActionLink("Details", "Details", new { /*
                  id=item.PrimaryKey */ })%> |
                <%: Html.ActionLink("Delete", "Delete", new { /*
                  id=item.PrimaryKey */ })%>
            </td>
            <td>
                <%: item.CourseId %>
            </td>
            <td>
                <%: item.CourseName %>
            </td>
            <td>
                <%: item.TrainerId %>
            </td>
            <td>
                <%: item.TrainerFullName %>
            </td>
            <td>
                <%: item.FacultyId %>
            </td>
            <td>
                <%: item.FacultyName %>
            </td>
        </tr>

    <% } %>
    </table>
    <p>
        <%: Html.ActionLink("Create New", "Create") %>
    </p>
</asp:Content>
```

Теперь надо добавить в меню веб-сайта в мастер-странице элемент, вызывающий метод действия `Course()`.

Для добавления вкладки в меню мастер-страницы используется метод `ActionLink()`. Это вспомогательный метод, создающий ссылку на метод действия.

Он принимает следующие параметры:

- текст ссылки;
- имя метода действия;
- имя контроллера.

В каталоге Shared откройте файл мастер-страницы Site.Master и найдите в нем элемент меню `<ul id="menu">`. Измените код ссылок в этом элементе, заодно переведя названия пунктов меню на русский язык, как показано в листинге 17.6.

Листинг 17.6. Добавление пункта меню в мастер-страницу Site.Master

```
<ul id="menu">
  <li><%: Html.ActionLink("Главная", "Index", "Home")%></li>
  <li><%: Html.ActionLink("Абитуриентам",
      "Registration","Applicants")%></li>
  <li><%: Html.ActionLink("Курсы", "Course", "College")%></li>
  <li><%: Html.ActionLink("Контакты", "About", "Home")%></li>
</ul>
```

Примечание

Представление `Applicants` мы будем разрабатывать позже в этой главе, пока добавьте только ссылку на него.

Теперь запустите наше веб-приложение в браузере. Внешний вид приложения, открытого на странице "Курсы" и отображающего представление `Course`, представлен на рис. 17.8.

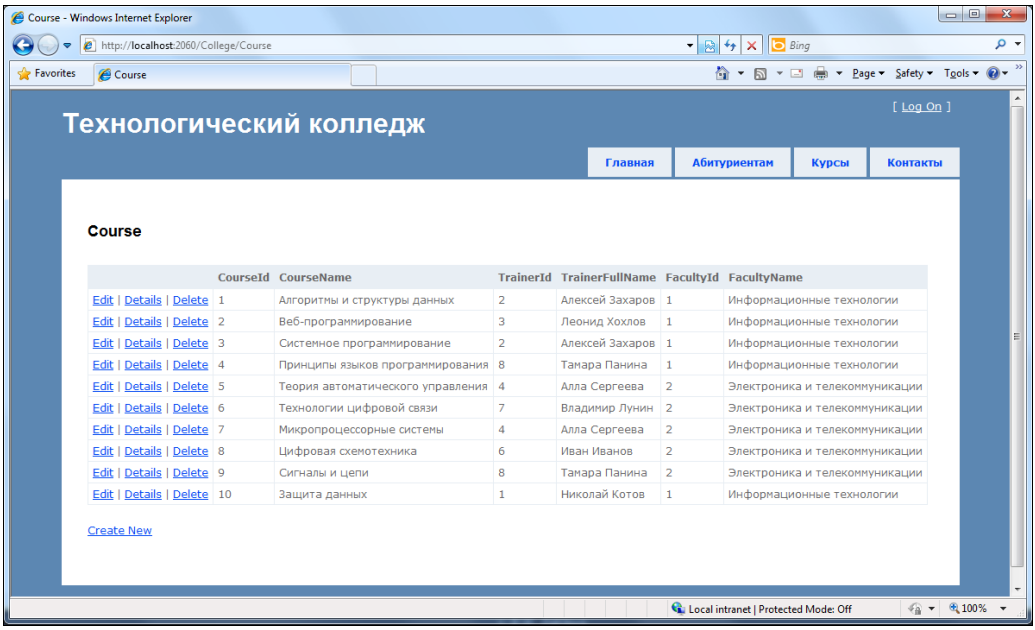


Рис. 17.8. Отображение представления для курсов

Создание и привязка моделей

В предыдущем случае мы использовали готовую модель данных `College`, подключив библиотеку базы данных `College`, созданную на основе Entity Data Model. Иногда требуется создать модель, не связанную с каким-либо хранилищем данных. Например, в предыдущих главах, посвященных созданию веб-сайтов на платформе ASP.NET, мы разрабатывали страницу регистрации абитуриентов. Давайте теперь попробуем создать аналогичную страницу с использованием MVC.

В архитектуре MVC есть понятие привязки модели (model binding). С его помощью осуществляется автоматический разбор входных данных, вводимых пользователем на странице представления, после чего, посредством сопоставления входящих пар "ключ/значение" с именами свойств заданного типа, результатами этого разбора заполняются параметры метода действия.

Привязка модели заменяет обработку HTTP-запросов в классическом ASP.NET, позволяя работать со строго типизированными объектами .NET. Поскольку у элементов управления вводом, определенных в Registration.aspx, есть имена, соответствующие именам свойств в классе `RegistrationRequest`, каркас передает методу действия экземпляр класса `RegistrationRequest`, заполненный данными, которые пользователь ввел в форму.

Также класс модели `RegistrationRequest` будет реализовывать логику проверки данных, вводимых пользователем на странице регистрации, и будет иметь заглушку для метода `Submit()` для отправления введенных данных. Код класса `RegistrationRequest` представлен в листинге 17.7.

Листинг 17.7. Код класса модели `RegistrationRequest`

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Web;

namespace College.web.Models
{
   public class RegistrationRequest
   {
      [Required(ErrorMessage = "Введите имя!")]
      public string FirstName { get; set; }

      [Required(ErrorMessage = "Введите фамилию!")]
      public string LastName { get; set; }

      [Required(ErrorMessage = "")]
      [RegularExpression(".*@.{2,}\\..{2,}",
        ErrorMessage = "Неправильный E-Mail")]
      public string Email { get; set; }
```

```
        [Required(ErrorMessage = "Выберите факультет!")]
        public bool? IsSelectFaculty { get; set; }

        [Required(ErrorMessage = "Выберите форму обучения!")]
        public bool? IsSelectEducationForm { get; set; }

        public void Submit()
        {
           // код для отправки сообщения
        }
    }
}
```

Чтобы получать и обрабатывать отправленные данные формы, создадим новый класс контроллера `ApplicantsController`. В этом классе будет два метода `Registration()`, возвращающих `ViewResult`, один из которых будет принимать в качестве параметра экземпляр класса `RegistrationRequest` из листинга 17.7.

Код класса контроллера `ApplicantsController` представлен в листинге 17.8.

Листинг 17.8. Код класса контроллера `ApplicantsController`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using College.Web.Models;

namespace College.web.Controllers
{
    [HandleError]
    public class ApplicantsController : Controller
    {
       [HttpGet]
       public ViewResult Registration()
       {
          return View();
       }

       [HttpPost]
       public ViewResult Registration(
          RegistrationRequest registrationRequest)
       {
          if (ModelState.IsValid)
          {
             RegistrationRequest.Submit();
             return View("Спасибо за регистрацию!", RegistrationRequest);
          }
```

```
            else
            {
               return View();
            }
        }
    }
}
```

Создайте новое представление, щелкнув правой кнопкой мыши внутри любого метода действия в коде класса `ApplicantsController` и выбрав в контекстном меню пункт **Add View**. Это представление будет несколько отличаться: оно будет предназначено для визуализации одного конкретного типа объекта модели, а не как предыдущие строго типизированные представления, которые просто визуализировали коллекцию элементов. То есть это будет строго типизированное представление для страницы регистрации абитуриента.

Для этого откройте файл Registration.aspx и добавьте в него следующий код, представленный в листинге 17.9.

Листинг 17.9. Код представления Registration.aspx

```
<%@ Page Title="" Language="C#"
  MasterPageFile="~/Views/Shared/Site.Master"
  Inherits=
  "System.Web.Mvc.ViewPage<College.web.Models.RegistrationRequest>"%>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
   runat="server">
       Registration
</asp:Content>

<asp:Content ID="Content2"
  ContentPlaceHolderID="MainContent" runat="server">
    <h1>Регистрация абитуриента</h1>

    <% using(Html.BeginForm()) { %>
        <%: Html.ValidationSummary() %>
        <p>Имя <%: Html.TextBoxFor(x => x.FirstName) %></p>
        <p>Фамилия <%: Html.TextBoxFor(x => x.LastName) %></p>
        <p>E-Mail <%: Html.TextBoxFor(x => x.Email) %></p>
        <p>Факультет <%: Html.DropDownListFor
              (x => x.IsSelectFaculty, new[] {
                new SelectListItem {
                   Text = "Информационные технологии",
                   Value = bool.TrueString },
                new SelectListItem {
                   Text = "Электроника и телекоммуникации",
                   Value = bool.FalseString }
                },
```

```
                "--Выберите факультет--") %>
        <p>Обучение
            <%: Html.DropDownListFor(x => x.IsSelectEducationForm, new[] {
                new SelectListItem { Text = "Дневная форма обучения",
                                     Value = bool.TrueString },
                new SelectListItem { Text = "Вечерняя форма обучения",
                                     Value = bool.FalseString }
                }, "--Выберите форму обучения--")%>
        </p>
        <input type="submit" value="Отправить" />
    <% } %>
</asp:Content>
```

Этот код должен генерировать поля для регистрации абитуриента, похожие на те, которые мы создавали в предыдущих главах, посвященных разработке классических ASP.NET веб-сайтов.

Запустите проект на выполнение и в браузере перейдите на страницу **Курсы**. Эта страница аналогична созданным ранее ASP.NET страницам регистрации абитуриента и имеет такую же функциональность (рис. 17.9).

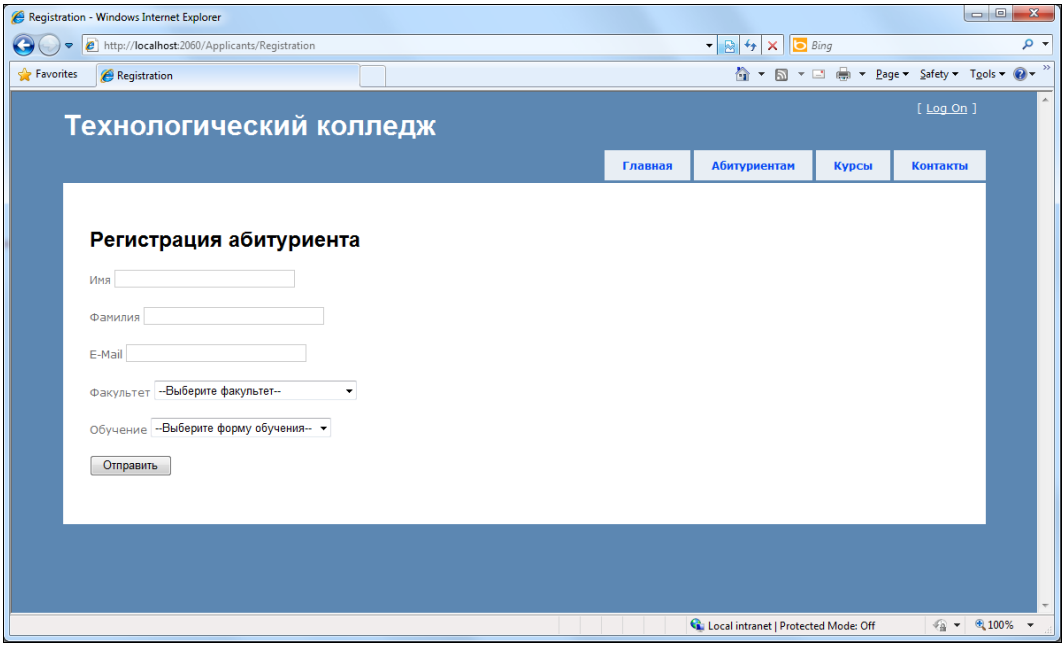


Рис. 17.9. Представление для регистрации абитуриента

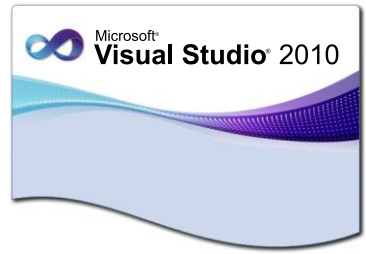
На этой странице также реализована валидация при отправке данных на сервер, однако, как вы видели, способ реализации валидации данных в корне отличается от классического ASP.NET.

Резюме

Платформа ASP.NET MVC значительно облегчает управление сложными структурами путем разделения веб-приложения на модель, представление и контроллер. Она не использует состояние просмотра и серверные формы. Это делает платформу MVC идеальной для разработчиков, которым необходим полный контроль над поведением приложения. MVC также обеспечивает расширенную поддержку разработки на основе тестирования и хорошо подходит для командной разработки веб-приложений.

Перед использованием платформы MVC или модели веб-форм для определенного веб-сайта следует взвесить все преимущества каждого из подходов. Необходимо внимательно продумать вопрос о создании веб-приложения на основе платформы ASP.NET MVC или на основе модели веб-форм ASP.NET.

Платформа MVC не заменяет собой модель веб-форм. Обе модели можно использовать для веб-приложений.



ГЛАВА 18

Windows Presentation Foundation

Windows Presentation Foundation (WPF) — это система нового поколения для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем. С помощью WPF можно создавать широкий спектр как автономных, так и размещенных в браузере приложений.

В основе Windows Presentation Foundation лежит векторная система визуализации, не зависящая от разрешения и созданная с расчетом на возможности современного графического оборудования. Windows Presentation Foundation располагает богатым набором инструментов для разработки приложений, элементами управления, включая элементы для привязки данных, элементы для работы с двухмерной и трехмерной графикой, анимацией, стилями, шаблонами, документами и мультимедийными файлами. Windows Presentation Foundation входит в состав Microsoft .NET Framework и позволяет создавать приложения, включающие другие элементы библиотеки классов .NET Framework.

Размеры окон и элементов управления приложений, созданных на Windows Forms, зависят от разрешения экрана. При разработке таких приложений обычно ориентируются на стандартное разрешение монитора и проектируют окна с учетом этого, стараясь обеспечить сохранение общего дизайна и компоновки окон при изменении размеров в большую и меньшую сторону. В результате при использовании монитора высокого разрешения, который располагает пикселы более плотно, окно вашего приложения становится меньше, и читать его будет труднее.

Приложения Windows Presentation Foundation не зависят от разрешения экрана, потому что самостоятельно визуализируют все элементы пользовательского интерфейса. Например, если программист создает кнопку шириной 2 см на своем мониторе, она останется шириной в 2 см и на мониторе высокого разрешения. Windows Presentation Foundation просто визуализирует ее более детализированно, с большим количеством пикселов.

Архитектура WPF

Основные пространства имен WPF начинаются с `System.Windows`. На рис. 18.1 показана иерархия основных классов WPF.

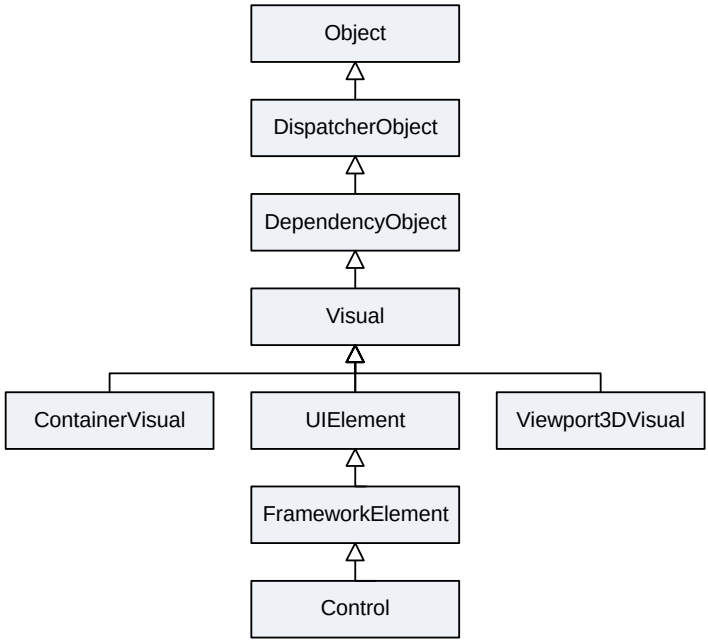


Рис. 18.1. Фундаментальные классы WPF

Как видно из рис. 18.1, классы WPF порождаются из класса `DispatcherObject`. Класс `DispatcherObject` предназначен для обеспечения доступа к объекту `Dispatcher`, который предоставляет службы для управления очередью рабочих элементов для потока. Дело в том, что приложения WPF используют *однопоточную модель* (Single-Thread Affinity, STA), а это означает, что весь пользовательский интерфейс принадлежит только одному потоку. Взаимодействовать с элементами пользовательского интерфейса из других потоков небезопасно. Чтобы содействовать работе этой модели, каждое приложение WPF управляется диспетчером, координирующим сообщения от клавиатуры, перемещений курсора мыши и др. Поэтому при наследовании от класса `DispatcherObject`, каждый элемент пользовательского интерфейса WPF может обратиться к диспетчеру, чтобы направить код в поток пользовательского интерфейса.

Класс `DependencyObject` включает службы системы свойств Windows Presentation Foundation (WPF) для множества своих производных классов. Основной функцией системы свойств является вычисление значений свойств и системное уведомление о значениях, которые изменились. Наследуясь от `DependencyObject`, классы WPF получают поддержку системы свойств WPF.

Класс `Visual` как единственный объект рисования инкапсулирует в себе инструкции рисования, дополнительные подробности рисования (наподобие отсечения, прозрачности и настроек трансформации) и базовую функциональность (такую как проверка попадания). Класс `Visual` также обеспечивает связь между управляемыми библиотеками WPF и библиотекой визуализации milcore.dll. Любой класс, унаследованный от Visual, обладает способностью отображаться в окне.

Класс `UIElement` добавляет поддержку компоновки, ввода, фокуса и событий. В этом классе обработка нажатий кнопок мыши и клавиатуры трансформируется в более удобные события, такие как `MouseEnter`. Класс `UIElement` также добавляет поддержку привязки данных, анимации и стилей.

Класс `FrameworkElement` — конечный пункт в центральном дереве наследования WPF. Он реализует некоторые члены, которые просто определены в классе `UIElement`.

Класс `Control` добавляет дополнительные свойства для установки шрифта, а также цветов переднего плана и фона. Но наиболее интересная деталь, которую он предоставляет, — это поддержка шаблонов, которая позволяет заменять стандартный внешний вид элемента управления вашим собственным рисованием.

В программировании с применением Windows Forms каждый визуальный компонент в форме называется элементом управления. В WPF это не так. Визуальные единицы называются элементами, и только некоторые из них являются элементами управления (те, что могут принимать фокус и взаимодействовать с пользователем).

Типы приложений WPF

Windows Presentation Foundation (WPF) поддерживает создание следующих типов приложений.

- Автономные приложения — Windows-приложения, построенные как исполняемые сборки, которые устанавливаются и запускаются с клиентского компьютера.
- XAML-приложения веб-браузера (XBAP) — приложения состоят из навигационных страниц, построенные как исполняемые сборки, которые просматриваются и размещаются с помощью Windows Internet Explorer.
- Пользовательские библиотеки элементов управления — неисполняемые сборки, содержащие многократно используемые элементы управления.
- Библиотеки классов — неисполняемые сборки, содержащие многократно используемые классы.

Для автономных приложений можно создавать доступные из меню и панелей инструментов окна и диалоговые окна с помощью класса `Window`.

Создание приложения WPF

Сейчас создадим проект WPF в Visual Studio. Мы выбираем шаблон проекта **WPF Application**, который находится в категории **Windows** в диалоговом окне **Add New Project** (рис. 18.2).

Этот шаблон добавляет все необходимые пространства имен WPF. В проект также будут включены файлы, которые определяют поведение приложения (App.xaml) и реализуют для приложения окно по умолчанию (MainWindow.xaml). Структура проекта WPF приведена на рис. 18.3.

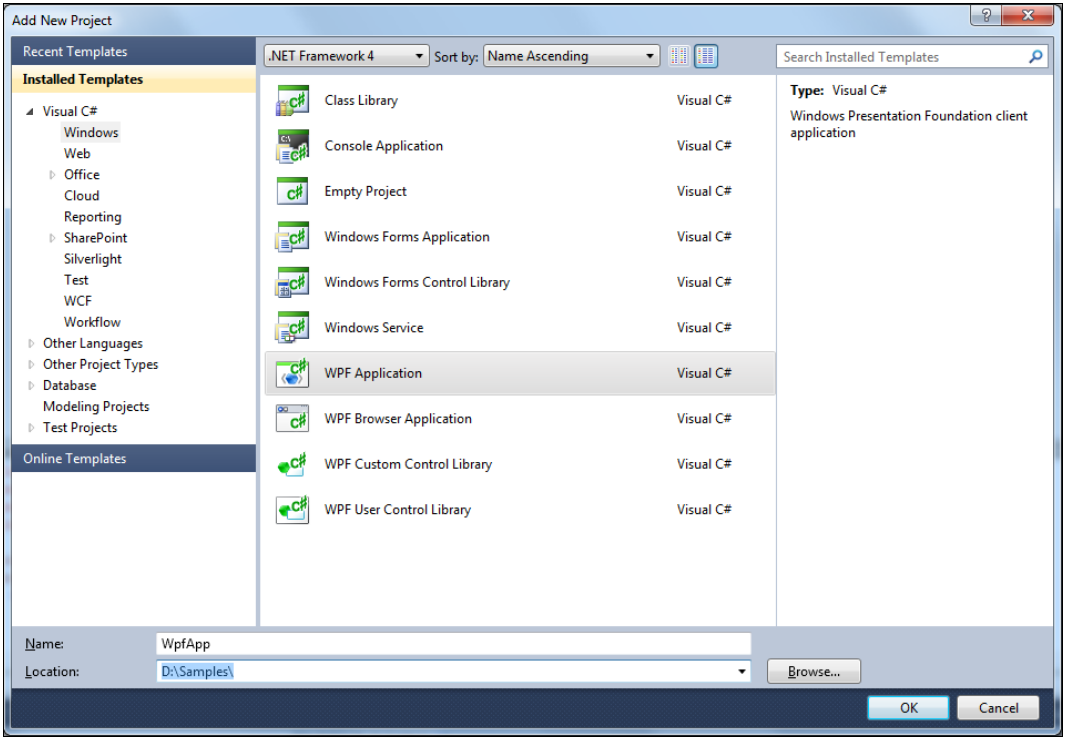


Рис. 18.2. Шаблоны проектов WPF в Visual Studio 2010

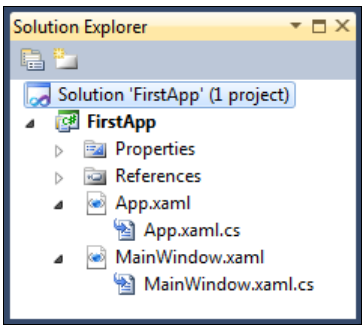


Рис. 18.3. Файлы проекта WPF Application

Файл App.xaml создается интегрированной средой разработки Visual Studio при создании проекта WPF по шаблону. Обычно он используется для хранения общей информации всего приложения и обеспечения функциональности для запуска приложения WPF. Файл App.xaml содержит объявления, определяющие поведение всего приложения. Пример файла App.xaml, создаваемого шаблоном по умолчанию, приведен в листинге 18.1.

Листинг 18.1. Файл App.xaml

```
<Application x:Class="WpfApp.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
            xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
            StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

Обратите внимание на атрибут `x:Class` в листинге 18.1. Атрибут `x:Class` сообщает анализатору XAML, чтобы он сгенерировал новый класс с указанным именем. Этот класс наследуется от базового класса `Application` и помещается в файл App.xaml.cs. Код этого класса приложения WPF, сгенерированного IDE Visual Studio, показан в листинге 18.2.

Листинг 18.2. Файл App.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Windows;

namespace WpfApp
{
   /// <summary>
   /// Interaction logic for App.xaml
   /// </summary>
   public partial class App : Application
   {
   }
}
```

В процессе выполнения каждое приложение WPF представлено экземпляром класса `System.Windows.Application`. Класс `Application` отслеживает все открытые окна в вашем приложении, решает, когда ваше приложение должно быть остановлено, и инициирует события приложения, которые вы можете обрабатывать для выполнения. Класс `Application` инкапсулирует функциональность приложения WPF, включая:

- создание и управление общей инфраструктурой приложений;
- отслеживание и взаимодействие со временем жизни приложения;
- извлечение и обработку параметров командной строки;
- совместное использование свойств области приложения и ресурсов;
- обнаружение и реагирование на необработанные исключения;
- возврат кодов завершения;

- управление окнами автономных приложений;
- отслеживание и управление навигацией.

Только один экземпляр класса `Application` может быть создан для одного `AppDomain`.

Файл MainWindow.xaml является для приложения пользовательским интерфейсом по умолчанию. При компиляции вместе с ассоциированным с ним файлом выделенного кода он формирует класс `MainWindow`, экземпляр которого будет объектом `MainWindow`. Код разметки на XAML, формируемый по умолчанию для MainWindow.xaml, приведен в листинге 18.3.

Листинг 18.3. Файл MainWindow.xaml

```
<Window x:Class="WpfApp.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>
```

Для того чтобы создать приложение, взаимодействующее с пользователем, необходимо подключение обработчиков событий, содержащих код приложения. Атрибут `x:Class` сообщает анализатору XAML, чтобы он сгенерировал новый класс `MainWindow`. `MainWindow` — это имя класса по умолчанию, присваиваемое при создании проекта с единственным окном. Этот класс наследуется от базового класса `Window`. Код класса `MainWindow` шаблона приложения WPF, сгенерированного IDE Visula Studio, показан в листинге 18.4.

Листинг 18.4. Файл MainWindow.xaml.cs

```
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 WpfApp
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

Метод `InitializeComponent()` вызывается из конструктора класса с выделенным кодом, чтобы объединить пользовательский интерфейс, определенный в разметке, с этим классом. Метод `InitializeComponent()` генерируется при построении приложения, поэтому нет необходимости реализовывать его вручную. Комбинация атрибута `x:Class` и метода `InitializeComponent()` позволяет гарантировать правильную инициализацию окна приложения.

Двойной щелчок мыши на файле MainWindow.xaml откроет дизайнер окон WPF, представленный на рис. 18.4.

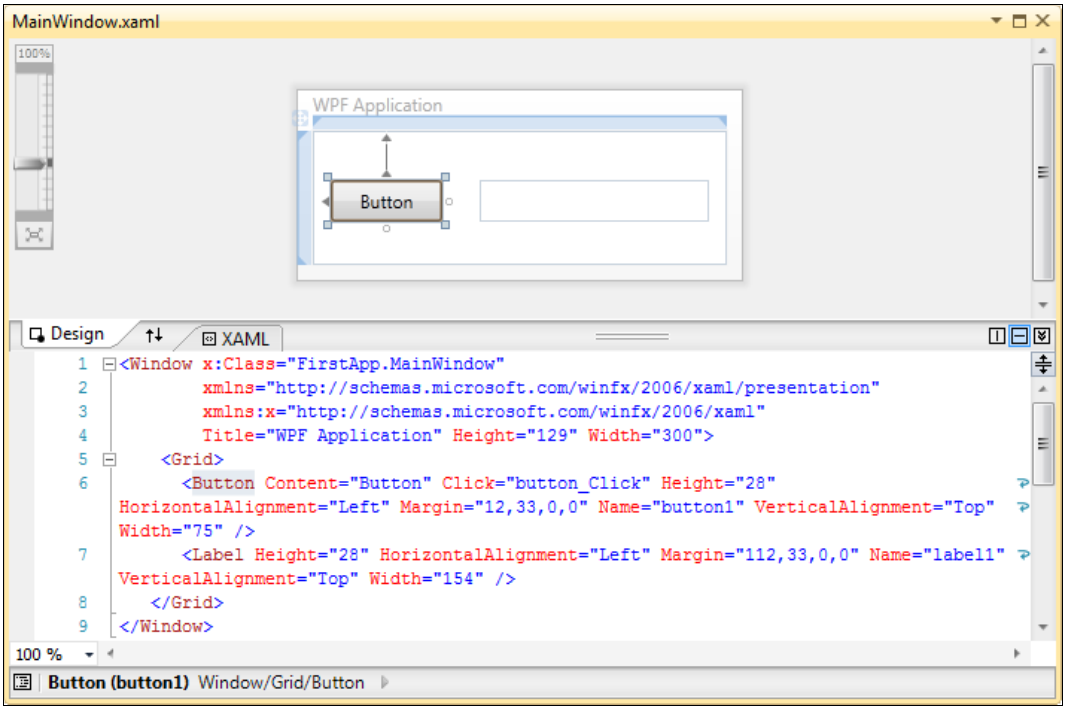


Рис. 18.4. Дизайнер WPF

Дизайнер окон WPF — это инструмент Visual Studio, который обеспечивает поверхность конструирования для создания окон WPF. Во многих отношениях он ведет себя точно так же, как те дизайнеры, которые мы используем при создании веб-форм и форм Windows.

В дизайнере WPF есть еще один дополнительный элемент в левом верхнем углу окна дизайнера: ползунок управления масштабированием окна. Он используется для изменения масштаба текущего окна от 10 до 2000 %.

Когда вы в визуальном дизайнере выбираете элемент управления, то можете управлять его атрибутами при помощи окна **Properties** (рис. 18.5).

В отличие от окон свойств Windows Forms или веб-форм, вы не можете отсортировать свойства в алфавитном порядке — это окно поддерживает только представление свойств с разбивкой по категориям. Кроме того, окно свойств WPF имеет область заголовка, которая используется для именования элементов управления и которую вы можете применять для поиска свойств элемента управления. Во время ввода текста в этом поле окно начинает автоматически фильтровать список свойств, чтобы отображать только те, которые соответствуют вашему критерию поиска.

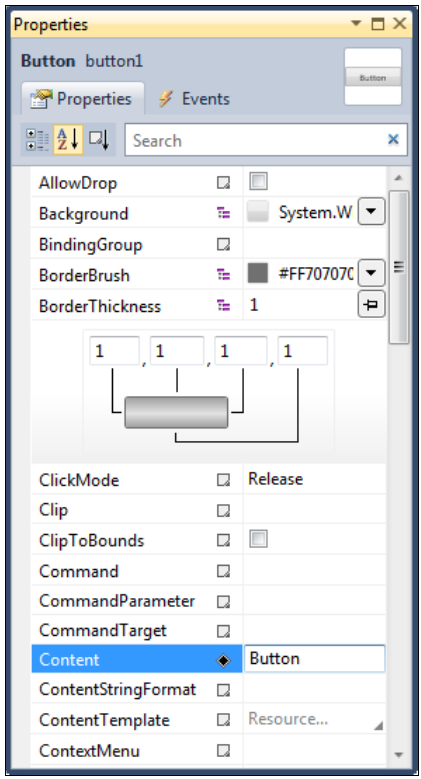


Рис. 18.5. Окно свойств дизайнера WPF

Приложение в основном предназначено для реализации функциональных возможностей, которые отвечают на взаимодействия с пользователем, включая обработку событий (например, нажатие меню, панели инструментов или кнопки) и вызов бизнес-логики и логики доступа к данным в ответ на события. В WPF такое поведение обычно реализуется в коде, который связан с разметкой. Этот тип кода называется *кодом программной части*. В следующем примере показан код программной части и обновленная разметка из предыдущего примера.

Язык XAML — это язык разметки, основанный на XML, который используется для декларативной реализации внешнего вида приложения. Обычно он применяется для создания окон, диалоговых окон, страниц и пользовательских элементов управления, а также для их заполнения элементами управления, фигурами и графикой.

Граница между XAML и кодом оказывается также и разделяющей линией между внешним видом и поведением. При этом XAML используется для создания объектов пользовательского интерфейса, а процедурный код применяется для реализации бизнес-логики и для реагирования на ввод пользователя. Это приводит нас к еще одному важному преимуществу XAML — совместной работе.

В следующем примере с помощью языка XAML реализуется внешний вид окна, содержащего одну кнопку.

Поскольку XAML основан на языке XML, пользовательский интерфейс, который формируется с его помощью, организуется в виде иерархии вложенных элементов, называемой *деревом элементов*. Дерево элементов предоставляет логический и интуитивно понятный способ для создания и управления пользовательскими интерфейсами для WPF.

Для первого примера создадим простейшее приложение с элементами `Button` и `Label`. При нажатии кнопки будет отображаться текст "Hello, WPF!". В файл MainWindow.xaml добавьте код, представленный в листинге 18.5.

> ***Примечание***
>
> Полный код примера приложения находится в каталоге Chapter18\WpfApp на прилагаемом к книге диске.

Листинг 18.5. Файл MainWindow.xaml

```
<Window x:Class="FirstApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="WPF Application" Height="129" Width="300">
    <Grid>
      <Button Content="Button" Click="button_Click" Height="28"
          HorizontalAlignment="Left" Margin="12,33,0,0" Name="button1"
          VerticalAlignment="Top" Width="75" />
      <Label Height="28" HorizontalAlignment="Left" Margin="112,33,0,0"
          Name="label1" VerticalAlignment="Top" Width="154" />
   </Grid>
</Window>
```

В коде программной части класса `MainWindow` напишите реализацию обработчика события нажатия кнопки `button_Click`, как показано в листинге 18.6.

Листинг 18.6. Файл MainWindow.xaml.cs

```
namespace WpfApp
{
   /// <summary>
   /// Interaction logic for MainWindow.xaml
   /// </summary>
   public partial class MainWindow : Window
```

```
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void button_Click(object sender, RoutedEventArgs e)
        {
            label1.Content = "Hello, WPF!";
        }
    }
}
```

Скомпилируйте и запустите приложение. Внешний вид приложения показан на рис. 18.6.

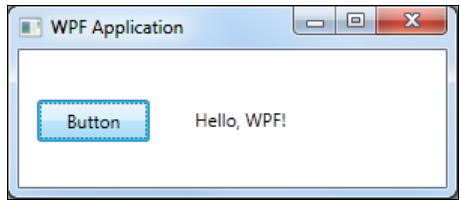


Рис. 18.6. Окно созданного приложения WPF

Компоновка окна приложения WPF

В WPF в качестве основной принята потоковая (flow-based) компоновка элементов, в отличие от в Windows Forms, где в основном используется координатная компоновка. Разработчики могут создавать независящие от разрешения и размера интерфейсы, которые масштабируются на разных мониторах, подгоняют себя при изменении содержимого и легко обрабатывают перевод на другие языки.

При создании пользовательского интерфейса происходит упорядочивание элементов управления по расположению и размеру для формирования структуры. Основным требованием любой структуры является адаптируемость к изменениям размера окна и параметрам отображения. Чтобы не пришлось создавать код для адаптации структуры в таких обстоятельствах, WPF предоставляет первоклассную расширяемую систему структуры.

Основой системы структуры является относительное позиционирование, что увеличивает способность адаптации к изменяемому окну и условиям отображения. Кроме того, система структуры управляет согласованием между элементами управления для определения структуры. Такое согласование состоит из двух этапов: сначала элемент управления сообщает родительскому элементу, какое расположение и размер требуется; затем родительский элемент сообщает элементу управления, какое пространство он может занять.

Дочерние элементы управления получают доступ к системе макета через базовые классы WPF. Для распространенных макетов, таких как сетки, вложение и закрепление, WPF включает несколько элементов управления макетом.

- `Canvas` — дочерние элементы управления предоставляют свои собственные макеты.
- `DockPanel` — дочерние элементы управления выравниваются по краям панели.
- `Grid` — дочерние элементы управления располагаются по строкам и столбцам.
- `StackPanel` — дочерние элементы управления располагаются либо горизонтально, либо вертикально.
- `VirtualizingStackPanel` — дочерние элементы управления являются виртуальными и располагаются в одной горизонтальной или вертикальной строке.
- `WrapPanel` — дочерние элементы управления располагаются в порядке слева направо и переносятся на следующую строку, когда в текущей строке не хватает места.

Контейнер *Grid*

Элемент `Grid` — наиболее часто используемый контейнер для компоновки элементов управления в WPF. `Grid` является в WPF компоновкой по умолчанию при создании новых шаблонов XAML, которые генерирует среда разработки Visual Studio. Большая часть того, что вы можете достичь с другими элементами управления компоновкой, также возможна и в `Grid`. Контейнер `Grid` является идеальным инструментом для разбиения вашего окна на меньшие области, которыми вы можете управлять с помощью других панелей.

`Grid` располагает элементы в сетке из строк и колонок. Хотя в отдельную ячейку сетки можно поместить несколько элементов, обычно в каждую ячейку помещают по одному элементу. Этот элемент, в свою очередь, сам может быть другим контейнером компоновки, который организует свою собственную группу содержащихся в нем элементов управления.

Для создания компоновки окна на основе `Grid` сначала выбирается количество колонок и строк. Затем элементам назначается соответствующая строка и колонка. В листинге 18.7 приведен пример компоновки `Grid` с двумя строками и двумя столбцами, в которых размещены командные кнопки. Чтобы поместить кнопки в ячейку, используются свойства `Grid.Row` и `Grid.Column`. Если свойство `ShowGridLines` установлено в `true`, будут отображаться границы между колонками и строками во время работы программы.

Пример разметки окна программы с использованием контейнерного элемента `Grid` показан в листинге 18.7.

> ***Примечание***
>
> Полный код примеров компоновок, представленных в этом разделе, находится в каталоге Chapter18\Layouts на прилагаемом к книге диске.

Листинг 18.7. Компоновка элементов в контейнере `Grid`

```
<Window x:Class="WPFCanvas.WGrid"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Grid Layout" Height="204" Width="275">

    <Grid ShowGridLines="True">
      <Grid.RowDefinitions>
         <RowDefinition></RowDefinition>
         <RowDefinition></RowDefinition>
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
         <ColumnDefinition></ColumnDefinition>
         <ColumnDefinition></ColumnDefinition>
      </Grid.ColumnDefinitions>
      <Button Content="Button" Grid.Row="0" Grid.Column="0"
         Margin="3" Click="button_Click" />
      <Button Content="Button" Grid.Row="0" Grid.Column="1"
         Margin="3" Click="button_Click" />
      <Button Content="Button" Grid.Row="1" Grid.Column="0"
         Margin="3" Click="button_Click" />
      <Button Content="Button" Grid.Row="1" Grid.Column="1"
         Margin="3" Click="button_Click" />
   </Grid>
</Window>
```

На рис. 18.7 показано приложение с этой компоновкой.

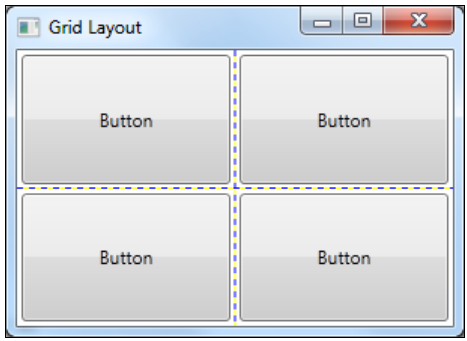


Рис. 18.7. Разметка `Grid`

Так же как при разметке веб-страницы с помощью HTML-элемента `<table>` можно при необходимости растянуть элемент на несколько ячеек, используя свойства `RowSpan` и `ColumnSpan`. Эти свойства принимают количество строк или колонок, которые должен занять элемент. Например, следующая кнопка займет все место, доступное в первой и второй ячейках первой строки:

```
<Button Content="Button" Grid.Row="0" Grid.Column="0" Grid.RowSpan="0"/>
```

Контейнер *UniformGrid*

Контейнерный элемент `UniformGrid` в отличие от `Grid` не требует определения колонок и строк. При использовании `UniformGrid` устанавливаются свойства `Rows` и `Columns`. Каждая ячейка всегда имеет одинаковый размер, потому что доступное пространство делится поровну. Элементы помещаются в соответствующую ячейку в порядке их определения.

Пример компоновки окна с элементом `UniformGrid` и четырьмя кнопками приведен в листинге 18.8.

Листинг 18.8. Компоновка элементов в контейнере `UniformGrid`

```
<Window x:Class="WPFCanvas.WUniformGrid"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="UniformGrid Layout" Height="300" Width="300">
   <UniformGrid Rows="2" Columns="2">
      <Button Content="Button" Grid.Row="0" Grid.Column="0"
         Margin="3" Click="button_Click" />
      <Button Content="Button" Grid.Row="0" Grid.Column="1"
         Margin="3" Click="button_Click" />
      <Button Content="Button" Grid.Row="1" Grid.Column="0"
         Margin="3" Click="button_Click" />
      <Button Content="Button" Grid.Row="1" Grid.Column="1"
         Margin="3" Click="button_Click" />
   </UniformGrid>
</Window>
```

Окно приложения с компоновкой `UniformGrid` и четырьмя кнопками на нем показано на рис. 18.8.

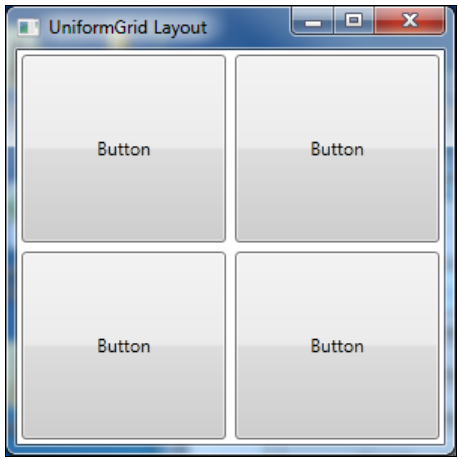


Рис. 18.8. Разметка `UniformGrid`

Контейнер `UniformGrid` используется намного реже, чем `Grid`. `UniformGrid` является более специализированным контейнером компоновки, который предназначен для размещения элементов в жесткой сетке.

Контейнер *Canvas*

Контейнер `Canvas` позволяет размещать элементы, используя точные координаты. Чтобы позиционировать элемент на `Canvas`, задают свойства `Canvas.Left` и `Canvas.Top`. Свойство `Canvas.Left` задает количество единиц измерения между левой гранью вашего элемента и левой границей `Canvas`. Свойство `Canvas.Top` устанавливает количество единиц измерения между вершиной вашего элемента и левой границей `Canvas`. Эти значения задаются в независимых от устройства единицах измерения, которые соответствуют обычным пикселам, когда системная установка DPI составляет 96 dpi.

В листинге 18.9 приведен пример компоновки с использованием контейнера `Canvas` с четырьмя командными кнопками.

Листинг 18.9. Компоновка элементов в контейнере `Canvas`

```
<Window x:Class="WPFCanvas.WCanvas"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Canvas Layout" Height="141" Width="186">
   <Canvas>
      <Button Canvas.Left="0" Canvas.Top="0" Content="Button"
         Height="23" Width="75" Click="button_Click" />
      <Button Canvas.Left="89" Canvas.Top="0" Content="Button"
         Height="23" Width="75" Click="button_Click" />
      <Button Canvas.Left="0" Canvas.Top="29" Content="Button"
         Height="41" Width="164" Click="button_Click" />
      <Button Canvas.Left="38" Canvas.Top="76" Content="Button"
         Height="23" Width="88" Click="button_Click" />
   </Canvas>
</Window>
```

Окно приложения с компоновкой `Canvas` представлено на рис. 18.9.

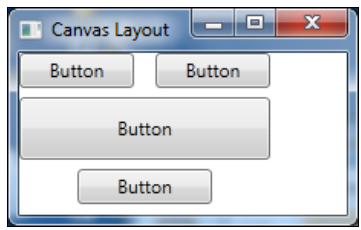


Рис. 18.9. Размещение элементов в разметке `Canvas`

Контейнер *DockPanel*

Контейнер `DockPanel` позволяет упорядочить дочерние элементы горизонтально или вертикально относительно друг друга. Для этого у `DockPanel` есть свойство `Dock`, с помощью которого дочерним элементам управления можно указать стиль закрепления.

В следующем примере кода разметки окна, показанном в листинге 18.10, в контейнере `DockPanel` размещаются пять кнопок.

Листинг 18.10. Компоновка элементов в контейнере `DockPanel`

```
<Window x:Class="WPFCanvas.WDockPanel"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="DockPanel Layout" Height="300" Width="300">
   <DockPanel>
      <Button DockPanel.Dock="Top" Content="Button"
         Margin="3" Click="button_Click" />
      <Button DockPanel.Dock="Bottom" Content="Button"
         Margin="3" Click="button_Click" />
      <Button DockPanel.Dock="Left" Content="Button"
         Margin="3" Click="button_Click" Width="49" />
      <Button DockPanel.Dock="Right" Content="Button"
         Margin="3" Click="button_Click" />
      <Button Content="Button" Grid.Row="0" Grid.Column="0"
         Margin="3" Click="button_Click" />
   </DockPanel>
</Window>
```

Окно приложения с компоновкой `DockPanel` и пятью кнопками на нем представлено на рис. 18.10.

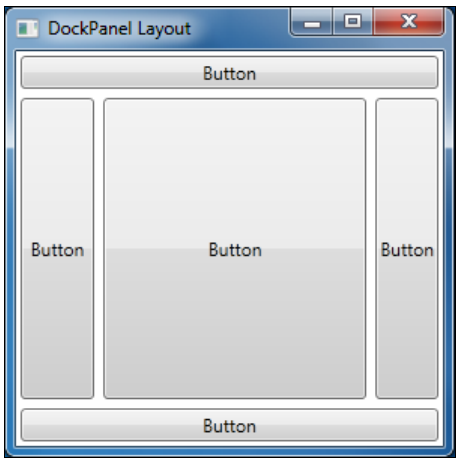


Рис. 18.10. Контейнер `DockPanel`

Контейнер *StackPanel*

Элементы управления `StackPanel` реализуют вертикальную (колонкой) или горизонтальную (строкой) компоновку дочерних элементов. Ориентация элементов определяется при помощи свойства `Orientation`, и после этого панель сама позаботится обо всем остальном. Размер каждого элемента внутри панели `StackPanel` будет масштабирован таким образом, чтобы они разместились в пределах высоты (вертикальная ориентация) или ширины (горизонтальная ориентация) панели.

В листинге 18.11 приведен пример горизонтальной компоновки элементов с использованием разметки `StackPanel`.

Листинг 18.11. Компоновка элементов в контейнере `StackPanel`

```
<Window x:Class="WPFCanvas.WStackPanelHorizontal"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="StackPanelHorizontal Layout" Height="300" Width="300">
   <StackPanel Orientation="Horizontal">
      <Button Margin="3" Content="Button" Height="142" Width="83"
Click="button_Click" />
      <Button Margin="3" Content="Button" Height="133" Width="68"
Click="button_Click" />
      <Button Margin="3" Content="Button" Height="51" Width="96"
Click="button_Click" />
   </StackPanel>
</Window>
```

Для реализации вертикальной ориентации элементов в контейнере `StackPanel` достаточно просто поменять свойство `Orientation` на `Vertical`, как показано в листинге 18.12.

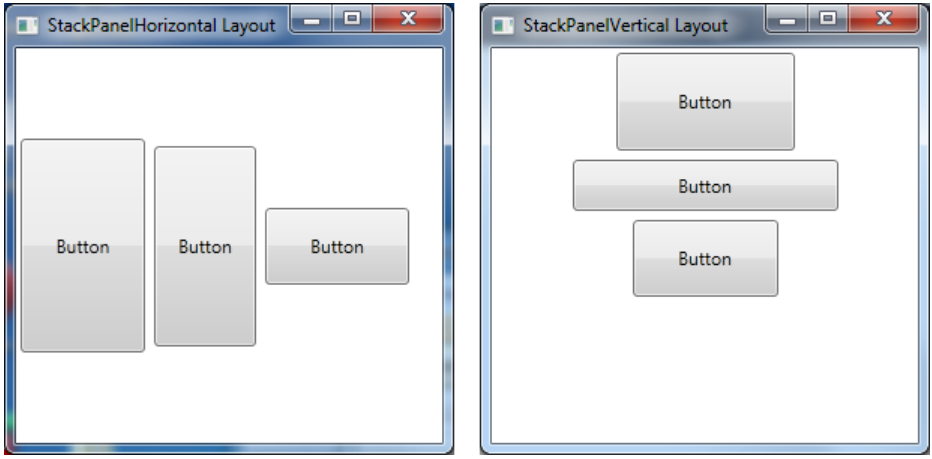


Рис. 18.11. Разметка окна `StackPanel`

Листинг 18.12. Создание вертикальной ориентации элементов в `StackPanel`

```
<StackPanel Orientation="Vertical">
   ...
</StackPanel>
```

Примеры окон приложения с различными вариантами разметки `StackPanel` показаны на рис. 18.11.

Контейнер *WrapPanel*

Контейнер `WrapPanel` располагает элементы управления в доступном пространстве — по одной строке или колонке за раз. Как и у контейнера `StackPanel`, у `WrapPanel` есть свойство `Orientation`. По умолчанию свойство `Orientation` установлено в значение `Horizontal`. При этом элементы управления располагаются слева направо, затем — в следующих строках. Однако вы можете использовать свойство `Orientation`, равное `Vertical`, для размещения элементов в нескольких колонках.

Пример размещения элементов в окне с использованием разметки `WrapPanel` показан в листинге 18.13.

Листинг 18.13. Компоновка элементов в контейнере `WrapPanel`

```
<Window x:Class="WPFCanvas.WWrapPanel"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="WrapPanel Layout" Height="243" Width="367">

   <WrapPanel Margin="3">
      <Button Content="Button" Grid.Row="0" Grid.Column="0"
         Margin="3" Click="button_Click" Width="100" />
      <Button Content="Button" Grid.Row="0" Grid.Column="1"
         Margin="3" Click="button_Click" Width="100" />
      <Button Content="Button" Grid.Row="1" Grid.Column="0"
         Margin="3" Click="button_Click" Width="100" />
      <Button Content="Button" Grid.Row="1" Grid.Column="1"
         Margin="3" Click="button_Click" Width="100" />
   </WrapPanel>
</Window>
```

При изменении размеров окна будет меняться и расположение элементов: если увеличить ширину окна, элементы выстроятся в линию, и наоборот, при уменьшении горизонтального размера элементы расположатся в колонку. Примеры окон приложения с различными вариантами разметки `WrapPanel` показаны на рис. 18.12.

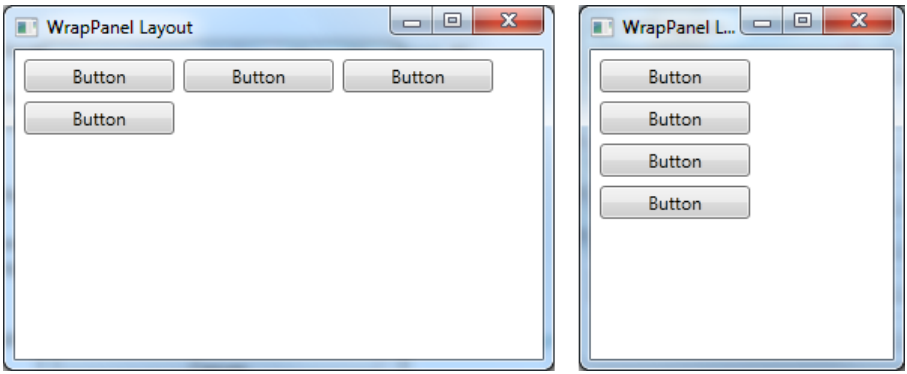


Рис. 18.12. Варианты расположения элементов в разметке `WrapPanel`

Переключение между окнами

Так же как и в Windows Forms, в приложениях WPF окна бывают модальными и немодальными. При отображении другой формы используются два метода, аналогичные методам в Windows Forms:

- `Show()` — для вызова немодального окна;
- `ShowDialog()` — для вызова модального окна.

Однако, в отличие от Windows Forms, метод `ShowDialog()` возвращает не перечислениие, а значение типа `bool?`, т. е. при закрытии диалогового окна с подтверждением метод `ShowDialog()` будет возвращать `true`, а при отмене — `false` или `null`.

Давайте рассмотрим пример переключения окон в WPF. Создадим приложение с двумя окнами: `MainWindow` (главное окно, открывающееся по умолчанию) и `ChildModal` — дочернее модальное окно.

В окне `MainWindow` будет кнопка для вызова окон с обработчикам события `Click`. Код XAML для этого окна приведен в листинге 18.14.

Примечание

Полный код примера приложения находится в каталоге Chapter18\Windows на прилагаемом к книге диске.

Листинг 18.14. Файл MainWindow.xaml

```
<Window x:Class="Windows.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="MainWindow" Height="156" Width="218">
   <Grid>
      <Button Content="Show Modal Window" Height="23"
         HorizontalAlignment="Left"
```

```
        Margin="22,22,0,0" Name="bShowModal" VerticalAlignment="Top"
        Width="146" Click="bShowModal_Click" />
   </Grid>
</Window>
```

В коде программной части реализуем обработку событий нажатия кнопок для вызова окон, как показано в листинге 18.15.

Листинг 18.15. Файл MainWindow.xaml.cs

```
private void bShowModal_Click(object sender, RoutedEventArgs e)
{
   ChildModal child = new ChildModal();
   if (child.ShowDialog().Value)
   {
      MessageBox.Show(String.Format(
         "Value from Modal window: {0}", child.OutputValue));
   }
}
```

На форме `ChildModal` разместим элемент `TextBox`, присвоим ему имя `tbValue` и две кнопки: одну с именем `bOK` и надписью **OK** и кнопку с именем `bCancel` и надписью **Cancel**. Код разметки XAML окна `ChildModal` показан в листинге 18.16.

Листинг 18.16. Файл ChildModal.xaml

```
<Window x:Class="Windows.ChildModal"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Modal Window" Height="161" Width="298">

    <Grid>
      <Button Content="OK" Height="23" HorizontalAlignment="Left"
         Margin="89,81,0,0" Name="bOK" VerticalAlignment="Top"
         Width="75" Click="bOK_Click" />
      <Button Content="Cancel" Height="23" HorizontalAlignment="Left"
         Margin="181,81,0,0" Name="bCancel" VerticalAlignment="Top"
         Width="75" Click="bCancel_Click" />
      <TextBox Height="23" HorizontalAlignment="Left" Margin="89,30,0,0"
         Name="tbValue" VerticalAlignment="Top" Width="167" />
      <Label Content="Enter value:" Height="28" HorizontalAlignment="Left"
         Margin="12,28,0,0" Name="label1" VerticalAlignment="Top" />
   </Grid>
</Window>
```

В коде программной части реализуем обработку событий нажатия кнопок и открытое свойство `Output` для чтения введенного текста из текстового поля, как показано в листинге 18.17.

Листинг 18.17. Файл ChildModal.xaml.cs

```
private void bOK_Click(object sender, RoutedEventArgs e)
{
   this.DialogResult = true;
}

private void bCancel_Click(object sender, RoutedEventArgs e)
{
   this.DialogResult = false;
}

public string OutputValue
{
   get { return tbValue.Text; }
}
```

Запустите приложение. Возвращаемое из метода `ShowDialog()` значение `true` говорит, что пользователь щелкнул кнопку **OK**. Если `ShowDialog()` возвращает значение, отличное от `true` (`false` или `null`), то вызывающая форма игнорирует введенные данные. Работа приложения показана на рис. 18.13.

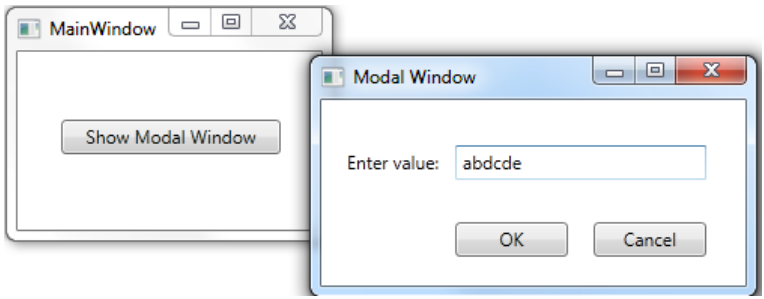


Рис. 18.13. Обмен данными между формами

Работа WPF с документами

WPF предоставляет встроенную поддержку работы с тремя типами документов: документами нефиксированного формата, документами фиксированного формата и документами в формате XPS (XML Paper Specification). WPF также предоставляет службы для создания и просмотра документов, управления документами, добавления заметок, упаковки и печати документов.

WPF обеспечивает поддержку каждого типа документов посредством различных контейнерных элементов. Так, например, `DocumentViewer` позволяет показывать фиксированные документы в окне WPF. Контейнеры `FlowDocumentReader`, `FlowDocumentPageViewer` и `FlowDocumentScrollViewer` предоставляют разные способы просмотра потоковых данных.

Ввиду недостатка места в книге мы рассмотрим только работу с документами XPS. Документы XPS построены на основе документов фиксированного формата WPF. Загружать и отображать документы в WPF очень легко. Для примера поддержки в WPF документов формата XPS создадим приложение, загружающее документ в окно программы. Код разметки окна приложения показан в листинге 18.18.

Примечание

Полный код примера приложения находится в каталоге Chapter18\Documents на прилагаемом к книге диске.

Листинг 18.18. Файл MainWindow.xaml

```
<Window x:Class="Documents.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="ViewXps" Height="300" Width="300"
    Loaded="window_Loaded" WindowStartupLocation="CenterScreen">
   <DocumentViewer Name="docViewer">

   </DocumentViewer>
</Window>
```

Для поддержки документов в проект необходимо будет добавить ссылку на .NET-сборку ReachFramework, как показано на рис. 18.14, и в код класса `MainWindow` подключить пространство имен `System.Windows.Xps.Packaging`.

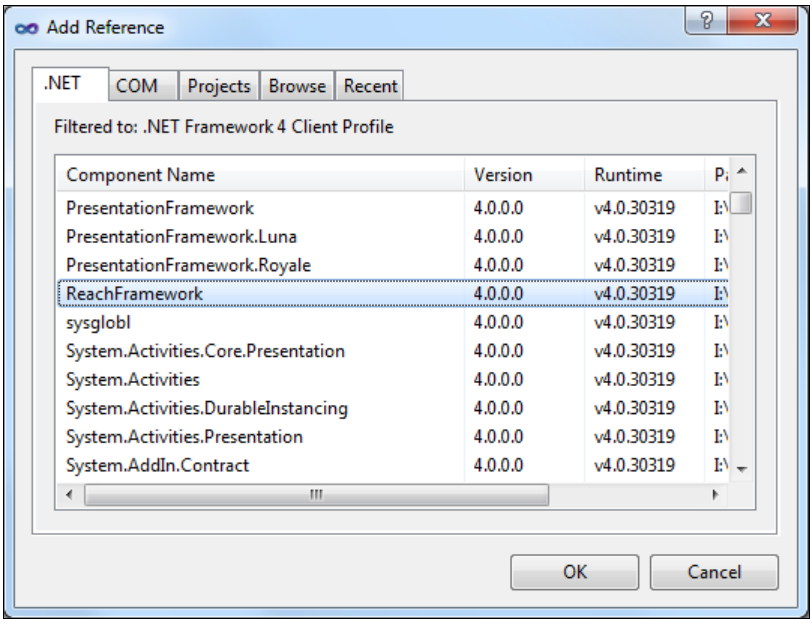


Рис. 18.14. Добавление ссылки на сборку ReachFramework

Документ XPS вы можете создать сами, на основе любого документа Word, пересохранив его в формате XPS. В класс `MainWindow` окна добавим обработчик события `window_Loaded()`, в теле которого будет реализована загрузка документа XPS из файла на диске, как показано в листинге 18.19.

Листинг 18.19. Файл MainWindow.xaml.cs

```
namespace Documents
{
   public partial class MainWindow : Window
   {
      public MainWindow()
      {
         InitializeComponent();
      }

      private void window_Loaded(object sender, RoutedEventArgs e)
      {
         XpsDocument doc = new XpsDocument(
            @"..\..\Test.xps", FileAccess.Read);
         docViewer.Document = doc.GetFixedDocumentSequence();
         doc.Close();
      }
   }
}
```

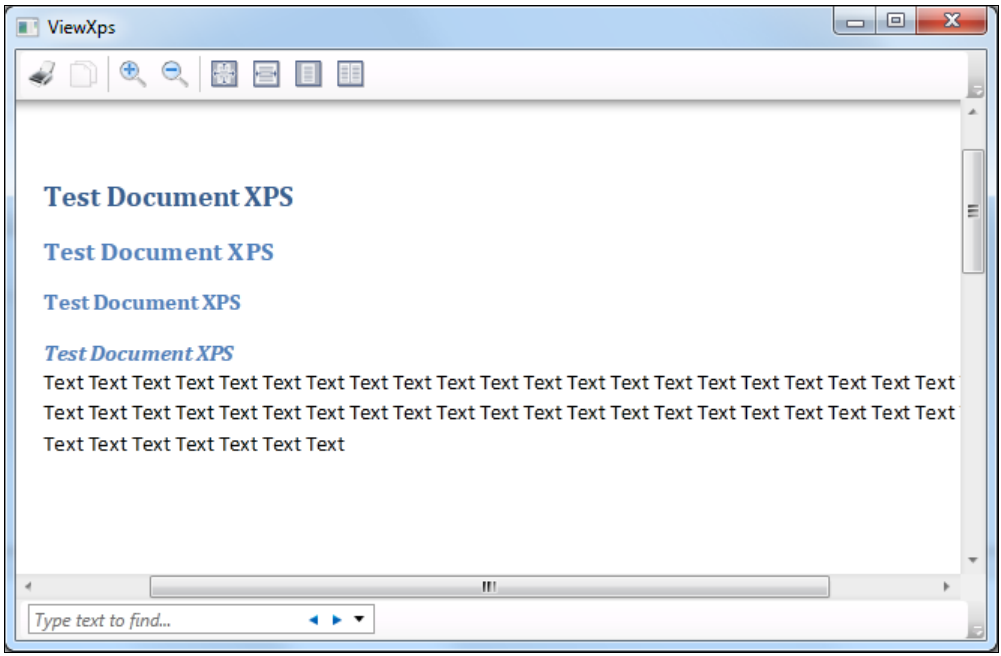


Рис. 18.15. Загрузка документа XPS

Приложение WPF с документом XPS, который отображается с помощью элемента `DocumentViewer`, представлено на рис. 18.15.

Элемент `DocumentViewer` также дает возможность пользователям изменять способ просмотра загруженного документа, осуществлять поиск и печать документов XPS.

Создание проектов XBAP

Приложения XBAP (XAML Browser Application, браузерное приложение XAML) — это приложения, которые выполняются внутри браузера. Приложения XBAP являются полноценными приложениями Windows Presentation Foundation, но имеют несколько основных отличий.

- XBAP-приложения выполняются внутри окна браузера.
- XBAP-приложения имеют ограниченные права.
- Приложению XBAP предоставляются те же разрешения, что и приложению .NET, которое запускается из веб. По умолчанию приложение XBAP, например, не может записывать файлы, взаимодействовать с реестром, подключаться к базам данных.
- XBAP-приложения не инсталлируются. При запуске приложения XBAP это приложение загружается и помещается в кэш браузера. Однако инсталлированным на компьютере оно не остается.

В приложение XBAP можно преобразовать любое автономное приложение WPF, хотя Visual Studio для создания такого приложения вынуждает создавать новый проект с помощью шаблона **WPF Browser Application** (рис. 18.16).

Приложение XBAP создается точно так же, как и автономное приложение WPF, можно даже использовать код из созданных ранее автономных приложений.

Для примера создадим приложение XBAP и разместим в нем двумерные графические формы, созданные нами в предыдущем разделе. Код разметки окна представлен в листинге 18.20.

Примечание

Полный код примера приложения находится в каталоге Chapter18\BrowserApp на прилагаемом к книге диске.

Листинг 18.20. Файл MainWindow.xaml

```
<Page x:Class="BrowserApp.Page1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="300" d:DesignWidth="300"
      Title="Page1">
```

```
    <Grid>
        <Ellipse Height="62" Margin="30,12,153,125" Name="ellipse1"
            Stroke="Black" Width="117" Fill="#FFFF0000"
            MouseUp="shape_MouseUp" />
        <Rectangle Height="69" Margin="30,95,153,35" Name="rectangle1"
             Stroke="#FF06FF00" Width="117" MouseUp="shape_MouseUp"
             StrokeThickness="5">
            <Rectangle.Fill>
                <ImageBrush
                    ImageSource="/BrowserApp;component/Images/Hydrangeas.jpg"/>
            </Rectangle.Fill>
        </Rectangle>
        <Ellipse Height="61" Margin="204,12,35,126" Name="ellipse2"
            Stroke="Red" Width="61" StrokeThickness="3" Fill="Blue"
            MouseUp="shape_MouseUp"/>
        <Rectangle Height="70" Margin="182,94,9,35" Name="rectangle2"
             Stroke="#FF000019" Width="109" RadiusX="20" RadiusY="20"
             Fill="#FF00FF00" StrokeThickness="10" MouseUp="shape_MouseUp"/>
        <Label Height="28" Margin="-1,170,1,0" Name="label1"
             HorizontalContentAlignment="Center" />
    </Grid>
</Page>
```

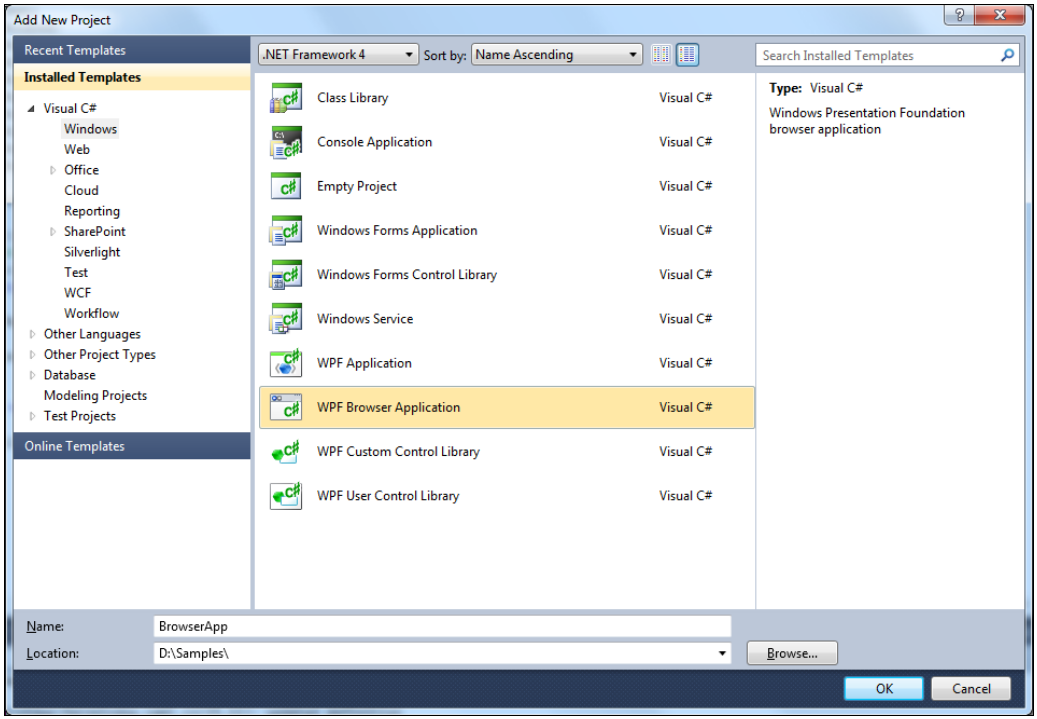


Рис. 18.16. Создание проекта приложения XBAP

При запуске приложение XBAP загружается в браузер и выглядит, как показано на рис. 18.17.

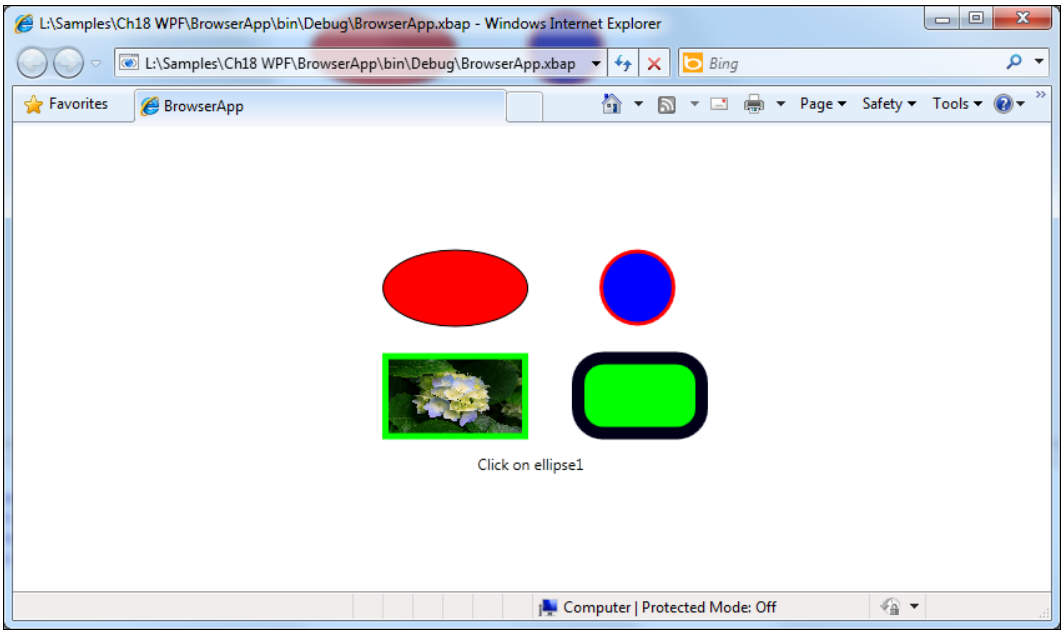


Рис. 18.17. Окно проиложения WPF в Internet Explorer

Привязка данных

Мы не будем подробно рассматривать привязку данных в WPF, ее основные принципы такие же, как и в Windows Forms и ASP.NET. Привязка данных WPF позволяет извлекать информацию практически из любого свойства того или иного элемента. WPF также включает набор списочных элементов управления, которые могут обрабатывать целые коллекции информации и позволяют осуществлять навигацию по этой информации.

Многие концепции привязки данных остались прежними, но код изменился. Например, для привязки данных к элементу `DataGrid` (аналог элемента `DataGridView` в Windows Forms) используется свойство `ItemsSource`. Кроме того, для элемента `DataGrid` необходимо установить свойство `AutoGenerateColumns` в `true`.

Пример приложения с элементом `DataGrid` приведен в листингах 18.21 и 18.22.

Примечание

Полный код примера приложения находится в каталоге Chapter18\DataBind на прилагаемом к книге диске.

Листинг 18.21. Файл MainWindow.xaml

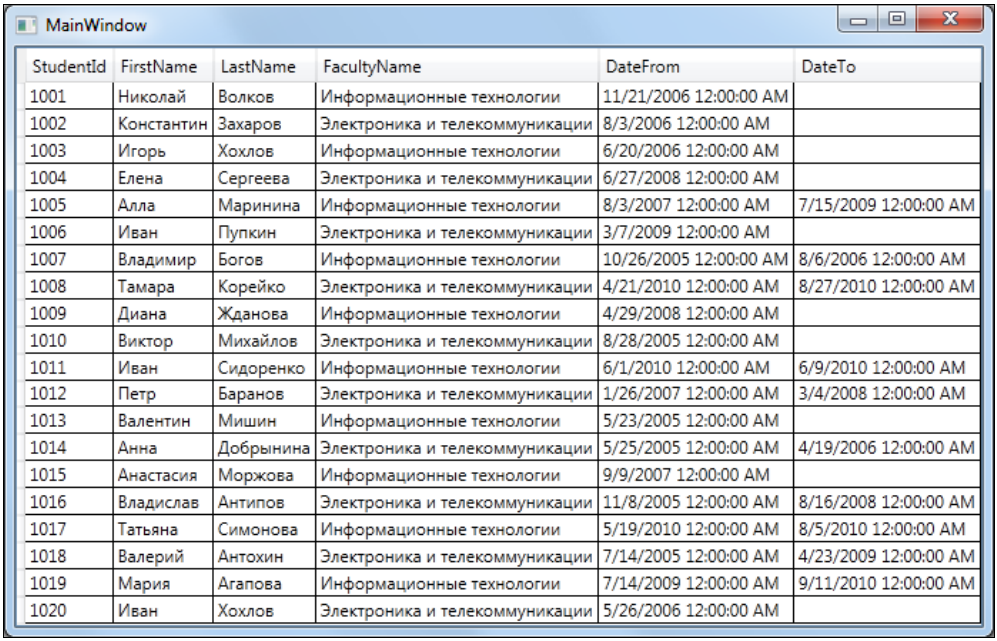
```
<Window x:Class="DataBind.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="300" Width="500" DataContext="{Binding}"
         Loaded="Window_Loaded">

    <Grid>
      <Grid.RowDefinitions>
         <RowDefinition></RowDefinition>
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
         <ColumnDefinition></ColumnDefinition>
      </Grid.ColumnDefinitions>
      <DataGrid Grid.Row="0" Grid.Column="0" AutoGenerateColumns="True"
         Name="dataGrid1" />
   </Grid>
</Window>
```

Листинг 18.22. Файл MainWindow.xaml.cs

```
using College.Data;

namespace DataBind
{
   public partial class MainWindow : Window
   {
      private College.Data.CollegeEntities collegeEnt;

      public MainWindow()
      {
         InitializeComponent();
      }

      private void Window_Loaded(object sender, RoutedEventArgs e)
      {
         collegeEnt = new CollegeEntities();
         dataGrid1.ItemsSource = from c in collegeEnt.Students
                                 select new
                                 {
                                    c.StudentId,
                                    c.FirstName,
                                    c.LastName,
                                    c.Faculty.FacultyName,
```

```
                                    c.DateFrom,
                                    c.DateTo
                                };
        }
    }
}
```

На рис. 18.18 показано это приложение с использованием элемента `DataGrid` для отображения данных из таблицы `Student` нашей базы данных `College`.



StudentId	FirstName	LastName	FacultyName	DateFrom	DateTo
1001	Николай	Волков	Информационные технологии	11/21/2006 12:00:00 AM	
1002	Константин	Захаров	Электроника и телекоммуникации	8/3/2006 12:00:00 AM	
1003	Игорь	Хохлов	Информационные технологии	6/20/2006 12:00:00 AM	
1004	Елена	Сергеева	Электроника и телекоммуникации	6/27/2008 12:00:00 AM	
1005	Алла	Маринина	Информационные технологии	8/3/2007 12:00:00 AM	7/15/2009 12:00:00 AM
1006	Иван	Пупкин	Электроника и телекоммуникации	3/7/2009 12:00:00 AM	
1007	Владимир	Богов	Информационные технологии	10/26/2005 12:00:00 AM	8/6/2006 12:00:00 AM
1008	Тамара	Корейко	Электроника и телекоммуникации	4/21/2010 12:00:00 AM	8/27/2010 12:00:00 AM
1009	Диана	Жданова	Информационные технологии	4/29/2008 12:00:00 AM	
1010	Виктор	Михайлов	Электроника и телекоммуникации	8/28/2005 12:00:00 AM	
1011	Иван	Сидоренко	Информационные технологии	6/1/2010 12:00:00 AM	6/9/2010 12:00:00 AM
1012	Петр	Баранов	Электроника и телекоммуникации	1/26/2007 12:00:00 AM	3/4/2008 12:00:00 AM
1013	Валентин	Мишин	Информационные технологии	5/23/2005 12:00:00 AM	
1014	Анна	Добрынина	Электроника и телекоммуникации	5/25/2005 12:00:00 AM	4/19/2006 12:00:00 AM
1015	Анастасия	Моржова	Информационные технологии	9/9/2007 12:00:00 AM	
1016	Владислав	Антипов	Электроника и телекоммуникации	11/8/2005 12:00:00 AM	8/16/2008 12:00:00 AM
1017	Татьяна	Симонова	Информационные технологии	5/19/2010 12:00:00 AM	8/5/2010 12:00:00 AM
1018	Валерий	Антохин	Электроника и телекоммуникации	7/14/2005 12:00:00 AM	4/23/2009 12:00:00 AM
1019	Мария	Агапова	Информационные технологии	7/14/2009 12:00:00 AM	9/11/2010 12:00:00 AM
1020	Иван	Хохлов	Электроника и телекоммуникации	5/26/2006 12:00:00 AM	

Рис. 18.18. Приложение WPF с привязкой данных

2D-графика

Библиотека Windows Presentation Foundation предоставляет обширный, масштабируемый и гибкий набор возможностей для работы с графикой, которые имеют определенные преимущества.

- *Графика, не зависящая от разрешения и устройства.* Основной единицей измерения в графической системе WPF является аппаратно-независимая точка, которая составляет 1/96 часть дюйма независимо от фактического разрешения экрана и предоставляет основу для создания изображения, независимого от разрешения и устройства. Каждый аппаратно-независимый пиксел автоматически масштабируется в соответствии с числом точек на дюйм в системе, в которой он отображается.

- *Повышенная точность.* В системе координат WPF используются числа с плавающей запятой двойной точности, вместо одиночной точности. Значения преобразований и прозрачности также выражаются с помощью чисел двойной точности. Кроме того, WPF поддерживает расширенную цветовую палитру и предоставляет встроенную поддержку для управления входными данными из различных цветовых пространств. Дополнительная поддержка графики и анимацииWPF упрощает программирование графики за счет автоматического управления анимацией.
- *Аппаратное ускорение.* Графическая система WPF использует преимущества графического оборудования, чтобы уменьшить загрузку центрального процессора.

Двумерные формы

Windows Presentation Foundation предоставляет библиотеку общих двумерных фигур, нарисованных с помощью векторов, таких как прямоугольники и эллипсы. Их можно создавать непосредственно в коде XAML, например, так, как показано в листинге 18.23.

ПРИМЕЧАНИЕ

Полный код примера приложения находится в каталоге Chapter18\Shapes2D на прилагаемом к книге диске.

Листинг 18.23. Файл MainWindow.xaml

```
<Window x:Class="Shapes2D.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="238" Width="335">
    <Grid>
      <Ellipse Height="62" Margin="31,12,165,125" Name="ellipse1"
         Stroke="Black" Width="117" Fill="#FFFF0000"
         MouseUp="shape_MouseUp" />
      <Rectangle Height="69" Margin="31,95,165,35" Name="rectangle1"
         Stroke="#FF06FF00" Width="117" StrokeThickness="5"
         MouseUp="shape_MouseUp">
         <Rectangle.Fill><ImageBrush ImageSource=
            "/Shapes2D;component/Images/Hydrangeas.jpg"/>
         </Rectangle.Fill>
      </Rectangle>
      <Ellipse Height="61" Margin="205,12,46,126" Name="ellipse2"
         Stroke="Red" Width="61" StrokeThickness="3" Fill="Blue"
         MouseUp="shape_MouseUp"/>
      <Rectangle Height="70" Margin="183,94,21,35" Name="rectangle2"
         Stroke="#FF000019" Width="109" RadiusX="20" RadiusY="20"
         Fill="#FF00FF00" StrokeThickness="10" MouseUp="shape_MouseUp"/>
```

```
      <Label Height="28" Margin="0,170,0,0" Name="label1"
         HorizontalContentAlignment="Center" />
   </Grid>
</Window>
```

Интересная особенность фигур в том, что они могут не только отображаться; фигуры реализовывают многие возможности, ожидаемые от элементов управления, включая ввод с клавиатуры и ввод с помощью мыши. В листинге 18.24 показан код программной части класса `MainWindow` с реализацией события `MouseUp()`, обрабатываемого размещенными в окне элементами.

Листинг 18.24. Файл MainWindow.xaml.cs

```
namespace Shapes2D
{
   public partial class MainWindow : Window
   {
      public MainWindow()
      {
         InitializeComponent();
      }

      private void shape_MouseUp(object sender, MouseButtonEventArgs e)
      {
         label1.Content = "Click on " + ((Shape)sender).Name;
      }
   }
}
```

На рис. 18.19 показан набор двумерных фигур, создаваемых с помощью приведенного в листинге 18.24 кода.

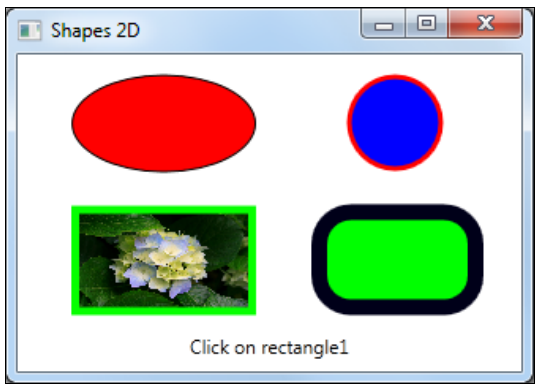


Рис. 18.19. Двумерные формы с подключенными обработчиками событий

Шаблоны элементов управления

Создание шаблонов элементов управления — одна из многих возможностей, предоставляемых моделью создания стилей и шаблонов WPF. Модель создания стилей и шаблонов обеспечивает такую гибкость, что в большинстве случаев писать собственные элементы управления не будет необходимости.

В пространстве имен `System.Windows.Controls` есть класс `ControlTemplate`. Он задает визуальную структуру и аспекты поведения `Control`, которые могут совместно использоваться несколькими экземплярами элемента управления.

Пользовательские интерфейсы по умолчанию для элементов управления WPF обычно создаются из других элементов управления и фигур. Например, `Button` состоит из элементов управления `ButtonChrome` и `ContentPresenter`. Класс `ButtonChrome` обеспечивает стандартный внешний вид кнопки, в то время как класс `ContentPresenter` отображает содержимое кнопки, заданное свойством `Content`. С помощью класса `ControlTemplate` можно изменить внешний вид элемента управления без изменения его содержимого и поведения.

В следующем примере, приведенном в листинге 18.25, показано изменение внешнего вида элемента `Button` с помощью объекта `ControlTemplate`.

Примечание

Полный код примера приложения находится в каталоге Chapter18\CustomControl на прилагаемом к книге диске.

Листинг 18.25. Создание шаблона элемента управления

```
<Window x:Class="ControlTemplate.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="ControlTemplate" Height="144" Width="249">
   <Grid>
      <Grid.RowDefinitions>
         <RowDefinition Height="153*" />
         <RowDefinition Height="155*" />
      </Grid.RowDefinitions>
      <Button Content="ON" Height="72" Width="72" Margin="31,16,124,17"
            Grid.RowSpan="2">
         <Button.Template>
            <ControlTemplate TargetType="{x:Type Button}">
               <Grid Margin="5">
                  <Ellipse Stroke="DarkBlue" StrokeThickness="2">
                     <Ellipse.Fill>
                        <RadialGradientBrush Center="0.3,0.2"
                              RadiusX="0.5"
                              RadiusY="0.5">
                           <GradientStop Color="Azure" Offset="0.1" />
                           <GradientStop Color="Green" Offset="1.1" />
```

```
                    </RadialGradientBrush>
                 </Ellipse.Fill>
              </Ellipse>
              <ContentPresenter Name="content"
                 HorizontalAlignment="Center"
                 VerticalAlignment="Center"/>
           </Grid>
        </ControlTemplate>
     </Button.Template>
  </Button>
  <Button Content="OFF" Height="72" Margin="122,16,33,17"
         Width="72" Grid.RowSpan="2">
     <Button.Template>
        <ControlTemplate TargetType="{x:Type Button}">
           <Grid Margin="5">
              <Ellipse Stroke="DarkBlue" StrokeThickness="2">
                 <Ellipse.Fill>
                    <RadialGradientBrush Center="0.3,0.2"
                          RadiusX="0.5"
                          RadiusY="0.5">
                       <GradientStop Color="Azure" Offset="0.1" />
                       <GradientStop Color="Red" Offset="1.1" />
                    </RadialGradientBrush>
                 </Ellipse.Fill>
              </Ellipse>
              <ContentPresenter HorizontalAlignment="Center"
                 Name="content" VerticalAlignment="Center" />
           </Grid>
        </ControlTemplate>
     </Button.Template>
  </Button>
 </Grid>
</Window>
```

Скомпилируйте и запустите приложение. В результате у нас получилось окно с двумя красивыми кнопками (рис. 18.20).

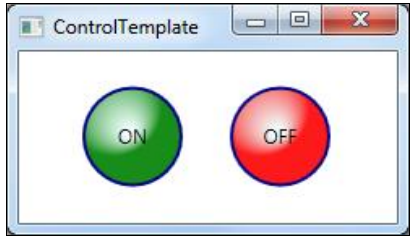


Рис. 18.20. Измененный вид стандартных кнопок

3D-графика

Windows Presentation Foundation предлагает новую трехмерную графическую модель. Используя WPF, вы сможете строить сложные трехмерные сцены на основе понятного кода разметки.

Трехмерная графика в WPF включает в себя:

- окно просмотра (viewport) с вашим трехмерным содержимым;
- трехмерный объект;
- источник света, освещающий часть вашей трехмерной сцены;
- камеру — точку наблюдения трехмерной сцены.

Окно просмотра

Для работы с 3D-графикой нужен контейнерный объект, в котором будет размещен сам графический объект. Этот контейнер представлен классом `Viewport3D`, находящимся в пространстве имен `System.Windows.Controls`. Класс `Viewport3D` унаследован от `FrameworkElement` и потому может находиться везде, где размещается любой обычный элемент WPF. Например, класс `Viewport3D` можно использовать в качестве содержимого окна WPF-приложения.

```
<Viewport3D>
  <ModelVisual3D>...</ModelVisual3D>
  <ModelVisual3D>...</ModelVisual3D>
  <ModelVisual3D>...</ModelVisual3D>
</Viewport3D>
```

Класс `Viewport3D` предоставляет два дополнительных свойства:

- `Camera` — определяет точку наблюдения;
- `Children` — представляет коллекцию трехмерных объектов на сцене.

Окно просмотра может содержать в себе любой трехмерный объект, унаследованный от `Visual3D` из пространства имен `System.Windows.Media.Media3D`.

Класс `Visual3D` является базовым классом для всех трехмерных объектов. Подобно классу `Visual`, можно использовать класс `Visual3D` для наследования от него трехмерных фигур или создания трехмерных элементов управления.

Материал поверхности

Поверхность важна по двум причинам. Во-первых, она определяет цвет объекта (хотя вы можете использовать и более сложные кисти, рисующие текстуры вместо сплошного цвета). Во-вторых, она определяет то, как материал реагирует на свет. WPF включает четыре класса материалов, каждый из которых наследуется от абстрактного класса `Material` из пространства имен `System.Windows.Media.Media3D`.

- `DiffuseMaterial` — создает плоскую матовую поверхность, распределяющую свет равномерно во всех направлениях;

- `SpecularMaterial` — создает блестящую бесцветную поверхность (типа металла или стекла). Отражает свет в противоположном направлении подобно зеркалу;
- `EmissiveMaterial` — создает раскаленную поверхность, которая сама излучает свет (хотя этот свет не отражают другие объекты сцены);
- `MaterialGroup` — комбинирует несколько материалов. Комбинируемые материалы накладываются друг на друга в том порядке, в каком были добавлены в объект `MaterialGroup`.

Класс `DiffuseMaterial` применяется наиболее часто, поскольку его поведение ближе всего к реальным поверхностям. Он предоставляет свойство `Brush`, принимающее объект `Brush`, который вы хотите использовать для рисования поверхности трехмерного объекта.

```
<DiffuseMaterial>
    <DiffuseMaterial.Brush>
        <LinearGradientBrush>
            <GradientStop Offset="0.0" Color="Green" />
            <GradientStop Offset="1.0" Color="Yellow" />
        </LinearGradientBrush>
    </DiffuseMaterial.Brush>
</DiffuseMaterial>
```

Объект класса `SpecularMaterial` имеет сильную отражающую способность. Вы можете управлять яркостью отраженного света через свойство `SpecularPower`. Малое значение `SpecularPower` дает размытый, рассеянный эффект, в то время как большое его значение — яркий, резкий.

Объект класса `EmissiveMaterial` сам является источником света, однако свет, исходящий от `EmissiveMaterial`, не подсвечивает другие объекты. Для освещения объектов используют специальные объекты, например `PointLight`.

Источники света

Источник света может быть один, бывает и несколько источников. Характер освещения зависит от выбранного типа источника света, его положения, направления и интенсивности.

WPF предлагает четыре класса источников света, каждый из которых унаследован от абстрактного класса `Light`:

- `DirectionalLight` — заполняет сцену параллельными лучами света, идущими в указанном вами направлении;
- `AmbientLight` — наполняет сцену рассеянным светом;
- `PointLight` — свет распространяется во все стороны из точечного источника;
- `SpotLight` — свет распространяется из одной точки по конусу.

Вот пример наполнения сцены желтым рассеянным светом:

```
<AmbientLight Color="Yellow"></AmbientLight>
```

В данном примере мы ограничимся только одним DirectionalLight, который представляет наиболее распространенный тип освещения.

```
<DirectionalLight Color="Red" Direction="3,2,1" />
```

Камера

Камера определяет способ проекции трехмерной сцены на двумерную поверхность окна отображения. WPF включает три класса камер:

- PerspectiveCamera;
- OrchcgrapnicCamera;
- MatrixCamera.

Чаще всего используют класс PerspectiveCamera. Камера такого типа отображает сцену так, что объекты, находящиеся дальше, всегда выглядят меньше, чем расположенные ближе к камере.

Для использования камеры ее надо правильно расположить и сконфигурировать. Для позиционирования камеры служит свойство Position. Для ориентации камеры — свойство LookDirection. Наклон камеры определяет свойство UpDirection. По умолчанию свойство UpDirection имеет значение (0, 1, 0), т. е. камера направлена вертикально вверх.

Вот пример фрагмента кода, описывающий реализацию объекта Camera:

```
<Viewport3D.Camera>
    <PerspectiveCamera
        LookDirection="2,-1,-1"
        Position="-15,15,15"
        UpDirection="1,1,0">
    </PerspectiveCamera>
</Viewport3D.Camera>
```

Построение геометрической модели

Чтобы построить трехмерный объект, надо начать с построения геометрии. Как вы уже знаете, для этой цели используется класс MeshGeometry3D. Класс MeshGeometry3D представляет сетку. Если раньше вы имели дело с трехмерным рисованием (или читали что-нибудь о технологиях, положенных в основу современных видеокарт), то уже должны знать, что компьютеры предпочитают строить трехмерные объекты из треугольников. Это объясняется тем, что треугольники — это простейший, наиболее подробный способ описания поверхности. Треугольники просты, потому что каждый из них определяется всего тремя точками их вершин.

Треугольники удобно применять для построения всех фигур с прямыми гранями, например квадраты, прямоугольники могут быть представлены треугольниками.

Большинство трехмерных объектов не будут выглядеть как простые плоские треугольники. Вместо этого вам придется комбинировать треугольники для их по-

строения — иногда всего несколько, но чаще — сотни и тысячи, соединенных друг с другом под разными углами. Такая комбинация треугольников образует пространственную сетку. При достаточном количестве треугольников вы можете, в конечном счете, создать иллюзию чего угодно, включая наиболее сложные поверхности. Разумеется, следует учитывать фактор производительности, к тому же трехмерные сцены часто используют битовые карты или двумерную графику в треугольниках объемной сетки, создавая иллюзию сложной поверхности с минимальными накладными расходами. WPF поддерживает эту технику.

Представление о том, как определяется сетка, является одним из ключевых моментов трехмерного программирования. Определив нужное свойство `MeshGeometry3D`, его нужно поместить в оболочку объекта `GeometryModel3D`.

Класс `GeometryModel3D` имеет три свойства:

- `Geometry`
- `Material`
- `BackMaterial`

Свойство `Geometry` принимает `MeshGeometry3D`, определяющий фигуру вашего трехмерного объекта. В дополнение вы можете применять свойства `Material` и `BackMaterial` для определения поверхностей, из которых состоит ваша фигура.

Куб состоит из шести квадратных сторон. Каждая квадратная сторона требует для своего отображения два треугольника.

Для повышения производительности в программе, работающей с трехмерной графикой, обычно не визуализируют невидимые части фигур, например заднюю сторону куба. Однако в данном примере мы определим все стороны, так так нам потребуется вращать куб во все стороны.

Затем необходимо установить свойство `Normals`, заполнив его векторами. Свойство `Normals` позволяет сгладить цвета соседних треугольников, образующих общий прямоугольник, за счет разделения нормалей. Соседние треугольники на поверхности куба разделяют две общие точки. Поэтому нужно подкорректировать освещенность только тех двух точек, которые не являются общими. Пока они соответствуют друг другу, затенение будет согласовано по всей общей поверхности.

С учетом всех определений свойств, необходимых для построения трехмерной модели, код объекта `MeshGeometry3D`, описывающий геометрическую модель куба, приведен в листинге 18.26.

Листинг 18.26. Объект `MeshGeometry3D`, описывающий куб

```
<GeometryModel3D.Geometry>
    <MeshGeometry3D
        Positions=
            "0,0,0  10,0,0   0,10,0   10,10,0
            0,0,0  0,0,10   0,10,0   0,10,10
            0,0,0  10,0,0   0,0,10   10,0,10
            10,0,0 10,10,10 10,0,10  10,10,0
            0,0,10 10,0,10  0,10,10  10,10,10
            0,10,0 0,10,10  10,10,0  10,10,10"
```

```
        TriangleIndices=
            "0,2,1 1,2,3
            4,5,6 6,5,7
            8,9,10 9,11,10
            12,13,14 12,15,13
            16,17,18 19,18,17
            20,21,22 22,21,23"
        TextureCoordinates=
            "0,0 0,1 1,0 1,1
            1,1 0,1 1,0 0,0
            0,0 1,0 0,1 1,1
            0,0 1,0 0,1 1,1
            1,1 0,1 1,0 0,0
            1,1 0,1 1,0 0,0"/>
</GeometryModel3D.Geometry>
```

В листинге 18.26 коллекция `Positions` определяет вершины куба. Она начинается с четырех точек задней стороны (где z = 0), а затем добавляет четыре точки передней стороны (где z = 10).

Свойство `TriangleIndices` отображает эти точки на треугольники. При отображении треугольников их надо определять в направлении против часовой стрелки, чтобы их лицевая сторона смотрела вперед. Однако для куба треугольники для задней стороны должны быть определены в противоположном направлении — по часовой стрелке. Квадраты передней стороны определяются в порядке против часовой стрелки — индексы TriangleIndices 4, 5, 6 и 7, 6, 5, но поверхность задней стороны куба описана по часовой стрелке, включая индексы 0, 2, 1 и 1, 2, 3. Это объясняется тем, что обратная сторона куба должна обращать свою лицевую сторону назад. Чтобы лучше представить это, предположим, что наш куб будет вращаться вокруг оси Y, так что обратная сторона переместится вперед. Теперь треугольники, которые до этого находились на задней стороне куба, будут повернуты вперед, что сделает их видимыми.

Теперь соберем все вместе, создав приложение, отображающее трехмерную модель куба, код которого представлен в листинге 18.27.

> ***Примечание***
>
> Полный код примера приложения находится в каталоге Chapter18\Shapes3D_Cube на прилагаемом к книге диске.

Листинг 18.27. Класс MainWindow.xaml

```
<Window x:Class="Shapes3D.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Rotate cube" Height="393" Width="410">
   <Grid>
      <Border BorderThickness="1" Margin="5">
```

```
<Viewport3D>
   <Viewport3D.Camera>
      <PerspectiveCamera
           LookDirection="2,-1,-1"
           Position="-15,15,15"
           UpDirection="0,1,0">
      </PerspectiveCamera>
   </Viewport3D.Camera>
   <ModelVisual3D>
      <ModelVisual3D.Content>
         <AmbientLight Color="Yellow"></AmbientLight>
      </ModelVisual3D.Content>
   </ModelVisual3D>
   <ModelVisual3D>
      <ModelVisual3D.Content>
         <DirectionalLight Color="Red" Direction="3,2,1" />
      </ModelVisual3D.Content>
   </ModelVisual3D>
   <ModelVisual3D>
      <ModelVisual3D.Content>
         <GeometryModel3D x:Name="cubeGeometry">
            <GeometryModel3D.Geometry>
               <MeshGeometry3D
                  Positions=
                    "0,0,0  10,0,0   0,10,0   10,10,0
                     0,0,0  0,0,10   0,10,0   0,10,10
                     0,0,0  10,0,0   0,0,10   10,0,10
                     10,0,0 10,10,10 10,0,10  10,10,0
                     0,0,10 10,0,10  0,10,10  10,10,10
                     0,10,0 0,10,10  10,10,0  10,10,10"
                  TriangleIndices=
                    "0,2,1 1,2,3
                     4,5,6 6,5,7
                     8,9,10 9,11,10
                     12,13,14 12,15,13
                     16,17,18 19,18,17
                     20,21,22 22,21,23"
                  TextureCoordinates=
                    "0,0 0,1 1,0 1,1
                     1,1 0,1 1,0 0,0
                     0,0 1,0 0,1 1,1
                     0,0 1,0 0,1 1,1
                     1,1 0,1 1,0 0,0
                     1,1 0,1 1,0 0,0"/>
            </GeometryModel3D.Geometry>
            <GeometryModel3D.Material>
               <DiffuseMaterial>
```

```
                        <DiffuseMaterial.Brush>
                           <LinearGradientBrush>
                              <GradientStop
                                 Offset="0.0" Color="Green" />
                              <GradientStop
                                 Offset="1.0" Color="Yellow" />
                           </LinearGradientBrush>
                        </DiffuseMaterial.Brush>
                     </DiffuseMaterial>
                  </GeometryModel3D.Material>
               </GeometryModel3D>
            </ModelVisual3D.Content>
         </ModelVisual3D>
      </Viewport3D>
   </Border>
</Grid>
</Window>
```

Скомпилируйте и запустите приложение. В результате у нас получился красивый куб, представленный на рис. 18.21.

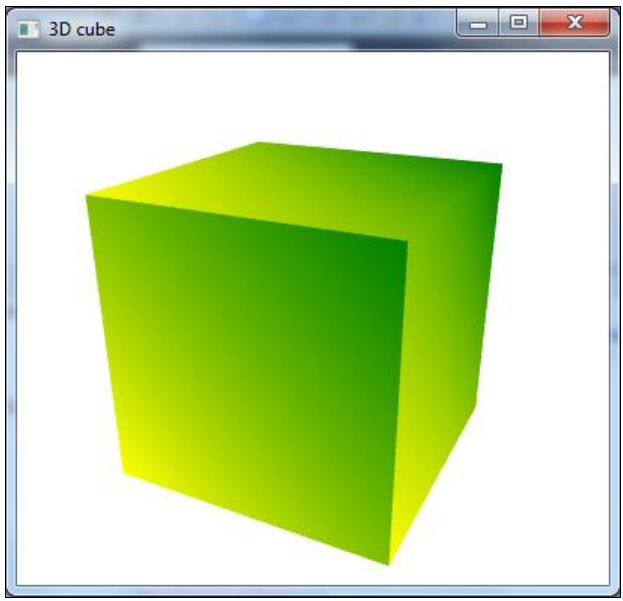


Рис. 18.21. Трехмерный куб

Вращения

Рассмотрим теперь повороты трехмерных объектов. В библиотеке WPF для этого предназначен класс `RotateTransform3D`, который позволяет вращать трехмерный объект вокруг указанных осей.

Для поворота двумерного объекта обычно используют два аргумента: точку, вокруг которой вращается объект, и значение, задающее величину поворота. Для трехмерного объекта требуется указать объект `Vector3D`, а не точку, вокруг которой производится вращение.

Повороты для трехмерного объекта становятся еще сложнее, если требуется анимация поворотов. Чтобы анимировать трехмерный поворот с помощью объекта `RotateTransform3D`, надо произвести анимацию свойства `Angle` или `Axis` объекта `AxisAngleRotation3D` или указать объект `QuaternionRotation3D` для преобразования.

Примечание

Поворот для трехмерного объекта в WPF задается в градусах, а не в радианах.

Класс `AxisAngleRotation3D` представляет 3D-поворот на заданный угол относительно указанной оси:

```
<ModelVisual3D.Transform>
   <RotateTransform3D>
      <RotateTransform3D.Rotation>
         <AxisAngleRotation3D x:Name="rotation"/>
      </RotateTransform3D.Rotation>
   </RotateTransform3D>
</ModelVisual3D.Transform>
```

Используя именованное вращение, вы можете создать привязанный к данным бегунок `Slider`, который позволит пользователям поворачивать куб вокруг его осей:

```
<Slider Grid.Row="1"
   Minimum="0" Maximum="360" Orientation="Horizontal"
   Value="{Binding ElementName=rotation, Path=Angle}" >
</Slider>
```

Столь же легко вы можете использовать это вращение в анимации. Рассмотрим пример приложения WPF с анимацией, вращающей куб вокруг разных осей, направление и поворот которых можно задавать и изменять во время выполнения программы.

В следующем примере показано, как заставить трехмерный объект вращаться с помощью анимации `Axis` объекта `AxisAngleRotation3D`. Объект `AxisAngleRotation3D` задает преобразование вращения трехмерного объекта, поэтому анимирование его свойств приведет к созданию желаемого эффекта вращения. В раскадровке `DoubleAnimation` используется для анимации свойства `Angle`, в то время как `Vector3DAnimation` используется для анимации свойства `Axis`.

Код класса MainWindow.xaml представлен в листинге 18.28.

Примечание

Полный код примера приложения находится в каталоге Chapter18\Shapes3D_Rotation на прилагаемом к книге диске.

Листинг 18.28. Код класса MainWindow.xaml

```
<Window x:Class="Shapes3D.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Rotate cube" Height="500" Width="410">
   <Grid>
      <Grid.RowDefinitions>
         <RowDefinition></RowDefinition>
         <RowDefinition Height="90"></RowDefinition>
      </Grid.RowDefinitions>
      <Border BorderThickness="1" Margin="5">

         <Viewport3D>
            <Viewport3D.Camera>
               <PerspectiveCamera
                   LookDirection="2,-1,-1"
                   Position="-15,15,15"
                   FieldOfView="70">
               </PerspectiveCamera>
            </Viewport3D.Camera>

            <ModelVisual3D>
               <ModelVisual3D.Content>
                  <AmbientLight Color="Gray"></AmbientLight>
               </ModelVisual3D.Content>
            </ModelVisual3D>

            <ModelVisual3D>
               <ModelVisual3D.Content>
                  <DirectionalLight Color="White" Direction="3,2,1" />
               </ModelVisual3D.Content>
            </ModelVisual3D>

            <ModelVisual3D>
               <ModelVisual3D.Content>
                  <GeometryModel3D x:Name="cubeGeometry">
                     <GeometryModel3D.Geometry>
                        <MeshGeometry3D
                           Positions=
                             "0,0,0  10,0,0   0,10,0   10,10,0
                              0,0,0  0,0,10   0,10,0   0,10,10
                              0,0,0  10,0,0   0,0,10   10,0,10
                              10,0,0 10,10,10 10,0,10  10,10,0
                              0,0,10 10,0,10  0,10,10  10,10,10
                              0,10,0 0,10,10  10,10,0  10,10,10"
```

```
                        TriangleIndices=
                           "0,2,1 1,2,3
                            4,5,6 6,5,7
                            8,9,10 9,11,10
                            12,13,14 12,15,13
                            16,17,18 19,18,17
                            20,21,22 22,21,23"
                        TextureCoordinates=
                           "0,0 0,1 1,0 1,1
                            1,1 0,1 1,0 0,0
                            0,0 1,0 0,1 1,1
                            0,0 1,0 0,1 1,1
                            1,1 0,1 1,0 0,0
                            1,1 0,1 1,0 0,0"/>
                  </GeometryModel3D.Geometry>
                  <GeometryModel3D.Material>
                     <DiffuseMaterial>
                        <DiffuseMaterial.Brush>
                           <LinearGradientBrush>
                              <GradientStop Offset="0.0"
                                Color="Green"/>
                              <GradientStop Offset="1.0"
                                Color="Yellow"/>
                           </LinearGradientBrush>
                        </DiffuseMaterial.Brush>
                     </DiffuseMaterial>
                  </GeometryModel3D.Material>
               </GeometryModel3D>
            </ModelVisual3D.Content>

            <ModelVisual3D.Transform>
               <RotateTransform3D>
                  <RotateTransform3D.Rotation>
                     <AxisAngleRotation3D x:Name="rotation"/>
                  </RotateTransform3D.Rotation>
               </RotateTransform3D>
            </ModelVisual3D.Transform>

         </ModelVisual3D>
      </Viewport3D>
   </Border>
   <Slider Margin="83,35,10,13" Grid.Row="1"
      Minimum="0" Maximum="360" Orientation="Horizontal"
           Value="{Binding ElementName=rotation, Path=Angle}" >
   </Slider>
   <TextBox Grid.Row="1" HorizontalAlignment="Left" Margin="30,6,0,61"
      Name="textBoxX" Width="41"
      TextChanged="textBox_TextChanged" Text="0" />
```

```
      <TextBox Grid.Row="1" HorizontalAlignment="Left" Margin="30,33,0,0"
         Name="textBoxY" VerticalAlignment="Top" Width="41"
         TextChanged="textBox_TextChanged" Text="1" />
      <TextBox Grid.Row="1" HorizontalAlignment="Left" Margin="30,61,0,0"
         Name="textBoxZ" VerticalAlignment="Top" Width="41"
         TextChanged="textBox_TextChanged" Text="0" />
      <Label Content="X" HorizontalAlignment="Left" Margin="10,6,0,0"
         Name="label1" VerticalAlignment="Top" Width="20" Grid.Row="1" />
      <Label Content="Y" HorizontalAlignment="Left" Margin="10,33,0,0"
         Name="label2" VerticalAlignment="Top" Width="20" Grid.Row="1" />
      <Label Content="Z" HorizontalAlignment="Left" Margin="10,61,0,0"
         Name="label3" VerticalAlignment="Top" Width="20" Grid.Row="1" />
   </Grid>
</Window>
```

Свойство `Axis` получает или задает ось 3D-поворота объекта. Для изменения этого свойства надо инициализировать новый экземпляр структуры `Vector3D`. Код для изменения свойства `Axis` реализован в обработчике события `TextChanged()`, обрабатывающем событие изменения содержимого в текстовых полях `textBoxX`, `textBoxY`, `textBoxZ`. Этот код представлен в листинге 18.29.

Листинг 18.29. Обработчик события `TextChanged` для полей ввода

```
private void textBox_TextChanged(object sender, TextChangedEventArgs e)
{
   try
   {
     rotation.Axis = new Vector3D(
            Int32.Parse(textBoxX.Text),
            Int32.Parse(textBoxY.Text),
            Int32.Parse(textBoxZ.Text));
   }
   catch
   {
     ((TextBox)sender).Text = "1";
   }
}
```

Скомпилируйте и запустите приложение. В результате у нас получился куб, представленный на рис. 18.22.

Программирование трехмерной графики — очень объемная тема. Здесь были представлены только основы трехмерной графики и классы WPF для работы с графикой. При желании вы также можете найти много книг, посвященных программированию трехмерной графики.

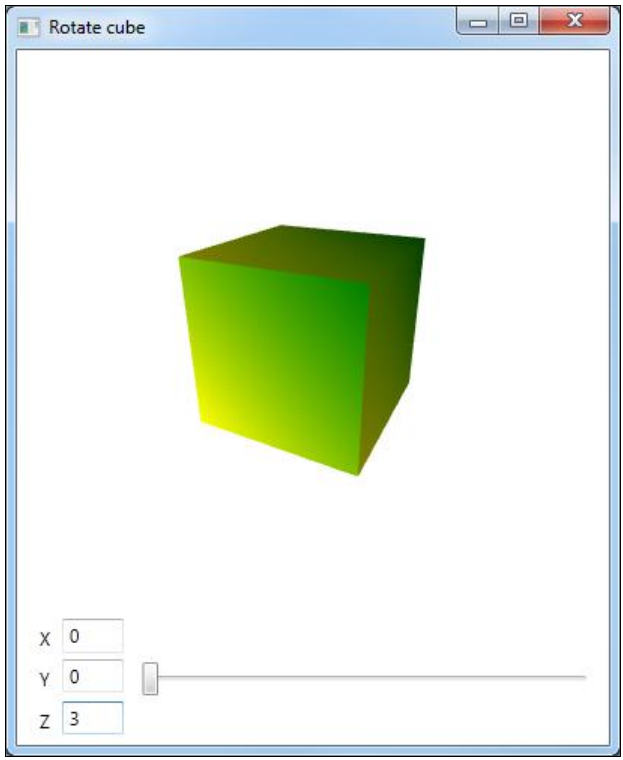
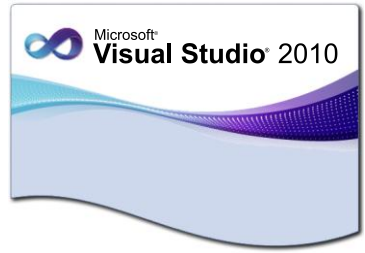


Рис. 18.22. Вращение куба с произвольным выбором осей

Резюме

В этой главе вы изучили основы Windows Presentation Foundation. Мы исследовали общую архитектуру этой инфраструктуры и ее модель программирования, в том числе и новую концепцию использования декларативной разметки для дизайна и компоновки пользовательского интерфейса клиентских приложений WPF.

Windows Presentation Foundation можно использовать для того, чтобы быстро начать создавать красивые пользовательские интерфейсы, применяя те же самые процессы разработки, которые мы используем при создании приложений Windows Forms или ASP.NET.



ГЛАВА 19

Silverlight

Технология Silverlight — это относительно новая технология, поддерживаемая множеством обозревателей, платформ и устройств для работы с мультимедиа и насыщенными интерактивными веб-приложениями нового поколения на основе технологии .NET.

Технология Silverlight является подмножеством Windows Presentation Foundation (WPF), которое значительно расширяет возможности элементов браузера для создания пользовательского интерфейса графики, анимации и мультимедиа.

Дополнительным преимуществом Silverlight является возможность использования разработанных приложений WPF. Также вы сможете переносить приложения Silverlight на WPF, повторно используя код XAML. Поскольку Silverlight содержит подмножество функциональных возможностей, предоставляемых WPF, существуют некоторые ограничения, налагаемые при переносе кода в Silverlight. Например, в Silverlight не поддерживаются документы нефиксированного формата. Silverlight не поддерживает динамические ресурсы. Также в WPF существуют некоторые возможности привязок данных, которые не поддерживаются в Silverlight.

Настройка среды разработки

Первый шаг в построении приложений Silverlight состоит в настройке вашей среды разработки. Последние версии инструментов для дизайна и разработки лучше всего искать по адресу: **http://www.silverlight.net**. На этом сайте вы найдете ссылки для скачивания различных версий исполняющей среды, инструментов конструирования, шаблонов разработки и SDK.

По умолчанию Visual Studio 2010 включает поддержку Silverlight 3. Последняя версия на момент написания этой книги — Silverlight 4, который надо будет установить для работы с примерами в этой главе. Silverlight 4 доступен для свободного скачивания по следующей ссылке:

http://www.microsoft.com/downloads/details.aspx?FamilyID=
40ef0f31-cb95-426d-9ce0-00dcfabf3df5&displaylang=en

Инсталляция пакета Silverlight 4 простая и не должна вызвать у вас затруднений.

Для запуска приложения Silverlight пользователям необходим небольшой подключаемый модуль для браузера. Подключаемый модуль предоставляется бесплатно. Если у пользователя еще не установлен подключаемый модуль, ему автоматически предлагается установить его. Загрузка и установка занимают всего несколько секунд и не требуют от пользователя никаких действий, кроме разрешения на установку.

Создание приложения Silverlight

Существует несколько способов использования Silverlight.

- Создание автономного приложения Silverlight. В этом случае приложение запускается внутри веб-браузера и выполняется независимо.
- Создание веб-страницы, в которую Silverlight встроен как часть страницы.

Все шаблоны проектов Silverlight находятся в отдельном узле в диалоговом окне **Add New Project** (рис. 19.1).

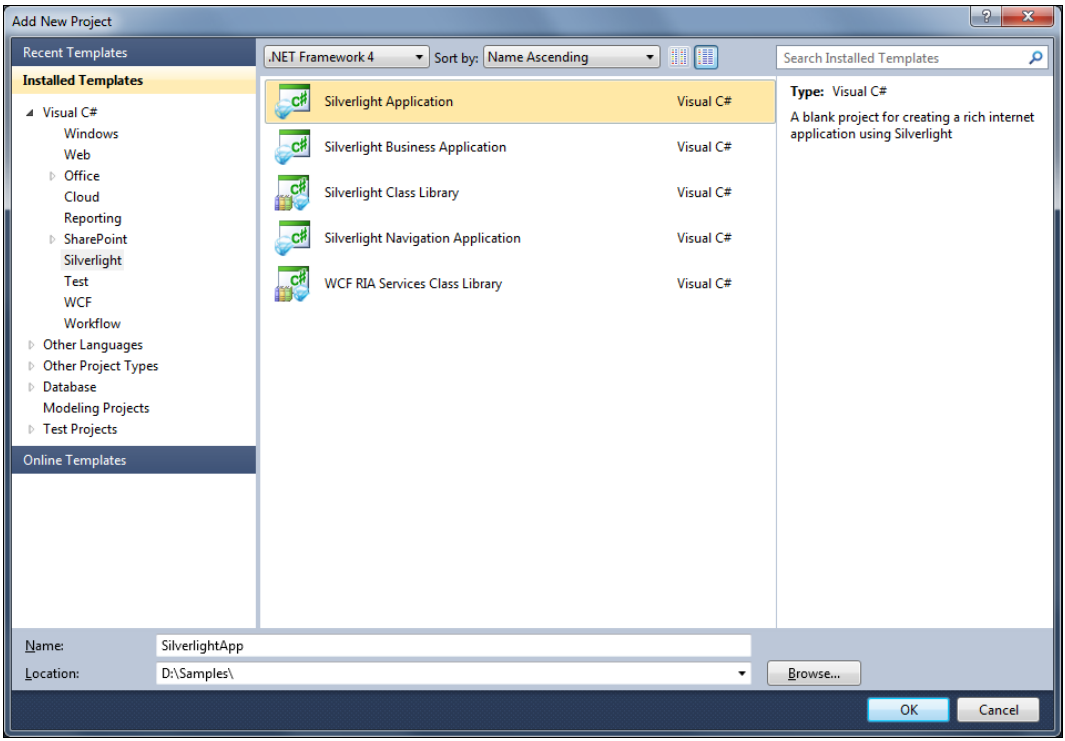


Рис. 19.1. Шаблоны проектов Silverlight в Visual Studio 2010

В этом диалоговом окне мы выберем тип проекта **Silverlight Application** для создания автономного приложения Silverlight.

Введите имя проекта — `SilverlightApp` и имя каталога, где будет располагаться приложение, и нажмите кнопку **OK**. При этом откроется диалоговое окно **New Silverlight Application**, как показано на рис. 19.2.

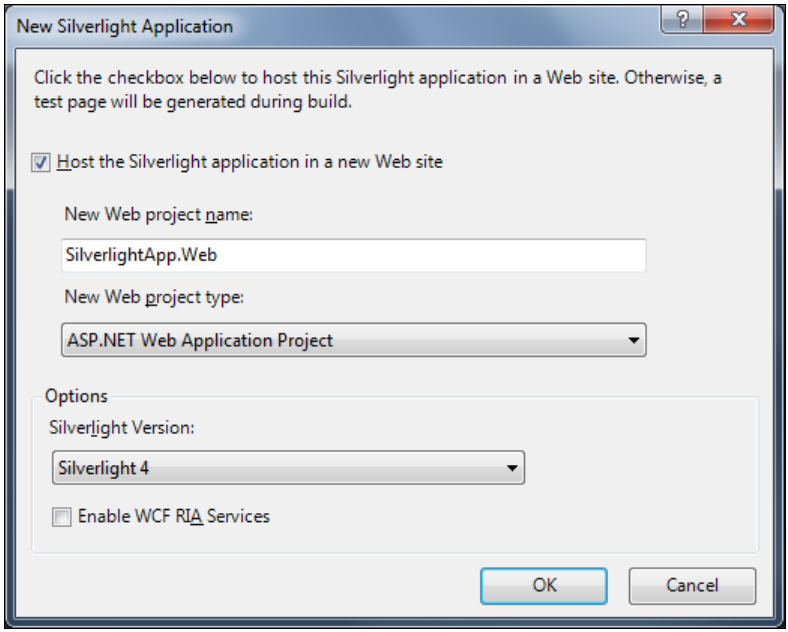


Рис. 19.2. Окно **New Silverlight Application**

Затем установите флажок **Host the Silverlight application in a new Web site**, если требуется добавить к решению для размещения приложения Silverlight отдельный веб-сайт ASP.NET или проект веб-приложения ASP.NET. При выборе этого параметра необходимо также указать имя веб-проекта в текстовом поле **New Web project name** и тип веб-проекта в выпадающем списке **New Web project type**. Кроме этого, в выпадающем списке **Silverlight Version** выберите нужную версию Silverlight и нажмите кнопку **OK**.

После закрытия диалогового окна **New Silverlight Application** в **Solution Explorer** появятся два новых узла: проект **SilverligthApp** и тестовый веб-сайт **SilverlightApp.Web** (рис. 19.3).

Файлы проекта `SilverligthApp` аналогичны проектам WPF, за исключением того, что вместо класса окна приложения MainWindow.xaml в проекте Silverlight среда разработки создает класс страницы MainPage.xaml.

Так же как в проектах WPF, класс `App` является обязательным для управления созданием приложения Silverlight. Класс `App` реализуется с помощью файла разметки App.xaml и файла кода страницы App.xaml.cs. Для класса `App` создается экземпляр с помощью подключаемого модуля Silverlight после создания пакета приложения (XAP-файл).

При создании приложения на основе Silverlight с помощью управляемого API процесс построения создает пакет приложения. Пакет приложения является ZIP-

файлом с расширением XAP-файла. Этот файл обычно содержит основные сборки и ресурсы приложения. Он также содержит создаваемый построением манифест, описывающий приложение и указывающий все сборки, требуемые при запуске. Эти сборки могут быть внутренними или внешними по отношению к пакету приложения.

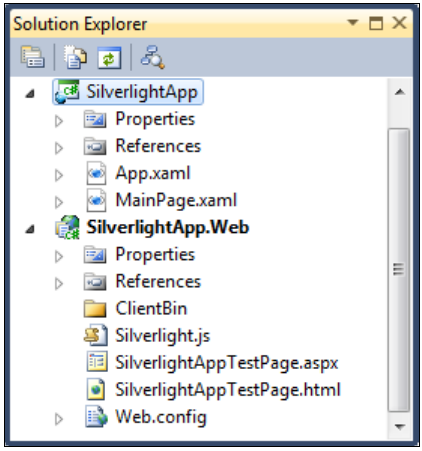


Рис. 19.3. Дерево файлов проектов Silverlight

При внедрении подключаемого модуля Silverlight в веб-страницу требуется указать пакет приложения, загружаемый подключаемым модулем. Подключаемый модуль использует файл манифеста в пакете приложения, чтобы определить класс приложения, для которого создается экземпляр. Этот класс известен как точка входа приложения, и он должен быть производным от класса `Application`.

При использовании кэширования библиотеки приложения манифест также указывает, какие требуемые сборки являются внешними по отношению к пакету приложения. Подключаемый модуль извлечет все такие файлы. Для локализованных приложений подключаемый модуль также извлечет вспомогательные сборки определенного языка и региональных параметров для всех необходимых внутренних и внешних сборок.

Класс `Application` предоставляет событие `Startup`, которое можно обработать для инициализации приложения и его пользовательского интерфейса. Класс `Application` также предоставляет другие службы приложений, которые часто бывают необходимы. Например, его можно использовать для извлечения файлов ресурсов из пакета приложения или из загруженных ZIP-файлов.

Процесс загрузки файлов запуска приложения и создания экземпляра класса приложения называется *системой активации Silverlight*. Система активации позволяет указать минимальную начальную загрузку одного или нескольких пакетов для оптимизации кэширования. После активации приложение может извлекать дополнительные сборки библиотеки и файлы ресурсов по требованию.

При установке флажка **Host the Silverlight application in a new Web site** в диалоговом окне **New Silverlight Application** создается тестовый веб-сайт ASP.NET и добавляется к решению Silverlight.

Веб-сайт содержит следующие файлы:

- Silverlight.js — вспомогательный файл JavaScript, содержащий функции для инициализации экземпляров подключаемого модуля Silverlight и функции для определения установленной версии клиента подключаемого модуля;
- SilverligthAppTestPage.html — HTML-файл, используемый для настройки и создания экземпляра подключаемого модуля Silverlight, который загружает и запускает приложение Silverlight. Имя этого файла — объединение имени проекта приложения Silverlight и текста TestPage.html;
- SilverligthAppTestPage.html — ASPX-файл, являющийся веб-страницей запуска по умолчанию. Имя этого файла — объединение имени проекта приложения Silverlight и текста TestPage.aspx;
- Web.config — файл конфигурации приложения.

При создании приложения используйте XAML-файл разметки для написания вашего интерфейса пользователя и вносите весь необходимый код в файл выделенного кода. Этот процесс должен быть вам хорошо знаком по прошлой главе. Шаблон приложения Silverlight (листинг 19.1) несколько отличается от шаблона WPF корневым тегом — вместо `<Window>` используется тег `<UserControl>`.

Листинг 19.1. Пустой шаблон класса модуля Silverlight

```
<UserControl x:Class="SilverlightApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White">

    </Grid>
</UserControl>
```

Использование кода приложений WPF в Silverlight

Как упоминалось в начале главы, в приложениях Silverlight возможно повторно использовать код XAML приложений WPF. В качестве примера возьмем приложения WPF из *разд. "Двумерные формы"* предыдущей главы (см. листинги 18.23 и 18.24) и внедрим его в приложение Silverlight.

Однако, на самом деле все не так просто. При компиляции приложения будут ошибки, и код из приложения WPF нуждается в небольшой доработке. Во-первых, у классов `Ellipse` и `Rectangle` не определены события `MouseUp`, поэтому их придется

заменить событием `MouseLeftButtonUp`. Во-вторых, теги элементов управления в Silverlight имеют префикс `sdk`, например:

```
<sdk:Label>
```

Поэтому в код все-таки необходимо будет внести некоторые изменения. Код нового приложения для Silvirlight показан в листингах 19.2 и 19.3.

Примечание

Полный код примера приложения находится в каталоге Chapter19\SilverlightApp на прилагаемом к книге диске.

Листинг 19.2. Файл SilverlightApp.xaml

```
<UserControl x:Class="SilverlightApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc=
      "http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400"
  xmlns:sdk=
    "http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk">

    <Grid x:Name="LayoutRoot" Background="White">
       <Ellipse Height="62" Margin="31,12,165,125" Name="ellipse1"
            Stroke="Black" Width="117" Fill="#FFFF0000"
            MouseLeftButtonUp="shape_MouseLeftButtonUp" />
       <Rectangle Height="69" Margin="31,95,165,35" Name="rectangle1"
            Stroke="#FF06FF00" Width="117" StrokeThickness="5"
            MouseLeftButtonUp="shape_MouseLeftButtonUp">
          <Rectangle.Fill>
             <ImageBrush ImageSource=
                 "/Shapes2D;component/Images/Hydrangeas.jpg"/>
          </Rectangle.Fill>
       </Rectangle>
       <Ellipse Height="61" Margin="205,12,46,126" Name="ellipse2"
            Stroke="Red" Width="61" StrokeThickness="3" Fill="Blue"
            MouseLeftButtonUp="shape_MouseLeftButtonUp" />
       <Rectangle Height="70" Margin="183,94,21,35" Name="rectangle2"
            Stroke="#FF000019" Width="109" RadiusX="20" RadiusY="20"
            Fill="#FF00FF00" StrokeThickness="10"
            MouseLeftButtonUp="shape_MouseLeftButtonUp" />
       <sdk:Label Height="28" Margin="0,170,0,0" Name="label1"
             HorizontalContentAlignment="Center" />
    </Grid>
</UserControl>
```

Листинг 19.3. Файл SilverlightApp.xaml.cs

```
namespace SilverlightApp
{
   public partial class MainPage : UserControl
   {
      public MainPage()
      {
         InitializeComponent();
      }

      private void shape_MouseLeftButtonUp(
             object sender, MouseButtonEventArgs e)
      {
         label1.Content = "Click on " + ((Shape)sender).Name;
      }
   }
}
```

Запустите тестовый веб-сайт `SilverlightApp.Web` в браузере. Браузер отобразит содержимое Silverlight-приложения (проекта `SilverlightApp`) — группу различных геометрических фигур, созданных нами в предыдущей главе (рис. 19.4).

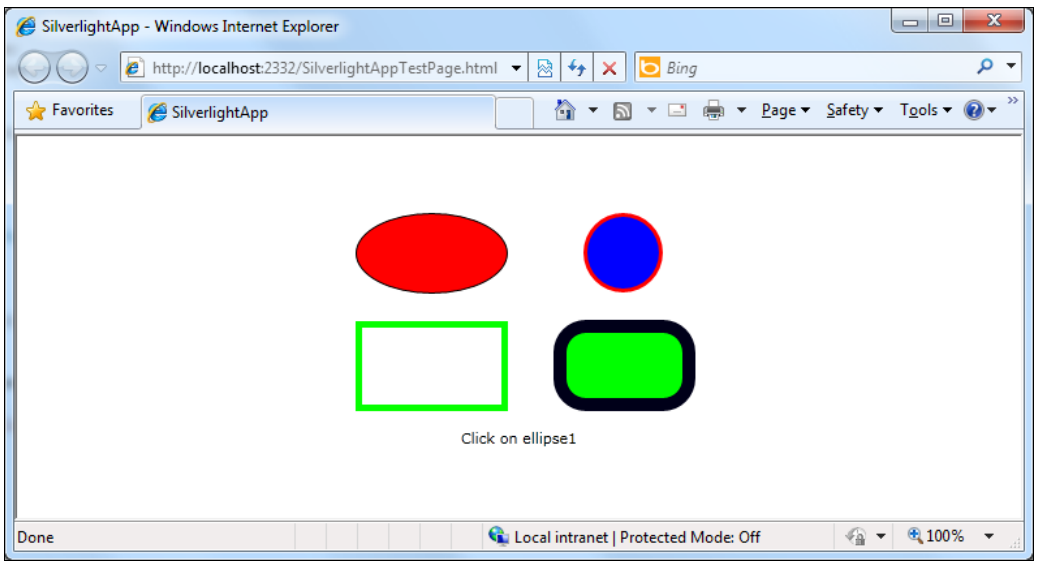


Рис. 19.4. Приложение Silverlight в браузере Internet Explorer

Анимация в Silverlight

Предыдущая глава, посвященная созданию приложений WPF, была насыщена большим количеством информации, поэтому вопросы создания анимации я решил

рассмотреть в этой главе. Кроме того, вопросы анимации все-таки более актуальны в веб-приложениях, чем в WPF. Однако общие принципы работы с анимацией в WPF такие же, как и в Silverlight.

В XAML анимация элемента осуществляется за счет изменения одного или более его свойств во времени. Это время определяется с помощью временной шкалы.

Анимации в Silverlight выполняются в ответ на некоторое событие, которое определяется с помощью триггера в XAML или в программном коде класса, в обработчике какого-либо события. С обработчиками событий мы уже хорошо знакомы, однако с триггерами еще не работали.

В настоящее время Silverlight поддерживает только один тип триггеров — `EventTrigger`. Элементы пользовательского интерфейса имеют коллекцию `Triggers`, которая используется для определения одного или нескольких объектов `EventTrigger`.

Например, для анимации прямоугольника определение коллекции триггеров выглядит следующим образом:

```
<Rectangle x:Name="Rect" Width="200" Height="100" Fill="Magenta">
   <Rectangle.Triggers>
      ...
   </Rectangle.Triggers>
</Rectangle>
```

Далее необходимо определить `EventTrigger`, который будет добавлен в эту коллекцию. В данном `EventTrigger` с помощью свойства `RoutedEvent` задается событие, в ответ на возникновение которого будет запускаться анимация.

```
<Rectangle.Triggers>
   <EventTrigger RoutedEvent="Rectangle.Loaded">
      ...
   </EventTrigger>
</Rectangle.Triggers>
```

Класс `Storyboard` управляет анимацией с помощью временной шкалы и предоставляет сведения о целевых объектах и свойствах своим дочерним анимациям. С помощью класса `Storyboard` можно выполнить анимацию разворота картинки. В результате создается эффект постоянного вращения изображения вокруг оси X в трехмерном пространстве.

Для запуска экземпляра `Storyboard` используется класс `BeginStoryboard`. `BeginStoryboard` — это триггерное действие, содержащее объект `Storyboard`. В объектах `Storyboard` находятся описания анимации. При описании анимации эти объекты просто встраиваются в определение `EventTrigger`, например, таким образом:

```
<EventTrigger RoutedEvent="Rectangle.Loaded">
   <BeginStoryboard>
      <Storyboard>
         ...
      </Storyboard>
   </BeginStoryboard>
 </EventTrigger>
```

В Silverlight возможна анимация трех разных типов свойств:

- *анимация точки* — производится с помощью класса `PointAnimiation` или `PointAnimationUsingKeyFrames`;
- *анимация типа* `Double` — производится с помощью класса `DoubleAnimation` или `DoubleAnimationUsingKeyFrames`;
- *анимация цвета* — производится с помощью класса `ColorAnimation` или `ColorAnimationUsingKeyFrames`.

Давайте для примера рассмотрим `DoubleAnimation`. Объект `DoubleAnimation` позволяет задавать изменение значения типа `Double` по времени. Для этого надо задать значение в начале анимации с помощью свойства `From` и его конечное значение с помощью свойства `To`. Также задается свойство `Duration` — время продолжительности анимации в формате HH:MM:SS.

Например, чтобы изменить свойство `Opacity` (прозрачность) прямоугольника от 1.0 (непрозрачный) до 0.0 (полностью прозрачный) в течение 5 секунд, можно задать свойству `From` значение `1.0`, а свойству `To` значение `0.0`:

```
<Storyboard>
   <DoubleAnimation
      Storyboard.TargetName="Rect"
      Storyboard.TargetProperty="Opacity"
      From="1.0"
      To="0.0"
      Duration="00:00:05"
      AutoReverse="True"
      RepeatBehavior="Forever"/>
</Storyboard>
```

В листинге 19.4 приведена полная разметка XAML для создания прямоугольника, который исчезает и появляется при загрузке веб-страницы.

> ***Примечание***
>
> Полный код примера веб-страницы находится на прилагаемом к книге диске в каталоге Chapter19\AnimationRect.

Листинг 19.4. Код XAML приложения с анимацией прямоугольника

```
<UserControl x:Class="Animation.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc=
      "http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Width="400" Height="300">

   <Rectangle
       x:Name="Rect"
```

```
        Width="200"
        Height="100"
    Fill="Magenta">
      <Rectangle.Triggers>
        <EventTrigger RoutedEvent="Rectangle.Loaded">
          <BeginStoryboard>
            <Storyboard>
              <DoubleAnimation
                Storyboard.TargetName="Rect"
                Storyboard.TargetProperty="Opacity"
                From="1.0"
                To="0.0"
                Duration="00:00:05"
                AutoReverse="True"
                RepeatBehavior="Forever" />
            </Storyboard>
          </BeginStoryboard>
        </EventTrigger>
      </Rectangle.Triggers>
  </Rectangle>

</UserControl>
```

После запуска приложения в браузере мы получим постепенно, в течение 5 секунд, исчезающий и вновь появляющийся прямоугольник (рис. 19.5).

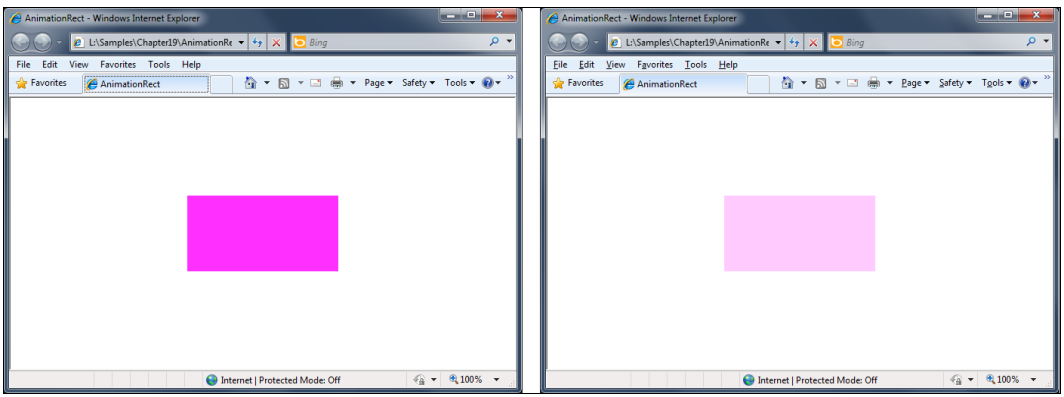


Рис. 19.5. Анимация прямоугольника

Трехмерные эффекты с трансформацией перспективы

Трехмерные эффекты с трансформацией перспективы могут применяться к элементам XAML для имитации их вращения в трехмерном пространстве. Следует

отметить, что это на самом деле имитация 3D-пространства, в отличие от WPF, где обеспечивается полная поддержка 3D-визуализации.

Трехмерные эффекты с трансформацией перспективы предоставляет класс `PlaneProjection`. Он позволяет вращать двумерные изображения в трехмерном пространстве. В объекте `PlaneProjection` определены три свойства для вращения по соответствующим осям:

- `RotationX`
- `RotationY`
- `RotationZ`

Кроме перечисленных, в классе `PlaneProjection` существуют и другие свойства, которые можно использовать для управления вращением и перемещением объекта, например, `CenterOfRotationX`, `CenterOfRotationY`, `CenterOfRotationZ`, `GlobalOffsetX`, `GlobalOffsetY`, `GlobalOffsetZ`, `LocalOffsetX`, `LocalOffsetY` и `LocalOffsetZ`.

В листинге 19.5 показан пример кода приложения, обеспечивающего перспективное представление путем вращения изображения относительно оси X.

> ***Примечание***
>
> Полный код примера приложения находится на прилагаемом к книге диске в каталоге Chapter19\PlaneProjection.

Листинг 19.5. Файл MainPage.XAML

```
<UserControl x:Class="PlaneProjection.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Width="800" Height="600">

   <UserControl.Resources>
      <Storyboard x:Name="StoryboardImage">
         <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
            Storyboard.TargetName="image"
            Storyboard.TargetProperty=
               "(UIElement.Projection).(RotationX)"
            RepeatBehavior="Forever">

            <SplineDoubleKeyFrame KeyTime="00:00:00" Value="0"/>
            <SplineDoubleKeyFrame KeyTime="00:00:03" Value="360"/>
         </DoubleAnimationUsingKeyFrames>
      </Storyboard>
   </UserControl.Resources>
   <Grid x:Name="LayoutRoot" Background="White">
      <Image x:Name="image" Source="Tulips.jpg" Width="640" Height="480"
```

```
            Canvas.Top="150" Canvas.Left="0"
            MouseLeftButtonDown="image_MouseLeftButtonDown">

         <Image.Projection>
            <PlaneProjection RotationX="0"></PlaneProjection>
         </Image.Projection>
      </Image>
   </Grid>
</UserControl>
```

Для запуска и остановки анимации можно использовать событие клика левой кнопки мыши `MouseLeftButtonDown()` на картинке. Для его реализации в классе `MainPage` создайте обработчик события `MouseLeftButtonDown()` и напишите следующий код, представленный в листинге 19.6.

Листинг 19.6. Файл MainPage.XAML.cs

```
namespace PlaneProjection
{
   public partial class MainPage : UserControl
   {
      bool isStart = false;

      public MainPage()
      {
         InitializeComponent();
      }

      private void image_MouseLeftButtonDown(
            object sender, MouseButtonEventArgs e)
      {
         isStart = !isStart;

         if (isStart)
            StoryboardImage.Begin();
         else
            StoryboardImage.Stop();
      }
   }
}
```

Этот класс используется для создания преобразования перспективы (эффекта объемного изображения). Например, можно создать иллюзию поворота объекта по направлению к пользователю или от него (рис. 19.6).

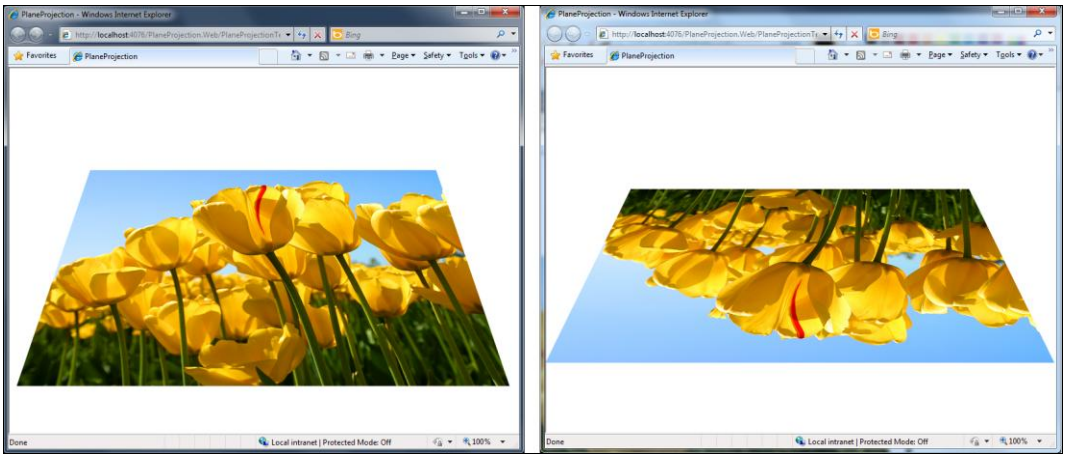


Рис. 19.6. Вращение изображения относительно оси X

Интеграция Silverlight с веб-страницей

До этого момента мы запускали приложения Silverlight в тестовой среде. Для практического использования на веб-страницах можно внедрить подключаемый модуль Silverlight одним из двух способов:

- использовать HTML-элемент `object`;
- использовать вспомогательный файл Silverlight.js.

Элемент HTML `object` совместим со всеми поддерживаемыми веб-браузерами и служит основой внедрения методики Silverlight.js. Функции внедрения Silverlight.js в конечном итоге формируют HTML-элементы `object` и предоставляют все возможности, которые предоставляет элемент `object`. Эта общая основа позволяет сочетать обе методики внедрения.

Для поддержки приложений на основе Silverlight веб-сервер должен быть настроен для связи расширения имени XAP-файла с MIME-типом:

```
application/x-silverlight-app
```

Для IIS 7.0 и более новых версий это значение уже должно быть настроено по умолчанию.

Использование HTML

HTML-элемент `object` является наиболее простым способом внедрения подключаемого модуля Silverlight, и рекомендуется его использовать. Это подход по умолчанию, который используется средой Visual Studio при создании нового приложения на основе Silverlight и его размещении на динамически создаваемой странице HTML.

Элемент `object` можно использовать для интеграции Silverlight в код JavaScript на веб-странице. Тем не менее, чтобы внедрить элемент управления, JavaScript

не требуется. Это полезно, если JavaScript отключен в клиенте или запрещен на сервере.

В листинге 19.7 показано использование элемента `object` для внедрения модуля Silverlight на веб-страницу.

Примечание

Полный код примера веб-страницы находится на прилагаемом к книге диске в каталоге Chapter19\IntegrationSilverlight\EmbedObject.aspx.

Листинг 19.7. Внедрение модуля Silverlight в элемент `object`

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="EmbedObject.aspx.cs"
Inherits="_Default" %>

<!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></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
       <object width="800" height="600"
             data="data:application/x-silverlight-2,"
             type="application/x-silverlight-2" >
          <param name="source" value="ClientBin/PlaneProjection.xap"/>
       </object>
    </div>
    </form>
</body>
</html>
```

Использование Silverlight.js

Файл Silverlight.js необязателен для интеграции Silverlight в JavaScript. Тем не менее при использовании JavaScript он предоставляет функции внедрения для удобства. В основном Silverlight.js полезен для включения установки Silverlight с усовершенствованным JavaScript. Даже при использовании элемента HTML `object` для внедрения подключаемого модуля можно воспользоваться преимуществами этих возможностей.

Файл Silverlight.js является вспомогательным файлом JavaScript, на который можно ссылаться на веб-странице. Файл Silverlight.js содержит определенные в нем функции `createObject()` и `createObjectEx()`, которые можно вызывать для внедрения подключаемого модуля Silverlight на веб-странице.

Вид Silverlight.js зависит от сведений о реализации браузера, которые могут отличаться в различных выпусках Silverlight. По этой причине не рекомендуется использовать Silverlight.js. Однако, если все-таки используется Silverlight.js, следует периодически проверять наличие обновленной версии на странице Silverlight.js коллекции кода MSDN.

Встраиваемые функции Silverlight.js принимают сведения конфигурации как входные параметры и создают элементы `object`.

Несмотря на то, что использование Silverlight.js создает проблемы совместимости с браузером, встраиваемые функции Silverlight.js позволяют программно настроить подключаемый модуль Silverlight при его внедрении на страницу и указать уникальный идентификатор для каждого внедряемого подключаемого модуля. Это полезно, если внедрить несколько экземпляров подключаемого модуля в одной веб-странице.

В листинге 19.8 показано использование функции `createObject()` для внедрения модуля Silverlight на веб-страницу. Сам файл Silverlight.js можно скопировать в каталог веб-сайта из тестовых веб-сайтов, созданных в этой главе.

Примечание

Полный код примера веб-страницы находится на прилагаемом к книге диске в каталоге Chapter19\IntegrationSilverlight\EmbedJS.aspx.

Листинг 19.8. Использование вспомогательного файла Silverlight.js

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="EmbedJS.aspx.cs"
Inherits="EmbedJS" %>

<!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></title>
        <script type="text/javascript" src="Silverlight.js"></script>
</head>
  <body>
    <form id="form1" runat="server">
      <div>
        <div id="silverlightControlHost">
          <script type="text/javascript">
            Silverlight.createObject(
               "ClientBin/PlaneProjection.xap",
               silverlightControlHost,
               "slPlugin",
               {
                 width: "800", height: "600", background: "white",
                 version: "4.0.50303.0"
               },
```

```
            { onError: onSLError, onLoad: onSLLoad },
            "param1=value1,param2=value2",
            "context"
          );

          function onSLLoad(plugIn, userContext, sender) {
            window.status +=
                plugIn.id + " loaded into " + userContext + ". ";
          }

          function onSLError(sender, args) {
            // Отобразить сообщение об ошибке при загрузке
          }

        </script>
      </div>
    </div>
  </form>
 </body>
</html>
```

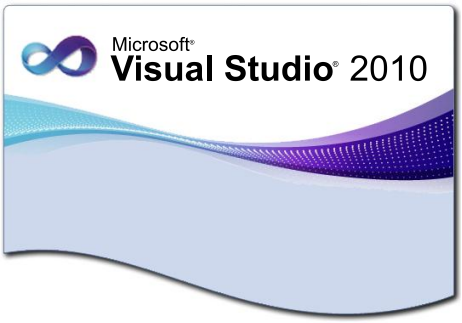
Примечание

Пример реализации обработчика ошибок можно посмотреть в коде веб-страницы тестового веб-сайта, создаваемого в паре с приложением Silverlight. Загрузка модуля Silverlight в тестовом веб-сайте осуществляется по такому же принципу, с использованием файла Silverlight.js.

Резюме

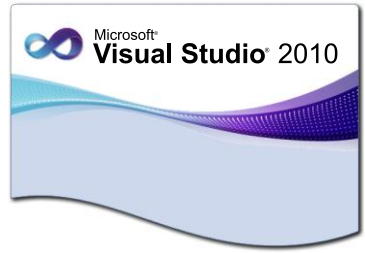
Технология Silverlight значительно расширяет возможности браузера для создания пользовательского интерфейса. Silverlight позволяет создавать впечатляющие графику, анимации, мультимедиа и другие многофункциональные клиентские функции. Кроме того, технология Silverlight обеспечивает возможность совместной работы дизайнеров и разработчиков при создании интернет-приложений.

ЧАСТЬ IV



Сервисы и коммуникации

ГЛАВА 20



Windows Communication Foundation

Windows Communication Foundation (WCF) является единой моделью программирования Microsoft для построения служб. Она позволяет разработчикам построить безопасные надежные решения с поддержкой транзакций и возможностью межплатформенной интеграции.

Microsoft собрала вместе веб-службы ASP.NET, технологию .NET Remoting, Enterprise Services, и начиная с версии .NET Framework 3.0 появился единый способ коммуникации — Windows Communication Foundation.

Создание проекта сервиса WCF

Visual Studio предоставляет инструменты, которые облегчают создание сервисов WCF. Для создания службы надо сделать следующее:

1. Создать проект WCF.
2. Создать сервисный контракт.
3. Реализовать сервисный контракт.
4. Настроить конечные точки связи для сервиса.

После этого создаете клиент, который будет пользоваться сервисом, и выбираете модель хостинга для вашего сервиса, далее развертываете его соответствующим образом.

Вы можете использовать Visual Studio для создания проекта сервиса WCF аналогично тому, как вы описываете другие проекты. Выбор **File | New | Project** приведет к открытию диалогового окна **Add New Project**. Здесь вы можете выбрать узел WCF. Это позволит вам выбрать шаблон проекта сервиса WCF (рис. 20.1). В этом диалоге вы можете задать имя сервиса и местонахождение его кода.

Обратите внимание, что здесь имеется несколько шаблонов сервисов WCF. Эти шаблоны позволяют вам создавать сервисы WCF под ваши конкретные запросы.

Сборка службы может представлять собой сборку библиотеки классов (DLL) или сборку приложения (EXE). Узел `WcfSvcHost` запустит новый процесс, в котором

будут автоматически размещаться все классы, перечисленные в файле конфигурации в разделе служб. Стоит отметить, что классы служб, равно как и контракты служб и данных, не обязательно должны быть открытыми — это могут быть внутренние классы. Кроме того, при саморазмещении службам не обязательно предоставлять метаданные, но в случае необходимости они могут это делать.

Шаблон **WCF Service Library** представляет собой всего-навсего готовый вариант использования узла `WcfSvcHost` и клиента `WcfTestClient`. По сути своей он весьма близок к тем методам, которые были описаны ранее (при сочетании обоих компонентов). Если вы используете проект **WCF Service Library**, то ни указывать WcfSvcHost.exe в качестве выполняемого приложения, ни задавать файл CONFIG не нужно, поскольку в файле проекта есть элемент `ProjectTypeGuids`, предназначенный для библиотеки служб WCF.

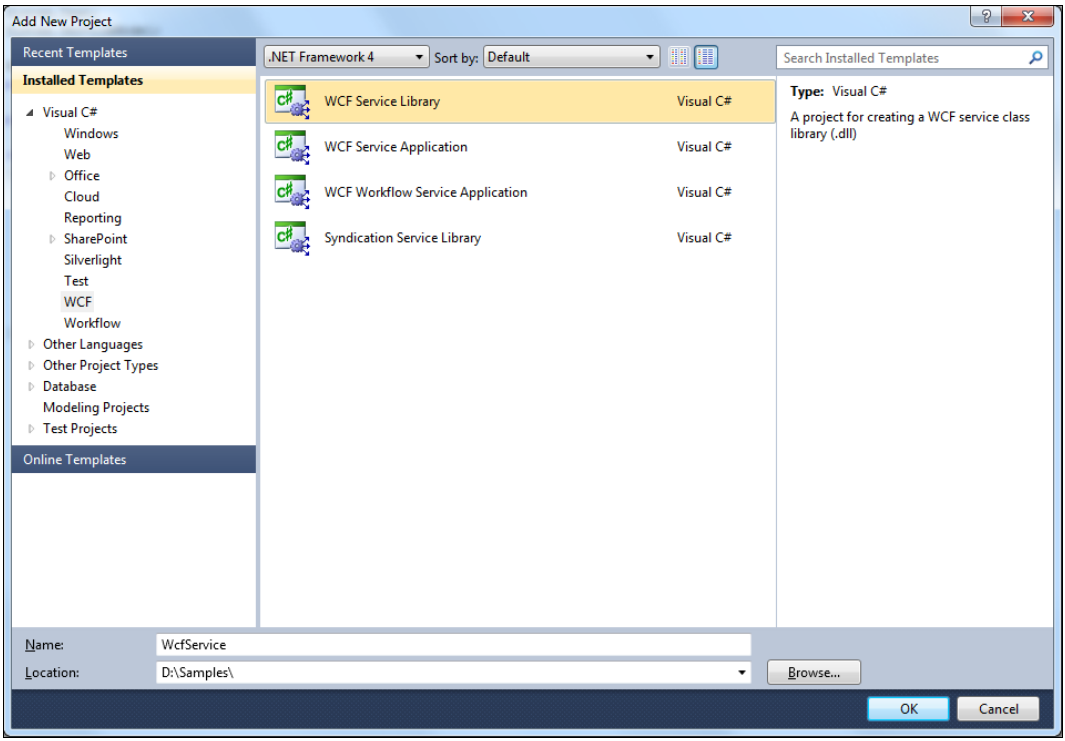


Рис. 20.1. Шаблоны проектов WCF

Созданный проект **WCF Service Library**, содержит интерфейс, наследуемый от `IService`, для описания сервисного контракта, класс сервиса для реализации интерфейса, файл Web.config для настройки сервиса, а также ссылки на другие библиотеки .NET. На рис. 20.2 показана структура проекта в **Solution Explorer** на основе этого шаблона.

Класс сервиса IService1.cs — это тот интерфейс, который используется для описания контракта вашего сервиса. Контракт включает в себя сервисные операции и

контракт данных. Выделение интерфейса в отдельный файл помогает абстрагировать все атрибуты WCF и элементы контракта от логики вашего сервиса.

Шаблон класса `Service1`, сгенерированный средой Visual Studio при создании проекта WCF, представлен в листинге 20.1.

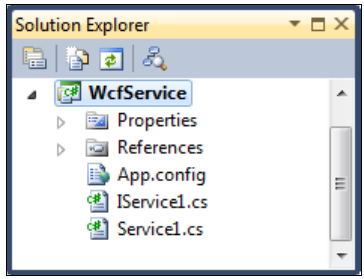


Рис. 20.2. Проект **WCF Service Library**

Примечание

Код приложения находится в каталоге Chapter20\WcfService на прилагаемом к книге диске.

Листинг 20.1. Класс `Service1`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfService
{
   // NOTE: You can use the "Rename" command on the "Refactor" menu
   // to change the interface name "IService1" in both code and
   // config file together.
   // Примечание: Вы можете использовать команду "Rename" в меню
   // "Refactor", чтобы изменить название интерфейса "IService1"
   // одновременно в коде и в файле конфигурации.
   [ServiceContract]
   public interface IService1
   {
      [OperationContract]
      string GetData(int value);

      [OperationContract]
      CompositeType GetDataUsingDataContract(CompositeType composite);

      // TODO: Add your service operations here
      // TODO: Добавьте ваши сервисные операции здесь
   }
```

```
    // Use a data contract as illustrated in the sample below to
    // add composite types to service operations
    // Используйте контракт данных, как показано в образце ниже,
    // чтобы добавить сервисные операции с композитными типами
    [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; }
        }
    }
}
```

Класс определяется как сервис WCF при помощи атрибута `[ServiceContract]` в верхней части класса. Класс сервиса `Service1` реализует интерфейс сервиса. Именно здесь располагается логика сервиса независимо от того, содержится ли в ней реальная бизнес-функциональность или вызовы другой библиотеки, в которой находится фактический код реализации.

Листинг 20.2. Код реализации сервиса WCF

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfService
{
    // NOTE: You can use the "Rename" command on the "Refactor"
    // menu to  change the class name "Service1" in both code
    // and config file together.
```

```
    // Примечание: Вы можете использовать команду "Rename" в меню
    // "Refactor", чтобы изменить название класса "Service1"
    // одновременно в коде и в файле конфигурации.
    public class Service1 : IService1
    {
        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;
        }
    }
}
```

Для тестирования сервиса можно создать консольное приложение-клиент. Назовите его `WcfClient`. Для работы с сервисом нам потребуется добавить в него ссылку на сервис. Для этого выделите файл проекта, щелкните на нем правой кнопкой мыши и в контекстном меню выберите пункт **Add Service Reference**. При этом откроется диалоговое окно **Add Service Reference**, представленное на рис. 20.3.

Вы можете использовать кнопку **Discover** для поиска сервиса WCF в данном решении. Настройте пространство имен для ссылки в значение `IService1` и нажмите кнопку **OK**. Теперь у вас имеется новая ссылка на сервис и новый файл App.config.

После добавления ссылки на сервис в клиентском приложении средой Visual Studio будет сгенерирован прокси-класс, являющийся оболочкой для нашего созданного ранее сервиса (рис. 20.4).

Теперь через созданный прокси-класс мы можем программно обращаться к нашему сервису из клиентского приложения. Код клиентского приложения представлен в листинге 20.3.

Примечание

Код приложения находится в каталоге Chapter20\WcfClient на прилагаемом к книге диске.

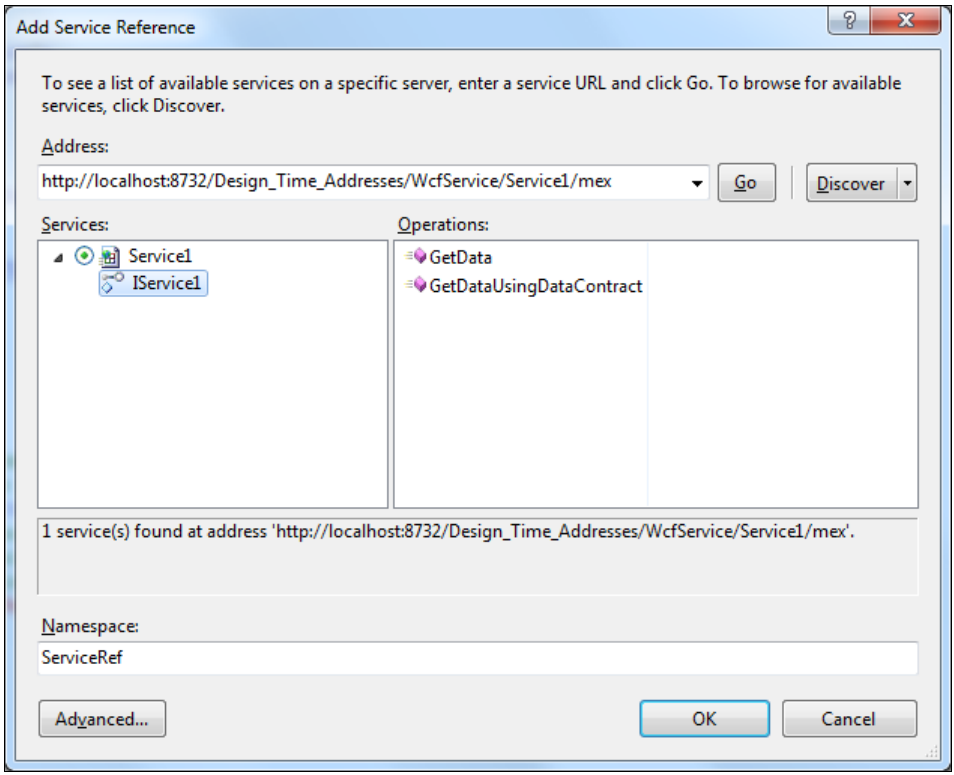


Рис. 20.3. Диалоговое окно **Add Service Reference**

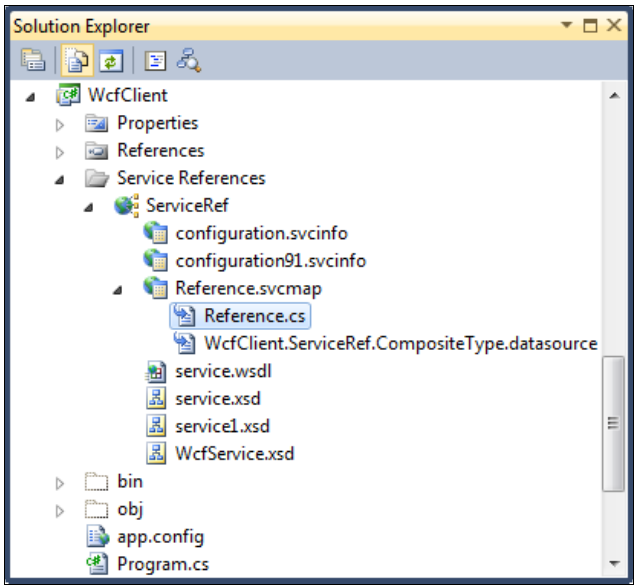


Рис. 20.4. Сгенерированный прокси-класс

Листинг 20.3. Код клиентского приложения

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

using WcfClient.ServiceRef;
namespace WcfClient
{
   class Program
   {
      static void Main(string[] args)
      {
         Service1Client proxy = new Service1Client();
         Console.WriteLine(proxy.GetData(99));

         CompositeType comp = new CompositeType();
         comp.BoolValue = true;
         Console.WriteLine("CompositeType string: {0}",
            proxy.GetDataUsingDataContract(comp).StringValue);
         proxy.Close();

         Console.Read();
      }
   }
}
```

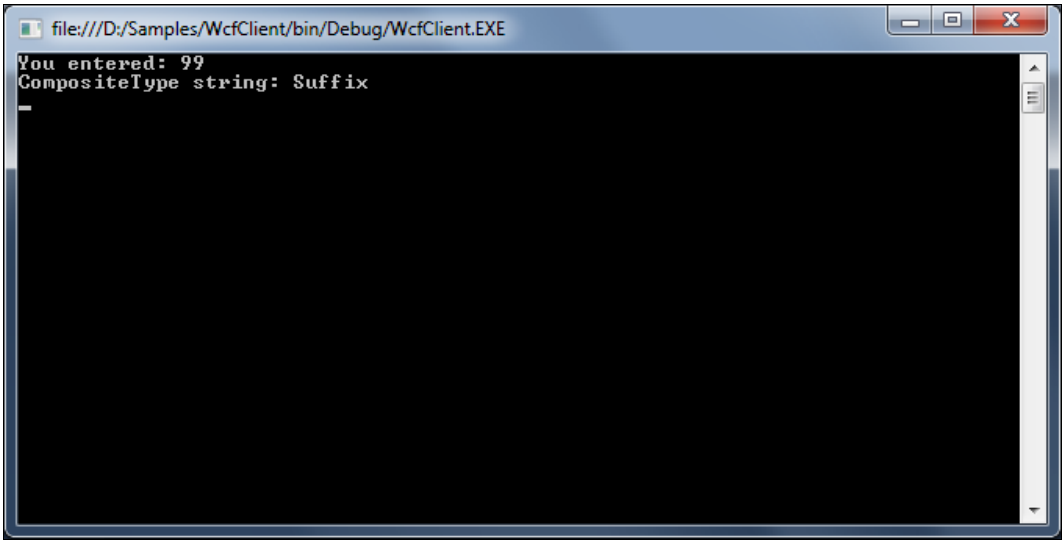


Рис. 20.5. Приложение-клиент

Запустите это приложение и протестируйте. Внешний вид консольного приложения показан на рис. 20.5.

Создание сервиса для работы с данными

Теперь создадим более реальное приложение WCF. Ранее в книге мы создавали веб-сайты, на которых была страница для регистрации абитуриентов. Создадим для нее небольшую базу данных для хранения данных абитуриентов.

SQL-скрипт для создания базы данных представлен в листинге 20.4.

Листинг 20.4. SQL-скрипт для создания таблиц в базе данных `CollegeApplicants`

```
CREATE TABLE [dbo].[Applicant](
       [ApplicantId] [int] IDENTITY(1,1) NOT NULL,
       [FirstName] [nvarchar](50) NOT NULL,
       [LastName] [nvarchar](50) NOT NULL,
       [YearOfBirth] [int] NOT NULL,
       [EMail] [nvarchar](50) NULL,
       [FacultyId] [int] NOT NULL,
       [EducationFormId] [int] NOT NULL,
       [AdditionalInfo] [nvarchar](255) NULL,
 CONSTRAINT [PK_Applicant] PRIMARY KEY CLUSTERED([ApplicantId] ASC))
GO

CREATE TABLE [dbo].[Faculty](
       [FacultyId] [int] IDENTITY(1,1) NOT NULL,
       [FacultyName] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Faculty] PRIMARY KEY CLUSTERED([FacultyId] ASC)
 )
GO

CREATE TABLE [dbo].[EducationForm](
       [EducationFormId] [int] IDENTITY(1,1) NOT NULL,
       [EducationFormName] [nvarchar](50) NULL,
 CONSTRAINT [PK_EducationForm] PRIMARY KEY CLUSTERED([EducationFormId] ASC))
GO

ALTER TABLE [dbo].[Applicant]  WITH CHECK ADD  CONSTRAINT
[FK_Applicant_EducationForm] FOREIGN KEY([EducationFormId])
REFERENCES [dbo].[EducationForm] ([EducationFormId])
GO

ALTER TABLE [dbo].[Applicant] CHECK CONSTRAINT [FK_Applicant_EducationForm]
GO

ALTER TABLE [dbo].[Applicant]  WITH CHECK ADD  CONSTRAINT
[FK_Applicant_Faculty] FOREIGN KEY([FacultyId])
```

```
REFERENCES [dbo].[Faculty] ([FacultyId])
GO

ALTER TABLE [dbo].[Applicant] CHECK CONSTRAINT [FK_Applicant_Faculty]
GO
```

Эта база содержит всего три таблицы и предназначена для хранения персональных данных абитуриентов и выбранной ими форме обучения, факультете (рис. 20.6).

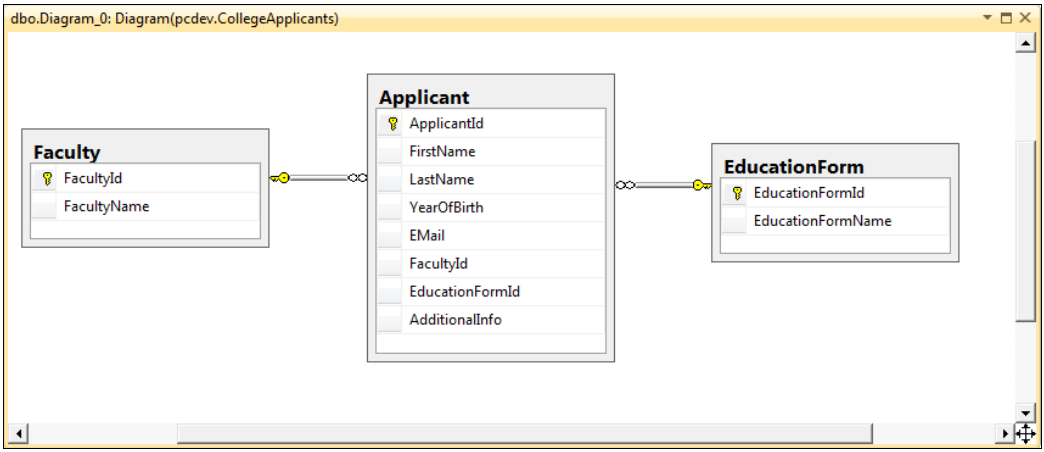


Рис. 20.6. Схема базы данных `CollegeApplicants`

Для работы с базой данных создадим хранимые процедуры, скрипт для создания которых представлен в листингах 20.5 и 20.6.

Листинг 20.5. Хранимая процедура `Applicant_Select`

```
CREATE PROCEDURE Applicant_Select (
  @applicantId int = NULL,
  @facultyId int = NULL,
  @educationFormId int = NULL)
AS
BEGIN
  SELECT
    A.[ApplicantId],
    A.[FirstName],
    A.[LastName],
    A.[YearOfBirth],
    A.[EMail],
    A.[FacultyId],
    F.[FacultyName],
    A.[EducationFormId],
    E.[EducationFormName],
    A.[AdditionalInfo]
```

```
  FROM [Applicant] A
  INNER JOIN [Faculty] F ON A.[FacultyId]=F.[FacultyId]
  INNER JOIN [EducationForm] E ON A.[EducationFormId]=E.[EducationFormId]
  WHERE
    (@applicantId IS NULL OR A.[ApplicantId] = @applicantId) AND
    (@facultyId IS NULL OR A.[FacultyId] = @facultyId) AND
    (@educationFormId IS NULL OR A.[EducationFormId] = @educationFormId)
END
```

Листинг 20.6. Хранимая процедура `Applicant_Insert`

```
CREATE PROCEDURE Applicant_Insert(
   @firstName nvarchar(50),
   @lastName nvarchar(50),
   @yearOfBirth int,
   @eMail nvarchar(50),
   @facultyId int,
   @educationFormId int,
   @additionalInfo nvarchar(255) = NULL)
AS
BEGIN
  INSERT [Applicant]
  VALUES(
    @firstName,
    @lastName,
    @yearOfBirth,
    @eMail,
    @facultyId,
    @educationFormId,
    @additionalInfo)
END
```

Теперь модифицируем интерфейс `ICollegeService`. Удалим из файла сгенерированный код и добавим вместо него объявления методов для работы с нашей базой данных. Каждое объявление сервисного метода должно быть помечено атрибутом `[OperationContract]`, как показано в листинге 20.7.

Примечание

Полный код сервиса находится в каталоге Chapter20\College.Service на прилагаемом к книге диске.

Листинг 20.7. Интерфейс `ICollegeService`

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

```
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using System.Data;

namespace College.Service
{
   [ServiceContract]
   public interface ICollegeService
   {
      [OperationContract]
      List<Applicant> GetApplicants(
         int? applicantId, int? facultyId, int? educationFormId);

      [OperationContract]
      bool NewApplicant(Applicant applicant);

      [OperationContract]
      bool EditApplicant(Applicant applicant);

      [OperationContract]
      bool DeleteApplicant(int applicantId);

      [OperationContract]
      List<Faculty> GetFaculties();

      [OperationContract]
      List<EducationForm> GetEducationForms();
   }
}
```

Добавьте в проект новый класс с именем `Applicant`. Этот класс будет реализовывать контракт данных. Он содержит набор свойств, описывающих сущность `Applicant`. Код класса представлен в листинге 20.8.

Листинг 20.8. Код класса `Applicant`

```
[DataContract]
public class Applicant
{
   [DataMember]
   public int ApplicantId { get; set; }

   [DataMember]
   public string FirstName { get; set; }

   [DataMember]
   public string LastName { get; set; }
```

```
    [DataMember]
    public int YearOfBirth { get; set; }

    [DataMember]
    public string EMail  { get; set; }

    [DataMember]
    public int FacultyId  { get; set; }

    [DataMember]
    public string FacultyName { get; set; }

    [DataMember]
    public int EducationFormId { get; set; }

    [DataMember]
    public string EducationFormName { get; set; }

    [DataMember]
    public string AdditionalInfo { get; set; }
}
```

Аналогичную функцию будут выполнять классы `EducationForm` и `Faculty`, код которых представлен в листингах 20.9 и 20.10.

Листинг 20.9. Код класса `EducationForm`

```
[DataContract]
public class EducationForm
{
    public EducationForm()
    {
        this.EducationFormId = -1;
    }

    [DataMember]
    public int EducationFormId { get; set; }

    [DataMember]
    public string EducationFormName { get; set; }
}
```

Листинг 20.10. Код класса `Faculty`

```
[DataContract]
public class Faculty
```

```
{
   public Faculty()
   {
      this.FacultyId = -1;
   }

   [DataMember]
   public int FacultyId { get; set; }

   [DataMember]
   public string FacultyName { get; set; }
}
```

В классе `CollegeApplicantAction` будет реализован доступ к базе данных `CollegeApplicants`. Код этого класса представлен в листингах 20.11—20.15.

Листинг 20.11. Закрытые переменные и конструктор класса `CollegeApplicantAction`

```
public class CollegeApplicantAction
{
   private SqlConnection connection;

   public CollegeApplicantAction()
   {
      connection = new SqlConnection(
         Properties.Settings.Default.CollegeApplicationConString);
   }
}
```

Листинг 20.12. Метод `GetApplicants()` класса `CollegeApplicantAction`

```
public List<Applicant> GetApplicants(int? applicantId, int? facultyId, int?
educationFormId)
{
   List<Applicant> data = new List<Applicant>();
   SqlCommand command = connection.CreateCommand();
   command.CommandType = CommandType.StoredProcedure;
   command.CommandText = "Applicant_Select";

   command.Parameters.Add("@applicantId", SqlDbType.Int);
   command.Parameters.Add("@facultyId", SqlDbType.Int);
   command.Parameters.Add("@educationFormId", SqlDbType.Int);

   if (applicantId < 1) applicantId = null;
   if (facultyId < 1) facultyId = null;
```

```
   if (educationFormId < 1) educationFormId = null;
   command.Parameters["@applicantId"].Value = applicantId;
   command.Parameters["@facultyId"].Value = facultyId;
   command.Parameters["@educationFormId"].Value = educationFormId;

   connection.Open();
   try
   {
      SqlDataReader rdr = command.ExecuteReader();
      while (rdr.Read())
      {
         Applicant app = new Applicant();
         app.ApplicantId = rdr.GetInt32(0);
         app.FirstName = rdr.GetString(1);
         app.LastName = rdr.GetString(2);
         app.YearOfBirth = rdr.GetInt32(3);
         if (!rdr.IsDBNull(4))
         {
            app.EMail = rdr.GetString(4);
         }
         app.FacultyId = rdr.GetInt32(5);
         app.FacultyName = rdr.GetString(6);
         app.EducationFormId = rdr.GetInt32(7);
         app.EducationFormName = rdr.GetString(8);
         if (!rdr.IsDBNull(9))
         {
            app.AdditionalInfo = rdr.GetString(9);
         }
         data.Add(app);
      }
   }
   finally
   {
      if (connection.State == ConnectionState.Open)
         connection.Close();
   }
   return data;
}
```

Листинг 20.13. Метод `NewApplicant()` класса `CollegeApplicantAction`

```
public bool NewApplicant(Applicant applicant)
{
   SqlCommand command = connection.CreateCommand();
   command.CommandType = CommandType.StoredProcedure;
   command.CommandText = "Applicant_Insert";
```

```
   command.Parameters.Add("@firstName", SqlDbType.NVarChar, 50);
   command.Parameters.Add("@lastName", SqlDbType.NVarChar, 50);
   command.Parameters.Add("@yearOfBirth", SqlDbType.Int, 50);
   command.Parameters.Add("@eMail", SqlDbType.NVarChar, 50);
   command.Parameters.Add("@facultyId", SqlDbType.Int);
   command.Parameters.Add("@educationFormId", SqlDbType.Int);
   command.Parameters.Add("@additionalInfo", SqlDbType.NVarChar, 255);

   command.Parameters["@firstName"].Value = applicant.FirstName;
   command.Parameters["@lastName"].Value = applicant.LastName;
   command.Parameters["@yearOfBirth"].Value = applicant.YearOfBirth;
   command.Parameters["@eMail"].Value = applicant.EMail;
   command.Parameters["@facultyId"].Value = applicant.FacultyId;
   command.Parameters["@educationFormId"].Value = applicant.EducationFormId;
   command.Parameters["@additionalInfo"].Value = applicant.AdditionalInfo;

   connection.Open();
   int retValue = 0;
   try
   {
      retValue = command.ExecuteNonQuery();
   }
   finally
   {
      if (connection.State == ConnectionState.Open)
         connection.Close();
   }
   if (retValue == 1) return true;
   else return false;
}
```

Листинг 20.14. Метод `GetFaculties()` класса `CollegeApplicantAction`

```
public List<Faculty> GetFaculties()
{
   List<Faculty> data = new List<Faculty>();
   data.Add(new Faculty());

   SqlCommand command = connection.CreateCommand();
   command.CommandText = "SELECT * FROM Faculty";
   connection.Open();
   try
   {
      SqlDataReader rdr = command.ExecuteReader();
      while (rdr.Read())
      {
         Faculty fac = new Faculty();
```

```
            fac.FacultyId = rdr.GetInt32(0);
            fac.FacultyName = rdr.GetString(1);
            data.Add(fac);
         }
      }
      finally
      {
         if (connection.State == ConnectionState.Open)
            connection.Close();
      }
      return data;
   }
```

Листинг 20.15. Метод `GetEducationForms()` класса `CollegeApplicantAction`

```
public List<EducationForm> GetEducationForms()
{
   List<EducationForm> data = new List<EducationForm>();
   data.Add(new EducationForm());

   SqlConnection connection = new SqlConnection(
      Properties.Settings.Default.CollegeApplicationConString);
   SqlCommand command = connection.CreateCommand();
   command.CommandText = "SELECT * FROM EducationForm";
   connection.Open();

   try
   {
      SqlDataReader rdr = command.ExecuteReader();
      while (rdr.Read())
      {
         EducationForm ef = new EducationForm();
         ef.EducationFormId = rdr.GetInt32(0);
         ef.EducationFormName = rdr.GetString(1);
         data.Add(ef);
      }
   }
   finally
   {
      if (connection.State == ConnectionState.Open)
      {
         connection.Close();
      }
   }
   return data;
}
```

В классе `CollegeService` будет реализован интерфейс `ICollegeService`. Код этого класса представлен в листинге 20.16.

Листинг 20.16. Класс `CollegeService`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using System.Data;
using System.Data.SqlClient;

namespace College.Service
{
   public class CollegeService : ICollegeService
   {
      CollegeApplicantAction act;

      public CollegeService()
      {
         act = new CollegeApplicantAction();
      }

      public List<Applicant> GetApplicants(int? applicantId,
         int? facultyId, int? educationFormId)
      {
         return act.GetApplicants(
            applicantId, facultyId, educationFormId);
      }

      public bool NewApplicant(Applicant applicant)
      {
         return act.NewApplicant(applicant);
      }

      public bool EditApplicant(Applicant applicant)
      {
         return act.EditApplicant(applicant);
      }

      public bool DeleteApplicant(int applicantId)
      {
         return act.DeleteApplicant(applicantId);
      }
```

```
        public List<Faculty> GetFaculties()
        {
            return act.GetFaculties();
        }

        public List<EducationForm> GetEducationForms()
        {
            return act.GetEducationForms();
        }
    }
}
```

Тестирование сервиса WCF

После создания сервиса WCF нам необходимо его протестировать. Visual Studio 2010 поставляется с хост-приложением, которое вы можете использовать для хостинга, выполнения и тестирования ваших сервисов. Вы настраиваете приложение вашего сервиса как стартовый проект Visual Studio. После этого вы можете выполнить ваше приложение в режиме отладки. При этом запустится хост-приложение, а также будет создан клиент для тестирования вашего сервиса. После запуска он виден в панели задач, и его можно открыть. Окно хост-сервиса показано на рис. 20.7.

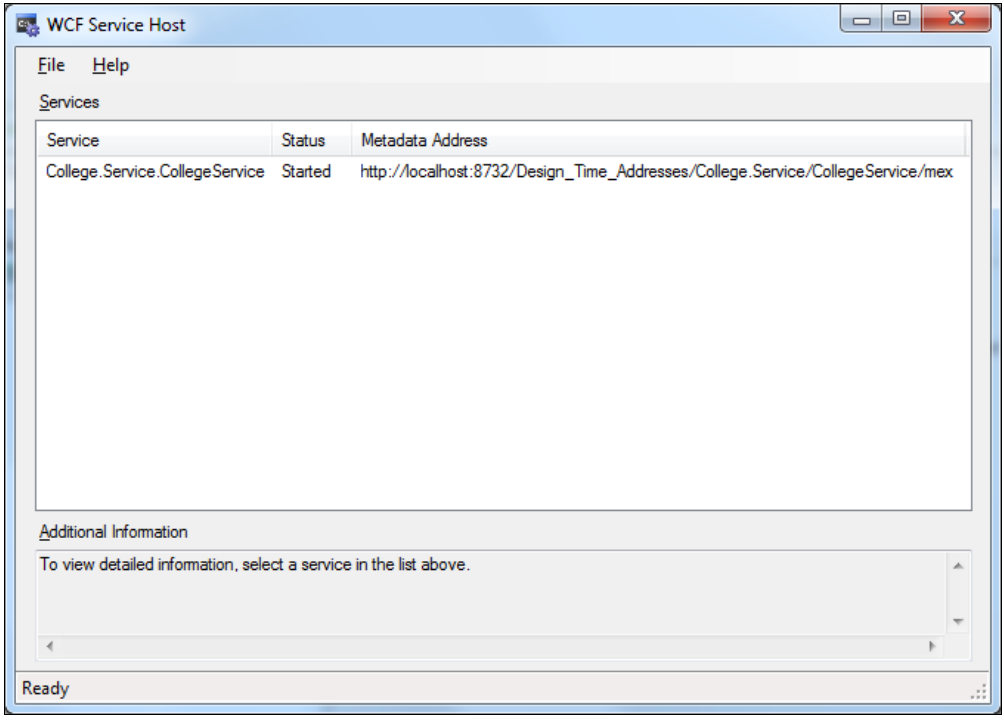


Рис. 20.7. Хост-сервис WCF

Тестовый клиент WCF представляет ваш сервис и его операции. На рис. 20.8 показан написанный в данном примере сервис (который выполняется внутри тестового клиента).

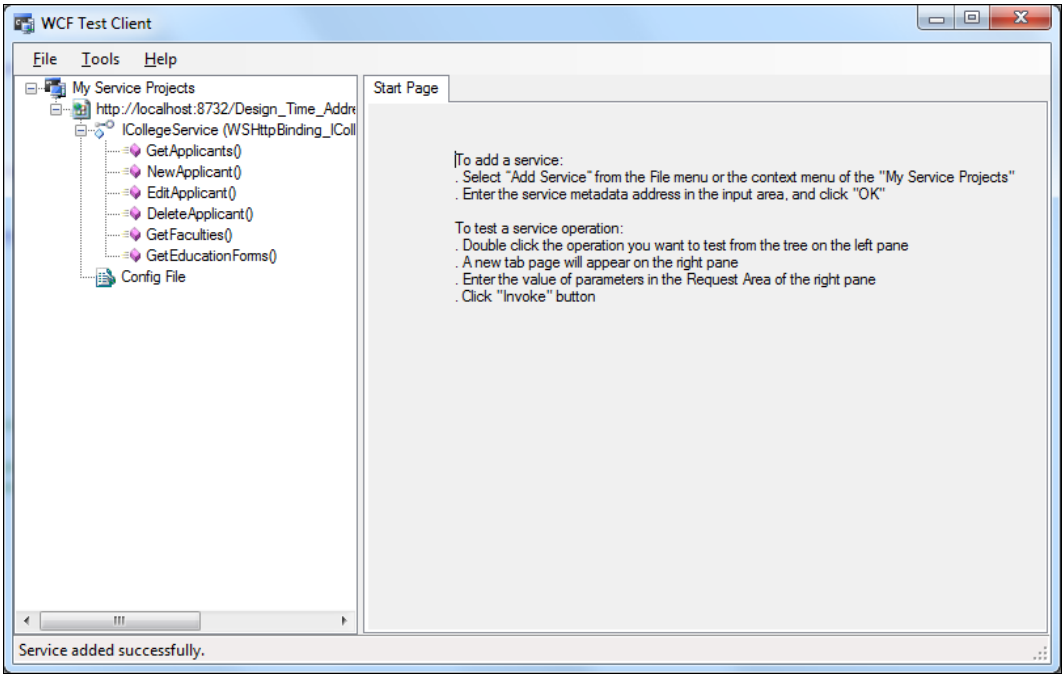


Рис. 20.8. Тестовый клиент WCF

Обратите внимание на интерфейс `ICollegeService`. Вы можете раскрыть его для того, чтобы увидеть операции вашего сервиса. Если вы дважды щелкнете по сервису, то в правой части получите результат вызова метода тестовым клиентом. Для примера мы можем протестировать сервисный метод `GetApplicants()`.

Для вызова сервиса введите параметры (если для метода необходимы параметры) на панели **Request** и нажмите кнопку **Invoke**. Результаты теста отобразятся на панели **Response**. На рис. 20.9 показаны результаты тестирования для сервисного метода `GetApplicants()` с параметрами `applicantId`, `facultyId`, `educationFormId`, равными `null` (возвращающими все записи из таблицы `Applicant`).

Вы можете также переключить отображение ваших результатов из представления **Formatted** в представление **XML**. Представление **XML** может быть полезным при отладке. Переключение представлений происходит при нажатии на вкладку **XML** в нижней части тестового клиента WCF. На рис. 20.10 показаны те же самые результаты в виде **XML**.

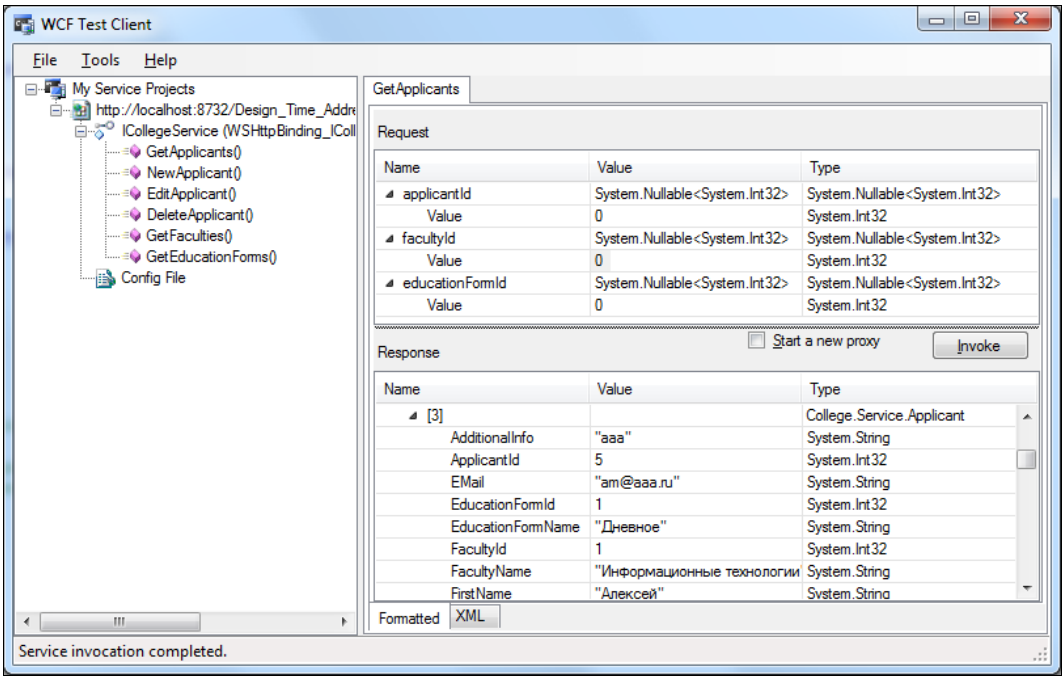


Рис. 20.9. Сервис, вызываемый тестовым клиентом WCF

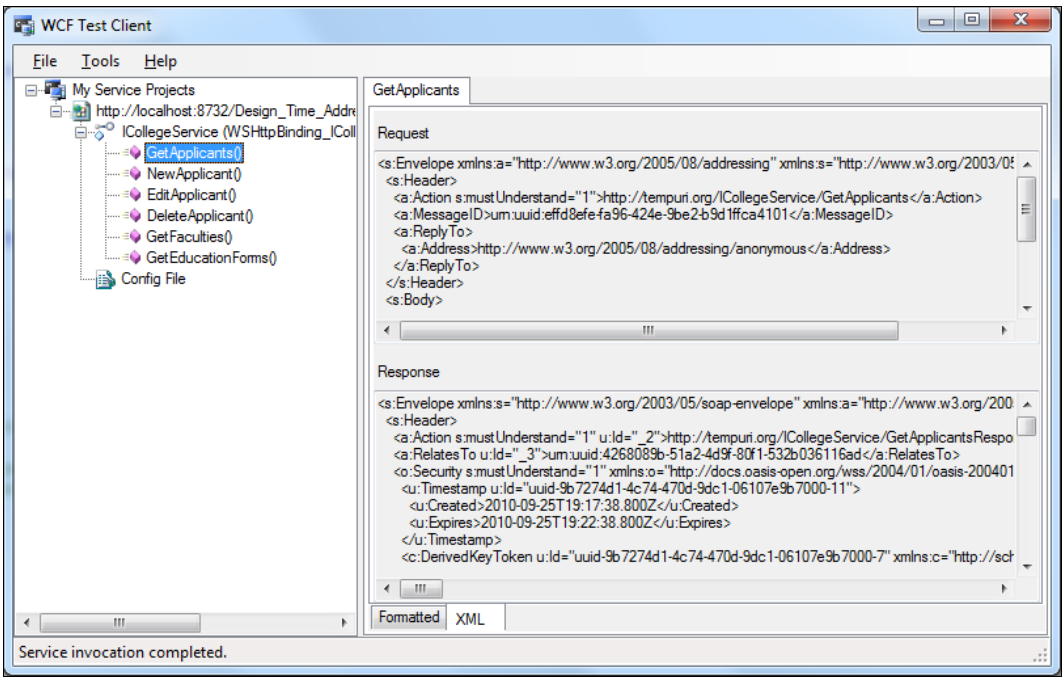


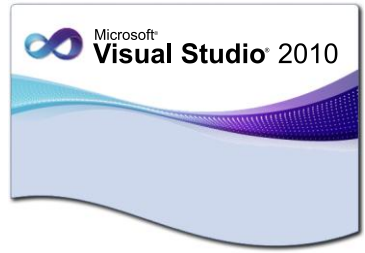
Рис. 20.10. Результаты тестового клиента в виде **XML**

Резюме

В этой главе вы узнали о том, как использовать Windows Communication Foundation для коммуникаций между клиентом и сервером. WCF является независимой от платформы технологией, подобно веб-службам ASP.NET, но предоставляет значительно большую функциональность.

Технология Windows Communication Foundation поддерживает развитую модель многопоточности и ограничения пропускной способности, которая позволяет управлять созданием экземпляров с минимальными усилиями. Вне зависимости от того, обрабатываются ли одновременно поступающие запросы в одном или в нескольких потоках, модель программирования остается одинаковой, так что разработчик может не вдаваться в детали.

ГЛАВА 21



Windows Workflow Foundation

Технология Windows Workflow Foundation (WWF) позволяет содавать рабочие процессы. Windows Workflow Foundation может значительно упростить разработку приложений, которые имеют дело со сложными и длительными процессами.

Библиотека рабочих процессов Windows Workflow Foundation впервые появилась в составе .NET Framework 3.0. В новой версии WWF в .NET Framework 4.0 существенно изменились принципы разработки рабочих процессов. Теперь рабочие процессы стало еще проще создавать, выполнять и поддерживать, была значительно повышена производительность, и значительно улучшилась интеграция между Windows Communication Foundation и Windows Workflow Foundation.

Кроме того, был полностью изменен визуальный конструктор WWF. Однако в новой версии Windows Workflow Foundation теперь отсутствует поддержка рабочих процессов конечных автоматов (State Machine Workflow), хотя, возможно, данный тип процессов снова появится в будущих версиях WWF.

Компоненты Windows Workflow Foundation

Windows Workflow Foundation состоит из трех основных компонентов:

- среда выполнения рабочих процессов;
- действия;
- визуальный конструктор рабочих процессов **Workflow Designer**.

Среда выполнения рабочих процессов предназначена для создания индивидуальных экземпляров рабочих процессов и обеспечения их выполнения и синхронизации. Среда выполнения рабочих процессов создается внутри приложения-хоста. Таким приложением-хостом может быть приложения WPF, Windows Forms, вебсайты ASP.NET, сервисы WCF.

Рабочий процесс представляет собой набор элементарных единиц, которые называются *действиями* и хранятся в виде модели, описывающей реальный процесс. Рабочие процессы позволяют описывать порядок выполнения этапов краткосрочных и долгосрочных работ, а также зависимости между этими этапами.

Конструктор **Workflow Designer** построен на основе Windows Presentation Foundation. В новой версии WWF он обеспечивает создание пользовательских действий рабочего процесса.

Среда выполнения WWF

Каждый работающий экземпляр рабочего процесса создается и обслуживается внутрипроцессной подсистемой среды выполнения. Рабочий процесс может взаимодействовать со средой выполнения с помощью одного из следующих объектов:

- `WorkflowInvoker`
- `WorkflowApplication`
- `WorkflowServiceHost`

Класс `WorkflowInvoker` предоставляет средства для вызова рабочего процесса. Он содержит методы для вызова рабочих процессов в синхронном или асинхронном режиме.

Класс `WorkflowApplication` представляет собой прокси-класс для экземпляра рабочего процесса. С помощью класса `WorkflowApplication` можно создавать новый экземпляр рабочего процесса или загружать экземпляр рабочего процесса из хранилища экземпляров, управлять выполнением экземпляра рабочего процесса. Этот класс используют для сохранения и выгрузки экземпляра рабочего процесса. Из объекта `WorkflowApplication` можно также получать уведомления о событиях жизненного цикла экземпляра рабочего процесса.

Класс `WorkflowServiceHost` предоставляет узел для служб, основанных на рабочем процессе. Этот класс используется для настройки и отображения рабочего процесса в виде службы.

С помощью этих классов можно выполнять рабочие процессы, используя для управления ими консольные приложения, приложения Windows Forms, веб-сайты ASP.NET и службы WCF.

Для создания и выполнения рабочего процесса проще всего использовать объект класса `WorkflowInvoker`. С помощью класса `WorkflowInvoker` можно выполнять следующие операции:

- производить синхронный вызов рабочего процесса;
- загружать входные данные для рабочего процесса;
- извлекать выходные данные из рабочего процесса.

Среда выполнения в WWF представлена как объект класса `ActivityInstance`. В домене приложения можно создавать несколько экземпляров класса `ActivityInstance`, которые могут выполняться параллельно.

Экземпляр класса `ActivityInstance` представляет собой потокобезопасный прокси-сервер, с помощью которого узлы рабочего процесса могут взаимодействовать со средой выполнения. Класс `ActivityInstance` используется для создания экземпляра рабочего процесса, получения уведомлений о событиях жизненного цикла экземпляра, управления выполнением рабочего процесса, предоставления входных

данных для рабочего процесса или извлечения выходных данных из него и сохранения данных рабочего процесса.

Среду для выполнения конкретного действия представляет класс `ActivityExecutionContext`. С помощью класса `ActivityExecutionContext` элементы рабочих процессов получают доступ к среде выполнения, а также используют этот класс для отслеживания состояния аргументов и переменных, планирования дочерних действий и многих других операций.

Взаимодействие между компонентами рабочего процесса

Для создания рабочего процесса используется метод `Invoke()` класса `WorkflowInvoker`. Он также используется для создания нескольких экземпляров рабочего процесса.

Класс `WorkflowInvoker` применяется для упрощенных рабочих процессов, не требующих управления с сервера. Рабочие процессы, требующие управления с сервера, должны выполняться с помощью метода `Run()`. Для того чтобы вызвать экземпляр рабочего процесса, совсем не обязательно ждать завершения другого процесса. Среда выполнения допускает одновременное выполнение нескольких экземпляров рабочего процесса.

Блоки обработки рабочих процессов инкапсулируются в виде действий.

Архитектура действий

Действие является базовой единицей для создания рабочего процесса. Класс `Activity` обеспечивает базовую абстракцию поведения рабочего процесса.

Рабочий процесс состоит из действий. Действия работают в исполняемой среде, которая обеспечивает средства для управления рабочим процессом, обработки исключений, распространения ошибок, сохранения данных состояния, загрузки и выгрузки текущих рабочих процессов из памяти, отслеживания и управления потоком транзакций.

Множество строительных блоков действий, обеспечивающих базовую логику и потоковые операторы, находится в библиотеке Base Activity Library (BAL, библиотека базовых действий). На рис. 21.1 показана иерархия классов действий Windows Workflow Foundation.

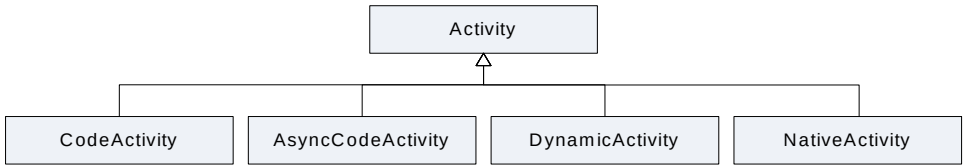


Рис. 21.1. Иерархия классов действий Workflow Foundation

При разработке приложения WWF программист создает собственные действия. Собственные действия должны наследоваться от классов `Activity`, `CodeActivity` `AsyncCodeActivity`, `DynamicActivity` и `NativeActivity`.

Действие, производное от класса `CodeActivity`, позволяет создавать логику выполнения в управляемом коде с использованием `CodeActivityContext` в основном для доступа к аргументам действий. Действие, производное от класса `NativeActivity`, позволяет получить доступ к среде выполнения рабочих процессов с помощью метода `ActivityExecutionContext()`. Доступ к среде выполнения рабочих процессов позволяет реализовать функциональность для создания закладок, асинхронных вызовов, регистрации транзакций и других операций.

Также действия могут создаваться в виде композиции других, уже существующих действий.

Жизненный цикл действия

Экземпляр действия запускается при выполнении рабочего процесса. Экземпляр действия остается в этом состоянии до тех пор, пока дочерние действия и любые другие ожидающие операции, например, объекты `Bookmark`, не будут завершены, после чего экземпляр переходит в закрытое состояние.

Всего существует три состояния завершения действия.

- `Closed` — действие завершило работу и выполнило выход.
- `Canceled` — действие правильно прекратило работу и выполнило выход. При переходе в это состояние откат операции не будет выполнен явным образом.
- `Faulted` — действие обнаружило ошибку и выполнило выход без завершения работы.

Родитель экземпляра действия может указать дочернему действию прекратить работу. Если дочернее действие может быть завершено, оно будет завершено в состоянии `Canceled`. Если при выполнении действия возникает исключение, среда выполнения переводит экземпляр `WorkflowElement` в состояние `Faulted` и распространяет исключение по родительской цепочке действий.

Действия остаются в состоянии выполнения во время сохранения или выгрузки рабочего процесса.

Рабочие процессы и действия

Рабочий процесс представляет собой коллекцию действий, которая моделирует процесс. Узел взаимодействует с рабочим процессом, используя `WorkflowInvoker` для вызова рабочего процесса, как если бы он был методом, `WorkflowInstance` для явного управления выполнением отдельного экземпляра рабочего процесса и `WorkflowServiceHost` для взаимодействия на основе сообщений в сценариях с несколькими экземплярами.

Так как шаги рабочего процесса определены в виде иерархии действий, то действие, занимающее верхнюю позицию в иерархии, определяет сам рабочий про-

цесс. Сами действия, в свою очередь, могут разрабатываться в виде коллекций других действий.

Модель данных действия

Действия сохраняют данные и обмениваются ими с помощью трех составляющих:

- переменных — для сохранения данных в действии;
- аргументов — для ввода и вывода данных из действия;
- выражений — действий с возвращаемым значением, которое используется для привязки аргументов.

Модель данных действия в Windows Workflow Foundation можно представить в виде диаграммы, показанной на рис. 21.2.

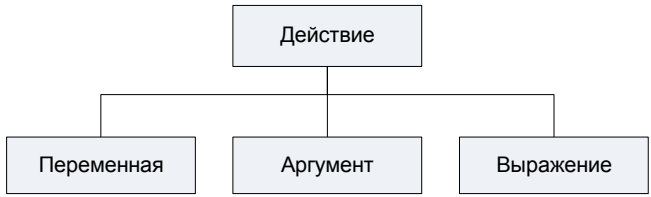


Рис. 21.2. Модель данных действия

Данные можно сохранить в действии с помощью переменной `Variable`. При перемещении данных в действие и из него для определения направления перемещения данных используются специальные типы аргументов. Такими типами являются:

- `InArgument`
- `InOutArgument`
- `OutArgument`

Выгрузка и сохранение рабочих процессов

Windows Workflow Foundation упрощает создание длительных реактивных программ путем обеспечения действий, получающих доступ к внешним входным данным. Также в Windows Workflow Foundation предусмотрена возможность создания объектов `Bookmark` (закладок), которые могут быть использованы для останова и последующего возобновления рабочего процесса.

Windows Workflow Foundation обеспечивает возможность сохранения данных рабочего процесса во внешнем хранилище данных, выгрузки рабочего процесса, повторной загрузки и активации рабочего процесса при возобновлении объектов `Bookmark` в определенном рабочем процессе.

Рабочий процесс выполняет действия до тех пор, пока не закончатся все действия либо пока все выполняемые действия не будут находиться в состоянии ожида-

ния входных данных. В состоянии ожидания рабочий процесс неактивен. Класс `WorkflowServiceHost` выгружает ставшие неактивными рабочие процессы и загружает их повторно при получении сообщения на продолжение выполнения.

Для блоков выполнения, использующих неустойчивое состояние или данные, которые не могут быть сохранены иначе, действие может указать узлу, что их не следует сохранять при помощи метода `ActivityExecutionContext`. Рабочий процесс также может явным образом сохранить данные во внешнем хранилище, например, в базе данных, при помощи действия `Persist`.

Создание проектов Windows Workflow Foundation

Для создания проекта Windows Workflow Foundation в среде Visual Studio 2010 используется узел **Workflow** в диалоговом окне **Add New Project**. Всего существует 4 варианта шаблонов проектов WWF (рис. 21.3).

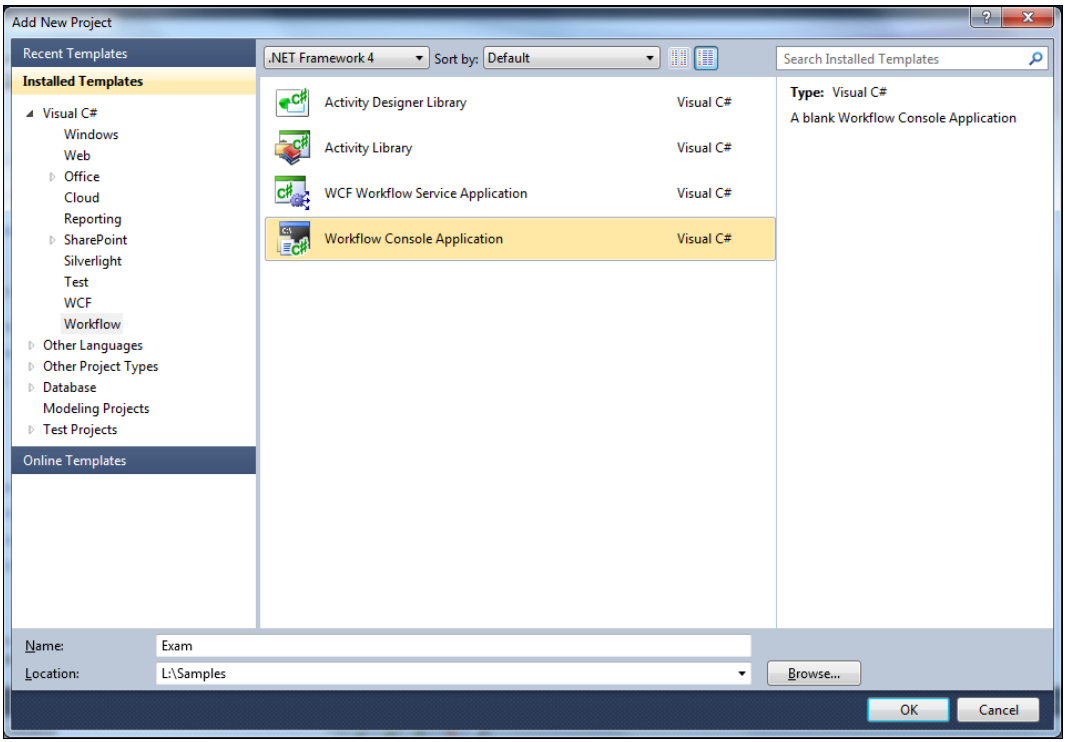


Рис. 21.3. Шаблоны проектов Windows Workflow Foundation

В качестве первого простейшего примера напишем рабочий процесс, который будет проверять сдачу экзамена студентом. В случае неуспешной сдачи экзамена (с оценкой ниже 3 баллов) студент будет направляться на повторную сдачу экзаме-

на. Создайте в окне **New Project** новое консольное приложение типа **Workflow Console Application** по имени `Exam`.

Среда разработки сгенерирует шаблон консольного приложения Workflow Foundation, структура которого представлена на рис. 21.4.

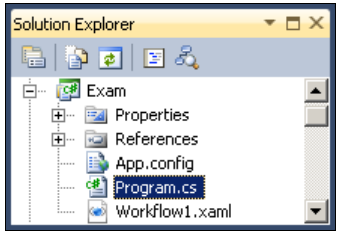


Рис. 21.4. Структура файлов проекта

Файл Program.cs содержит стандартный класс `Program`, такой же, как и в обычных консольных приложениях, с точкой входа в программу — методом `Main()`. Однако в теле метода `Main()` добавлен вызов рабочего процесса `WorkflowInvoker.Invoke()`.

Код файла Program.cs показан в листинге 21.1.

Примечание

Полный код приложения находится в каталоге Chapter21\Exam.v1 на прилагаемом к книге диске.

Листинг 21.1. Файл Program.cs

```
using System;
using System.Linq;
using System.Activities;
using System.Activities.Statements;

namespace Exam
{
   class Program
   {
      static void Main(string[] args)
      {
         WorkflowInvoker.Invoke(new Workflow1());
      }
   }
}
```

Класс `WorkflowInvoker` содержит методы для вызова рабочих процессов в синхронном режиме, а также методы экземпляра для вызова рабочих процессов в асинхронном режиме. Метод `Invoke()` вызывает рабочий процесс в синхронном режиме с помощью указанного определения рабочего процесса.

Файл конфигурации App.config содержит единственный раздел `<startup>` и вложенный подраздел `<supportedRuntime>`. Подраздел `<supportedRuntime>` имеет два обязательных атрибута:

- `version`, который указывает, какие версии среды CLR поддерживает приложение;
- `sku`, определяющий номер SKU для запуска приложения. SKU — это клиентский профиль .NET Framework 4, то есть подмножество компонентов .NET Framework 4, оптимизированное для клиентских приложений. Клиентский профиль содержит набор функций, достаточный для большинства клиентских приложений.

Код файла App.config приведен в листинге 21.2.

Листинг 21.2. Файл App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
       <supportedRuntime version="v4.0"
          sku=".NETFramework,Version=v4.0,Profile=Client" />
    </startup>
</configuration>
```

В файле Workflow1.xaml создается рабочий процесс приложения с помощью Workflow Designer — визуального конструктора рабочих процессов, который мы рассмотрим в следующем разделе этой главы.

Визуальный конструктор WWF

Визуальный конструктор Workflow Designer используется для построения рабочего процесса и содержит инструменты для его отладки и мониторинга. Визуальный конструктор рабочих процессов можно даже встроить в приложение, чтобы пользователь приложения мог самостоятельно изменять структуру рабочего процесса. Окно визуального конструктора Workflow Designer представлено на рис. 21.5.

Сначала мы создадим последовательный рабочий процесс. Для этого откройте панель **Toolbox** и перетащите мышью действие `Sequence` из группы **Control Flow** на рабочую поверхность визуального конструктора (рис. 21.6).

В рабочий процесс необходимо передавать информацию о полученной студентом оценке. В Windows Forkflow Foundation для передачи данных между действиями используются аргументы и переменные.

Аргументы и переменные

Аргументы и переменные уникальны для каждого работающего экземпляра рабочего процесса, и они хранят и возвращают значения. Формально аргументы и

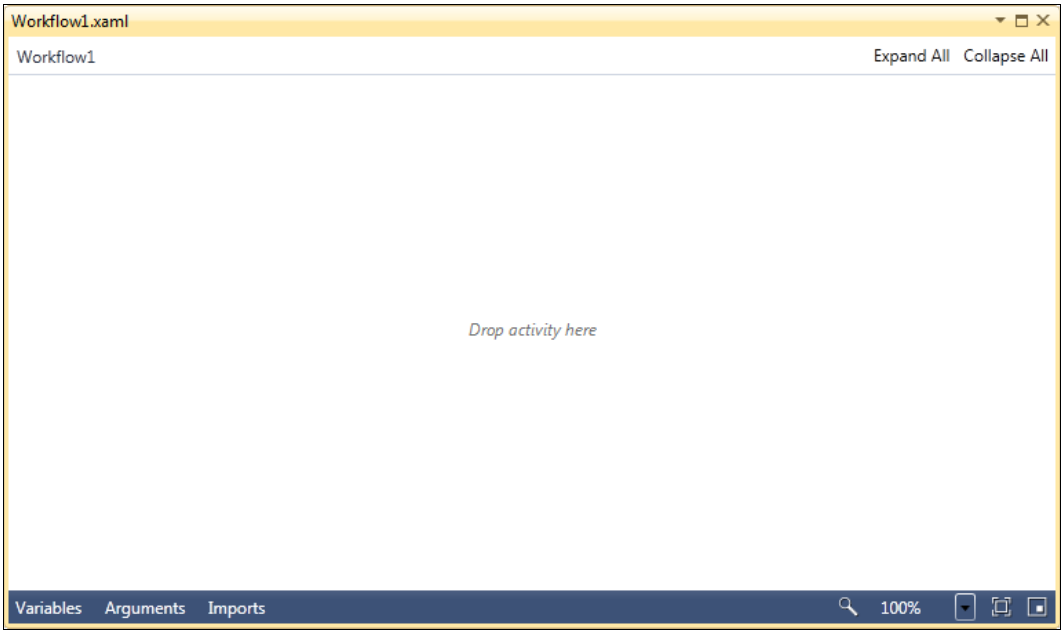


Рис. 21.5. Визуальный конструктор рабочих процессов

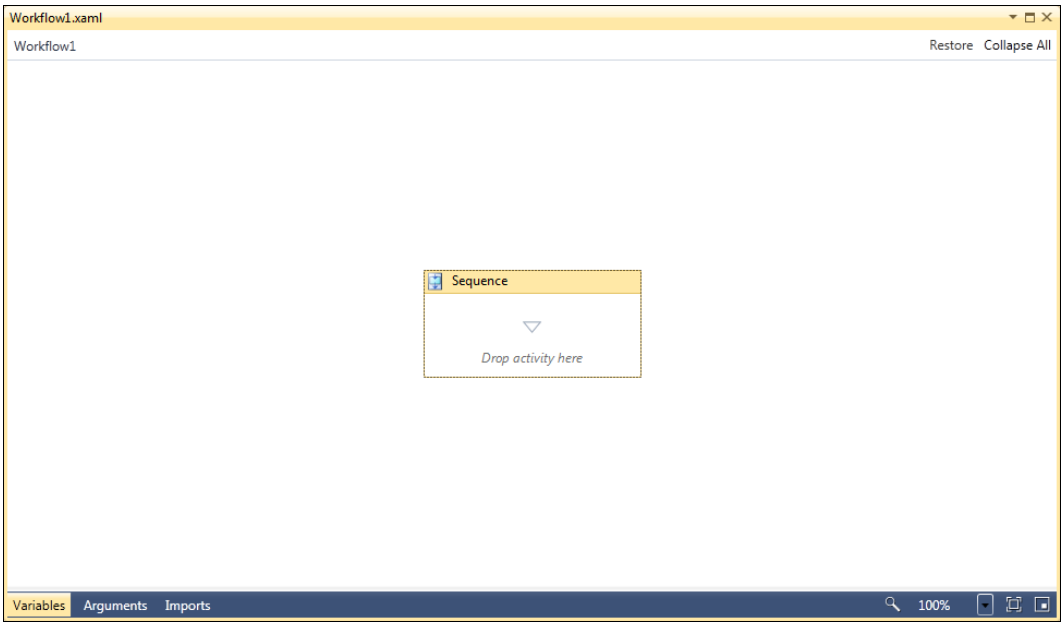


Рис. 21.6. Новый последовательный рабочий процесс

переменные не хранят значений, а вместо этого сообщают рабочему процессу, как он может получить эти значения.

> ***Примечание***
>
> В предыдущей версии Windows Workwlow Foundation для передачи данных между действиями приходилось применять свойства зависимости.

В нашем случае понадобится переменная `Grade` типа `integer`, которая будет хранить оценку, полученную студентом на экзамене. Переменная создается непосредственно в **Workflow Designer**.

Для создания переменной нажмите кнопку **Variables** в нижнем левом углу конструктора **Workflow Designer**, чтобы отобразить область **Variables**, и щелкните мышью на поле **Create Variable**. В поле **Name** введите имя переменной `Grade`, выберите тип **Int32** в раскрывающемся списке **Variable type**, в поле **Default** установите значение 0, а затем нажмите клавишу <Enter>, чтобы сохранить переменную (рис. 21.7).

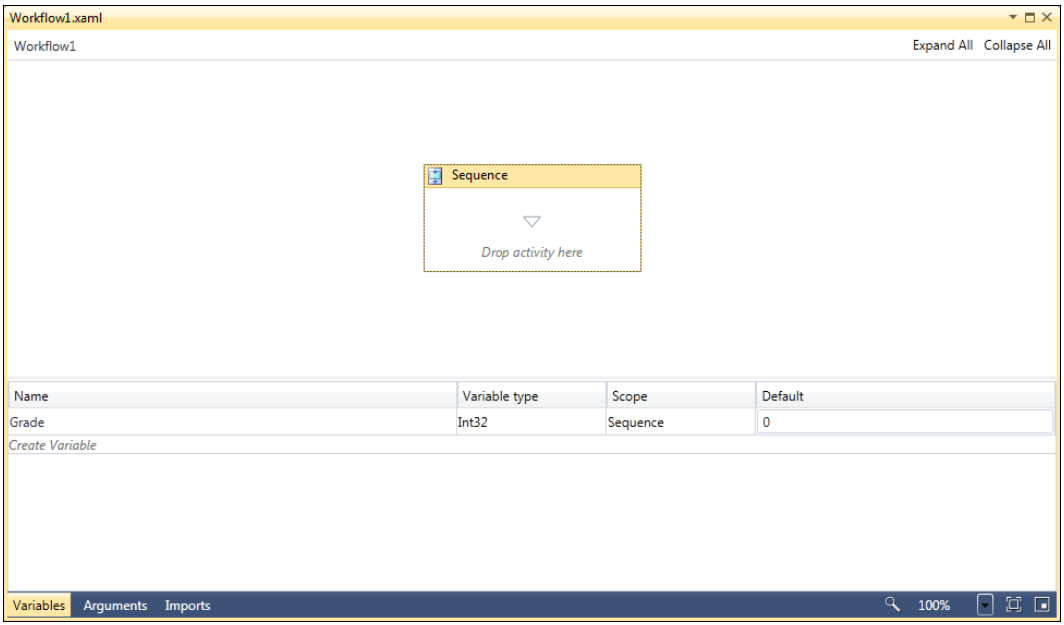


Рис. 21.7. Объявление переменной

Переменные и аргументы могут использоваться для ввода, вывода данных либо того и другого. Область видимости переменной зависит от того, где она объявлена. Поэтому, если переменная объявлена внутри действия, она будет видима его дочерним действиям, но не родительским.

Добавление действий

Первое действие, которое мы добавим в последовательный процесс, будет `WriteLine`. Это встроенное действие, которое позволяет выполнять вывод в окно

консольного приложения. Это действие будет использоваться для вывода сообщения, чтобы информировать пользователя о запуске рабочего процесса.

Для добавления действия перетащите действие `WriteLine` из группы **Primitives** панели **Toolbox** на действие `Sequence`, поместив его на серую стрелку внутри белого поля. Щелкните мышью на действии `WriteLine` и в окне **Properties** для свойства `Text` задайте значение `"Введите оценку (1-5)"`.

> ***Примечание***
>
> При написании значений в выражениях Workflow Foundation используется синтаксис выражений VB.NET, который требует заключения строк в кавычки.

Внешний вид рабочего процесса в конструкторе **Workflow Designer** теперь будет таким, как показано на рис. 21.8.

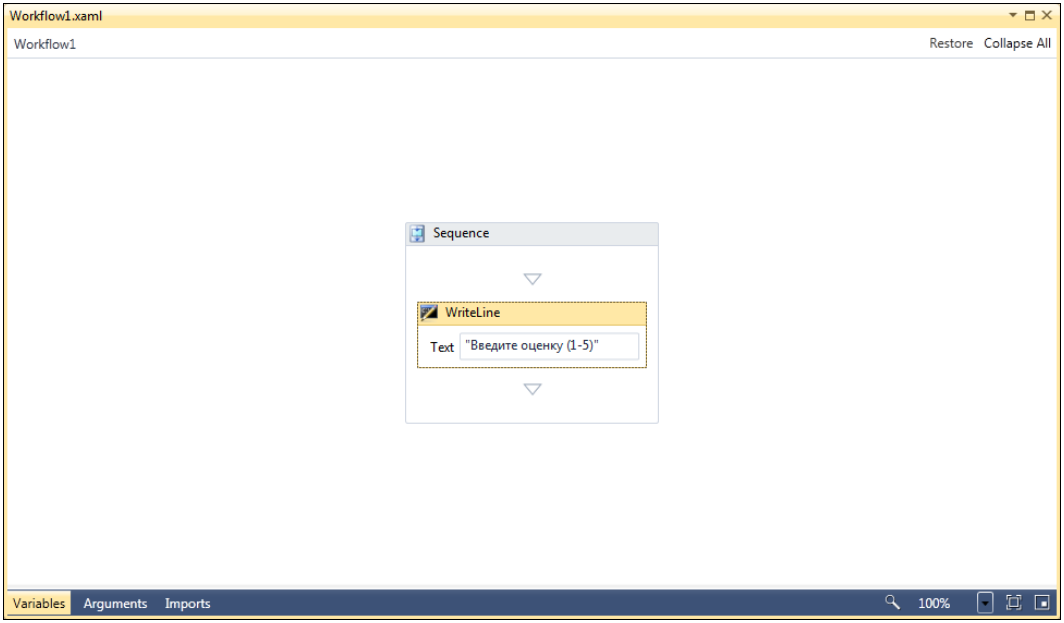


Рис. 21.8. Добавление действия `WriteLine`

Теперь нам нужно действие, которое читает пользовательский ввод с консоли. Однако среди действий, предоставляемых панелью **Toolbox**, такого нет. Поэтому мы создадим пользовательское действие, которое читает ввод оценки из консоли. Для этого создайте новый класс с названием `ReadGrade` и напишите в нем код, который представлен в листинге 21.3.

Листинг 21.3. Класс действия, читающего ввод с консоли

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

```
using System.Text;
using System.Activities;

namespace Exam
{
   public class ReadGrage : CodeActivity<Int32>
   {
      protected override Int32 Execute(CodeActivityContext context)
      {
         try
         {
            return Convert.ToInt32(Console.ReadLine());
         }
         catch
         {
            return 0;
         }
      }
   }
}
```

Если сейчас произвести компиляцию проекта, в окне **Toolbox** появится новый узел с именем нашего проекта **Exam**, содержащим значок созданного нами действия `ReadGrade`, как показано на рис. 21.9.

Перенесите это действие в окно **Workflow Designer** ниже действия `WriteLine`, как показано на рис. 21.10.

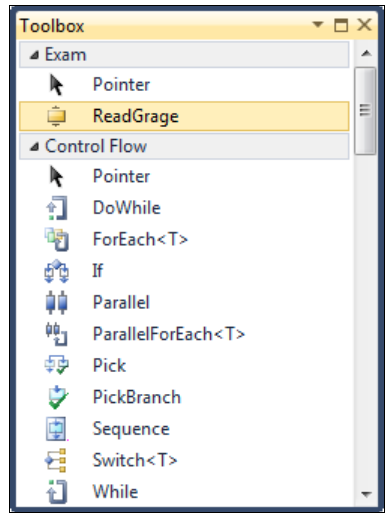


Рис. 21.9. Узел **Exam** и действие `ReadGrade`

Теперь надо связать переменную `Grade` со значением, которое будет вводиться пользователем в окно консольного приложения. Выделите действие `ReadGrade` в окне конструктора, откройте окно **Properties** и в свойстве `Result` укажите переменную `Grade`.

Далее необходимо проверить оценку, которую ввел пользователь, и если она больше или равна 3, то считать предмет успешно сданным. Для этого нам понадобится действие `If` из группы **Control Flow** панели **Toolbox**. Перетащите действие на поверхность визуального конструктора ниже действия `ReadGrade`. Дважды щелк-

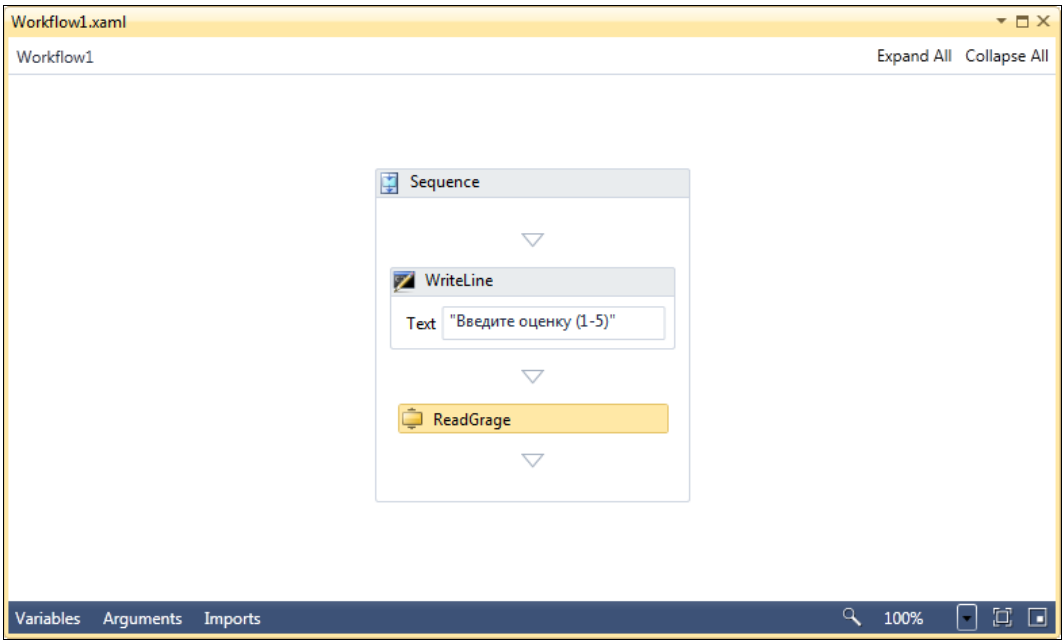


Рис. 21.10. Добавление действия `ReadGrade`

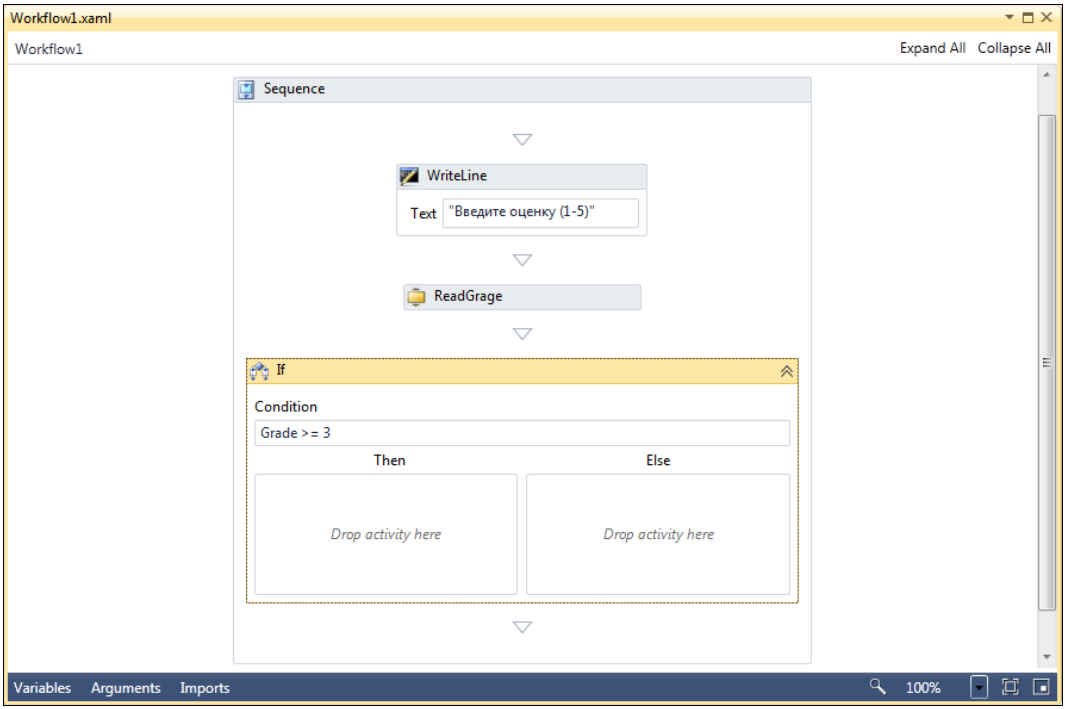


Рис. 21.11. Добавление действия `If`

ните мышью на действии `If`. В окне **Properties** выберите свойство `Condition` и щелкните мышью на кнопке с многоточием. В открывшемся окне **Expression Editor** введите следующее выражение:

```
Grade >= 3
```

и нажатием кнопки **OK** закройте окно.

Внешний вид схемы рабочего процесса в конструкторе теперь будет таким, как представлено на рис. 21.11.

> ***Примечание***
>
> Все выражения в конструкторе рабочих процессов записываются в синтаксисе VB.NET, причем независимо от того, на каком языке создается проект рабочего процесса: VB .NET или C# .NET.

Отображение вывода рабочего процесса

Теперь воспользуемся другим действием `WriteLine` для отображения вывода рабочего процесса. Перетащите два действия `WriteLine` в области `Then` и `Else` действия `If`. Присвойте свойствам `Text` этих действий значения `"Экзамен сдан"` для области `Then` и `"Экзамен не сдан"` для области `Else` (рис. 21.12).

Перед закрытием консольного окна необходимо увидеть вывод рабочего процесса, поэтому откройте файл Program.cs и добавьте вызов метода `Console.Read()` в конец метода `Main()`, как показано в листинге 21.4.

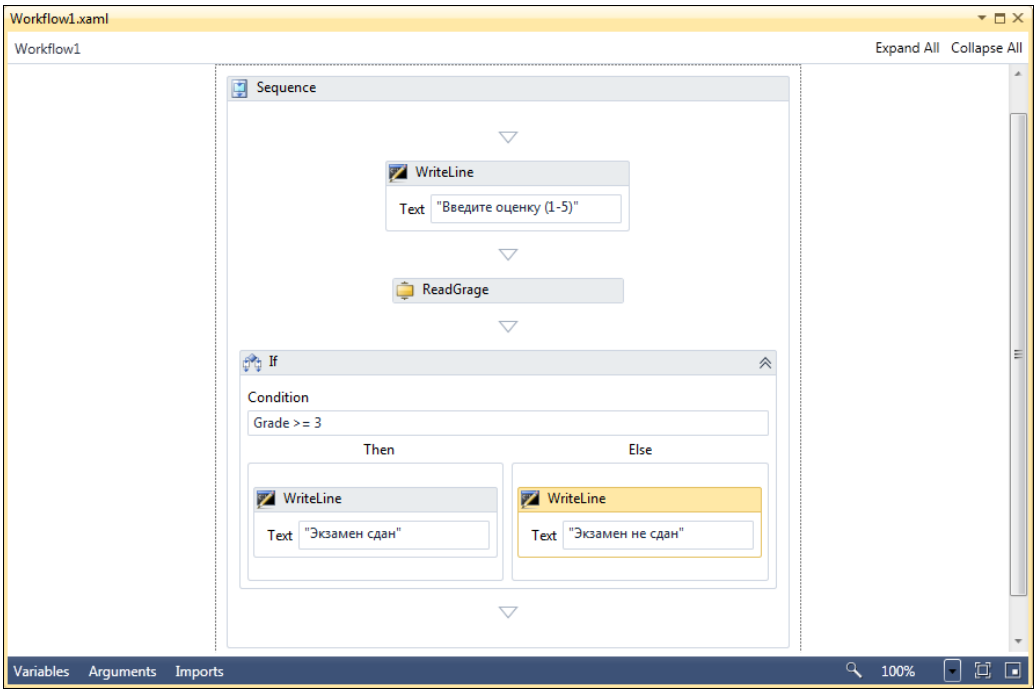


Рис. 21.12. Добавление действий для вывода рабочего процесса

Листинг 21.4. Измененный код метода `Main()`

```
static void Main(string[] args)
{
   WorkflowInvoker.Invoke(new Workflow1());
   Console.Read();
}
```

Скомпилируйте и запустите приложение. Внешний вид окна приложения и ввод данных показан на рис. 21.13.

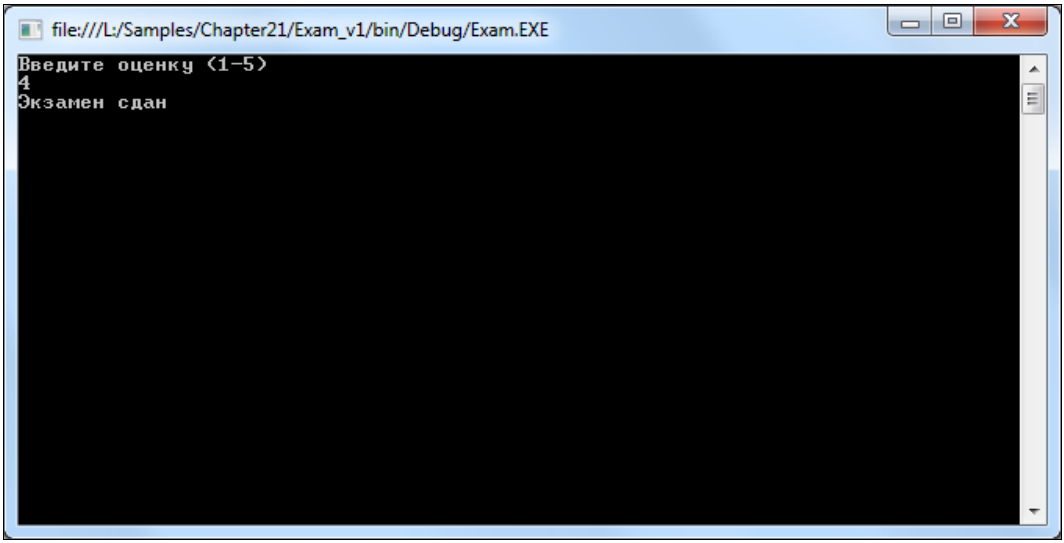


Рис. 21.13. Рабочий процесс в консольном окне

Создание циклических процессов

В этом разделе мы рассмотрим создание действий для выполнения циклических процессов. Для этого обычно используют два типа действий:

- `While`
- `DoWhile`

Действие *While*

Для приложения с циклическим рабочим процессом создайте в Visual Studio новый проект и разместите в **Workflow Designer** последовательный рабочий процесс, внутри которого поместите действия `WriteLine` и `ReadGrade`.

Примечание

Полный код приложения находится в каталоге Chapter21\Exam.v2 на прилагаемом к книге диске.

Добавьте после действия `ReadGrade` действие `While`. Действие `While` обеспечивает выполнение кода, пока истинно указанное условие. В нашем случае процесс будет выполняться до тех пор, пока студент не получит положительную оценку. Для этого установите в действии `While` выражение для свойства `Condition`:

```
Grade < 3
```

Внешний вид конструктора **Workflow Designer** с добавленными элементами представлен на рис. 21.14.

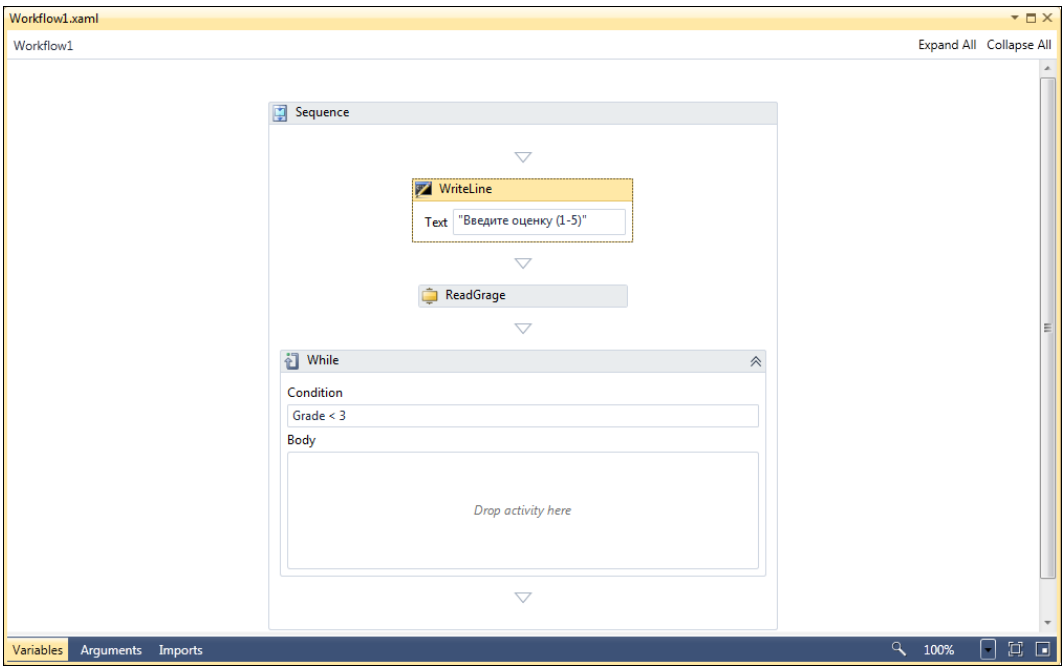


Рис. 21.14. Добавление действия `While`

В теле действия `While` поместите еще одно действие `Sequence`. В `Sequence` вложите два действия: `WriteLine` с текстом `"Пересдача. Введите оценку"` и `ReadLine`.

Примечание

Если в действии (в нашем примере это действие `While`) располагается больше одного вложенного действия, необходимо их поместить в действие `Sequence`, которое будет использоваться как контейнер для последовательных действий.

Окончательный внешний вид конструктора **Workflow Designer** для нашего приложения должен соответствовать рис. 21.15.

Скомпилируйте и запустите приложение. Внешний вид окна приложения и ввод данных показан на рис. 21.16.

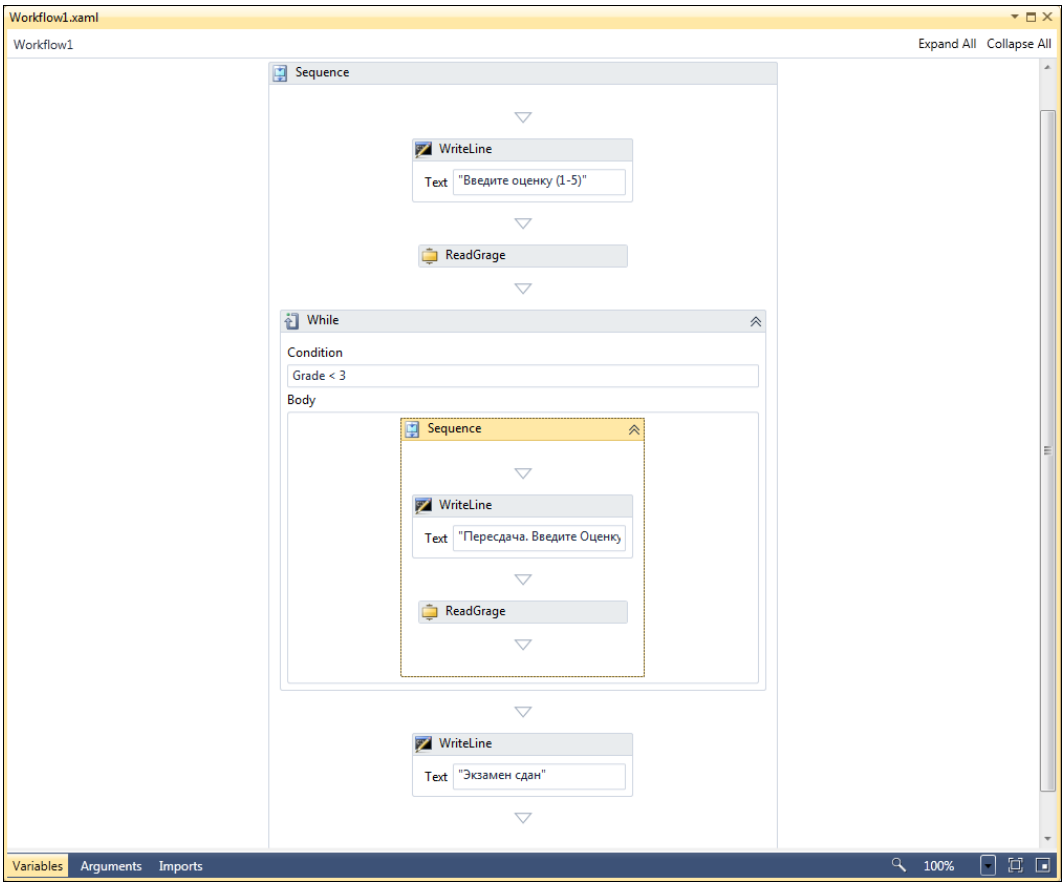


Рис. 21.15. Вложенный в действие `While` последовательный рабочий процесс

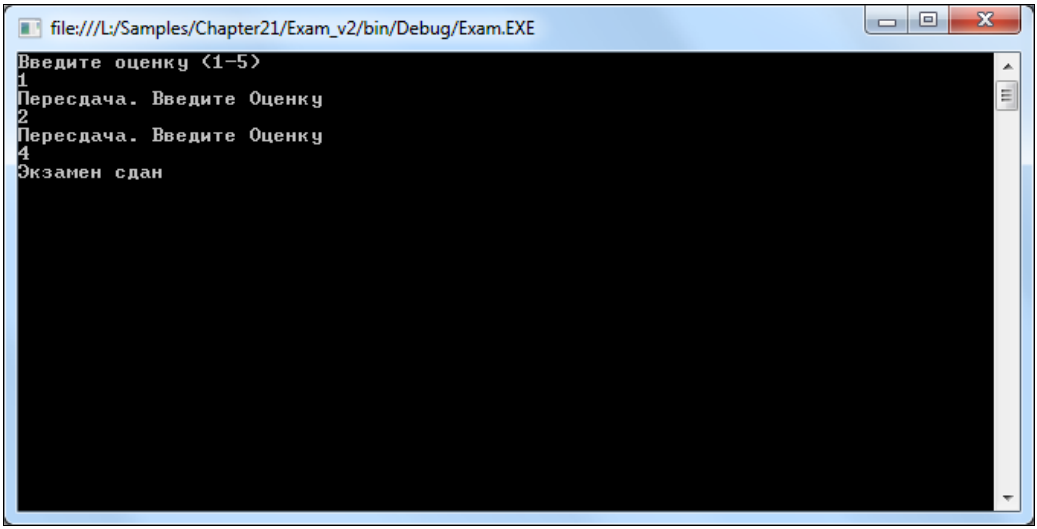


Рис. 21.16. Усовершенствованное приложение Workflow Foundation

Попробуйте вводить различные оценки и удостоверьтесь, что до тех пор, пока не будет введено значение как минимум 3, рабочий процесс будет сообщать: "Пересдача. Введите оценку".

Действие *DoWhile*

Действие `DoWhile` — это циклическое действие, которое выполняет содержащиеся действия хотя бы один раз, пока условие не примет значение `false`. Если мы применим его для нашего рабочего процесса, то можем упростить приложение, уменьшив количество действий. Для этого создайте в **Workflow Designer** рабочий процесс, аналогичный представленному на рис. 21.17.

> ***Примечание***
>
> Полный код приложения находится в каталоге Chapter21\Exam.v2.1 на прилагаемом к книге диске.

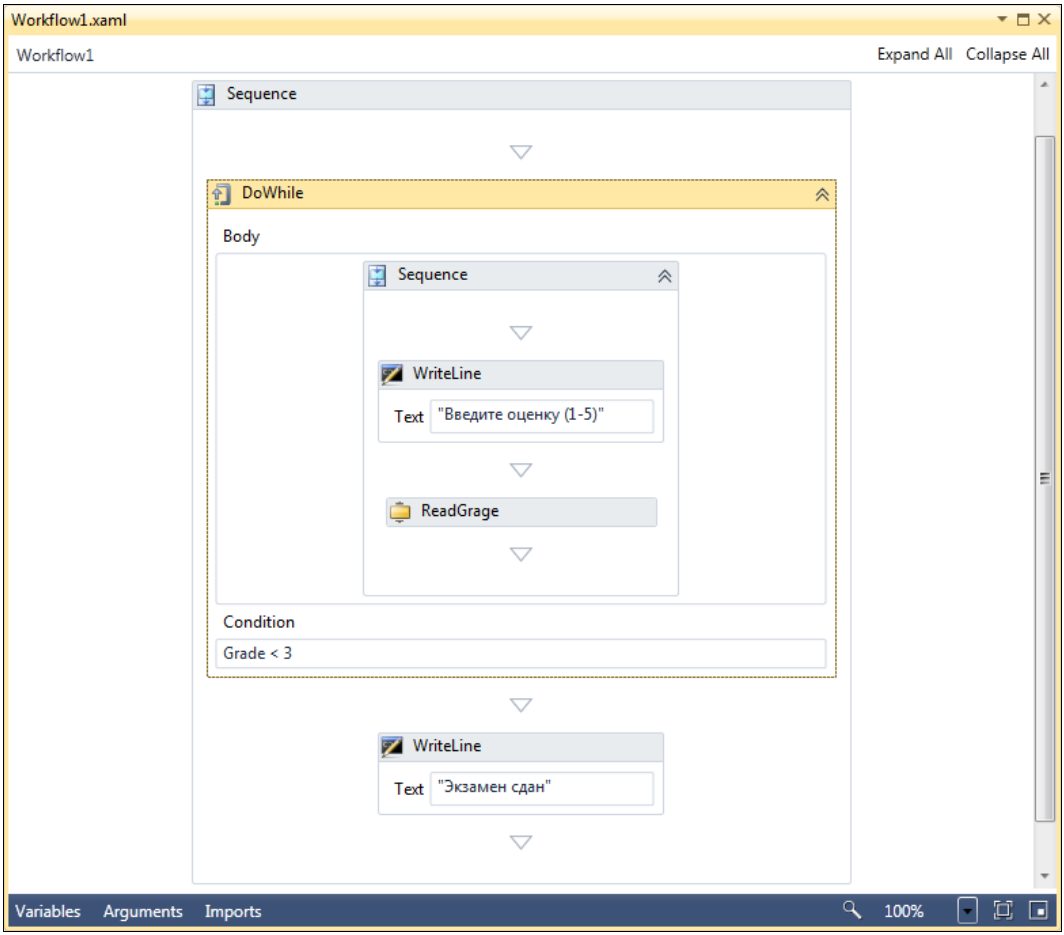


Рис. 21.17. Действие `DoWhile` в конструкторе рабочих процессов

Скомпилируйте и запустите приложение. Внешний вид окна приложения и ввод данных показан на рис. 21.18.

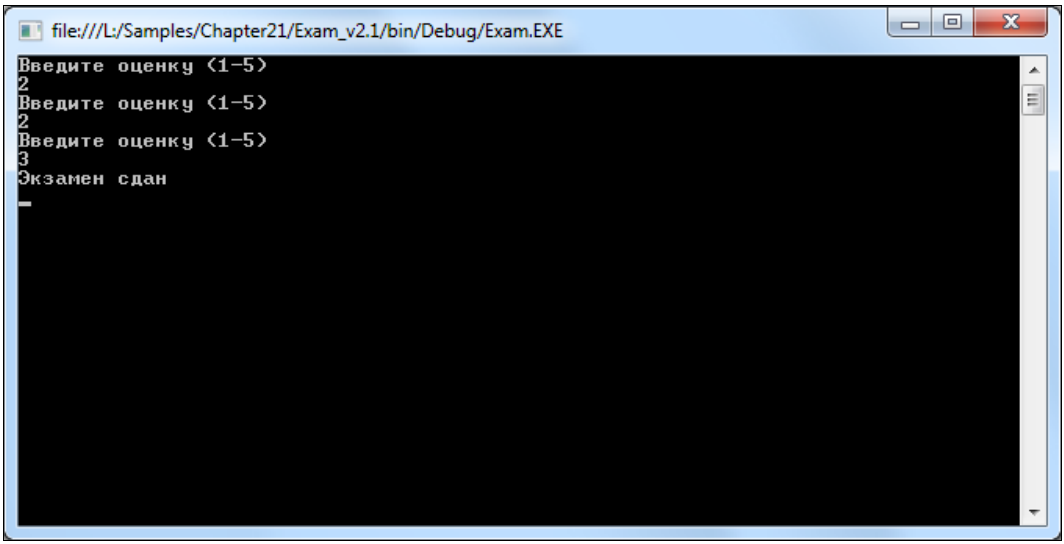


Рис. 21.18. Приложение с использованием действия `DoWhile`

Моделирование рабочих процессов с помощью блок-схем

Для моделирования рабочих процессов с помощью блок-схем используется `Flowchart` (блок-схема). `Flowchart` — это новый тип рабочего процесса, появившегося в новой версии Windows Workwlow Foundation, который облегчает решение определенных типов задач, например таких, которые могут возвращаться к уже ранее пройденным действиям.

В старой версии Windows Workflow Foundation смоделировать возврат к ранее пройденным действиям можно было с помощью цикла `While`, но рабочие процессы типа блок-схемы предлагают более понятный подход при разработке. Давайте для практики попробуем создать такой рабочий процесс.

Откройте в Visual Studio диалоговое окно **Add New Project** и создайте новое консольное приложение типа Workflow Console Application. Перетащите действие `Flowchart` из группы **Flowchart** на поверхность визуального конструктора.

> ***Примечание***
>
> Полный код приложения находится в каталоге Chapter21\Exam.v3 на прилагаемом к книге диске.

Представление дизайна для рабочего процесса типа блок-схем несколько отличается от последовательных рабочих процессов. Зеленый кружок показывает нача-

ло рабочего процесса. Внешний вид окна **Workflow Designer** с пустым действием `Flowchart` показан на рис. 21.19.

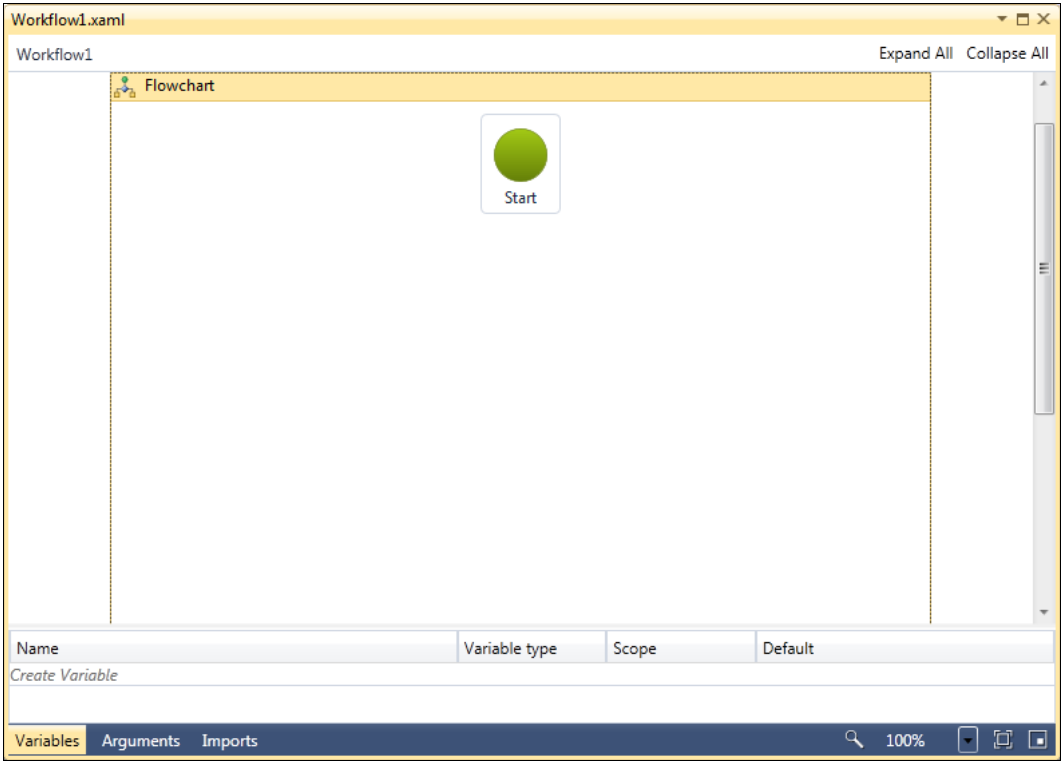


Рис. 21.19. Действие `Flowchart` в окне конструктора

Разместите внутри `Flowchart` действия `WriteLine` и `ReadGrade`. Теперь необходимо связать созданные действия. Наведите мышь на зеленый кружок, обозначающий начало рабочего процесса типа блок-схемы, и вокруг появятся три серых точки. Щелкните кнопкой мыши на той из них, которая находится снизу от кружка, и затем перетащите мышь вниз, на действие `ReadGrade`.

В группе `Flowchart` есть действие `Decision`, которое позволяет моделировать условный узел с двумя выходами. Попробуем это действие на практике, применив его в нашем приложении. Внешний вид рабочего процесса `Flowchart` с добавленными действиями показан на рис. 21.20.

Объект `FlowDecision`, представляющий это действие `Decision`, использует условие и определяет выполняемое действие, если значение условия равно `True` или `False`, аналогично действию `If`, рассмотренному ранее (рис. 21.21).

Задайте все необходимые условия для `FlowDecision`. Скомпилируйте и запустите приложение. Внешний вид окна приложения и ввод данных показан на рис. 21.22.

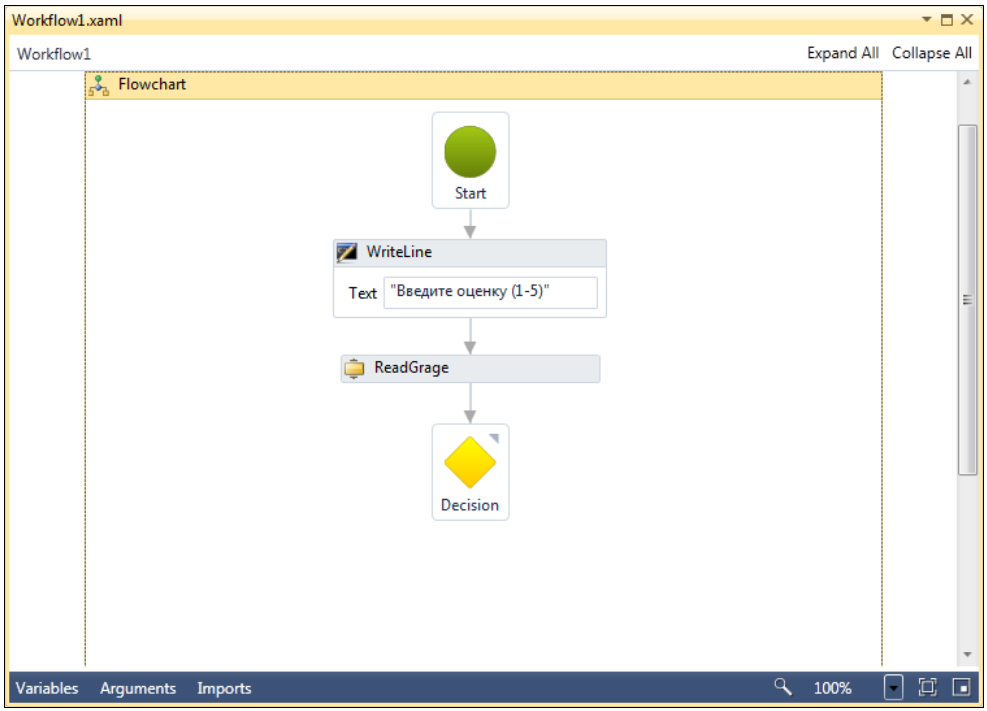


Рис. 21.20. Добавление действия `Decision`

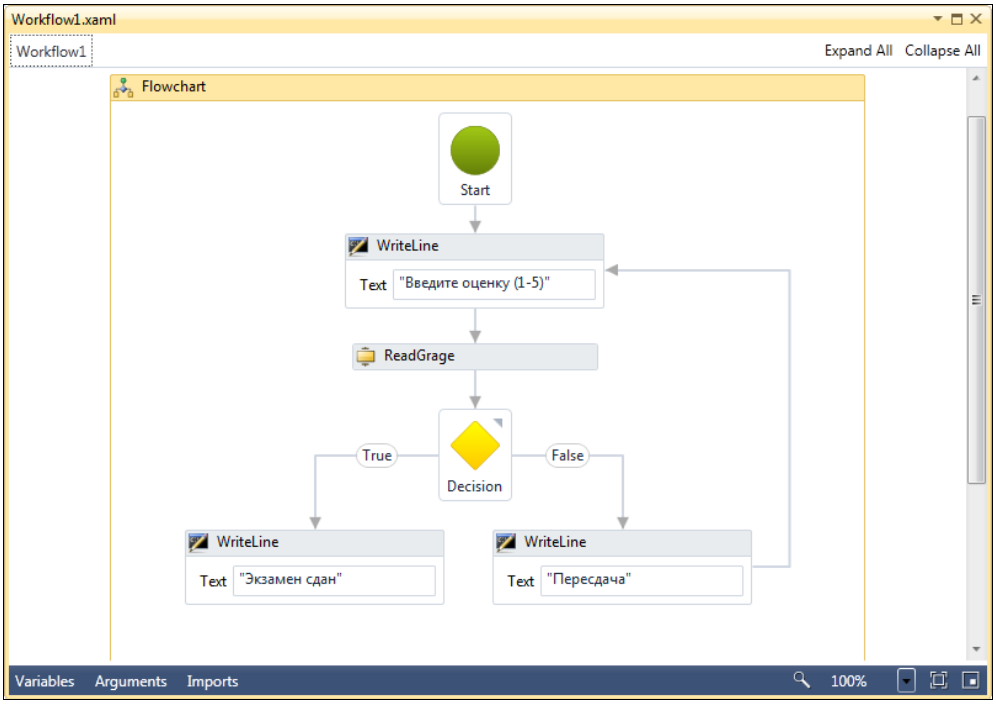


Рис. 21.21. Окончательная схема рабочего процесса

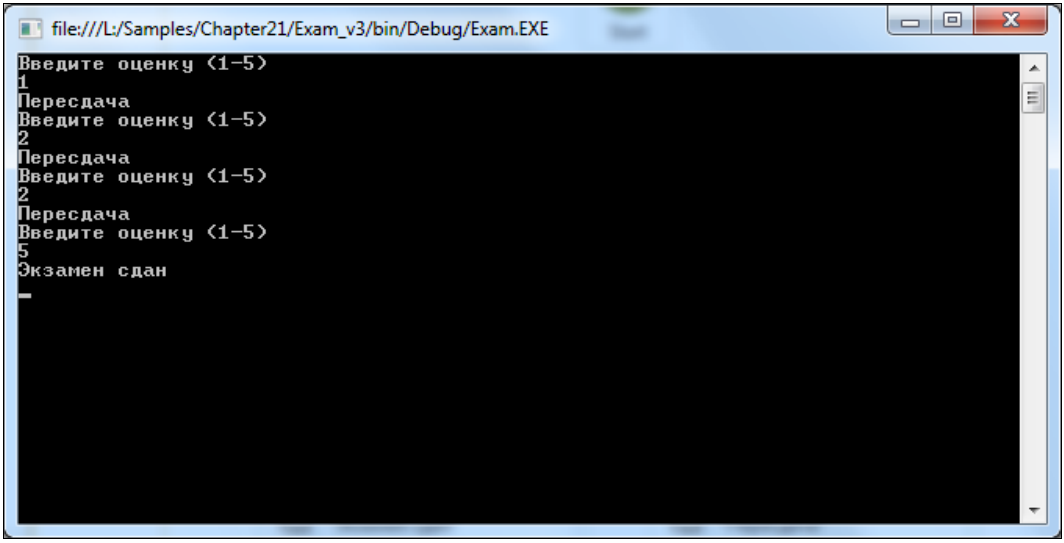
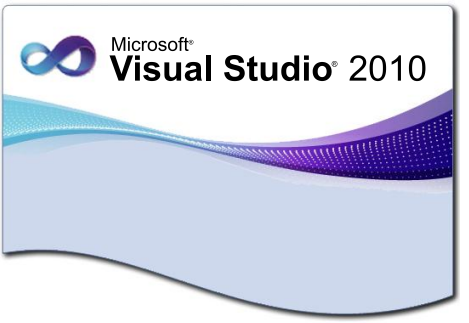


Рис. 21.22. Внешний вид окна приложения и ввод данных

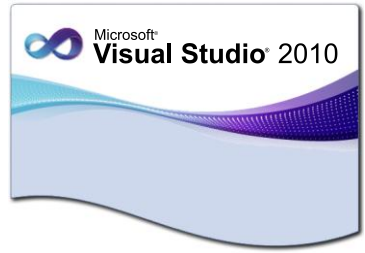
Резюме

В этой главе мы рассмотрели основы создания рабочих процессов с использованием технологии Windows Workflow Foundation. В новой версии Windows Workflow Foundation появилось множество изменений, что существенно упростило работу программистам в Visual Studio 2010. Добавление модели блок-схем и новых типов действий существенно облегчает моделирование разнообразных рабочих процессов, которые, возможно, придется разрабатывать в процессе вашей работы программистом.

ЧАСТЬ V



Локализация и развертывание приложений



ГЛАВА 22

Локализация приложений

Для успешной работы приложений на международном рынке необходима поддержка различных региональных стандартов. Среда разработки Visual Studio 2010 обладает необходимыми инструментами, которые позволяют достаточно легко создавать локализованные приложения.

Библиотека .NET Framework предоставляет возможности для разработки приложений, которые будут поддерживать различные языки, форматы денежных единиц, даты, времени и других значений.

Локализация позволяет изменять вид пользовательского интерфейса в зависимости от культуры. .NET Framework облегчает создание локализованных форм благодаря использованию файлов ресурсов. Эти файлы хранят данные для альтернативных версий форм, соответствующих различным культурам, поддерживаемым приложением.

Обычно ресурсы создаются для каждого региона, который должно поддерживать приложение. Эти ресурсы затем используются общеязыковой исполняющей средой CLR.

CLR определяет поведение по поиску ресурсов, специфических для данного региона. Исходя из этого, каждый набор ресурсов должен определять базовое имя, заданное в первой части имени файла ресурсов. Вторая часть имени, которая была опущена в первом примере этой главы, определяет регион. Если "региональная" часть в имени не будет указана, то ресурсы, определенные в файле ресурсов, будут использоваться в качестве ресурсов по умолчанию. Например, если базовым именем внедренного файла ресурса является MyResourceString.resx, то именем, специфическим для данного региона, будет MyResoureeStrings.en-US.resx.

Концепция культур

Под понятием *"культура"* подразумевается совокупность региональных стандартов, принятых в стране, где используется данное приложение. Культура определяется специальным значением, которое называется *код культуры*. Код культуры

описывает применяемый язык и сведения о регионе, где используется это приложение.

Обычно код культуры состоит из одной пары букв — в этом случае он определяет только язык, или из двух пар букв, разделенных дефисом, которые задают коды языка и региона соответственно Культуры, заданные двузначными кодами и определяющие только язык, называются *нейтральными*, например:

- en — английский язык без указания региона;
- ru — русский язык без указания региона.

Культуры, заданные четырехзначным кодом, определяющие язык и регион, называются *специфичными*. Вот примеры *специфичных* кодов культур:

- en-CA — вариант английского языка для Канады;
- en-UK — вариант английского языка для Великобритании;
- ru-RU — русский язык для России.

На основании этих кодов среда CLR автоматически находит ресурсы, специфические для данного региона. Если ресурсы определенного региона отсутствуют, используются ресурсы по умолчанию. Если ресурсы по умолчанию не включают значение для запрошенного ресурса, CLR генерирует исключение.

Информация, специфическая для данного региона, находится в экземпляре класса `CultureInfo` из пространства имен `System.Globalization`. В объекте `CultureInfo` могут храниться следующие данные:

- имя региона;
- форматы даты и времени;
- форматы чисел;
- форматы национальной валюты.

Определять файлы ресурсов, специфических для данного региона, в программном коде не обязательно. CLR может загрузить ресурсы, специфические для данного региона, из файлов ресурсов на основании региона, установленного на компьютере.

Таким образом, приложение может автоматически считывать код культуры, заданный параметрами, установленными на компьютере пользователя, и использовать соответствующую культуру.

Локализация приложений Windows Forms

Среда Visual Studio 2010 позволяет легко создавать локализованные приложения Windows Forms. Формы поддерживают свойство `Localizable`, которое определяет, является ли форма локализуемой. Чтобы сделать форму локализуемой, достаточно установить свойство `Localizable` в значение `true`.

Обычно локализуют строковые свойства элементов управления, однако при необходимости можно локализовать любое свойство элемента управления. Например, можно изменять размер кнопки в зависимости от длины надписи на ней, которая может отличаться для разных языков.

Версия формы, загружаемая в период выполнения приложения, определяется свойством `CurrentUICulture` класса `CultureInfo`. Свойство `CurrentUICulture` служит для установки или предоставления сведений об определенном языке и региональных параметрах.

В классе `CultureInfo` есть три свойства, которые определяют форматирование, применяемое при глобализации приложений:

- `DateTimeFormat`
- `NumberFormat`
- `TextInfo`

Свойства этих членов можно модифицировать для создания форматов, отличающихся от стандартных культур. Например, можно создать новый экземпляр класса `CultureInfo` на основе параметров культуры для русского языка, но использовать для символа валюты знак доллара:

```
CultureInfo customCulture = new CultureInfo("ru-RU");
modJPCulture.NumberFormat.CurrencySymbol = "$";
Thread.CurrentThread.CurrentCulture = customCulture;
```

Создание локализованного приложения

Для примера создания локализованного приложения возьмем простой проект из *главы 5* (на диске с примерами это будет проект Chapter05\TemplateApp), представляющий собой форму с главным меню, панелью инструментов и строкой состояния. В этот проект мы добавим форму для переключения языков пользовательского интерфейса, появляющуюся при запуске приложения.

> ***Примечание***
>
> Полный код приложения находится в каталоге Chapter22\TemplateApp на прилагаемом к книге диске.

В окне дизайнера форм выберите форму `MainForm` и в окне **Properties** присвойте ее свойству `Localizable` значение `true`.

Обратите внимание, что свойство `Language` по умолчанию установлено в `Default`. Текущие свойства пользовательского интерфейса станут параметрами, определяющими его облик при выборе языка по умолчанию. В окне **Properties** поменяйте значение свойства `Language` с `Default` на `Russian (Russia)` и установите свойство формы `Text` в значение `"Шаблон приложения"`. Проделайте такую же операцию для пунктов главного меню через их окна. В результате для формы `MainForm` будет сгенерирован файл ресурсов формы для русского языка (рис. 22.1).

Если дважды щелкнуть этот файл мышью, то он открывается в редакторе ресурсов, где созданные строковые ресурсы, при необходимости, можно дополнительно отредактировать (рис. 22.2).

> ***Примечание***
>
> Мы не перевели все пункты меню, вы можете, при желании, перевести и добавить их в ресурсы самостоятельно.

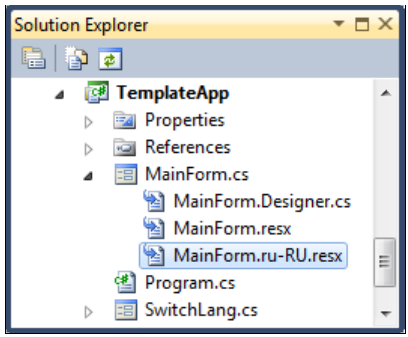


Рис. 22.1. Файл ресурсов для русского языка

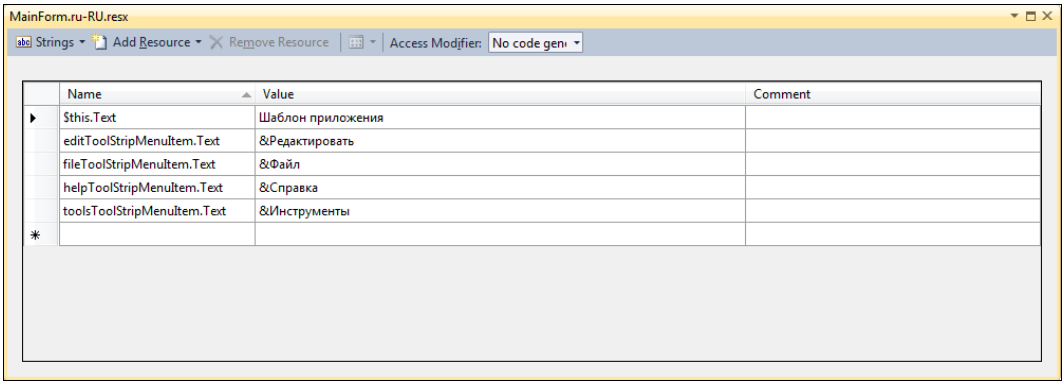


Рис. 22.2. Строковые ресурсы для формы `MainForm`

Теперь мы создадим форму для выбора языка с именем `SwitchLang`. На форме разместим элементы управления `Label`, `ComboBox` и `Button`. Внешний вид формы в дизайнере показан на рис. 22.3.

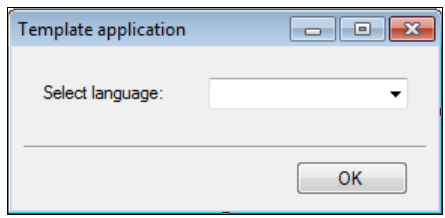


Рис. 22.3. Форма для выбора языка

Свойству `DialogResul` кнопки **OK** присвоим значение `DialogResult.OK`. Элементу `ComboBox` присвоим имя `cbLanguage` и создадим для него обработчик события `cbLanguage_SelectedIndexChanged()`. Кроме того, надо добавить в его коллекцию `Items` строки с названиями трех регионов, как показано на рис. 22.4.

В классе формы `SwitchLang` в теле обработчика события `SelectedIndexChanged()` для элемента управления `cbLanguage` надо написать код, который будет назначать приложению и его интерфейсу выбранную культуру. Код класса формы `SwitchLang` представлен в листинге 22.1.

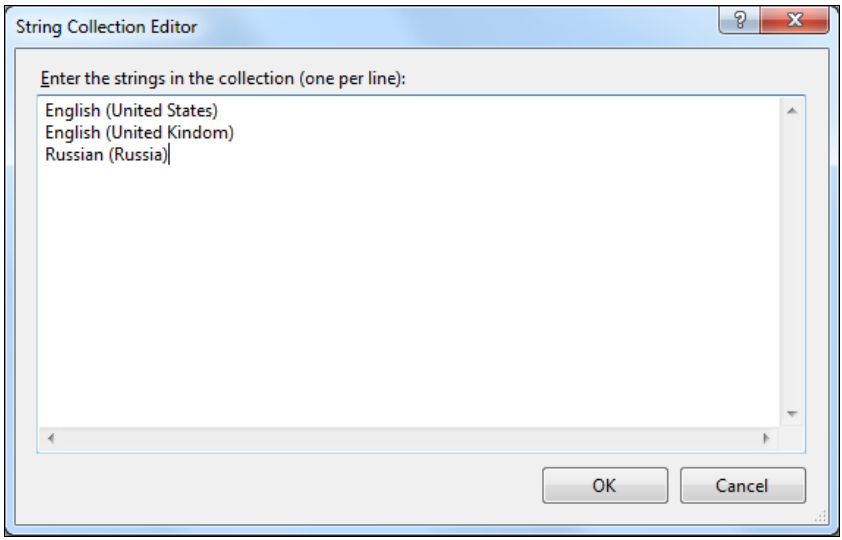


Рис. 22.4. Коллекция значений для выбора языка приложения

Листинг 22.1. Код класса формы `SwitchLang`

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

namespace TemplateApp
{
   public partial class SwitchLang : Form
   {
      public SwitchLang()
      {
         InitializeComponent();
      }

      private void cbLanguage_SelectedIndexChanged(
         object sender, EventArgs e)
      {
         switch (cbLanguage.SelectedIndex)
         {
            case 0:
               // Локализация по умолчанию — English (United States)
               this.Close();
               break;
```

```
            case 1:
               System.Threading.Thread.CurrentThread.CurrentCulture = new
                  System.Globalization.CultureInfo("en-GB");
               System.Threading.Thread.CurrentThread.CurrentUICulture =
                  new System.Globalization.CultureInfo("en-GB");
               this.Close();
               break;

            case 2:
               System.Threading.Thread.CurrentThread.CurrentCulture = new
                  System.Globalization.CultureInfo("ru-RU");
               System.Threading.Thread.CurrentThread.CurrentUICulture =
                  new System.Globalization.CultureInfo("ru-RU");
               this.Close();
               break;
         }
      }
   }
}
```

Откройте форму `MainForm` в редакторе кода и измените ее конструктор, добавив в него код, приведенный в листинге 22.2.

Листинг 22.2. Конструктор класса главной формы приложения `MainForm`

```
public MainForm()
{
   SwitchLang fLang = new SwitchLang();
   fLang.ShowDialog();

   InitializeComponent();

   slDataTime.Text = DateTime.Now.ToString();
}
```

В метке `slDataTime`, расположенной на строке состояния формы, будем выводить текущую дату и время для сравнения локализованных форматов даты/времени для различных культур.

Скомпилируйте приложение и протестируйте его. В первой форме выберите регион **Russian (Russia)** и посмотрите приложение с русской версией интерфейса (рис. 22.5). Обратите внимание на форматирование даты в строке состояния, принятой для российского региона.

Закройте приложение и запустите его еще раз, выбрав регион **English (United States)**. Обратите внимание, что загружается локализованная версия формы с соответствующим заголовком, главным меню, датой и временем в строке состояния. При этом дата и время отображаются согласно правилам, принятым в США (рис. 22.6).

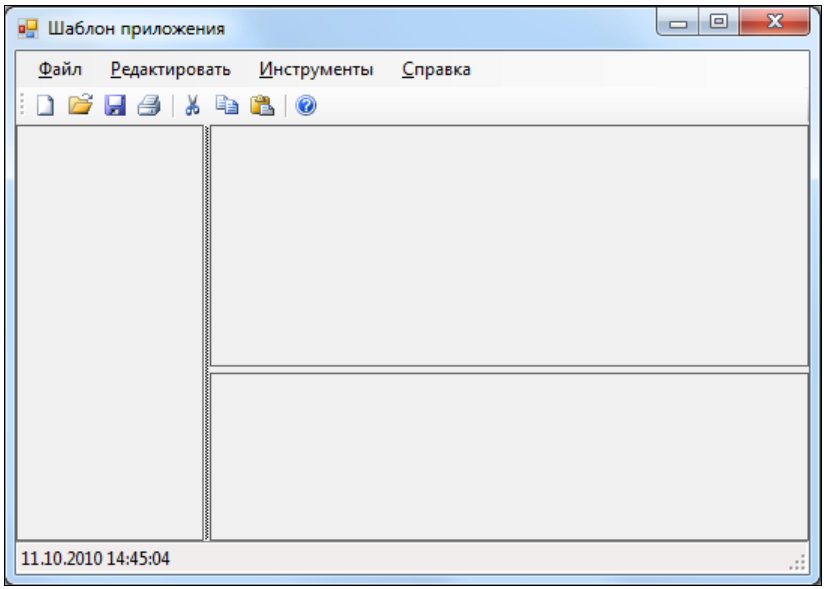


Рис. 22.5. Приложение с русским языком интерфейса

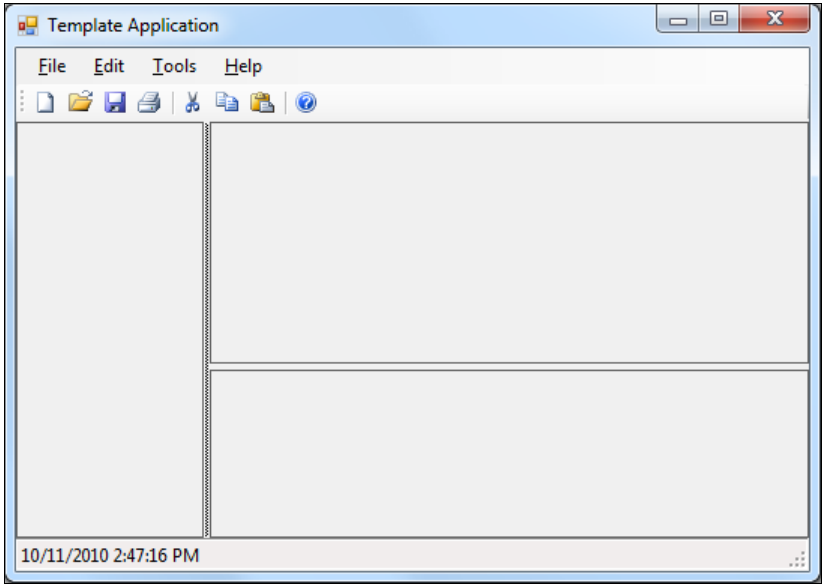


Рис. 22.6. Приложение с английским языком интерфейса для региона США

Закройте приложение и запустите его еще раз, выбрав регион **English (United Kingdom)**. Язык приложения остается английским, но при этом дата и время отображаются согласно правилам, принятым в Великобритании:

```
11/10/2010 17:56:38
```

Локализация веб-приложений

Для локализации веб-приложения используется аналогичный подход, применяемый для приложений Windows Forms, однако для веб-приложения среда разработки Visual Studio 2010 предоставляет отдельный инструмент — генератор ресурсов.

Для примера локализуем наше веб-приложение из *главы 13*. В этом приложении уже есть в наличии мастер-страница с навигацией и несколько страниц содержимого.

> ***Примечание***
>
> Полный код локализованного веб-приложения находится в каталоге Chapter22\CollegeWebSite на прилагаемом к книге диске.

Чтобы локализовать конкретную веб-страницу, необходимо открыть эту страницу в дизайнере веб-форм и выбрать в меню **Tools | Generate Local Resources** (Сгенерировать локальные ресурсы). Для начала мы локализуем мастер-страницу нашего веб-сайта MasterPage.master.

Среда разработки Visual Studio 2010 генерирует ресурсы для нескольких свойств каждого элемента управления, например, для каждого пункта меню `MenuItem` будут сгенерированы ресурсы для свойств `Text`, `Tooltip` и `Value`.

Сгенерированные идентификаторы ресурсов будут иметь префикс, обозначающий имя элемента управления, и суффикс, обозначающий имя свойства, например, `MenuItemResource1.Text`. Среда Visual Studio 2010 автоматически генерирует ресурсы по умолчанию только для тех элементов управления, которые находятся на веб-странице.

Файлы ресурсов для дополнительных культур добавляются вручную. Для этого копируют уже сгенерированные файлы ресурсов и присваивают им соответствующее имя, например MasterPage.master.ru-RU.resx для русского языка, а затем делают перевод строк в ресурсах на нужный язык.

> ***Примечание***
>
> Когда мы разрабатывали веб-сайт, мы уже изначально сделали для него интерфейс на русском языке. Обычно при создании приложений, которые будут локализовываться, сначала лучше создавать интерфейс по умолчанию на английском языке.

Для нашего веб-сайта с помощью генератора ресурсов создадим файлы ресурсов для русского и английского языка. На рис. 22.7 и 22.8 показаны соответственно файлы ресурсов MasterPage.master.ru-RU.resx для русского языка и MasterPage.master.resx для английского языка по умолчанию.

Файлы ресурсов будут добавлены в папку App_LocalResources веб-сайта, как показано на рис. 22.9.

Генератор ресурсов также создает запись для каждого свойства, помеченного атрибутом `[Localizable]` в элементе управления. Таким образом, если требуется создать специальный локализуемый элемент управления, нужно отметить все локализуемые свойства с помощью этого атрибута, например:

```
[Localizable(true)]
public string CourseName { get;  set; }
```

App_LocalResources/MasterPage.master.ru-RU.resx

Strings | Add Resource | Remove Resource | Access Modifier:

Name	Value	Comment
MenuItemResource5.Text	Главная страница	
MenuItemResource5.ToolTip		
MenuItemResource5.Value	Главная страница	
MenuItemResource6.Text	Учебные программы	
MenuItemResource6.ToolTip		
MenuItemResource6.Value	Учебные программы	
MenuItemResource7.Text	Абитуриентам	
MenuItemResource7.ToolTip		
MenuItemResource7.Value	Абитуриентам	
MenuItemResource8.Text	Контакты	
MenuItemResource8.ToolTip		
MenuItemResource8.Value	Контакты	
NavigationMenuResource2.ScrollDownText	Scroll down	
NavigationMenuResource2.ScrollUpText	Scroll up	
NavigationMenuResource2.SkipLinkText	Skip Navigation Links	
NavigationMenuResource2.ToolTip		
TreeNodeResource5.ImageToolTip		
TreeNodeResource5.Text	Учебные программы	
TreeNodeResource5.ToolTip		
TreeNodeResource5.Value	Учебные программы	

Рис. 22.7. Файл ресурсов мастер-страницы для русского языка

App_LocalResources/MasterPage.master.resx

Strings | Add Resource | Remove Resource | Access Modifier:

Name	Value	Comment
MenuItemResource5.Value	Main Page	
MenuItemResource6.Text	Education Programs	
MenuItemResource6.ToolTip		
MenuItemResource6.Value	Education Programs	
MenuItemResource7.Text	For Applicants	
MenuItemResource7.ToolTip		
MenuItemResource7.Value	For Applicants	
MenuItemResource8.Text	Contacts	
MenuItemResource8.ToolTip		
MenuItemResource8.Value	Contacts	
NavigationMenuResource2.ScrollDownText	Scroll down	
NavigationMenuResource2.ScrollUpText	Scroll up	
NavigationMenuResource2.SkipLinkText	Skip Navigation Links	
NavigationMenuResource2.ToolTip		
TreeNodeResource5.ImageToolTip		
TreeNodeResource5.Text	Education Programs	
TreeNodeResource5.ToolTip		
TreeNodeResource5.Value	Education Programs	
TreeNodeResource6.ImageToolTip		

Рис. 22.8. Файл ресурсов мастер-страницы для английского языка

После создания локализованных файлов ресурсов среда CLR сможет инициализировать свойства элемента управления на основании свойства `CurrentUICulture` потока с использованием строк, содержащихся в файле внедренных ресурсов для определенного региона.

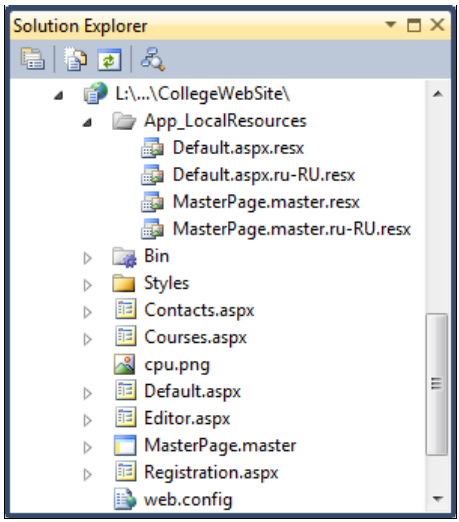


Рис. 22.9. Файлы дополнительных ресурсов в каталоге App_LocalResources

При генерации ресурсов среда разработки Visual Studio для каждого элемента управления, расположенного на веб-странице, добавляет выражение локализации:

```
meta:resourcekey
```

В результате код страницы MasterPage.Master будет выглядеть, как показано в листинге 22.3.

Листинг 22.3. Фрагмент кода с элементом управления `Menu` мастер-страницы

```
<asp:Menu ID="NavigationMenu" runat="server" CssClass="menu"
  EnableViewState="False" IncludeStyleBlock="False"
  Orientation="Horizontal"
  meta:resourcekey="NavigationMenuResource2">
  <Items>
    <asp:MenuItem Text="Главная страница"
      NavigateUrl="~/Default.aspx"
      meta:resourcekey="MenuItemResource5"/>
    <asp:MenuItem Text="Учебные программы"
      NavigateUrl="~/Courses.aspx"
      meta:resourcekey="MenuItemResource6"/>
    <asp:MenuItem Text="Абитуриентам"
      NavigateUrl="~/Registration.aspx"
      meta:resourcekey="MenuItemResource7"/>
    <asp:MenuItem Text="Контакты"
      NavigateUrl="~/Contacts.aspx"
      meta:resourcekey="MenuItemResource8"/>
  </Items>
</asp:Menu>
```

Все элементы для локализации идентифицируются посредством атрибута `meta:resourceKey`. При разборе страницы среда CLR перебирает все элементы управления и генерирует необходимый код.

Объект `Request` также имеет свойство `UserLanguages`. Это свойство содержит информацию о языке, которую посылает браузер в HTTP-заголовке на сервер. Свойство `UserLanguages` можно использовать при загрузке веб-страницы, в обработчике события `Page_Load()` мастер-страницы, как показано в листинге 22.4.

Листинг 22.4. Файл MasterPage.master.cs и определение текущей культуры

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Globalization;
using System.Threading;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class MasterPage : System.Web.UI.MasterPage
{
    protected void Page_Load(object sender, EventArgs e)
    {
       CultureInfo cultInfo;
       if (Request.UserLanguages != null &&
          Request.UserLanguages.Length > 0)
       {
          cultInfo = new CultureInfo(Request.UserLanguages[0]);
          Thread.CurrentThread.CurrentUICulture = cultInfo;
       }
       else
       {
          cultInfo = Thread.CurrentThread.CurrentUICulture;
       }

    }
}
```

Однако отправка информации о регионе с помощью HTTP-заголовка является для клиентского браузера необязательной и может быть недоступна для сервера, на котором установлено веб-приложение. Поэтому часто устанавливают регион, используемый по умолчанию, в файле web.config приложения, используя элемент `globalization`, как показано в листинге 22.5.

Листинг 22.5. Установка региона по умолчанию в файле web.config

```
<configuration>
  <system.web>
    <globalization enableClientBasedCulture="true"
      culture="ru-RU" uiCulture="ru-RU"/>
    ...
```

Если теперь мы запустим наш веб-сайт в браузере, мы увидим, что меню и дерево навигации отображаются на английском языке (при установленных локальных настройках для английского региона), как показано на рис. 22.10.

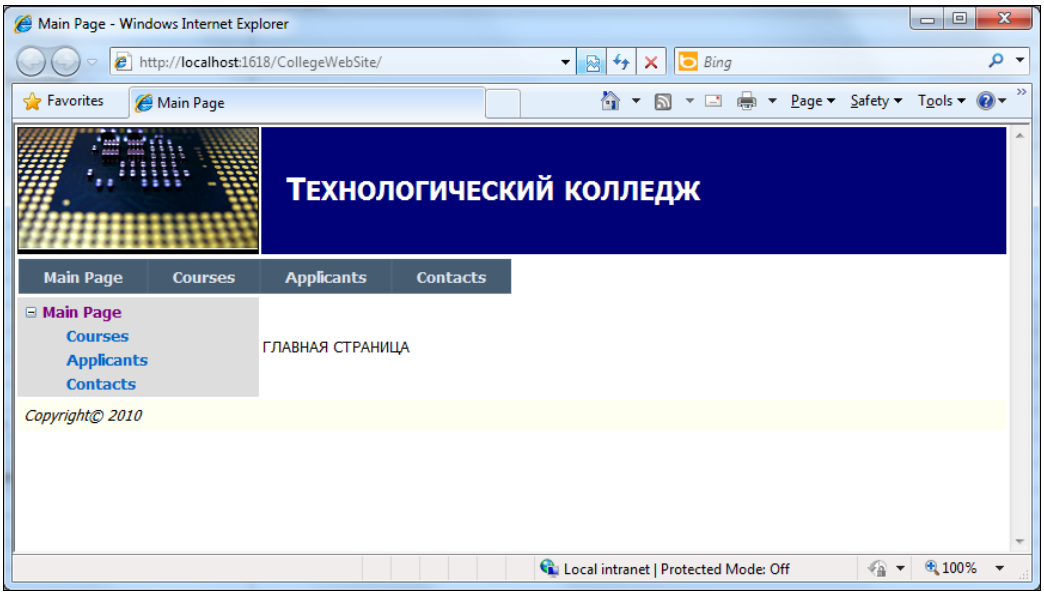


Рис. 22.10. Веб-страница с локальными настройками English

Однако заголовок и строка текста остались на русском языке. Эти ресурсы являются статическим текстом и не имеют атрибута `runat="server"`. Поскольку они не обрабатываются на стороне сервера, генератор ресурсов Visual Studio их не "видит", и в файле ресурсов их не будет.

Для решения этой проблемы в ASP.NET имеется элемент управления `Localize`, в который необходимо поместить статический текст:

```
<asp:Localize ID="Localize1" runat="server"
   meta:resourcekey="Localize1Resource1"
   Text="ГЛАВНАЯ СТРАНИЦА">
</asp:Localize>
```

или таким способом:

```
<h1>
  <asp:Localize ID="Localize1"
    meta:resourcekey="Localize1Resource1" runat="server">
```

```
      Технологический колледж
  </asp:Localize>
</h1>
```

Если этот элемент управления будет охватывать некоторый текст, то текстовая часть страницы будет автоматически включена в процесс генерирования ресурсов подобно любому другому элементу управления, и элемент будет локализован, как и остальные ресурсы на странице (рис. 22.11).

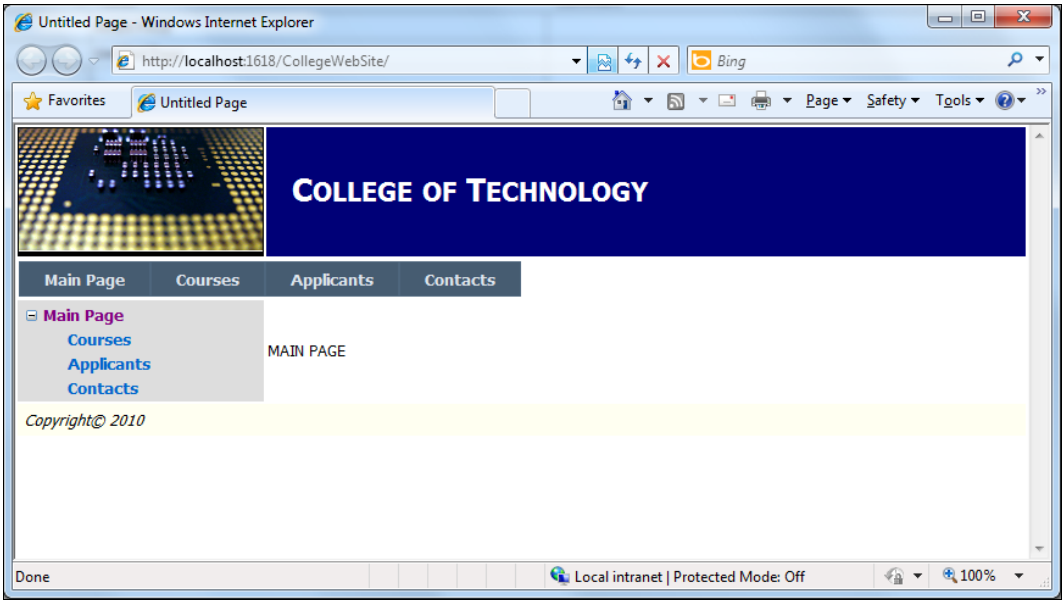
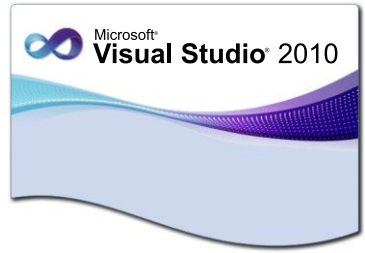


Рис. 22.11. Полностью локализованная страница

Резюме

В этой главе рассматривались основы создания локализованных приложений для Windows Forms и веб-приложений ASP.NET. Также было описано, как среда CLR загружает ресурсы для приложений в зависимости от региональных настроек на компьютере пользователя, и как можно работать с ресурсами в программном коде.



ГЛАВА 23

Развертывание приложений

При создании большого приложения процесс его развертывания на целевых компьютерах может быть сложным. Для успешного развертывания приложения необходимо также развернуть все компоненты, на которые ссылается приложение. Например, большинство приложений, созданных с помощью Visual Studio 2010, зависят от платформы .NET Framework. Перед установкой приложения на конечном компьютере должна присутствовать требуемая версия среды CLR. Средства развертывания, входящие в состав Visual Studio 2010, позволяют установить платформу .NET Framework и другие компоненты в процессе установки приложения.

Эта глава предоставляет способы развертывания приложений, которые доступны для разработчиков .NET в среде Visual Studio и которые мы еще не рассматривали в предыдущих главах.

Простые приложения, созданные в Visual Studio, можно развертывать, просто копируя их каталоги на клиентские компьютеры. Для развертывания более сложных приложений в Visual Studio 2010 предусмотрена технология Windows Installer, позволяющая создавать проекты установочных программ с широкими возможностями настройки.

Windows Installer

Технология Windows Installer — это управляемая данными служба установки и настройки, поставляемая с операционной системой Windows. Windows Installer используется уже длительное время, более 10 лет, и является частью операционных систем Windows XP, Windows Vista и Windows 7, а также доступен для старых систем Windows 95, Windows 98 и Windows NT 4.0. Текущая версия на момент написания книги — Windows Installer 4.5.

Программа Windows Installer отслеживает инсталляцию приложений в своей базе данных. В случае, если потребуется деинсталляция приложения, Windows Installer позволяет легко отследить и удалить параметры настройки системного реестра, файлы приложения на жестком диске, значки в меню **Start**, меню **Programs** и на рабочем столе, добавленные при инсталляции приложения.

Если некоторый файл деинсталлируемого приложения может использовать другое приложение, Windows Installer оставляет его на клиентском компьютере. Благодаря этому при удалении приложения предотвращается возможность нарушения работы другого приложения.

База данных Windows Installer также позволяет выполнить восстановление приложения, если ключи системного реестра или библиотеки, связанные с приложением, будут повреждены или удалены.

При создании инсталляционных пакетов для приложений основная проблема состоит в том, чтобы идентифицировать все внешние ресурсы, которые потребуются для приложения, включая файлы конфигурации, различные сторонние библиотеки, скрипты для развертывания базы данных и т. д.

Создание проекта развертывания

Среда Visual Studio 2010 предоставляет пять проектов установки, основанных на технологии Microsoft Windows Installer:

- **Setup Wizard** — создание мастера установки для клиентских приложений;
- **Setup Project** — создание установки для клиентских приложений;
- **Web Setup Project** — создание установки для веб-приложений;
- **Cab Project** — создание CAB-файла;
- **Merge Module Project** — создание модулей слияния.

Все эти проекты установки доступны в окне **New Project** в узле **Other Project Types | Setup and Deployment | Wizual Studio Installer**.

Проекты развертывания предлагают большую гибкость и настройку для процесса установки приложения. Один из этих проектов — **Setup Wizard** — наиболее подходит для развертывания больших приложений, и в то же время является наиболее сложным в настройке.

Проект типа **Setup Wizard** мы и рассмотрим в качестве примера создания инсталлятора для нашего приложения. Назовем его `College.Setup` (рис. 23.1).

Мастер **Setup Project** из Visual Studio .NET позволяет выбрать для вашего проекта подходящий тип установочной программы. Направляя действия пользователя, он помогает быстро создать заготовку проекта установочной программы, которую затем "доводят" с помощью Visual Studio .NET.

Чтобы добавить к решению проект устанавливаемой программы, выберите в меню **File** команду **Add Project | New Project** — откроется диалоговое окно **Add New Project**. Далее вызовите мастер **Setup Project** — для этого на панели **Project Types** выберите **Setup and Deployment Projects,** а в окне **Templates** — **Setup Wizard** (рис. 23.2).

Щелкните кнопку **Next**, чтобы открыть страницу **Choose a project type** (рис. 23.3). На этой странице мастера задают тип проекта. Если нужно создать проект установочной программы для приложения, выберите **Create a setup for a Windows application**, а если для установки файлов библиотек, то укажите **Create a merge module for Windows Installer**. Щелкните кнопку **Next**.

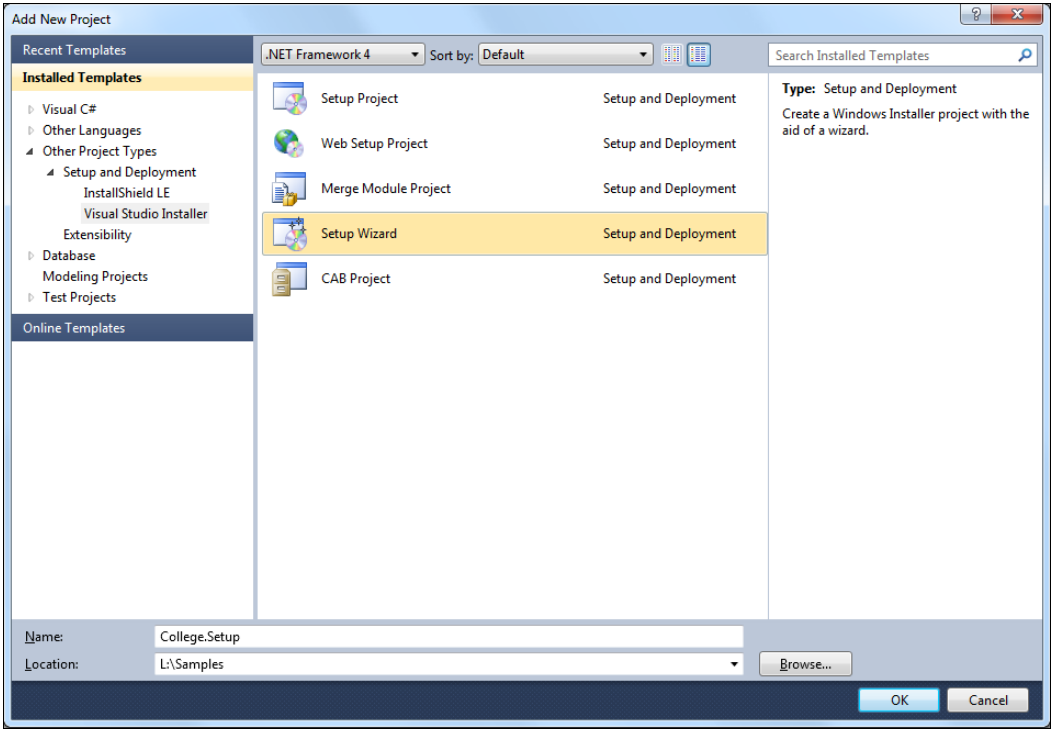


Рис. 23.1. Проекты развертывания

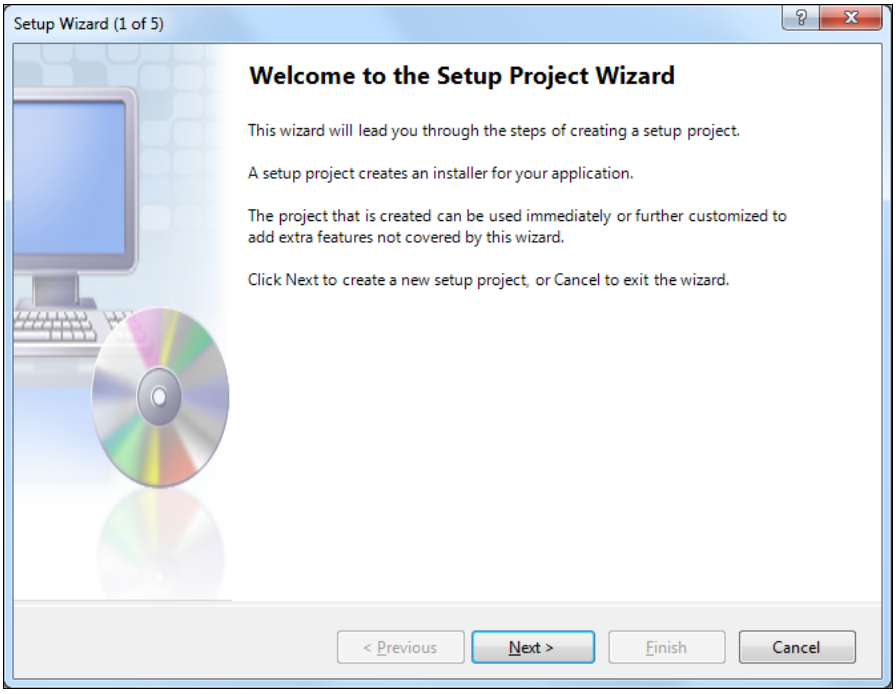


Рис. 23.2. Мастер создания установочного пакета

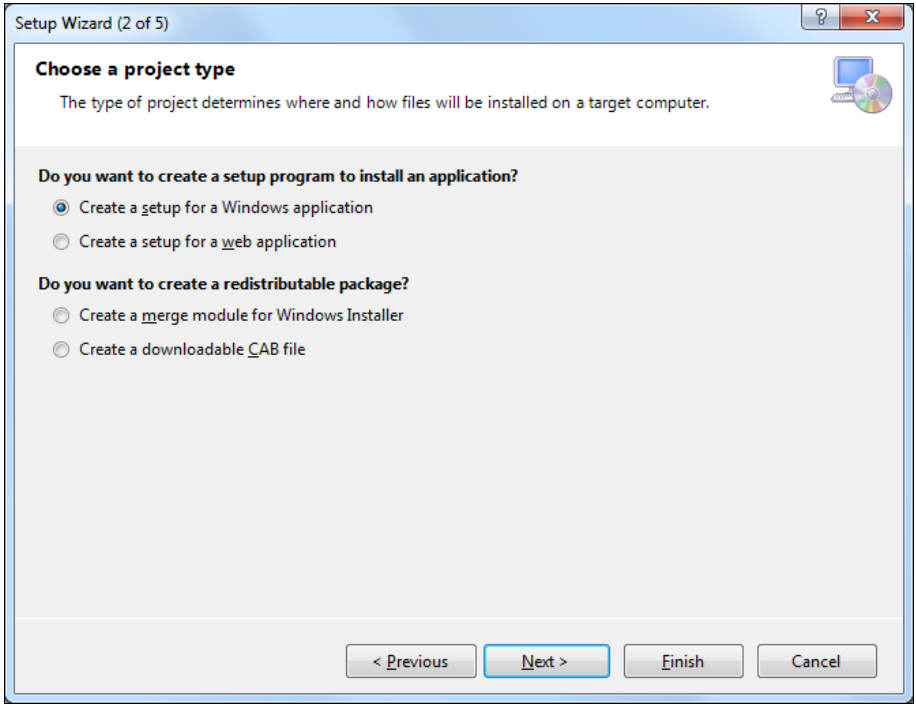


Рис. 23.3. Страница выбора типа проекта

Страница **Choose project outputs to include** (рис. 23.4) позволяет выбрать файлы решения для включения в проект установочной программы. Здесь для каждого проекта решения отображается 8 флажков — по одному для каждого типа содержимого, добавляемого к проекту. Если щелкнуть какой-либо значок, в поле **Description** появится его описание.

Чтобы включить в проект EXE- и DLL-файлы, установите флажок **Primary output from...**. При развертывании приложения для тестирования удобно также добавлять файлы с исходным текстом и символами для отладки, но в окончательной версии дистрибутива их быть не должно. Щелкните кнопку **Next**, чтобы перейти на следующую страницу.

Страница **Choose files to include** (рис. 23.5) позволяет добавить к проекту любые дополнительные файлы, например текстовые файлы Readme, HTML-страницы со справочной информацией, а также файлы поддержки, которые обычно не включают в установочный комплект приложения. Чтобы выбрать эти файлы, щелкните кнопку **Add** и найдите нужный файл. Закончив, щелкните кнопку **Next.**

Страница **Create Project** (рис. 23.6) отображает подробные сведения о параметрах, заданных в ходе работы с мастером. Щелкните кнопку **Finish**, чтобы создать проект установочной программы и добавить его к решению.

К готовому проекту можно в любое время добавить дополнительные файлы. Для этого щелкните его правой кнопкой мыши в окне **Solution Explorer** и выберите в контекстном меню **Add** подходящий элемент.

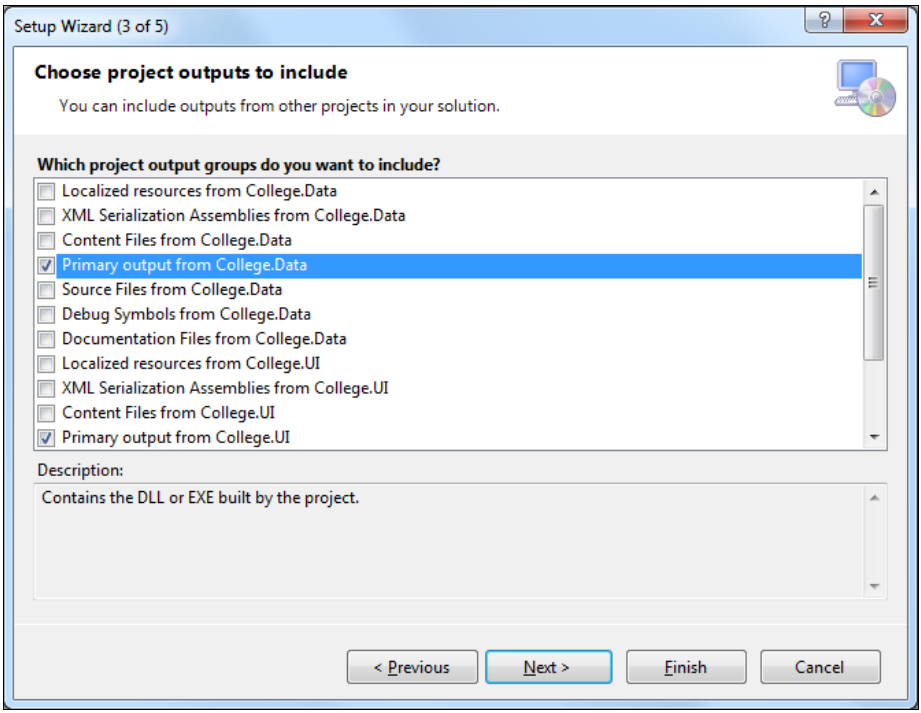


Рис. 23.4. Выбор файлов для включения в проект установочной программы

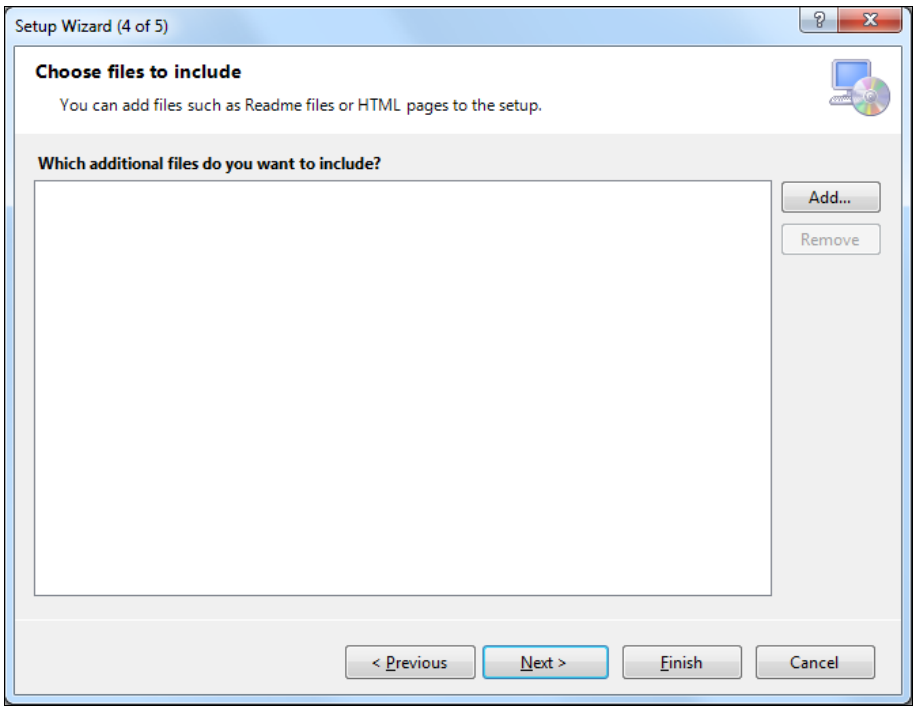


Рис. 23.5. Страница выбора дополнительных файлов

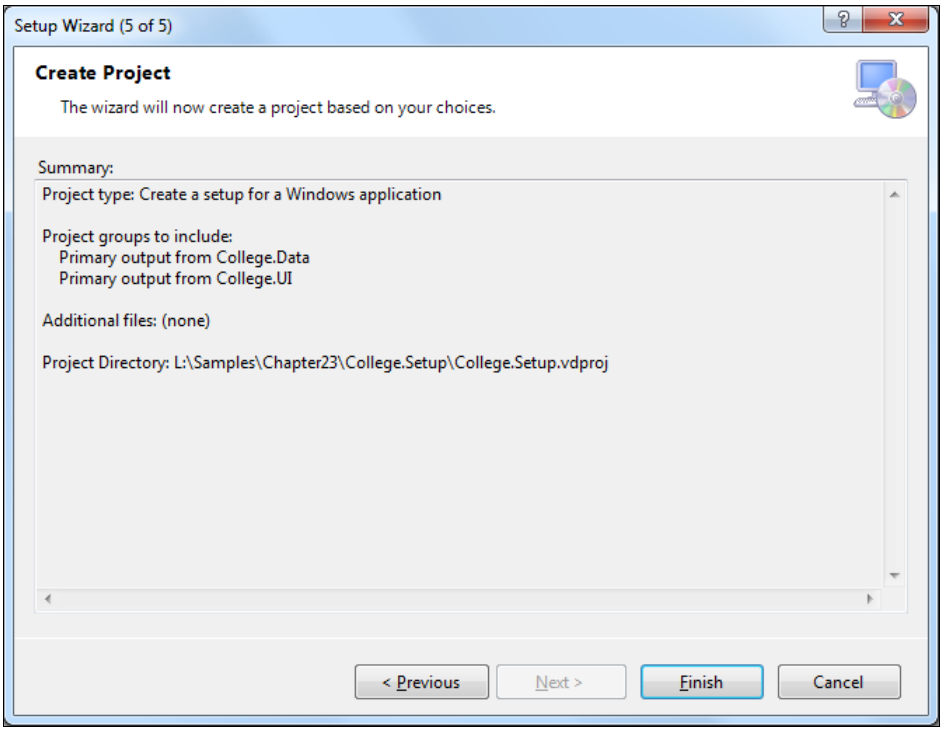


Рис. 23.6. Заключительная страница мастера

Новый проект будет добавлен к решению и отобразится в окне **Solution Explorer** (рис. 23.7).

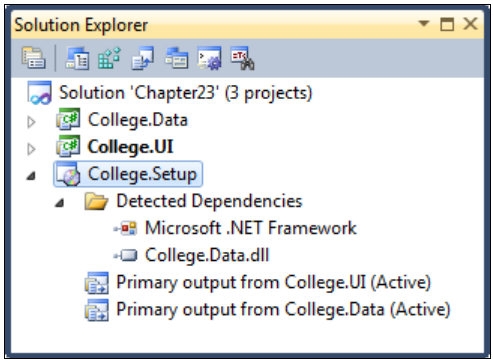


Рис. 23.7. Проект установки в **Solution Explorer**

В дополнение к выбранным вами файлам Visual Studio .NET автоматически найдет все возможные зависимости для нового проекта и добавит их в каталог Detected Dependencies. Учтите, что зависимости, не добавленные к проекту установочной программы явно, по умолчанию исключаются из сборки. Чтобы добавить найденную зависимость к проекту, щелкните правой кнопкой мыши соответствующий файл в окне **Solution Explorer** и в раскрывающемся списке сбросьте флажок **Exclude** для этого файла. Обратите внимание: включение установочных файлов

.NET Framework существенно увеличивает размер дистрибутива, поэтому добавлять их стоит только при отсутствии .NET Framework на целевом компьютере.

Параметры компоновки проекта

После создания инсталляционного проекта в Visual Studio 2010 необходимо настроить параметры компоновки проекта. При компиляции инсталляционного проекта получается файл с расширением msi (или несколько файлов, в зависимости от настроек, которые мы рассмотрим далее). В файле msi находятся все конфигурационные данные приложения, необходимые для установки на целевой компьютер, и само устанавливаемое приложение в сжатом виде. Этот файл будет использоваться программой Windows Installer.

Чтобы вызвать окно параметров компоновки проекта установочной программы, в **Solution Explorer** щелкните проект правой кнопкой мыши и выберите из контекстного меню пункт **Properties** — откроется диалоговое окно свойств инсталляционного проекта (рис. 23.8).

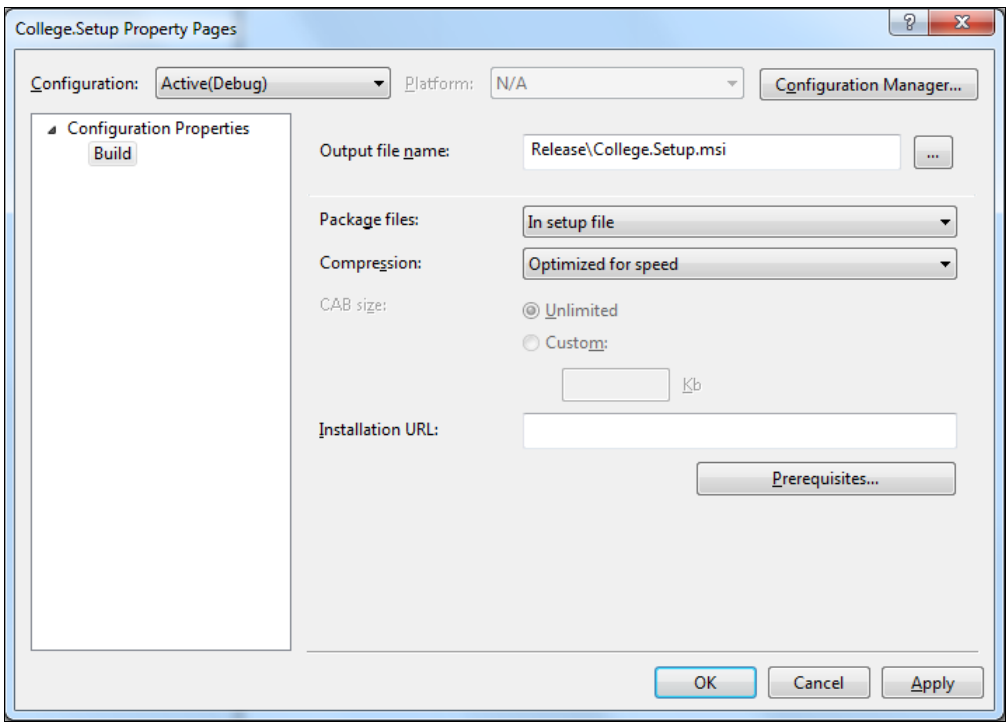


Рис. 23.8. Диалоговое окно свойств проекта

Параметры компоновки определяют результат компиляции проекта установочной программы, их набор зависит от способа развертывания приложения.

- **Output file name** — определяет имя, которое будет назначено выходному файлу, предназначенному для использования программой Windows Installer, а также каталог, в который он будет записан. По умолчанию его значение имеет вид

<configurafion>\<project name>, <extension>, где <configuration> — каталог в структуре каталогов проекта; <project name> (по умолчанию) — имя проекта, а <extension> — расширение msi для приложений Windows Installer или msm — для дополнительных модулей Windows Installer. Чтобы изменить путь выходного файла, щелкните кнопку **Browse** и укажите новый каталог.

- **Package files** — определяет способ упаковки выходных файлов решения в дистрибутиве приложения. По умолчанию все выходные файлы упаковываются в один файл вместе с установочной программой. Это обеспечивает высокую степень сжатия при минимальной сложности установки, поскольку все необходимое для развертывания приложения содержится в единственном файле. Однако в некоторых случаях требуется упаковать данные приложения в несколько CAB-файлов. При этом можно указать размер выходных CAB-файлов, что удобно, если размер выходных файлов ограничен. Например, если предполагается распространять приложение на диске, логично задать для CAB-файлов размер в 700 Мбайт. Полученные в результате CAB-файлы можно затем скопировать на диски. Наконец, параметр **Package files** позволяет скомпоновать выходные данные в виде набора несжатых файлов. При этом выходные файлы проекта просто копируются в тот же каталог, что и MSI-файл.
- **CAB size** — определяет размер CAB-файла. Если назначить ему значение **Unlimited**, создается единственный CAB-файл, в который упаковываются все выходные файлы проекта. Если же выбрано значение **Custom**, максимальный размер выходных CAB-файлов задает пользователь — это удобно, если предполагается распространять приложение на дисках или других сменных носителях.
- **Compression** — определяет схему сжатия для проекта установочной программы. Если сжатие не используется, этот параметр недоступен, в противном случае можно выбрать один из трех вариантов сжатия.
  - **None** — отключает сжатие файлов. Тогда установка приложения выполняется быстрее, однако при этом значительно увеличивается размер инсталлируемых файлов.
  - **Optimized for speed** — создает небольшое сжатие инсталлируемых файлов оптимальное для уменьшения времени установки приложения.
  - **Optimized for size** — задает максимальное сжатие, однако в этом случае потребуется больше времени на установку приложения.

Проект инсталляции содержит множество свойств, которые предоставляют пользователю сведения о приложении:

- `Title` — название приложения;
- `Description` — описание приложения;
- `Version` — содержит информацию о версии приложения;
- `AddRemoveProgramsIcon` — значок, который будет отображаться в окне **Add/Remove Programs** на клиентском компьютере;
- `Author` — сведения об авторе программы;

- `Localization` — сведения о локализации приложения;
- `Manufacturer` — информация о производителе приложения;
- `ManufacturerURL` — URL веб-сайта производителя;
- `ProductName` — имя продукта;
- `Subject` — название области применения приложения;
- `SupportPhone` — номер телефона службы технической поддержки приложения;
- `SupportURL` — адрес сайта службы технической поддержки приложения.

Заполнять все свойства проекта необязательно, если у вас, например, не предусмотрена техническоя поддержка приложения, можете свойства `SupportPhone` и `SupportURL` не заполнять.

Кроме этой группы, существуют свойства, определяющие некоторые особенности установки приложения:

- `DetectNewerInstall` — при установке этого свойства в `true` программа установки ищет на целевом компьютере более новую версию данного приложения. Если более новая версия уже имеется на целевом компьютере, установка приложения будет прекращена;
- `RemovePreviousVersion` — при установке этого свойства в `true` программа установки ищет на целевом компьютере предыдущие версии данного приложения и производит их удаление.

Также могут задаваться два свойства, которые являются глобальными идентификаторами устанавливаемого приложения:

- `ProductCode` — получает строку, указывающую уникальный идентификатор данного продукта;
- `UpgradeCode` — задает идентификатор, представляющий различные версии одного приложения, которое используется Windows Installer для проверки наличия установленных версий приложения в ходе установки.

Регистрация компонентов приложения

Проект установочной программы может включать шрифты, объекты COM или другие компоненты, требующие регистрации в системном реестре во время установки. Для этого необходимо установить в окне **Properties** свойство `Register` нужного файла.

Свойство `Register` может принимает следующие значения:

- `vsdrpDoNotRegister` — объект не требует регистрации;
- `vsdrpCOM` — объект регистрируется как объект COM;
- `vsdrpCOMRelativePath` — объект регистрируется как изолированный объект COM;
- `vsdrpCOMSellReg` — объект самостоятельно регистрируется как объект COM (для .NET-сборок не используется);
- `vsdrpFont` — объект будет зарегистрирован как компонент `Font`.

Для .NET-сборок регистрация не требуется, поэтому для них данное свойство устанавливают в значение `vsdrpDoNotRegister`.

Редакторы свойств установки

В Visual Studio имеется шесть редакторов свойств, позволяющие настраивать программу установки:

- **File System Editor** (редактор файловой системы) — определяет каталог в файловой системе целевого компьютера, в который будет произведена установка приложения;
- **Registry Editor** (редактор реестра) — создает ключи и заполняет их в реестре Windows целевого компьютера при инсталляции приложения;
- **File Types Editor** (редактор типов файлов) — связывает типы файлов с обрабатывающими их приложениями;
- **User Interface Editor** (редактор пользовательского интерфейса) — модифицирует пользовательский интерфейс установочной программы. Используется как при обычной установке, так и при установке с помощью сценария;
- **Custom Actions Editor** (редактор нестандартных действий) — задает нестандартные действия, выполняемые установочной программой;
- **Launch Conditions Editor** (редактор условий установки) — определяет необходимые условия для начала установки.

Далее эти редакторы обсуждаются более подробно.

File System Editor

С помощью редактора **File System Editor** можно выполнять следующие действия:

- создавать на целевом компьютере новые каталоги;
- записывать файлы в созданные каталоги;
- создавать ярлыки и добавлять их на рабочий стол и в меню **Programs**.

По умолчанию в редакторе файловой системы отображается стандартный набор папок для развертывания приложения (рис. 23.9).

На правой панели редактора отображается список выходных файлов проекта установочной программы, а на левой — структура каталогов целевого компьютера, исходно она включает три папки: каталог приложения, рабочий стол и меню **Programs**.

По умолчанию выходные файлы записываются в каталог приложения, однако, при желании, целевые каталоги можно изменить. Редактор файловой системы также позволяет создавать дополнительные каталоги. Чтобы создать такой каталог, надо щелкнуть на его левой панели правой кнопкой мыши и выбрать из контекстного меню команду **Add Special Folder** (рис. 23.10).

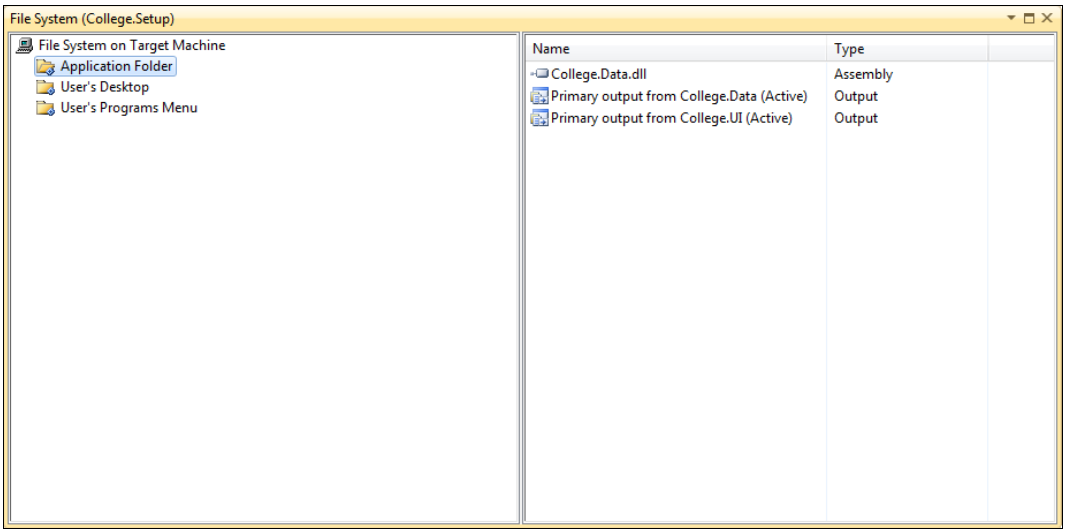


Рис. 23.9. Окно редактора файловой системы

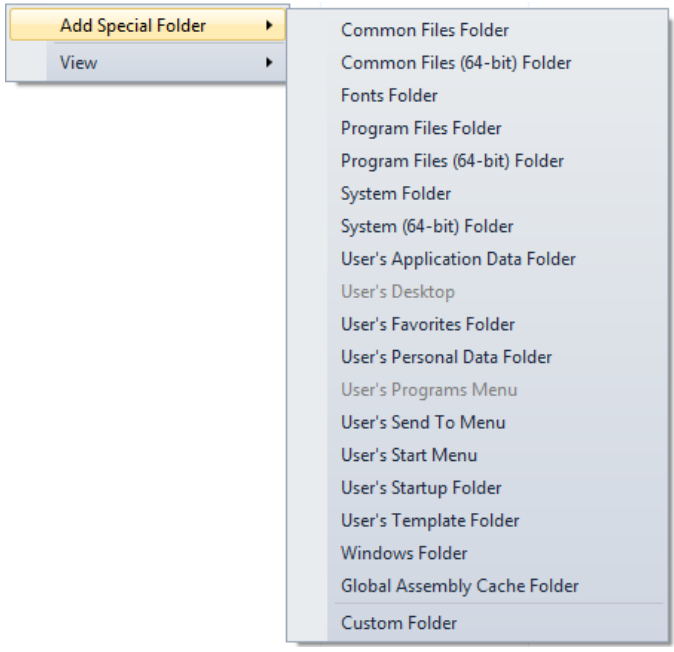


Рис. 23.10. Выбор типа дополнительных папок

Для создания ярлыка, который будет установлен на рабочем столе и в меню **Programs** целевого компьютера, в окне **Solution Explorer** щелкните правой кнопкой мыши файл, для которого требуется создать ярлык, и выберите в контекстном меню пункт **Choose Find In Editor**. При этом на правой панели окна редактора файловой системы появится каталог с выбранным файлом. Далее щелкните правой кнопкой мыши этот файл и выберите в контекстном меню пункт **Create Shortcut**.

Ярлык для этого файла создается и добавляется на панель. После этого перетащите ярлык с правой панели в требуемый каталог на левой панели редактора.

Registry Editor

Редактор реестра **Registry Editor** позволяет создавать ключи в системном реестре целевого компьютера. Окно редактора реестра разделено на две панели; на левой панели отображается реестр целевого компьютера, а на правой — значения текущего раздела.

Внешний вид редактора реестра представлен на рис. 23.11.

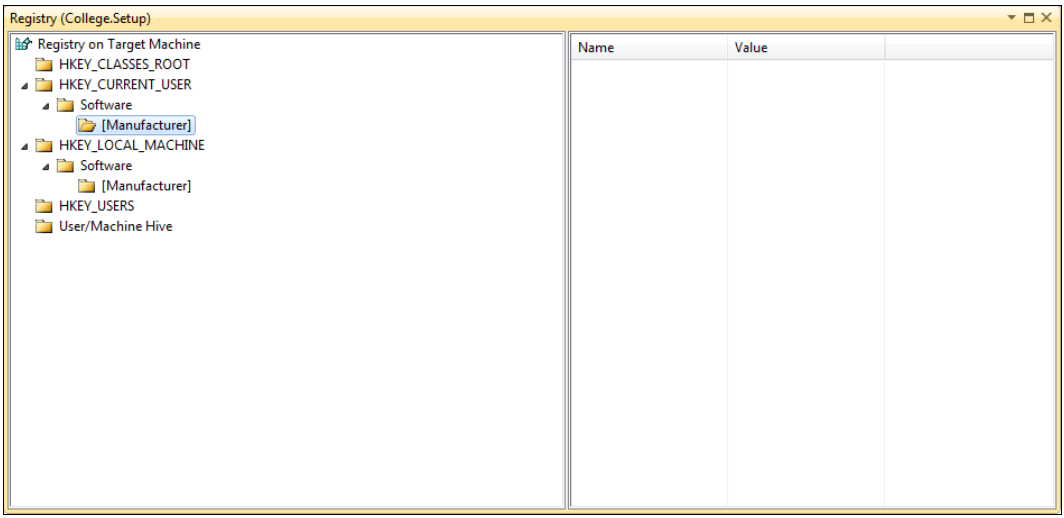


Рис. 23.11. Окно редактора реестра

Редактор реестра Visual Studio похож на редактор системного реестра Windows. Во время установки приложения ключи и значения ключей, созданные в редакторе реестра Visual Studio, записываются в системный реестр целевого компьютера.

При открытии редактора отображается стандартный набор главных ключей в виде дерева:

- `HKEY_CLASSES_ROOT`
- `HKEY_CURRENT_USER`
- `HKEY_LOCAL_MACHINE`
- `HKEY_USERS`

Ключи `HKEY_CURRENT_USER` и `HKEY_LOCAL_MACHINE` содержат дочерние ключи в `Software/[Manufacturer]`, где `[Manufacturer]` — информация о производителе, которая была определена в свойстве `Manufacturer` проекта.

Чтобы добавить новое значение в существующий раздел реестра, на левой панели щелкните правой кнопкой мыши раздел реестра, к которому вы хотите добавить новое значение, найдите в контекстном меню пункт **New** и выберите тип добавляе-

мого значения: **String Value**, **Environment**, **Binary Value** или **DWORD Value**. После этого в окне **Properties** определите свойство `Value` для нового ключа реестра.

File Types Editor

Редактор типов файлов **File Types Editor** позволяет связать приложение с файлами определенного типа, которые он будет открывать.

Внешний вид окна редактора представлен на рис. 23.12.

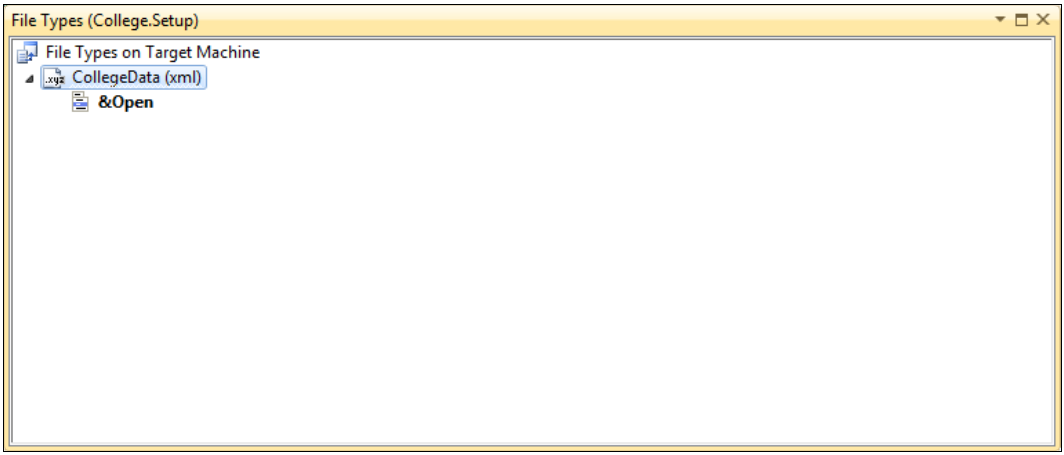


Рис. 23.12. Окно редактора типов файлов

Чтобы добавить ассоциацию, щелкните правой кнопкой мыши на значке **File Types on Target Machine** (рис. 23.12) и в контекстном меню выберите **Add File Type**. В окне редактора к узлу **File Types on Target Machine** добавится дочерний узел с именем по умолчанию **New Document Type**, для которого в окне **Properties** вы можете определить в свойстве `Name` его имя, а в свойстве `Extension` расширение для файлов, которое будет ассоциировано с приложением.

> ***Примечание***
>
> Точки вводить не нужно. Если требуется создать несколько расширений файлов, которые будут ассоциированы с устанавливаемой программой, их отделяют точкой с запятой, например: `ex1;ex2`.

> ***Обратите внимание***
>
> Расширение для файла может быть ассоциировано с одним приложением.

В окне редактора типов файлов в узел, отображающий тип ассоциированного с приложением документа, добавляется дочерний узел, отображающий команду по умолчанию **&Open** для контекстного меню. У связанной с типом файла команды контекстного меню есть три свойства:

- `Name` — текст, который отображается в контекстном меню при щелчке правой кнопкой мыши на файле с данным типом;
- `Verb` — описание действия, выполняемого над данным типом файлов;

- `Arguments` — аргументы командной строки, передаваемые приложению при запуске.

User Interface Editor

Редактор **User Interface Editor** позволяет модифицировать пользовательский интерфейс установочной программы приложения. Окно редактора пользовательского интерфейса разделено на две части:

- **Install** — для стандартной инсталляции;
- **Administrative Install** — для инсталляции с привилегиями администратора.

Внешний вид окна редактора пользовательского интерфейса представлен на рис. 23.13.

Рис. 23.13. Редактор пользовательского интерфейса

В редакторе пользовательского интерфейса можно предоставить администратору дополнительные возможности, недоступные при установке с правами пользователя. Например, администратор может задавать каталог для развертывания приложения, а пользователи могут использовать для развертывания только каталог по умолчанию.

Каждый раздел разбит на три подраздела. Эти подразделы представляют три основных стадии процесса установки приложения в редакторе пользовательского интерфейса.

- **Start** — программа Windows Installer собирает сведения о компьютере и пользователе, необходимые для развертывания приложения. Все диалоговые окна, требующие взаимодействия с пользователем, такие как окно выбора каталога для установки, помещают в **Start**. Также в этот раздел помещают проверку количества доступного места на диске целевого компьютера.

- **Progress** — выполняется копирование файлов приложения и отображается окно с индикатором прогресса.
- **End** — пользователю обычно отображается окно, сообщающее об окончании установки.

Порядок диалоговых окон можно поменять, перетаскивая их мышью. Можно также добавить дополнительные диалоговые окна. Внешний вид этих диалоговых окон можно изменять, редактируя соответствующие свойства в окне **Properties.**

Custom Actions Editor

Редактор **Custom Actions Editor** предназначен для создания нестандартных действий, выполняемых в процессе установки приложения.

Действия могут вызываться при появлении одного из четырех событий, генерируемых программой установки:

- `Install`
- `Commit`
- `Rollback`
- `Uninstall`

Событие `Install` генерируется после копирования файлов на целевой компьютер, но перед подтверждением окончания установки приложения. Событие `Commit` генерируется после подтверждения завершения установки на целевом компьютере. Событие `Rollback` генерируется при неудачной установке приложения. В нем реализуют логику отмены изменений, внесенных программой установки на клиентском компьютере. Событие `Uninstall` генерируется при удалении приложения с компьютера пользователя.

Редактор нестандартных действий, внешний вид которого представлен на рис. 23.14, позволяет привязать код к этим событиям.

Действия представляют собой внешние библиотечные исполняемые файлы или скрипты.

Чтобы создать нестандартное действие, в окне редактора щелкните правой кнопкой мыши требуемое событие и выберите из контекстного меню команду **Add Custom Action.** При этом откроется диалоговое окно **Select Item In Project**.

После добавления действия можно установить следующие свойства:

- `Condition` — задает условие — логический оператор, вычисляемый перед выполнением действия. Если оператор возвращает `true`, действие выполняется, в противном случае — нет;
- `CustomDataAction` — пользовательские данные, которые будут доступны для данного действия;
- `EntryPoint` — точка входа в библиотеку DLL, в которой содержится данное действие. Если действие содержится в исполняемом файле (EXE), это свойство не применяется;
- `InstallerClass` — булево значение, указывающее, реализовано ли данное действие в классе `Installer`. Если реализовано, его значение `true`, если нет — `false`;

Рис. 23.14. Редактор нестандартных действий

- `Name` — имя нестандартного действия. Передает параметры командной строки программе, реализующей нестандартное действие. Предназначается только для нестандартных действий, которые находятся в исполняемых файлах;
- `SourcePath` — содержит полный путь на компьютере разработчика к файлу программы, реализующей нестандартное действие.

Launch Conditions Editor

С помощью редактора **Launch Conditions Editor** определяют условия, которым целевой компьютер должен удовлетворять для начала установки. Например, можно

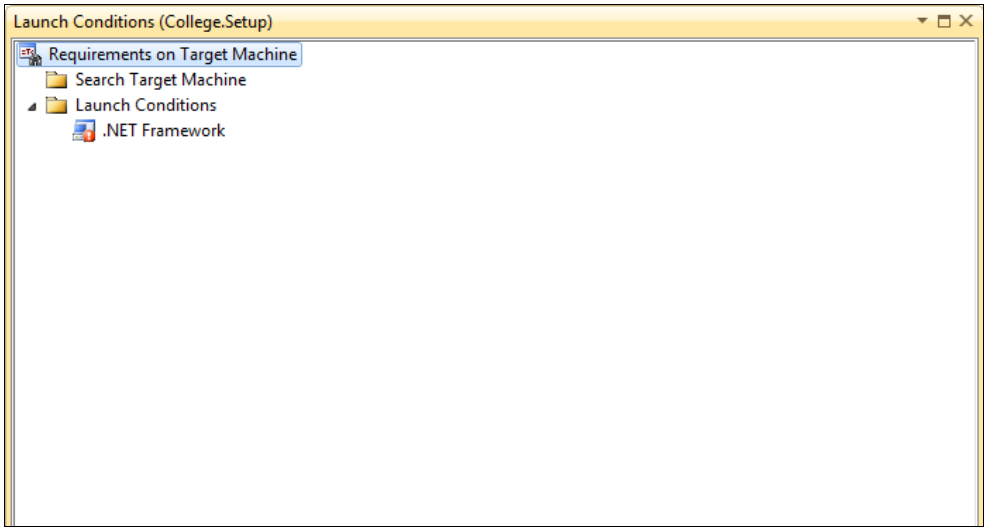


Рис. 23.15. Редактор условий установки

задать условие, чтобы на целевом компьютере была установлена конкретная версия Windows.

Кроме того, можно проверять наличие файлов библиотек DLL, записей реестра, версии Windows Installer и версии дистрибутива .NET Framework, установленной на целевом компьютере.

Внешний вид окна редактора **Launch Conditions Editor** представлен на рис. 23.15.

Сборка пакета установки

После добавления выходных файлов и определения всех необходимых свойств проекта установочной программы проект компонуют.

Сгенерированные при компоновке файлы дистрибутива записываются в каталог, заданный в свойствах проекта — это значение свойства **Output File Name**, откуда их можно скопировать на внешний носитель.

Установка приложения

Для начала инсталляции приложения надо запустить файл Setup.exe. При этом открывается MSI-файл, запускающий мастер установки (рис. 23.16).

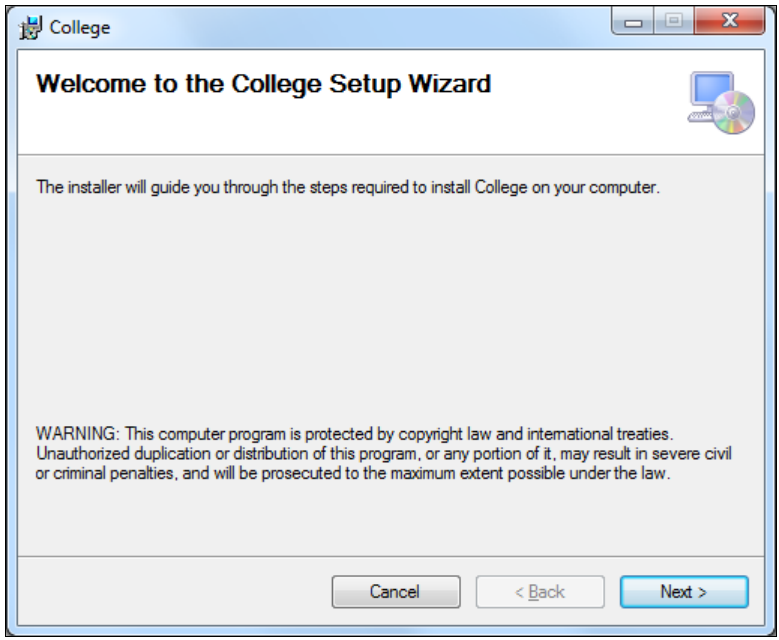


Рис. 23.16. Мастер установки приложения

Мастер установки автоматически устанавливает приложение в заданный каталог, как показано на рис. 23.17.

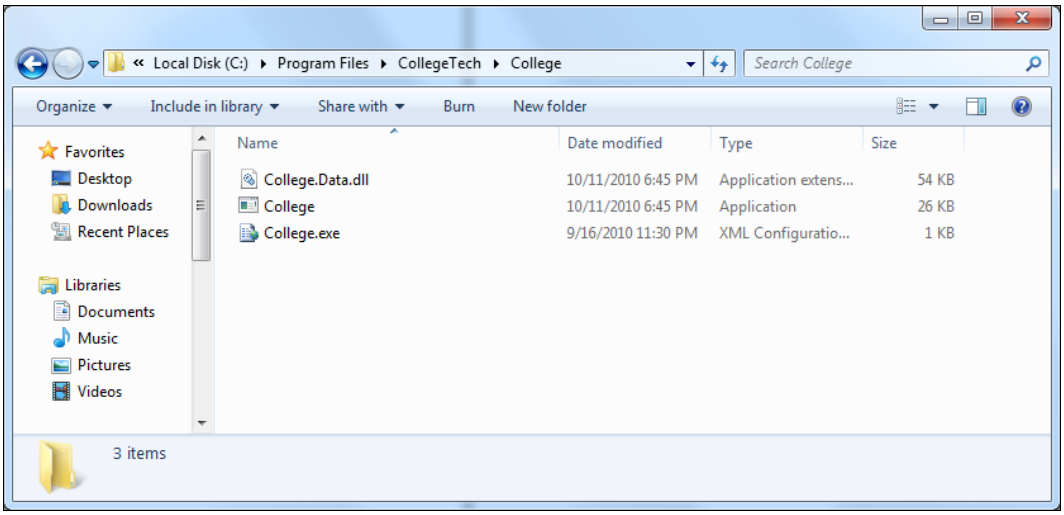


Рис. 23.17. Каталог с установленным приложением

После того как программа Windows Installer завершит развертывание, вы можете запустить установленное приложение, чтобы убедиться, что инсталляция была завершена успешно.

Резюме

В этой главе были показаны возможности среды Visual Studio 2010 для создания проектов развертывания с использованием технологии Windows Installer. Технология Windows Installer облегчает процесс установки и обновления программного обеспечения пользователям. Создаваемые пакеты инсталляции могут работать на любой версии Windows.

Для разработчика программного обеспечения технология Windows Installer предоставляет возможность гибко управлять развертыванием приложения, что позволяет достаточно легко поддерживать и распространять программное обеспечение и обновления для него.

ПРИЛОЖЕНИЕ

Описание компакт-диска и установка примеров

На компакт-диске в папке Samples расположены все примеры приложений, рассмотренных в книге. Структура Samples показана в табл. П1.

Таблица П1

Каталоги	Описание
Chapter03	Глава 3. Интегрированная среда разработки
Chapter04	Глава 4. Отладка приложений
Chapter05	Глава 5. Создание приложений Windows Forms
Chapter06	Глава 6. Проектирование баз данных в Visual Studio 2010
Chapter07	Глава 7. Технология доступа к данным ADO.NET
Chapter08	Глава 8. Работа с автономными данными в ADO.NET
Chapter09	Глава 9. LINQ
Chapter10	Глава 10. Entity Framework
Chapter11	Глава 11. Веб-формы ASP.NET
Chapter12	Глава 12. Стили и темы
Chapter13	Глава 13. Мастер-страницы и управление навигацией
Chapter14	Глава 14. ASP.NET AJAX
Chapter15	Глава 15. Библиотека jQuery
Chapter16	Глава 16. ASP.NET Dynamic Data
Chapter17	Глава 17. ASP.NET MVC
Chapter18	Глава 18. Windows Presentation Foundation
Chapter19	Глава 19. Silverlight
Chapter20	Глава 20. Windows Communication Foundation
Chapter21	Глава 21. Windows Workflow Foundation
Chapter22	Глава 22. Локализация приложений

Таблица П1 *(окончание)*

Каталоги	Описание
Chapter23	Глава 23. Развертывание приложений
Database	Содержит скрипты, файлы MDF и архивы баз данных, используемых в примерах

Кроме того, в каждой из этих папок находится локальный файл решения для данной главы.

Проекты не содержат исполняемых файлов. После копирования примеров на локальный компьютер необходимо провести компиляцию проектов, если вы собираетесь использовать скомпилированные сборки.

Для примеров из каталога Chapter14 требуется установленный на компьютере набор Ajax Control Toolkit, а для каталога Chapter19 — пакет Silverlight 4. Эти пакеты не входят в состав Visual Studio 2010, и при их отсутствии примеры из этих глав не будут компилироваться. Установка этих пакетов описана в соответствующих главах книги.

Начиная с главы 6 и далее, примеры используют базы данных, находящиеся на компакт-диске. Их также необходимо скопировать на локальный компьютер и сделать привязку к вашему экземпляру SQL Server 2008.

Файлы конфигурации приложений, представленных на компакт-диске, сконфигурированы для подключения к локальному экземпляру SQL Server по умолчанию. Если у вас установлен именованный экземпляр SQL Server, или вы используете SQL Server, находящийся на удаленном компьютере, необходимо будет изменить строки подключения к базе данных в файлах конфигурации приложений.

Предметный указатель

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Л

М

Н

О

П

Р

С

Т

У

Ф

X

Ш

Э

Я