Microsoft

Переход к Microsoft® Visual Studio® 2010

Патрис Пелланд,
Паскаль Паре, и Кен Хайнс

ОПУБЛИКОВАНО

Microsoft Press

A Division of Microsoft Corporation

One Microsoft Way

Redmond, Washington 98052-6399

Контрольный номер библиотеки конгресса (LCCN): 2010934433

Книги издательства Microsoft Press доступны по каналам оптовых и розничных продаж по всему миру. Для получения дополнительной информации о переводных изданиях обратитесь в местное отделение Корпорации Майкрософт или свяжитесь непосредственно с международным отделом Microsoft Press International по факсу (425) 936-7329. Посетите наш Веб-сайт www.microsoft.com/mspress. Комментарии присылайте по адресу mspinput@microsoft.com.

Обложка: Tom Draper Design

Body Part No. X17-13257

Содержание

Каково Ваше мнение об этой книге? Поделитесь им с нами!

Отзывы читателей представляют большой интерес и ценность для Майкрософт. На их основании мы можем постоянно улучшать качество наших книг и учебных ресурсов. Примите участие в экспресс-опросе онлайн по адресу microsoft com/learning/booksurvey

Введение

Выход каждой новой версии Microsoft Visual Studio является грандиозным событием для сообщества разработчиков, и данный выпуск, безусловно, не стал исключением, но сейчас можно почувствовать некоторую разницу в настроениях. Все участники Конференции Майкрософт для профессиональных разработчиков (PDC), состоявшейся в ноябре 2009 года в Лос-Анджелесе, получили последнюю бета-версию этой инкарнации Visual Studio и смогли испытать ее на практике еще до выхода окончательной версии. Разработчики увидели, насколько эта версия отличается от всех ее предшественников. Возможно, это звучит не ново, но Visual Studio 2010, по нашему мнению, являет собой большой шаг вперед и абсолютно меняет правила игры, поскольку была полностью спроектирована и разработана заново.

Публикации на форумах MSDN и во многих других популярных сообществах разработчиков также свидетельствуют о том, что многие профессиональные разработчики до сих пор используют предыдущие версии Visual Studio. В данной книге будет показано, как перейти на Visual Studio 2010, и приведены аргументы, почему сейчас самый подходящий момент для этого.

Для кого эта книга?

Данная книга ориентирована на профессиональных разработчиков, которые работают с предыдущими версиями Visual Studio и собираются переходить к Visual Studio 2010 Professional.

О чем эта книга?

Данная книга не является пособием для начинающих, справочником или книгой, посвященной какой-то одной технологии. Это книга, которая поможет профессиональным разработчикам перейти с предыдущих версий Visual Studio (начиная с версии 2003) на новую. Возможности Visual Studio 2010 рассматриваются в ней на примере типового приложения. Читателю будет представлено множество замечательных возможностей языков программирования и новые версии наиболее популярных технологий без заострения внимания на самих технологиях. Основной упор делается на работе с новыми инструментами и возможностями Visual Studio 2010. В этой книге вы не найдете подробного анализа новых Entity Framework или ASP.NET MVC 2, главным предметом рассмотрения здесь является Visual Studio 2010 и основания для перехода на Visual Studio 2010.

Как эта книга поможет перейти на Visual Studio 2010?

Для ответа на данный вопрос в этой книге используется практический подход. Новые возможности и характеристики Visual Studio 2010 рассматриваются с точки зрения ее пользователя, т.е. разработчика, использующего Visual Studio 2005, к примеру. Чтобы быть последовательными и рассмотреть все темы, мы создали и использовали реальное приложение, охватывающее многие аспекты продукта. На наш взгляд, это более эффективно, чем приводить множество разрозненных небольших примеров. Это приложение называется *Plan My Night*, остановимся на нем более подробно во Введении несколько ниже.

Чтобы сделать нашу книгу полезной максимально широкой аудитории разработчиков, было решено разбить ее на три части:

- Часть I предназначена для разработчиков, переходящих с Visual Studio 2003
- Часть II предназначена для разработчиков, переходящих с Visual Studio 2005
- Часть III предназначена для разработчиков, переходящих с Visual Studio 2008

Каждая из частей поможет разработчикам понять, как использовать Visual Studio 2010 для создания множества различных типов приложений и реализовать их творческий потенциал независимо от используемой на данный момент версии. Основное внимание в данной книге сосредоточено на Visual Studio, но также будут рассмотрены многие возможности языков программирования, которые являются дополнительным аргументом для перехода на новую версию.

Во всех частях используется один и тот же подход и включены такие главы:

- «Бизнес-логика и данные»
- «Проектирование внешнего вида и поведения»
- «Отладка приложения»

Например, Часть I «Переход от Microsoft Visual Studio 2003 к Visual Studio 2010» включает главу под названием «От 2003 к 2010: отладка приложения». Аналогично Часть II «Переход от Microsoft Visual Studio 2005 к Visual Studio 2010» включает главу «От 2005 к 2010: отладка приложения».

Проектирование внешнего вида и поведения

В этих главах сравнивается, как менялся и совершенствовался процесс создания пользовательского интерфейса в различных версиях Visual Studio. Основное внимание уделяется дизайнеру, редактору кода,

инструментам и различным элементам управления, а также сравнению методов валидации пользовательского интерфейса. Данные главы также касаются вопросов расширяемости приложений.

Бизнес-логика и данные

В данных главах обсуждаются вопросы структурирования приложения и демонстрируется эволюция инструментов и возможностей языков программирования для работы с данными. В них описываются разные слои приложения, рассматривается создание среднего уровня и кэширование данных в различных версиях, а также управление передачей данных в и из базы данных.

Отладка приложения

В данных главах демонстрируется развитие всех инструментов отладки и вспомогательных средств для разработчиков, а также сравниваются различные способы повышения производительности приложения. Обсуждается важная задача по модульному тестированию кода.

Развертывание приложения Plan My Night

Часть I, адресованная разработчикам, использующим Visual Studio 2003, также включает дополнительную главу, «От 2003 к 2010: развертывание приложения». В данной главе рассматриваются различные способы упаковки, развертывания и доставки приложения конечным пользователям. Также обсуждаются вопросы обновления и передачи пользователям новых версий приложения. По нашему мнению, в частях II и III, предназначенных для разработчиков в Visual Studio 2005 и Visual Studio 2008, соответственно, главы по развертыванию не требуется.

Что такое Plan My Night?

Назначение приложения Plan My Night (PMN) (Спланировать мой вечер) уже вполне понятно из его названия, но чтобы не осталось никаких сомнений приведем его аннотацию (блиц-резюме):

> *Приложение Plan My Night спроектировано и разработано как инструмент планирования и организации вечерних мероприятий. С его помощью пользователи могут создавать события, выполнять поиск мероприятий и мест проведения, собирать сведения о мероприятиях и местах их проведения и, наконец, делиться информацией или создавать собственные сообщения о мероприятиях.*

Но, как говорится, лучше один раз увидеть, чем сто раз услышать, поэтому посмотрим на пользовательский интерфейс Plan My Night, показанный на рис. I-1.

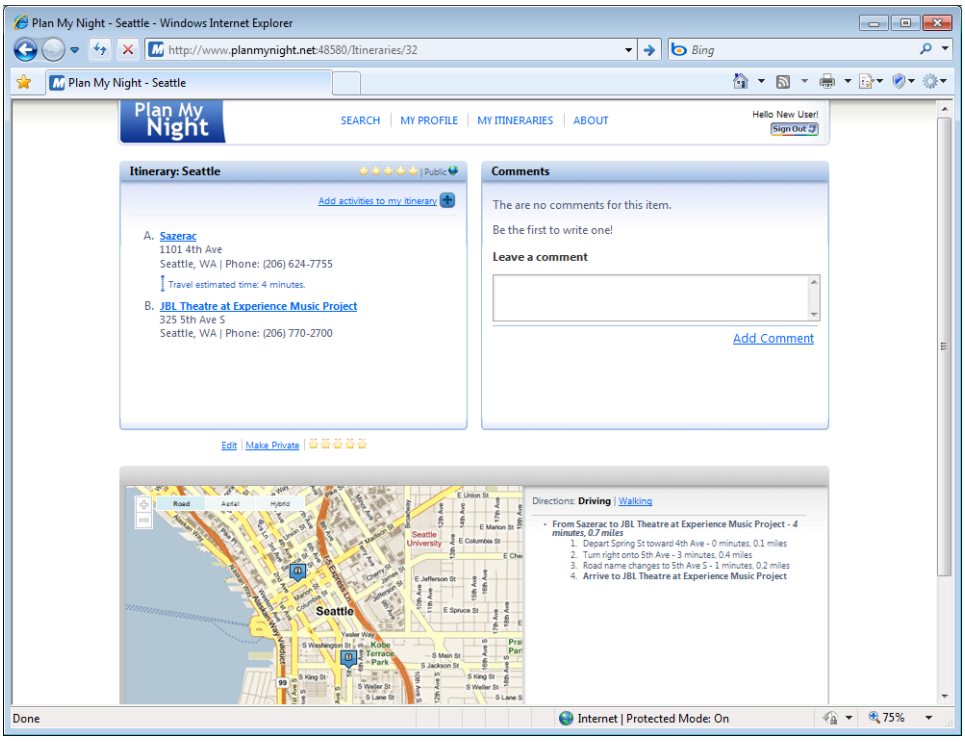


Рис. I-1 Пользовательский интерфейс приложения PMN

Plan My Night в Visual Studio 2010 создавалось с помощью ASP.NET MVC 2.0 с применением jQuery и Ajax для валидации и анимации пользовательского интерфейса. Расширение возможностей приложения реализовано посредством Managed Extensibility Framework (MEF) через создание подключаемых модулей для публикации на социальных сайтах, печати, отправки по электронной почте и т.д. Слой доступа к

данным реализован с помощью Entity Framework; кэширование в памяти данных, переданных и полученных из базы данных Microsoft SQL Server 2008, реализовано с применением Windows Server AppFabric (ранее известной под кодовым названием «Velocity»).

Несомненно, три иллюстрации лучше, чем одна, поэтому приводим на рис. I-2 диаграмму, показывающую различные части и их взаимодействие друг с другом, и на рис. I-3 различные технологии, используемые при создании Plan My Night.

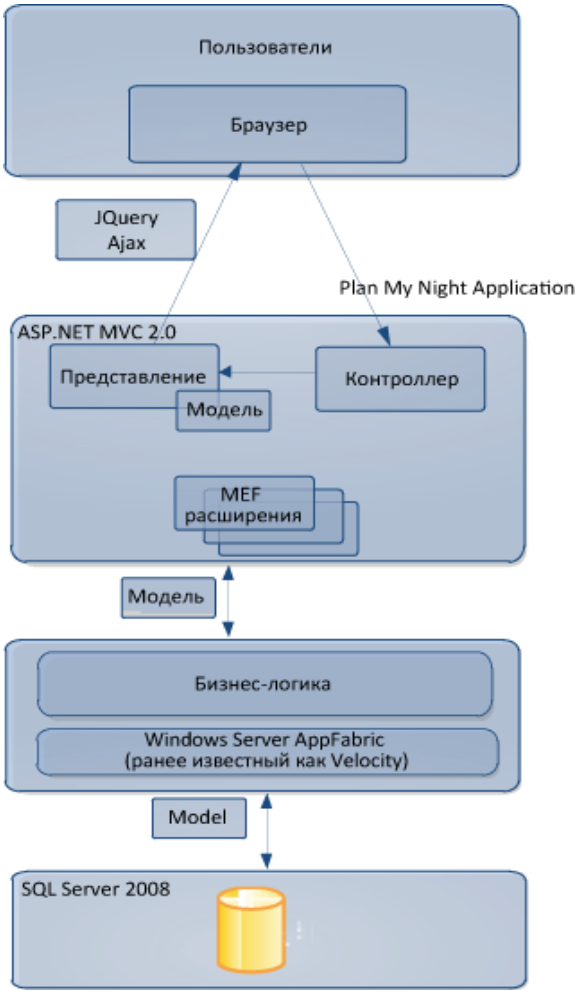


Рис. I-2 Компоненты приложения Plan My Night и их взаимодействие

Рис. I-3 PMN 1.0 и различные технологии, используемые при его создании

Зачем переходить на Visual Studio 2010?

Существует множество причин для перехода на Visual Studio 2010 Professional. Прежде чем углубиться в их подробное рассмотрение, приведем их краткий список (без какой-либо приоритезации):

- Встроенные инструменты для создания приложений для Windows 7, включая компоненты для реализации мультисенсорного ввода и пользовательского интерфейса «лента».
- Новый редактор с богатой функциональностью, встроенной поддержкой Windows Presentation Foundation (WPF) и широкими возможностями настройки соответственно вашему стилю работы (рис. I-4).
- Поддержка одновременной работы с несколькими мониторами.
- Новая функция быстрого поиска (Quick Search), которая помогает находить соответствующие результаты уже по первым нескольким буквам любого метода, класса или свойства.
- Замечательная поддержка разработки и развертывания приложений Microsoft Office 2010, SharePoint 2010 и Windows Azure.
- Поддержка разработки приложений с многопоточной обработкой с возможностью организации параллельных потоков в приложениях и новый специализированный отладчик, позволяющий отслеживать задачи и потоки.
- Улучшенные возможности инфраструктуры ASP.NET AJAX, базовая поддержка IntelliSense для JavaScript, введение в Visual Studio 2010 поддержки jQuery, библиотеки с открытым исходным кодом для взаимодействия с элементами DOM.
- Поддержка различных версий и платформ. Этой замечательной возможности посвящена публикация в блоге Скотта Гатри (Scott Guthrie) (*http://weblogs.asp.net/scottgu/archive/2009/08/27/multi-targeting-support-vs-2010-and-net-4-series.aspx*).
- Возможности разработки приложений на WPF и Silverlight с улучшенной поддержкой функции «drag-and-drop» и привязкой данных. Сюда относятся замечательные новые возможности дизайнеров, обеспечивающие более высокую точность при формировании визуального представления элементов управления и позволяющие выявлять ошибки до запуска приложения (что является громадным улучшением по сравнению с предыдущими версиями Visual Studio). Новые инструменты WPF и Silverlight позволяют просматривать визуальное дерево элементов и объекты насыщенных приложений на WPF и Silverlight.
- Замечательная поддержка Team Foundation Server (TFS) 2010 (и предыдущих версий[1]) с использованием Team Explorer. Позволяет работать с данными и отчетами, автоматически собираемыми и формируемыми Visual Studio 2010, отслеживать и анализировать состояние проектов на основании интегрированных отчетов, а также синхронизировать требующие исправления дефекты и задачи соответственно текущему состоянию.
- Интегрированная поддержка разработки через тестирование. Автоматическое формирование тестовых заглушек и богатая инфраструктура модульного тестирования значительно упростят для разработчиков задачи по созданию и выполнению модульных тестов. Visual Studio 2010 обладает замечательными возможностями расширения, благодаря чему прямо в рамках Visual Studio 2010 могут использоваться популярные инфраструктуры модульного тестирования с открытым исходным кодом или сторонних производителей.

[1] Начиная с TFS 2008 (*прим. научного редактора*)

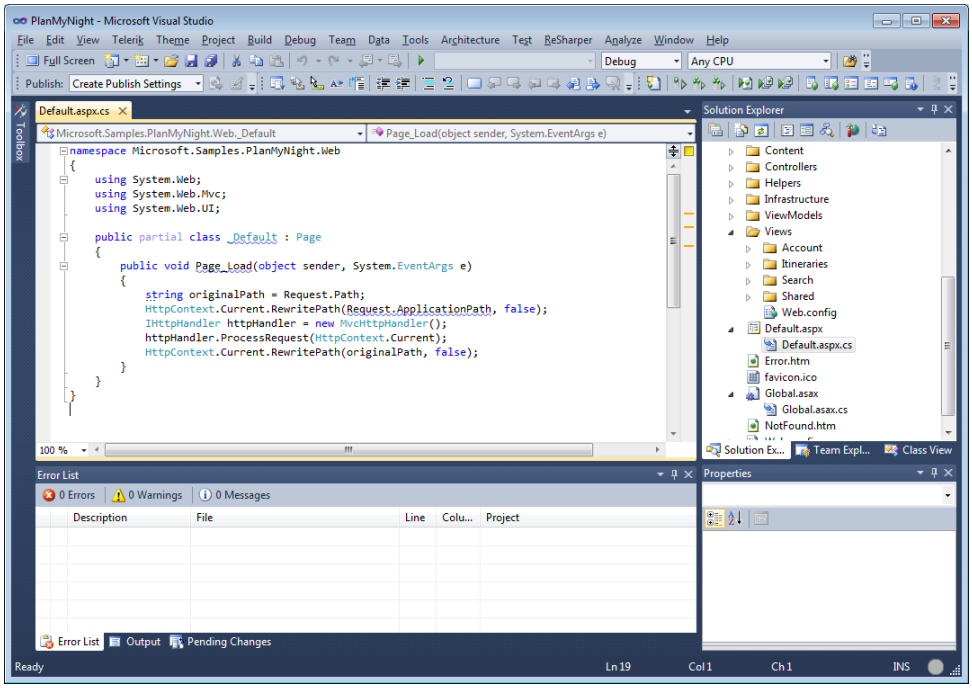


Рис. I-4 Новый редактор кода WPF в Visual Studio

Это всего лишь краткий список новых возможностей Visual Studio 2010 Professional. Некоторые из них будут представлены «из первых рук» в данной книге. Полный список всех новых возможностей можно найти в следующих материалах: *http://msdn.microsoft.com/en-us/library/dd547188(VS.100).aspx* и *http://msdn.microsoft.com/en-us/library/bb386063(VS.100).aspx*.

Но самым важным основанием для перехода на новую версию для многих разработчиков и организаций является то, что с ней вы не будете тратить время на интерпретацию кода и сможете сосредоточиться на решении реальных проблем. Visual Studio 2010 предоставляет новые мощные дизайнеры и инструменты разработки, которые позволяют сократить объем необходимого кода, тратить меньше времени на его написание, но создавать при этом более качественный код.

Опечатки и поддержка

Мы сделали все возможное, чтобы обеспечить максимальную точность данной книги и сопроводительных материалов к ней. Обо всех выявленных ошибках, пожалуйста, сообщайте на сайт Microsoft Press на портале Oreilly.com:

1. Перейдите по адресу http://microsoftpress.oreilly.com.
2. В строке поиска введите номер ISBN или название книги.
3. Выберите необходимую книгу среди результатов поиска.
4. На странице каталога для вашей книги под изображением обложки можно увидеть список ссылок.
5. Щелкните ссылку View/Submit Errata (Просмотреть/Передать сообщение об опечатке).

На странице каталога для книги можно также найти дополнительные сведения и услуги. Для получения дополнительной поддержки обращайтесь в отдел поддержки Microsoft Press Book Support по адресу *mspinput@microsoft.com*. Обратите внимание, что техническая поддержка программного обеспечения Майкрософт по этому адресу не предоставляется.

Часть I

Переход от Microsoft Visual Studio 2003 к Visual Studio 2010

Авторы Патрис Пелланд, Кен Хайнс и Паскаль Паре

В данной части:

Глава 1

От 2003 к 2010: бизнес-логика и данные

В данной главе рассматривается

- Применение Entity Framework (EF) для создания слоя доступа к данным с использованием существующей базы данных или модель-ориентированного подхода
- Создание типов сущностей в дизайнере Entity Data Model (Модель сущность-данные) с использованием POCO шаблонов в ADO.NET Entity Framework
- Получение данных от Веб-сервисов
- Кэширование данных с использованием Windows Server AppFabric (ранее известной под кодовым названием «Velocity»)

Архитектура приложения

С помощью приложения PlanMyNight (PMN) пользователь может составлять программы своих мероприятий и делиться ими с остальными. Данные хранятся в базе данных Microsoft SQL Server. Программы составляются на основании обращений к Веб-сервисам Bing Maps.

Рассмотрим исходную блок-схему модели данных приложения (рис. 1-1).

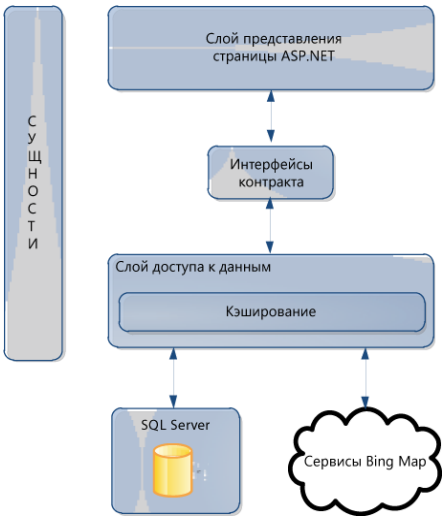


Рис. 1-1 Архитектурная диаграмма приложения PlanMyNight

Описание контрактов и классов сущностей, свободных от каких-либо ограничений, налагаемых методом хранения, позволяет объединять их в сборки, не зависящие от метода хранения. Такой подход обеспечивает четкое разделение слоев представления и доступа к данным.

Опишем интерфейсы контрактов основных компонентов приложения PMN:

- *IItinerariesRepository* (Хранилище маршрутов) – интерфейс нашего хранилища данных (базы данных Microsoft SQL Server).
- *IActivitiesRepository* (Хранилище действий) позволяет выполнять поиск мероприятий (используя Веб-сервисы Bing Maps).
- *ICachingProvider* (Поставщик кэширования) обеспечивает интерфейс кэширования данных (кэширование ASP.NET или кэширование AppFabric Windows Server).

Примечание Это неполный список контрактов, реализованных в приложении PMN.

PMN сохраняет программы мероприятий пользователя в базе данных SQL. Остальные пользователи смогут комментировать и рейтинговать программы друг друга. На рис. 1-2 представлены таблицы, используемые приложением PMN.

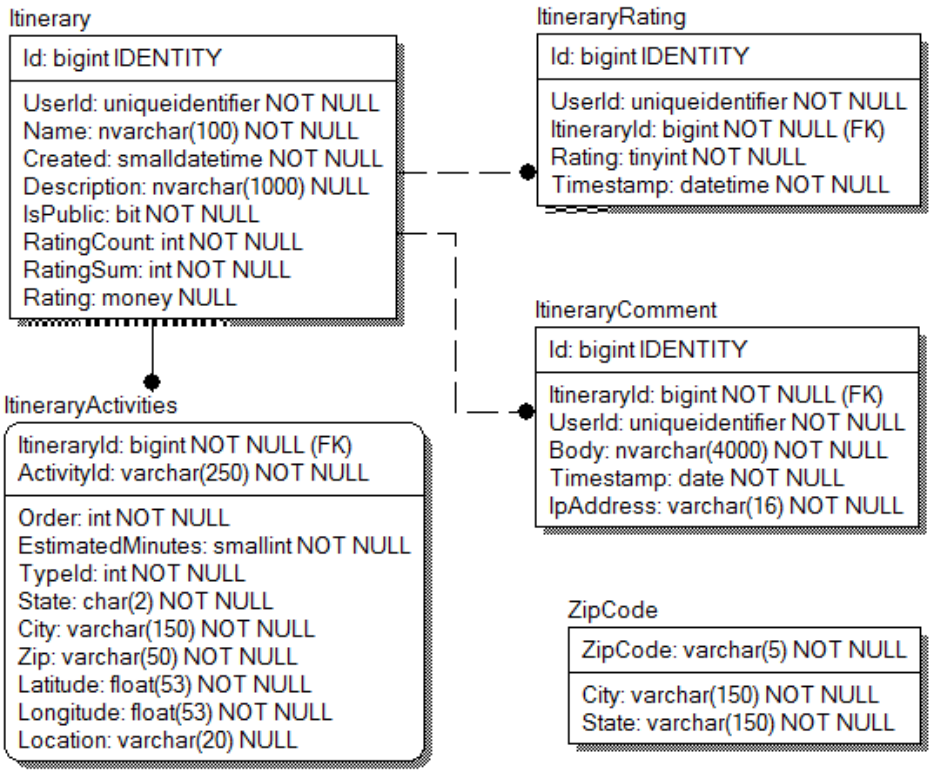


Рис. 1-2 Схема базы данных PlanMyNight

> **Важно** В приложении PlanMyNight безопасное хранение учетных данных пользователей обеспечивается посредством возможностей членства ASP.NET. Таблицы для хранения данных о пользователе на рис. 1-2 не показаны. Более подробную информацию об этих возможностях можно найти на сайте MSDN в разделе ASP.NET 4 – Introduction to Membership (ASP.NET 4 – введение в членство) по адресу *http://msdn.microsoft.com/en-us/library/yh26yfzy(VS.100).aspx*.

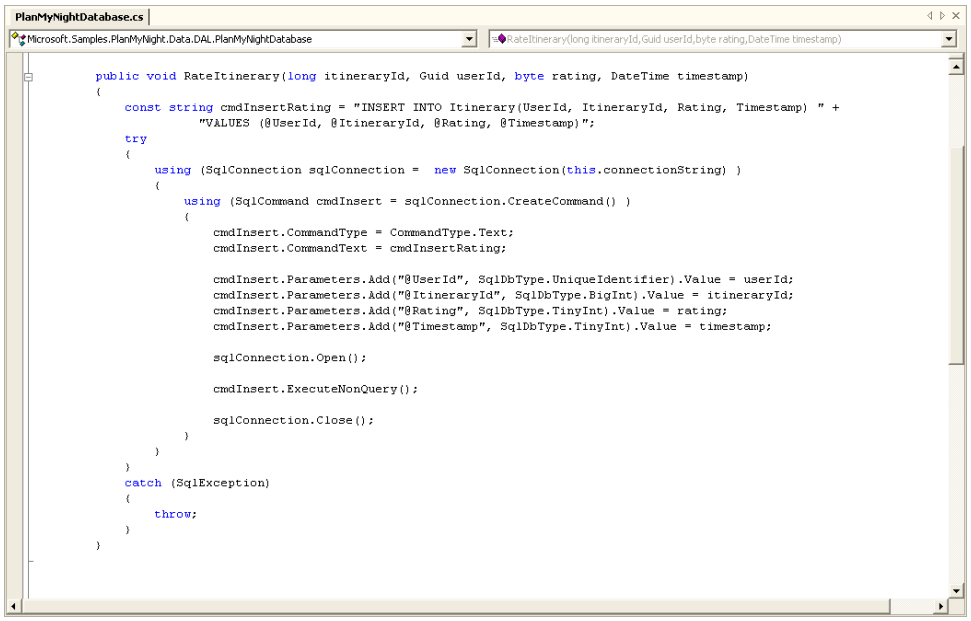
> **Примечание** Таблица ZipCode (Почтовый индекс) используется как хранилище справочных ресурсов и предоставляет список доступных почтовых индексов и городов. Благодаря этому обеспечивается функциональность автозаполнения при вводе поискового запроса пользователем в приложении.

Данные Plan My Night в Microsoft Visual Studio 2003

Visual Studio 2003 предоставляет все необходимые инструменты для создания приложения, поэтому написание PlanMyNight в ней не составило бы никакого труда. Однако некоторые применяемые тогда технологии требовали написания намного большего объема кода для достижения тех же целей.

В Visual Studio 2003 необходимый слой доступа к данным можно было создать, используя *DataSet* (Множество данных) или *DataReader* (Средство чтения данных) ADO.NET (рис. 1-3). Такое решение предлагает большую гибкость, поскольку обеспечивает полный контроль над доступом к базе данных. С другой стороны, оно имеет некоторые недостатки:

- Необходимо знать синтаксис SQL.
- Все запросы специализированы. Любое изменение в требованиях или в таблицах приводит к необходимости обновления всех соответствующих запросов.
- Приходится сопоставлять свойства классов сущностей, используя имя столбца, что довольно утомительно и чревато многочисленными ошибками.
- Приходится самостоятельно управлять связями между таблицами.




```
public void RateItinerary(long itineraryId, Guid userId, byte rating, DateTime timestamp)
{
    const string cmdInsertRating = "INSERT INTO Itinerary(UserId, ItineraryId, Rating, Timestamp) " +
            "VALUES (@UserId, @ItineraryId, @Rating, @Timestamp)";
    try
    {
        using (SqlConnection sqlConnection =  new SqlConnection(this.connectionString) )
        {
            using (SqlCommand cmdInsert = sqlConnection.CreateCommand() )
            {
                cmdInsert.CommandType = CommandType.Text;
                cmdInsert.CommandText = cmdInsertRating;

                cmdInsert.Parameters.Add("@UserId", SqlDbType.UniqueIdentifier).Value = userId;
                cmdInsert.Parameters.Add("@ItineraryId", SqlDbType.BigInt).Value = itineraryId;
                cmdInsert.Parameters.Add("@Rating", SqlDbType.TinyInt).Value = rating;
                cmdInsert.Parameters.Add("@Timestamp", SqlDbType.TinyInt).Value = timestamp;

                sqlConnection.Open();

                cmdInsert.ExecuteNonQuery();

                sqlConnection.Close();
            }
        }
    }
    catch (SqlException)
    {
        throw;
    }
}
```

Рис. 1-3 Запрос Insert ADO.NET

В следующих разделах данной главы будут рассмотрены некоторые новые возможности, предоставляемые Visual Studio 2003. Благодаря им создание слоя доступа к данным приложения PMN связано с меньшим объемом написания кода, предоставляется больший контроль над автоматически формируемым кодом, обслуживать и расширять его становится намного проще.

Работа с данными в Visual Studio 2003 с использованием Entity Framework

ADO.NET Entity Framework (EF) позволяет без труда создавать слой доступа к данным приложения через абстрагирование работы с данными от базы данных и создание модели, приближенной к бизнес-требованиям приложения. EF была существенно доработана и улучшена в четвертой версии .NET Framework.

> *Дополнительные сведения Множество ресурсов, посвященных ADO.NET Entity Framework в .NET 4, предлагает Центр решений по работе с данными (Data Developer Center) на сайте MSDN (http://msdn.microsoft.com/en-us/data/aa937723.aspx).*

Возьмем приложение PlanMyNight в качестве примера использования некоторых возможностей EF. В следующих двух разделах демонстрируются два разных подхода к автоматическому формированию модели данных PMN. В первом случае EF создаст Entity Data Model (EDM) из существующей базы данных. Во втором – используется модель-ориентированный подход (Model First), при котором сначала в дизайнере EF создаются сущности, а затем для создания базы данных, в которой может храниться EDM, автоматически формируются сценарии на языке описания данных (Data Definition Language, DDL).

EF: импорт существующей базы данных

Мы будем работать с уже существующим решением, в котором описаны основные проекты приложения PMN. Если сопроводительные материалы данной книги установлены в каталог по умолчанию, интересующее нас решение находится по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 1\Code\ExistingDatabase. Щелкните двойным щелчком файл PlanMyNight.sln.

Это решение включает все проекты, которые перечислены в следующем списке (рис. 1-4):

- PlanMyNight.Data: слой доступа к данным приложения
- PlanMyNight.Contracts: сущности и контракты
- PlanMyNight.Bing: сервисы Bing Map
- PlanMyNight.Web: слой представления
- PlanMyNight.AppFabricCaching: кэширование AppFabric

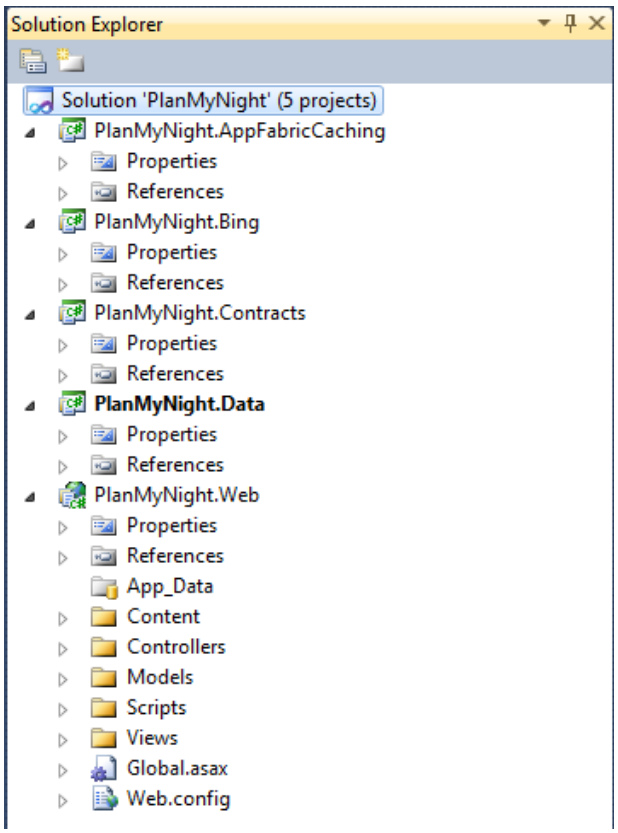


Рис. 1-4 Решение PlanMyNight

EF позволяет без труда импортировать существующую базу данных. Рассмотрим этот процесс поэтапно.

Первый шаг – добавление EDM в проект PlanMyNight.Data. Щелкаем правой кнопкой мыши проект PlanMyNight.Data, выбираем Add (Добавить) и New Item (Новый элемент). Выбираем элемент ADO.NET Entity Data Model и меняем его имя на **PlanMyNight.edmx**, как показано на рис. 1-5.

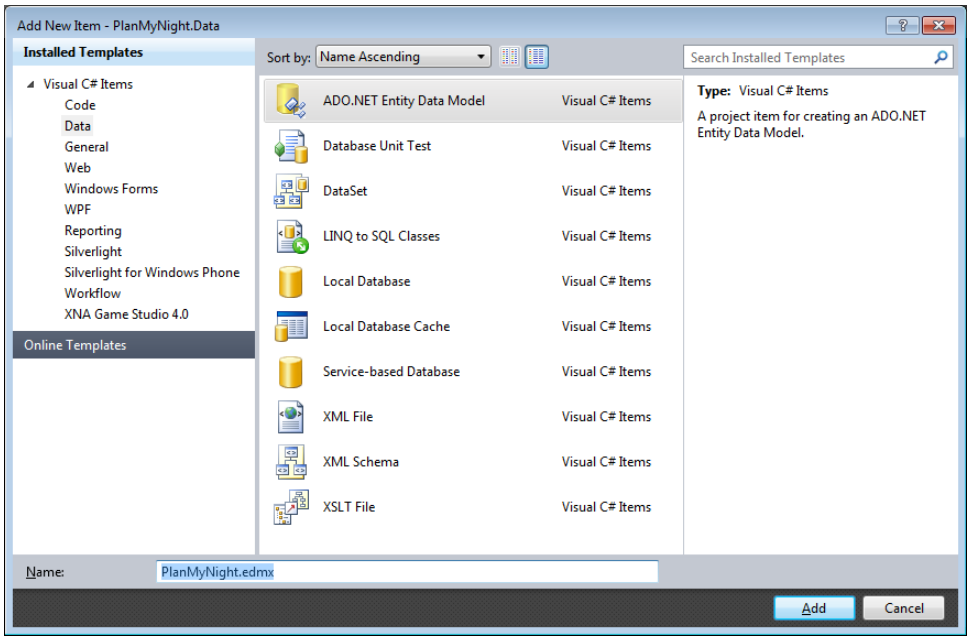


Рис. 1-5 Диалоговое окно Add New Item с выбранной ADO.NET Entity Data Model

Первое диалоговое окно мастера Entity Data Model Wizard позволяется выбрать содержимое модели. Мы собираемся создать модель из существующей базы данных. Выбираем Generate From Database (Создать из базы данных) и щелкаем Next (Далее).

Необходимо подключиться к существующему файлу базы данных. Щелкаем New Connection (Создать подключение). В открывшемся диалоговом окне Choose Data Source (Выбрать источник данных) выбираем Microsoft SQL Server Database File (Файл базы данных Microsoft SQL Server). Выбираем файл

%userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 1\ExistingDatabase\PlanMyNight.Web\App_Data\PlanMyNight.mdf (рис. 1-6).

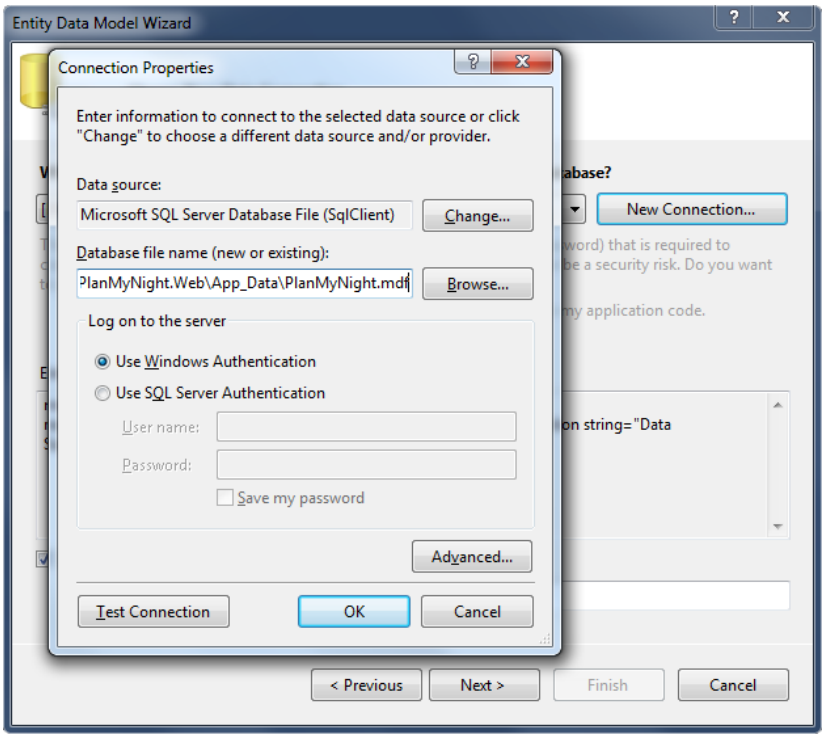


Рис. 1-6 Мастер EDM, диалоговое окно для создания подключения к базе данных

Все остальные поля формы пока оставляем без изменений и щелкаем Next.

> **Примечание** На экран будет выведено предупреждение о том, что локальный файл данных не входит в состав текущего проекта. Поскольку мы не хотим копировать файл базы данных в текущий проект, щелкните No (Нет), чтобы закрыть этот диалог.

В диалоговом окне Choose Your Database Objects (Выберите объекты своей базы данных) выберите таблицы Itinerary (План мероприятий), ItineraryActivities (Мероприятия), ItineraryComment (Комментарий к плану мероприятий), ItineraryRating (Рейтинг плана мероприятий) и ZipCode (Почтовый индекс), а также представление UserProfile (Профиль пользователя). Выберите хранимую процедуру *RetrieveItinerariesWithinArea* (Извлечение планов мероприятий в определенном районе). В поле Model Namespace (Пространство имен модели) зададим значение **Entities** (Сущности), как показано на рис. 1-7.

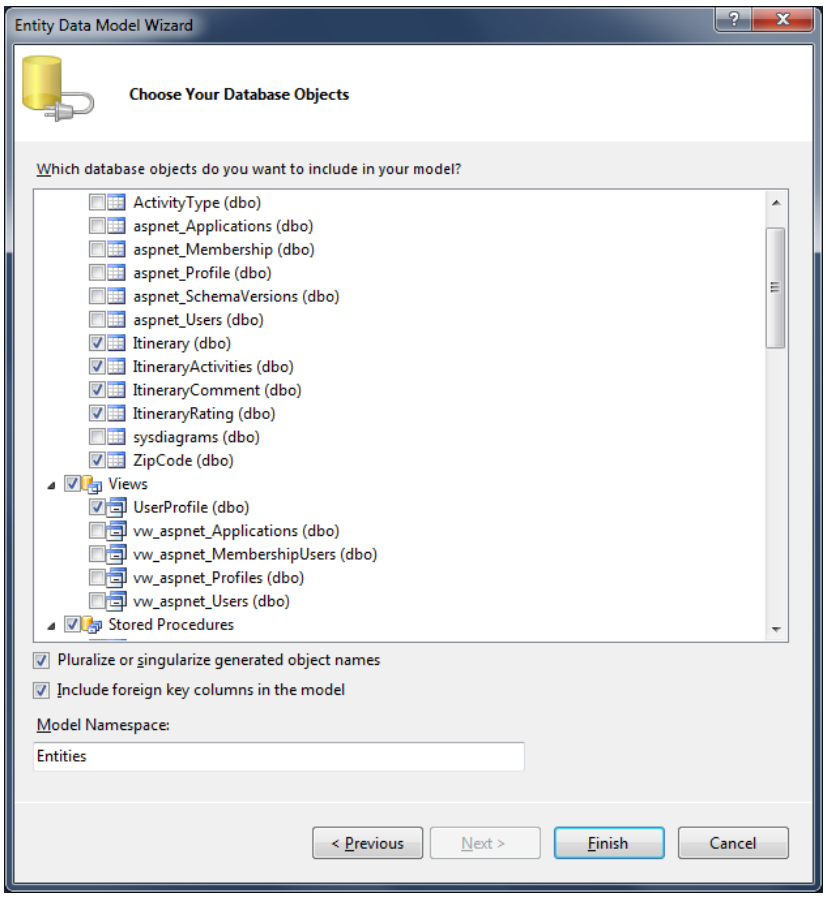


Рис. 1-7 Мастер EDM: страница Choose Your Database Objects

Щелкаем Finish (Готово), чтобы сформировать EDM.

Корректировка сформированной модели данных

Теперь мы имеем модель, представляющую множество сущностей, соответствующее используемой базе данных. Мастер сформировал все навигационные свойства, ассоциированные с внешними ключами базы данных.

Приложению PMN необходимо только навигационное свойство *ItineraryActivities*, связанное с таблицей Itinerary, поэтому все остальные навигационные свойства можно смело удалить. Также придется переименовать навигационное свойство *ItineraryActivities* в **Activities**. Обновленная модель представлена на рис. 1-8.

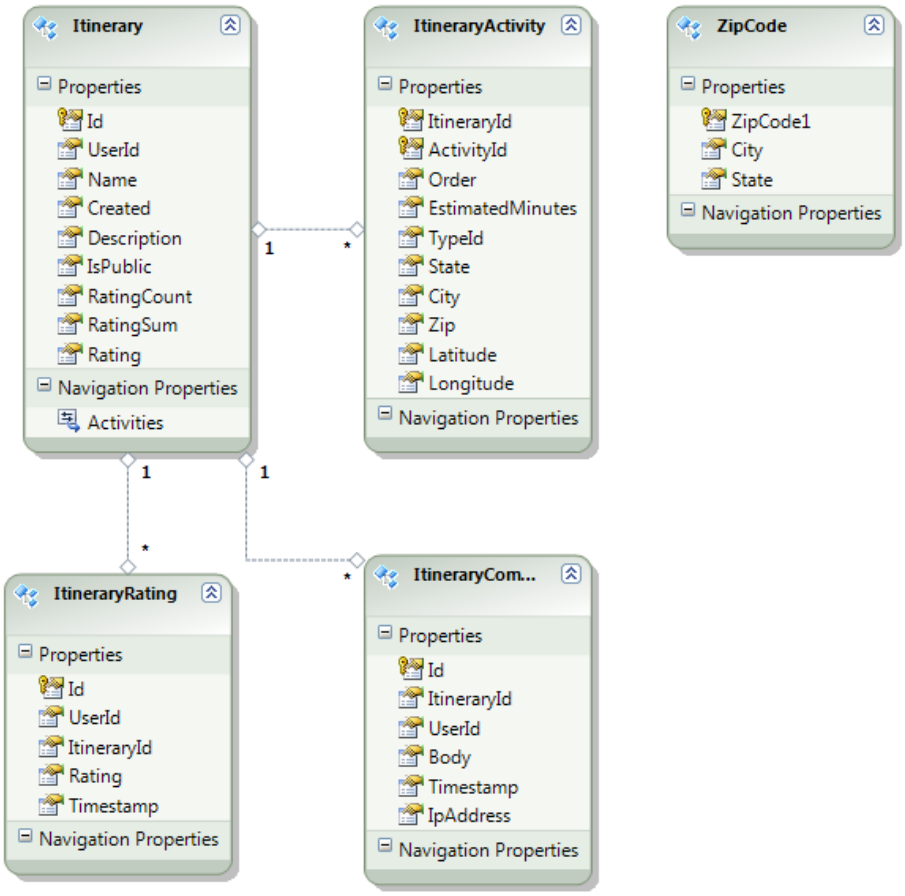


Рис. 1-8 Модель, импортированная из базы данных PlanMyNight

Можно заметить, что одному из свойств сущности ZipCode автоматически присвоено имя *ZipCode1*, потому что сама таблица уже называется ZipCode, а имя должно быть уникальным. Чтобы исправить это имя, щелкнем его двойным щелчком. Зададим имя **Code**, как показано на рис. 1-9.

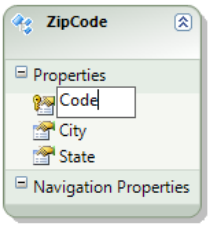


Рис. 1-9 Сущность ZipCode

Выполним сборку решения, нажав `Ctrl+Shift+B`. В окне вывода можно увидеть два сообщения от сформированной EDM. Первое можно сразу удалить, поскольку столбец Location (Местоположение) в PMN не нужен. Второе сообщение гласит:

> *Для таблицы/представления 'dbo.UserProfile' первичный ключ не задан и не может быть сформирован. Данная таблица/представление не включена в модель. Для использования этой сущности необходимо пересмотреть схему, добавить соответствующие ключи и раскомментировать ее.*

Взглянув на представление UserProfile можно заметить, что первичный ключ не задан явно, даже несмотря на то что значения столбца UserName (Имя пользователя) уникальны.

Чтобы иметь возможность выполнять доступ к данным UserProfile из приложения, придется вручную подкорректировать EDM и исправить сопоставление для представления UserProfile.

В обозревателе проекта щелкните правой кнопкой мыши файл PlanMyNight.edmx и выберите Open With (Открыть в). В диалоговом окне Open With выберите XML (Text) Editor ((Текстовый) редактор XML), как показано на рис. 1-10. Щелкните OK, чтобы открыть XML-файл, связанный с нашей моделью[1].

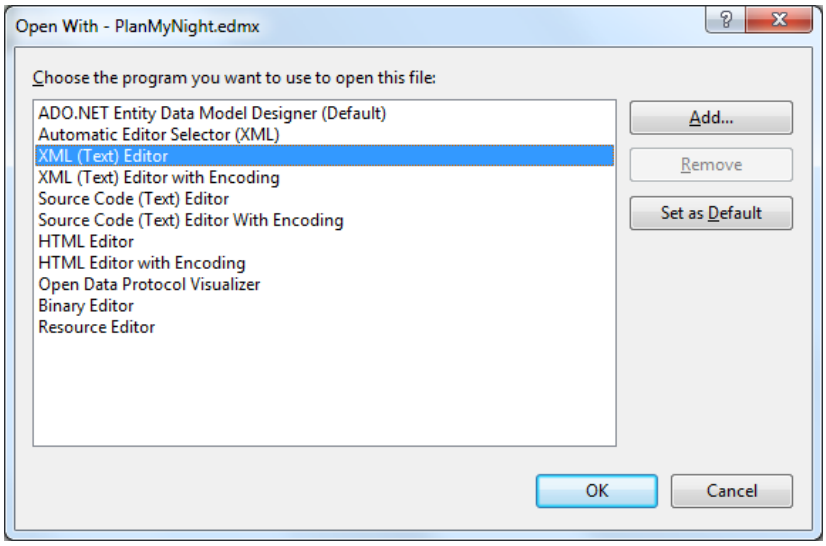


Рис. 1-10 Открываем PlanMyNight.edmx в редакторе XML

> **Примечание** На экран будет выведено сообщение о том, что файл PlanMyNight.edmx уже открыт. Щелкните Yes, чтобы закрыть его

Механизм формирования кода закомментировал созданный код из-за отсутствия первичного ключа. Чтобы использовать представление UserProfile из дизайнера, необходимо раскомментировать тип сущности UserProfile и добавить в него тег Key (Ключ). Найдите UserProfile в файле. Раскомментируйте тип сущности, добавьте тег ключа, задайте для него имя **UserName** и сделайте свойство *UserName* не допускающим значение null. Обновленный тип сущности представлен в листинге 1-1.

Листинг 1-1 XML-описание типа сущности UserProfile

```
<EntityType Name="UserProfile">
      <Key>
         <PropertyRef Name="UserName"/>
      </Key>
      <Property Name="UserName" Type="uniqueidentifier" Nullable="false" />
      <Property Name="FullName" Type="varchar" MaxLength="500" />
      <Property Name="City" Type="varchar" MaxLength="500" />
      <Property Name="State" Type="varchar" MaxLength="500" />
      <Property Name="PreferredActivityTypeId" Type="int" />
</EntityType>
```

Если закрыть XML-файл и попытаться открыть дизайнер EDM, в нем будет выведена такая ошибка: «The Entity Data Model Designer is unable to display the file you requested. You can edit the model using the XML Editor[2]».

На панели Error List (Список ошибок) выводится следующее предупреждение, более подробно раскрывающее суть этой ошибки:

Error 11002: Entity type 'UserProfile' has no entity set.[3]

Необходимо задать множество сущностей для типа UserProfile, чтобы тип сущности мог проецироваться на схему хранилища. Для этого открываем файл PlanMyNight.edmx в редакторе XML. Вверху файла, прямо над множеством сущностей Itinerary, добавляем XML-код, представленный в листинге 1-2.

Листинг 1-2 XML-описание множества сущностей для UserProfile

```
<EntitySet Name="UserProfile" EntityType="Entities.Store.UserProfile"

     store:Type="Views" store:Schema="dbo" store:Name="UserProfile">
   <DefiningQuery>
```

[1] Этот XML-файл содержит описание всей созданной модели (*прим. технического редактора*)

[2] Дизайнер Entity Data Model не может вывести на экран запрашиваемый файл. Скорректируйте модель в редакторе XML (*прим. переводчика*).

[3] Для типа сущности 'UserProfile' не задано множество сущностей (*прим. переводчика*).

```
      SELECT
     [UserProfile].[UserName] AS [UserName],
     [UserProfile].[FullName] AS [FullName],
           [UserProfile].[City] AS [City],
         [UserProfile].[State] AS [State],
         [UserProfile].[PreferredActivityTypeId] as [PreferredActivityTypeId]
         FROM [dbo].[UserProfile] AS [UserProfile]
   </DefiningQuery>
</EntitySet>
```

Сохраним XML-файл EDM и повторно откроем дизайнер EDM. На рис. 1-11 показано представление UserProfile в разделе Entities.Store браузера модели (Model Browser).

> **Подсказка** Model Browser можно открыть из меню View (Вид), щелкнув Other Windows (Другие окна) и выбрав Entity Data Model Browser.

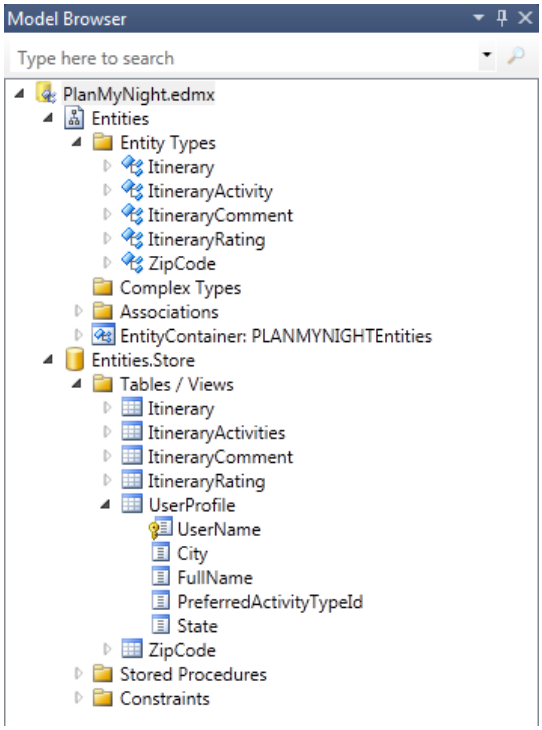


Рис. 1-11 Представление UserProfile в Model Browser

Теперь, когда представление доступно в метаданных хранилища, добавим сущность UserProfile и сопоставим ее с представлением UserProfile. Щелкаем правой кнопкой мыши фон дизайнера EDM, выбираем Add и затем Entity. На экран будет выведено диалоговое окно, показанное на рис. 1-12.

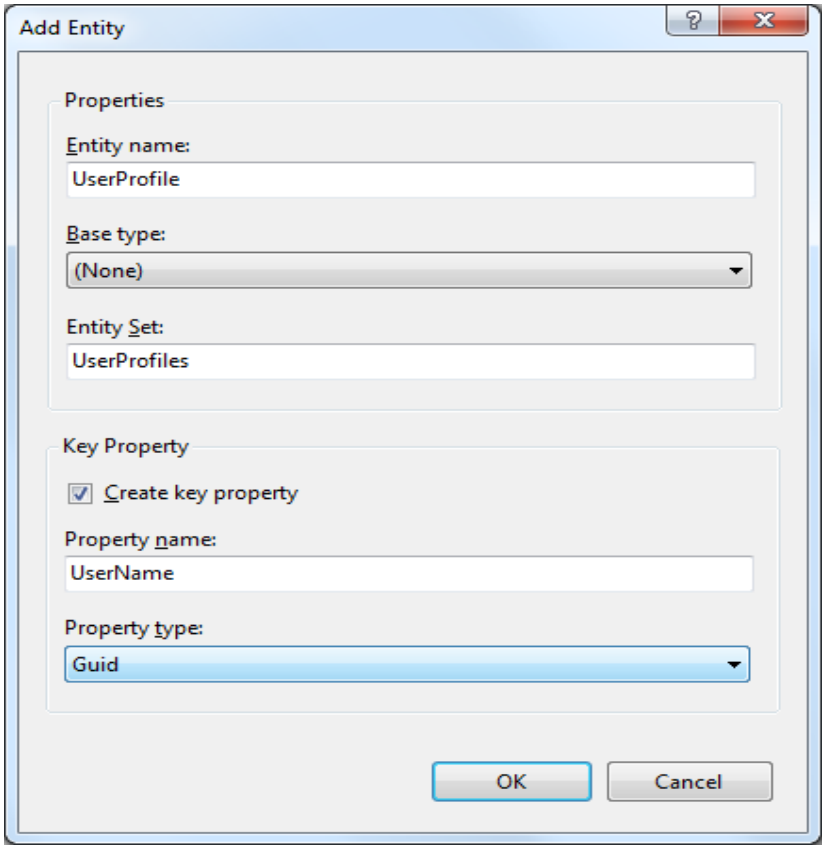


Рис. 1-12 Диалоговое окно Add Entity

Заполните диалоговое окно, как показано на рис. 1-12, и щелкните OK, чтобы создать сущность.

После этого понадобится добавить остальные свойства: *City* (Город), *State* (Страна) и *PreferedActiviyTypeId* (Идентификатор предпочтительного типа мероприятий). Для этого щелкаем правой кнопкой мыши сущность UserProfile, выбираем Add и Scalar Property (Скалярное свойство). Как только свойство добавлено, задаем значения полей *Type* (Тип), *Max Length* (Максимальная длина) и *Unicode*. В табл. 1-1 представлены ожидаемые значения для каждого из полей.

Таблица 1-1 Свойства сущности UserProfile

Name	Type	Max Length	Unicode
FullName	*String*	500	False
City	*String*	500	False
State	*String*	500	False
PreferredActivityTypeId	*Int32*	Недоступно	Недоступно

Теперь, когда сущность UserProfile создана, ее требуется сопоставить с представлением UserProfile. Щелкаем правой кнопкой мыши сущность UserProfile и выбираем Table Mapping (Сопоставление таблиц), как показано на рис. 1-13.

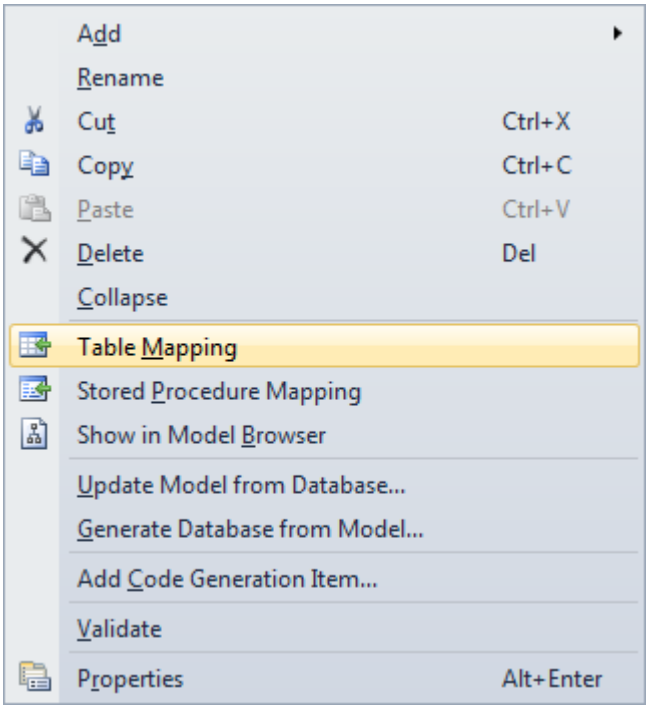


Рис. 1-13 Пункт меню Table Mapping

Затем выбираем представление UserProfile из раскрывающегося списка, как показано на рис. 1-14. Убедитесь, что все столбцы и свойства сущности сопоставлены правильно. Теперь представление UserProfile нашего хранилища доступно из кода через сущность UserProfile.

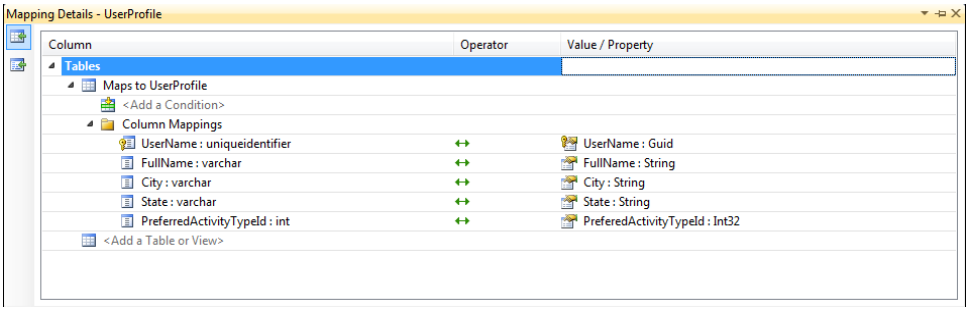


Рис. 1-14 Детали сопоставления UserProfile

Хранимая процедура и импортированные функции

Мастер Entity Data Model Wizard (Мастер модели сущность-данные) создал в модели хранилища запись для хранимой процедуры *RetrieveItinerariesWithinArea*, которая была выбрана в мастере в заключительном шаге. Необходимо создать соответствующую запись в концептуальной модели, добавив Function Import (Импортированная функция).

В разделе Entities.Store браузера модели откройте папку Stored Procedures (Хранимые процедуры). Щелкните правой кнопкой мыши *RetrieveItineraryWithinArea* и выберите Add Function Import (Добавить импортированную функцию). Диалоговое окно Add Function Import представлено на рис. 1-15. Задайте возвращаемый тип Entities и затем выберите элемент Itinerary в окне раскрывающегося списка. Щелкните OK.

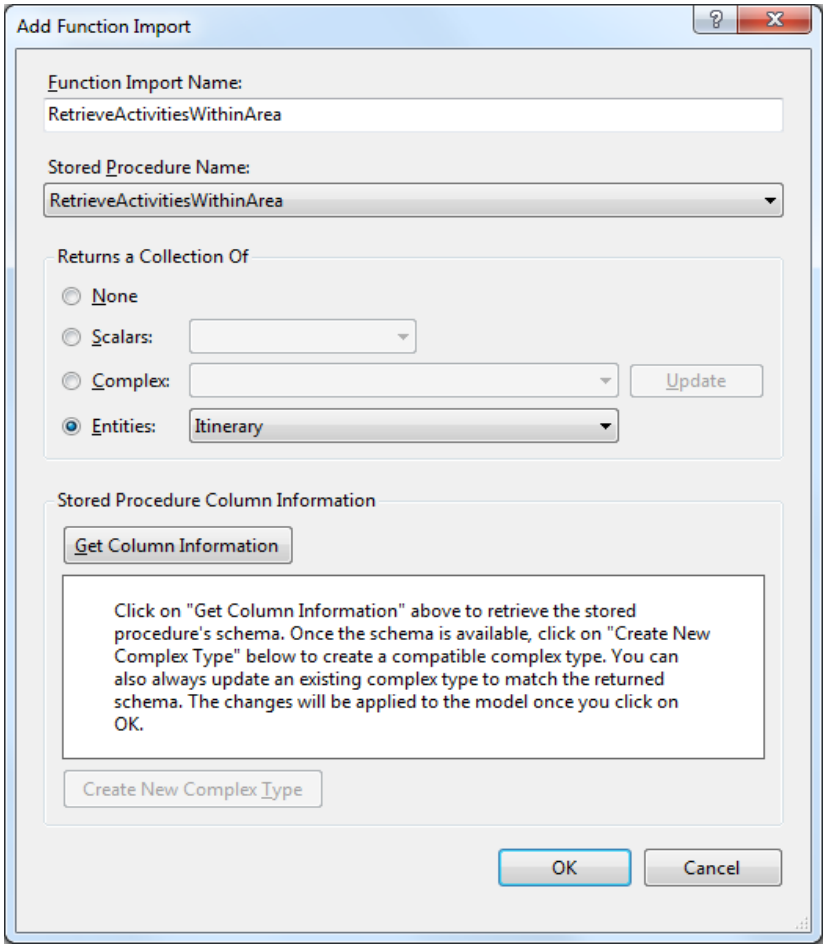


Рис. 1-15 Диалоговое окно Add Function Import

В браузер модели добавлена импортированная функция *RetrieveItinerariesWithinArea*, как показано на рис. 1-16.

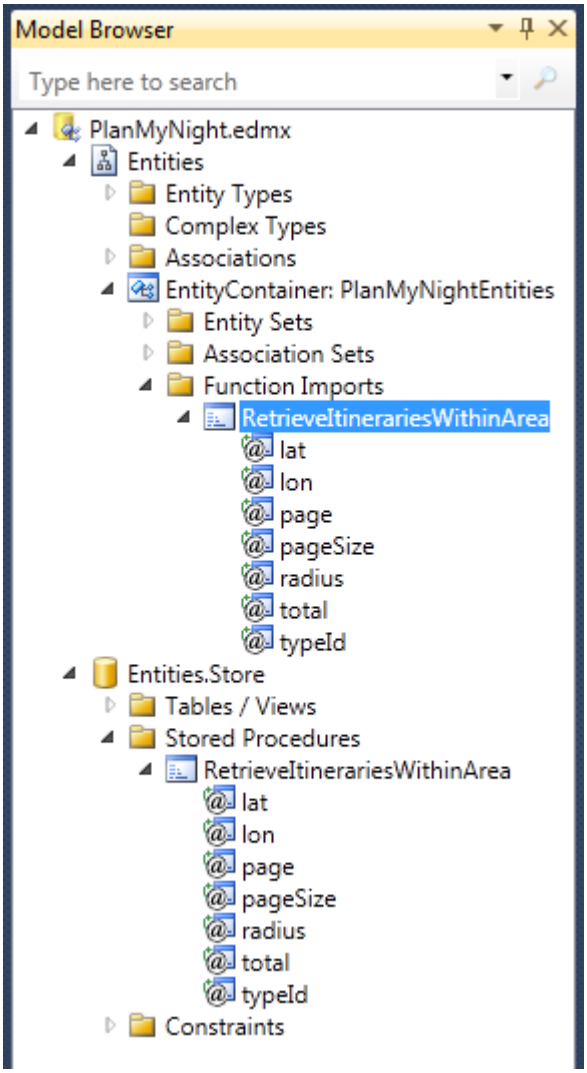


Рис. 1-16 Импортированные функции в браузере модели

Теперь можно проверить правильность EDM, щелкнув правой кнопкой мыши рабочую область дизайнера и выбрав Validate (Проверить). При этом не должно появиться никаких ошибок или предупреждений.

EF: сначала модель[1]

В предыдущем разделе мы увидели, как использовать дизайнер EF для создания модели путем импорта существующей базы данных. Дизайнер EF в Visual Studio 2003 также поддерживает возможность автоматического формирования файла Data Definition Language (DDL), который позволит создать базу данных из модели сущностей. В этом разделе мы используем новое решение, чтобы научиться автоматически формировать сценарий базы данных из модели.

Можно начать с пустой модели. Для этого в мастере Entity Data Model Wizard выбираем опцию Empty model (Пустая модель), как показано на рис. 1-17.

> **Примечание** Чтобы открыть мастер, щелкните правой кнопкой проект PlanMyNight.Data, выберите Add и затем New Item. Выберите элемент ADO.NET Entity Data Model.

[1] Модель-ориентированный подход (*прим. технического редактора*)

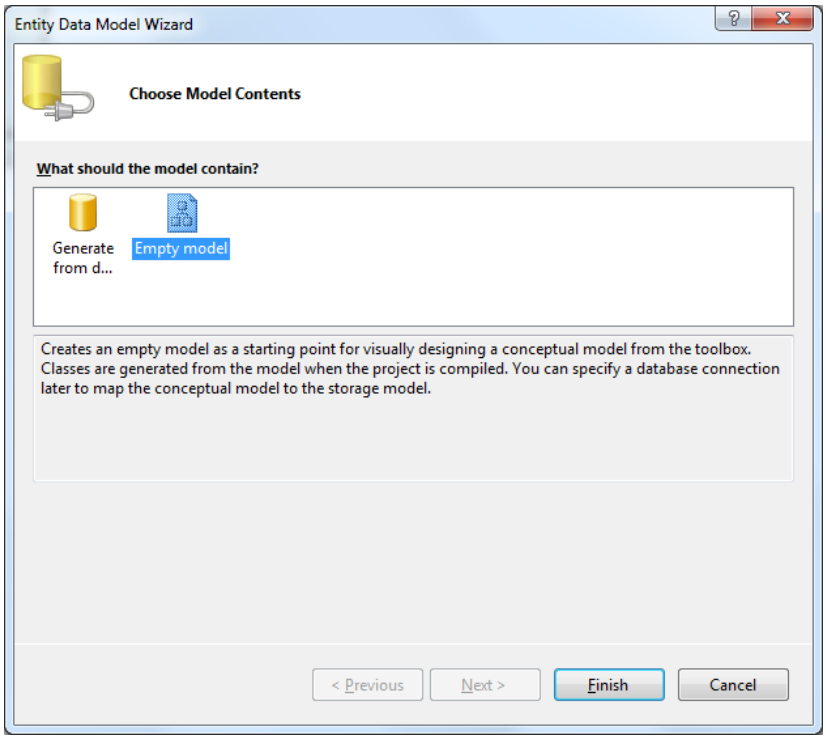


Рис. 1-17 Мастер EDM: Empty model

Открываем решение PMN по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 1\Code\ModelFirst, щелкнув двойным щелчком файл PlanMyNight.sln.

Проект PlanMyNight.Data этого решения уже содержит EDM-файл PlanMyNight.edmx с некоторыми уже созданными сущностями. Эти сущности соответствуют схеме данных, представленной на рис. 1-2.

Дизайнер Entity Model позволяет без труда добавлять сущности в модель данных. Добавим в нашу модель недостающую сущность ZipCode. Из панели инструментов перенесите в дизайнер элемент Entity, как показано на рис. 1-18. Присвойте этой сущности имя **ZipCode**, свойству *Id* – имя **Code** и задайте для него тип *String*.

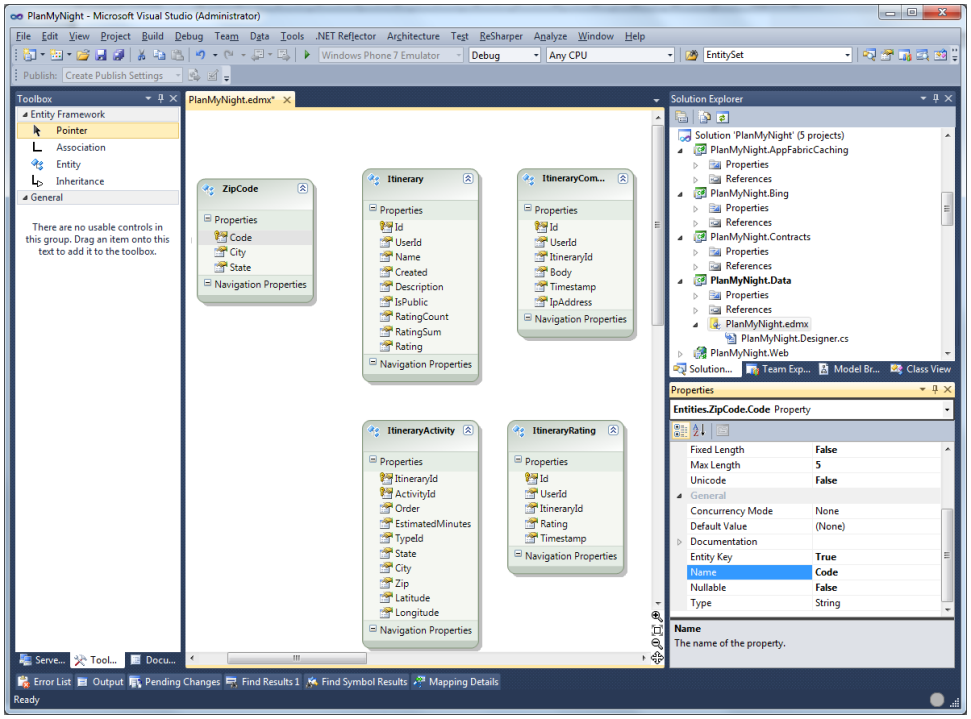


Рис. 1-18 Дизайнер модели сущностей

Теперь необходимо добавить в сущность свойства *City* и *State*. Щелкаем правой кнопкой сущность ZipCode, выбираем Add и затем Scalar Property. Убеждаемся, что значения всех свойств соответствуют приведенным в табл. 1-2.

Таблица 1-2 Свойства сущности ZipCode

Name	Type	Fixed Length	Max Length	Unicode
Code	*String*	False	5	False
City	*String*	False	150	False
State	*String*	False	150	False

Добавим отношения между ItineraryComment и сущностями Itinerary. Щелкаем правой кнопкой поверхность дизайнера, выбираем Add и затем Association (Связь), как показано на рис. 1-19.

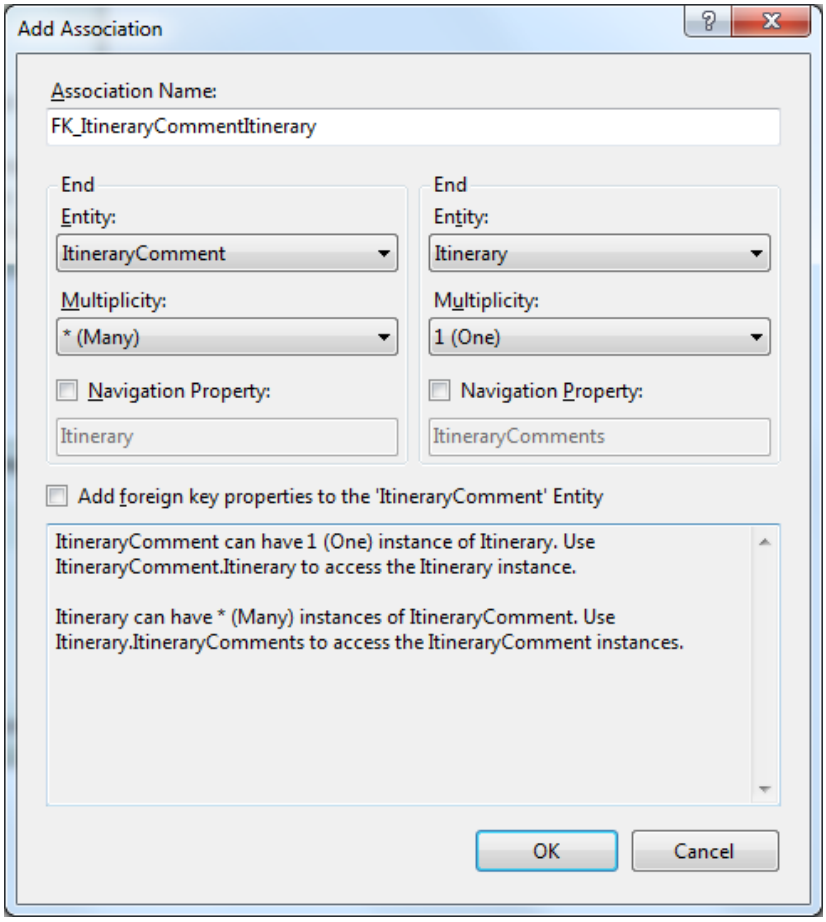


Рис. 1-19 Диалоговое окно Add Association для FK_ItineraryCommentItinerary

Задайте имя связи **FK_ItineraryCommentItinerary**, выберите сущность и кратность для каждого конца связи, как показано на рис. 1-19. Как только связь создана, щелкните двойным щелчком строку связи, чтобы задать Referential Constraint (Ссылочное ограничение), как показано на рис. 1-20.

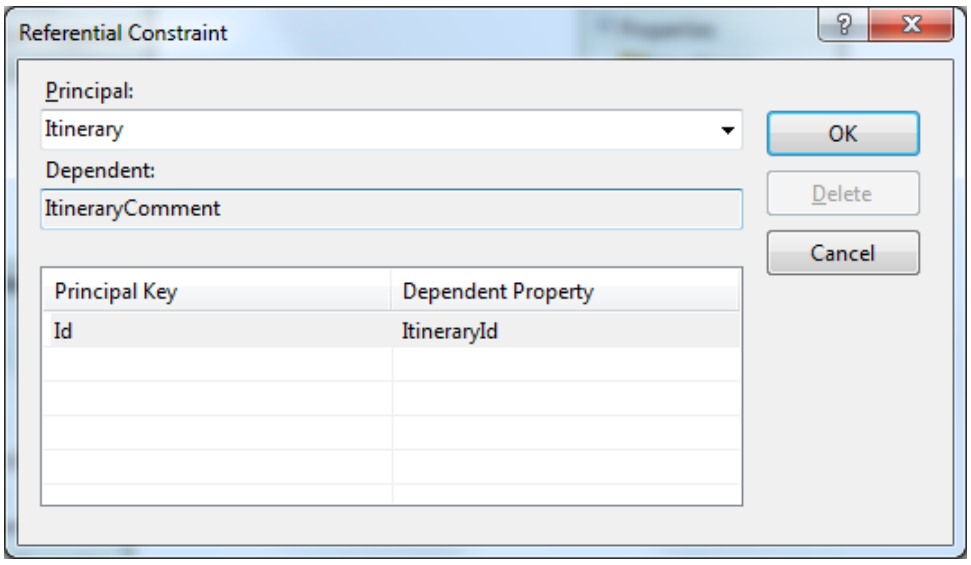


Рис. 1-20 Диалоговое окно Association Referential Constraint

Добавьте связь между сущностями ItineraryRating и Itinerary. Щелкните правой кнопкой фон дизайнера, выберите Add и затем Association. Задайте имя связи **FK_ItineraryItineraryRating**. После этого выберите сущность и кратность для каждого конца связи, как делали это в предыдущем случае, только в качестве первого конца связи задайте **ItineraryRating**. Щелкните двойным щелчком линию связи и задайте Referential Constraint, как показано на рис. 1-20. Обратите внимание, что в поле Dependent (Зависимый) отображается не ItineraryComment, а ItineraryRating.

Создайте новую связь между сущностями ItineraryActivity и Itinerary. Также для связи *FK_ItineraryItineraryActivity* потребуется создать навигационное свойство и назвать его **Activities**, как показано на рис. 1-21. Когда связь создана, задайте для нее Referential Constraint, щелкнув двойным щелчком линию связи.

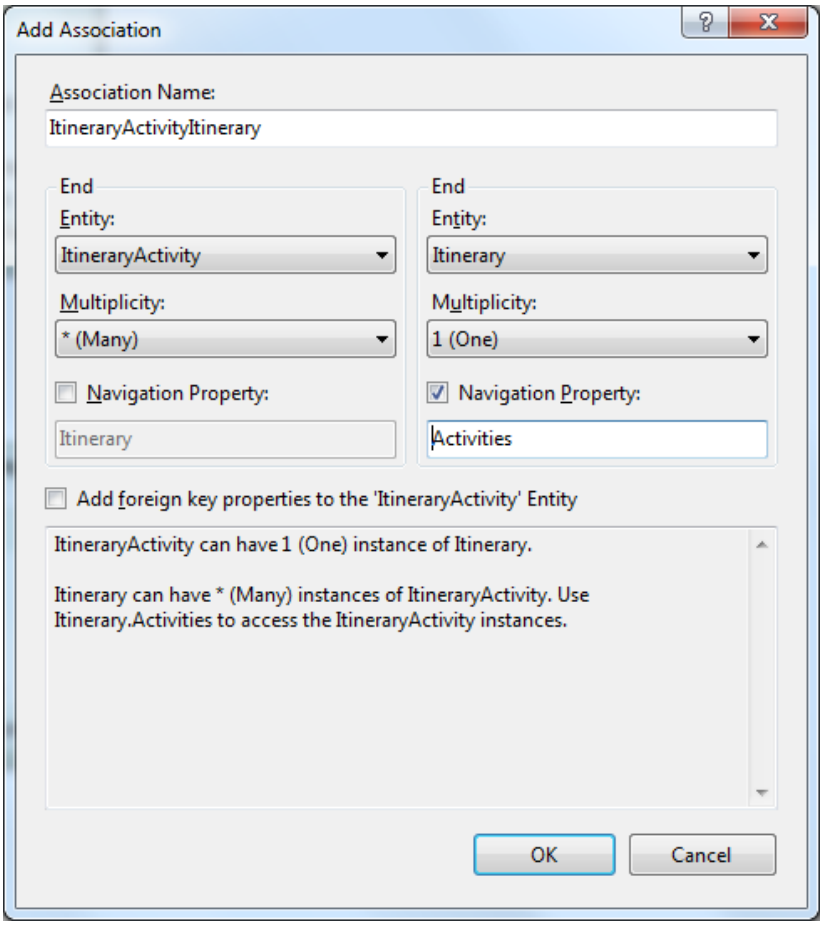


Рис. 1-21 Диалоговое окно Add Association для FK_ItineraryActivityItinerary

Автоматическое формирование сценария базы данных из модели

Модель данных готова, но с ней не ассоциировано никакое хранилище или сопоставление. Дизайнер EF предлагает возможность автоматического формирования сценария базы данных из модели.

Щелкните правой кнопкой дизайнер и выберите пункт меню Generate Database From Model (Сформировать базу данных из модели), как показано на рис. 1-22.

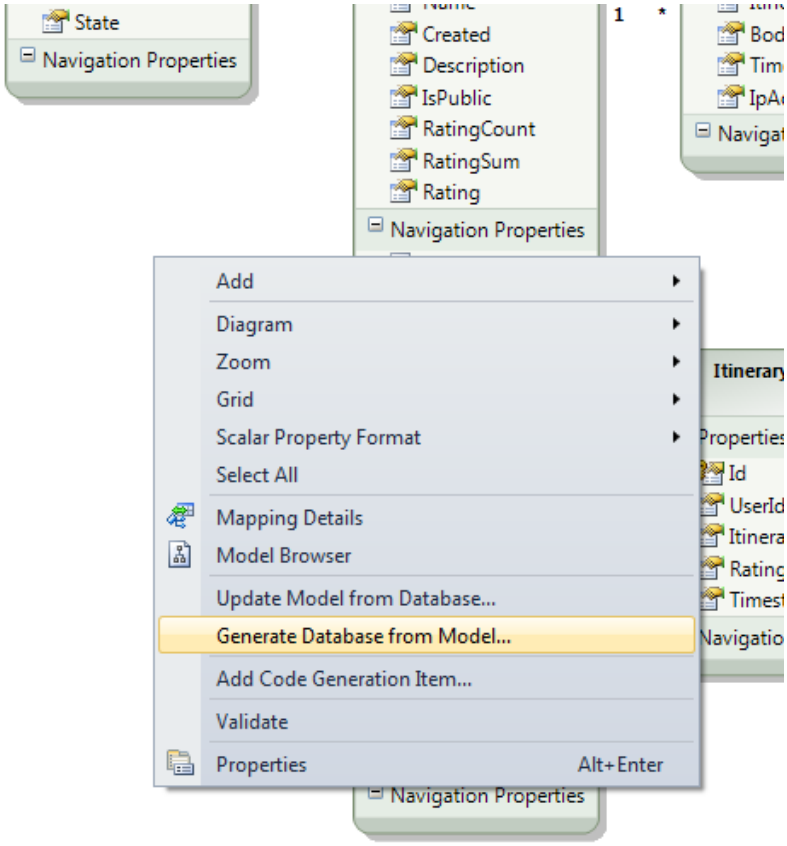


Рис. 1-22 Пункт меню Generate Database From Model

Мастер Generate Database Wizard требует подключения к данным. Этот мастер использует данные подключения для трансляции типов модели в типы базы данных и для формирования DDL-сценария для этой базы данных.

Выберите New Connection, в диалоговом окне Choose Data Source укажите Microsoft SQL Server File и щелкните Continue (Продолжить). Выберите файл базы данных, находящийся по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 1\Code\ModelFirst\Data\PlanMyNight.mdf (рис. 1-23).

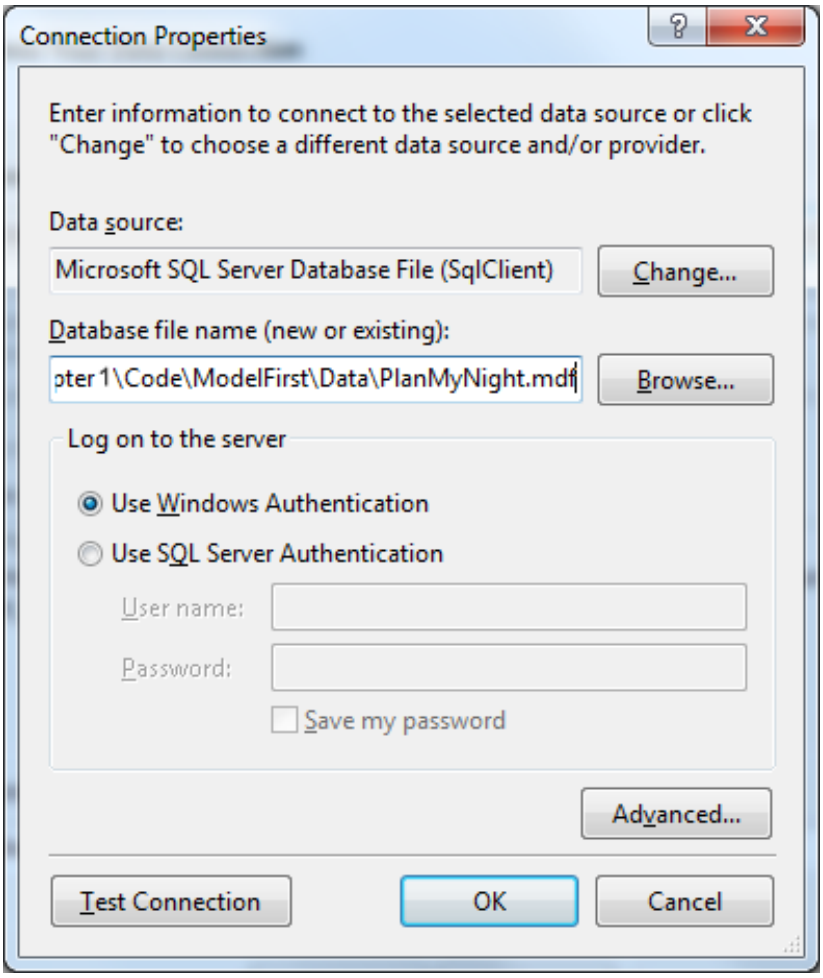


Рис. 1-23 Подключение базы данных для автоматического формирования сценария

Как только подключение настроено, щелкните Next, чтобы перейти к последнему окну мастера, как показано на рис. 1-24. По нажатию Finish сформированный файл T-SQL PlanMyNight.edmx.sql добавляется в проект. DDL-сценарий сформирует ограничения первичного и внешнего ключей для модели.

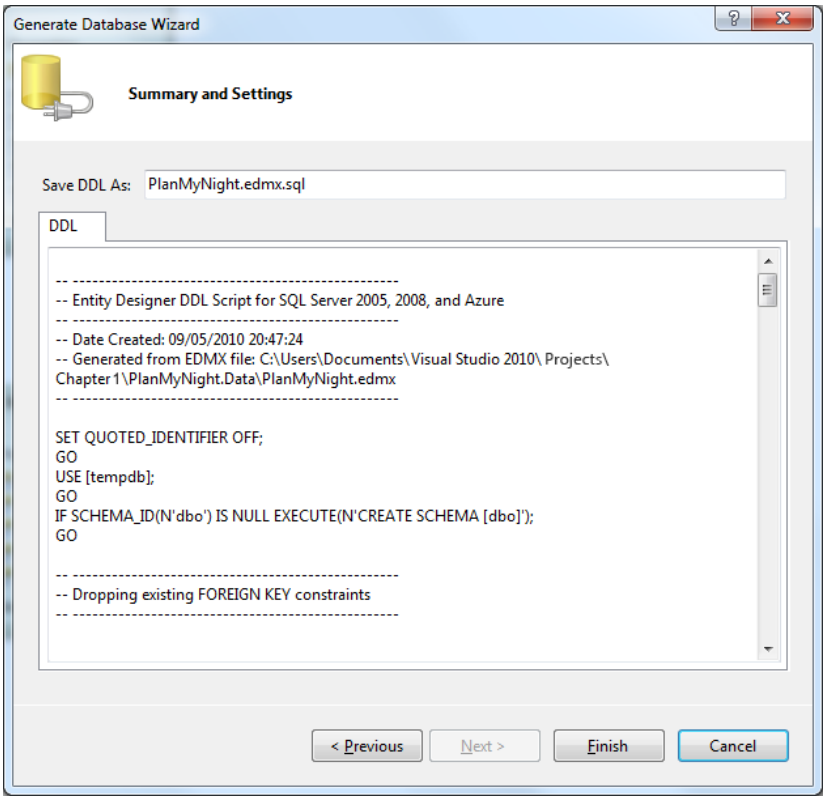


Рис. 1-24 Сформированный файл T-SQL

EDM также обновляется, чтобы вновь созданное хранилище гарантированно сопоставлялось с сущностями. Теперь сформированный DDL-сценарий может использоваться для добавления таблиц в базу данных. Также мы получили слой доступа к данным, предоставляющий строго типизированные сущности, которые могут использоваться в приложении.

> **Важно** Для создания полной базы данных PMN потребовалось бы добавить оставшиеся таблицы, хранимые процедуры и триггеры, используемые приложением. Вместо того чтобы осуществлять все эти операции, вернемся к решению, каким оно было в конце раздела «EF: импорт существующей базы данных».

POCO-шаблоны

Для формирования кода сущностей дизайнер EDM использует шаблоны T4. До сих пор мы позволяли дизайнеру создавать сущности на базе шаблонов по умолчанию. Автоматически сформированный код можно увидеть в файле PlanMyNight.Designer.cs, ассоциированном с PlanMyNight.edmx. Созданные в нем сущности типа EntityObject и снабжены атрибутами, которые обеспечивают возможность EF управлять ими во время выполнения.

> **Примечание** T4 расшифровывается как *Text Template Transformation Toolkit* (Набор инструментов для преобразования текстовых шаблонов). Поддержка T4 в Visual Studio 2010 позволяет без труда создавать собственные шаблоны и формировать любые текстовые файлы (Веб, ресурсы или источники). Более подробные сведения об автоматическом формировании кода в Visual Studio 2010 можно найти в статье Code Generation and Text Templates (Автоматическое формирование кода и текстовые шаблоны) по адресу *http://msdn.microsoft.com/en-us/library/bb126445(VS.100).aspx*.

EF также поддерживает типы сущностей POCO. POCO-классы – это простые объекты без атрибутов или базового класса инфраструктуры. (Листинг 1-3 в следующем разделе представляет POCO-класс для сущности ZipCode.) EF использует имена типов и свойства этих объектов для их сопоставления с моделью во время выполнения.

> **Примечание** POCO расшифровывается как *Plain-Old CLR Objects*[1].

Шаблон ADO.NET POCO Entity Generator

Повторно откроем файл %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 1\Code\ExistingDatabase\PlanMyNight.sln.

Выберем файл PlanMyNight.edmx, щелкнем правой кнопкой дизайнер и выберем Add Code Generation Item (Добавить элемент формирования кода). При этом откроется диалоговое окно, показанное на рис. 1-25, которое предоставляет возможность выбрать необходимый шаблон. Выберем шаблон ADO.NET POCO Entity Generator[2] и назовем его **PlanMyNight.tt**. После этого щелкнем кнопку Add.

> **Примечание** Возможно, на экран будет выведено предупреждение об угрозе безопасности в случае выполнения этого текстового шаблона. Источник данного шаблона доверенный, поэтому просто щелкните OK, чтобы закрыть это диалоговое окно.

[1] Обычные объекты CLR (*прим. переводчика*)

[2] По-умолчанию данный шаблон не установлен. Если у Вас его нет, выберите страницу Online Templates (он-лайн шаблоны), а далее введите слово POCO в поле Search Online Templates (Поиск он-лайн шаблонов) и нажмите Enter, через некоторое время шаблон будет найден, его можно будет установить и использовать (*прим. технического редактора*)

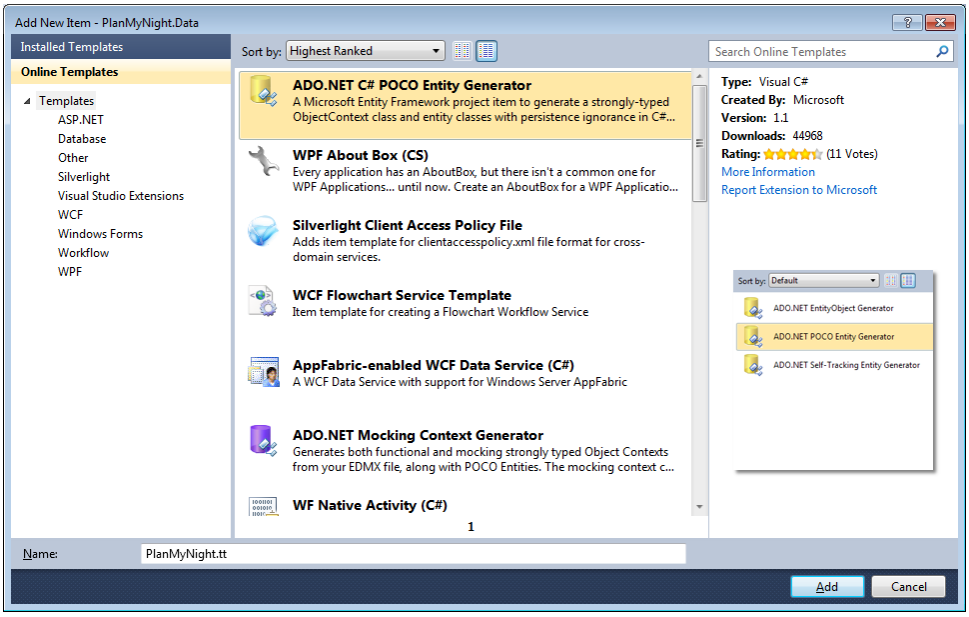


Рис. 1-25 Диалоговое окно Add New Item

В проект были добавлены два файла, PlanMyNight.tt и PlanMyNight.Context.tt (рис. 1-26). Эти файлы заменяют шаблон формирования кода по умолчанию, и код больше не формируется в файле PlanMyNight.Designer.cs.

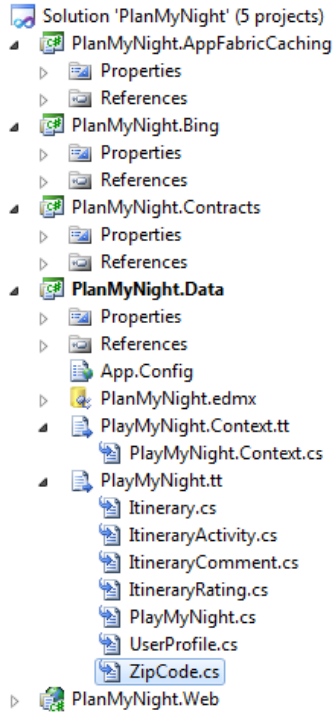


Рис. 1-26 Добавленные шаблоны

Шаблон PlanMyNight.tt создает по файлу класса для каждой сущности в модели. Листинг 1-3 представляет POCO-версию класса *ZipCode*.

Листинг 1-3 POCO-версия класса ZipCode

```
namespace Microsoft.Samples.PlanMyNight.Data
{
    public partial class ZipCode
    {
        #region Primitive Properties
        public virtual string Code
        {
            get;
            set;
```

```
        }
        public virtual string City
        {
            get;
            set;
        }
        public virtual string State
        {
            get;
            set;
        }
        #endregion
    }
}
```

Подсказка В C# 2.0 появились частичные классы, благодаря которым реализация класса может быть разделена между множеством файлов. Каждый из этих файлов может включать один или более членов. Файлы компонуются во время компиляции приложения. Частичные классы особенно полезны, когда требуется добавить код в автоматически сформированные классы. Код добавляется вне сформированного файла и не будет переопределен в случае повторного формирования класса.

Подсказка В C# 3.0 появляется новая возможность, называемая *автоматические свойства*. Для каждого обнаруженного во время компиляции пустого блока *get* или *set* компилятор создает резервное поле.

Другой файл, PlanMyNight.Context.cs, формирует объект *ObjectContext* (Контекст объекта) для модели PlanMyNight.edmx. Этот объект мы будем использовать для взаимодействия с базой данных.

Подсказка Шаблоны POCO будут автоматически обновлять сформированные классы соответственно внесенным в модель изменениям при сохранении файла .edmx.

Перенос классов сущностей в проект контрактов

Архитектура приложения PMN спроектирована таким образом, что слой представления гарантированно безразличен к методу хранения. Это реализовано путем перемещения контрактов и классов сущностей в сборку, которая не имеет ссылки на хранилище.

Visual Studio 2003 Даже несмотря на то, что в Visual Studio 2003 существовала возможность написания надстроек для автоматического формирования кода, исходя из базы данных, это было связано с определенными сложностями и требовало обслуживания этих инструментов. EF использует шаблоны T4 для формирования и схемы базы данных, и кода. Эти шаблоны можно без труда настроить соответственно конкретным требованиям.

Шаблоны POCO ADO.NET выносят формирование классов сущностей в отдельный шаблон, что дает нам возможность с легкостью переносить эти сущности в другой проект.

Перенесем файл PlanMyNight.tt в проект PlanMyNight.Contracts. Щелкнув правой кнопкой файл PlanMyNight.tt, выбираем Cut (Вырезать). Щелкаем правой кнопкой папку Entities проекта PlanMyNight.Contracts и выбираем Paste (Вставить). Полученный результат представлен на рис. 1-27.

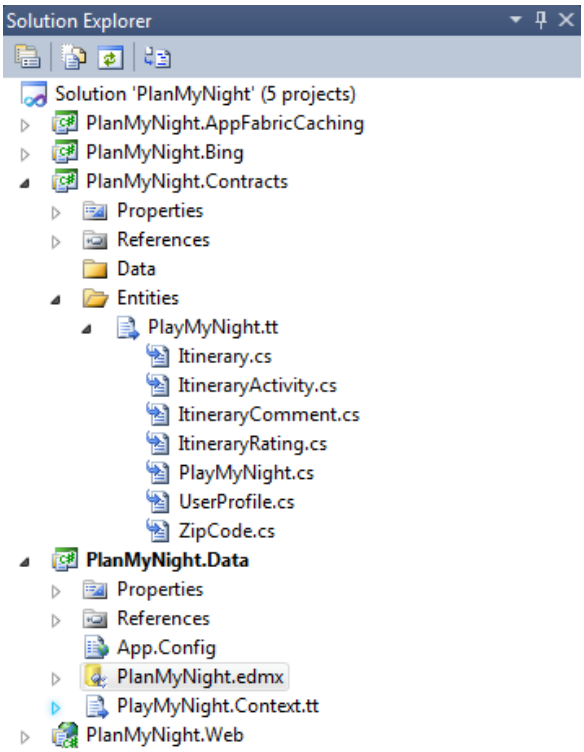


Рис. 1-27 Шаблон POCO, перенесенный в проект Contracts

Для формирования кода типа сущности шаблон PlanMyNight.tt использует метаданные модели EDM. Необходимо скорректировать относительный путь, используемый шаблоном для доступа к файлу EDMX.

Откройте шаблон PlanMyNight.tt и найдите строку:

```
string inputFile = @"PlanMyNight.edmx";
```

Исправьте путь к файлу, чтобы он указывал на файл PlanMyNight.edmx в проекте PlanMyNight.Data:

```
string inputFile = @"..\..\PlanMyNight.Data\PlanMyNight.edmx";
```

Классы сущностей будут сформированы повторно, как только вы сохраните шаблон.

Также необходимо обновить шаблон PlanMyNight.Context.tt в проекте PlanMyNight.Contracts, поскольку теперь классы сущностей находятся в пространстве имен *Microsoft.Samples.PlanMyNight.Entities*, а не в *Microsoft.Samples.PlanMyNight.Data*. Откройте файл PlanMyNight.Context.tt и включите новое пространство имен в раздел директив *using*.

```
using System;
using System.Data.Objects;
using System.Data.EntityClient;
using Microsoft.Samples.PlanMyNight.Entities;
```

Нажав `Ctrl+Shift+B`, выполните сборку решения. Теперь компиляция проекта должна пройти успешно.

Сведение воедино

Универсальный код для взаимодействия с базой данных SQL создан, теперь все готово для реализации специальной функциональности приложения PMN. В следующих разделах мы поэтапно рассмотрим этот процесс, вкратце остановимся на получении данных с сервисов Bing Maps и пройдем небольшой вводный курс в возможность кэширования AppFabric Windows Server, используемую в PMN.

Чтобы свести все это воедино, требуется довольно большой объем вспомогательного кода. Упростить процесс поможет использование обновленного решения, в котором уже написана большая часть кода для контрактов и сущностей, а также для работы с сервисами Bing Maps. Решение также будет включать проект PlanMyNight.Data.Test для валидации кода проекта PlanMyNight.Data.

> **Примечание** Тестирование в Visual Studio 2010 будет рассмотрено в главе 3.

Получение данных из базы данных

В начале этой главы нами было принято решение сгруппировать операции над сущностью Itinerary в интерфейс хранилища *IItinerariesRepository*. Вот некоторые из этих операций:

- Поиск плана мероприятий по мероприятию
- Поиск плана мероприятий по почтовому индексу
- Поиск плана мероприятий по радиусу
- Добавление нового плана мероприятий

Рассмотрим соответствующие методы интерфейса *IItinerariesRepository*:

- *SearchByActivity* обеспечит поиск планов мероприятий по мероприятию и возвращение страницы данных.
- *SearchByZipCode* обеспечит поиск планов мероприятий по почтовому индексу и возвращение страницы данных.
- *SearchByRadius* обеспечит поиск планов мероприятий из определенного пункта и возвращение страницы данных.
- *Add* обеспечит добавление плана мероприятий в базу данных.

Откроем решение PMN, располагающееся по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 1\Code\Final, щелкнув двойным щелчком файл PlanMyNight.sln.

Выберем проект PlanMyNight.Data и откроем файл ItinerariesRepository.cs. Это реализация интерфейса *IItinerariesRepository*. Используя сформированный ранее объект контекста PlanMyNightEntities, мы можем создавать запросы LINQ к модели, и EF транслирует их в обычный T-SQL, который будет применяться для работы с базой данных.

Примечание LINQ расшифровывается как Language Integrated Query[1] и появился в .NET Framework 3.5. Он обеспечивает встроенную возможность построения запросов к данным в .NET Framework, так что разработчику не приходится беспокоиться об изучении и обслуживании специфических запросов SQL. LINQ позволяет использовать строго типизированные объекты, а Visual Studio IntelliSense обеспечивает возможность выбирать свойства или методы, присутствующие в текущем контексте, как показано на рис. 1-28. Больше сведений о LINQ предоставлено в ресурсе .NET Framework Developer Center (*http://msdn.microsoft.com/en-us/netframework/aa904594.aspx*).

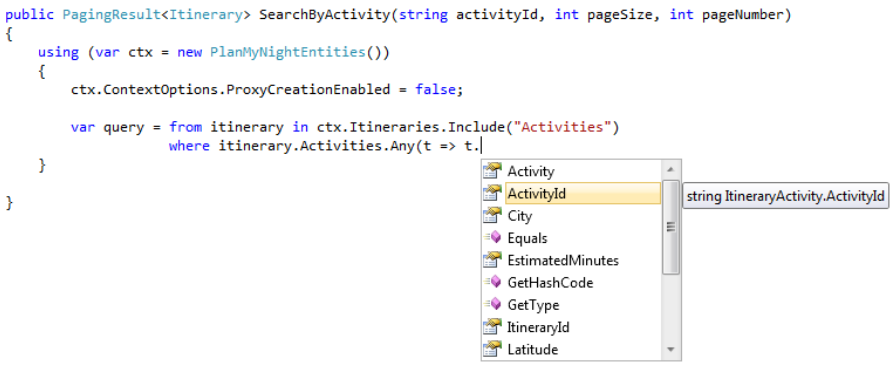


Рис. 1-28 Поддержка IntelliSense для запросов LINQ

Перейдите к описанию функции *SearchByActivity*. Этот метод должен возвращать набор планов мероприятий, для которых флаг *IsPublic* (Общедоступный) установлен в значение true, и значение *activityId* одного из мероприятий соответствует переданному в аргументе функции значению. Полученные в результате планы мероприятий должны быть сортированы по полю рейтинга.

Visual Studio 2003 Реализация каждого метода для извлечения плана мероприятий в Visual Studio 2003 потребовала бы написания специализированных SQL-запросов. С EF и LINQ любой запрос становится типовым, а изменения вносятся без труда на уровне кода!

SearchByActivity реализуется с применением стандартных операторов LINQ, как показано в листинге 1-4. Добавим выделенный код в тело метода *SearchByActivity*.

Visual Studio 2003 Универсальные типы были добавлены в версию 2.0 C# и CLR (Общеязыковая среда выполнения). Универсальные типы вводят концепцию параметров типов, что делает возможным создавать классы и методы, которые откладывают описание типов до объявления этих классов или методов. Часто они применяются с коллекцией, где параметр типа выступает в роли заполнителя для типа хранимых в ней объектов.

В PMN универсальные типы используются для хранения результатов разных типов. Подобный класс в Visual Studio 2003 можно было бы написать с помощью *ArrayList* (Список массивов):

[1] Язык интегрированных запросов (*прим. переводчика*)

```
public class PagingResult
{
    private ArrayList items;
    ...
    public PagingResult(Array items)
    {
        this.items = new ArrayList(items);
    }
    ...
    public ArrayList Items
    {
        get { return this.items; }
    }
}
```

Чтобы использовать этот класс в своем коде, необходимо знать тип входящих в него объектов и оценивать затраты или риски приведений или операций упаковки во время выполнения. Применение универсального параметра типа *T* позволяет создать класс со строгой типизацией, таким образом, компилятор предотвратит любое его неправильное использование во время сборки:

```
public class PagingResult<T>
{
    public PagingResult(IEnumerable<T> items)
    {
        this.Items = new List<T>(items);
    }
    ...
    public ICollection<T> Items { get; }
}
```

Более подробно универсальные типы рассматриваются в материале Generics in the .NET Framework (Универсальные типы в .NET Framework) по адресу *http://msdn.microsoft.com/en-us/library/ms172192.aspx*.

Листинг 1-4 Реализация SearchByActivity

```
public PagingResult<Itinerary> SearchByActivity(string activityId, int pageSize,
int pageNumber)
{
    using (var ctx = new PlanMyNightEntities())
    {
        ctx.ContextOptions.ProxyCreationEnabled = false;

        var query = from itinerary in ctx.Itineraries.Include("Activities")
                    where itinerary.Activities.Any(t => t.ActivityId ==
                            activityId) && itinerary.IsPublic
                    orderby itinerary.Rating
                    select itinerary;

        return PageResults(query, pageNumber, pageSize);
    }
}
```

Примечание Разбиение результатов на страницы реализовано в методе *PageResults*:

```
private static PagingResult<Itinerary> PageResults(IQueryable<Itinerary> query,
int page, int pageSize)
{
    int rowCount = rowCount = query.Count();
    if (pageSize > 0)
```

```
    {
        query = query.Skip((page - 1) * pageSize) .Take(pageSize);
    }
    var result = new PagingResult<Itinerary>(query.ToArray())
    {
        PageSize = pageSize,
        CurrentPage = page,
        TotalItems = rowCount
    };
    return result;
}
```

В эту функцию передается *IQueryable<Itinerary>*, что позволяет добавить разбиение на страницы в состав базового запроса. Передача *IQueryable* вместо *IEnumerable* гарантирует, что T-SQL для запроса к хранилищу будет формироваться только при вызове *query.ToArray()*.

Метод *SearchByZipCode* аналогичен методу *SearchByActivity*, но также добавляет фильтр по почтовому индексу места выполнения действия. Опять же, поддержка LINQ упрощает реализацию, как показано в листинге 1-5. Добавим выделенный код в тело метода *SearchByZipCode*.

Листинг 1-5 Реализация SearchByZipCode

```
public PagingResult<Itinerary> SearchByZipCode(int activityTypeId, string zip, int
pageSize, int pageNumber)
{
    using (var ctx = new PlanMyNightEntities())
    {
        ctx.ContextOptions.ProxyCreationEnabled = false;

        var query = from itinerary in ctx.Itineraries.Include("Activities")
                    where itinerary.Activities.Any(t => t.TypeId == activityTypeId
&& t.Zip == zip)
                          && itinerary.IsPublic
                    orderby itinerary.Rating
                    select itinerary;

        return PageResults(query, pageNumber, pageSize);
    }
}
```

Функция *SearchByRadius* вызывает импортированную функцию *RetrieveItinerariesWithinArea*, которая была сопоставлена с хранимой процедурой. После этого загружаются действия для каждого найденного маршрута. Вы можете скопировать выделенный код листинга 1-6 в тело метода *SearchByRadius* в файле ItinerariesRepository.cs.

Листинг 1-6 Реализация SearchByRadius

```
public PagingResult<Itinerary> SearchByRadius(int activityTypeId, double longitude,
double latitude, double radius, int pageSize, int pageNumber)
{
    using (var ctx = new PlanMyNightEntities())
    {
        ctx.ContextOptions.ProxyCreationEnabled = false;

        // Хранимая процедура с выходным параметром
        var totalOutput = new ObjectParameter("total", typeof(int));
        var items = ctx.RetrieveItinerariesWithinArea(activityTypeId,
            latitude, longitude, radius, pageSize, pageNumber,
            totalOutput).ToArray();

        foreach (var item in items)
        {
         item.Activities.ToList().AddRange(this.Retrieve(item.Id).Activities);
        }

        int total = totalOutput.Value == DBNull.Value ? 0 :
                  (int)totalOutput.Value;
```

```
        return new PagingResult<Itinerary>(items)
        {
            TotalItems = total,
            PageSize = pageSize,
            CurrentPage = pageNumber
        };
    }
}
```

С помощью метода *Add* планы мероприятий добавляются в хранилище данных. Реализация этой функциональности упростилась, поскольку наш контракт и объект контекста используют один и тот же объект сущности. Вставьте выделенный код листинга 1-7 в тело метода *Add*.

Листинг 1-7 Реализация Add

```
public void Add(Itinerary itinerary)
{
    using (var ctx = new PlanMyNightEntities())
    {
        ctx.Itineraries.AddObject(itinerary);
        ctx.SaveChanges();
    }
}
```

Готово! Мы завершили реализацию *ItinerariesRepository*, применив объект контекста, сформированный дизайнером EF. Выполните все тесты решения, нажав CTRL+R, A. Все тесты, касающиеся *ItinerariesRepository*, должны быть пройдены успешно.

Получение данных с Веб-сервисов Bing Maps

Поиск мероприятий для построения собственных планов мероприятий пользователя приложение PMN обеспечивает через применение Веб-сервисов Bing Maps. Чтобы получить ключ Bing Maps для использования в приложении PMN, необходимо создать учетную запись разработчика Bing Maps. Создать бесплатную учетную запись разработчика можно в центре управления учетными записями Bing Maps (Bing Maps Account Center) по адресу *https://www.bingmapsportal.com/*.

*Дополнительные сведения Веб-сервисы Microsoft Bing Maps – это набор программируемых сервисов, работающих по протоколу Simple Object Access Protocol (SOAP)[1], которые обеспечивают возможность находить местоположение на карте по заданному адресу, выполнять поиск местоположения по координатам, интегрировать карты и изображения, возвращают направления движения и позволяют включать в Веб-приложение всевозможную логику операций с определением местоположения. Более подробные сведения об этих сервисах можно найти на сайте, посвященном Bing Maps Web Services SDK[2] (*http://msdn.microsoft.com/en-us/library/cc980922.aspx*).*

Visual Studio 2003 В Visual Studio 2003 чтобы добавить ссылку на Веб-сервис, необходимо было выбрать в контекстном меню опцию Add Web Service Reference (Добавить ссылку на Веб-сервис). Это обеспечило бы открытие диалогового окна Add Web Reference (Добавить Веб-ссылку), в котором можно добавить ссылку на Веб-сервис в проект (рис. 1-29.).

[1] Простой протокол доступа к объектам (*прим. переводчика*).

[2] Пакет средств разработки Веб-сервисов Bing Maps (*прим. переводчика*).

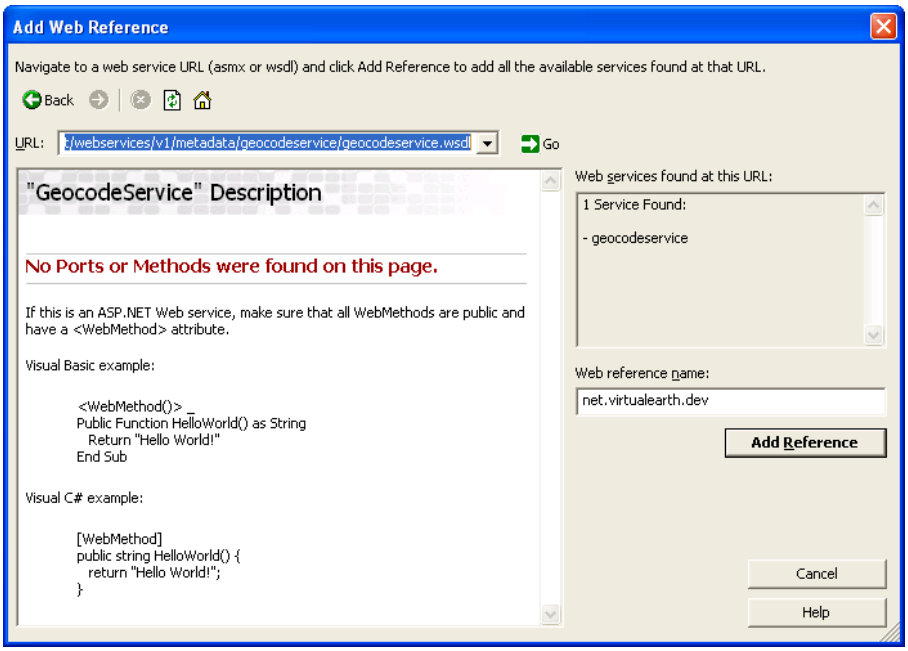


Рис. 1-29 Диалоговое окно Add Web Reference в Visual Studio 2003

Появившиеся в .NET Framework 3.0 сервисы Windows Communication Foundation (WCF) привели Веб-сервисы ASMX и другие технологии связи к одной унифицированной модели программирования.

Visual Studio 2010 обеспечивает инструментарий для работы с сервисами WCF. Чтобы вывести на экран новое диалоговое окно Add Service Reference (Добавить ссылку на сервис), щелкните правой кнопкой узел проекта и выберите опцию Add Service Reference, как показано на рис. 1-30. В этом диалоговом окне прежде всего необходимо задать адрес метаданных сервиса в поле Address (Адрес) и затем щелкнуть Go (Перейти) для просмотра доступных конечных точек сервисов. После этого можно задать пространство имен для автоматически формируемого кода и щелкнуть OK, чтобы добавить прокси в проект.

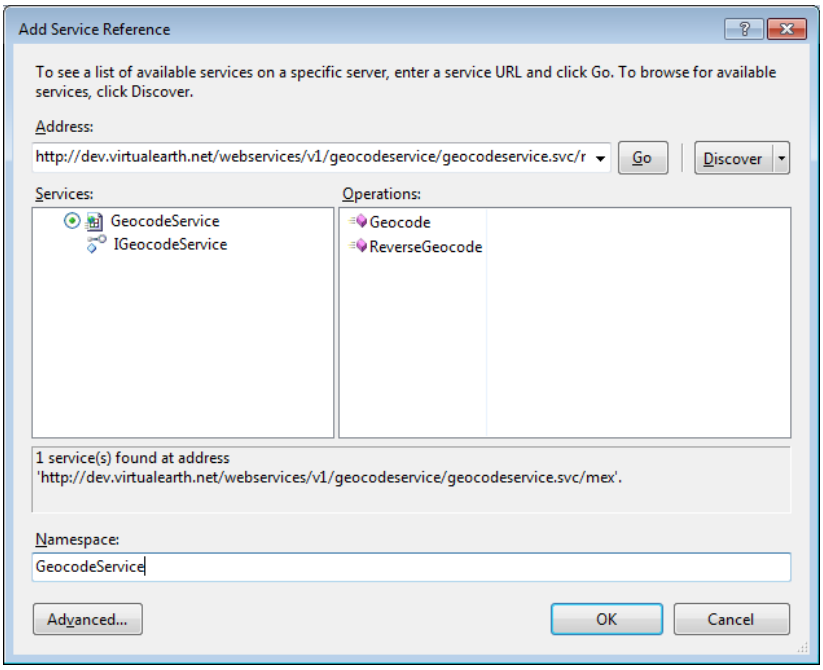


Рис. 1-30 Диалоговое окно Add Service Reference

Подсказка Для просмотра сервисов WCF, включенных в текущее решение, щелкните кнопку Discover (Просмотр).

Дополнительные сведения Чтобы открыть диалоговое око Service Reference Settings (Настройки ссылок на сервисы), щелкните кнопку Advanced (Дополнительно). Это диалоговое окно позволяет настроить конфигурацию прокси WCF-сервиса. Щелкнув кнопку Add Web Service, можно добавить ссылку в стиле .NET Framework 2.0. Более подробно эти настройки рассматриваются на сайте MSDN в разделе

Configure Service Reference Dialog Box (Настройка диалогового окна для добавления ссылки на сервис) по адресу http://msdn.microsoft.com/en-us/library/bb514724(VS.100).aspx.

Автоматически сформированный прокси WCF может использоваться точно так же, как и прокси ASMX, что продемонстрировано в листинге 1-8.

Листинг 1-8 Использование прокси Веб-сервиса

```
public BingCoordinate GeocodeAddress(ActivityAddress address, string token)
{
    ...
    Microsoft.Samples.PlanMyNight.Bing.VEGeocodingService.GeocodeResponse
geocodeResponse = null;
    // Выполнение запроса к сервису геокода
    using (var geocodeService = new
Microsoft.Samples.PlanMyNight.Bing.VEGeocodingService.GeocodeServiceClient())
    {
        try
        {
            geocodeResponse = geocodeService.Geocode(geocodeRequest);
            geocodeService.Close();
        }
        catch
        {
            geocodeService.Abort();
        }
    }

    if (geocodeResponse != null && geocodeResponse.Results != null &&
geocodeResponse.Results.Length > 0)
    {
        var location = geocodeResponse.Results[0].Locations[0];
        return new BingCoordinate { Latitude = (float)location.Latitude, Longitude
= (float)location.Longitude };
    }

    return default(BingCoordinate);
}
```

Разработка многопоточных приложений

С развитием многопроцессорной и многоядерной обработки данных все большее и большее значение приобретает обеспечение разработчикам возможностей для создания многопоточных приложений. Visual Studio 2010 и .NET Framework 4.0 предлагают новые пути реализации многопоточности в приложениях. Библиотека Task Parallel Library (TPL)[1] стала частью библиотеки Base Class Library (BCL)[2] для .NET Framework. Это означает, что теперь любое .NET-приложение может использовать TPL без всяких ссылок на сборки.

Для каждого ItineraryActivity PMN сохраняет в базе данных только идентификатор действия в Bing (Bing Activity ID). Когда приходит время извлечь весь объект Bing Activity (Действие в Bing), для заполнения сущности Bing Activity данными Веб-сервиса Bing Maps используется функция, которая выбирает все ItineraryActivity текущего Itinerary.

Один из способов выполнения этой операции – последовательное обращение к сервису для каждого действия Itinerary, как показано в листинге 1-9. Эта функция, прежде чем выполнять последующий вызов *RetrieveActivity* (Извлечь мероприятие), будет ожидать завершения предыдущего вызова, что делает ее выполнение линейным по времени.

Листинг 1-9 Последовательное извлечение мероприятий

```
public void PopulateItineraryActivities(Itinerary itinerary)
{
    foreach (var item in itinerary.Activities.Where(i =>i.Activity == null))
    {
        item.Activity = this.RetrieveActivity(item.ActivityId);
    }
}
```

[1] Библиотека для параллельного выполнения задач (*прим. переводчика*).

[2] Библиотека базовых классов (*прим. переводчика*).

В прошлом для организации многопоточной обработки этой задачи потребовалось бы использовать потоки и выполнить огромный объем работы. Теперь, с появлением TPL, для этого достаточно использовать статический метод *Parallel.ForEach*, который позаботиться обо всем, что связано с многопоточной обработкой, как показано в листинге 1-10.

Листинг 1-10 Параллельное извлечение мероприятий

```
public void PopulateItineraryActivities(Itinerary itinerary)
{
    Parallel.ForEach(itinerary.Activities.Where(i => i.Activity == null),
        item =>
        {
            item.Activity = this.RetrieveActivity(item.ActivityId);
        });
}
```

Дополнительные сведения .NET Framework 4.0 теперь включает библиотеки Parallel Linq (в System.Core.dll). PLinq представляет расширение .AsParallel для реализации многопоточной обработки в запросах LINQ. Благодаря расширениям .AsOrdered не составляет труда инициировать обработку источника данных, как будто данные были изначально отсортированы. Также в пространство имен System.Collections.Concurrent добавлены новые потокобезопасные коллекции. Более подробные сведения об этих новых возможностях можно найти в разделе Parallel Computing (Многопоточная обработка данных) на сайте MSDN по адресу http://msdn.microsoft.com/en-us/concurrency/default.aspx.

Кэширование AppFabric

PMN – это управляемое данными приложение, которое получает данные из базы данных приложения и от Веб-сервисов Bing Maps. Одна из возможных сложностей при создании Веб-приложения – реализация поддержки большого числа пользователей с обеспечением необходимой производительности и времени отклика. Обращения к хранилищу данных и к сервисам для поиска действий могут значительно повысить использование ресурсов сервера для элементов, совместно используемых множеством пользователей. Например, многие пользователи имеют доступ к общедоступным планам мероприятий. Просмотр этих планов мероприятий обусловит многочисленные обращения к одним и тем же элементам в базе данных. Реализация кэширования на Веб-уровне приведет к сокращению использования ресурсов хранилища данных и снизит задержку при выполнении повторных обращений к Веб-сервисам Bing Maps. На рис. 1-31 представлена архитектура приложения, реализующего кэширование на уровне Веб-сервера.

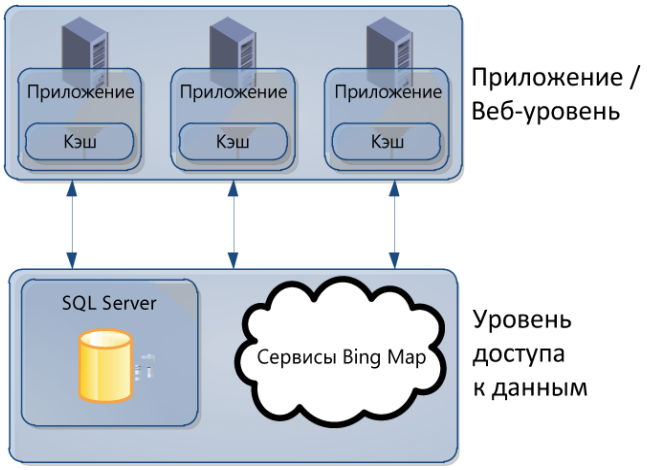


Рис. 1-31 Типовая архитектура Веб-приложения

Применение этого подхода сократит нагрузку на слой доступа к данным, но кэширование по-прежнему связано с конкретным сервером, обслуживающим запрос. Каждый сервер Веб-уровня будет иметь собственный кэш, и по-прежнему существует вероятность неравномерного распределения нагрузки между этими серверами.

Кэширование AppFabric Windows Server предлагает платформу распределенного кэширования в памяти. Клиентская библиотека AppFabric обеспечивает приложению возможность доступа к кэшу как к унифицированному событию просмотра, если кэш распределен на несколько компьютеров, как показано на рис. 1-32. API предоставляет простые методы *get* и *set*, что позволяет без труда извлекать и сохранять сериализуемые объекты общеязыковой среды выполнения (CLR). С кэшем AppFabric вы можете добавлять компьютер для размещения кэша по мере надобности, т.е. масштабирование становится прозрачным для клиента. Другое преимущество – кэш может также разделять копии данных между кластерами, обеспечивая высокую доступность данных даже в случае сбоев.

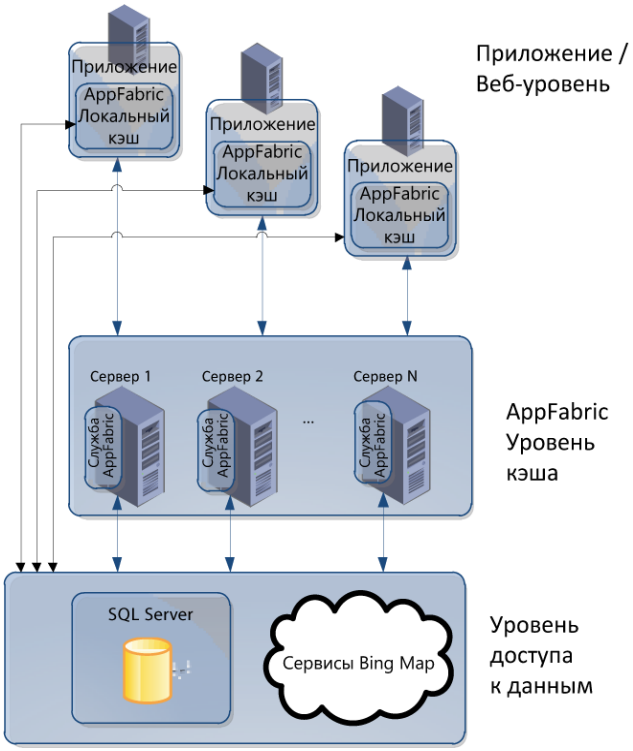


Рис. 1-32 Веб-приложение, использующее кэширование AppFabric Windows Server

*Дополнительные сведения Кэширование AppFabric Windows Server предлагается как набор расширений .NET Framework 4.0. Подробные сведения о том, как найти, установить и настроить Windows Server AppFabric, можно найти на сайте Windows Server AppFabric (*http://msdn.microsoft.com/en-us/windowsserver/ee695849.aspx*).*

Дополнительные сведения PMN может быть конфигурировано на использование либо кэширования ASP.NET, либо кэширования AppFabric Windows Server. Подробное пошаговое руководство с описанием того, как добавить кэширование AppFabric Windows Server в PMN предлагается в разделе PMN: Adding Caching using Velocity (Добавление кэширования с использованием Velocity) по адресу http://channel9.msdn.com/learn/courses/VS2010/ASPNET/EnhancingAspNetMvcPlanMyNight/Exercise-1-Adding-Caching-using-Velocity/.

Заключение

В данной главе мы рассмотрели ряд новых возможностей Visual Studio 2010 для структурирования слоя доступа к данным приложения PlanMyNight и использовали Entity Framework v4.0 для организации доступа к базе данных. Также были представлены автоматическое формирование сущностей с помощью POCO-шаблонов ADO.NET Entity Framework и расширения кэширования AppFabric Windows Server.

В следующей главе мы займемся созданием замечательных Веб-приложений с помощью инфраструктуры ASP.NET MVC и Managed Extensibility Framework (Инфраструктура расширения приложений).

Глава 2

От 2003 к 2010: проектирование внешнего вида и поведения

В данной главе рассматривается

- Создание контроллера ASP.NET MVC, взаимодействующего с моделью данных
- Создание представления ASP.NET MVC для отображения данных контроллера и проверки пользовательского ввода
- Расширение приложения внешним подключаемым модулем с помощью Managed Extensibility Framework

За годы, прошедшие с момента выхода ASP.NET 1.0, разработка Веб-приложений в Microsoft Visual Studio, безусловно, существенно улучшилась. Visual Studio 2003 включила такие возможности, как автоматическая проверка ввода и элементы управления для мобильных устройств (и многие другие), благодаря которым разработчики получили возможность создавать эффективные простые в обслуживании приложения.

Жажда улучшений, призванных помочь разработчикам в создании высококлассных приложений, не иссякла и в Visual Studio 2010. В этой главе будут рассмотрены некоторые новые возможности на примере расширения функциональности используемого нами для примера приложения Plan My Night.

> **Примечание** Приложение-пример – это проект ASP.NET MVC 2, но в Visual Studio 2010 у Веб-разработчика есть выбор использовать либо новую форму приложения ASP.NET, либо более традиционную ASP.NET (которую в сообществе разработчиков *для отличия называют Веб-формами* (Web Forms)). ASP.NET 4.0 была существенно дополнена, чтобы облегчить жизнь разработчиков и при этом обеспечить крайне эффективный подход к созданию Веб-приложений.

В данной главе будет использоваться модифицированный вариант решения приложения-примера. Если сопроводительные материалы данной книги установлены в каталог по умолчанию, интересующее нас решение находится в Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 2\ в папке UserInterface-Start.

Введение в проект PlanMyNight.Web

Пользовательский интерфейс Plan My Night в Visual Studio 2010 реализован как ASP.NET MVC-приложение, компоновка которого отличается от того, к чему, возможно, привыкли разработчики, создавая приложения Веб-форм ASP.NET в Visual Studio 2003. Некоторые элементы проекта (как видно на рис. 2-1) будут знакомыми (такие как Global.asax), но все остальные абсолютно новые. Некоторые из них являются обязательными для инфраструктуры ASP.NET MVC.

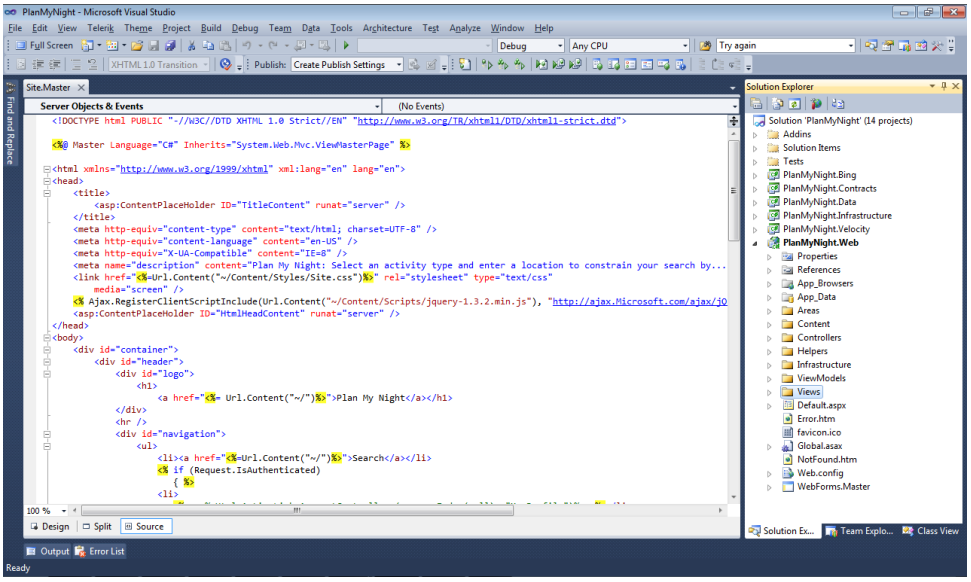


Рис. 2-1 Внешний вид проекта PlanMyNight.Web

Обязательные элементы ASP.NET MVC:

- **Areas (Области)** Эта папка используется инфраструктурой ASP.NET MVC для организации больших Веб-приложений в небольшие компоненты без разделения решений или проектов. Эта

возможность не используется в приложении Plan My Night, но представлена здесь, потому что данная папка создается шаблоном MVC-проекта.

- **Controllers (Контроллеры)** Во время обработки запроса инфраструктура ASP.NET MVC ищет контроллеры для обработки запроса в этой папке.
- **Views (Представления)** Папка Views на самом деле является структурой каталогов. Его подпапки носят имена соответствующие классам в папке Controllers. Также имеется подпапка Shared (Общие). Она предназначена для представлений, частичных представлений, главных страниц и любых других ресурсов, которые должны быть доступны всем котроллерам.

Дополнительные сведения Более подробные сведения о компонентах ASP.NET MVC, а также отличия обработки запросов от того, как это происходит в Веб-формах ASP.NET, представлены по адресу http://asp.net/mvc.

В большинстве случаев web.config является последним файлом в корневой папке проекта. В Visual Studio 2010 для этого файла предлагается существенное обновление: Преобразование Web.config. Эта возможность позволяет создавать на базе основного web.config файлы web.config для конкретных сборок, переопределяющие настройки базового файла во время сборки, развертывания и выполнения. Эти файлы отображаются под базовым web.config, как показано на рис. 2-2.

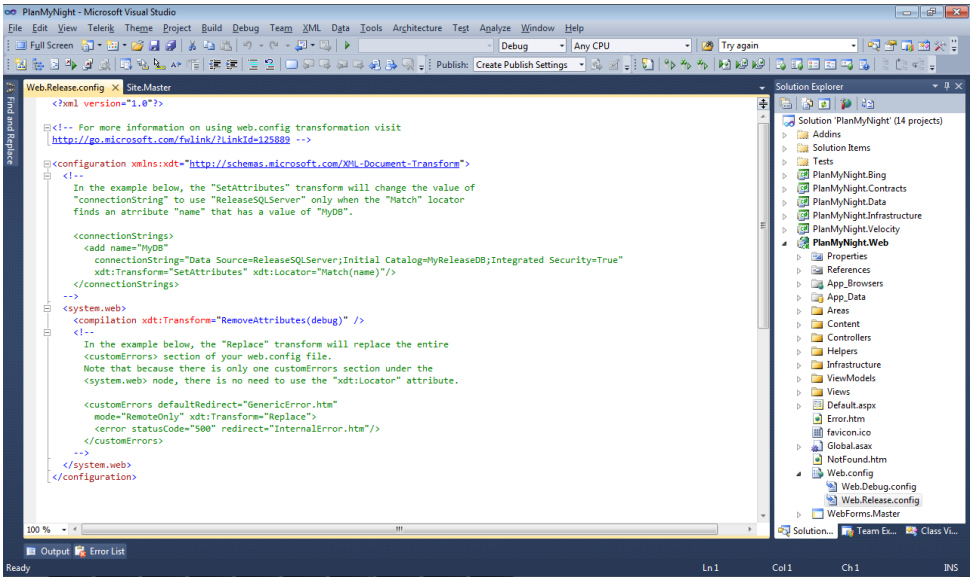


Рис. 2-2 Файл Web.config и специальные файлы конфигурации для конкретных сборок

> **Visual Studio 2003** В Visual Studio 2003 приходилось всегда помнить о том, что параметры в web.config не должны переопределяться параметрами отладки. Или нельзя было забывать вносить в web.config правильные настройки после его публикации для окончательной сборки. В Visual Studio 2010 таких проблем нет. Для переопределения значений в web.config при окончательных сборках используется набор параметров файла web.Release.config; при отладочных сборках используется файл web.Debug.config.

Проект также включает разделы:

- **Content (Содержимое)** Набор папок, содержащих изображения, сценарии и файлы стилей.
- **Helpers (Вспомогательные классы)** Различные классы, включающие ряд методов расширения, которые расширяют функциональность типов, используемых в проекте.
- **Infrastructure (Инфраструктура)** В данную папку включены элементы, обеспечивающие взаимодействие с более низкоуровневой инфраструктурой ASP.NET MVC (например, фабрики кэширования и контроллеров).
- **ViewModels (Модели представлений)** Объекты данных, заполненные классами контроллеров (Controller) и используемые представлениями (Views) для отображения данных.

Запуск проекта

Если скомпилировать и запустить проект, на экране должно быть выведено следующее (рис. 2-3).

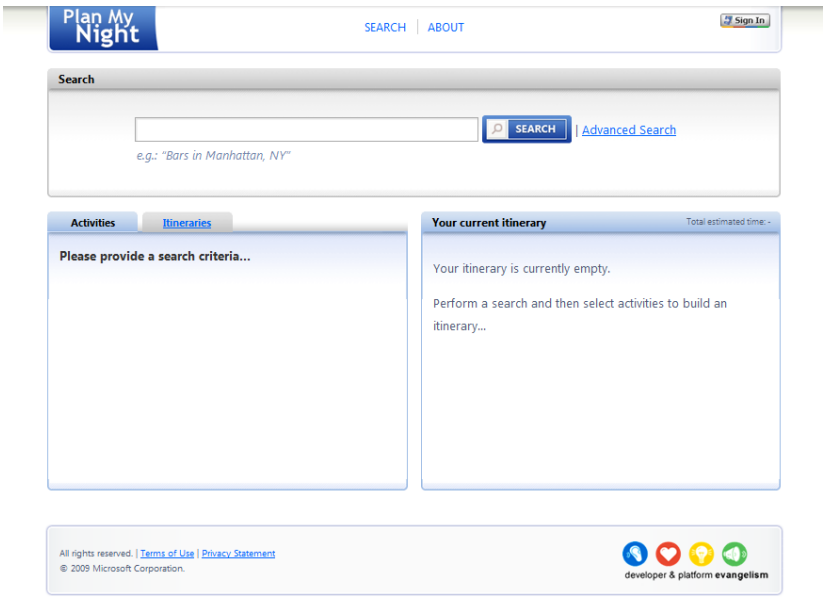


Рис. 2-3 Страница по умолчанию приложения Plan My Night

Поиск и организация исходного списка элементов плана мероприятий функционируют, но если попытаться сохранить разрабатываемый план мероприятий или выполнить вход с использованием Windows Live ID, приложение возвратит ошибку 404 Not Found (Не найден) (как показано на рис. 2-4).

Рис. 2-4 Ошибка, возвращаемая приложением Plan My Night при попытке регистрации

Это происходит потому, что на данный момент в проекте нет контроллера учетной записи для обработки таких запросов.

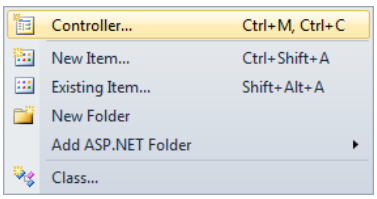
Создание контроллера учетной записи

Класс *AccountController* (Контроллер учетной записи) обеспечивает приложение Plan My Night критически важными функциями:

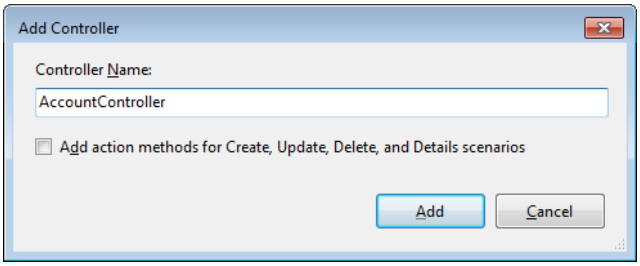
- Он обрабатывает вход и выход пользователей из приложения (с использованием Windows Live ID).
- Обеспечивает действия для отображения и обновления данных профиля пользователя.

Для создания нового контроллера ASP.NET MVC:

6. В Solution Explorer (Обозреватель решений) перейдите к папке Controllers проекта PlanMyNight.Web и щелкните ее правой кнопкой мыши.
7. Откройте подменю Add и выберите пункт Controller.



8. Введите имя контроллера, **AccountController**.



Примечание Не устанавливайте флажок Add Action Methods For Create, Update, And Delete Scenarios (Добавить методы для сценариев создания, обновления и удаления). Установка этого флажка обеспечит вставку некоторых методов-«заглушек», но поскольку мы не будем использовать методы по умолчанию, в их создании нет необходимости

По щелчку кнопки Add в диалоговом окне Add Controller откроется базовый класс *AccountController* с единственным методом *Index ()*:

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

namespace Microsoft.Samples.PlanMyNight.Web.Controllers
{

    public class AccountController : Controller
    {
        //
        // GET: /Account/

        public ActionResult Index()
        {
            return View();
        }

    }
}
```

Visual Studio 2003 Следует отметить отличие от разработки приложений ASP.NET Web Forms в Visual Studio 2003: в приложениях ASP.NET MVC для файлов .aspx не создаются файлы выделенного кода. Котроллеры, созданием которого мы только что занимались, осуществляют логику, необходимую для обработки ввода и подготовки вывода. Такой подход обеспечивает четкое разделение логики отображения и бизнес-логики, и это является ключевым аспектом ASP.NET MVC.

Реализация функциональности

Для взаимодействия с любым слоем доступа к данным и сервисами (Моделью) потребуется создать и инициализировать некоторые поля экземпляров. Но перед этим следует добавить ряд пространств имен в блок директив *using*:

```
using System.IO;
using Microsoft.Samples.PlanMyNight.Data;
using Microsoft.Samples.PlanMyNight.Entities;
using Microsoft.Samples.PlanMyNight.Infrastructure;
using Microsoft.Samples.PlanMyNight.Infrastructure.Mvc;
using Microsoft.Samples.PlanMyNight.Web.ViewModels;
using System.Collections.Specialized;
using WindowsLiveId;
```

Вот теперь добавим поля экземпляров. Эти поля являются интерфейсами различных разделов Модели:

```
public class AccountController : Controller
{
    private readonly IWindowsLiveLogin windowsLogin;
    private readonly IMembershipService membershipService;
    private readonly IFormsAuthentication formsAuthentication;
```

```
    private readonly IReferenceRepository referenceRepository;
    private readonly IActivitiesRepository activitiesRepository;
.
.
.
```

Примечание Использование интерфейсов для взаимодействия со всеми внешними зависимостями обеспечивает лучшую портируемость кода на различные платформы. Также интерфейсы более эффективно изолируют отдельные компоненты, что облегчает моделирование зависимостей при тестировании

Как уже упоминалось, эти поля представляют части Модели, с которой будет взаимодействовать данный контроллер для реализации функциональных требований. Рассмотрим общие описания каждого из этих интерфейсов:

- **IWindowsLiveLogin (Регистрация WindowsLive)** Обеспечивает функциональность для взаимодействия с сервисом Windows Live ID.
- **IMembershipService (Сервис членства)** Обеспечивает данные профиля пользователя и методы авторизации. В используемом приложении-примере это абстракция ASP.NET Membership Service (Сервис членства).
- **IFormsAuthentication (Аутентификация с помощью форм)** Абстракция ASP.NET Forms Authentication (Аутентификация с помощью форм).
- **IReferenceRepository (Хранилище справочных ресурсов)** Обеспечивает справочные ресурсы, такие как списки состояний и другие характерные для модели данные.
- **IActivitiesRepository (Хранилище действий)** Интерфейс для извлечения и обновления данных действий.

В этот класс добавим два конструктора: один – общий для времени выполнения, который использует класс *ServiceFactory* (Фабрика сервисов) для получения ссылок на необходимые интерфейсы, и другой – для обеспечения возможности вводить конкретные экземпляры интерфейсов при тестировании.

```
public AccountController() :
    this(
        new ServiceFactory().GetMembershipService(),
        new WindowsLiveLogin(true),
        new FormsAuthenticationService(),
        new ServiceFactory().GetReferenceRepositoryInstance(),
        new ServiceFactory().GetActivitiesRepositoryInstance())
{
}
public AccountController(
    IMembershipService membershipService,
    IWindowsLiveLogin windowsLogin,
    IFormsAuthentication formsAuthentication,
    IReferenceRepository referenceRepository,
    IActivitiesRepository activitiesRepository)
{
    this.membershipService = membershipService;
    this.windowsLogin = windowsLogin;
    this.formsAuthentication = formsAuthentication;
    this.referenceRepository = referenceRepository;
    this.activitiesRepository = activitiesRepository;
}
```

Аутентификация пользователя

Первой настоящей функциональностью, реализованной нами в этом контроллере, будет вход и выход из приложения. Большинство методов, которые будут реализовываться в дальнейшем, требуют аутентификации, поэтому именно с нее мы и начнем.

В нашем приложении для обеспечения аутентификации используется одновременно несколько технологий: Windows Live ID, ASP.NET Forms Authentication и ASP.NET Membership Services. Эти три технологии применяются в действии LiveID, которое будет реализовано следующим.

Начнем с создания в классе *AccountController* следующего метода:

```
public ActionResult LiveId()
{
    return Redirect("~/");
}
```

Это основной метод для взаимодействия с сервисами Windows Live ID. Если вызвать его прямо сейчас, он просто перенаправит пользователя к корню приложения.

> **Примечание** Вызов *Redirect* (Перенаправить) возвращает *RedirectResult* (Результат перенаправления). В данном примере для определения цели перенаправления используется строка, но в разных ситуациях могут использоваться различные перегрузки этого метода

Когда Windows Live ID возвращает пользователя в приложение, может быть предпринято несколько разных типов действий. Пользователь может выполнить вход в Windows Live ID, выполнить выход или очистить «cookies» Windows Live ID. Когда Windows Live ID возвращает пользователя, в его URL присутствует строковый параметр запроса *action*, поэтому переключение ветвей логики выполняется на основании значения этого параметра.

Добавим следующий код в метод *LiveId* над выражением *return*:

```
string action = Request.QueryString["action"];
switch (action)
{
    case "logout":
        this.formsAuthentication.SignOut();
        return Redirect("~/");

    case "clearcookie":
        this.formsAuthentication.SignOut();
        string type;
        byte[] content;
        this.windowsLogin.GetClearCookieResponse(out type, out content);
        return new FileStreamResult(new MemoryStream(content), type);
}
```

Дополнительные сведения Полную документацию по системе Windows Live ID можно найти по адресу http://dev.live.com/.

В только что добавленном коде обрабатываются два действия выхода для Windows Live ID. В обоих случаях используется интерфейс *IFormsAuthentication* для удаления файла «cookie» ASP.NET Forms Authentication, чтобы все будущие http-запросы (до момента повторной регистрации пользователя) не считались аутентифицированными. Во втором случае выполняется на одну операцию больше, и очищаются файлы «cookies» Windows Live ID (те, в которых сохраняется регистрационное имя, но не пароль).

Сценарий входа включает немного больший объем кода, поскольку требуется проверка присутствия аутентифицируемого пользователя в базе данных сервиса членства (Membership Database). Если этого пользователя там нет, для него создается новый профиль. Однако перед этим должны быть переданы данные, которые Windows Live ID переслал в интерфейс Windows Live ID, что позволит проверить их и предоставить объект *WindowsLiveLogin.User*:

```
default:
    // вход
    NameValueCollection tokenContext;
    if ((Request.HttpMethod ?? "GET").ToUpperInvariant() == "POST")
    {
        tokenContext = Request.Form;
    }
    else
    {
        tokenContext = new NameValueCollection(Request.QueryString);
        tokenContext["stoken"] =
            System.Web.HttpUtility.UrlEncode(tokenContext["stoken"]);
    }

    var liveIdUser = this.windowsLogin.ProcessLogin(tokenContext);
```

На данном этапе, если пользователь выполнит вход, *liveIdUser* будет либо ссылаться на аутентифицированный объект *WindowsLiveLogin.User*, либо будет null. Исходя из этого, добавим следующий блок кода, который обеспечивает выполнение некоторых действий, если значение *liveIdUser* не null:

```
    if (liveIdUser != null)
    {
        var returnUrl = liveIdUser.Context;
        var userId = new Guid(liveIdUser.Id).ToString();
        if (!this.membershipService.ValidateUser(userId, userId))
        {
```

```
                this.formsAuthentication.SignIn(userId, false);
                this.membershipService.CreateUser(userId, userId, string.Empty);
                var profile = this.membershipService.CreateProfile(userId);
                profile.FullName = "New User";
                profile.State = string.Empty;
                profile.City = string.Empty;
                profile.PreferredActivityTypeId = 0;
                this.membershipService.UpdateProfile(profile);

                if (string.IsNullOrEmpty(returnUrl)) returnUrl = null;
                return RedirectToAction("Index", new { returnUrl = returnUrl });
            }
            else
            {
                this.formsAuthentication.SignIn(userId, false);
                if (string.IsNullOrEmpty(returnUrl)) returnUrl = "~/";
                return Redirect(returnUrl);
            }
        }
    break;
```

Вызов метода *ValidateUser* (Проверить пользователя) в *IMembershipService* позволяет приложению проверить, посещал ли данный пользователь этот сайт ранее и существует ли его профиль. Поскольку аутентификация пользователя выполняется по Windows Live ID, значение его ID (которое является GUID) используется и как имя пользователя, и как пароль для ASP.NET Membership Service.

Если в приложении нет записи для данного пользователя, она создается путем вызова метода *CreateUser* (Добавить пользователя). Затем посредством *CreateProfile* (Создать профиль) создается и профиль параметров пользователя. Профиль заполняется некоторыми значениями по умолчанию и сохраняется в хранилище. Пользователь перенаправляется на основную страницу ввода, где может обновить сведения.

> **Примечание** На основании сочетания входных параметров *Controller.RedirectToAction* определяет, какой URL должен быть создан. В данном случае требуется перенаправить пользователя к действию *Index* (Корень) этого контроллера и передать текущее значение URL-адреса возврата

Также в данном коде реализована регистрация пользователя в сервисе аутентификации ASP.NET Forms, т.е. обеспечивается создание файла «cookie» с идентификационными данными для использования в запросах, требующих аутентификации, в будущем.

Параметры профиля также управляются сервисами ASP.NET Membership Services и объявляются в файле web.config приложения:

```
<system.web>
…
<profile enabled="true">
    <properties>
        <add name="FullName" type="string" />
        <add name="State" type="string" />
        <add name="City" type="string" />
        <add name="PreferredActivityTypeId" type="int" />
    </properties>

    <providers>
        <clear />
        <add name="AspNetSqlProfileProvider"
type="System.Web.Profile.SqlProfileProvider,
             System.Web, Version=4.0.0.0, Culture=neutral,
             PublicKeyToken=b03f5f7f11d50a3a"
        connectionStringName="ApplicationServices"
        applicationName="/" />
    </providers>
</profile>
…
</system.web>
```

Теперь метод *LiveID* полностью готов и должен выглядеть, как следующий фрагмент кода. Приложение принимает сведения для аутентификации от Windows Live ID, подготавливает профиль ASP.NET Membership Service и создает маркер аутентификации ASP.NET Forms.

```
public ActionResult LiveId()
{
string action = Request.QueryString["action"];
```

```
switch (action)
{
    case "logout":
        this.formsAuthentication.SignOut();
        return Redirect("~/");

    case "clearcookie":
        this.formsAuthentication.SignOut();
        string type;
        byte[] content;
        this.windowsLogin.GetClearCookieResponse(out type, out content);
        return new FileStreamResult(new MemoryStream(content), type);

    default:
        // вход
        NameValueCollection tokenContext;
        if ((Request.HttpMethod ?? "GET").ToUpperInvariant() == "POST")
        {
            tokenContext = Request.Form;
         }
        else
        {
            tokenContext = new NameValueCollection(Request.QueryString);
            tokenContext["stoken"] =
                System.Web.HttpUtility.UrlEncode(tokenContext["stoken"]);
        }
        var liveIdUser = this.windowsLogin.ProcessLogin(tokenContext);

        if (liveIdUser != null)
        {
            var returnUrl = liveIdUser.Context;
            var userId = new Guid(liveIdUser.Id).ToString();
            if (!this.membershipService.ValidateUser(userId, userId))
            {
                this.formsAuthentication.SignIn(userId, false);
                this.membershipService.CreateUser(userId, userId,
                                                  string.Empty);
                var profile = this.membershipService.CreateProfile(userId);
                profile.FullName = "New User";
                profile.State = string.Empty;
                profile.City = string.Empty;
                profile.PreferredActivityTypeId = 0;
                this.membershipService.UpdateProfile(profile);

                if (string.IsNullOrEmpty(returnUrl)) returnUrl = null;
                return RedirectToAction("Index",
                                        new { returnUrl = returnUrl });
            }
            else
            {
                this.formsAuthentication.SignIn(userId, false);
                if (string.IsNullOrEmpty(returnUrl)) returnUrl = "~/";
                return Redirect(returnUrl);
            }
        }
    break;
}
return Redirect("~/");
}
```

Конечно, пользователь должен иметь возможность перед регистрацией попасть на страницу входа Windows Live ID. На данный момент в приложении Plan My Night имеется кнопка входа с помощью Windows Live ID. Но возможны ситуации, когда понадобится перенаправить пользователя на страницу входа прямо из кода. Для реализации такого сценария добавим в контроллер небольшой метод под именем *Login* (Вход):

```
public ActionResult Login(string returnUrl)
{
    var redirect = HttpContext.Request.Browser.IsMobileDevice ?
                   this.windowsLogin.GetMobileLoginUrl(returnUrl) :
                   this.windowsLogin.GetLoginUrl(returnUrl);
    return Redirect(redirect);
```

```
}
```

Данный метод просто извлекает URL-адрес входа для Windows Live и перенаправляет пользователя по этому адресу. Это также удовлетворяет конфигурации для аутентификации ASP.NET Forms в web.config с той точки зрения, что любой запрос, требующий аутентификации, будет перенаправлен в этот метод:

```
<authentication mode="Forms">
    <forms loginUrl="~/Account/Login" name="XAUTH" timeout="2880" path="~/" />
</authentication>
```

Извлечение профиля для текущего пользователя

Теперь, когда методы аутентификации описаны, что реализует первую цель данного контролера – обработка входа и выхода пользователей из приложения – можно переходить к извлечению данных текущего пользователя.

Метод *Index* является методом по умолчанию для контроллера на основании конфигурации отображения URL в Global.asax. Он будет располагаться там, где должны извлекаться данные текущего пользователя и возвращаться представление, отображающее эти данные. Метод *Index*, изначально описывавшийся при создании класса *AccountController*, необходимо заменить следующим:

```
 [Authorize()]
[AcceptVerbs(HttpVerbs.Get)]
public ActionResult Index(string returnUrl)
{
        var profile = this.membershipService.GetCurrentProfile();
        var model = new ProfileViewModel
        {
            Profile = profile,
            ReturnUrl = returnUrl ?? this.GetReturnUrl()
        };

        this.InjectStatesAndActivityTypes(model);

        return View("Index", model);
}
```

> **Visual Studio 2003** Атрибуты, такие как *[Authorize()]*, не имели такого широкого распространения в Visual Studio 2003, но в ASP.NET MVC они часто используются. Атрибуты позволяют описывать метаданные цели, к которой они относятся. Благодаря этому появляется возможность проверять данные во время выполнения (через отражение) и предпринимать действия в случае необходимости.

Очень удобен атрибут *Authorize* (Авторизовать). Он показывает, что данный метод может вызываться только для уже аутентифицированных http-запросов. Если запрос не аутентифицирован, он будет перенаправлен в заданную операцию ASP.NET Forms Authentication, которая только что была настроена. Атрибут *AcceptVerbs* (Допустимые команды) также ограничивает способы вызова этого метода, задавая допустимые Http-команды. В данном случае этот метод вызывается по запросу HTTP GET. В сигнатуру метода добавлен строковый параметр *returnUrl*. Он обеспечивает возможность возвращения пользователя на исходный адрес по завершении просмотра или обновления его сведений.

> **Примечание** Это приводит нас к части инфраструктуры ASP.NET MVC под названием *Привязка модели* (*Model Binding*), подробное рассмотрение которой выходит за рамки данной книги. Однако необходимо знать, что она пытается найти источник *returnUrl* (поле формы, данные таблицы маршрутизации или параметр строки запроса с таким же именем) и выполняет его привязку к этому значению при вызове данного метода. Если средству привязки модели не удается найти подходящего источника, значение будет null. Такое поведение может обусловить проблемы для типов значений, которые не могут быть null, потому что приведет к формированию исключения *InvalidOperationException* (Недопустимая операция).

В целом, этот метод прост: он принимает возвращаемое значение метода *GetCurrentProfile* (Получить профиль текущего пользователя) интерфейса ASP.NET Membership Service и настраивает объект модели представления, который будет использоваться представлением. Вызов *GetReturnUrl* (Получить URL возврата) – это пример метода расширения, описанного в проекте PlanMyNight.Infrastructure. Он не является членом класса Controller, но в среде разработки обеспечивает намного более надежный код (рис. 2-5).

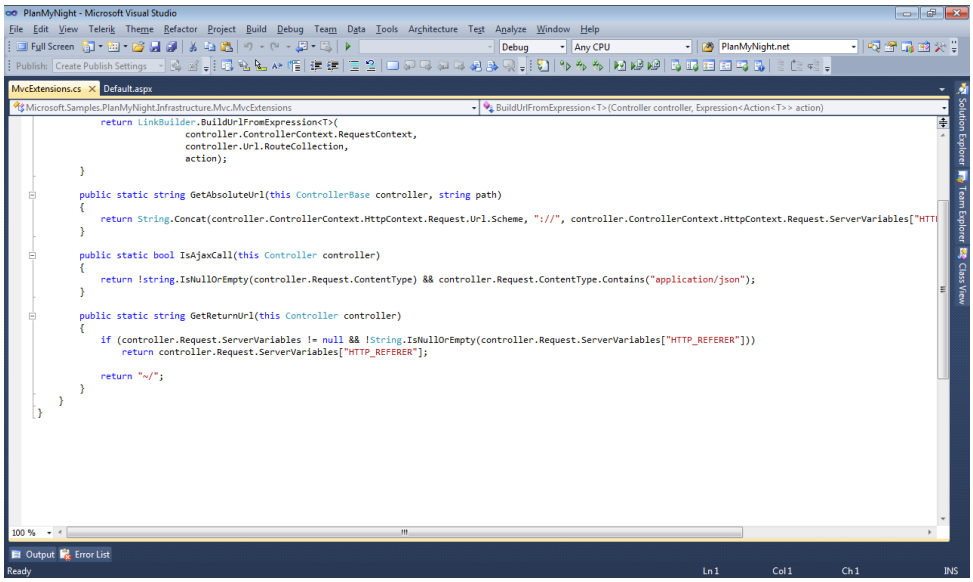


Рис. 2-5 Пример методов расширения в MvcExtensions.cs

> **Visual Studio 2003** В .NET Framework 1.1, используемой Visual Studio 2003, методов расширения не было. Сейчас мы просто вызываем *this.GetReturnUrl()*, и этот метод появляется в списке IntelliSense для этого объекта. Ранее же приходилось вводить **MvcExtensions.GetReturnUrl(this)**, передавая контроллер в качестве параметра. Безусловно, методы расширения делают код более удобным для восприятия, при этом от разработчика не требуется знания статического класса, в котором описан этот метод расширения. Для обеспечения работы IntelliSense это пространство имен просто должно быть указано в директивах *using*.

InjectStatesAndActivityTypes (Ввести состояния и типы действий) – метод, который требуется реализовать. Он собирает данные имен состояния из хранилища справочных ресурсов и данные о типах действий из хранилища действий. Он создает две коллекции *SelectListItem* (HTML-класс для MVC): одну для списка состояний, и другую для списка разных типов действий, доступных в приложении. Также он задает соответствующее значение.

```
private void InjectStatesAndActivityTypes(ProfileViewModel model)
{
    var profile = model.Profile;
    var types = this.activitiesRepository.RetrieveActivityTypes().Select(
                o => new SelectListItem {
                          Text = o.Name,
                          Value = o.Id.ToString(),
                          Selected = (profile != null && o.Id ==
                                      profile.PreferredActivityTypeId)
                }).ToList();

    types.Insert(0, new SelectListItem { Text = "Select...", Value = "0" });
    var states = this.referenceRepository.RetrieveStates().Select(
                 o => new SelectListItem {
                           Text = o.Name,
                           Value = o.Abbreviation,
                           Selected = (profile != null && o.Abbreviation ==
                                       profile.State)
                }).ToList();

    states.Insert(0, new SelectListItem {
                          Text = "Any state",
                          Value = string.Empty
                });

    model.PreferredActivityTypes = types;
    model.States = states;
}
```

Visual Studio 2003 В Visual Studio 2003 реализовать метод *InjectStatesAndActivities* сложнее, потому что разработчик не может использовать расширения LINQ (вызов *Select*) и лямбда выражения, которые являются формой анонимного делегата, применяемого методом *Select* к каждому перечисляемому члену коллекции. Вместо этого разработчику пришлось бы писать собственный цикл и выполнять перечисление всех элементов вручную.

Обновление данных профиля

Закончив с инфраструктурой для извлечения данных текущего профиля, перейдем к обновлению данных модели посредством формы, передаваемой пользователем. После этого можно создавать собственные страницы представления и видеть, как все это сочетается. Метод *Update* прост, однако, в нем появляются некоторые новые до сих пор не встречавшиеся нам возможности:

```
[Authorize()]
[AcceptVerbs(HttpVerbs.Post)]
[ValidateAntiForgeryToken()]
public ActionResult Update(UserProfile profile)
{
    var returnUrl = Request.Form["returnUrl"];
    if (!ModelState.IsValid)
    {
        // ошибка валидации
        return this.IsAjaxCall() ? new JsonResult { JsonRequestBehavior =
                    JsonRequestBehavior.AllowGet, Data = ModelState }
                    : this.Index(returnUrl);
    }

    this.membershipService.UpdateProfile(profile);
    if (this.IsAjaxCall())
    {
        return new JsonResult { JsonRequestBehavior =
                                JsonRequestBehavior.AllowGet,
                                Data = new { Update = true,
                                             Profile = profile,
                                             ReturnUrl = returnUrl } };
    }
    else
    {
        return RedirectToAction("UpdateSuccess", "Account", new { returnUrl =
                                 returnUrl });
    }
}
```

Атрибут *ValidateAntiForgeryToken* (Проверить маркер, препятствующий фальсификации) подтверждает то, что форма не была сфальсифицирована. Для использования этой возможности необходимо добавить *AntiForgeryToken* (Маркер, препятствующий фальсификации) в форму ввода представления. Проверка действительности *ModelState* (Состояние модели) – наш первый опыт валидации ввода. Это валидация на стороне сервера, и ASP.NET MVC предлагает простую в использовании возможность убедиться в том, что поступающие данные удовлетворяют некоторым правилам. Одно из свойств объекта *UserProfile*, созданного для обеспечения ввода в этот метод через привязку модели MVC, имеет атрибут *System.ComponentModel.DataAnnotations.Required*. При привязке модели инфраструктура MVC проверяет атрибуты *DataAnnotation* (Аннотация данных) и объявляет *ModelState* действительным только в случае выполнения всех правил.

В случае, когда *ModelState* недействительный, пользователь перенаправляется в метод *Index*, где *ModelState* используется для отображения формы ввода. Или, если запрос был AJAX-вызовом, возвращается *JsonResult* с прикрепленными к нему данными *ModelState*.

Visual Studio 2003 В ASP.NET MVC запросы маршрутизируются не через страницы, а через контроллеры. Поэтому один URL может обрабатывать множество разных запросов и отвечать на них соответствующим представлением. В Visual Studio 2003 для реализации этой функциональности разработчику пришлось бы создавать два разных URL и вызывать метод в третьем классе

Если *ModelState* действительный, сервис членства обновляет профиль, и для AJAX-запросов возвращается JSON-результат со сведениями об успешности операции. Для «обычных» запросов пользователь перенаправляется к действию *UpdateSuccess* (Успешное обновление) контроллера Account. Метод *UpdateSuccess* – последний метод, необходимый для завершения выполнения данного контроллера:

```
public ActionResult UpdateSuccess(string returnUrl)
{
```

```
    var model = new ProfileViewModel
    {
        Profile = this.membershipService.GetCurrentProfile(),
        ReturnUrl = returnUrl
    };
    return View(model);
}
```

Этот метод используется для возвращения представления успешного выполнения в браузер, отображения некоторых обновленных данных и обеспечения ссылки для возвращения пользователя туда, где он находился перед началом процесса обновления профиля.

Теперь, когда контроллер Account полностью реализован, полученный класс должен выглядеть следующим образом:

```
    using System;
    using System.Collections.Specialized;
    using System.IO;
    using System.Linq;
    using System.Web;
    using System.Web.Mvc;
    using Microsoft.Samples.PlanMyNight.Data;
    using Microsoft.Samples.PlanMyNight.Entities;
    using Microsoft.Samples.PlanMyNight.Infrastructure;
    using Microsoft.Samples.PlanMyNight.Infrastructure.Mvc;
    using Microsoft.Samples.PlanMyNight.Web.ViewModels;
    using WindowsLiveId;

namespace Microsoft.Samples.PlanMyNight.Web.Controllers
{
    [HandleErrorWithContentType()]
    [OutputCache(NoStore = true, Duration = 0, VaryByParam = "*")]
    public class AccountController : Controller
    {
        private readonly IWindowsLiveLogin windowsLogin;
        private readonly IMembershipService membershipService;
        private readonly IFormsAuthentication formsAuthentication;
        private readonly IReferenceRepository referenceRepository;
        private readonly IActivitiesRepository activitiesRepository;

        public AccountController() :
            this(
                new ServiceFactory().GetMembershipService(),
                new WindowsLiveLogin(true),
                new FormsAuthenticationService(),
                new ServiceFactory().GetReferenceRepositoryInstance(),
                new ServiceFactory().GetActivitiesRepositoryInstance())
        {
        }

        public AccountController(IMembershipService membershipService,
                                 IWindowsLiveLogin windowsLogin,
                                 IFormsAuthentication formsAuthentication,
                                 IReferenceRepository referenceRepository,
                                 IActivitiesRepository activitiesRepository)
        {
            this.membershipService = membershipService;
            this.windowsLogin = windowsLogin;
            this.formsAuthentication = formsAuthentication;
            this.referenceRepository = referenceRepository;
            this.activitiesRepository = activitiesRepository;
        }

        public ActionResult LiveId()
        {
            string action = Request.QueryString["action"];
            switch (action)
            {
                case "logout":
                    this.formsAuthentication.SignOut();
                    return Redirect("~/");
```

```
                case "clearcookie":
                    this.formsAuthentication.SignOut();
                    string type;
                    byte[] content;
                    this.windowsLogin.GetClearCookieResponse(out type, out
content);
                    return new FileStreamResult(new MemoryStream(content), type);
                default:
                    // вход
                    NameValueCollection tokenContext;
                    if ((Request.HttpMethod ?? "GET").ToUpperInvariant() == "POST")
                    {
                        tokenContext = Request.Form;
                    }
                    else
                    {
                        tokenContext = new
                     NameValueCollection(Request.QueryString);
                        tokenContext["stoken"] =
                     System.Web.HttpUtility.UrlEncode(tokenContext["stoken"]);
                    }

                    var liveIdUser =
                            this.windowsLogin.ProcessLogin(tokenContext);
                    if (liveIdUser != null)
                    {
                        var returnUrl = liveIdUser.Context;
                        var userId = new Guid(liveIdUser.Id).ToString();
                        if (!this.membershipService.ValidateUser(userId, userId))
                        {
                            this.formsAuthentication.SignIn(userId, false);
                            this.membershipService.CreateUser( userId, userId,
                                                               string.Empty);
                            var profile =
                                this.membershipService.CreateProfile(userId);
                            profile.FullName = "New User";
                            profile.State = string.Empty;
                            profile.City = string.Empty;
                            profile.PreferredActivityTypeId = 0;
                            this.membershipService.UpdateProfile(profile);
                            if (string.IsNullOrEmpty(returnUrl)) returnUrl =
                                                               null;
                            return RedirectToAction("Index", new { returnUrl =
                                                               returnUrl });
                        }
                        else
                        {
                            this.formsAuthentication.SignIn(userId, false);
                            if (string.IsNullOrEmpty(returnUrl)) returnUrl =
                                                               "~/";
                            return Redirect(returnUrl);
                        }
                    }
                    break;
                }
                return Redirect("~/");
            }

            public ActionResult Login(string returnUrl)
            {
                var redirect = HttpContext.Request.Browser.IsMobileDevice ?
                          this.windowsLogin.GetMobileLoginUrl(returnUrl) :
                          this.windowsLogin.GetLoginUrl(returnUrl);
                return Redirect(redirect);
            }

            [Authorize()]
            [AcceptVerbs(HttpVerbs.Get)]
            public ActionResult Index(string returnUrl)
            {
```

```
            var profile = this.membershipService.GetCurrentProfile();
            var model = new ProfileViewModel
            {
                Profile = profile,
                ReturnUrl = returnUrl ?? this.GetReturnUrl()
            };

            this.InjectStatesAndActivityTypes(model);
            return View("Index", model);
        }

        [Authorize()]
        [AcceptVerbs(HttpVerbs.Post)]
        [ValidateAntiForgeryToken()]
        public ActionResult Update(UserProfile profile)
        {
            var returnUrl = Request.Form["returnUrl"];
            if (!ModelState.IsValid)
            {
                // ошибка валидации
                return this.IsAjaxCall() ?
                new JsonResult { JsonRequestBehavior =
                JsonRequestBehavior.AllowGet, Data = ModelState }
                                    : this.Index(returnUrl);
            }
            this.membershipService.UpdateProfile(profile);
            if (this.IsAjaxCall())
            {
                return new JsonResult {
                       JsonRequestBehavior = JsonRequestBehavior.AllowGet,
                       Data = new {
                       Update = true,
                       Profile = profile,
                       ReturnUrl = returnUrl } };
            }
            else
            {
                return RedirectToAction("UpdateSuccess",
                        "Account", new { returnUrl = returnUrl });
            }
        }
        public ActionResult UpdateSuccess(string returnUrl)
        {
            var model = new ProfileViewModel
            {
                Profile = this.membershipService.GetCurrentProfile(),
                ReturnUrl = returnUrl
            };
            return View(model);
        }

        private void InjectStatesAndActivityTypes(ProfileViewModel model)
        {
            var profile = model.Profile;
            var types = this.activitiesRepository.RetrieveActivityTypes()
                        .Select(o => new SelectListItem { Text = o.Name,
                              Value = o.Id.ToString(),
                              Selected = (profile != null &&
                              o.Id == profile.PreferredActivityTypeId) })
                              .ToList();
            types.Insert(0, new SelectListItem { Text = "Select...",
                                                 Value = "0" });
            var states = this.referenceRepository.RetrieveStates().Select(
                              o => new SelectListItem {
                              Text = o.Name,
                              Value = o.Abbreviation,
                              Selected = (profile != null &&
                              o.Abbreviation == profile.State) })
                              .ToList();
            states.Insert(0, new SelectListItem { Text = "Any state",
```

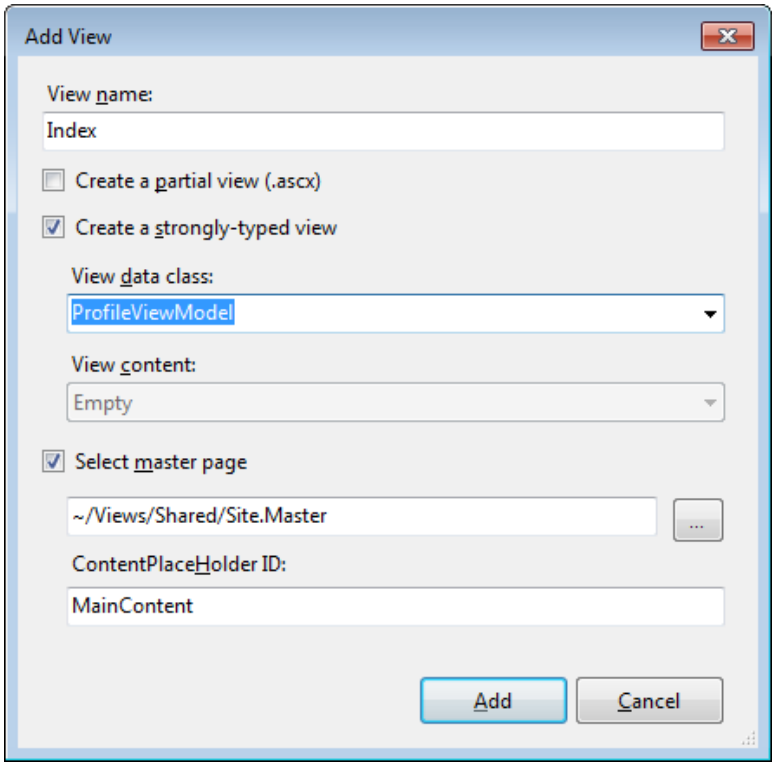
```
                                                    Value = string.Empty });
                model.PreferredActivityTypes = types;
                model.States = states;
            }
        }
    }
```

Создание представления учетной записи

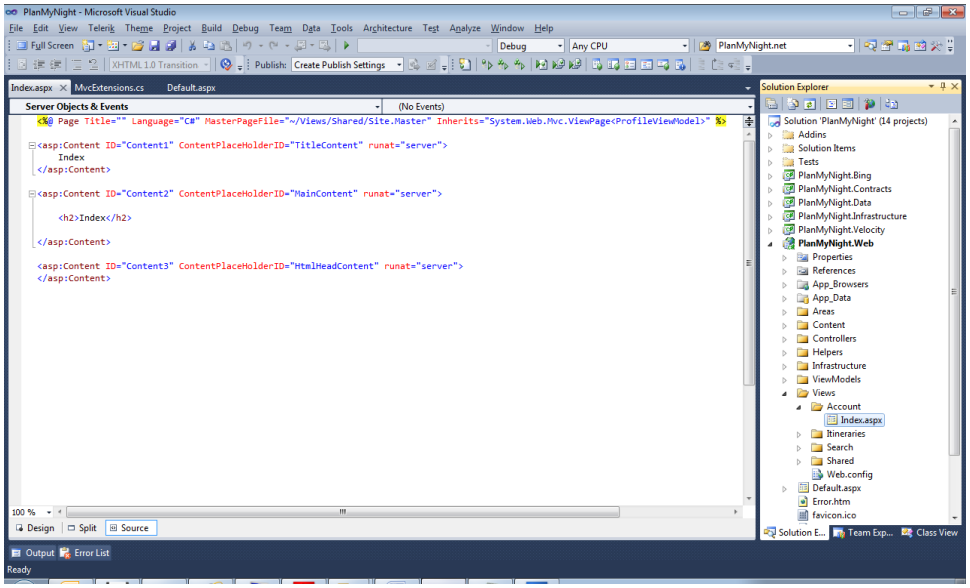
В предыдущем разделе был создан контроллер, реализующий функциональность для обновления и просмотра данных пользователя. В этом разделе мы поэтапно рассмотрим возможности Visual Studio 2010, позволяющие создавать представления, которые будут отображать эту функциональность пользователю.

Для создания представления Index контроллера Account:

1. Перейдем к папке Views проекта PlanMyNight.Web.
2. Щелкнем папку Views правой кнопкой мыши, раскроем подменю Add и выберем New Folder.
3. Назовем новую папку **Account**.
4. Щелкнем правой кнопкой мыши новую папку Account, раскроем подменю Add и выберем View.
5. Заполним поля диалогового окна Add View (Добавить представление), как показано на рисунке:



6. Щелкнем Add. Мы должны получить HTML-страницу с несколькими элементами управления *<asp:Content>*:



Можно заметить, здесь нет особых отличий от того, что мы привыкли видеть в Visual Studio 2003. По умолчанию ASP.NET MVC 2 использует механизм формирования представлений Веб-форм ASP.NET, поэтому страницы MVC и Веб-форм будут иметь некоторое сходство. Основное отличие на этом этапе в том, что класс *page* наследуется от *System.Web.Mvc.ViewPage<ProfileViewModel>*, и отсутствует файл выделенного кода. MVC не использует файлы выделенного кода, в отличие от Веб-форм ASP.NET, чтобы обеспечить четкое разделение функциональных областей. Редактирование страниц MVC, как правило, осуществляется через разметку. Конструктор применяется преимущественно для приложений Веб-форм ASP.NET.

Чтобы этот каркас страницы стал основным представлением контроллера Account, необходимо изменить содержимое заголовка, обеспечив его единообразие с остальными представлениями:

```
<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Plan My Night - Profile
</asp:Content>
```

Далее требуется добавить клиентские сценарии, которые будут использоваться в содержимом *HtmlHeadContent* (Содержимое HTML-заголовка):

```
<asp:Content ID="Content3" ContentPlaceHolderID="HtmlHeadContent" runat="server">

    <% Ajax.RegisterClientScriptInclude(Url.Content("~/Content/Scripts/jquery-
1.4.1.min.js")); %>
    <%
Ajax.RegisterCombinedScriptInclude(Url.Content("~/Content/Scripts/MicrosoftAjax.js"
), "pmn"); %>
    <%
Ajax.RegisterCombinedScriptInclude(Url.Content("~/Content/Scripts/MicrosoftAjaxMvc.
js"), "pmn"); %>
    <%
Ajax.RegisterCombinedScriptInclude(Url.Content("~/Content/Scripts/MicrosoftMvcValid
ation.js"), "pmn"); %>
    <% Ajax.RegisterCombinedScriptInclude(
        Url.Content("~/Content/Scripts/ajax.common.js"), "pmn"); %>
    <% Ajax.RegisterCombinedScriptInclude(
        Url.Content("~/Content/Scripts/ajax.profile.js"), "pmn"); %>
    <%= Ajax.RenderClientScripts() %>
</asp:Content>
```

Этот сценарий использует методы расширения *System.Web.Mvc.AjaxHelper*, описанные в проекте PlanMyNight.Infrastructure в папке MVC.

Когда содержимое заголовка настроено, можно заняться основным содержимым представления:

```
<asp:Content ContentPlaceHolderID="MainContent" runat="server">
<div class="panel" id="profileForm">
    <div class="innerPanel">
        <h2><span>My Profile</span></h2>
        <% Html.EnableClientValidation(); %>
        <% using (Html.BeginForm("Update", "Account")) %>
        <% { %>
```

```
        <%=Html.AntiForgeryToken()%>
        <div class="items">
            <fieldset>
                <p>
                    <label for="FullName">Name:</label>
                    <%=Html.EditorFor(m => m.Profile.FullName)%>
                    <%=Html.ValidationMessage("Profile.FullName",
                        new { @class = "field-validation-error-wrapper" })%>
                </p>
                <p>
                    <label for="State">State:</label>
                    <%=Html.DropDownListFor(m => m.Profile.State,
                                            Model.States)%>
                </p>
                <p>
                    <label for="City">City:</label>
                    <%=Html.EditorFor(m => m.Profile.City,
                                      Model.Profile.City)%>
                </p>
                <p>
                    <label for="PreferredActivityTypeId">Preferred
activity:</label>
                    <%=Html.DropDownListFor(m =>
                        m.Profile.PreferredActivityTypeId,
                        Model.PreferredActivityTypes)%>
                </p>
            </fieldset>
            <div class="submit">
                <%=Html.Hidden("returnUrl", Model.ReturnUrl)%>
                <%=Html.SubmitButton("submit", "Update")%>
            </div>
        </div>
        <div class="toolbox"></div>
        <% } %>
        </div>
</div>
</asp:Content>
```

Если не обращать внимания на некоторый встроенный код, все это выглядит как обычная HTML-разметка. Рассмотрим фрагменты встроенного кода и продемонстрируем, какую мощь они обеспечивают (и простоту при этом).

> **Visual Studio 2003** В Visual Studio 2003 более типичным для отображения данных было использовать серверные элементы управления и логику времени отображения. Но поскольку страницы представлений ASP.NET MVC не имеют файла выделенного кода, эта логика должна быть описана в одном файле с разметкой. По-прежнему могут применяться элементы управления ASP.NET Web Forms. В нашем примере используется элемент управления *<asp:Content>*. Однако, как правило, функциональность элементов управления ASP.NET Web Forms ограничена из-за отсутствия файла выделенного кода

В MVC широко применяются вспомогательные классы HTML. Методы, содержащиеся в *System.Web.Mvc.HtmlHelper*, генерируют компактные соответствующие стандартам HTML-теги для использования в различных целях. Для этого MVC-разработчику в некоторых случаях приходится писать больше кода разметки, чем разработчику Веб-форм, но он получает непосредственный контроль над формированием вывода. Строго типизированная версия этого класса расширения (*HtmlHelper<TModel>*) может использоваться в разметке представления через свойство *ViewPage<TModel>.Html*.

В данной форме используются такие *HTML*-методы (и это лишь часть того, что доступно по умолчанию):

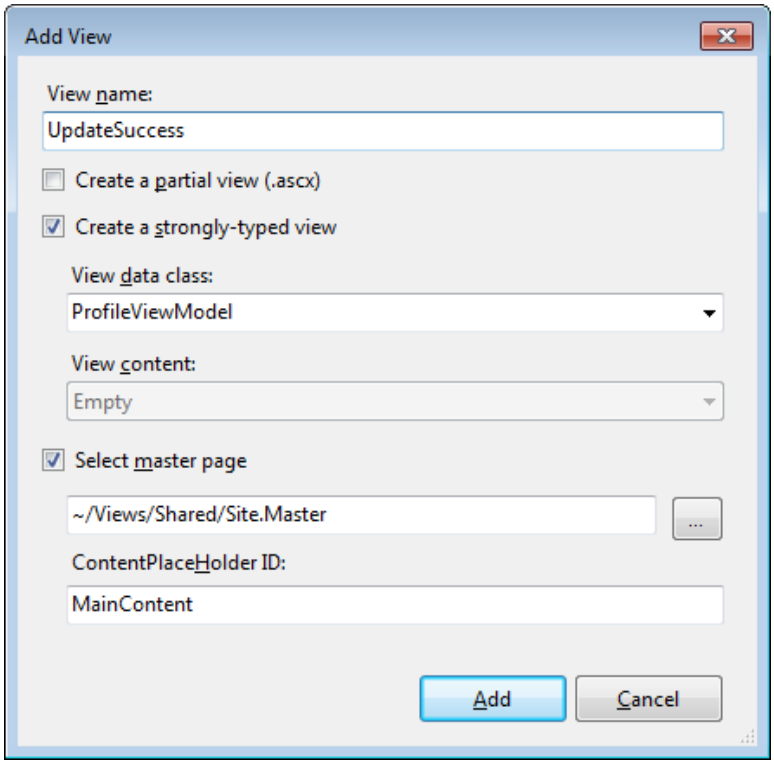
- *Html.EnableClientValidation* (Включить проверку на стороне клиента) обеспечивает проверку данных на стороне клиента на основании строго типизированного словаря ModelState
- *Html.BeginForm* (Начать форму) помещает в разметку тег *<form>* и закрывает форму в конце раздела директив *using*. Принимает различные параметры, но наиболее часто используемыми являются имя действия и контроллер, для которого это действие должно быть вызвано. Благодаря этому инфраструктура MVC может автоматически формировать URL конкретной формы во время выполнения, что избавляет от необходимости вводить строку URL в разметку.
- *Html.AntiForgeryToken* размещает в форме скрытое поле с проверочным значением, которое также сохраняется на стороне сервера и проверяется, если цель формы имеет атрибут *ValidateAntiForgeryToken*. Мы добавили этот атрибут в методе *Update* контроллера.

- *Html.EditorFor* (Редактор для) – перегруженный метод, обеспечивающий вставку текстового поля в разметку. Это строго типизированная версия метода *Html.Editor*.
- *Html.DropDownListFor* (Раскрывающийся список для) – перегруженный метод, обеспечивающий вставку раскрывающегося списка в разметку. Это строго типизированная версия метода *Html.DropDownList*.
- *Html.ValidationMessage* (Сообщение проверки) – вспомогательный метод, который обеспечит вывод на экран сообщения об ошибке проверки, если заданный ключ уже имеется в словаре ModelState.
- *Html.Hidden* (Скрытый) помещает в форму скрытое поле с переданными именем и значением.
- *Html.SubmitButton* (Кнопка Передать) создает в форме кнопку Submit (Передать).

> **Примечание** Когда разметка представления Index готова, остается лишь добавить представления для действия *UpdateSuccess*, и можно будет просматривать результаты.

Для создания представление UpdateSuccess:

7. Раскроем проект PlanMyNight.Web в Solution Explorer и затем раскроем папку Views.
8. Щелкнем правой кнопкой мыши папку Account.
9. Откроем подменю Add и щелкнем View.
10. Заполним поля диалогового окна Add View, чтобы оно выглядело следующим образом:



После создания страницы представления заполним содержимое заголовка следующим образом:

```
<asp:Content ContentPlaceHolderID="TitleContent" runat="server">Plan My Night -
Profile Updated</asp:Content>
```

И содержимое *MainContent* должно выглядеть так:

```
<asp:Content ContentPlaceHolderID="MainContent" runat="server">
<div class="panel" id="profileForm">
    <div class="innerPanel">
        <h2><span>My Profile</span></h2>
        <div class="items">
            <p>Your profile has been successfully updated.</p>
            <h3>» <a href="<%=Html.AttributeEncode(Model.ReturnUrl ??
                          Url.Content("~/"))%>">Continue</a></h3>
        </div>
        <div class="toolbox">
        </div>
    </div>
</div>
```

```
</asp:Content>
```

Чтобы увидеть созданные представления, необходимо отредактировать файл Site.Master (располагающийся в папке Views/Shared в корневом каталоге Веб-проекта). Раскомментируем строку 33, удалив теги комментариев. Код должен выглядеть следующим образом:

```
<%=Html.ActionLink<AccountController>(c => c.Index(null), "My Profile")%>
```

Теперь, когда это последнее представление создано, можно откомпилировать и выполнить приложение. Щелкнем кнопку Sign In (Вход), которая показана в верхнем правом углу формы на рис. 2-6, и выполним вход с помощью Windows Live ID.

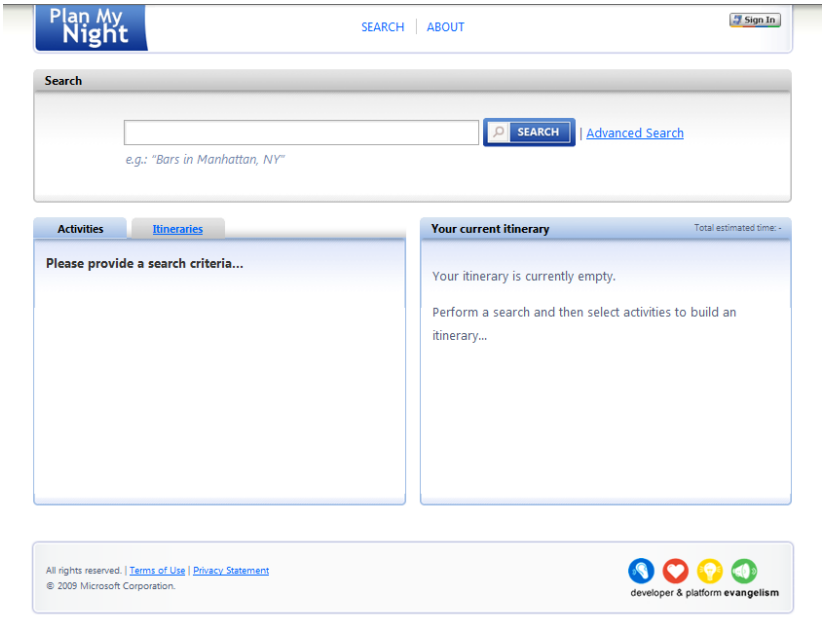


Рис. 2-6 Окно по умолчанию приложения Plan My Night

После того, как вход выполнен, вы должны быть перенаправлены к созданному представлению Index контроллера Account (рис. 2-7).

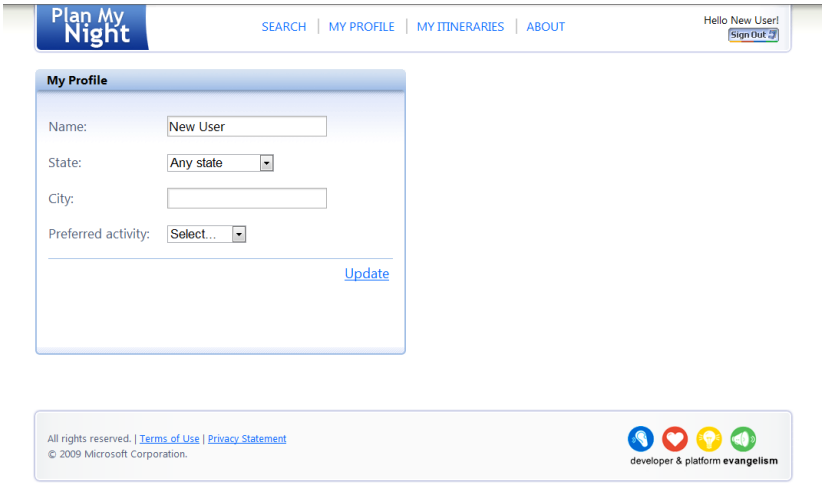


Рис. 2-7 Окно настроек профиля, возвращенное методом Index контроллера Account

Если вместо этого вы оказались на странице поиска, просто щелкните ссылку My Profile (Мой профиль), которая находится в центре вверху интерфейса рядом с остальными ссылками. Чтобы увидеть новые возможности проверки данных в действии, попробуйте сохранить форму, не заполнив поле Full Name (Полное имя). На экране должно появиться следующее сообщение (рис. 2-8).

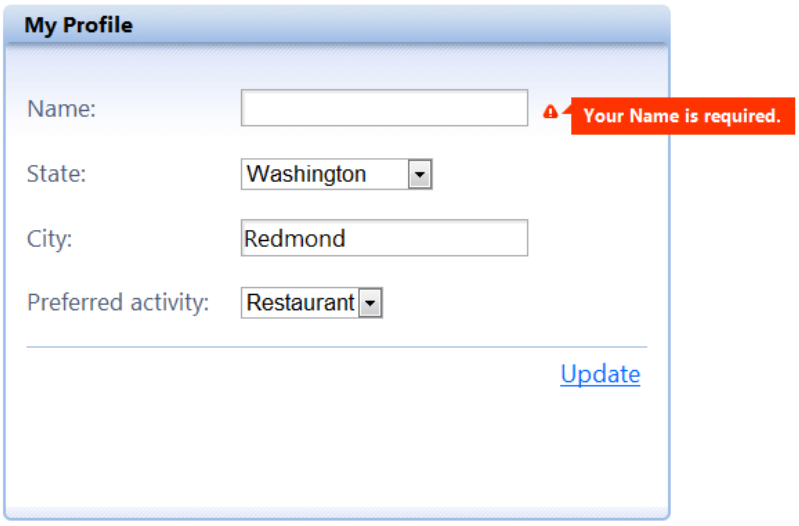


Рис. 2-8 Пример сбоя при проведении проверки Model Binding

Поскольку в приложении включена проверка на стороне клиента, обратного вызова не было. Чтобы увидеть, как работает проверка на стороне сервера, потребовалось бы внести изменения в файл Index.aspx в папке Account, закомментировав вызов *Html.EnableClientValidation*. Благодаря тесной интеграции и поддержке AJAX и JavaScript в приложениях MVC намного упрощается перенос на сторону клиента таких серверных операций, как проверка, по сравнению с тем, как это было ранее.

> **Visual Studio 2003** В приложениях ASP.NET MVC значение атрибута ID отдельного HTML-элемента не преобразовывается, как это происходит в ASP.NET Web Forms 1.0. В Visual Studio 2003 разработчик должен был сохранять *UniqueID* элемента управления/элемента в переменной JavaScript, чтобы обеспечить возможность доступа к нему внешнего JavaScript. Это делалось, чтобы гарантировать уникальность ID, но всегда вводило дополнительный уровень сложности во взаимодействие между элементами управления ASP.NET 1.0 Web Forms и JavaScript. В MVC такого преобразования нет, но обеспечение уникальности ID – сфера ответственности разработчиков. Также следует отметить, что теперь ASP.NET 4.0 Web Forms поддерживает отключение преобразования ID на уровне элемента управления по желанию разработчика

Контроллер Account и связанные представления и являются той недостающей «базовой» функциональностью Plan My Night. В ходе работы над ними мы рассмотрели некоторые новые возможности приложений Visual Studio 2010 и MVC 2.0. Но MVC не единственный доступный выбор для Веб-разработчиков. Веб-формы ASP.NET являются основным типом приложений ASP.NET с момента ее создания, и были улучшены в Visual Studio 2010. В следующем разделе Веб-форма ASP.NET для приложения MVC будет создана в визуальном дизайнере.

Использование дизайнера для создания Веб-формы

Все приложения рано или поздно сталкиваются с непредвиденными условиями, и наше приложение-пример не исключение. При возникновении непредвиденной ситуации оно возвращает сообщение об ошибке, такое как показано на рис. 2-9.

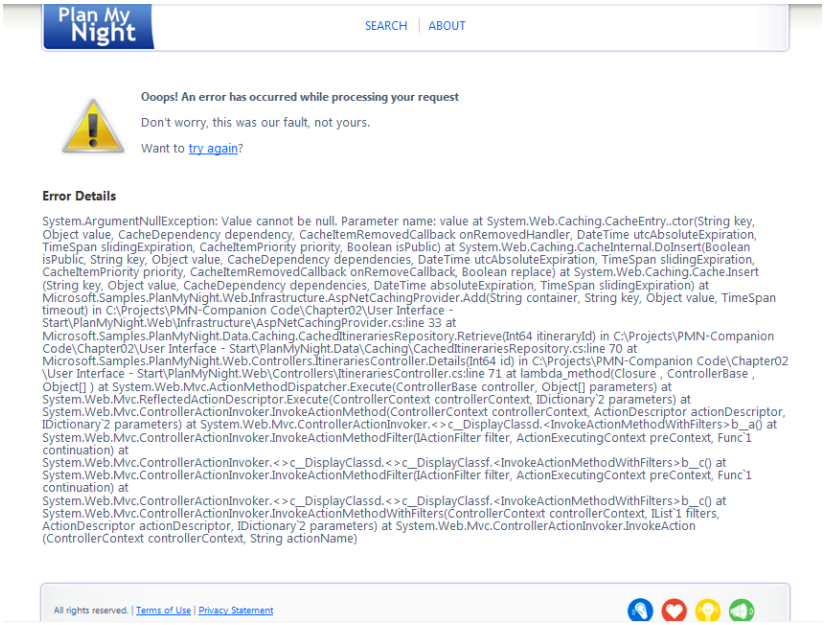
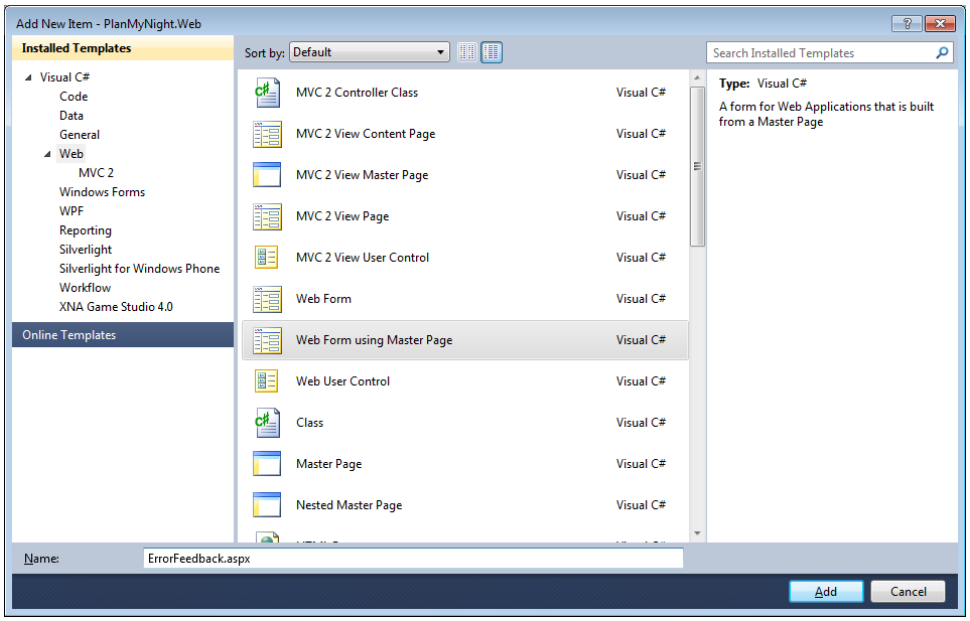


Рис. 2-9 Пример сообщения об ошибке приложения Plan My Night

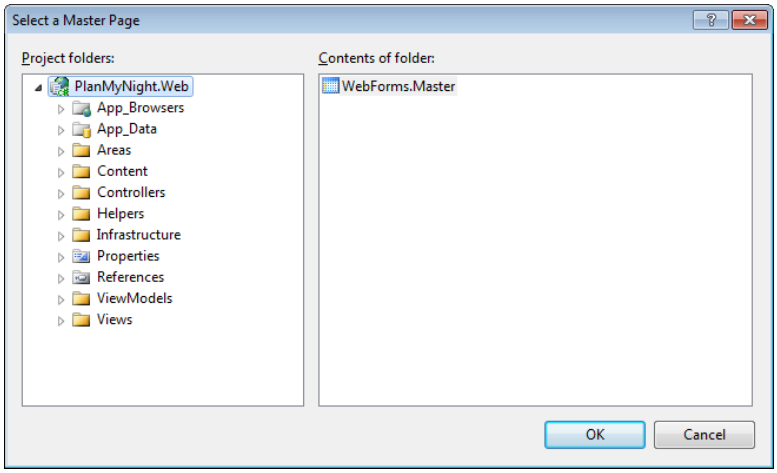
В настоящее время у пользователя, получившего такое сообщение, небогатый выбор: либо попытаться повторить это действие, либо использовать навигационные ссылки в верхней части окна приложения. (Конечно, это может привести к еще одной ошибке.) Предоставление пользователю возможности обратной связи позволит разработчикам получать сведения о ситуации, которых могут не обеспечивать стандартное сообщение об ошибке и трассировка стека. Чтобы рассмотреть другой способ создания компонента пользовательского интерфейса для приложения Plan My Night, спроектируем страницу обратной связи в случае возникновения ошибки как Веб-форму ASP.NET, преимущественно используя для этого дизайнер Visual Studio. Прежде чем приступать к дизайну формы, необходимо создать файл формы.

Для создания новой Веб-формы:

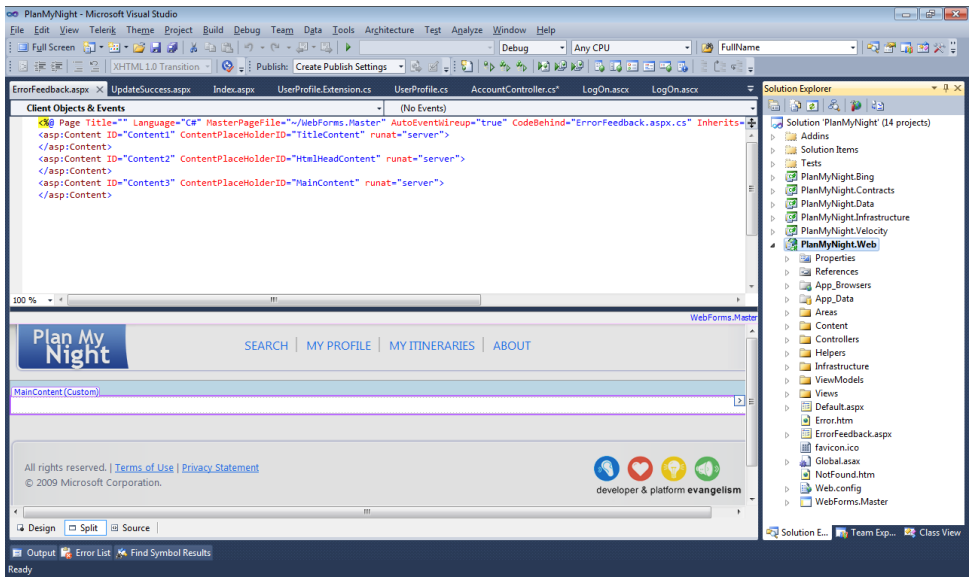
11. Откроем контекстное меню проекта PlanMyNight.Web (щелкнув его правой кнопкой мыши), откроем подменю Add и выберем New Item.
12. В диалоговом окне Add New Item выберем Web Form using Master Page (Веб-форма, использующая главную страницу) и в поле Name введем имя элемента: **ErrorFeedback.aspx**.



13. Появится диалоговое окно, которое позволяет ассоциировать главную страницу с этой Веб-формой. Убедитесь, что в части Project Folders (Папки проекта) окна выбрана основная папка PlanMyNight.Web, и затем выберите элемент WebForms.Master в правой части окна.



14. Результирующая страница может быть представлена не в режиме разделенного представления (Split view), а в режиме просмотра исходного кода (или в режиме дизайнера (Design view)). Перейдите к разделенному представлению (соответствующая кнопка располагается внизу окна, как и в предыдущих версиях Visual Studio). После этого экран должен выглядеть следующим образом:



> **Примечание** Рекомендуется использовать разделенное представление, поскольку оно позволяет видеть исходный код, формируемый дизайнером, и добавлять разметку в случае необходимости.

Раскройте панель элементов управления и оставьте ее на экране, поскольку в ходе работы над формой будете постоянно брать из нее элементы управления и элементы и переносить в область содержимого. Если эта панель еще не вынесена на экран, ее можно найти в меню View (Вид).

Начнем с переноса методом drag-and-drop элемента div (из группы элементов HTML) из панели элементов управления в раздел MainContent (Основное содержимое) дизайнера. При этом появится вкладка div, свидетельствующая о том, что добавленный новый элемент выбран в настоящий момент. Откроем контекстное меню div и выберем Properties (Свойства) (которые также можно открыть, нажав клавишу F4). В раскрывшемся окне Properties задаем свойству *(Id)* значение *profileForm* (Форма профиля) (регистр имеет значение). Также изменим значение свойства *Class* (Класс) на *panel* (панель). После редактирования этих значений размер области содержимого изменится, потому что к представлению дизайнера применены CSS.

> **Visual Studio 2003** Таким важным и необходимым усовершенствованием дизайнера Веб-форм, по сравнению с Visual Studio 2003, стало применение CSS. Благодаря этому разработчик получает возможность в реальном масштабе времени видеть эффекты от вносимых в стили изменений без необходимоысти выполнения приложения. В Visual Studio 2003 дизайнер для страницы search.aspx выглядит, как показано на рис. 2-10.

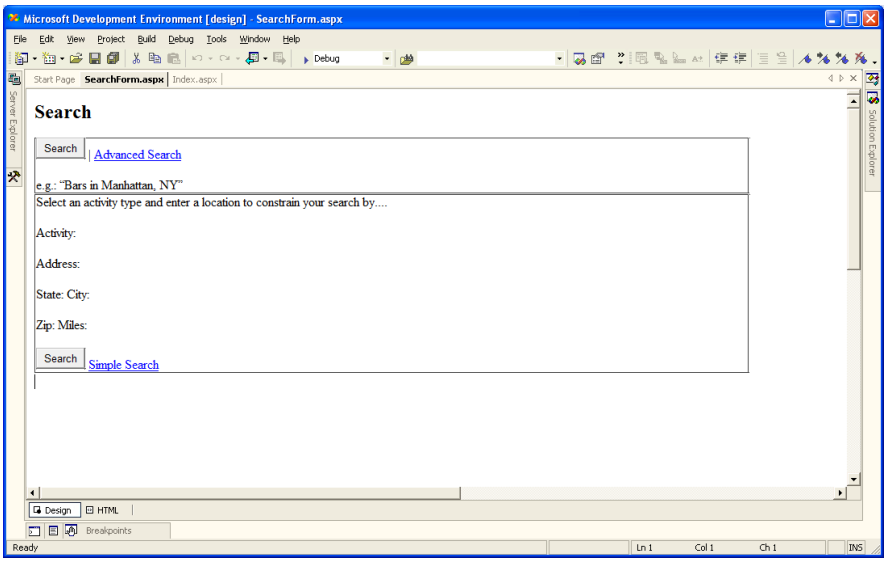


Рис. 2-10 Дизайнерское представление Веб-страницы ASP.NET в Visual Studio 2003

Поместим другой div внутрь первого и зададим его свойству *class* значение *innerPanel* (Внутренняя панель). На панели разметки добавим в *innerPanel* следующую разметку:

```
<h2><span>Error Feedback</span></h2>
```

Закрыв тег <h2>, добавим новую строку и откроем контекстное меню. Выберем Insert Snippet (Вставить фрагмент) и последовательно щелкнем ASP.NET > formr. Это обеспечит создание тега серверной формы, которая будет служить контейнером для Веб-элементов управления. Между тегами формы поместим тег div, атрибуту class которого присвоено значение *items* (элементы), и затем в тег тег div вставим тег fieldset (набор полей).

Далее перетягиваем в тег fieldset элемент управления TextBox (находится в разделе Standard (Стандартные) панели элементов управления). Задаем ID этого текстового поля значение **FullName**. В представлении разметки перед этим элементом управления добавляем тег <label> (метка), в качестве значения его свойства *for* задаем значение ID текстового поля и в качестве значения метки задаем **Full Name:** (двоеточие обязательно). Значение тега <label> - это текст, размещаемый между тегами <label> и </label>. Поместите эти два элемента в тег <p>, и представление дизайнера должно выглядеть, как показано на рис. 2-11.

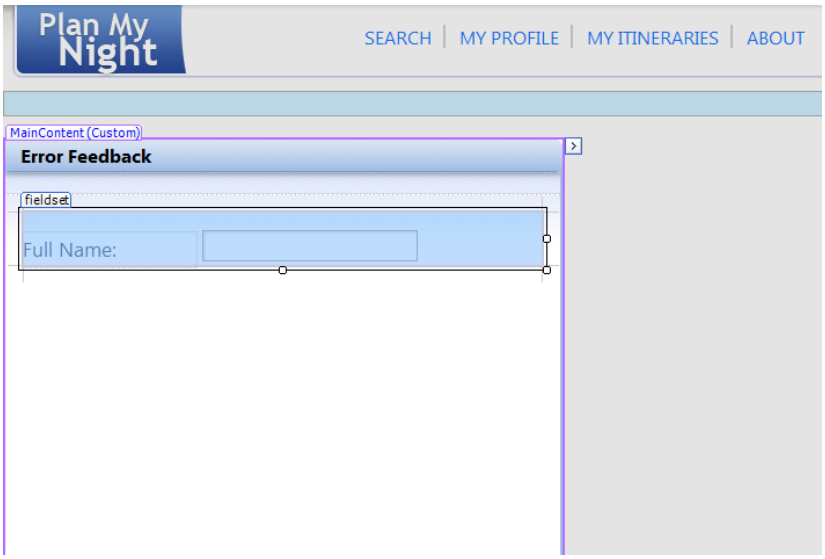


Рис. 2-11 Так на данный момен выглядит ErrorFeedback.aspx в дизайнере

Вышеописанным способом добавим еще одно текстовое поле и метку, но в качестве значения ID текстового поля зададим **EmailAddress** (Адрес электронной почты) и в качестве значения метки – **Email Address:** (не забывая о двоеточии). Повторим весь этот процесс третий раз и зададим TextBox ID и метку **Comments** (Комментарии). Теперь в дизайнере должно находиться три метки и три однострочных элемента управления TextBox. Для элемента управления Comments необходимо обеспечить многострочный ввод, поэтому открываем его свойства и задаем *TextMode* (Текстовый режим) значение *Multiline*

(Многострочный), *Rows* (Строки) значение *5* и *Columns* (Столбцы) значение *40*. После этого текстовое поле станет намного шире, что позволит пользователю вводить комментарии.

Снова воспользуемся возможностью Insert Snippet и после текстового поля Comments вставим тег «div with class» (HTML>divc). В качестве класса тега div зададим *submit*. Из панели инструментов перетащим в этот div элемент управления Button (Кнопка). Зададим свойству *Text* (Текст) кнопки значение *Send Feedback* (Отправить отзыв).

В дизайнере должно отображаться нечто похожее представленному на рис. 2-12. На данный момент мы получили страницу, которая будет передавать форму.

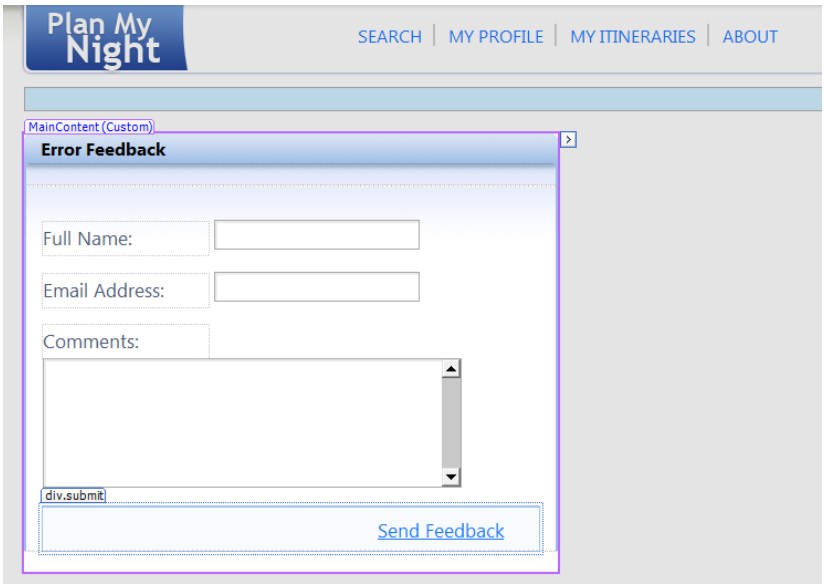


Рис. 2-12 Форма ErrorFeedback.aspx с полным набором полей

Но эта форма не выполняет никакой проверки передаваемых ею данных. Для реализации такой функциональности воспользуемся некоторыми элементами управления для проверки ввода пользователя, предлагаемыми ASP.NET. Сделаем поля Full Name и Comments обязательными для заполнения и осуществим проверку адреса электронной почты при помощи регулярных выражений, чтобы гарантировать его соответствие заданному шаблону.

В группе Validation (Проверка) панели инструментов имеются некоторые готовые элементы управления для проверки ввода пользователя. Возьмите на панели инструментов объект *RequiredFieldValidator* (Средство проверки обязательного поля) и поместите его справа от текстового поля Full Name. Откройте свойства этого элемента управления для проверки и задайте свойству *ControlToValidate* (Проверяемый элемент управления) значение *FullName*. (Для удобства предлагается раскрывающийся список элементов управления страницы.) Также задайте свойству *CssClass* (Класс CSS) значение *field-validation-error* (ошибка проверки поля). Это обусловит то, что все ошибки в приложении будут обозначаться красным треугольником. Наконец, свойству Error Message (Сообщение об ошибке) задаем значение **Name is Required** (Необходимо указать имя) (рис. 2-13).

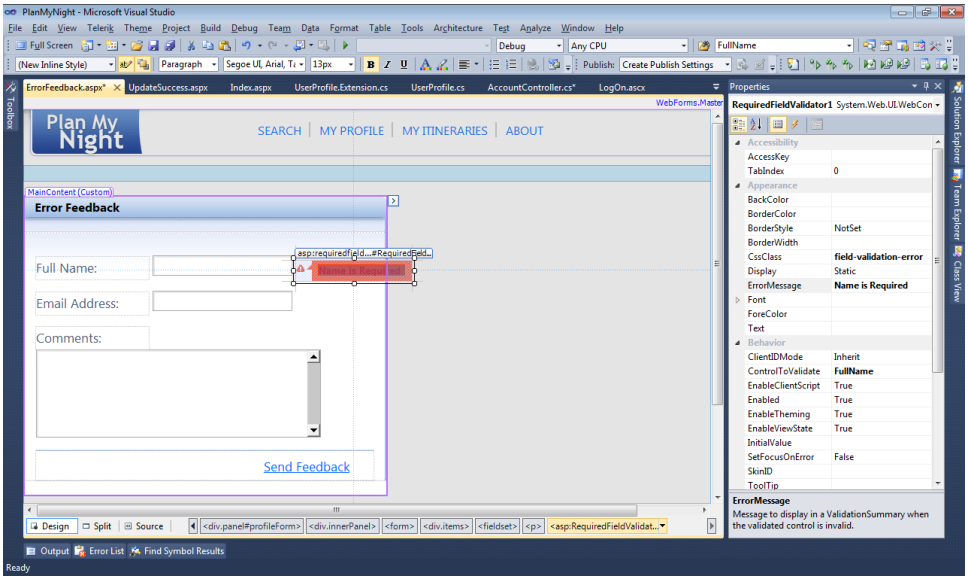


Рис. 2-13 Пример элемента управления для проверки ввода пользователя

Эти же шаги повторим для поля Comments, но зададим соответствующие значения свойствам *ErrorMessage* (Сообщение об ошибке) и *ControlToValidate*.

Пользователь должен гарантированно вводить действительный адрес электронной почты в поле Email Address, поэтому возьмем на панели инструментов элемент управления *RegularExpressionValidator* и поместим его рядом с этим текстовым полем. Значения свойствам этого элемента управления задаем по используемой ранее схеме, т.е. свойству *ControlToValidate* присваиваем значение *EmailAddress* и свойству *CssClass* – значение *field-validation-error*. Но для этого элемента управления также задается регулярное выражение, которое должно применяться к вводимым данным. Это осуществляется посредством свойства *ValidationExpression* (Выражение проверки), значение которого должно быть определено следующим образом:

```
[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}
```

Сообщение об ошибке этого средства проверки должно быть сформулировано примерно так: «Введите действительный адрес электронной почты».

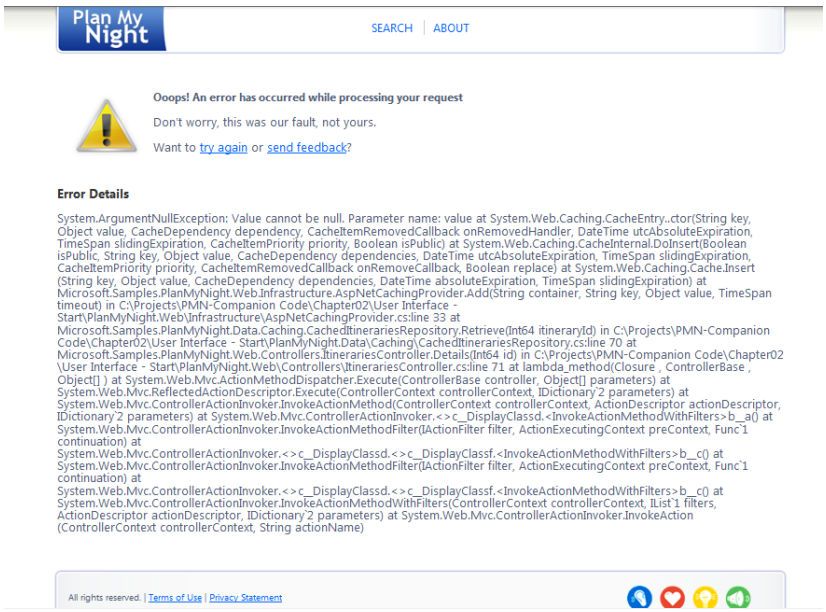
Форма готова. Чтобы увидеть ее в приложении, необходимо добавить опцию обратной связи при возникновении ошибки для пользователя. В Solution Explorer перейдите по дереву проекта PlanMyNight.Web к папке Views и затем к подпапке Shared. Откройте файл Error.aspx в средстве просмотра разметки и перейдите к строке 35. В этой строке обрабатывается предложение пользователю сделать еще одну попытку выполнить действие, и сюда мы поместим опцию для отправки отзыва. После текста вопроса в тот же абзац добавьте следующую разметку:

```
or <a href="/ErrorFeedback.aspx">send feedback</a>?
```

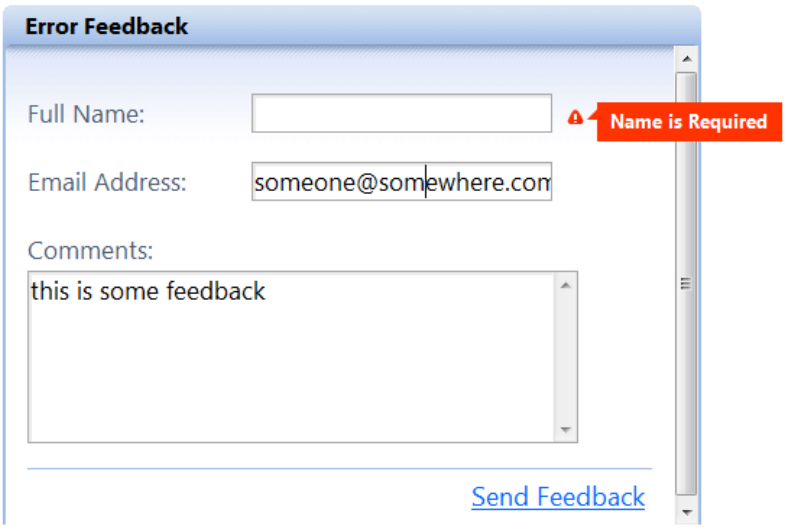
Это обеспечит добавление опции перехода к только что созданной нами форме при возникновении в приложении MVC любой ошибки общего характера. Чтобы увидеть эту форму, необходимо спровоцировать ошибку в приложении.

Чтобы спровоцировать возникновение ошибки в приложении Plan My Night:

15. Запустите приложение.
16. Увидев на экране стандартную страницу поиска, введите в адресную строку браузера следующий адрес: **http://www.planmynight.net:48580/Itineraries/Details/38923828**.
17. Очень маловероятно, что в базе данных имеется план мероприятий с таким ID, поэтому на экране появится страница с сообщением об ошибке.



18. На странице с сообщением об ошибке щелкните ссылку для перехода к форме обратной связи. Попробуйте передать форму, введя в нее недействительные данные.



Для осуществления проверки ASP.NET использует сценарий на стороне клиента (если браузер поддерживает это), поэтому пока данные вводятся, никаких обратных вызовов не выполняется. Когда же сервер получает введенные данные, разработчик может проконтролировать состояние проверки на стороне сервера с помощью свойства *Page.IsValid* в файле выделенного кода. Однако поскольку использовалась проверка на стороне клиента (по умолчанию), значение этого свойства всегда будет *true*. В файл выделенного кода необходимо добавить лишь код, обеспечивающий перенаправление пользователя при обратном вызове (и проверку свойства *Page.IsValid* в случае, если что-то было упущено при проверке на стороне клиента):

```
protected void Page_Load(object sender, EventArgs e)
    {
        if (this.IsPostBack && this.IsValid)
        {
            this.Response.Redirect("/", true);
        }
    }
```

В этом нет особого смысла для пользователя, но нашей целью в данном разделе было создание Веб-формы ASP.NET с помощью дизайнера. Поэтому в проекте PlanMyNight.Web появился новый интерфейс. Но что, если бы потребовалось обеспечить большую модульность функциональности, скажем, реализовать некую функцию, которую можно было бы добавлять или удалять без компиляции основного проекта приложения. Вот здесь-то и может продемонстрировать свои преимущества такая инфраструктура расширения как Managed Extensibility Framework (MEF). .

Расширение приложения с помощью MEF

Visual Studio 2010 предлагает новую технологию в составе .NET Framework 4 – Managed Extensibility Framework (MEF). Managed Extensibility Framework обеспечивает разработчикам простой, но при этом мощный механизм, с помощью которого сторонние производители получают возможность расширять приложения уже после их поставки. Благодаря MEF даже в рамках одного приложения разработчики могут создавать полностью изолированные компоненты, что обеспечивает возможность их независимого обслуживания или замены. Для этого MEF использует контейнер разрешений, который позволяет сопоставлять компоненты, обеспечивающие конкретную функцию (экспортеры), и компоненты, нуждающиеся в этой функциональности (импортеры), и при этом двум конкретным компонентам даже не надо ничего знать друг о друге. Разрешения выполняются только на контрактной основе, что делает компоненты взаимозаменяемыми или позволяет вводить их в приложения с очень небольшими издержками.

> *Дополнительные сведения Веб-сайт сообщества разработчиков MEF, на котором представлена детальная информация об этой архитектуре, можно найти по адресу* http://mef.codeplex.com.

Приложение Plan My Night создавалось с учетом возможности расширения. В папке Addins (Надстройки) его решения имеется три проекта модулей «надстроек» (рис. 2-14).

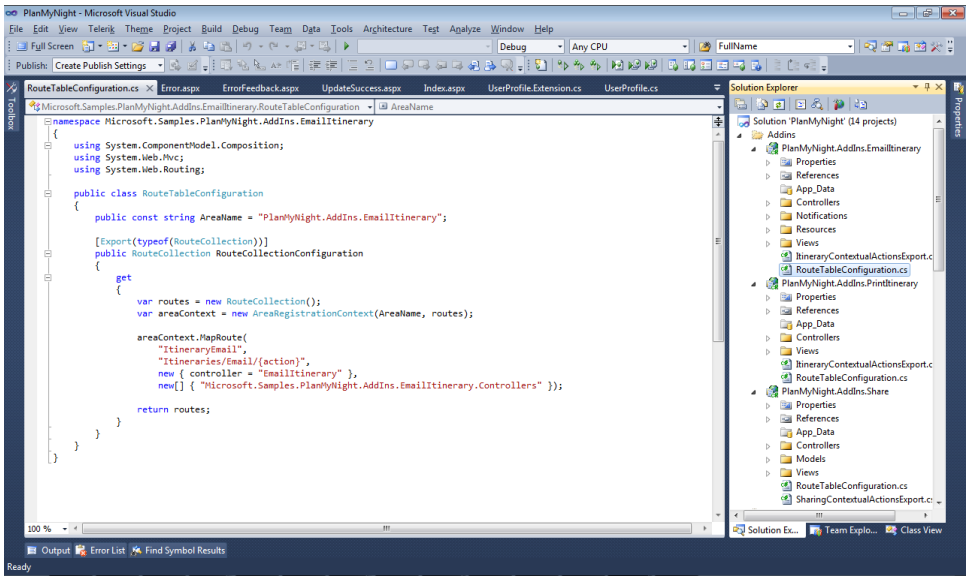


Рис. 2-14 Надстройки приложения Plan My Night

PlanMyNight.Addins.EmailItinerary добавляет возможность отправлять списки маршрутов всем, кому пожелает пользователь. PlanMyNight.Addins.PrintItinerary обеспечивает версию для печати маршрута. Наконец, PlanMyNight.Addins.Share добавляет функции публикации на социальных сайтах (что позволяет пользователю публиковать ссылку на маршрут), а также операции «обрезки» URL. Ни один из этих проектов не ссылается на основное приложение и не упоминается в нем. Да, в них имеются ссылки на проекты PlanMyNight.Contracts и PlanMyNight.Infrastructure, что позволяет экспортировать (и импортировать в некоторых случаях) соответствующие контракты через MEF, а также использовать любые специальные расширения проекта инфраструктуры.

> **Примечание** Если Веб-приложение еще не выполняется, прежде чем переходить к следующему этапу, запустите проект PlanMyNight.Web, чтобы видеть пользовательский интерфейс.

Чтобы добавить модули в выполняющееся приложение, запустим файл DeployAllAddins.bat, располагающийся в одной папке с файлом PlanMyNight.sln. Это обеспечит создание новых папок в разделе Areas проекта PlanMyNight.Web. Новые папки, по одной для каждой надстройки, будут включать файлы, необходимые для добавления их функциональности в основное Веб-приложение. Надстройки появляются в приложении как дополнительные опции на странице результатов поиска и на странице данных маршрута под разделом текущего маршрута. После того как командный файл закончит выполнение, перейдите к интерфейсу PlanMyNight, выполните поиск действия и добавьте его в текущий маршрут. Теперь под панелью маршрута, кроме New (Создать) и Save (Сохранить), должны появиться дополнительные опции (рис. 2-15).

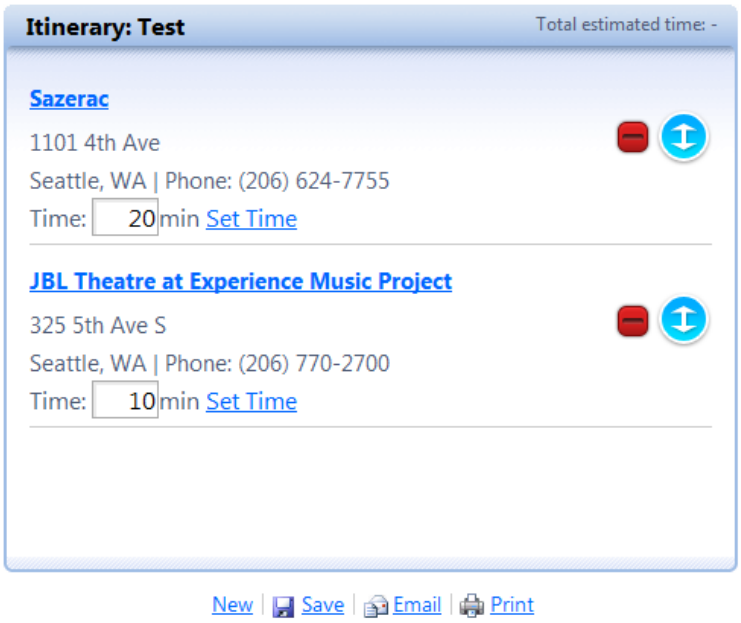


Рис. 2-15 Пользовательский интерфейс с надстройкой для отправки маршрута по электронной почте

Опции публикации на социальных сайтах начнут отображаться в интерфейсе только после того, как маршрут будет сохранен и отмечен как общедоступный (рис. 2-16).

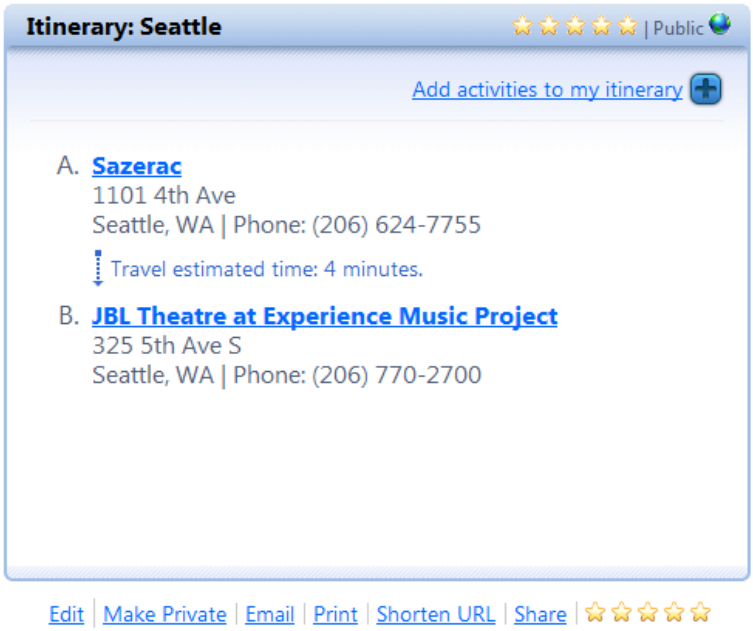


Рис. 2-16 Пользовательский интерфейс с надстройкой для публикации маршрута на социальных сайтах

> **Visual Studio 2003** В Visual Studio 2003 нет ничего похожего на MEF. Для обеспечения поддержки подключаемых модулей разработчику пришлось бы создавать инфраструктуру подключаемого модуля с нуля или приобретать коммерческий продукт. Любой из этих вариантов приводил к созданию собственных решений, в которых внешнему разработчику приходилось разбираться в случае необходимости реализации компонента для них. Добавление MEF в .NET Framework помогает убрать барьеры для перехода к разработке расширяемых приложений и подключаемых модулей для них.

Надстройка для создания печатной версии плана мероприятий

Рассмотрим использование подключаемых модулей в приложении на примере проекта PrintItinerary.Addin. Дерево проекта в развернутом виде должно быть похожим на структуру, представленную на рис. 2-17.

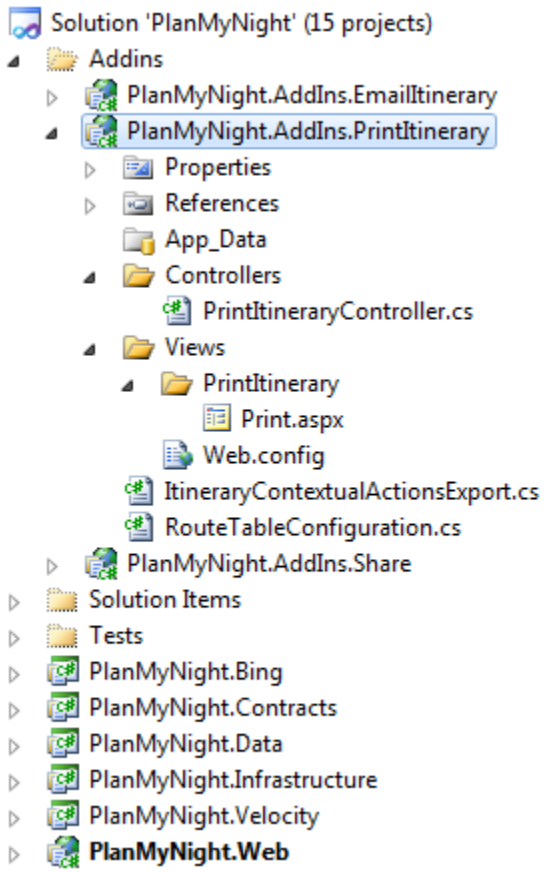


Рис. 2-17 Структура проекта PrintItinerary

Некоторые моменты данной структуры аналогичны проекту PlanMyNight.Web (Controllers и Views), и поэтому эта надстройка будет помещена в приложение MVC как Area. Если более внимательно посмотреть на файл PrintItineraryController.cs в папке Controller, можно заметить, что его структура очень похожа на структуру контроллера, созданного нами ранее в этой главе (и любого другого контроллера Веб-приложения), но при этом имеются и существенные отличия от контроллеров, компилируемых в основном приложении PlanMyNight.Web.

В описании класса присутствуют дополнительные атрибуты:

```
[Export("PrintItinerary", typeof(IController))]
[PartCreationPolicy(CreationPolicy.NonShared)]
```

Эти два атрибута описывают данный тип контейнера разрешений MEF. Первый атрибут, *Export* (Экспорт), помечает этот класс как предоставляющий *IController* с именем контракта *PrintItinerary*. Второй атрибут объявляет, что этот объект поддерживает только индивидуальное создание и не может создаваться как совместно используемый/singleton-объект. Задание этих двух атрибутов – это все что необходимо для создания типа, используемого MEF. На самом деле, *PartCreationPolicy* (Политика создания части) является необязательным атрибутом, но он должен быть определен на случай, если тип не может обрабатывать все типы политики создания.

Далее в файле PrintItineraryController.cs можно заметить, что конструктор снабжен атрибутом *ImportingConstructor* (Конструктор импорта):

```
[ImportingConstructor]
public PrintItineraryController(IServiceFactory serviceFactory) :
this(
    serviceFactory.GetItineraryContainerInstance(),
    serviceFactory.GetItinerariesRepositoryInstance(),
    serviceFactory.GetActivitiesRepositoryInstance())
    {
    }
```

Атрибут *ImportingConstructor* указывает MEF о необходимости предоставления параметров при создании этого объекта. В данном конкретном случае MEF обеспечивает экземпляр *IServiceFactory* для использования этим объектом. Классу *this* не важно, откуда поступает экземпляр, участвующий в создании

модульных приложений. Для наших целей оговоренный контрактом *IServiceFactory* экспортируется в проект PlanMyNight.Web файлом ServiceFactory.cs.

Файл RouteTableConfiguration.cs регистрирует сведения о маршруте URL, которые должны быть направлены в PrintItineraryController. Этот маршрут, и маршруты других надстроек, регистрируются в приложении в ходе выполнения метода *Application_Start*, описанного в файле Global.asax.cs приложения PlanMyNight.Web:

```
// Фабрика EF Controller
var controllerFactory = new MefControllerFactory(container);
ControllerBuilder.Current.SetControllerFactory(controllerFactory);

// Регистрация маршрутов из надстроек
foreach (RouteCollection routes in container.GetExportedValues<RouteCollection>())
{
    foreach (var route in routes)
    {
        RouteTable.Routes.Add(route);
    }
}
```

controllerFactory – это контейнер MEF, включающий путь к подпапке Areas (в которой находятся все надстройки). Он назначен фабрикой контроллера на весь срок жизни приложения. Благодаря этому контроллеры, импортированные через MEF, могут использоваться в любом месте приложения. Маршруты этих надстроек извлекаются из контейнера MEF и регистрируются в таблице маршрутизации MVC.

Файл ItineraryContextualActionsExport.cs экспортирует сведения для создания ссылки на этот подключаемый модуль, а также создания метаданных для отображения. Эти сведения используются в файле ViewModelExtensions.cs проекта PlanMyNight.Web при построении модели представления для отображения пользователю:

```
// получаем ссылки и панели инструментов надстроек
var addinBoxes = new List<RouteValueDictionary>();
var addinLinks = new List<ExtensionLink>();

addinBoxes.AddRange(AddinExtensions.GetActionsFor("ItineraryToolbox", model.Id == 0
? null : new { id = model.Id }));

addinLinks.AddRange(AddinExtensions.GetLinksFor("ItineraryLinks", model.Id == 0 ?
null : new { id = model.Id }));
```

Вызов *AddinExtensions.GetLinksFor* обеспечивает перечисление экспортированных элементов в поставщике экспорта MEF и возвращает их коллекцию, которая может быть добавлена в локальную коллекцию *addinLinks* (Ссылки на надстройки). Впоследствии эта коллекция используется в представлении для отображения большего числа опций, если они имеются.

Заключение

В данной главе мы обсудили лишь несколько возможностей из большого числа новых технологий, предлагаемых Visual Studio 2010. Поэтапно рассмотрели создание контроллера и его связанного представления, а также мощную возможность, предлагаемую Веб-разработчикам инфраструктурой ASP.NET MVC для создания Веб-приложений. Мы познакомились, как с помощью Managed Extensibility Framework можно создавать внешние модули надстроек и подключать их к приложению во время выполнения. В следующей главе будет показано, какие улучшения предлагает Visual Studio 2010 для отладки приложений.

Глава 3

От 2003 к 2010: отладка приложения

В данной главе рассматривается

- Использование новых возможностей отладки Microsoft Visual Studio 2010
- Создание модульных тестов и их выполнение в Visual Studio 2010
- Проводится сравнение возможностей и отличий Visual Studio 2003 и 2010

В ходе написания данной книги мы увидели, насколько сильно эволюционировали инструменты отладки и различные вспомогательные средства для разработчиков в трех последних версиях Visual Studio. В Visual Studio 2010 внимание акцентируется на отладке приложения и написании модульных тестов, что лишь расширяет предлагаемые возможности.

Возможности отладки в Visual Studio 2010

В данной главе рассматриваются различные возможности отладки. В качестве примера используется модифицированное приложение Plan My Night. Если сопроводительные материалы данной книги установлены в каталог по умолчанию, интересующее нас приложение находится по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 3\Code. Щелкните двойным щелчком файл PlanMyNight.sln.

Прежде чем переходить непосредственно к процессу отладки, необходимо выполнить некоторую настройку:

19. Проверьте в Solution Explorer, является ли PlanMyNight.Web автозагружаемым проектом. Если его имя не выделено жирным шрифтом, щелкните PlanMyNight.Web правой кнопкой и выберите Set As StartUp Project (Назначить автозагружаемым проектом).
20. Для подготовки к следующим этапам в решении PlanMyNight.Web откройте файл Global.asax.cs, щелкнув треугольник рядом с папкой Global.asax, и затем щелкните двойным щелчком файл Global.asax.cs, как показано на рис. 3-1.

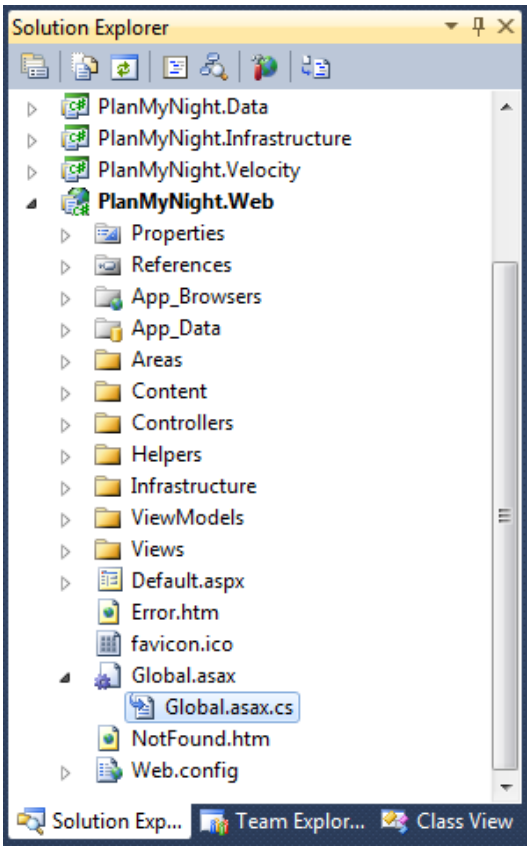


Рис. 3-1 Внешний вид Solution Explorer до открытия файла Global.asax.cs

Управление сеансом отладки

На примере приложения Plan My Night мы увидим, как может разработчик управлять точками останова и организовывать их совместное использование. Новые улучшенные возможности работы с точками останова позволяют значительно повысить скорость и эффективность проверки различных элементов данных в приложении. Также будут рассмотрены новые минидампы и новый интерпретатор промежуточного языка (intermediate language, IL), с помощью которого разработчик может видеть свойства и функции управляемого кода во время отладки минидампа.

Новые улучшенные возможности работы с точками останова

Итак, на данный момент в редакторе открыт файл Global.ascx.cs. Рассмотрим поэтапно некоторые из возможных путей управления точками останова и организации их совместного использования.

21. Перейдем к методу *Application_BeginRequest(object sender, EventArgs e)* и установим точку останова в строке *var url = HttpContext.Current.Request.Url;*, щелкнув поле слева или нажав F9. Рис. 3-2 демонстрирует, как это выглядит в редакторе.

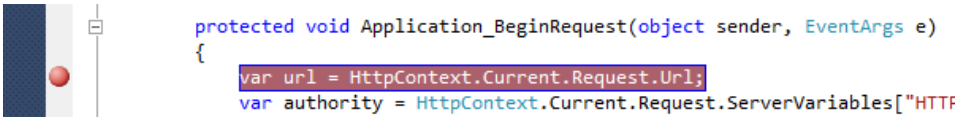


Рис. 3-2 Создание точки останова

22. Нажмем F5, чтобы запустить приложение в режиме отладки. В панели задач должен быть отражен запуск Веб-сервера разработчика, откроется новое окно браузера, и приложение немедленно прекратит выполнение в только что созданной точке останова. Если окно Breakpoints (Точки останова) не появилось на экране даже после запуска приложения в режиме отладки, в меню Debug (Отладка) выберите опцию Windows (Окна) и затем Breakpoints, или используйте комбинацию горячих клавиш Ctrl+D+B.

> **Visual Studio 2003** Отладка в Visual Studio 2003 и в Visual Studio 2010 отличаются, как день и ночь. То насколько просто выполнять отладку в Visual Studio 2010, не идет ни в какое сравнение с теми сложностями, с которыми приходилось сталкиваться разработчикам в Visual Studio 2003.
>
> Для успешной отладки Веб-приложений в Visual Studio 2003 требовалось четко следовать определенной схеме. Например, необходимо было организовать отладку в нескольких разных местах с помощью Configuration Manager (Диспетчер конфигураций), тщательно настроить IIS во избежание получения ужаснейшего исключения системы безопасности Temporary ASP.NET Files Access Denied (Доступ к временным файлам ASP.NET запрещен). В общем и целом, весь процесс не был ориентирован на разработчика и требовал выполнения большого количества настроек без всяких гарантий на успех. Было написано множество статей, призванных помочь разобраться со сложностями отладки Веб-приложений ASP.NET с использованием Internet Information Services (IIS) 5 и 6 и Visual Studio 2003. Очень хорошим источником справочной информации, которым пользовался, вероятно, каждый разработчик, является материал *http://msdn.microsoft.com/en-us/library/aa290100(VS.71).aspx*. Множество блогов или статей было написано с одной лишь целью – объяснить профессиональным разработчикам, как выполнять отладку на их рабочих станциях. Даже беглого просмотра сетевых форумов достаточно, чтобы понять, как многим людям приходилось годами сражаться с десятками сложнейших ситуаций, возникавших при отладке Веб-приложений. Например, довольно большую проблему представляла отладка приложения ASP.NET, использующего несколько Веб-сервисов.
>
> Создание собственного Веб-сервера (под именем «Cassini») значительно улучшило положение, а благодаря ряду дополнений, внесенных в Visual Studio 2010, упростились задачи по изменению необходимых файлов конфигурации, изменению параметров проекта и развертыванию на IIS. Эти улучшения позволили разработчикам создавать более качественный код и тратить меньше времени на его отладку. В Visual Studio 2010 отладка дает совершенно другие ощущения, выполнять ее приятно и легко.

Теперь на экран должно быть выведено окно Breakpoints, как показано на рис. 3-3.

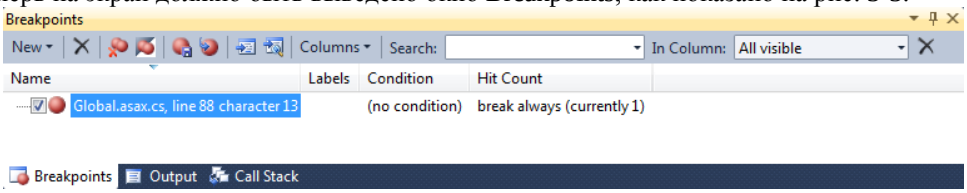


Рис. 3-3 Окно Breakpoints

23. В том же методе добавьте еще три точки останова. Редактор кода и окно Breakpoints должны выглядеть, как показано на рис. 3-4.

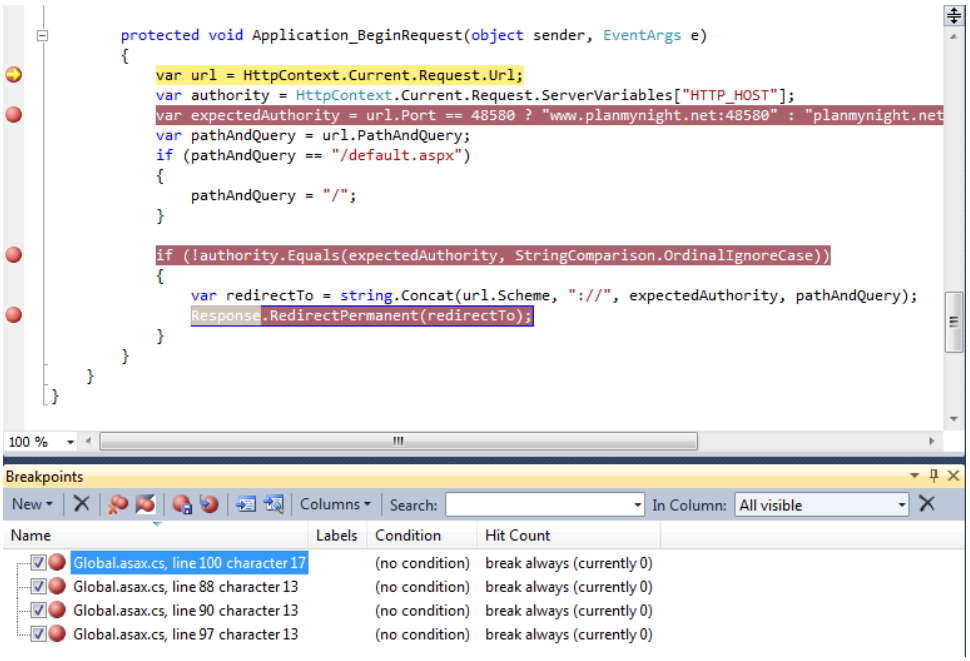
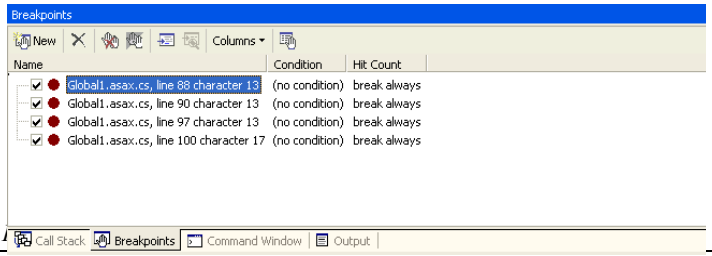


Рис. 3-4 Редактор кода и окно Breakpoints с тремя новыми точками останова

Visual Studio 2003 Внимательный читатель и профессиональный разработчик, часто использовавший Visual Studio 2003, вероятно, заметил в этом упражнении в окне Breakpoints ряд новых кнопок и полей. Напомним, как это окно выглядело в Visual Studio 2003 (рис. 3-5).



24. Обратите внимание, в окне Breakpoints теперь имеется столбец Labels (Метки), что упрощает индексацию и поиск точек останова. Это на самом деле замечательная и очень полезная возможность, появившаяся в Visual Studio 2010. Чтобы воспользоваться ею, достаточно щелкнуть правой кнопкой точку останова в окне Breakpoints и выбрать Edit Labels (Редактировать метки) или использовать сочетание горячих клавиш Alt+F9, L. Это показано на рис. 3-6.

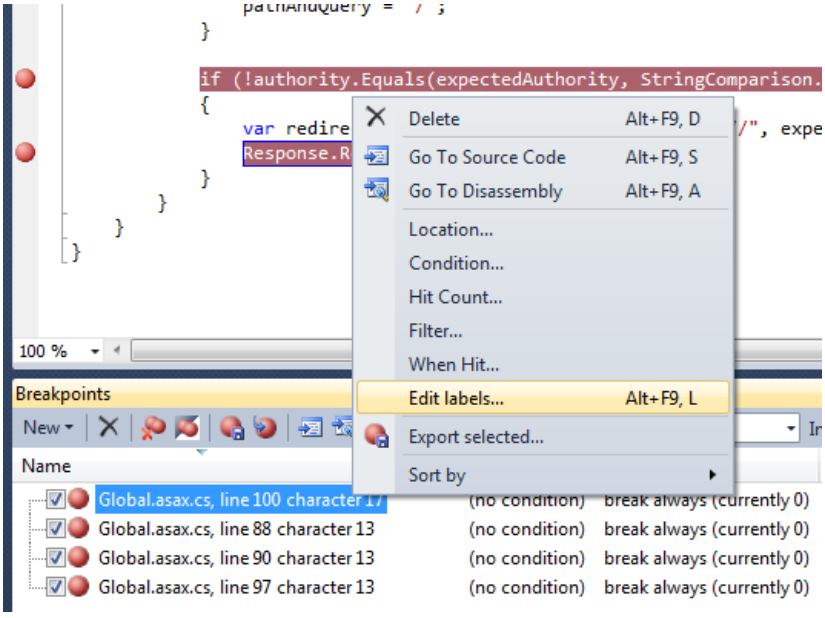


Рис. 3-6 Опция Edit Labels

25. В окне Edit Breakpoint Labels (Редактор меток точек останова) добавьте метки для выбранной точки останова (первой в окне Breakpoints). В текстовом поле Type a New Label (Введите новую метку) введите **ContextRequestUrl** и щелкните Add. Повторите эту операцию для следующей точки останова и введите имя метки **Url**. Когда все готово, щелкните OK. На рис. 3-7 представлено, как выглядит окно Edit Breakpoint Labels в момент ввода меток, и окно Breakpoints после выполнения этих двух операций.

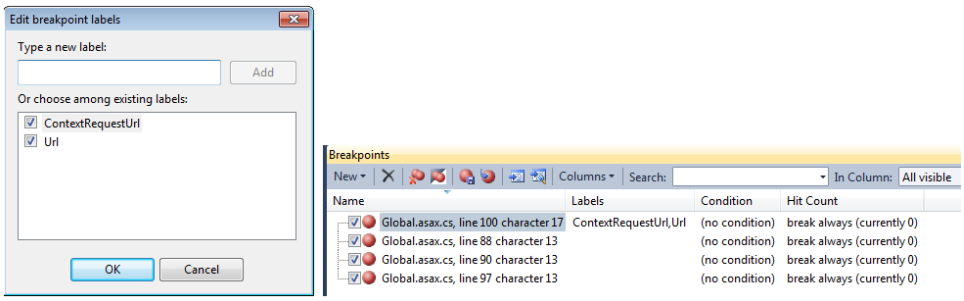


Рис. 3-7 Добавление меток, которые отображаются в окне Breakpoints

> **Примечание** Также вышеописанные задачи можно выполнить, щелкнув правой кнопкой точку останова в поле слева и выбрав в появившемся меню опцию Edit Labels.

> **Примечание** При добавлении меток для новой точки останова есть возможность выбрать любую из существующих меток, которые перечислены в области Or Choose Among Existing Labels (Или выбрать из существующих меток) диалогового окна Edit Breakpoint Labels (рис. 3-7).

26. Любым из рассмотренных способов добавляем метки для каждой точки останова. После этого окно Breakpoints должно выглядеть, как представлено на рис. 3-8.

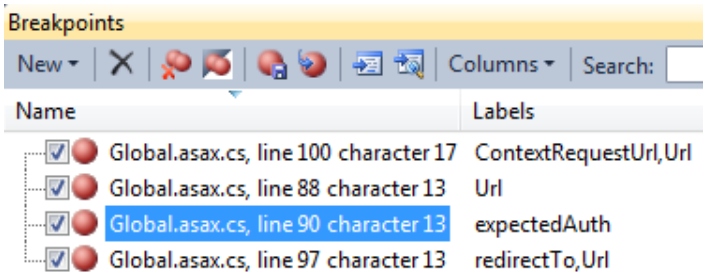


Рис. 3-8 Окно Breakpoints со всеми метками

В ходе отладки больших объемов кода очень пригодится возможность применения фильтра к отображаемому списку точек остановка. Именно это позволяет делать новая функция поиска в Visual Studio 2010.

27. Чтобы увидеть возможность поиска в действии, просто введите **url** в строку поиска и получите список, включающий только точки останова, в метках которых имеется строка *url*.

При групповой разработке, когда в одном проекте занято множество разработчиков и тестировщиков, над одними и теми же ошибками часто работают два человека одновременно. В Visual Studio 2003 этим двум сотрудникам приходилось бы сидеть рядом, обмениваться мгновенными снимками экрана или пересылать друг другу номера строк кода, в которых требуется поставить точки останова, чтобы уточнить, с каким фрагментом кода работать при отладке конкретной ошибки.

> **Важно** Еще одним замечательным дополнением к управлению точками останова в Visual Studio 2010 является появившаяся возможность экспортировать их в файл, который затем можно переслать коллеге. Из этого файла точки останова могут быть импортированы в другую среду. Эта возможность также пригодится, если требуется распространить точки останова на несколько компьютеров. Далее мы рассмотрим, как это делается.

28. В окне Breakpoints щелкните кнопку Export (Экспортировать)  для экспорта точек останова в файл и сохраните этот файл на рабочем столе. Назовите файл **breakexports.xml**.

29. Удалите все точки останова, либо щелкнув кнопку Delete All Breakpoints Matching The Current Search Criteria (Удалить все точки останова, соответствующие текущему критерию поиска)  , либо выбрав все точки останова и нажав кнопку Delete The Selected Breakpoints (Удалить выбранные точки останова)  . Удаляем точки останова с целью моделировать их совместное использование двумя разработчиками или распространение их на два компьютера.

30. Теперь выполните импорт точек останова, нажав кнопку Import (Импортировать)  и загрузив их с рабочего стола. Обратите внимание, что все точки останова со всеми свойствами опять появились в среде разработки. Для целей данной главы удалите все точки останова.

В Visual Studio 2003 размещение точек останова в клиентском коде (JavaScript) и отладка не были особо приятной процедурой. Разработчику приходилось вводить выражение debugger (или stop в VBScript) в клиентский код и затем выполнять его построчную трассировку. Кроме того, не было поддержки IntelliSense для клиентского кода. В Visual Studio 2010 предоставляется замечательная поддержка для JavaScript, а также jQuery. Уже в Visual Studio 2008 все было замечательно, но интеграция в Visual Studio 2010 намного более производительная, и разработчик может просто воспользоваться ею, без всяких дополнительных операций.

Проверка данных

Всем известно, как много времени при отладке приложения может уйти на проверку содержимого переменных, аргументов и пр. Помните, еще совсем недавно, когда отладчики не были реальностью или находились в зачаточном состоянии, как много выражений *printf* или *WriteLn* приходилось писать для проверки содержимого различных элементов данных? Хотя, возможно вы и не помните, вероятно, вы не так стары, как мы.

Visual Studio 2003 Уже Visual Studio 2003 предлагала настоящий отладчик с новой функциональностью, что являло собой значительный прогресс по сравнению с временами, когда приходилось писать все виды выражений в консоли. Новые средства визуализации данных позволяли видеть форматированный XML, а не просто длинную строку кода. Более того, они представляли массивы в более удобном для восприятия виде – как список элементов и их индексов – и чтобы увидеть это, достаточно было навести курсор мыши на объект. Пример представлен на рис. 3-9.

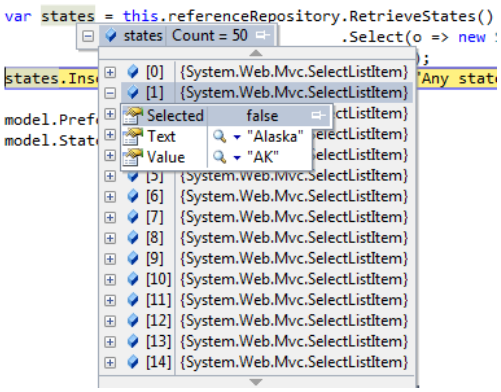


Рис. 3-9 Сравнение представления в виде коллекции и в виде массива в отладчике в Visual Studio 2010 и в Visual Studio 2003

Visual Studio 2003 В Visual Studio 2003 единственными средствами визуализации данных были окна Watch (Контрольные значения), Locals (Локальные переменные) и Quick Watch (Контрольное значение) и другие еще более рудиментарные способы, описываемые выше. При наведении курсора мыши на поле в массиве его содержимое не отображалось. Любой XML или форматированный текст в строке нельзя было прочитать иным способом, кроме как открыв Notepad или другой файл в Visual Studio и вставив содержимое строки. Это позволяло увидеть форматированное содержимое переменной.

Все эти техники визуализации данных в виде всплывающей подсказки по-прежнему доступны в Visual Studio 2010, но при этом предлагается ряд нововведений, которые повышают эффективность всплывающих подсказок по данным. В сочетании с улучшениями всплывающих подсказок в Visual Studio 2010 появилась еще одна возможность – поддержка одновременной работы с несколькими мониторами. Всплывающие подсказки по данным очень полезны для разработчика. А наличие возможности поместить их на второй экран может намного облегчить отладку, позволяя всегда иметь перед глазами

данные, которые всегда должны быть в контексте, на втором мониторе. Рассмотрим поэтапно, как использовать эти возможности:

31. В файле Global.ascx.cs установите точки останова в строках 89 и 91 – строках, начинающихся с *var authority* и *var pathAndQuery*, соответственно.
32. Поэкспериментируем с новыми возможностями всплывающих подсказок по данным. Запустим отладчик, нажав F5. Когда отладчик встречает первую точку останова, наведите курсор на слово *url* и щелкните значок вешки, как показано на рис. 3-10.

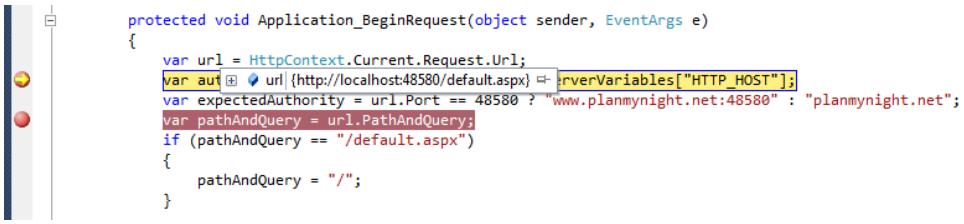
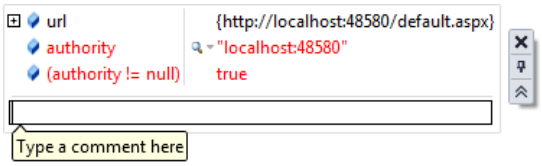


Рис. 3-10 Новая возможность всплывающих подсказок по данным – вешка

33. Справа от строки кода должен располагаться закрепленная всплывающая подсказка по данным (как показано на рис. 3-11 слева). Если навести курсором мыши на эту подсказку, появится панель управления всплывающей подсказки по данным (как показано на рис. 3-11 справа).

Рис. 3-11 Слева – закрепленная всплывающая подсказка по данным, и справа – ее панель управления

> **Примечание** Также в поле точек останова должна отображаться голубая вешка, указывающая на то, что данная всплывающая подсказка закреплена. Вешка выглядит так: . Поскольку в этой строке располагается точка останова, фактически вешка находится под ней. Чтобы увидеть вешку, просто переключите точку останова, щелкнув ее в поле. Один щелчок снимет точку останова, второй – вернет ее назад.

> **Примечание** По щелчку указывающей вниз двойной стрелки на панели управления всплывающей подсказки по данным появляется строка, в которую можно ввести комментарий к этой подсказке, как показано на рис. 3-12. Также можно полностью удалить всплывающую подсказку, нажав кнопку X на панели управления всплывающей подсказки.
>
> 
>
>
> *Рис. 3-12 Вставка комментария к всплывающей подсказке по данным*

34. Еще одно замечательное свойство новой всплывающей подсказки по данным в том, что есть возможность вставить любое выражение, и оно будет вычислено прямо в ходе сеанса отладки. Например, щелкните правой кнопкой имя всплывающей подсказки, в данном случае это *url*, выберите Add Expression (Добавить выражение), введите **authority** и добавьте еще одно, например, **(authority != null)**. Выражение тут же вычисляется и будет вычисляться на протяжении всего сеанса отладки при каждой остановке отладчика на этих точках останова. На данном этапе сеанса отладки эти выражения должны быть равны *null* и *false*, соответственно.
35. Нажмите F10, чтобы выполнить строку, в которой остановился отладчик, и проверьте всплывающую подсказку для url и оба выражения. Они должны содержать значения, соответствующие текущему контексту. На рис. 3-13 представлено, как все это выглядит в действии.

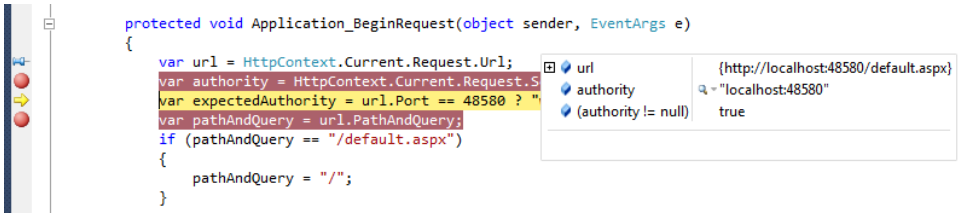


Рис. 3-13 Закрепленная всплывающая подсказка для url с двумя вычисленными выражениями

36. Замечательно иметь возможность просматривать мини-окно в том месте, где оно имеет значение – прямо там, где выполняется код – но оно перекрывает отлаживаемый исходный код. Окно всплывающей подсказки можно перетянуть в любую точку редактора кода. Пример показан на рис. 3-14.

Рис. 3-14 Убираем закрепленное окно подсказки с исходного кода

37. Закрепленное окно подсказки остается там, где оно закреплено, поэтому при переходе к другому файлу его не будет видно. Но в некоторых случаях окно всплывающей подсказки должно быть в поле зрения постоянно, например, для глобальных переменных, которые всегда в контексте, или для сценариев с использованием множества мониторов. Чтобы переместить окно подсказки, надо сначала открепить его, щелкнув вешку на панели управления всплывающей подсказки. Окно изменит цвет и станет желтым, что свидетельствует о возможности перемещать его в любое место, например, поверх Solution Explorer, на второй монитор, на рабочий стол или в любое другое окно. Пример можно увидеть на рис. 3-15.

Рис. 3-15 Открепленная всплывающая подсказка поверх Solution Explorer и рабочего стола Windows

Примечание Когда отладчик переходит к другому файлу или методу, и окно всплывающей подсказки не закреплено, оно может отображать элементы, не соответствующие текущему контексту. Тогда окно всплывающей подсказки будет выглядеть, как показано на рис. 3-16. Можно щелкнуть кнопку  , чтобы отладчик повторил попытку вычислить значение элемента. Но если элемент не имеет значения в данном контексте, весьма вероятно, что это ни к чему не приведет.

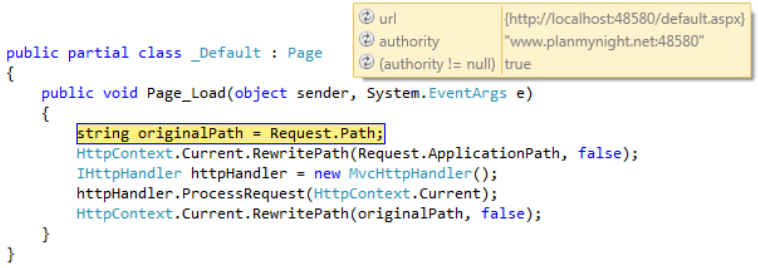


Рис. 3-16 Окно подсказки с элементами, не входящими в текущий контекст

Примечание При попытке закрепить окно всплывающей подсказки вне редактора будет выведено сообщение об ошибке, как показано на рис. 3-17.

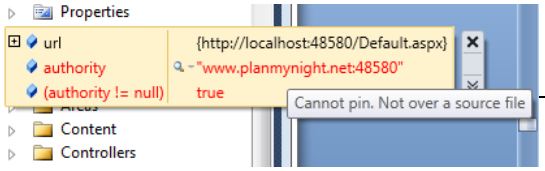


Рис. 3-17 Сообщение об ошибке, которое появляется при попытке закрепить окно всплывающей подсказки вне редактора кода

Примечание Используемый вами номер порта может отличаться от того, который представлен на приводимых здесь снимках экрана. Это нормально, поскольку Веб-сервер, включенный с Visual Studio, выбирает порт для использования случайным образом.

Примечание Закрепить можно и любой дочерний элемент закрепленного элемента. Например, разверните содержимое элемента url, нажав знак плюс (+). Вы увидите, что имеется возможность закрепить его дочерний элемент, как показано на рис. 3-18

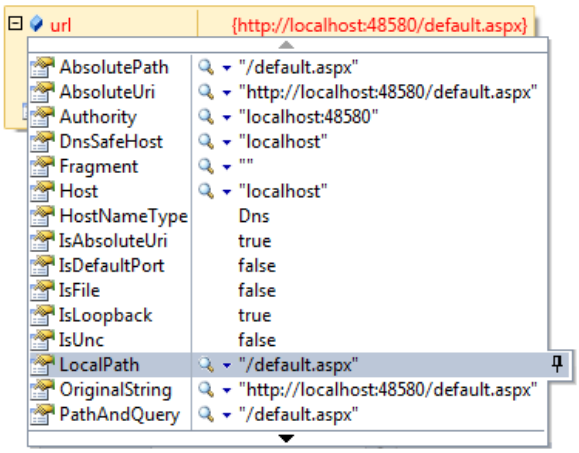


Рис. 3-18 Закрепленный дочерний элемент элемента url в окне всплывающей подсказки

38. Прежде чем остановить выполнение отладчика, вернитесь в Global.ascx.cs, если еще не сделали этого, и повторно закрепите окно всплывающей подсказки. После этого завершите сеанс отладки, щелкнув кнопку Stop Debugging (Остановить отладку) на панели инструментов отладки ( ) или нажав клавиши Shift+F5. Теперь если провести курсором мыши по голубой вешке в поле точек останова, можно увидеть значения последнего сеанса отладки. Такая функциональность является приятным улучшением по сравнению с окном контрольных значений. На рис. 3-19 показано, что должно быть выведено на экран.

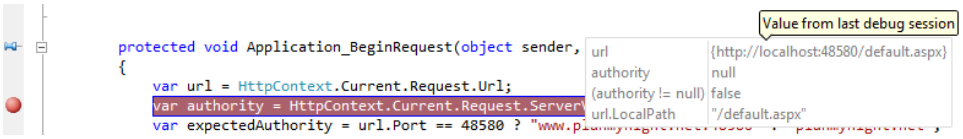


Рис. 3-19 Значения последнего сеанса отладки для закрепленной всплывающей подсказки

Примечание Как и точки останова, подсказки по данным могут экспортироваться или импортироваться. Для этого в меню Debug предусмотрены опции Export DataTips и Import DataTips, соответственно.

Использование отладчика минидампа

При эксплуатации реальных проектов часто возникают ситуации, когда группе поддержки продукта приходится работать с минидампом. Кроме описаний и шагов воспроизведения ошибок, это может быть единственным средством для отладки приложений заказчика. В Visual Studio 2010 отладка минидампа доработана и улучшена.

Visual Studio 2003 В Visual Studio 2003 для отладки управляемого кода или файлов минидампа приходилось использовать расширение – инструмент SOS, который загружался в отладчик с помощью окна интерпретации (Immediate). Отладчик подключался с разрешенной отладкой как управляемого, так и неуправляемого кода. При этом никакие данные не передавались ни в стек вызовов, ни в окно локальных переменных (Locals). При отладке файлов минидампа команды для инструмента SOS вводились в окне интерпретации. Отладка приложений, написанных неуправляемым кодом, выполнялась с использованием обычных окон и инструментов отладки. Познакомиться с этими вопросами более подробно или просто обновить их в памяти можно в колонке *Bug Slayer* (Убийца ошибок) журнала MSDN по адресу *http://msdn.microsoft.com/en-us/magazine/cc164138.aspx*.

Теперь рассмотрим, как был доработан отладчик минидампа. Для начала смоделируем сбой, на основании которого можно будет сформировать файл минидампа:

39. В Solution Explorer в проекте PlanMyNight.Web измените имя файла Default.aspx на **DefaultA.aspx** (обратите внимание на букву *A*, добавленную после слова «Default»).
40. Убедитесь, что убрали все точки останова из проекта. Для этого откройте окно Breakpoints и удалите все имеющиеся там точки останова любым из рассмотренных ранее в этой главе способом.
41. Нажмите F5, чтобы начать отладку приложения. В зависимости от производительности компьютера вскоре после завершения процесса сборки должно быть выведено необработанное исключение типа *HttpException*. Несмотря на то что в данном случае ошибка проста, последовательно пройдем все шаги создания и отладки файла минидампа. На рис. 3-20 показано, что должно отображаться на экране в данный момент.

Рис. 3-20 Необрабатываемое исключение, которое должно быть сформировано

42. Пора создавать файл минидампа для этого исключения. Перейдите в меню Debug и выберите Save Dump As (Сохранить дамп как), как показано на рис. 3-21. Вы должны видеть имя процесса, из которого было сформировано исключение. В данном случае это процесс Cassini или Personal Web Server в Visual Studio. Оставьте предлагаемое имя файла (WebDev.WebServer40.dmp) без изменений и сохраните файл на своем рабочем столе. Обратите внимание, что создание файла минидампа может занять некоторое время, потому что его размер будет около 300 MB.

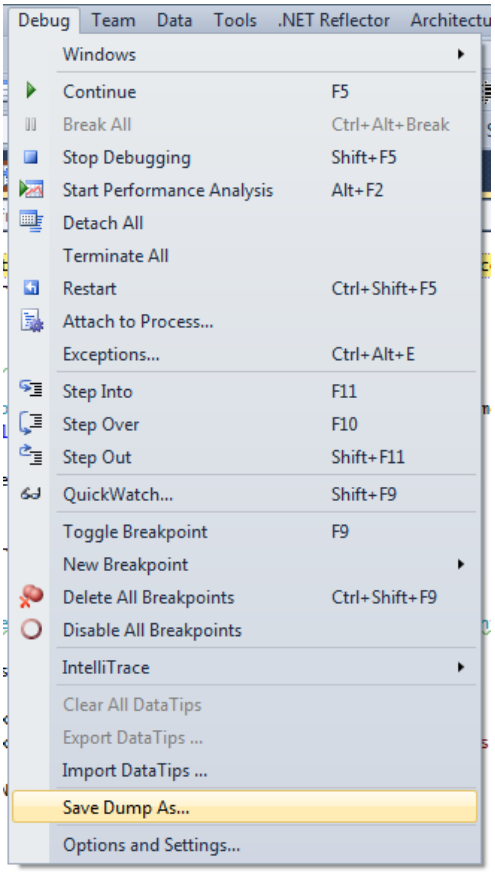
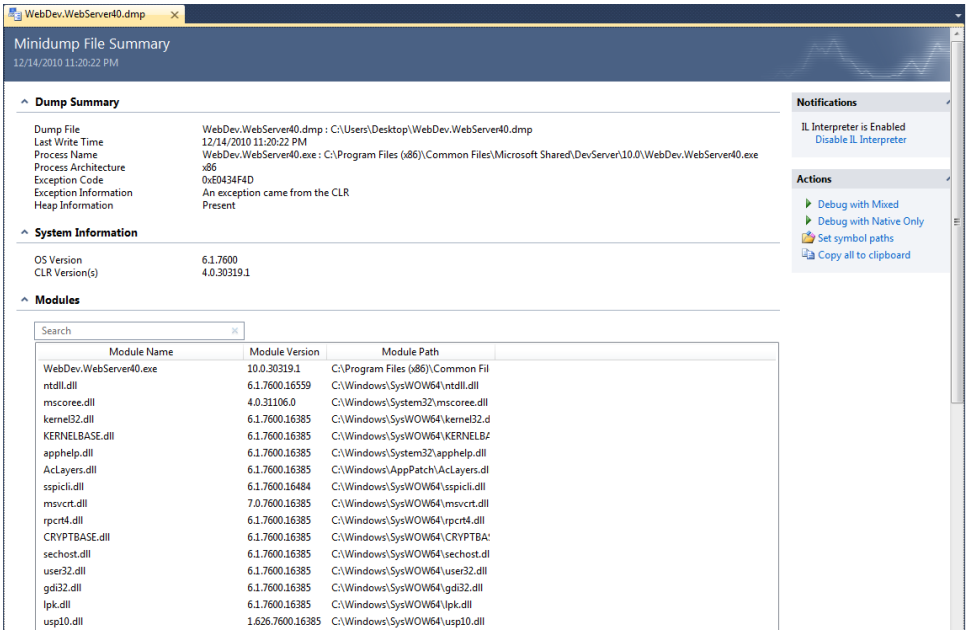


Рис. 3-21 Сохранение файла манидампа

43. Остановите отладку, нажав Shift+F5 или кнопку Stop Debugging.

44. Перейдите в меню File и закройте свое решение.
45. Чтобы загрузить свой файл манидампа WebDev.WebServer40.dmp, в меню File щелкните Open и выберите рабочий стол. При этом откроется страница Minidump File Summary (Сведения о файле минидампа), содержащая некоторые общие сведения об ошибке, которую мы пытаемся устранить (рис. 3-22). Еще до начала отладки эта страница предоставляет основные данные, такие как имя процесса, архитектура процесса, версия операционной системы, версия CLR, загруженные модули, а также некоторые действия, которые можно предпринять в этот момент. Прямо здесь можно задать пути к файлам символов. Очень удобно, что список модулей (Modules) включает версию и путь к модулю, что облегчает поиск символов и исходного кода. Используется CLR версия 4.0, т.е. отладку можно



выполнять в Visual Studio 2010.

Рис. 3-22 Страница сведений о файле минидампа

46. Чтобы начать отладку, в списке Actions (Действия) в правой части страницы Minidump File Summary щелкните ссылку Debug With Mixed (Отладка в смешанном режиме).
47. Практически сию минуту на экран должен быть выведен первый этап обработки исключения, как показано на рис. 3-23. В данном случае сообщается, какая ошибка произошла, хотя это не всегда так. Продолжим, нажав кнопку Break (Прервать).

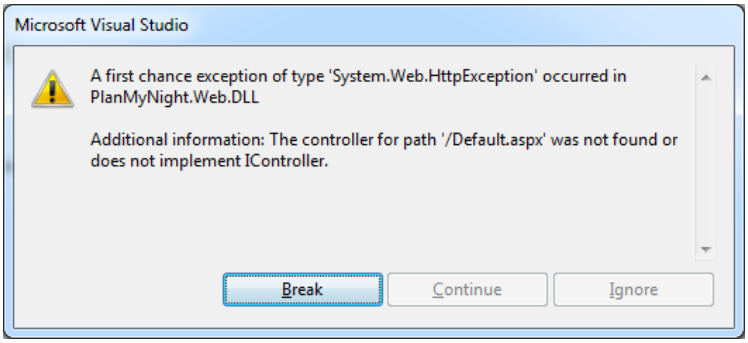


Рис. 3-23 Первый этап обработки исключения

48. Вы должны видеть подсвеченную зеленым строку, которая указывает на то, какая команда привела к формированию исключения. Если взглянуть на исходный код, в окне Autos (Автоматические значения) можно увидеть, что переменная *controllerExport* (Экспорт контроллера) имеет значение *null*. И прямо перед этим мы задали, что если эта переменная *null* и файл для загрузки не найден, должно формироваться исключение *HttpException*. В данном случае, как видно в окне Locals в переменной *controllerName* (Имя контроллера), должен загружаться файл Default.aspx. В окнах Locals и Autos можно также видеть множество других переменных, объектов и прочих элементов, отражающих текущий контекст. Здесь только один вызов относится к нашему коду, поэтому стек вызовов показывает, что весь код до и после этого вызова является внешним для нашего процесса. Если бы цепочка вызовов нашего

кода была длиннее, мы могли бы перемещаться назад и вперед по коду и просматривать переменные. На рис. 3-24 показано сводное представление всего вышеописанного.

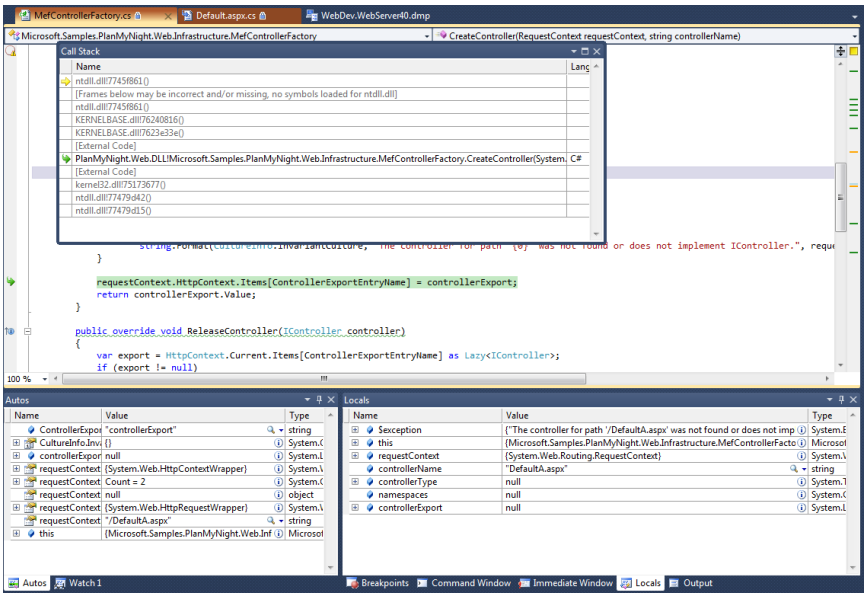


Рис. 3-24 Окна Autos, Locals и Call Stack и следующая команда к выполнению

49. Итак, ошибка найдена. Останавливаем отладку нажатием клавиш Shift+F5 или кнопки Stop Debugging. Чтобы исправить ошибку, опять загружаем решение PlanMyNight и исправляем имя файла **default.aspx**. Переходим в меню Build, выбираем Rebuild Solution (Выполнить повторную сборку решения) и выполняем повторную сборку решения. Затем нажимаем F5, и приложение должно опять исправно работать.

Преобразования Web.Config

Следующая возможность, хотя и незначительная, но она порадует многих разработчиков, потому что сбережет им время при отладке. Преобразования Web.Config позволяют сохранять файлы преобразований, отражающие разницу между средами отладки и производственной эксплуатации. Например, часто в разных средах используются разные строки подключения. Таким образом, создание файлов преобразований – ASP.NET предоставляет средства для изменения (преобразования) файлов web.config – обеспечит использование соответствующих строк для соответствующих сред. Более подробно о том, как это реализуется, рассказывает следующая статья на сайте MSDN: *http://go.microsoft.com/fwlink/?LinkId=125889*.

Разработка модульных тестов

Инфраструктура и средства модульного тестирования в Visual Studio 2010 Professional остались неизменными. В других редакциях Visual Studio 2010 в управление тестами и средства тестирования внесены действительно заметные изменения. В Visual Studio 2010 Premium и Visual Studio 2010 Ultimate предлагаются такие возможности, как модульные тесты пользовательского интерфейса, IntelliTrace и Microsoft Test Manager 2010. Все возможности Управления жизненным циклом приложения подробно рассматриваются на сайте MSDN в статье *http://msdn.microsoft.com/en-us/library/ee789810(VS.100).aspx*.

> **Visual Studio 2003** В Visual Studio 2003 возможности создания и выполнения модульных тестов ограничивались инструментами и инфраструктурами сторонних производителей, такими как nUnit, и другими коммерческими продуктами, создаваемыми партнерами Майкрософт.

В данной части главы мы просто покажем, как добавлять модульный тест для одного из классов приложения Plan My Night. Не будем тратить время на определение того, что такое модульный тест или что он должен включать. Лучше рассмотрим, как добавлять и выполнять тесты в Visual Studio 2010.

Добавим в приложение Plan My Night модульные тесты для надстройки Print Itinerary. Для создания модульных тестов откроем решение, находящееся в папке сопроводительных материалов книги. Если не помните, как это делается, можете вернуться к первой странице данной главы за инструкциями. Когда решение открыто, просто последовательно выполните следующие шаги:

50. В Solution Explorer разверните проект PlanMyNight.Web и папку Helpers. Щелкните двойным щелчком файл ViewHelper.cs, чтобы открыть его в редакторе кода. Взгляните на рис. 3-25 и убедитесь, что делаете все правильно.

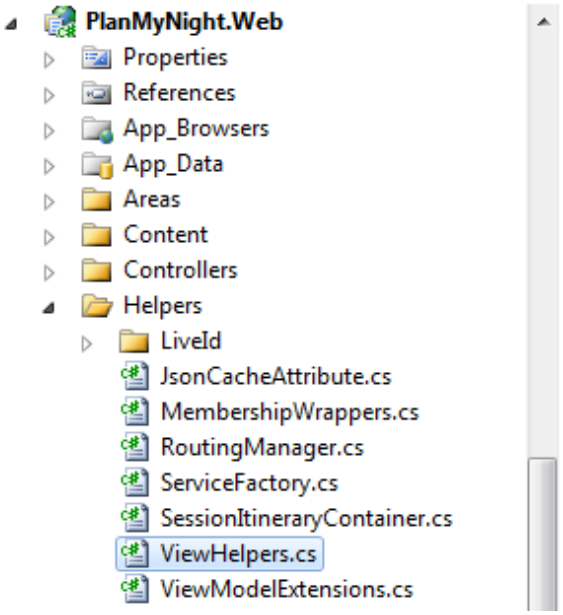


Рис. 3-25 Проект PlanMyNight.Web и файл ViewHelper.cs в Solution Explorer

51. Редактор кода позволяет добавлять модульные тесты двумя разными способами. Можно щелкнуть правой кнопкой имя класса или имя метода и выбрать в появившемся меню Create Unit Tests (Создать модульные тесты). Также можно перейти в меню Test (Тест) и выбрать New Test (Новый тест). Рассмотрим первый способ создания модульных тестов, при котором Visual Studio автоматически формирует некоторый исходный код. Щелкаем правой кнопкой метод *GetFriendlyTime* (Представить время в удобном для восприятия виде) и выбираем Create Unit Tests (рис. 3-26).

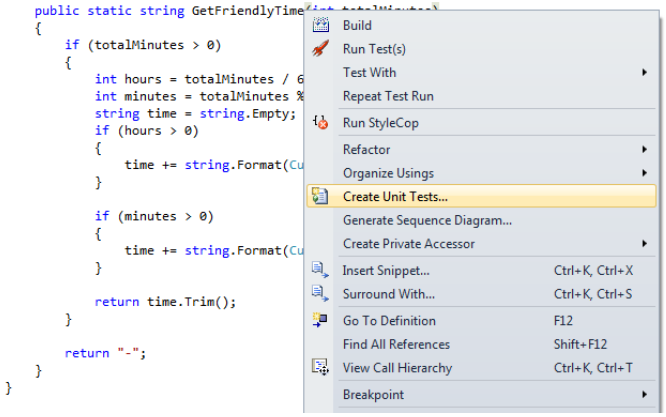


Рис. 3-26 Контекстное меню для создания модульных тестов

52. По нажатию Create Unit Tests на экран выводится диалоговое окно, в котором по умолчанию предлагается метод, выбранный нами в этом классе. Перейдем к полю с раскрывающимся списком внизу диалогового окна и выберем в нем, где будут создаваться модульные тесты. В данном случае это PlanMyNight.Web.Tests. Если есть необходимость создать новый проект для тестов, просто выбираем в этом списке Create A New Visual C# Test Project (Создать новый тестовый проект Visual C#). На рис. 3-27 представлено, как все это должно выглядеть.

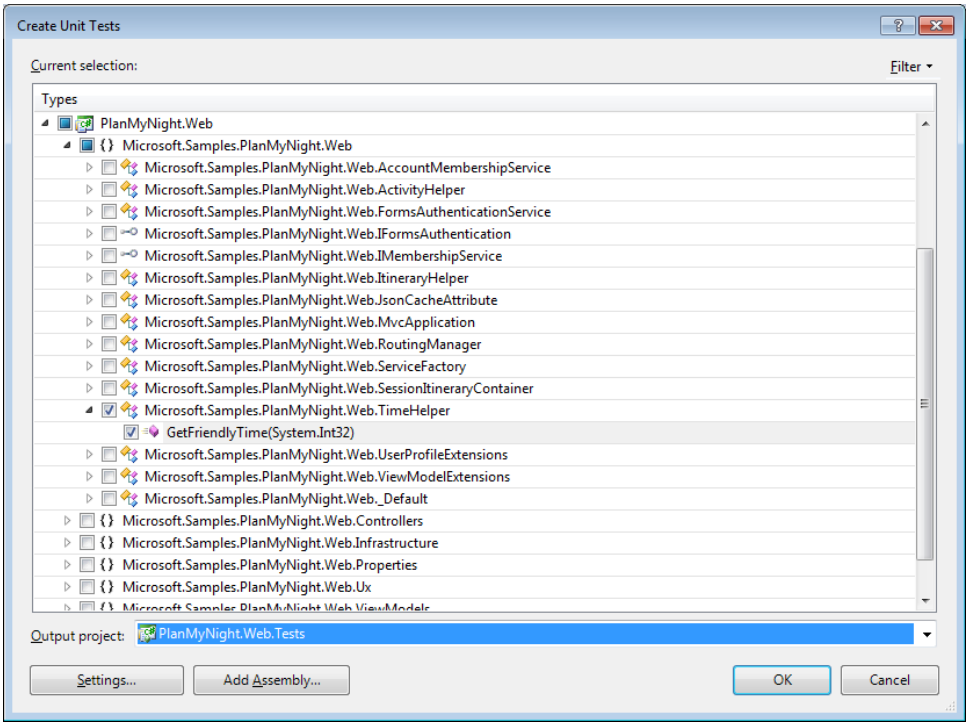


Рис. 3-27 Выбор метода, для которого требуется создать модульный тест

53. По щелчку OK диалоговое окно переключается в режим формирования тестового случая, отображается индикатор хода выполнения. По завершении этого процесса создается файл *TimeHelperTest.cs*, содержащий автоматически сформированные заглушки кода, которые мы будем дорабатывать.

54. Удалите этот метод и его атрибуты, потому что для него мы создадим три тестовых случая. Удаляем следующий код[1]:

```
/// <summary>
/// Тест для GetFriendlyTime
///</summary>
// TODO: Гарантированно обеспечить, чтобы атрибут UrlToTest содержал
// URL страницы ASP.NET (например, http://.../Default.aspx).
// Это необходимо, чтобы модульный тест выполнялся на Веб-сервере
// независимо от того, выполняется ли тестирование страницы,
// Веб-сервиса или WCF-сервиса.
[TestMethod()]
[HostType("ASP.NET")]
[AspNetDevelopmentServerHost("C:\\Users\\<имя пользователя>\\Documents\\Microsoft
Press\\Moving to Visual Studio 2010\\Chapter 3\\code\\PlanMyNight.Web", "/")]
[UrlToTest("http://localhost:48580/")]
public void GetFriendlyTimeTest()
{
    int totalMinutes = 0; // TODO: Инициализировать с соответствующим значением
    string expected = string.Empty; // TODO: Инициализировать с соответствующим
значением
    string actual; actual = TimeHelper.GetFriendlyTime(totalMinutes);
    Assert.AreEqual(expected, actual);
    Assert.Inconclusive("Verify the correctness of this test method[2].");
}
```

55. Добавляем три простых тестовых случая для проверки ключевых сценариев, используемых в PlanMyNight. Для этого вставляем следующий исходный код прямо под атрибутами метода, оставшимися после удаления блока кода в шаге 5:

```
[TestMethod]
public void ZeroReturnsSlash()
```

[1] Параметр для атрибута AspNetDevelopmentServerHost нужно изменить, чтобы получить правильный путь к папке с приложением (*прим. научного редактора*)

[2] Проверяем правильность этого тестового метода (*прим. переводчика*).

```
{
    Assert.AreEqual("-", TimeHelper.GetFriendlyTime(0));
}

[TestMethod]
public void LessThan60MinutesReturnsValueInMinutes()
{
    Assert.AreEqual("10m", TimeHelper.GetFriendlyTime(10));
}

[TestMethod()]
public void MoreThan60MinutesReturnsValueInHoursAndMinutes()
{
    Assert.AreEqual("2h 3m", TimeHelper.GetFriendlyTime(123));
}
```

56. В проекте PlanMyNight.Web.Tests создадим папку для решения и назовем ее **Helpers**. Перенесем файл TimeHelperTests.cs в эту папку. После этого проект должен выглядеть, как показано на рис. 3-28.

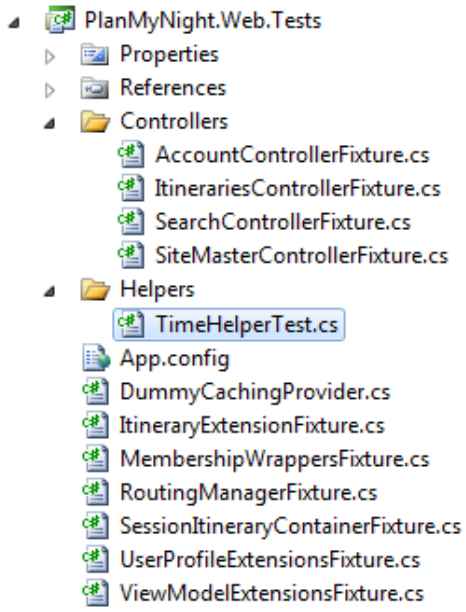


Рис. 3-28 TimeHelperTest.cs в папке Helpers

57. Пора выполнить только что созданные тесты. Чтобы выполнить только те тесты, которые мы только что создали, перейдем в редактор кода и наведем курсором на класс *public class TimeHelperTest*. Теперь можно либо зайти в меню Test, выбрать Run и затем Tests In Current Context (Тесты в текущем контексте); либо выполнить все то же самое посредством сочетания горячих клавиш CTRL+R, T (рис. 3-29).

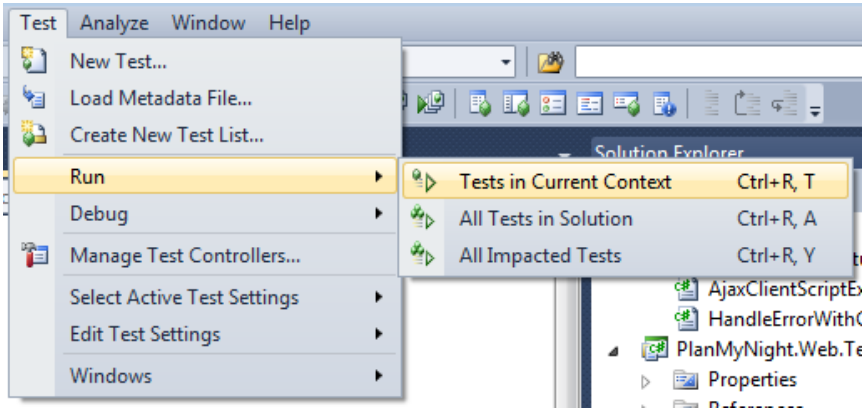


Рис. 3-29 Меню выполнения теста

58. Это действие обеспечит выполнение только трех созданных нами тестов. Внизу редактора должно появиться окно Test Results (Результаты тестирования) с результатами выполнения этих тестов (рис. 3-30).

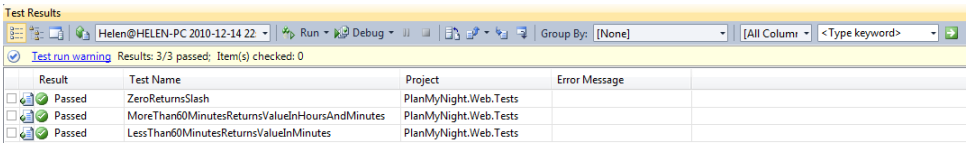


Рис. 3-30 Окно Test Results с результатами выполнения вновь созданных тестов

> **Дополнительные сведения** В зависимости от заданных параметров поведение при выборе опции Tests In Current Context может быть разным. Например, если выбран тестовый метод, такой как *ZeroReturnsSlash*, будет выполнен только этот тестовый случай. Но если щелкнуть вне класса теста, будут выполнены все тестовые случаи, что эквивалентно выбору опции All Tests In Solution (Все тесты решения).

Новое окно Threads

Появление многоядерных процессоров и многочисленных инструментов, предоставляемых языками программирования для работы с ними, обусловило новую проблему: отладка многопоточных приложений. Новое окно Threads (Потоки) обеспечивает разработчику возможность приостанавливать потоки и выполнять поиск артефактов по стеку вызовов, аналогично тому, как это происходило при использовании знаменитого Process Monitor от SysInternals (*http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx*). Окно Threads можно открыть во время отладки приложения из меню Debug, нажав опцию Windows (Окна), а потом Threads (Потоки). На рис. 3-31 представлено окно Threads во время отладки приложения Plan My Night.

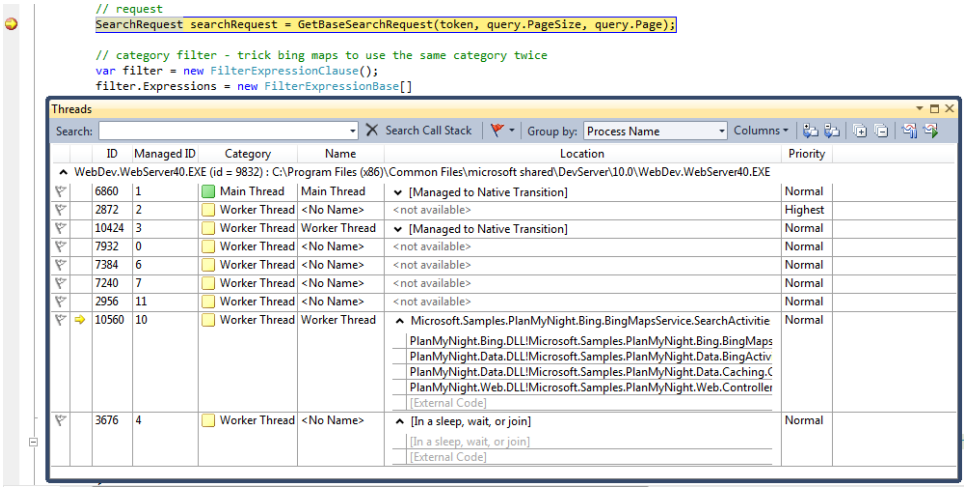


Рис. 3-31 Окно Threads во время отладки Plan My Night

Окно Threads позволяет приостанавливать потоки и вновь запускать их, когда требуется. Это может быть действительно полезным, когда необходимо изолировать конкретные эффекты. Можно выполнять отладку как управляемого, так и неуправляемого кода. Эта новая возможность отладчика в Visual Studio 2010, безусловно, понравится разработчикам, которые столкнуться с необходимостью отладки приложений, использующих потоки.

> **Visual Studio 2003** В Visual Studio 2003 окно Thread пребывало в зачаточном состоянии. Оно предоставляло возможность приостанавливать или переключаться между потоками. Узнать что-либо еще об этих потоках было сложно, поскольку никаких других сведений о них Visual Studio 2003 не предоставляла. Не было возможности фильтрации, группировки, поиска по стеку вызовов и расширения. Порядок столбцов был фиксирован. На то время понятие многоядерности уже существовало, но не использовалось так широко, как в наши дни. Более того, .NET Framework не предоставляла никаких средств, облегчающих разработку для многоядерных процессоров, как это делают сегодня .NET 4.0 и такие библиотеки, как PLINQ.

Заключение

В данной главе мы научились управлять сеансами отладки с помощью улучшенных возможностей работы с точками останова и использования новых техник проверки и визуализации данных. Также мы узнали, как с помощью отладки минидампа и новых инструментов решать проблемы, возникающие при работе приложений, не выводя их из эксплуатации. В этой главе было рассмотрено, как повысить качество кода

посредством модульных тестов и как сделать это в Visual Studio 2010 Professional. Компьютеры с множеством процессоров или с многоядерными процессорами в наши дни стали нормой, как и многопоточные приложения. Поэтому особенно приятным является тот факт, что Visual Studio 2010 предлагает специальные инструменты отладки для выявления проблем в многопоточных приложениях.

Наконец, в ходе главы было показано, насколько в Visual Studio 2010 Professional усовершенствован процесс отладки приложений в целом и какие инструменты получили в свое распоряжение профессиональные разработчики для отладки современных насыщенных разнообразнейшей функциональностью приложений. Прогресс по сравнению с Visual Studio 2003 очевиден. Примеры, рассмотренные в данной главе, лишь вкратце продемонстрировали, насколько выгоден с точки зрения экономии времени и средств переход к новой инфраструктуре отладки, но даже из них видно, Visual Studio 2010 не просто очередной этап, а значительная веха в развитии Visual Studio. Переход на Visual Studio 2010 обеспечит значительное повышение производительности разработки. С точки зрения возможностей отладки пропасть между Visual Studio 2003 и Visual Studio 2010 не так велика, чем между более ранними версиями. Объем сведений, предоставляемых отладчиком, способ визуализации данных, простота использования и настройки проектов – основные изменения, которые обеспечивают большую производительность разработчиков. В следующей главе будет показано, насколько упростилось развертывание Веб-приложений, их баз данных, настроек IIS и всех их конфигураций.

Различные редакции Visual Studio 2010 предоставляют целый ряд замечательных улучшений, касающихся отладки и тестирования. Особо из них выделяется возможность IntelliTrace – *http://msdn.microsoft.com/en-us/library/dd264915(VS.100).aspx* – которая доступна только в Visual Studio 2010 Ultimate и Microsoft Test Manager. IntelliTrace намного улучшает условия работы групп тестирования при использовании Visual Studio 2010 и Visual Studio 2010 Team Foundation Server – *http://msdn.microsoft.com/en-us/library/bb385901(VS.100).aspx*.

Глава 4

От 2003 к 2010: развертывание приложения

Данная глава посвящена техникам развертывания, и в ней рассматриваются:

- Развертывание Веб-приложения и базы данных SQL с использованием пакетов развертывания в Веб
- Развертывание Веб-приложения с использованием публикации одним щелчком мыши

Развертывание Веб-приложения всегда сопряжено с трудностями; так не должно быть, но так всегда происходит. Выполняется ли развертывание у провайдера, предоставляющего услуги размещения с помесячной оплатой, или в какой-либо центр обработки данных, суть процесса заключается обычно в перемещении файлов в некоторое местоположение с использованием протокола FTP, другого инструмента передачи данных или в виде .zip-архива. После этого, чтобы пользователи смогли успешно работать с Веб-приложением, разработчик или системный инженер должны выполнить еще и соответствующую настройку. В данной главе представлены самые последние и замечательные (при этом самые простые) способы развертывания Веб-приложения. Будет показано, что теперь этот процесс стал более управляемым и включает меньше ручных операций. Также предлагается возможность сравнить, как развертывание выполнялось в трех предыдущих версиях Microsoft Visual Studio.

Пакеты развертывания в Веб, предлагаемые Visual Studio 2010

В данной главе будет выполнено развертывание приложения-примера с использованием пакетов развертывания в Веб (Web Deployment Packages). Особое внимание мы уделим некоторым сложностям, которые возникали в связи с этим в предыдущих версиях Visual Studio. Воспользуемся примерами, упоминаемыми во введении к данной главе, и поэтапно рассмотрим все необходимые операции для развертывания приложения. После этого сравним, как реализовывалось развертывание в предыдущих версиях Visual Studio, начиная от 2003. Если сопроводительные материалы данной книги установлены в каталог по умолчанию, модифицированное приложение Plan My Night находится по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 4\Code. Щелкните двойным щелчком файл PlanMyNight.sln.

Начнем с примера развертывания у провайдера, предоставляющего услуги размещения на совместно используемом оборудовании с помесячной оплатой. Для развертывания Веб-приложения по такому сценарию необходимо выполнить пять основных шагов:

59. Передать файлы, необходимые для обеспечения работоспособности приложения, провайдеру, предоставляющему услуги размещения в Веб, с помощью FTP или их собственной панели управления для загрузки файлов. Если планируется передать только файлы, требуемые для исполнения, все файлы должны быть упорядочены, и провайдеру передаются только необходимые файлы.
60. После того как файлы скопированы, необходимо перейти к панели управления и убедиться, что все файлы располагаются в соответствующих местах. Далее должна быть выполнена настройка Internet Information Services (IIS) для приложения. В зависимости от специфики провайдера услуг размещения может понадобиться создать приложение в IIS, выбрать пул приложений и его тип и т.д.
61. Если приложение имеет базу данных SQL, ее необходимо создать и, возможно, заполнить некоторые таблицы доменов, выполнив SQL-сценарии.
62. Также может понадобиться задать настройки безопасности для SQL и приложения в целом.
63. Наконец, вероятно, придется внести изменения в файл web.config, указав в нем используемые серверы, и изменить некоторые опции конфигурации, связанные с базой данных, такие как строки подключения.

В крупных организациях часто приходится размещать приложения в центрах обработки данных и на серверах, к которым обычно вы не имеете ни физического, ни удаленного доступа посредством удаленного рабочего стола. В большинстве крупных организаций нет даже возможности пообщаться со специалистами, выполняющими развертывание, потому что обычно они работают в часы низкой нагрузки или в других временных зонах. Таким образом, единственное, что получают специалисты по развертыванию в свое распоряжение – это ваши надежные сценарии и документация по развертыванию. Сценарии развертывания должны быть просто «пуленепробиваемыми», документация по развертыванию должна быть проработана тщательнейшим образом. В документации не допускается никаких предполагаемых моментов, потому что с ней будут работать специалисты, имеющие очень слабое представление или вообще не имеющие никакого понятия о проекте.

Идеальным решением независимо от применяемой технологии развертывания является создание пакетов развертывания по мере разработки продукта. Если делать это после завершения всех работ по написанию кода, будет крайне тяжело добиться качественного и безошибочного результата. Здесь и пригодится Visual Studio 2010 с ее Web Deployment Tool (Инструмент развертывания в Веб).

Visual Studio 2010 и пакеты развертывания в Веб

На примере приложения Plan My Night и Visual Studio 2010 рассмотрим, как разработчик может выполнить развертывание Веб-приложения и гарантированно обеспечить его работоспособность после развертывания. Предлагаемые инструменты позволяют протестировать приложение, доработать его с помощью IIS на компьютере разработки и затем развернуть у провайдера сервиса размещения на совместно используемом оборудовании. При разработке приложения в крупной организации все эти операции оформляются в этап создания пакета развертывания, который добавляется в процессы MSBuild или TFSBuild.

Что было доступно до Visual Studio 2010

Без приобретения специализированного инструмента все варианты по развертыванию Веб-приложений в Visual Studio 2003 ограничивались следующими тремя:

64. Развертывание с использованием команды **xcopy**, которое также называли *XCOPY-развертыванием*.
65. Использование опции Copy Project (Копировать проект).
66. Использование опции Web Setup Project (Проект установки в Веб).

Варианты 1 и 2 использовались для развертывания простых приложений. Для более сложных случаев подходил только вариант 3 или приходилось использовать продукт InstallShield.

Первый вариант – это практически копирование необходимых файлов вручную с помощью команды **xcopy**. Вся настройка выполнялась посредством пакетных файлов, какого-либо языка сценариев или вручную.

Второй вариант не сильно отличался от первого, лишь требовалось принимать меньше решений по поводу того, какие файлы передавались на сервер и все процедуры передачи. Использовались расширения FrontPage или файловый ресурс; и можно было выбрать использовать только файлы, необходимые для выполнения приложения, или все файлы проекта или папки, в которой находилось приложение. По-прежнему требовалось выполнять настройку, как и в первом варианте развертывания.

Третий вариант позволял создавать MSI и выполнять достаточно детальную настройку Веб-приложения, но не предоставлял достаточных средств управления параметрами IIS, баз данных SQL и других вещей, используемых Веб-приложениями. И, кроме всего прочего, при таком варианте не обеспечивалась желаемая простота и надежность развертывания.

На рис. 4-1, 4-2 и 4-3 представлены три наиболее распространенных метода развертывания Веб-приложения в Visual Studio 2003. Рис. 4-1 иллюстрирует первый вариант из вышеперечисленных.

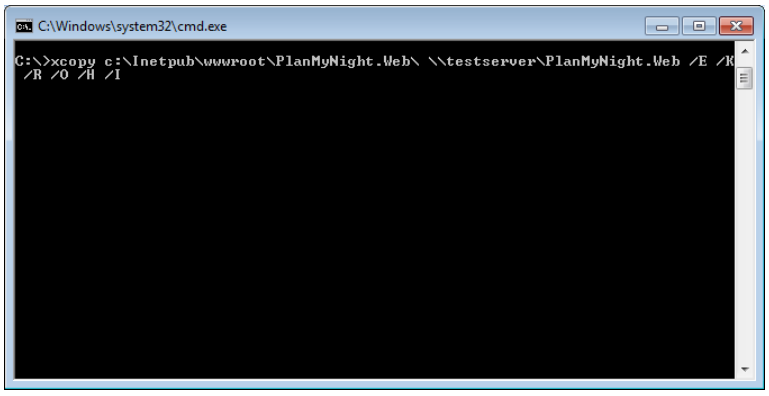


Рис. 4-1 XCOPY-развертывание

Рис. 4-2 иллюстрирует второй вариант, т.е. метод Copy Project.

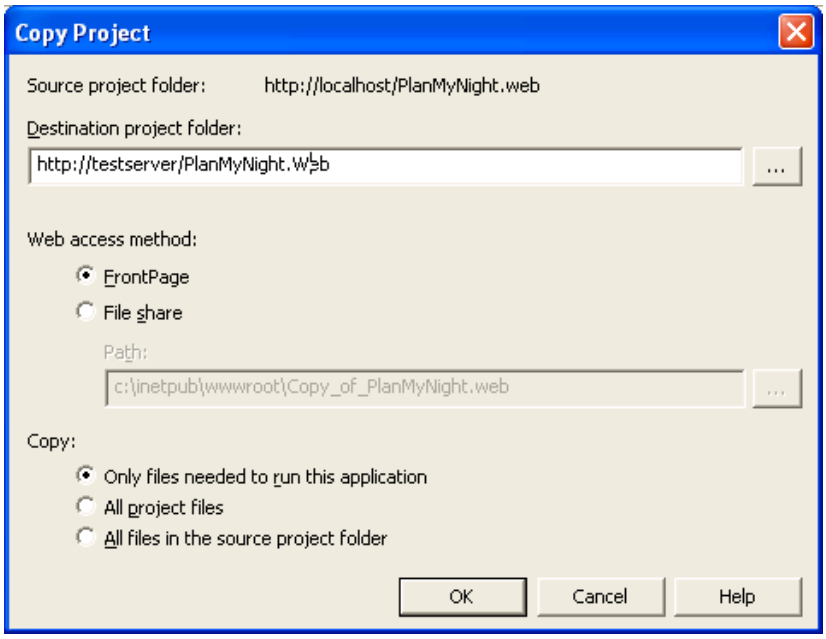


Рис. 4-2 Диалоговое окно Copy Project

Наконец, третий вариант, создание проекта Web Setup Project, представлен на рис. 4-3.

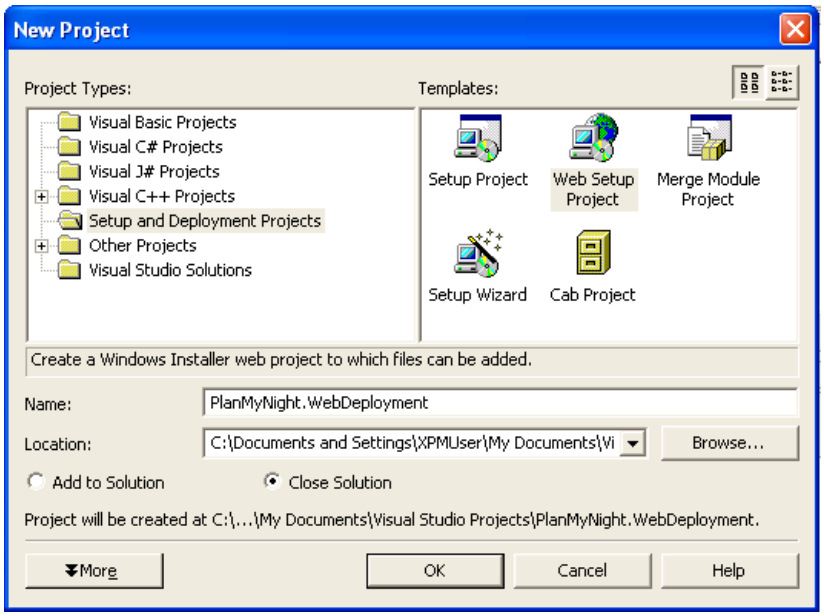


Рис. 4-3 Диалоговое окно New Project с предлагаемой опцией Web Setup Project

В Visual Studio 2005 и 2008 начали появляться новые концепции, касающиеся Веб-приложений. Начиная с Visual Studio 2005, Веб-приложения можно было создавать двумя разными способами: как Веб-сайт непосредственно в файловой системе (на жестком диске и без файла проекта) либо как проект Web Application Project (Веб-приложение), т.е. как обычный проект любого другого типа.

> **Примечание** Проект Web Application Project стал частью Visual Studio 2005 в виде надстройки через несколько месяцев после выхода продукта. Пользователи запросили Майкрософт предоставить оба варианта: Веб-сайт на жестком диске и проект Web Application Project, аналогично тому как это было в уже проверенном на практике шаблоне проекта, поставляемом в составе Visual Studio 2003. Этими проектами проще управлять в среде предприятия, и они могут быть без труда интегрированы с MSBuild.

Проект Web Deployment Project (Проект развертывания в Веб) в Visual Studio 2005 и 2008 был реализован как надстройка. Набор возможностей в обеих версиях был практически аналогичен, только в Visual Studio 2008 устранили имеющиеся дефекты.

Проект Web Deployment Project не вносит изменения в Веб-сайт или Web Application Project. Он использует их как входные данные и создает совершенно другой проект. Кстати, исходный код никогда не затрагивается, создается новый проект с необходимыми файлами на основании выбранных параметров конфигурации. Проект Web Deployment Project – это всего лишь файл проекта и ничего более. С помощью

нескольких диалоговых окон он позволяет разработчику определить, как и где должно быть развернуто приложение.

Затем во время сборки с помощью, главным образом, двух утилит эти настройки трансформируются в соответствующий Веб-сайт. Эти две утилиты командной строки называются aspnet_compiler и aspnet_merge. По сути, первая компилирует проекты, формируя Веб-приложение, и вторая консолидирует полученный результат и копирует файлы либо в папку, либо в виртуальную папку в IIS.

Для каждой обычной конфигурации (версия для отладки и выпускаемая версия) и любой пользовательской конфигурации может существовать одна конфигурация развертывания. На рис. 4-4 показан начальный этап создания проекта Web Deployment Project.

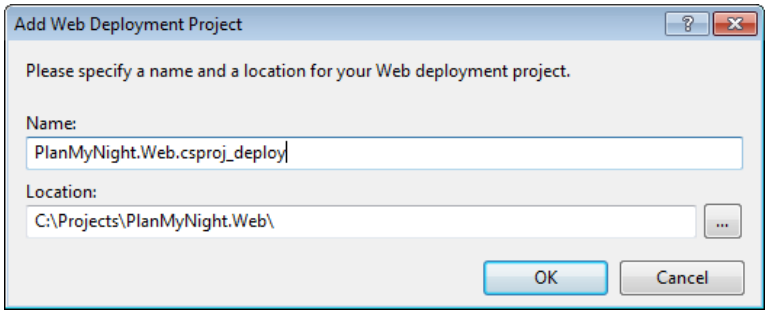


Рис. 4-4 Диалоговое окно Add Web Deployment Project

Теперь рассмотрим свойства Web Deployment Project в Visual Studio 2008. На рис. 4-5 можно увидеть предлагаемые возможности.

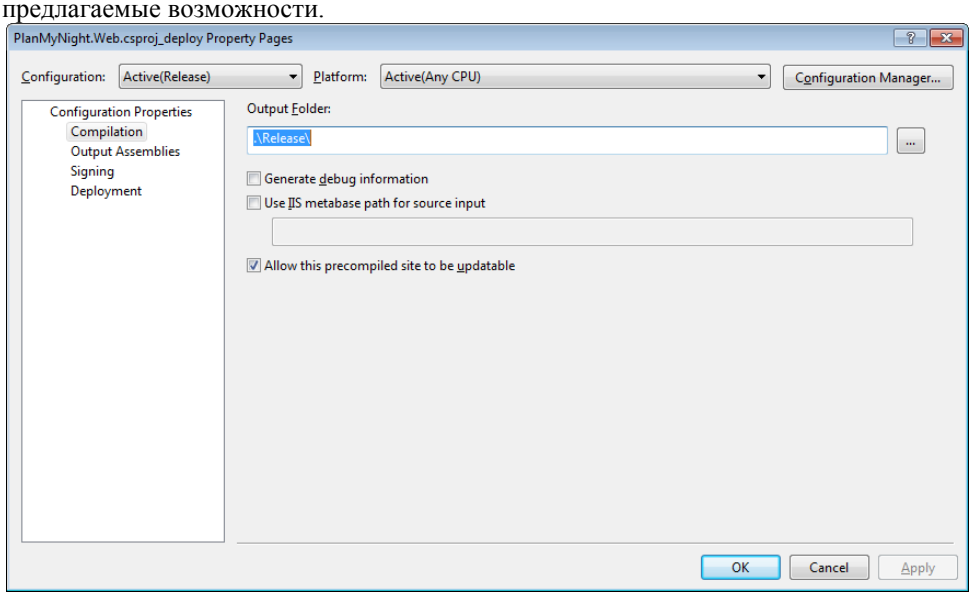


Рис. 4-5 Страница свойств Web Deployment Project

Поскольку это обычный файл проекта с директивами MSBuild, для него можно изменить этапы, предшествующие сборке, и выполняемые после нее, а также выполнить дополнительные сценарии. Для этого необходимо редактировать файл проекта и изменить соответствующие цели MSBuild. Итак, на рис. 4-6 представлен фрагмент файла проекта Web Deployment Project для приложения PlanMyNight в Visual Studio 2008.

```
<!--
  Microsoft Visual Studio 2008 Web Deployment Project
  http://go.microsoft.com/fwlink/?LinkID=104956

-->
<Project ToolsVersion="3.5" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003'
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProductVersion>9.0.21022</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{F1CFC25B-41B8-4ADA-AE6C-73F0A277C060}</ProjectGuid>
    <SourceWebPhysicalPath>..\PlanMyNight.Web</SourceWebPhysicalPath>
    <SourceWebProject>{12D3746F-9855-403B-A18B-86766681B38B}|PlanMyNight.Web\PlanMyNight.Web.csproj</SourceWeb
    <SourceWebVirtualPath>/PlanMyNight.Web.csproj</SourceWebVirtualPath>
    <TargetFrameworkVersion>v3.5</TargetFrameworkVersion>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <OutputPath>.\Debug</OutputPath>
    <EnableUpdateable>true</EnableUpdateable>
    <UseMerge>true</UseMerge>
    <SingleAssemblyName>PlanMyNight.Web.csproj_deploy</SingleAssemblyName>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <DebugSymbols>false</DebugSymbols>
    <OutputPath>.\Release</OutputPath>
    <EnableUpdateable>true</EnableUpdateable>
    <UseMerge>true</UseMerge>
    <SingleAssemblyName>PlanMyNight.Web.csproj_deploy</SingleAssemblyName>
  </PropertyGroup>
  <ItemGroup>
  </ItemGroup>
  <Import Project="$(MSBuildExtensionsPath)\Microsoft\WebDeployment\v9.0\Microsoft.WebDeployment.targets" />
  <!-- To modify your build process, add your task inside one of the targets below and uncomment it.
       Other similar extension points exist, see Microsoft.WebDeployment.targets.
  <Target Name="BeforeBuild">
  </Target>
```

Рис. 4-6 Исходный код файла проекта Web Deployment Packages

Web Deployment Project являлся громадным улучшением по сравнению с возможностями, предлагаемыми Visual Studio 2003, но, тем не менее, он по-прежнему не обеспечивал достаточной простоты, не предусматривал всего необходимого и не давал требуемой мощи. Он был лишь этапом. Настоящим прорывом стали пакеты Web Deployment Packages – новый тип проекта развертывания Веб-приложений в Visual Studio 2010.

Что такое Web Deployment Packages?

Пакет развертывания в Веб – это.zip-архив, включающий все необходимые файлы и метаданные для установки приложения в IIS, копирования файлов приложения в заданное местоположение, конфигурации разных приложений в IIS, установки связанных ресурсов, таких как ресурсы локализации, сертификаты, параметры реестра, установки сборок в глобальный кэш сборок и, наконец, настройки баз данных.

Эти пакеты устанавливаются на конечный сервер с помощью инструмента msdeploy. Самую последнюю информацию об этом инструменте можно найти по адресу *http://blogs.iis.net/msdeploy/default.aspx*. Аналогом данного решения для клиентского рабочего стола является MSI и установщик Windows.

> **Примечание** Существуют и другие замечательные решения, такие как InstallShield и Wix Toolset. У обоих есть определенные достоинства и недостатки, и оба этих решения подходят для других типов приложений. Инструмент msdeploy просто более специализирован и, следовательно, более прост в применении. На момент написания данной главы (май 2010) Wix 3.5 еще не выпущен, и изменение планов потенциально может привести к изъятию из выпускаемой версии дополнительного действия для IIS 7. Если его все-таки включат в поставку, он появится не ранее июля 2010. (Все новости мира Wix можно найти в блоге Роба Меншинга (Rob Mensching) по адресу *http://robmensching.com/blog/*.) Поэтому пока я бы не рекомендовал использовать пакет инструментов Wix, исключением является установка на IIS 6. InstallShield уже выпущен, поддерживает Visual Studio 2010 и, несомненно, позволяет создавать хорошие пакеты для развертывания Веб-приложений. Но этот инструмент не бесплатный.

В Visual Studio 2010 для создания пакетов предусмотрена вкладка Package/Publish Web (Упаковать/опубликовать в Веб) на странице свойств проекта. Задавая параметры на этой вкладке, мы определяем, что будет включено в пакет развертывания. Рассмотрим этот процесс шаг за шагом.

67. Откройте решение PlanMyNight. Щелкните правой кнопкой проект PlanMyNight.Web и в появившемся меню выберите Package/Publish Settings (Настройки упаковки/публикации), как показано на рис. 4-7.

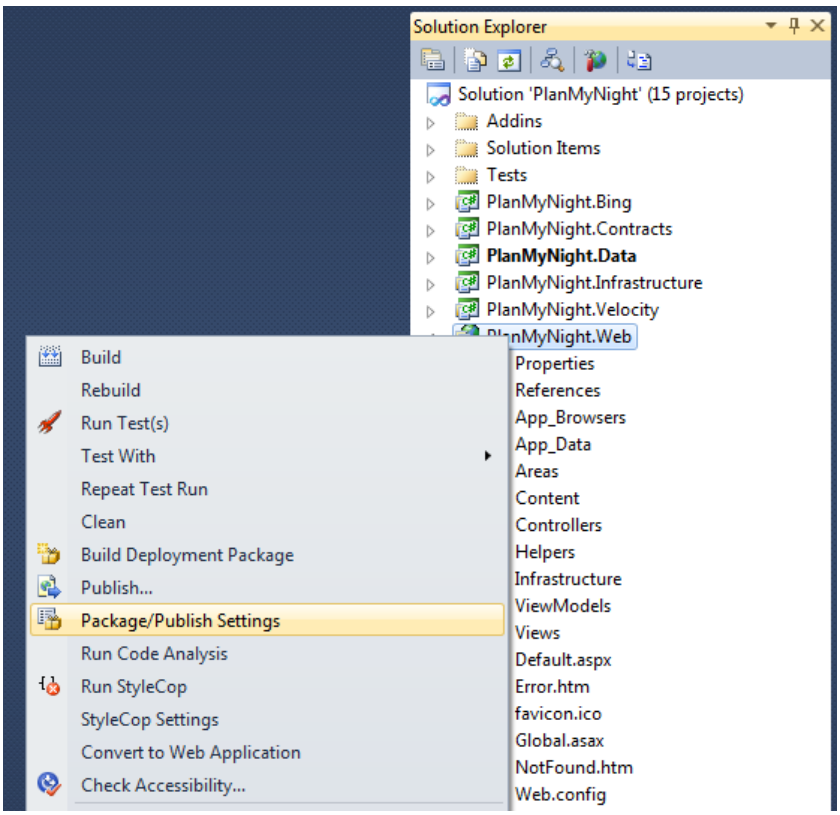


Рис. 4-7 Опция Package/Publish Settings

68. На рис. 4-8 представлено содержимое вкладки Package/Publish Web и показано, какие настройки необходимо выполнить для приложения PlanMyNight.Web.

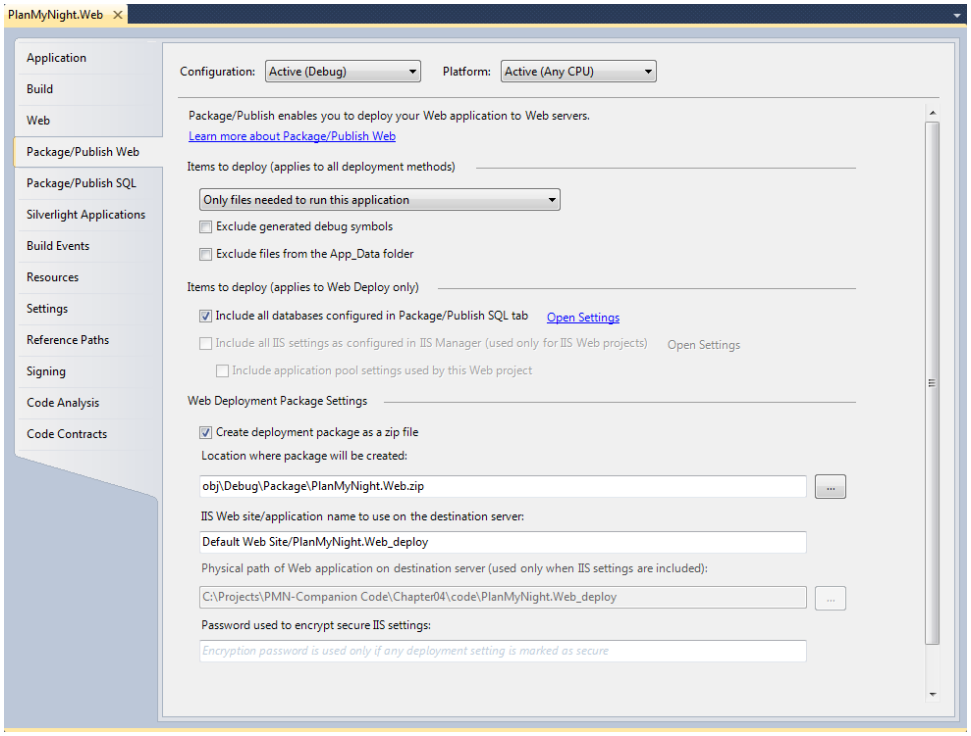


Рис. 4-8 Вкладка Package/Publish Web

69. Аналогично на вкладке Package/Publish SQL можно найти настройки, касающиеся создания пакета для базы данных. Щелкните вкладку Package/Publish SQL или ссылку Open Settings (Открыть настройки) на вкладке Package/Publish Web.

70. Щелкните кнопку Import From Web.Config (Импортировать из Web.Config) и затем скопируйте строку подключения к базе данных-источнику в строку подключения к базе данных-получателю. Настройки SQL должны быть аналогичны представленным на рис. 4-9. Сохраните файл.

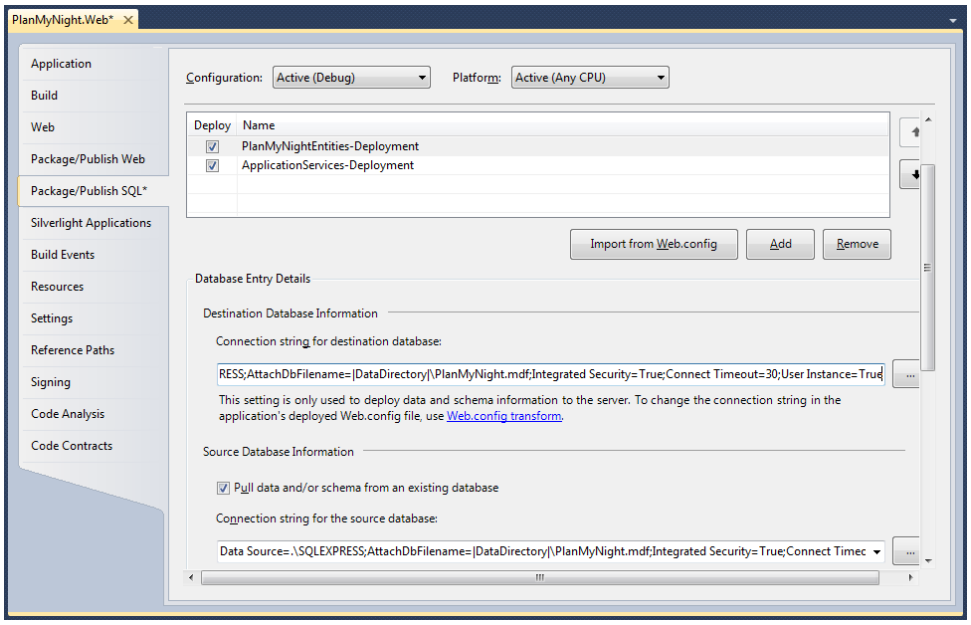


Рис. 4-9 Вкладка Package/Publish SQL

71. Пора выполнить сборку пакета. Щелкните правой кнопкой имя проекта и выберите Build Deployment Package (Сборка пакета развертывания). Начнется процесс сборки приложения. Результат этого процесса по его завершению станет исходными данными для создания пакета. Если все проходит успешно, пакет создается в каталоге, который задан в настройках пакета.
72. Папка пакета должна включать .zip-файл пакета; командный файл, вызывающий Web Deploy (Развертывание в Веб) для упрощения установки пакета из командной строки; файл SetParameters.xml со всеми параметрами, передаваемыми в Web Deploy; и файл SourceManifest.xml с параметрами, которые использовались Visual Studio 2010 при создании пакета.

Только что мы создали пакет, используя Visual Studio 2010. Но пакеты также могут создаваться с помощью MSBuild в командной строке или MSBuild с использованием Windows PowerShell или TFSBuild. Теперь этот пакет и командный файл можно размещать на доступном сервере. Однако публикация на сервере провайдера, поставляющего услуги размещения на совместно используемых ресурсах, или центра данных выполняется иначе. И здесь на помощь приходит публикация одним щелчком мыши или One-Click.

Публикация One-Click

Публикация One-Click – это инструмент Visual Studio 2010, позволяющий развертывать пакет с использованием различных технологий. Наиболее важно то, что он может публиковать пакет, созданный в предыдущем разделе, посредством Web Deploy. Также могут использоваться FTP и FrontPage Extension. Рассмотрим, как выглядит Publish Profile (Профиль публикации). Для этого щелкнем правой кнопкой Веб-проект PlanMyNight.Web и выберем Publish (Опубликовать). На экране должно появиться диалоговое окно, представленное на рис. 4-10, со сведениями Publish Profile.

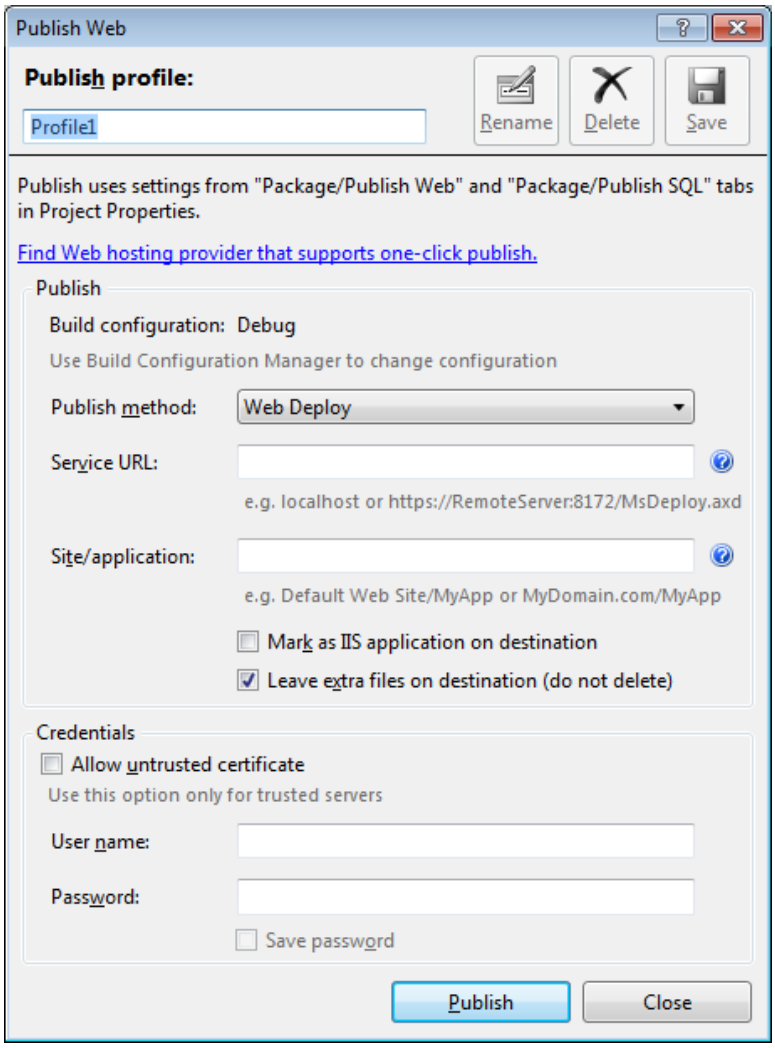


Рис. 4-10 Диалоговое окно Publish Profile

Если публикация выполняется на сервер провайдера, поставляющего услуги размещения на совместно используемых ресурсах, на серверах которого установлен Web Deploy, провайдер предоставит вам URL сервиса для использования. Он будет выглядеть примерно так: https:<адрес сайта>:8172/MsDeploy.axd. Если публикация выполняется во внутренней сети, просто задается имя сервера. Наконец, в поле site/application (сайт/приложение) должны быть указаны ИмяВебСайтаIIS/ВебПриложениеIIS. Нижняя часть представленного на рис. 4-10 диалогового окна используется для введения учетных данных, если таковые необходимы во внутренней сети или для предоставления провайдеру Веб-хостинга.

> **Примечание** Чтобы испытать этот инструмент в действии, просто удалите свою базу данных, удалите Веб-приложение IIS и настройки и затем выполните публикацию, используя этот инструмент.

На самом деле возможности намного шире, чем предлагает пользовательский интерфейс. Интеграция развертывания в TFSBuild или MSBuild позволяет менять многие параметры на разных этапах процесса.

В данной главе мы рассмотрели стандартные опции. Несомненно, Visual Studio 2010 предлагает множество других способов развертывания других типов приложений.

Заключение

В данной главе дан обзор различных способов развертывания Веб-приложений. Мы вспомнили, как это происходило в трех предыдущих версиях Visual Studio, и узнали, что в Visual Studio 2010 этот процесс упростился и стал намного более управляемым. Web Deploy – новая технология, позволяющая Веб-разработчикам искусно и без труда подготавливать и развертывать сложные установки и базы данных, обеспечивая при этом возможность расширения. Уже по приведенным небольшим примерам можно убедиться, что развертывание ПО выполняется уверенно и своевременно. Это значительный прогресс по сравнению с XCOPY-развертыванием, с которым приходилось работать большинству из нас на заре ASP.NET.

Чтобы идти в ногу со всеми изменениями в технологиях msdeploy, не забывайте посещать блог группы разработки по адресу *http://blogs.iis.net/msdeploy/default.aspx*. Набор инструментов Wix Toolset также хорошо интегрируется с Visual Studio и, следовательно, является еще одной замечательной альтернативой.

Часть II

Переход от Microsoft Visual Studio 2005 к Visual Studio 2010

Авторы Патрис Пелланд, Кен Хайнс и Паскаль Паре

В данной части:

От 2005 к 2010: Бизнес-логика и данные (Паскаль)

От 2005 к 2010: Проектирование восприятия и поведения (Кен)

От 2005 к 2010: отладка приложения (Патрис)

Глава 5

От 2005 к 2010: бизнес-логика и данные

В данной главе рассматривается

- Применение Entity Framework (EF) для создания слоя доступа к данным с использованием существующей базы данных или модель-ориентированного подхода
- Создание типов сущностей в дизайнере Entity Data Model (Модель сущность-данные) с использованием POCO шаблонов в ADO.NET Entity Framework
- Получение данных от Веб-сервисов
- Кэширование данных с использованием Windows Server AppFabric (ранее известной под кодовым названием «Velocity»)

Архитектура приложения

С помощью приложения PlanMyNight (PMN) пользователь может составлять программы своих мероприятий и делиться ими с остальными. Данные хранятся в базе данных Microsoft SQL Server. Программы составляются на основании обращений к Веб-сервисам Bing Maps

Рассмотрим исходную блок-схему модели данных приложения (рис. 5-1).

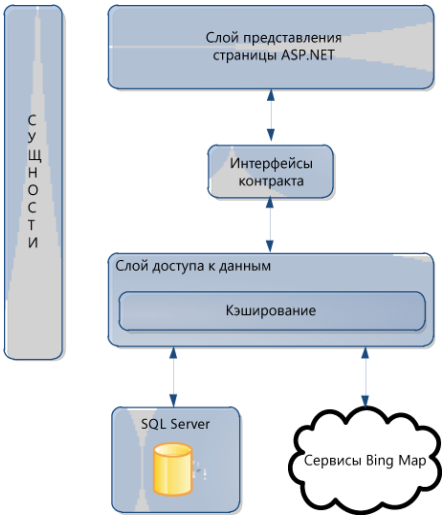


Рис. 5-1 Архитектурная диаграмма приложения PlanMyNight

Описание контрактов и классов сущностей, свободных от каких-либо ограничений, налагаемых методом хранения, позволяет объединять их в сборки, не зависящие от метода хранения. Такой подход обеспечивает четкое разделение слоев представления и доступа к данным.

Опишем интерфейсы контрактов основных компонентов приложения PMN:

- *IItinerariesRepository* (Хранилище маршрутов) – интерфейс нашего хранилища данных (базы данных Microsoft SQL Server).
- *IActivitiesRepository* (Хранилище действий) позволяет выполнять поиск мероприятий (используя Веб-сервисы Bing Maps).
- *ICachingProvider* (Поставщик кэширования) обеспечивает интерфейс кэширования данных (кэширование ASP.NET или кэширование AppFabric Windows Server).

Примечание Это неполный список контрактов, реализованных в приложении PMN.

PMN сохраняет программы мероприятий пользователя в базе данных SQL. Остальные пользователи смогут комментировать и рейтинговать программы друг друга. На рис. 5-2 представлены таблицы, используемые приложением PMN.

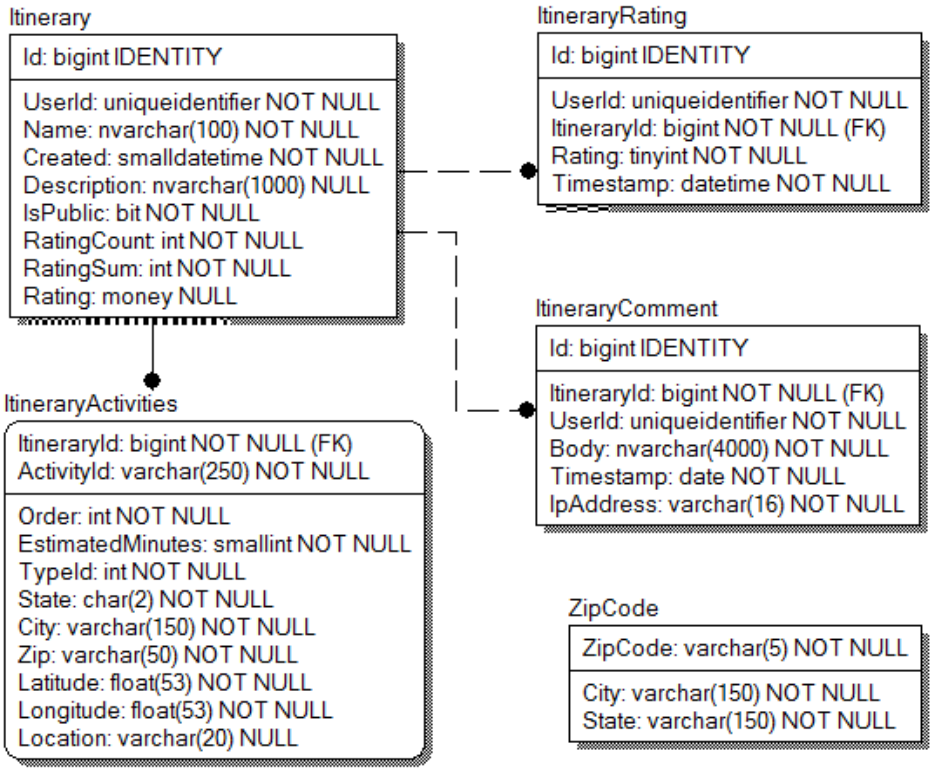


Рис. 5-2 Схема базы данных PlanMyNight

> **Важно** В приложении PlanMyNight безопасное хранение учетных данных пользователей обеспечивается посредством возможностей членства ASP.NET. Таблицы хранилища пользователя на Рис. 5-2 не показаны. Более подробную информацию об этих возможностях можно найти на сайте MSDN в разделе ASP.NET 4 – Introduction to Membership (ASP.NET 4 – введение в членство) по адресу *http://msdn.microsoft.com/en-us/library/yh26yfzy(VS.100).aspx*.

> **Примечание** Таблица ZipCode (Почтовый индекс) используется как хранилище справочных ресурсов и предоставляет список доступных почтовых индексов и городов. Благодаря этому обеспечивается функциональность автозаполнения запроса поиска пользователя в приложении.

Данные Plan My Night в Microsoft Visual Studio 2005

Visual Studio 2005 предоставляет все необходимые инструменты для создания приложения, поэтому написание PlanMyNight в ней не составило бы никакого труда. Однако некоторые применяемые тогда технологии требовали написания намного большего объема кода для достижения тех же целей.

Рассмотрим, как создавался бы требуемый слой доступа к данным в Visual Studio 2005. Один из возможных подходов – непосредственно использовать *DataSet* (Множество данных) или *DataReader* (Средство чтения данных) ADO.NET (рис. 5-3). Такое решение предлагает большую гибкость, поскольку обеспечивает полный контроль над доступом к базе данных. С другой стороны, оно имеет некоторые недостатки:

- Необходимо знать синтаксис SQL.
- Все запросы специализированы. Любое изменение в требованиях или в таблицах приводит к необходимости обновления всех соответствующих запросов.
- Приходится сопоставлять свойства классов сущностей, используя имя столбца, что довольно утомительно и чревато многочисленными ошибками.
- Приходится самостоятельно управлять отношениями между таблицами.

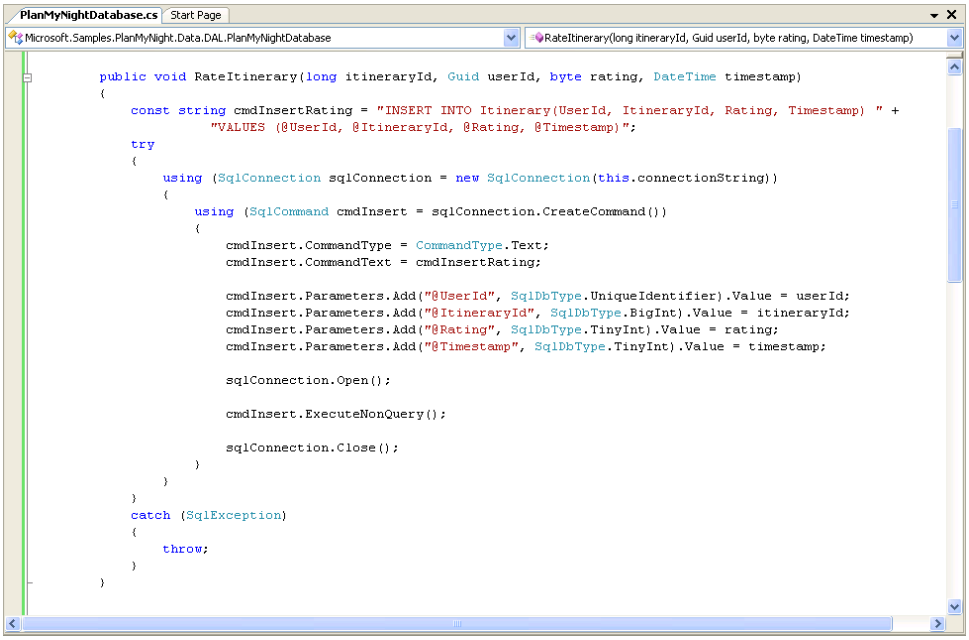


Рис. 5-3 Запрос Insert ADO.NET

Другой подход – использование дизайнера *DataSet*, предоставляемого Visual Studio 2005. При наличии базы данных с таблицами PMN, можно было бы использовать TableAdapter Configuration Wizard (Мастер настройки TableAdapter) для импорта таблиц базы данных, как показано на рис. 5-4. Автоматически сформированный код предлагает типизированный *DataSet*. К преимуществам относится контроль типов во время разработки, что обеспечивает автоматическое завершение выражений средствами редактора исходного кода. Но, тем не менее, данный подход имеет и недостатки:

- По-прежнему требуется знание синтаксиса SQL, хотя прямо из дизайнера *DataSet* имеется возможность доступа к построителю запросов.
- По-прежнему приходится создавать специализированные SQL-запросы, чтобы выполнять все требования контрактов данных.
- Нет возможности управления автоматически сформированными классами. Например, добавление или удаление запроса к таблице из *DataSet* приведет к повторной сборке автоматически сформированных классов *TableAdapter* и может изменить индекс, используемый для запроса. Это усложняет задачу по написанию предсказуемого кода с использованием таких автоматически формируемых классов.
- Автоматически формируемые классы, ассоциированные с таблицами, включают код, зависящий от метода хранения, поэтому придется создавать другое множество простых сущностей и копировать данные туда. Это означает больший объем обработки и более активное использование памяти.

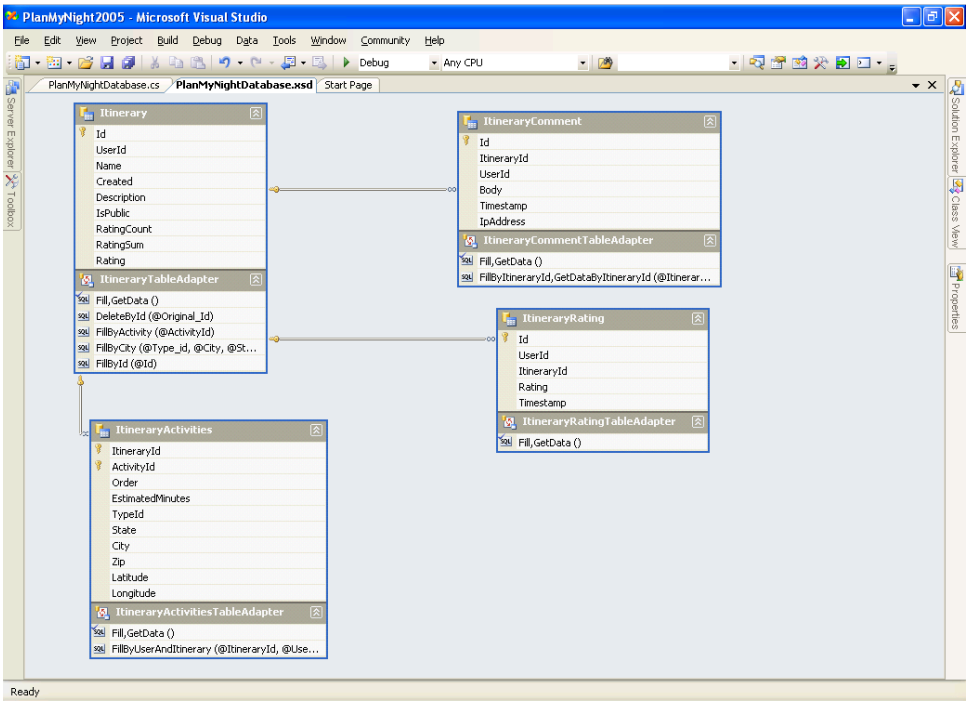


Рис. 5-4 Дизайнер DataSet в Visual Studio 2005

В следующих разделах данной главы будут рассмотрены некоторые новые возможности, предоставляемые Visual Studio 2010. Благодаря им создание слоя доступа к данным приложения PMN связано с меньшим объемом написания кода, предоставляется больший контроль над автоматически формируемым кодом, обслуживать и расширять его становится намного проще.

Работа с данными в Visual Studio 2010 с использованием Entity Framework

ADO.NET Entity Framework (EF) позволяет без труда создавать слой доступа к данным приложения через абстрагирование работы с данными от базы данных и создание модели, приближенной к бизнес-требованиям приложения. EF была существенно доработана и улучшена в выпущенной версии .NET Framework 4.

> *Дополнительные сведения Множество ресурсов, посвященных ADO.NET Entity Framework в .NET 4, предлагает Центр решений по работе с данными (Data Developer Center) на сайте MSDN (http://msdn.microsoft.com/en-us/data/aa937723.aspx).*

Возьмем приложение PlanMyNight в качестве примера использования некоторых возможностей EF. В следующих двух разделах демонстрируются два разных подхода к автоматическому формированию модели данных PMN. В первом случае EF создаст Entity Data Model (EDM) из существующей базы данных. Во втором – используется модель-ориентированный подход (Model First), при котором сначала в дизайнере EF создаются сущности, а затем для создания базы данных, в которой может храниться EDM, автоматически формируются сценарии на языке описания данных (Data Definition Language, DDL).

EF: импорт существующей базы данных

Мы будем работать с уже существующим решением, в котором описаны основные проекты приложения PMN. Если сопроводительные материалы данной книги установлены в каталог по умолчанию, интересующее нас решение находится по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 5\Code\ExistingDatabase. Щелкните двойным щелчком файл PlanMyNight.sln.

Это решение включает все проекты, которые перечислены в следующем списке (рис. 5-5):

- PlanMyNight.Data: слой доступа к данным приложения
- PlanMyNight.Contracts: сущности и контракты
- PlanMyNight.Bing: сервисы Bing Map
- PlanMyNight.Web: слой представления
- PlanMyNight.AppFabricCaching: кэширование AppFabric

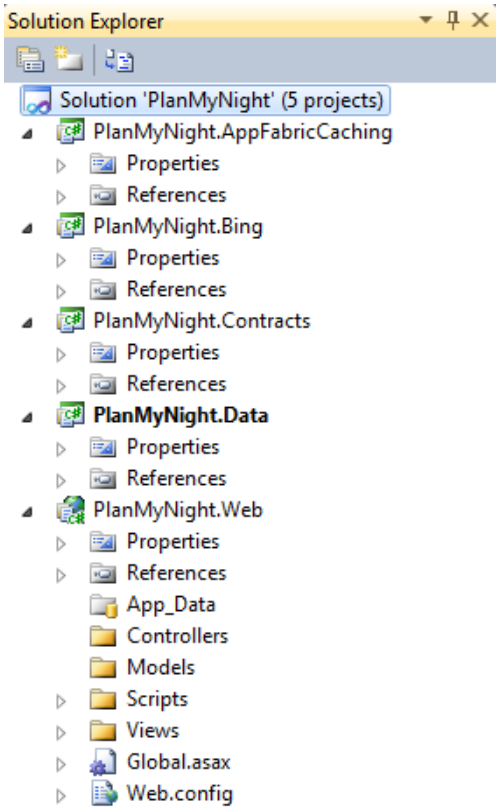


Рис. 5-5 Решение PlanMyNight

EF позволяет без труда импортировать существующую базу данных. Рассмотрим этот процесс поэтапно.

Первый шаг – добавление EDM в проект PlanMyNight.Data. Щелкаем правой кнопкой мыши проект PlanMyNight.Data, выбираем Add (Добавить) и New Item (Новый элемент). Выбираем элемент ADO.NET Entity Data Model и меняем его имя на **PlanMyNight.edmx**, как показано на рис. 5-6.

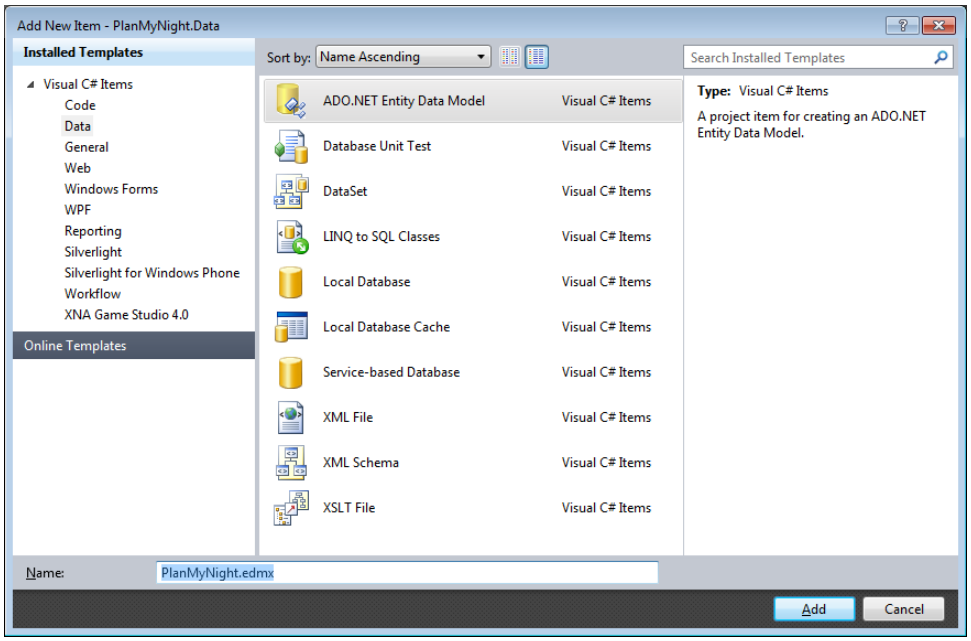


Рис. 5-6 Диалоговое окно Add New Item с выбранной ADO.NET Entity Data Model

Первое диалоговое окно мастера Entity Data Model Wizard позволяется выбрать содержимое модели. Мы собираемся создать модель из существующей базы данных. Выбираем Generate From Database (Создать из базы данных) и щелкаем Next (Далее).

Необходимо подключиться к существующему файлу базы данных. Щелкаем New Connection (Создать подключение). В открывшемся диалоговом окне Choose Data Source (Выбрать источник данных) выбираем Microsoft SQL Server Database File (Файл базы данных Microsoft SQL Server). Выбираем файл

%userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 5\ExistingDatabase\PlanMyNight.Web\App_Data\PlanMyNight.mdf (рис. 5-7).

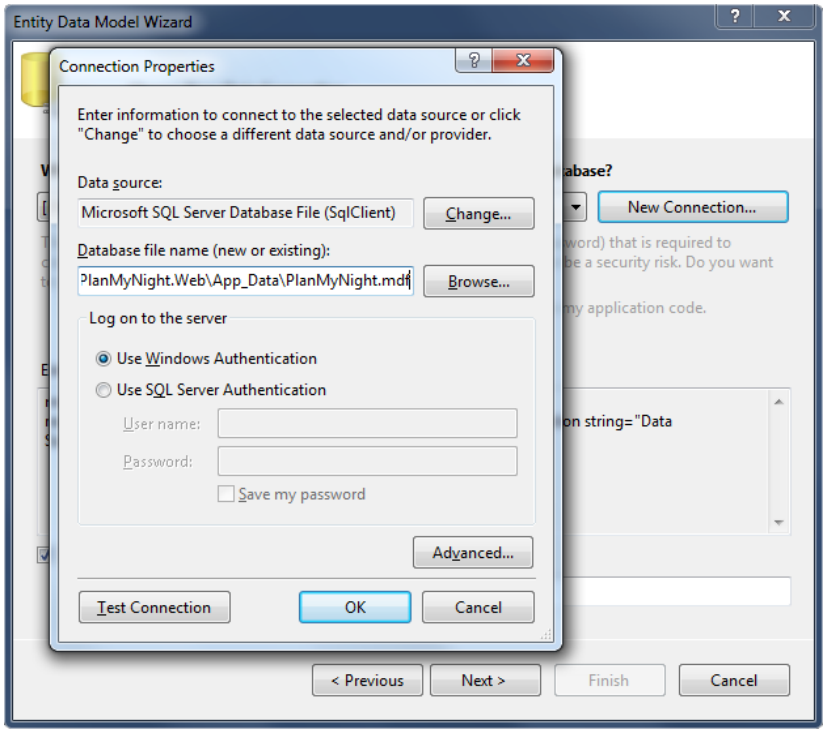


Рис. 5-7 Мастер EDM, диалоговое окно для создания подключения к базе данных

Все остальные поля формы пока оставляем без изменений и щелкаем Next.

> **Примечание** На экран будет выведено предупреждение о том, что локальный файл данных не входит в состав текущего проекта. Поскольку мы не хотим копировать файл базы данных в текущий проект, щелкните No (Нет), чтобы закрыть этот диалог.

В диалоговом окне Choose Your Database Objects (Выберите объекты своей базы данных) выберите таблицы Itinerary (План мероприятий), ItineraryActivities (Мероприятия), ItineraryComment (Комментарий к плану мероприятий), ItineraryRating (Рейтинг плана мероприятий) и ZipCode (Почтовый индекс), а также представление UserProfile (Профиль пользователя). Выберите хранимую процедуру *RetrieveItinerariesWithinArea* (Извлечение планов мероприятий в определенном районе). В поле Model Namespace (Пространство имен модели) зададим значение **Entities** (Сущности), как показано на рис. 5-8.

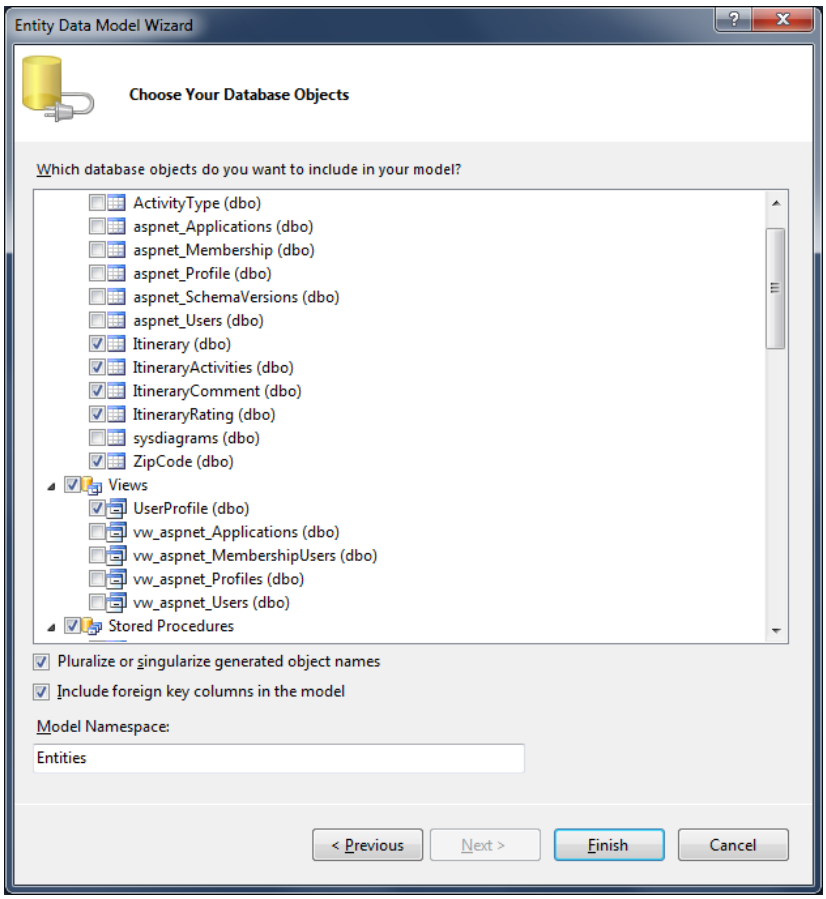


Рис. 5-8 Мастер EDM: страница Choose Your Database Objects

Щелкаем Finish (Готово), чтобы сформировать EDM.

Корректировка сформированной модели данных

Теперь мы имеем модель, представляющую множество сущностей, соответствующее используемой базе данных. Мастер сформировал все навигационные свойства, ассоциированные с внешними ключами базы данных.

Приложению PMN необходимо только навигационное свойство *ItineraryActivities*, связанное с таблицей Itinerary, поэтому все остальные навигационные свойства можно смело удалить. Также придется переименовать навигационное свойство *ItineraryActivities* в **Activities**. Обновленная модель представлена на рис. 5-9.

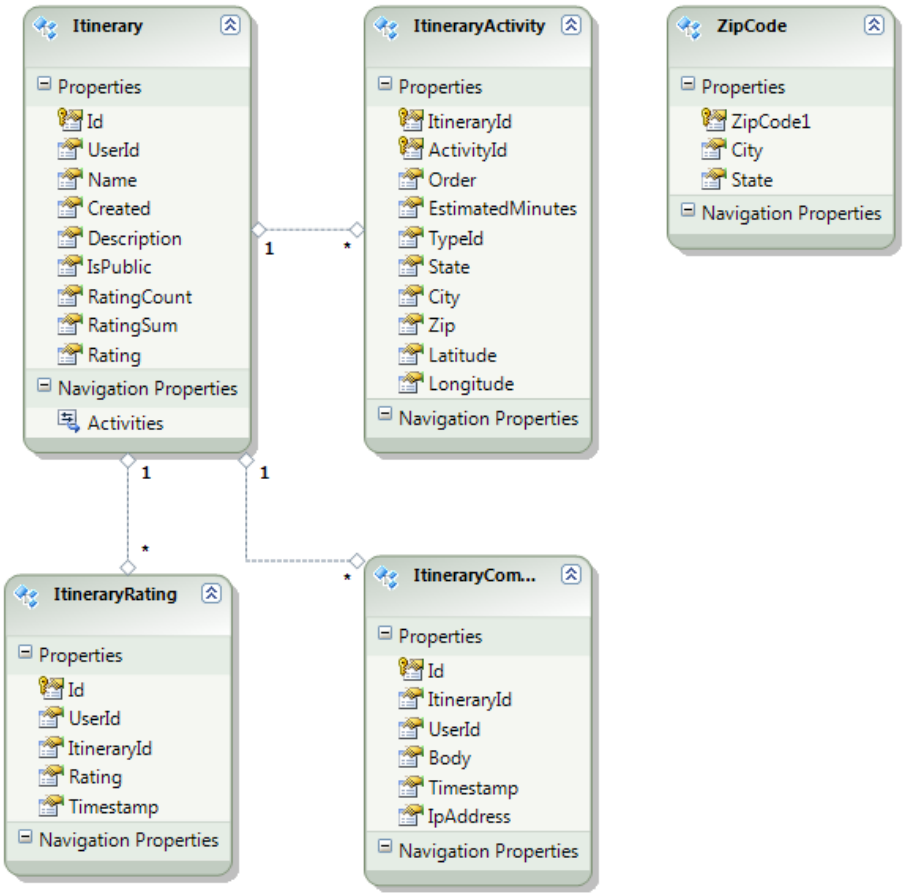


Рис. 5-9 Модель, импортированная из базы данных PlanMyNight

Можно заметить, что одному из свойств сущности ZipCode автоматически присвоено имя *ZipCode1*, потому что сама таблица уже называется ZipCode, а имя должно быть уникальным. Чтобы исправить это имя, щелкнем его двойным щелчком. Зададим имя **Code**, как показано на рис. 5-10.

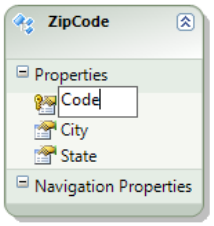


Рис. 5-10 Сущность ZipCode

Выполним сборку решения, нажав `Ctrl+Shift+B`. В окне вывода можно увидеть два сообщения от сформированной EDM. Первое можно сразу удалить, поскольку столбец Location (Местоположение) в PMN не нужен. Второе сообщение гласит:

> *Для таблицы/представления 'dbo.UserProfile' первичный ключ не задан и не может быть сформирован. Данная таблица/представление не включена в модель. Для использования этой сущности необходимо пересмотреть схему, добавить соответствующие ключи и раскомментировать ее.*

Взглянув на представление UserProfile можно заметить, что первичный ключ не задан явно, даже несмотря на то что значения столбца UserName (Имя пользователя) уникальны.

Чтобы иметь возможность выполнять доступ к данным UserProfile из приложения, придется вручную подкорректировать EDM и исправить сопоставление для представления UserProfile.

В обозревателе проекта щелкните правой кнопкой мыши файл PlanMyNight.edmx и выберите Open With (Открыть в). В диалоговом окне Open With выберите XML (Text) Editor ((Текстовый) редактор XML), как показано на рис. 5-11. Щелкните OK, чтобы открыть XML-файл, связанный с нашей моделью[1].

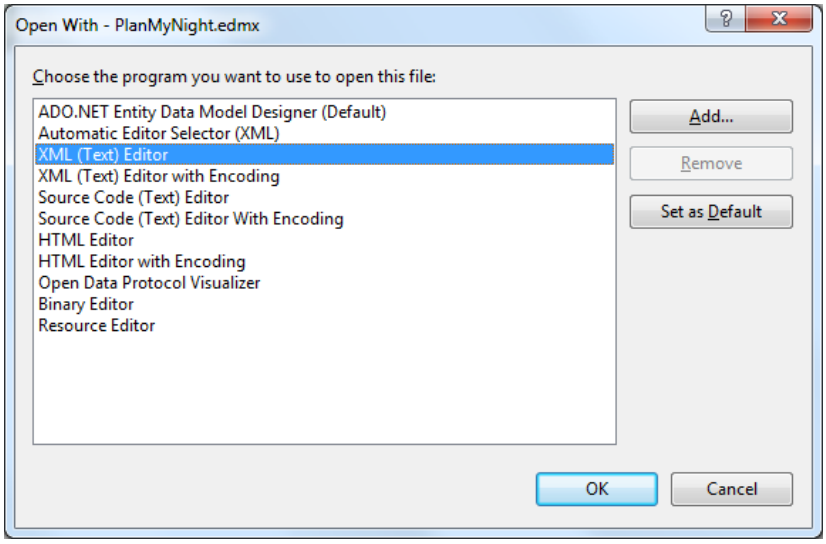


Рис. 5-11 Открываем PlanMyNight.edmx в редакторе XML

> **Примечание** На экран будет выведено сообщение о том, что файл PlanMyNight.edmx уже открыт. Щелкните Yes, чтобы закрыть его

Механизм формирования кода закомментировал созданный код из-за отсутствия первичного ключа. Чтобы использовать представление UserProfile из дизайнера, необходимо раскомментировать тип сущности UserProfile и добавить в него тег Key (Ключ). Найдите UserProfile в файле. Раскомментируйте тип сущности, добавьте тег ключа, задайте для него имя **UserName** и сделайте свойство *UserName* не допускающим значение null. Обновленный тип сущности представлен в листинге 5-1.

Листинг 5-1 XML-описание EntityType для UserProfile

```
<EntityType Name="UserProfile">
      <Key>
         <PropertyRef Name="UserName"/>
      </Key>
      <Property Name="UserName" Type="uniqueidentifier" Nullable="false" />
      <Property Name="FullName" Type="varchar" MaxLength="500" />
      <Property Name="City" Type="varchar" MaxLength="500" />
      <Property Name="State" Type="varchar" MaxLength="500" />
      <Property Name="PreferredActivityTypeId" Type="int" />
</EntityType>
```

Если закрыть XML-файл и попытаться открыть дизайнер EDM, в нем будет выведена такая ошибка: «The Entity Data Model Designer is unable to display the file you requested. You can edit the model using the XML Editor[2]».

На панели Error List (Список ошибок) выводится следующее предупреждение, более подробно раскрывающее суть этой ошибки:

Error 11002: Entity type 'UserProfile' has no entity set.[3]

Необходимо задать множество сущностей для типа UserProfile, чтобы тип сущности мог проецироваться на схему хранилища. Для этого открываем файл PlanMyNight.edmx в редакторе XML. Вверху файла, прямо над множеством сущностей Itinerary, добавляем XML-код, представленный в листинге 5-2.

Листинг 5-2 XML-описание EntitySet для UserProfile

```
<EntitySet Name="UserProfile" EntityType="Entities.Store.UserProfile"

     store:Type="Views" store:Schema="dbo" store:Name="UserProfile">
   <DefiningQuery>
```

[1] Этот XML-файл содержит описание всей созданной модели (*прим. технического редактора*)

[2] Дизайнер Entity Data Model не может вывести на экран запрашиваемый файл. Скорректируйте модель в редакторе XML (*прим. переводчика*).

[3] Для типа сущности 'UserProfile' не задано множество сущностей (*прим. переводчика*).

```
      SELECT
     [UserProfile].[UserName] AS [UserName],
     [UserProfile].[FullName] AS [FullName],
           [UserProfile].[City] AS [City],
         [UserProfile].[State] AS [State],
         [UserProfile].[PreferredActivityTypeId] as [PreferredActivityTypeId]
         FROM [dbo].[UserProfile] AS [UserProfile]
   </DefiningQuery>
</EntitySet>
```

Сохраним XML-файл EDM и повторно откроем дизайнер EDM. На рис. 5-12 показано представление UserProfile в разделе Entities.Store браузера модели (Model Browser).

> **Подсказка** Model Browser можно открыть из меню View (Вид), щелкнув Other Windows (Другие окна) и выбрав Entity Data Model Browser.

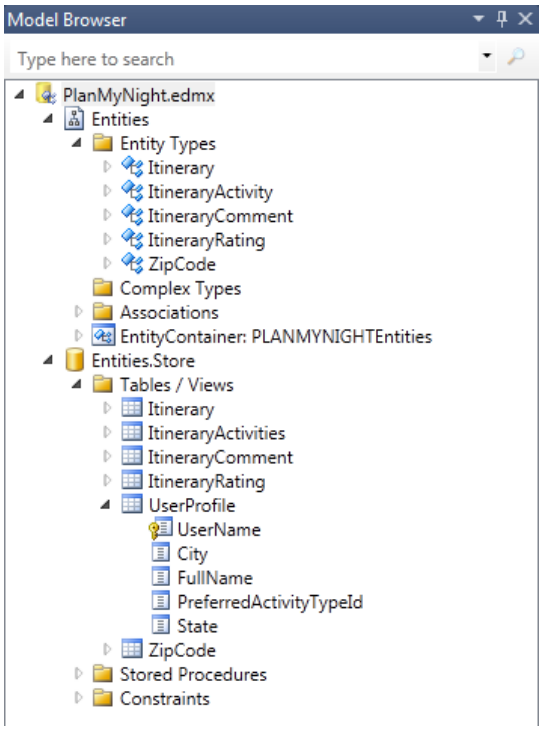


Рис. 5-12 Представление UserProfile в Model Browser

Теперь, когда представление доступно в метаданных хранилища, добавим сущность UserProfile и сопоставим ее с представлением UserProfile. Щелкаем правой кнопкой мыши фон дизайнера EDM, выбираем Add и затем Entity. На экран будет выведено диалоговое окно, показанное на рис. 5-13.

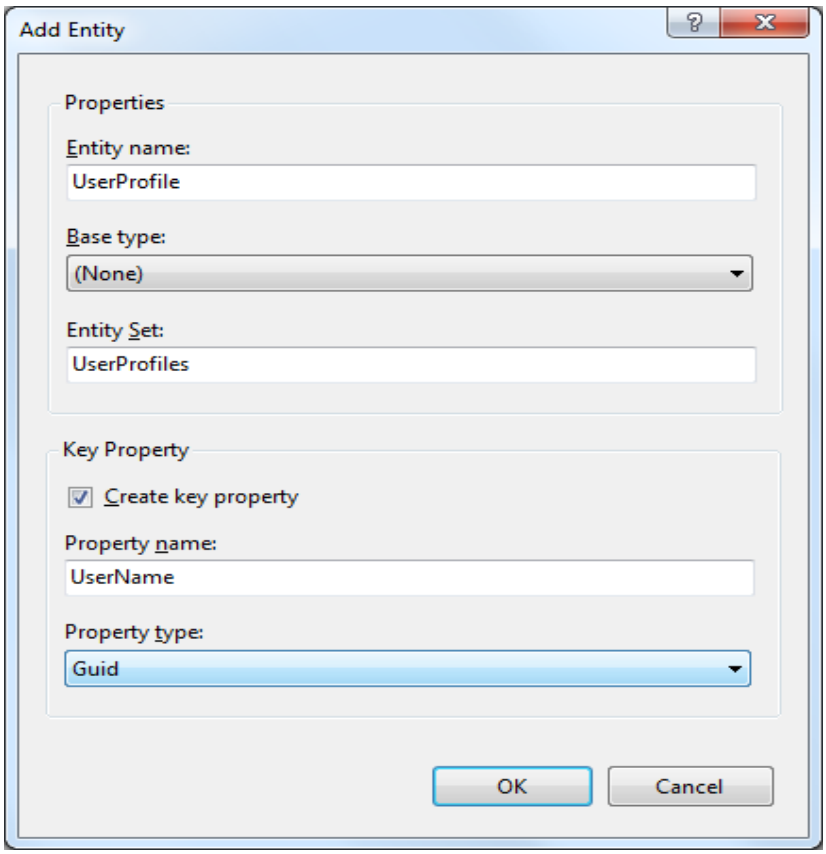


Рис. 5-13 Диалоговое окно Add Entity

Заполните диалоговое окно, как показано на рис. 5-13, и щелкните OK, чтобы создать сущность.

После этого понадобиться добавить остальные свойства: *City* (Город), *State* (Страна) и *PreferedActiviyTypeId* (Идентификатор предпочтительного типа мероприятий). Для этого щелкаем правой кнопкой мыши сущность UserProfile, выбираем Add и Scalar Property (Скалярное свойство). Как только свойство добавлено, задаем значения полей *Type* (Тип), *Max Length* (Максимальная длина) и *Unicode*. В табл. 5-1 представлены ожидаемые значения для каждого из полей.

Таблица 5-1 Свойства сущности UserProfile

Name	Type	Max Length	Unicode
FullName	*String*	500	False
City	*String*	500	False
State	*String*	500	False
PreferredActivityTypeId	*Int32*	Недоступно	Недоступно

Теперь, когда сущность UserProfile создана, ее требуется сопоставить с представлением UserProfile. Щелкаем правой кнопкой мыши сущность UserProfile и выбираем Table Mapping (Сопоставление таблиц), как показано на рис. 5-14.

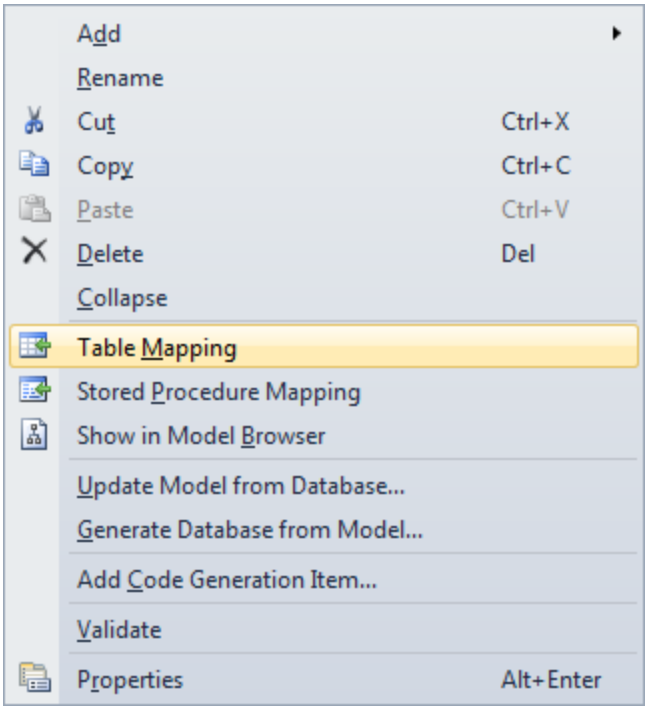


Рис. 5-14 Пункт меню Table Mapping

Затем выбираем представление UserProfile из раскрывающегося списка, как показано на рис. 5-15. Убедитесь, что все столбцы и свойства сущности сопоставлены правильно. Теперь представление UserProfile нашего хранилища доступно из кода через сущность UserProfile.

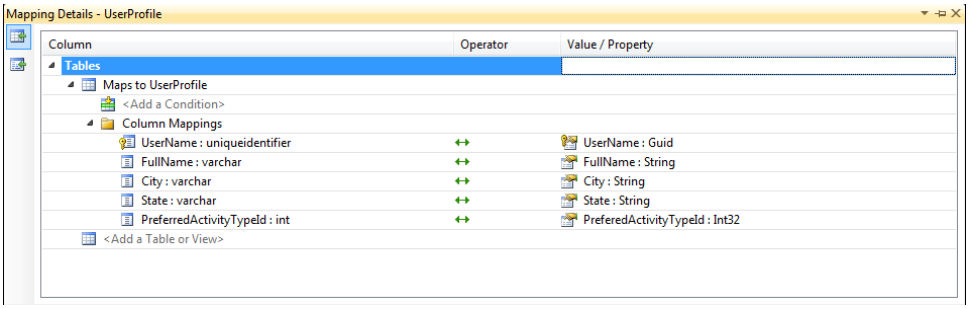


Рис. 5-15 Детали сопоставления UserProfile

Хранимая процедура и импортированные функции

Мастер Entity Data Model Wizard (Мастер модели сущность-данные) создал в модели хранилища запись для хранимой процедуры *RetrieveItinerariesWithinArea*, которая была выбрана в мастере в заключительном шаге. Необходимо создать соответствующую запись в концептуальной модели, добавив Function Import (Импортированная функция).

В разделе Entities.Store браузера модели откройте папку Stored Procedures (Хранимые процедуры). Щелкните правой кнопкой мыши *RetrieveItineraryWithinArea* и выберите Add Function Import (Добавить импортированную функцию). Диалоговое окно Add Function Import представлено на рис. 5-16. Задайте возвращаемый тип Entities и затем выберите элемент Itinerary в окне раскрывающегося списка. Щелкните OK.

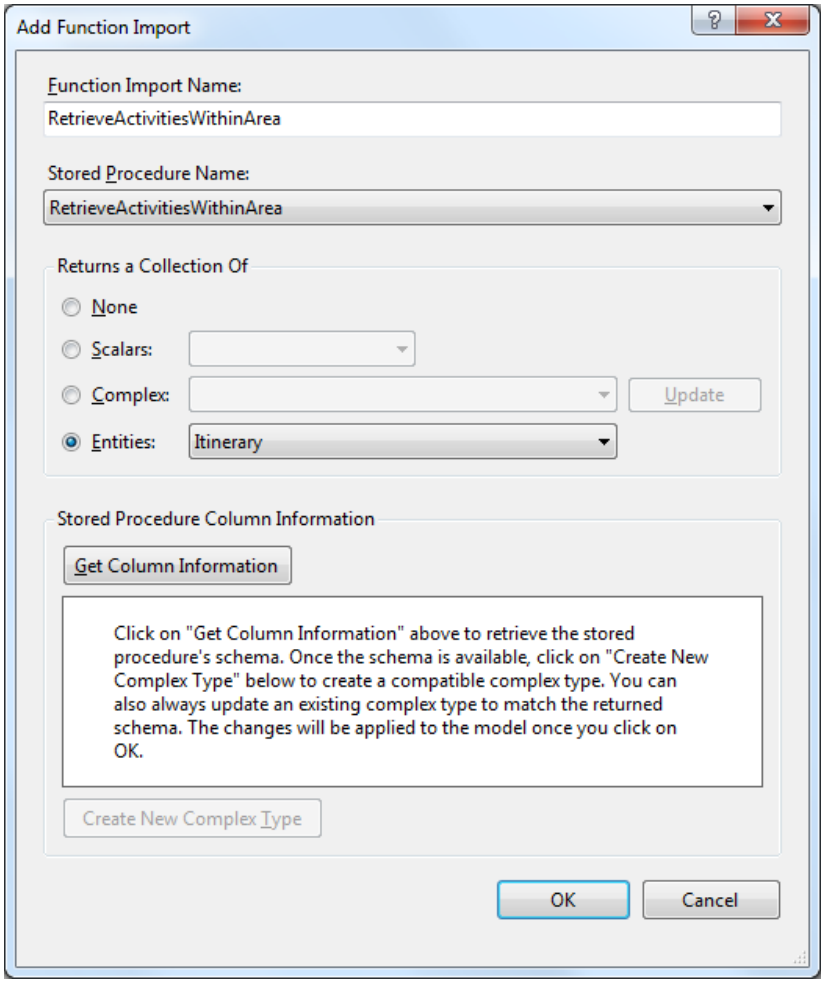


Рис. 5-16 Диалоговое окно Add Function Import

В браузер модели добавлена импортированная функция *RetrieveItinerariesWithinArea*, как показано на рис. 5-17.

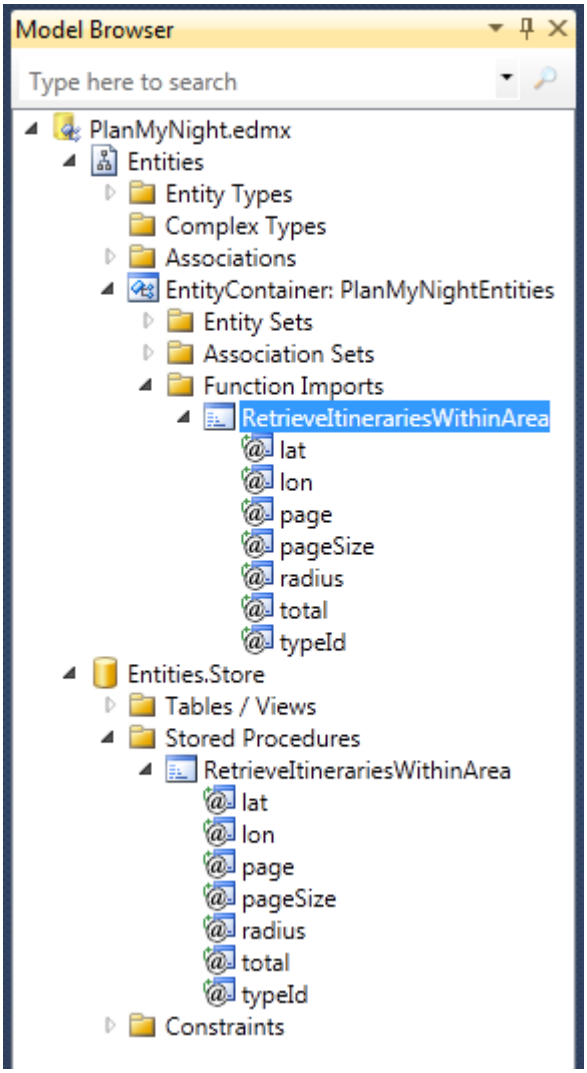


Рис. 5-17 Импортированные функции в браузере модели

Теперь можно проверить правильность EDM, щелкнув правой кнопкой мыши рабочую область дизайнера и выбрав Validate (Проверить). При этом не должно появиться никаких ошибок или предупреждений.

EF: сначала модель[1]

В предыдущем разделе мы увидели, как использовать дизайнер EF для создания модели путем импорта существующей базы данных. Дизайнер EF в Visual Studio 2010 также поддерживает возможность автоматического формирования файла Data Definition Language (DDL), который позволит создать базу данных из модели сущностей. В этом разделе мы используем новое решение, чтобы научиться автоматически формировать сценарий базы данных из модели.

Можно начать с пустой модели. Для этого в мастере Entity Data Model Wizard выбираем опцию Empty model (Пустая модель), как показано на рис. 5-18.

> **Примечание** Чтобы открыть мастер, щелкните правой кнопкой проект PlanMyNight.Data, выберите Add и затем New Item. Выберите элемент ADO.NET Entity Data Model.

[1] Модель-ориентированный подход (*прим. технического редактора*)

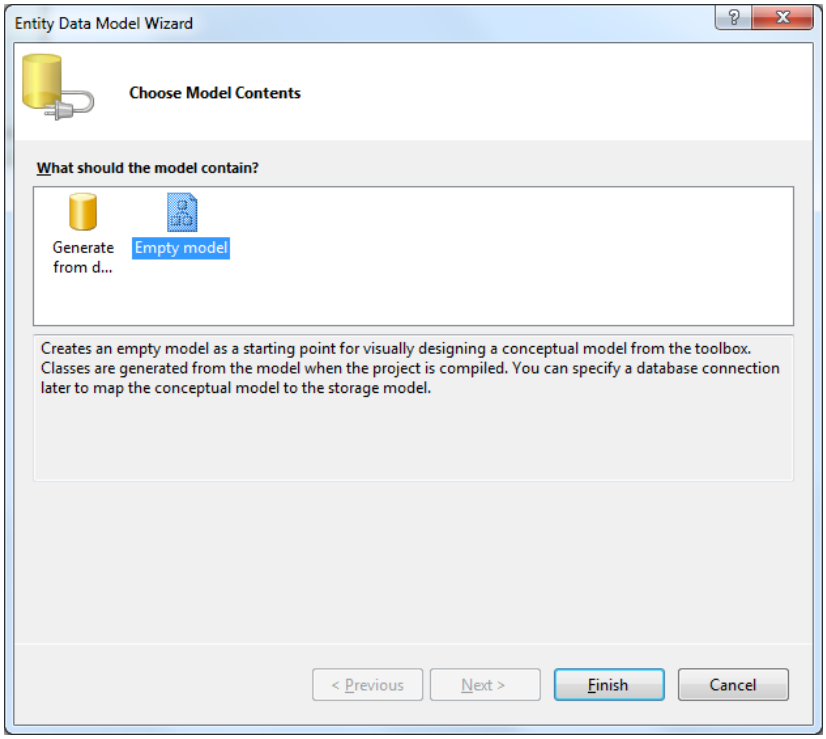


Рис. 5-18 Мастер EDM: Empty model

Открываем решение PMN по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 5\Code\ModelFirst, щелкнув двойным щелчком файл PlanMyNight.sln.

Проект PlanMyNight.Data этого решения уже содержит EDM-файл PlanMyNight.edmx с некоторыми уже созданными сущностями. Эти сущности соответствуют схеме данных, представленной на рис. 5-2.

Дизайнер Entity Model позволяет без труда добавлять сущности в модель данных. Добавим в нашу модель недостающую сущность ZipCode. Из панели инструментов перенесите в дизайнер элемент Entity, как показано на рис. 5-19. Присвойте этой сущности имя **ZipCode**, свойству *Id* – имя **Code** и задайте для него тип *String*.

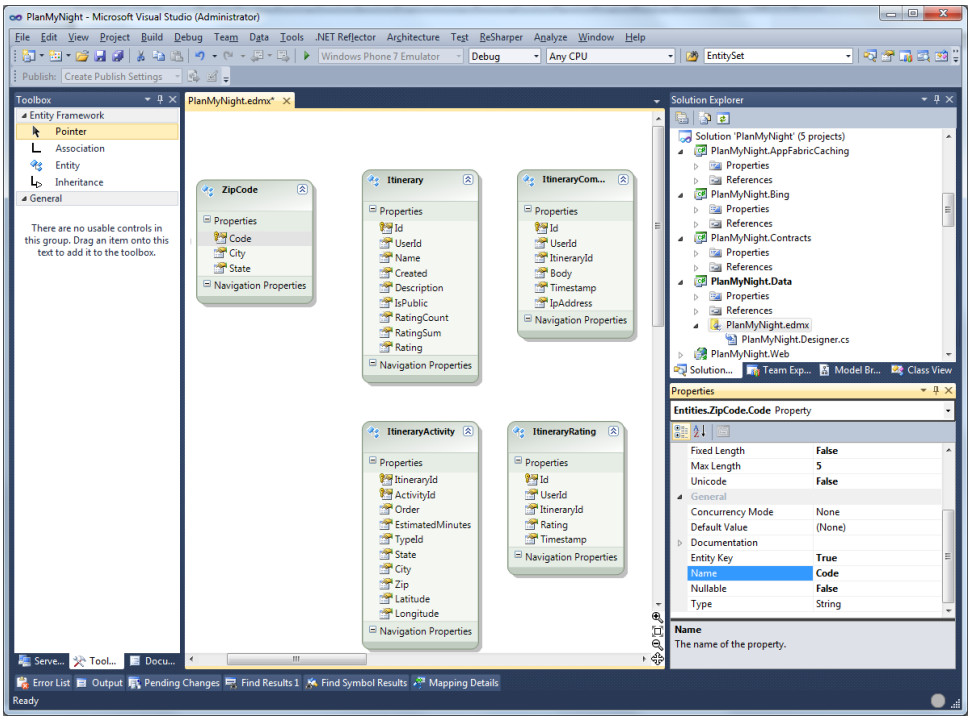


Рис. 5-19 Дизайнер модели сущностей

Теперь необходимо добавить в сущность свойства *City* и *State*. Щелкаем правой кнопкой сущность ZipCode, выбираем Add и затем Scalar Property. Убеждаемся, что значения всех свойств соответствуют приведенным в табл. 5-2.

Таблица 5-2 Свойства сущности ZipCode

Name	Type	Fixed Length	Max Length	Unicode
Code	*String*	False	5	False
City	*String*	False	150	False
State	*String*	False	150	False

Добавим отношения между ItineraryComment и сущностями Itinerary. Щелкаем правой кнопкой поверхность дизайнера, выбираем Add и затем Association (Связь), как показано на рис. 5-20.

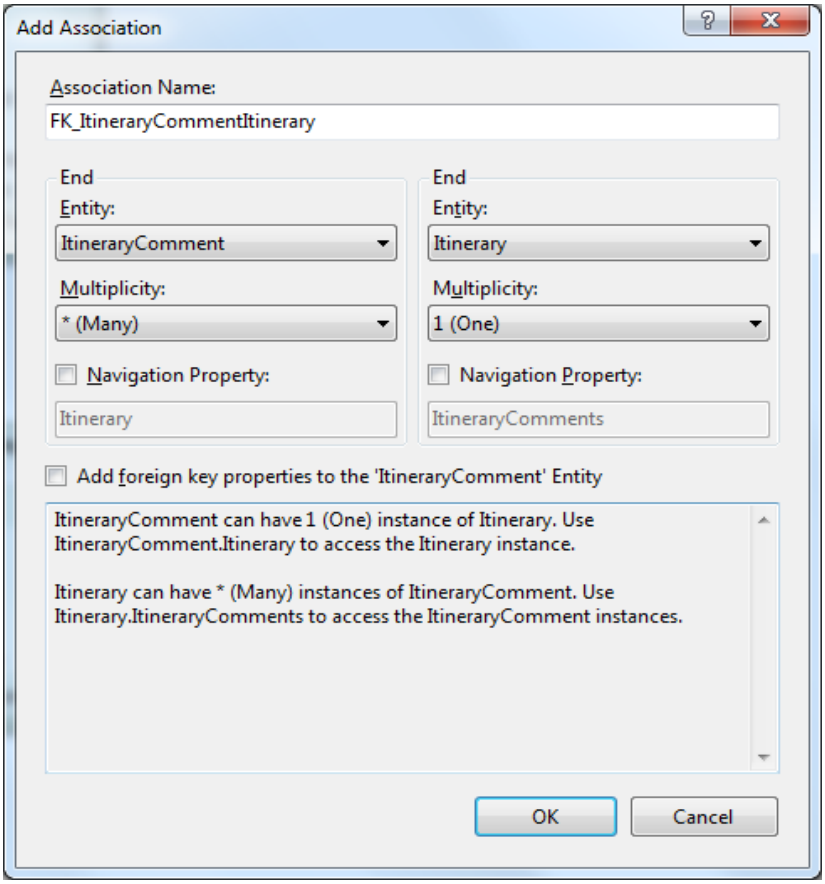


Рис. 5-20 Диалоговое окно Add Association для FK_ItineraryCommentItinerary

Задайте имя связи **FK_ItineraryCommentItinerary**, выберите сущность и кратность для каждого конца связи, как показано на рис. 5-20. Как только связь создана, щелкните двойным щелчком строку связи, чтобы задать Referential Constraint (Справочное ограничение), как показано на рис. 5-21.

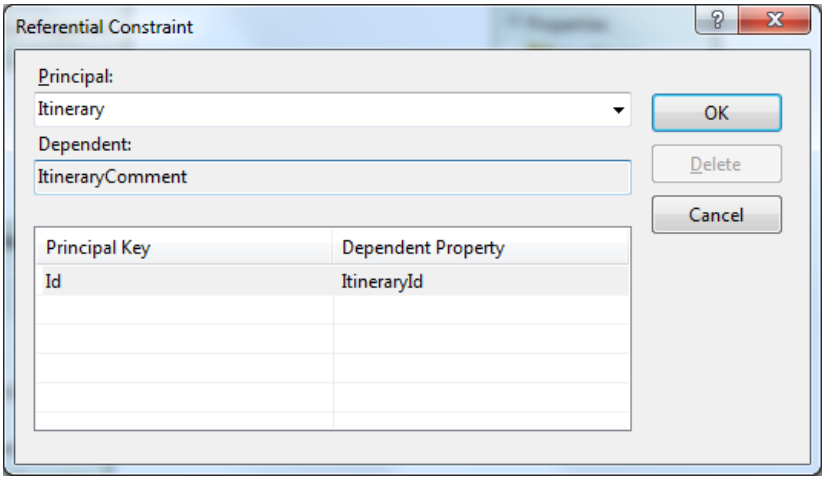


Рис. 5-21 Диалоговое окно Association Referential Constraint

Добавьте связь между сущностями ItineraryRating и Itinerary. Щелкните правой кнопкой фон дизайнера, выберите Add и затем Association. Задайте имя связи **FK_ItineraryItineraryRating**. После этого выберите сущность и кратность для каждого конца связи, как делали это в предыдущем случае, только в качестве

первого конца связи задайте **ItineraryRating**. Щелкните двойным щелчком линию связи и задайте Referential Constraint, как показано на рис. 5-21. Обратите внимание, что в поле Dependent (Зависимый) отображается не ItineraryComment, а ItineraryRating.

Создайте новую связь между сущностями ItineraryActivity и Itinerary. Также для связи *FK_ItineraryItineraryActivity* потребуется создать навигационное свойство и назвать его **Activities**, как показано на рис. 5-22. Когда связь создана, задайте для нее Referential Constraint, щелкнув двойным щелчком линию связи.

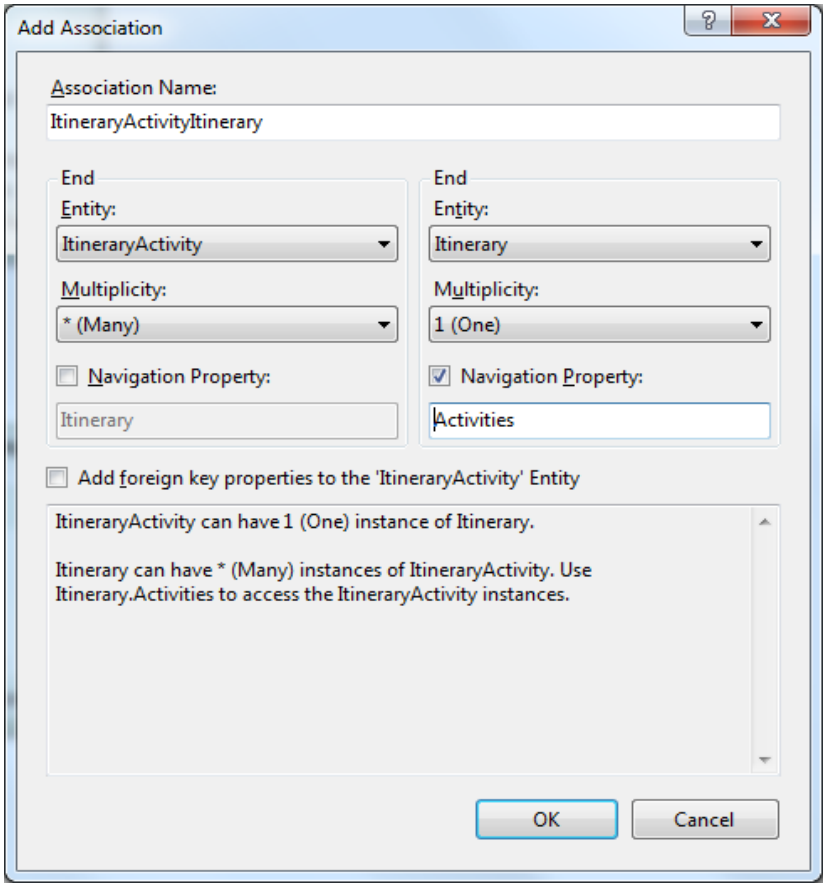


Рис. 5-22 Диалоговое окно Add Association для FK_ItineraryActivityItinerary

Автоматическое формирование сценария базы данных из модели

Модель данных готова, но с ней не ассоциировано никакое хранилище или сопоставление. Дизайнер EF предлагает возможность автоматического формирования сценария базы данных из модели.

Щелкните правой кнопкой дизайнер и выберите пункт меню Generate Database From Model (Сформировать базу данных из модели), как показано на рис. 5-23.

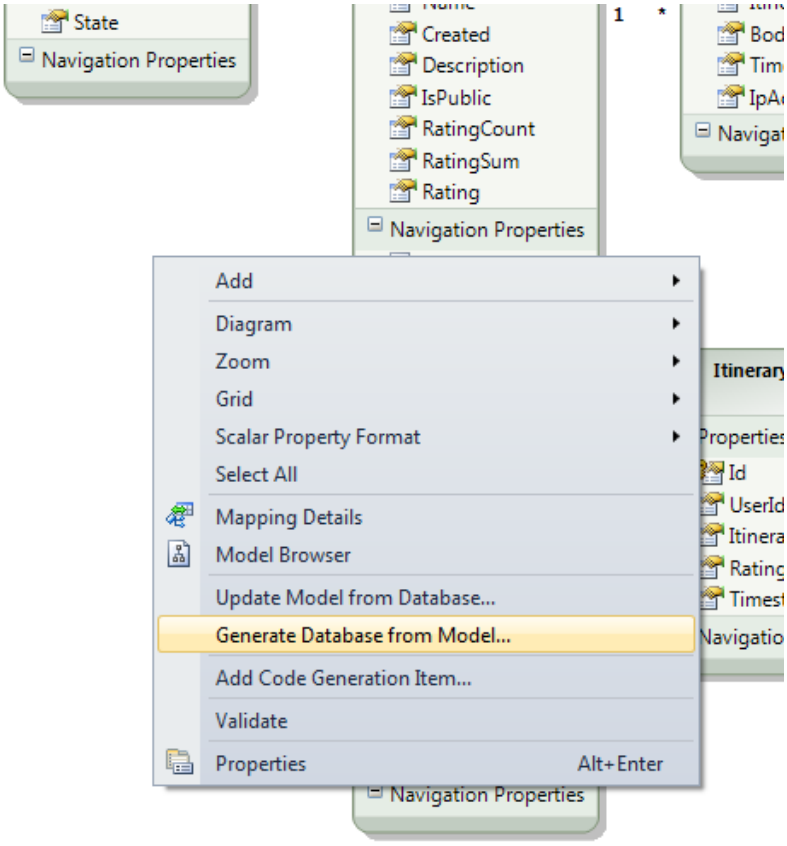


Рис. 5-23 Пункт меню Generate Database From Model

Мастер Generate Database Wizard требует подключения к данным. Этот мастер использует данные подключения для трансляции типов модели в тип базы данных и для формирования DDL-сценария для этой базы данных.

Выберите New Connection, в диалоговом окне Choose Data Source укажите Microsoft SQL Server File и щелкните Continue (Продолжить). Выберите файл базы данных, находящийся по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 5\Code\ModelFirst\Data\PlanMyNight.mdf (рис. 5-24).

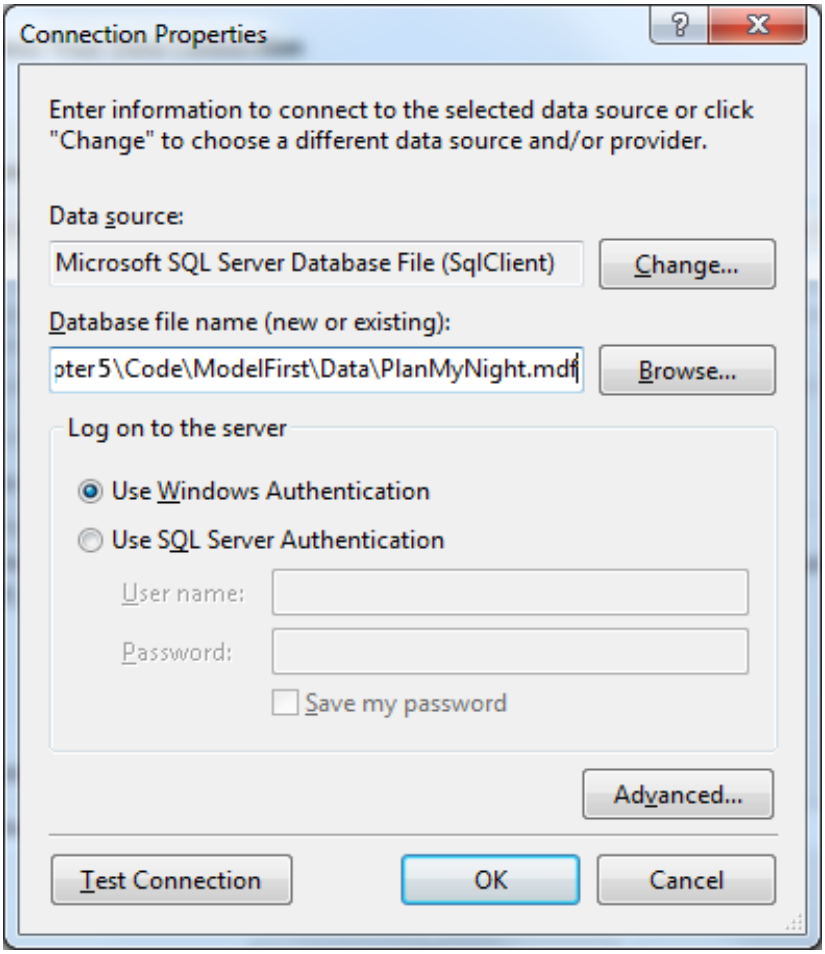


Рис. 5-24 Подключение базы данных для автоматического формирования сценария

Как только подключение настроено, щелкните Next, чтобы перейти к последнему окну мастера, как показано на рис. 5-25. По нажатию Finish сформированный файл T-SQL PlanMyNight.edmx.sql добавляется в проект. DDL-сценарий сформирует ограничения первичного и внешнего ключей для модели.

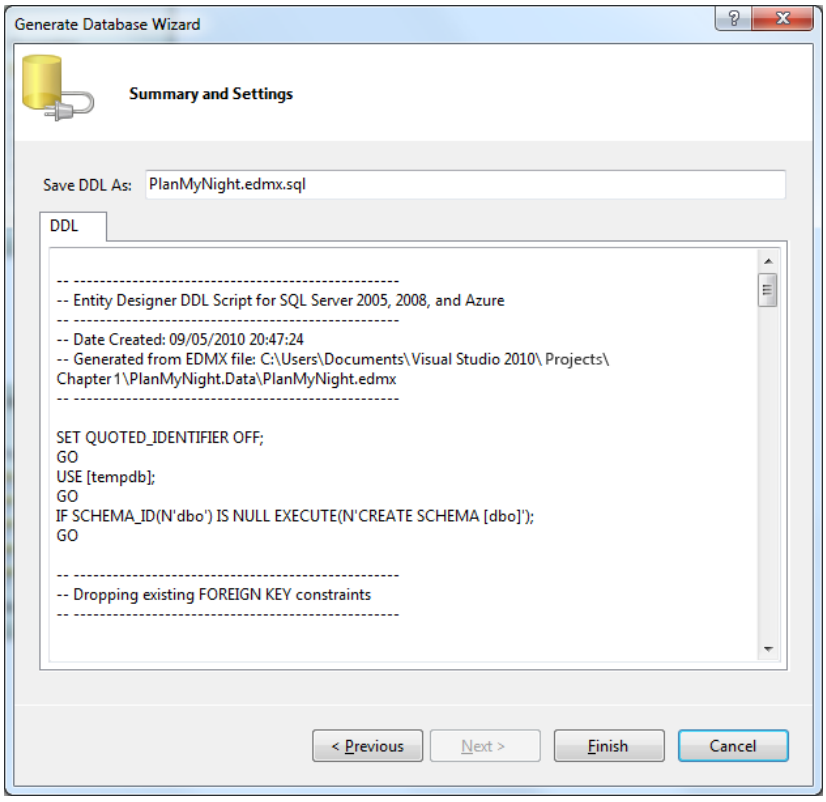


Рис. 5-25 Сформированный файл T-SQL

EDM также обновляется, чтобы вновь созданное хранилище гарантированно сопоставлялось с сущностями. Теперь сформированный DDL-сценарий может использоваться для добавления таблиц в базу данных. Также мы получили слой доступа к данным, предоставляющий строго типизированные сущности, которые могут использоваться в приложении.

> **Важно** Для создания полной базы данных PMN потребовалось бы добавить оставшиеся таблицы, хранимые процедуры и триггеры, используемые приложением. Вместо того чтобы осуществлять все эти операции, вернемся к решению, каким оно было в конце раздела «EF: импорт существующей базы данных».

POCO-шаблоны

Для формирования кода сущностей дизайнер EDM использует шаблоны T4. До сих пор мы позволяли дизайнеру создавать сущности на базе шаблонов по умолчанию. Автоматически сформированный код можно увидеть в файле PlanMyNight.Designer.cs, ассоциированном с PlanMyNight.edmx. Созданные в нем сущности типа EntityObject и снабжены атрибутами, которые обеспечивают возможность EF управлять ими во время выполнения.

> **Примечание** T4 расшифровывается как *Text Template Transformation Toolkit* (Набор инструментов для преобразования текстовых шаблонов). Поддержка T4 в Visual Studio 2010 позволяет без труда создавать собственные шаблоны и формировать любые текстовые файлы (Веб, ресурсы или источники). Более подробные сведения об автоматическом формировании кода в Visual Studio 2010 можно найти в статье Code Generation and Text Templates (Автоматическое формирование кода и текстовые шаблоны) по адресу *http://msdn.microsoft.com/en-us/library/bb126445(VS.100).aspx.*

EF также поддерживает типы сущностей POCO. POCO-классы – это простые объекты без атрибутов или базового класса инфраструктуры. (Листинг 5-3 в следующем разделе представляет POCO-класс для сущности ZipCode.) EF использует имена типов и свойства этих объектов для их сопоставления с моделью во время выполнения.

> **Примечание** POCO расшифровывается как *Plain-Old CLR Objects*[1].

Шаблон ADO.NET POCO Entity Generator

Повторно откроем файл %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 5\Code\ExistingDatabase\PlanMyNight.sln.

Выберем файл PlanMyNight.edmx, щелкнем правой кнопкой дизайнер и выберем Add Code Generation Item (Добавить элемент формирования кода). При этом откроется диалоговое окно, показанное на рис. 5-26, которое предоставляет возможность выбрать необходимый шаблон. Выберем шаблон ADO.NET POCO Entity Generator[2] и назовем его **PlanMyNight.tt**. После этого щелкнем кнопку Add.

> **Примечание** Возможно, на экран будет выведено предупреждение об угрозе безопасности в случае выполнения этого текстового шаблона. Источник данного шаблона доверенный, поэтому просто щелкните OK, чтобы закрыть это диалоговое окно.

[1] Обычные объекты CLR (*прим. переводчика*)

[2] По-умолчанию данный шаблон не установлен. Если у Вас его нет, выберите страницу Online Templates (он-лайн шаблоны), а далее введите слово POCO в поле Search Online Templates (Поиск он-лайн шаблонов) и нажмите Enter, через некоторое время шаблон будет найден, его можно будет установить и использовать (*прим. технического редактора*)

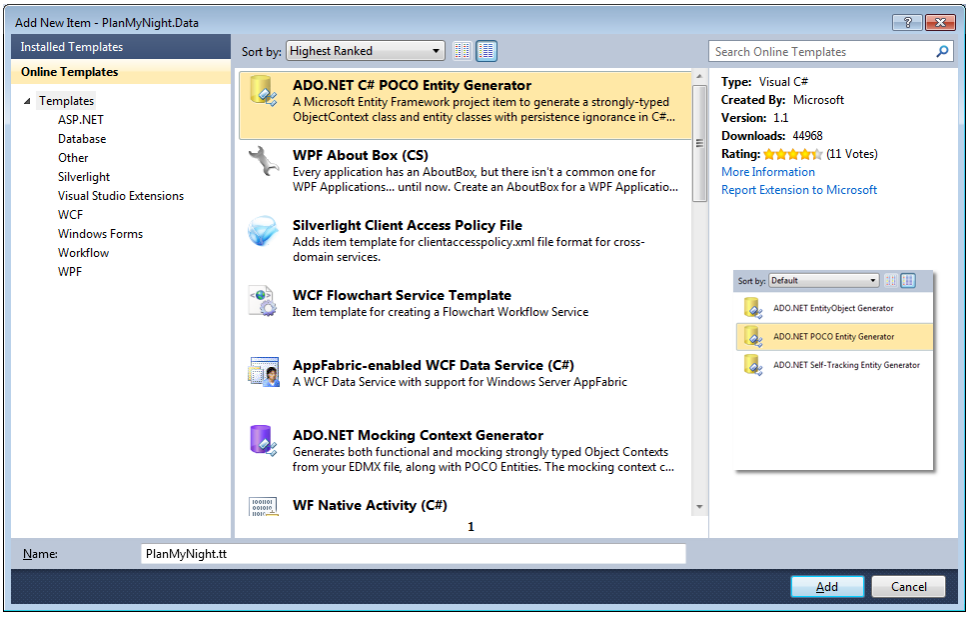


Рис. 5-26 Диалоговое окно Add New Item

В проект были добавлены два файла, PlanMyNight.tt и PlanMyNight.Context.tt (рис. 5-27). Эти файлы заменяют шаблон формирования кода по умолчанию, и код больше не формируется в файле PlanMyNight.Designer.cs.

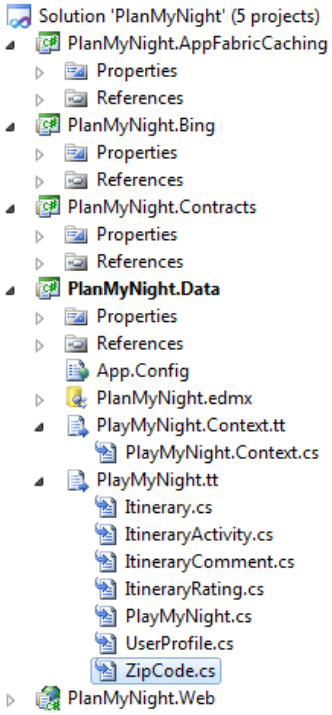


Рис. 5-27 Добавленные шаблоны

Шаблон PlanMyNight.tt создает по файлу класса для каждой сущности в модели. Листинг 5-3 представляет POCO-версию класса *ZipCode*.

Листинг 5-3 POCO-версия класса ZipCode

```
namespace Microsoft.Samples.PlanMyNight.Data
{
    public partial class ZipCode
    {
        #region Primitive Properties
        public virtual string Code
        {
            get;
            set;
```

```
        }
        public virtual string City
        {
            get;
            set;
        }
        public virtual string State
        {
            get;
            set;
        }
        #endregion
    }
}
```

Подсказка В C# 3.0 появляется новая возможность, называемая *автоматические свойства*. Для каждого обнаруженного во время компиляции пустого блока *get* или *set* компилятор создает резервное поле.

Другой файл, PlanMyNight.Context.cs, формирует объект *ObjectContext* (Контекст объекта) для модели PlanMyNight.edmx. Этот объект мы будем использовать для взаимодействия с базой данных.

Подсказка Шаблоны POCO будут автоматически обновлять сформированные классы соответственно внесенным в модель изменениям при сохранении файла .edmx.

Перенос классов сущностей в проект контрактов

Архитектура приложения PMN спроектирована таким образом, что слой представления гарантированно безразличен к методу хранения. Это реализовано путем перемещения контрактов и классов сущностей в сборку, которая не имеет ссылки на хранилище.

Visual Studio 2005 Даже несмотря на то, что существовала возможность расширять обработку XSD с помощью инструментов для автоматического формирования кода, это было связано с определенными сложностями и требовало обслуживания этих инструментов. EF использует шаблоны T4 для формирования и схемы базы данных, и кода. Эти шаблоны можно без труда настроить соответственно конкретным требованиям.

Шаблоны POCO ADO.NET выносят формирование классов сущностей в отдельный шаблон, что дает нам возможность с легкостью переносить эти сущности в другой проект.

Перенесем файл PlanMyNight.tt в проект PlanMyNight.Contracts. Щелкнув правой кнопкой файл PlanMyNight.tt, выбираем Cut (Вырезать). Щелкаем правой кнопкой папку Entities проекта PlanMyNight.Contracts и выбираем Paste (Вставить). Полученный результат представлен на рис. 5-28.

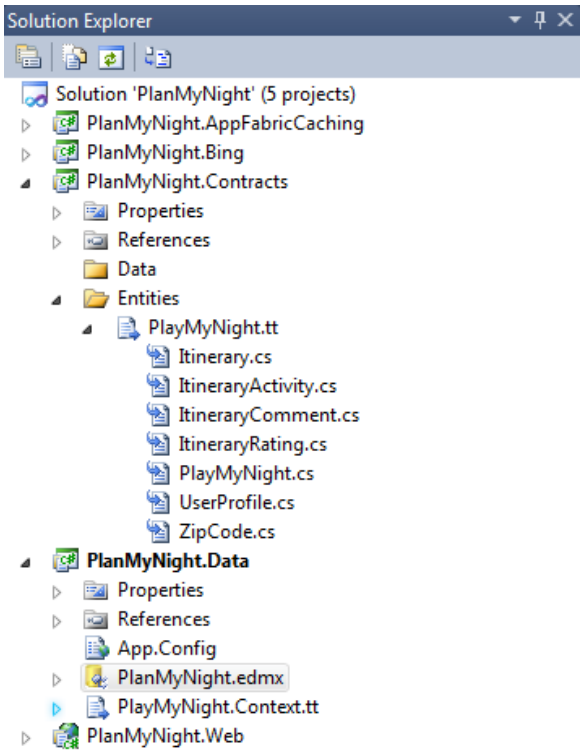


Рис. 5-28 Шаблон POCO, перенесенный в проект Contracts

Для формирования кода типа сущности шаблон PlanMyNight.tt использует метаданные модели EDM. Необходимо скорректировать относительный путь, используемый шаблоном для доступа к файлу EDMX.

Откройте шаблон PlanMyNight.tt и найдите строку:

```
string inputFile = @"PlanMyNight.edmx";
```

Исправьте путь к файлу, чтобы он указывал на файл PlanMyNight.edmx в проекте PlanMyNight.Data:

```
string inputFile = @"..\..\PlanMyNight.Data\PlanMyNight.edmx";
```

Классы сущностей будут сформированы повторно, как только вы сохраните шаблон.

Также необходимо обновить шаблон PlanMyNight.Context.tt в проекте PlanMyNight.Contracts, поскольку теперь классы сущностей находятся в пространстве имен *Microsoft.Samples.PlanMyNight.Entities*, а не в *Microsoft.Samples.PlanMyNight.Data*. Откройте файл PlanMyNight.Context.tt и включите новое пространство имен в раздел директив *using*.

```
using System;
using System.Data.Objects;
using System.Data.EntityClient;
using Microsoft.Samples.PlanMyNight.Entities;
```

Нажав `Ctrl+Shift+B`, выполните сборку решения. Теперь компиляция проекта должна пройти успешно.

Сведение воедино

Теперь, когда универсальный код для взаимодействия с базой данных SQL создан, все готово для реализации специальной функциональности приложения PMN. В следующих разделах мы поэтапно рассмотрим этот процесс, вкратце остановимся на получении данных с сервисов Bing Maps и пройдем небольшой вводный курс в возможность кэширования AppFabric Windows Server, используемую в PMN.

Чтобы свести все это воедино, требуется довольно большой объем вспомогательного кода. Упростить процесс поможет использование обновленного решения, в котором уже написана большая часть кода для контрактов и сущностей, а также для работы с сервисами Bing Maps. Решение также будет включать проект PlanMyNight.Data.Test для проверки кода проекта PlanMyNight.Data.

> **Примечание** Тестирование в Visual Studio 2010 будет рассмотрено в главе 7.

Получение данных из базы данных

В начале этой главы нами было принято решение сгруппировать операции над сущностью Itinerary в интерфейс хранилища *IItinerariesRepository*. Вот некоторые из этих операций:

- Поиск плана мероприятий по мероприятию
- Поиск плана мероприятий по почтовому индексу
- Поиск плана мероприятий по радиусу
- Добавление нового плана мероприятий

Рассмотрим соответствующие методы интерфейса *IItinerariesRepository*:

- *SearchByActivity* обеспечит поиск планов мероприятий по мероприятию и возвращение страницы данных.
- *SearchByZipCode* обеспечит поиск планов мероприятий по почтовому индексу и возвращение страницы данных.
- *SearchByRadius* обеспечит поиск планов мероприятий из определенного пункта и возвращение страницы данных.
- *Add* обеспечит добавление плана мероприятий в базу данных.

Откроем решение PMN, располагающееся по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 5\Code\Final, щелкнув двойным щелчком файл PlanMyNight.sln.

Выберем проект PlanMyNight.Data и откроем файл ItinerariesRepository.cs. Это реализация интерфейса *IItinerariesRepository*. Используя сформированный ранее объект контекста PlanMyNightEntities, мы можем создавать запросы LINQ к модели, и EF транслирует их в обычный T-SQL, который будет применяться для работы с базой данных.

Примечание LINQ расшифровывается как Language Integrated Query[1] и появился в .NET Framework 3.5. Он обеспечивает встроенную возможность построения запросов к данным в .NET Framework, так что разработчику не приходится беспокоиться об изучении или обслуживании специальных запросов SQL. LINQ позволяет использовать строго типизированные объекты, а Visual Studio IntelliSense обеспечивает возможность выбирать свойства или методы, присутствующие в текущем контексте, как показано на рис. 5-29. Больше сведений о LINQ предоставлено в ресурсе .NET Framework Developer Center (*http://msdn.microsoft.com/en-us/netframework/aa904594.aspx*).

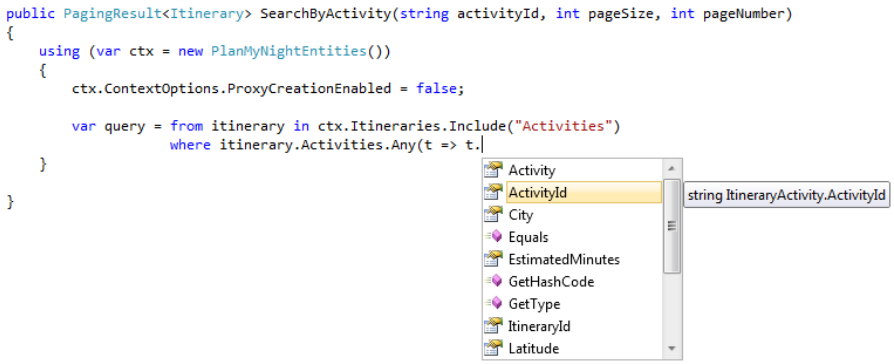


Рис. 1-28 Поддержка IntelliSense для запросов LINQ

Перейдите к описанию функции *SearchByActivity*. Этот метод должен возвращать набор планов мероприятий, для которых флаг *IsPublic* (Общедоступный) установлен в значение true, и значение *activityId* одного из мероприятий соответствует переданному в аргументе функции значению. Полученные в результате планы мероприятий должны быть сортированы по полю рейтинга.

Visual Studio 2005 Реализация каждого метода для извлечения плана мероприятий в Visual Studio 2005 потребовала бы написания специализированных SQL-запросов. С EF и LINQ любой запрос становится типовым, а изменения вносятся без труда на уровне кода!

SearchByActivity реализуется с применением стандартных операторов LINQ, как показано в листинге 5-4. Добавим выделенный код в тело метода *SearchByActivity*.

Листинг 5-4 Реализация SearchByActivity

```
public PagingResult<Itinerary> SearchByActivity(string activityId, int pageSize,
int pageNumber)
{
    using (var ctx = new PlanMyNightEntities())
    {
        ctx.ContextOptions.ProxyCreationEnabled = false;
```

[1] Язык интегрированных запросов (*прим. переводчика*)

```
        var query = from itinerary in ctx.Itineraries.Include("Activities")
                    where itinerary.Activities.Any(t => t.ActivityId ==
                            activityId) && itinerary.IsPublic
                    orderby itinerary.Rating
                    select itinerary;

        return PageResults(query, pageNumber, pageSize);
    }
}
```

Примечание Разбиение результатов на страницы реализовано в методе *PageResults*:

```
private static PagingResult<Itinerary> PageResults(IQueryable<Itinerary> query,
int page, int pageSize)
{
    int rowCount = rowCount = query.Count();
    if (pageSize > 0)
    {
        query = query.Skip((page - 1) * pageSize) .Take(pageSize);
    }
    var result = new PagingResult<Itinerary>(query.ToArray())
    {
        PageSize = pageSize,
        CurrentPage = page,
        TotalItems = rowCount
    };
    return result;
}
```

В эту функцию передается *IQueryable<Itinerary>*, что позволяет добавить разбиение на страницы в состав базового запроса. Передача *IQueryable* вместо *IEnumerable* гарантирует, что T-SQL для запроса к хранилищу будет формироваться только при вызове *query.ToArray()*.

Метод *SearchByZipCode* аналогичен методу *SearchByActivity*, но также добавляет фильтр по почтовому индексу места выполнения действия. Опять же, поддержка LINQ упрощает реализацию, как показано в листинге 5-5. Добавим выделенный код в тело метода *SearchByZipCode*.

Листинг 5-5 Реализация SearchByZipCode

```
public PagingResult<Itinerary> SearchByZipCode(int activityTypeId, string zip, int
pageSize, int pageNumber)
{
    using (var ctx = new PlanMyNightEntities())
    {
        ctx.ContextOptions.ProxyCreationEnabled = false;

        var query = from itinerary in ctx.Itineraries.Include("Activities")
                    where itinerary.Activities.Any(t => t.TypeId == activityTypeId
&& t.Zip == zip)
                          && itinerary.IsPublic
                    orderby itinerary.Rating
                    select itinerary;

        return PageResults(query, pageNumber, pageSize);
    }
}
```

Функция *SearchByRadius* вызывает импортированную функцию *RetrieveItinerariesWithinArea*, которая была сопоставлена с хранимой процедурой. После этого загружаются действия для каждого найденного маршрута. Вы можете скопировать выделенный код листинга 5-6 в тело метода *SearchByRadius* в файле ItinerariesRepository.cs.

Листинг 5-6 Реализация SearchByRadius

```
public PagingResult<Itinerary> SearchByRadius(int activityTypeId, double longitude,
double latitude, double radius, int pageSize, int pageNumber)
{
    using (var ctx = new PlanMyNightEntities())
    {
        ctx.ContextOptions.ProxyCreationEnabled = false;

        // Хранимая процедура с выходным параметром
        var totalOutput = new ObjectParameter("total", typeof(int));
        var items = ctx.RetrieveItinerariesWithinArea(activityTypeId,
            latitude, longitude, radius, pageSize, pageNumber,
            totalOutput).ToArray();

        foreach (var item in items)
        {
         item.Activities.ToList().AddRange(this.Retrieve(item.Id).Activities);
        }

        int total = totalOutput.Value == DBNull.Value ? 0 :
                 (int)totalOutput.Value;

        return new PagingResult<Itinerary>(items)
        {
            TotalItems = total,
            PageSize = pageSize,
            CurrentPage = pageNumber
        };
    }
}
```

С помощью метода *Add* планы мероприятий добавляются в хранилище данных. Реализация этой функциональности упростилась, поскольку наш контракт и объект контекста используют один и тот же объект сущности. Вставьте выделенный код листинга 5-7 в тело метода *Add*.

Листинг 5-7 Реализация Add

```
public void Add(Itinerary itinerary)
{
    using (var ctx = new PlanMyNightEntities())
    {
        ctx.Itineraries.AddObject(itinerary);
        ctx.SaveChanges();
    }
}
```

Готово! Мы завершили реализацию *ItinerariesRepository*, применив объект контекста, сформированный дизайнером EF. Выполните все тесты решения, нажав CTRL+R, A. Все тесты, касающиеся *ItinerariesRepository*, должны быть пройдены успешно.

Получение данных с Веб-сервисов Bing Maps

Поиск мероприятий для построения собственных планов мероприятий пользователя приложение PMN обеспечивает через применение Веб-сервисов Bing Maps. Чтобы получить ключ Bing Maps для использования в приложении PMN, необходимо создать учетную запись разработчика Bing Maps. Создать бесплатную учетную запись разработчика можно в центре управления учетными записями Bing Maps (Bing Maps Account Center).

*Дополнительные сведения Веб-сервисы Microsoft Bing Maps – это набор программируемых сервисов, работающих по протоколу Simple Object Access Protocol (SOAP)[1], которые обеспечивают возможность находить местоположение на карте по заданному адресу, выполнять поиск местоположения по координатам, интегрировать карты и изображения, возвращают направления движения и позволяют включать в Веб-приложение всевозможную логику операций с определением местоположения. Более подробные сведения об этих сервисах можно найти на сайте, посвященном Bing Maps Web Services SDK[2] (*http://msdn.microsoft.com/en-us/library/cc980922.aspx*).*

Visual Studio 2005 В Visual Studio 2005 чтобы добавить ссылку на Веб-сервис, необходимо было выбрать в контекстном меню опцию Add Web Service Reference (Добавить ссылку на Веб-сервис). Это обеспечило бы

[1] Простой протокол доступа к объектам (*прим. переводчика*).

[2] Пакет средств разработки Веб-сервисов Bing Maps (*прим. переводчика*).

открытие диалогового окна Add Web Reference (Добавить Веб-ссылку), в котором можно добавить ссылку на Веб-сервис в проект (рис. 5-30.).

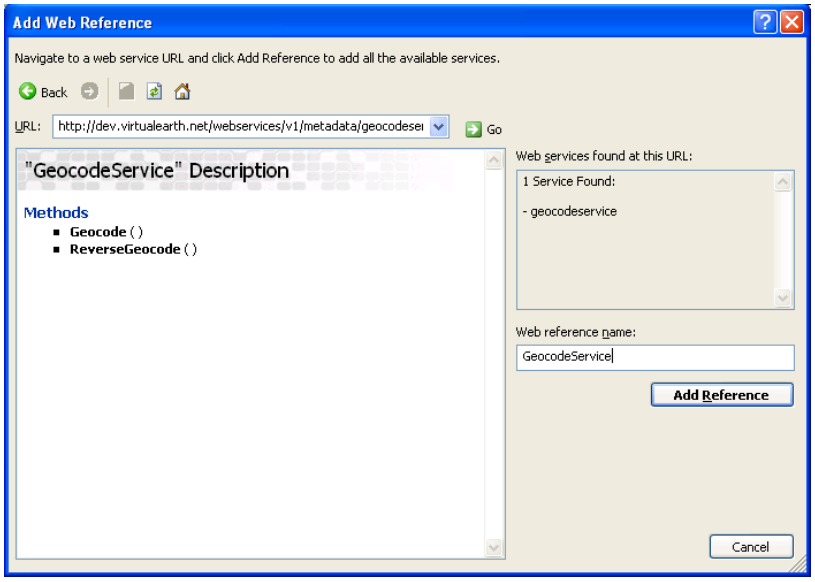


Рис. 5-30 Диалоговое окно Add Web Reference в Visual Studio 2005

Появившиеся в .NET Framework 3.0 сервисы Windows Communication Foundation (WCF) привели Веб-сервисы ASMX и другие технологии связи к одной унифицированной модели программирования.

Visual Studio 2010 обеспечивает инструментарий для работы с сервисами WCF. Чтобы вывести на экран новое диалоговое окно Add Service Reference (Добавить ссылку на сервис), щелкните правой кнопкой узел проекта и выберите опцию Add Service Reference, как показано на рис. 5-31. В этом диалоговом окне прежде всего необходимо задать адрес метаданных сервиса в поле Address (Адрес) и затем щелкнуть Go (Перейти) для просмотра доступных конечных точек сервисов. После этого в текстовом поле Namespace можно задать пространство имен для автоматически формируемого кода и щелкнуть OK, чтобы добавить прокси в проект.

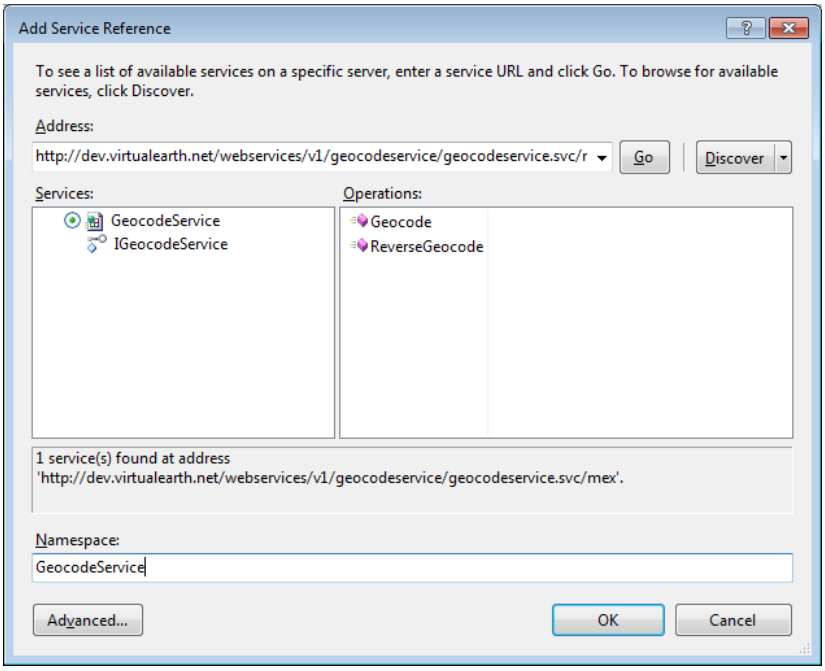


Рис. 5-31 Диалоговое окно Add Service Reference

Подсказка Для просмотра сервисов WCF, включенных в текущее решение, щелкните кнопку Discover (Просмотр).

Дополнительные сведения Чтобы открыть диалоговое око Service Reference Settings (Настройки ссылок на сервисы), щелкните кнопку Advanced (Дополнительно). Это диалоговое окно позволяет настроить конфигурацию прокси WCF-сервиса. Щелкнув кнопку Add Web Service, можно добавить ссылку в стиле .NET Framework 2.0. Более подробно эти настройки рассматриваются на сайте MSDN в разделе

Configure Service Reference Dialog Box (Настройка диалогового окна для добавления ссылки на сервис) по адресу http://msdn.microsoft.com/en-us/library/bb514724(VS.100).aspx.

Автоматически сформированный прокси WCF может использоваться точно так же, как и прокси ASMX, что продемонстрировано в листинге 5-8.

Листинг 5-8 Использование прокси Веб-сервиса

```
public BingCoordinate GeocodeAddress(ActivityAddress address, string token)
{
    ...
    Microsoft.Samples.PlanMyNight.Bing.VEGeocodingService.GeocodeResponse
geocodeResponse = null;
    // Выполнение запроса к сервису геокода
    using (var geocodeService = new
Microsoft.Samples.PlanMyNight.Bing.VEGeocodingService.GeocodeServiceClient())
    {
        try
        {
            geocodeResponse = geocodeService.Geocode(geocodeRequest);
            geocodeService.Close();
        }
        catch
        {
            geocodeService.Abort();
        }
    }

    if (geocodeResponse != null && geocodeResponse.Results != null &&
geocodeResponse.Results.Length > 0)
    {
        var location = geocodeResponse.Results[0].Locations[0];
        return new BingCoordinate { Latitude = (float)location.Latitude, Longitude
= (float)location.Longitude };
    }

    return default(BingCoordinate);
}
```

Разработка многопоточных приложений

С развитием многопроцессорной и многоядерной обработки данных все большее и большее значение приобретает обеспечение разработчикам возможностей для создания многопоточных приложений. Visual Studio 2010 и .NET Framework 4.0 предлагают новые пути реализации многопоточности в приложениях. Библиотека Task Parallel Library (TPL)[1] стала частью библиотеки Base Class Library (BCL)[2] для .NET Framework. Это означает, что теперь любое .NET-приложение может использовать TPL без всяких ссылок на сборки.

Для каждого ItineraryActivity PMN сохраняет в базе данных только идентификатор действия в Bing (Bing Activity ID). Когда приходит время извлечь весь объект Bing Activity (Действие в Bing), для заполнения сущности Bing Activity данными Веб-сервиса Bing Maps используется функция, которая выбирает все ItineraryActivity текущего Itinerary.

Один из способов выполнения этой операции – последовательное обращение к сервису для каждого действия Itinerary, как показано в листинге 5-9. Эта функция, прежде чем выполнять последующий вызов *RetrieveActivity* (Извлечь мероприятие), будет ожидать завершения предыдущего вызова, что делает ее выполнение линейным по времени.

Листинг 5-9 Последовательное извлечение мероприятий

```
public void PopulateItineraryActivities(Itinerary itinerary)
{
    foreach (var item in itinerary.Activities.Where(i =>i.Activity == null))
    {
        item.Activity = this.RetrieveActivity(item.ActivityId);
    }
}
```

[1] Библиотека для параллельного выполнения задач (*прим. переводчика*).

[2] Библиотека базовых классов (*прим. переводчика*).

В прошлом для организации многопоточной обработки этой задачи потребовалось бы использовать потоки и выполнить огромный объем работы. Теперь, с появлением TPL, для этого достаточно использовать статический метод *Parallel.ForEach*, который позаботиться обо всем, что связано с многопоточной обработкой, как показано в листинге 5-10.

Листинг 5-10 Параллельное извлечение мероприятий

```
public void PopulateItineraryActivities(Itinerary itinerary)
{
    Parallel.ForEach(itinerary.Activities.Where(i => i.Activity == null),
        item =>
        {
            item.Activity = this.RetrieveActivity(item.ActivityId);
        });
}
```

Дополнительные сведения .NET Framework 4.0 теперь включает библиотеки Parallel Linq (в System.Core.dll). PLinq представляет расширение .AsParallel для реализации многопоточной обработки в запросах LINQ. Благодаря расширениям .AsOrdered не составляет труда инициировать обработку источника данных, как будто данные были изначально отсортированы. Также в пространство имен System.Collections.Concurrent добавлены новые потокобезопасные коллекции. Более подробные сведения об этих новых возможностях можно найти в разделе Parallel Computing (Многопоточная обработка данных) на сайте MSDN по адресу http://msdn.microsoft.com/en-us/concurrency/default.aspx.

Кэширование AppFabric

PMN – это управляемое данными приложение, которое получает данные из базы данных приложения и от Веб-сервисов Bing Maps. Одна из возможных сложностей при создании Веб-приложения – реализация поддержки большого числа пользователей с обеспечением необходимой производительности и времени отклика. Обращения к хранилищу данных и к сервисам для поиска действий могут значительно повысить использование ресурсов сервера для элементов, совместно используемых множеством пользователей. Например, многие пользователи имеют доступ к общедоступным планам мероприятий. Просмотр этих планов мероприятий обусловит многочисленные обращения к одним и тем же элементам в базе данных. Реализация кэширования на Веб-уровне приведет к сокращению использования ресурсов хранилища данных и снизит задержку при выполнении повторных обращений к Веб-сервисам Bing Maps. На рис. 5-32 представлена архитектура приложения, реализующего кэширование на уровне Веб-сервера.

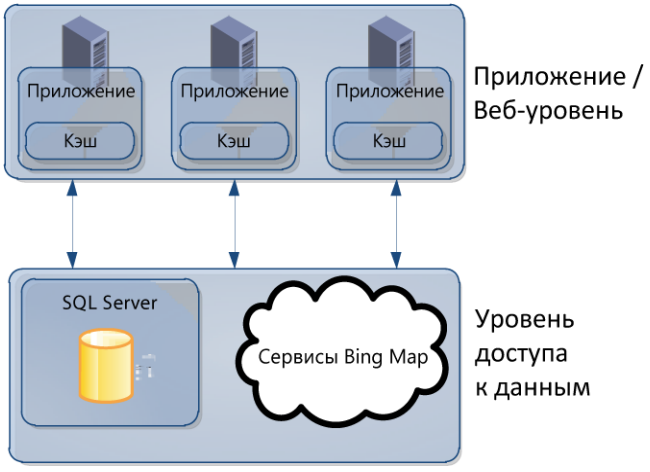


Рис. 5-32 Типовая архитектура Веб-приложения

Применение этого подхода сократит нагрузку на слой доступа к данным, но кэширование по-прежнему связано с конкретным сервером, обслуживающим запрос. Каждый сервер Веб-уровня будет иметь собственный кэш, и по-прежнему существует вероятность неравномерного распределения нагрузки между этими серверами.

Кэширование AppFabric Windows Server предлагает платформу распределенного кэширования в памяти. Клиентская библиотека AppFabric обеспечивает приложению возможность доступа к кэшу как к унифицированному событию просмотра, если кэш распределен на несколько компьютеров, как показано на рис. 5-33. API предоставляет простые методы *get* и *set*, что позволяет без труда извлекать и сохранять сериализуемые объекты общеязыковой среды выполнения (CLR). С кэшем AppFabric вы можете добавлять компьютер для размещения кэша по мере надобности, т.е. масштабирование становится прозрачным для клиента. Другое преимущество – кэш может также разделять копии данных между кластерами, обеспечивая высокую доступность данных даже в случае сбоев.

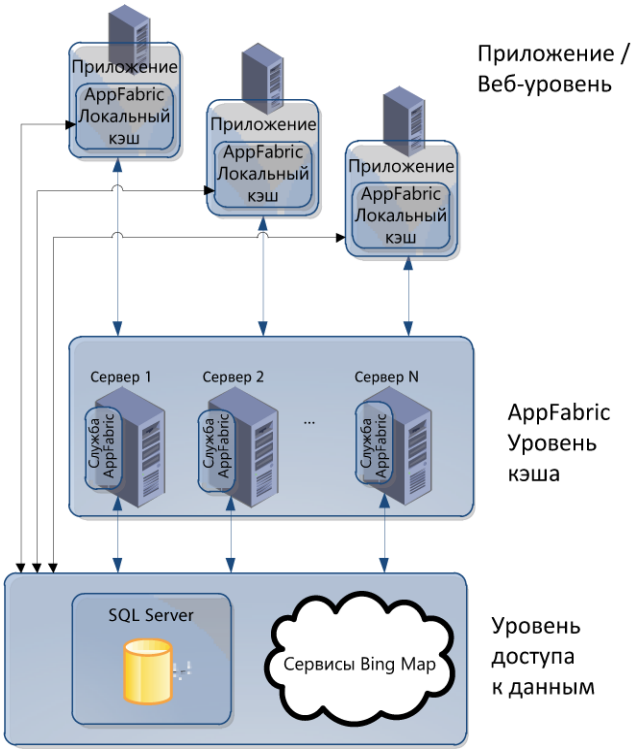


Рис. 5-33 Веб-приложение, использующее кэширование AppFabric Windows Server

Дополнительные сведения Кэширование AppFabric Windows Server предлагается как набор расширений .NET Framework 4.0. Подробные сведения о том, как найти, установить и настроить Windows Server AppFabric, можно найти на сайте Windows Server AppFabric (http://msdn.microsoft.com/en-us/windowsserver/ee695849.aspx).

Дополнительные сведения PMN может быть конфигурирован на использование либо кэширования ASP.NET, либо кэширования AppFabric Windows Server. Подробное пошаговое руководство с описанием того, как добавить кэширование AppFabric Windows Server в PMN предлагается в разделе PMN: Adding Caching using Velocity (Добавление кэширования с использованием Velocity) по адресу http://channel9.msdn.com/learn/courses/VS2010/ASPNET/EnhancingAspNetMvcPlanMyNight/Exercise-1-Adding-Caching-using-Velocity/.

Заключение

В данной главе мы рассмотрели ряд новых возможностей Visual Studio 2010 для структурирования слоя доступа к данным приложения PlanMyNight и использовали Entity Framework v4.0 для организации доступа к базе данных. Также были представлены автоматическое формирование сущностей с помощью POCO-шаблонов ADO.NET Entity Framework и расширения кэширования AppFabric Windows Server.

В следующей главе мы займемся созданием замечательных Веб-приложений с помощью инфраструктуры ASP.NET MVC и Managed Extensibility Framework (Инфраструктура расширения возможностей).

Глава 6

От 2005 к 2010: проектирование внешнего вида и поведения

В данной главе рассматривается

- Создание контроллера ASP.NET MVC, взаимодействующего с моделью данных
- Создание представления ASP.NET MVC для отображения данных контроллера и проверки пользовательского ввода
- Расширение приложения внешним подключаемым модулем с помощью Managed Extensibility Framework

За годы, прошедшие с момента выхода ASP.NET 1.0, разработка Веб-приложений в Microsoft Visual Studio, безусловно, существенно улучшилась. Visual Studio 2005 и .NET Framework 2.0 включили такие возможности, как более эффективное хранение состояния форм, частичные классы, универсальные типы и многое другое, благодаря чему разработчики смогли создавать эффективные простые в обслуживании приложения.

Жажда улучшений, призванных помочь разработчикам в создании высококлассных приложений, не иссякла и в Visual Studio 2010. В этой главе будут рассмотрены некоторые новые возможности на примере расширения функциональности используемого нами для примера приложения Plan My Night.

> **Примечание** Приложение-пример – это проект ASP.NET MVC 2, но в Visual Studio 2010 у Веб-разработчика есть выбор использовать либо новую форму приложения ASP.NET, либо более традиционную ASP.NET (которую в сообществе разработчиков *для отличия называют Веб-формами* (Web Forms)). ASP.NET 4.0 была существенно дополнена, чтобы облегчить жизнь разработчиков и при этом обеспечить крайне эффективный подход к созданию Веб-приложений.

В данной главе будет использоваться модифицированная форма решения приложения-примера. Если сопроводительные материалы данной книги установлены в каталог по умолчанию, интересующее нас решение находится в Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 6\ в папке UserInterface-Start.

Введение в проект PlanMyNight.Web

Пользовательский интерфейс Plan My Night в Visual Studio 2010 реализован как ASP.NET MVC-приложение, компоновка которого отличается от того, к чему, возможно, привыкли разработчики, создавая приложения Веб-форм ASP.NET в Visual Studio 2005. Некоторые элементы проекта (как видно на рис. 6-1) будут знакомыми (такие как Global.asax), но все остальные абсолютно новые. Некоторые из них являются обязательными для инфраструктуры ASP.NET MVC.

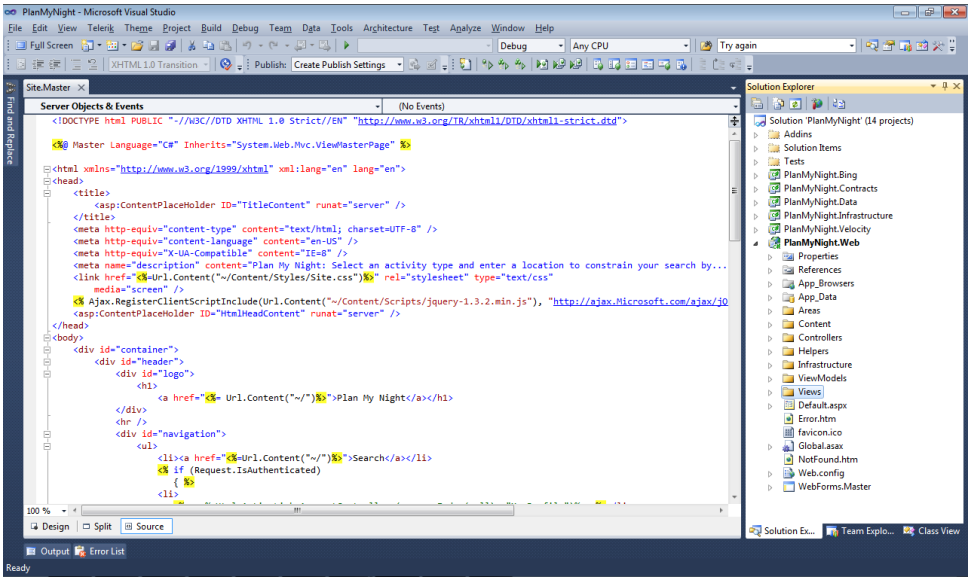


Рис. 6-1 Внешний вид проекта PlanMyNight.Web

Обязательные элементы ASP.NET MVC:

- **Areas (Области)** Эта папка используется инфраструктурой ASP.NET MVC для организации больших Веб-приложений в небольшие компоненты без разделения решений или проектов. Эта возможность не используется в приложении Plan My Night, но представлена здесь, потому что данная папка создается шаблоном MVC-проекта.
- **Controllers (Контроллеры)** Во время обработки запроса инфраструктура ASP.NET MVC ищет контроллеры для обработки запроса в этой папке.
- **Views (Представления)** Папка Views на самом деле является структурой каталогов. Его подпапки носят имена соответствующие классам в папке Controllers. Также имеется подпапка Shared (Общие). Она предназначена для представлений, частичных представлений, главных страниц и любых других ресурсов, которые должны быть доступны всем котроллерам.

Дополнительные сведения Более подробные сведения о компонентах ASP.NET MVC, а также отличия обработки запросов от того, как это происходит в Веб-формах ASP.NET, представлены по адресу http://asp.net/mvc.

В большинстве случаев web.config является последним файлом в корневой папке проекта. В Visual Studio 2010 для этого файла предлагается существенное обновление: Преобразование Web.config. Эта возможность позволяет создавать на базе основного web.config файлы web.config для конкретных сборок, переопределяющие настройки базового файла во время сборки, развертывания и выполнения. Эти файлы отображаются под базовым web.config, как показано на рис. 6-2.

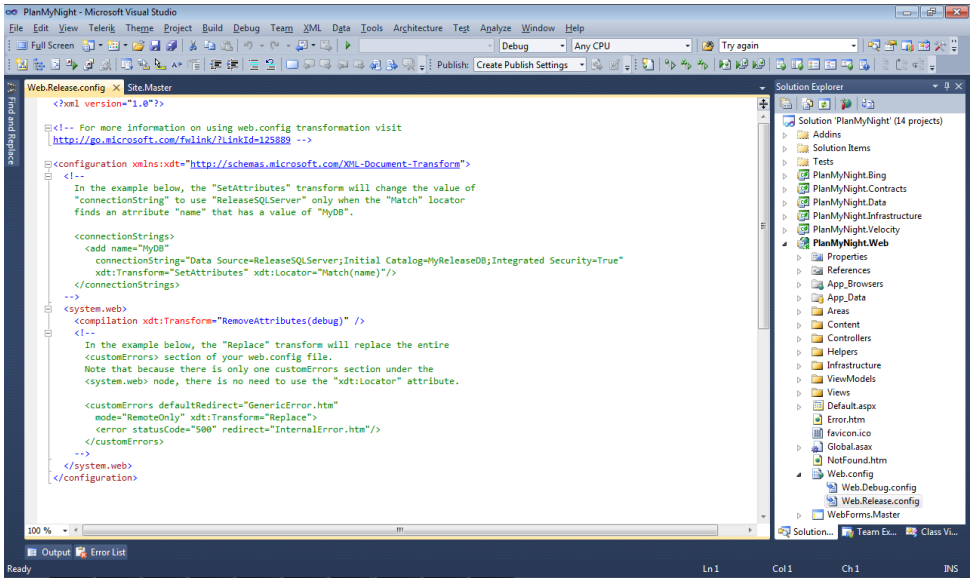


Рис. 6-2 Файл Web.config и специальные файлы конфигурации для конкретных сборок

> **Visual Studio 2005** В Visual Studio 2005 приходилось всегда помнить о том, что параметры в web.config не должны переопределяться параметрами отладки. Или нельзя было забывать вносить в web.config правильные настройки после его публикации для окончательной сборки. В Visual Studio 2010 таких проблем нет. Для переопределения значений в web.config при окончательных сборках используется набор параметров файла web.Release.config; при отладочных сборках используется файл web.Debug.config.

Проект также включает разделы:

- **Content (Содержимое)** Набор папок, содержащих изображения, сценарии и файлы стилей.
- **Helpers (Вспомогательные классы)** Различные классы, включающие ряд методов расширения, которые расширяют функциональность типов, используемых в проекте.
- **Infrastructure (Инфраструктура)** В данную папку включены элементы, обеспечивающие взаимодействие с более низкоуровневой инфраструктурой ASP.NET MVC (например, фабрики кэширования и контроллеров).
- **ViewModels (Модели представлений)** Объекты данных, заполненные классами контроллеров (Controller) и используемые представлениями (Views) для отображения данных.

Запуск проекта

Если скомпилировать и запустить проект, на экране должно быть выведено следующее (рис. 6-3).

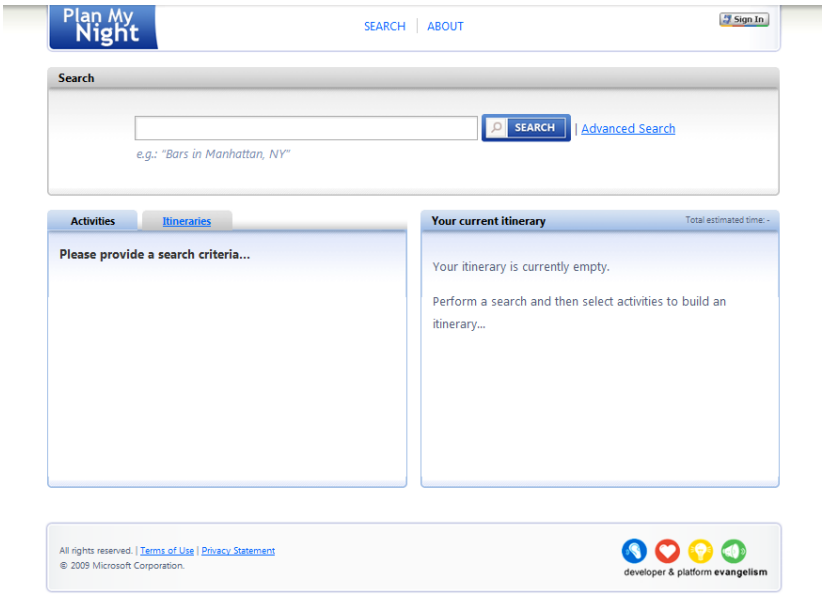


Рис. 6-3 Страница по умолчанию приложения Plan My Night

Поиск и организация исходного списка элементов плана мероприятий функционируют, но если попытаться сохранить разрабатываемый план мероприятий или выполнить вход с использованием Windows Live ID, приложение возвратит ошибку 404 Not Found (Не найден) (как показано на рис. 6-4).

Рис. 6-4 Ошибка, возвращаемая приложением Plan My Night при попытке регистрации

Это происходит потому, что на данный момент в проекте нет контроллера учетной записи для обработки таких запросов.

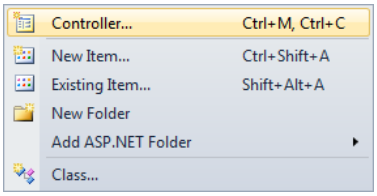
Создание контроллера учетной записи

Класс *AccountController* (Контроллер учетной записи) обеспечивает приложение Plan My Night критически важными функциями:

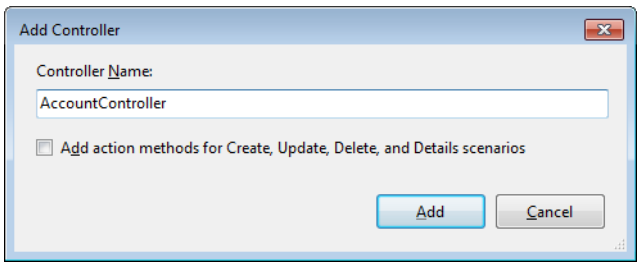
- Он обрабатывает вход и выход пользователей из приложения (с использованием Windows Live ID).
- Обеспечивает действия для отображения и обновления данных профиля пользователя.

Для создания нового контроллера ASP.NET MVC:

73. В Solution Explorer (Обозреватель решений) перейдите к папке Controllers проекта PlanMyNight.Web и щелкните ее правой кнопкой мыши.
74. Откройте подменю Add и выберите пункт Controller.



75. Введите имя контроллера, **AccountController**.



Примечание Не устанавливайте флажок Add Action Methods For Create, Update, And Delete Scenarios (Добавить методы для сценариев создания, обновления и удаления). Установка этого флажка обеспечит вставку некоторых методов-«заглушек», но поскольку мы не будем использовать методы по умолчанию, в их создании нет необходимости

По щелчку кнопки Add в диалоговом окне Add Controller откроется базовый класс *AccountController* с единственным методом *Index ()*:

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

namespace Microsoft.Samples.PlanMyNight.Web.Controllers
{

    public class AccountController : Controller
    {
        //
        // GET: /Account/

        public ActionResult Index()
        {
            return View();
        }

    }
}
```

Visual Studio 2005 Следует отметить отличие от разработки приложений ASP.NET Web Forms в Visual Studio 2005: в приложениях ASP.NET MVC для файлов .aspx не создаются файлы выделенного кода. Котроллеры, созданием которого мы только что занимались, осуществляют логику, необходимую для обработки ввода и подготовки вывода. Такой подход обеспечивает четкое разделение логики отображения и бизнес-логики, и это является ключевым аспектом ASP.NET MVC.

Реализация функциональности

Для взаимодействия с любым слоем доступа к данным и сервисами (Моделью) потребуется создать и инициализировать некоторые поля экземпляров. Но перед этим следует добавить ряд пространств имен в блок директив *using*:

```
using System.IO;
using Microsoft.Samples.PlanMyNight.Data;
using Microsoft.Samples.PlanMyNight.Entities;
using Microsoft.Samples.PlanMyNight.Infrastructure;
using Microsoft.Samples.PlanMyNight.Infrastructure.Mvc;
using Microsoft.Samples.PlanMyNight.Web.ViewModels;
using System.Collections.Specialized;
using WindowsLiveId;
```

Вот теперь добавим поля экземпляров. Эти поля являются интерфейсами различных разделов Модели:

```
public class AccountController : Controller
{
    private readonly IWindowsLiveLogin windowsLogin;
    private readonly IMembershipService membershipService;
    private readonly IFormsAuthentication formsAuthentication;
    private readonly IReferenceRepository referenceRepository;
```

```
    private readonly IActivitiesRepository activitiesRepository;
.
.
.
```

Примечание Использование интерфейсов для взаимодействия со всеми внешними зависимостями обеспечивает лучшую портируемость кода на различные платформы. Также интерфейсы более эффективно изолируют отдельные компоненты, что облегчает моделирование зависимостей при тестировании

Как уже упоминалось, эти поля представляют части Модели, с которой будет взаимодействовать данный контроллер для реализации функциональных требований. Рассмотрим общие описания каждого из этих интерфейсов:

- **IWindowsLiveLogin (Регистрация WindowsLive)** Обеспечивает функциональность для взаимодействия с сервисом Windows Live ID.
- **IMembershipService (Сервис членства)** Обеспечивает данные профиля пользователя и методы авторизации. В используемом приложении-примере это абстракция ASP.NET Membership Service (Сервис членства).
- **IFormsAuthentication (Аутентификация с помощью форм)** Абстракция ASP.NET Forms Authentication (Аутентификация с помощью форм).
- **IReferenceRepository (Хранилище справочных ресурсов)** Обеспечивает справочные ресурсы, такие как списки состояний и другие характерные для модели данные.
- **IActivitiesRepository (Хранилище действий)** Интерфейс для извлечения и обновления данных действий.

В этот класс добавим два конструктора: один – общий для времени выполнения, который использует класс *ServiceFactory* (Фабрика сервисов) для получения ссылок на необходимые интерфейсы, и другой – для обеспечения возможности вводить конкретные экземпляры интерфейсов при тестировании.

```
public AccountController() :
    this(
        new ServiceFactory().GetMembershipService(),
        new WindowsLiveLogin(true),
        new FormsAuthenticationService(),
        new ServiceFactory().GetReferenceRepositoryInstance(),
        new ServiceFactory().GetActivitiesRepositoryInstance())
{
}
public AccountController(
    IMembershipService membershipService,
    IWindowsLiveLogin windowsLogin,
    IFormsAuthentication formsAuthentication,
    IReferenceRepository referenceRepository,
    IActivitiesRepository activitiesRepository)
{
    this.membershipService = membershipService;
    this.windowsLogin = windowsLogin;
    this.formsAuthentication = formsAuthentication;
    this.referenceRepository = referenceRepository;
    this.activitiesRepository = activitiesRepository;
}
```

Аутентификация пользователя

Первой настоящей функциональностью, реализованной нами в этом контроллере, будет вход и выход из приложения. Большинство методов, которые будут реализовываться в дальнейшем, требуют аутентификации, поэтому именно с нее мы и начнем.

В нашем приложении для обеспечения аутентификации используется одновременно несколько технологий: Windows Live ID, ASP.NET Forms Authentication и ASP.NET Membership Services. Эти три технологии применяются в действии LiveID, которое будет реализовано следующим.

Начнем с создания в классе *AccountController* следующего метода:

```
public ActionResult LiveId()
{
    return Redirect("~/");
}
```

Это основной метод для взаимодействия с сервисами Windows Live ID. Если вызвать его прямо сейчас, он просто перенаправит пользователя к корню приложения.

> **Примечание** Вызов *Redirect* (Перенаправить) возвращает *RedirectResult* (Результат перенаправления). В данном примере для определения цели перенаправления используется строка, но в разных ситуациях могут использоваться различные перегрузки этого метода

Когда Windows Live ID возвращает пользователя в приложение, может быть предпринято несколько разных типов действий. Пользователь может выполнить вход в Windows Live ID, выполнить выход или очистить «cookies» Windows Live ID. Когда Windows Live ID возвращает пользователя, в его URL присутствует строковый параметр запроса *action*, поэтому переключение ветвей логики выполняется на основании значения этого параметра.

Добавим следующий код в метод *LiveId* над выражением *return*:

```
string action = Request.QueryString["action"];
switch (action)
{
    case "logout":
        this.formsAuthentication.SignOut();
        return Redirect("~/");

    case "clearcookie":
        this.formsAuthentication.SignOut();
        string type;
        byte[] content;
        this.windowsLogin.GetClearCookieResponse(out type, out content);
        return new FileStreamResult(new MemoryStream(content), type);
}
```

Дополнительные сведения Полную документацию по системе Windows Live ID можно найти по адресу http://dev.live.com/.

В только что добавленном коде обрабатываются два действия выхода для Windows Live ID. В обоих случаях используется интерфейс *IFormsAuthentication* для удаления файла «cookie» ASP.NET Forms Authentication, чтобы все будущие http-запросы (до момента повторной регистрации пользователя) не считались аутентифицированными. Во втором случае выполняется на одну операцию больше, и очищаются файлы «cookies» Windows Live ID (те, в которых сохраняется регистрационное имя, но не пароль).

Сценарий входа включает немного больший объем кода, поскольку требуется проверка присутствия аутентифицируемого пользователя в базе данных сервиса членства (Membership Database). Если этого пользователя там нет, для него создается новый профиль. Однако перед этим должны быть переданы данные, которые Windows Live ID переслал в интерфейс Windows Live ID, что позволит проверить их и предоставить объект *WindowsLiveLogin.User*:

```
default:
    // вход
    NameValueCollection tokenContext;
    if ((Request.HttpMethod ?? "GET").ToUpperInvariant() == "POST")
    {
        tokenContext = Request.Form;
    }
    else
    {
        tokenContext = new NameValueCollection(Request.QueryString);
        tokenContext["stoken"] =
            System.Web.HttpUtility.UrlEncode(tokenContext["stoken"]);
    }

    var liveIdUser = this.windowsLogin.ProcessLogin(tokenContext);
```

На данном этапе, если пользователь выполнит вход, *liveIdUser* будет либо ссылаться на аутентифицированный объект *WindowsLiveLogin.User*, либо будет null. Исходя из этого, добавим следующий раздел кода, который обеспечивает выполнение некоторых действий, если значение *liveIdUser* не null:

```
    if (liveIdUser != null)
    {
        var returnUrl = liveIdUser.Context;
        var userId = new Guid(liveIdUser.Id).ToString();
        if (!this.membershipService.ValidateUser(userId, userId))
        {
            this.formsAuthentication.SignIn(userId, false);
            this.membershipService.CreateUser(userId, userId, string.Empty);
```

```
                var profile = this.membershipService.CreateProfile(userId);
                profile.FullName = "New User";
                profile.State = string.Empty;
                profile.City = string.Empty;
                profile.PreferredActivityTypeId = 0;
                this.membershipService.UpdateProfile(profile);

                if (string.IsNullOrEmpty(returnUrl)) returnUrl = null;
                return RedirectToAction("Index", new { returnUrl = returnUrl });
            }
            else
            {
                this.formsAuthentication.SignIn(userId, false);
                if (string.IsNullOrEmpty(returnUrl)) returnUrl = "~/";
                return Redirect(returnUrl);
            }
        }
    break;
```

Вызов метода *ValidateUser* (Проверить пользователя) в *IMembershipService* позволяет приложению проверить, посещал ли данный пользователь этот сайт ранее и существует ли его профиль. Поскольку аутентификация пользователя выполняется по Windows Live ID, значение его ID (которое является GUID) используется и как имя пользователя, и как пароль для ASP.NET Membership Service.

Если в приложении нет записи для данного пользователя, она создается путем вызова метода *CreateUser* (Добавить пользователя). Затем посредством *CreateProfile* (Создать профиль) создается и профиль параметров пользователя. Профиль заполняется некоторыми значениями по умолчанию и сохраняется в хранилище. Пользователь перенаправляется на основную страницу ввода, где может обновить сведения.

> **Примечание** На основании сочетания входных параметров *Controller.RedirectToAction* определяет, какой URL должен быть создан. В данном случае требуется перенаправить пользователя к действию *Index* (Корень) этого контроллера и передать текущее значение URL-адреса возврата

Также в данном коде реализована регистрация пользователя в сервисе аутентификации ASP.NET Forms, т.е. обеспечивается создание файла «cookie» с идентификационными данными для использования в запросах, требующих аутентификации, в будущем.

Параметры профиля также управляются сервисами ASP.NET Membership Services и объявляются в файле web.config приложения:

```
<system.web>
…
<profile enabled="true">
    <properties>
        <add name="FullName" type="string" />
        <add name="State" type="string" />
        <add name="City" type="string" />
        <add name="PreferredActivityTypeId" type="int" />
    </properties>

    <providers>
        <clear />
        <add name="AspNetSqlProfileProvider"
type="System.Web.Profile.SqlProfileProvider,
             System.Web, Version=4.0.0.0, Culture=neutral,
             PublicKeyToken=b03f5f7f11d50a3a"
        connectionStringName="ApplicationServices"
        applicationName="/" />
    </providers>
</profile>
…
</system.web>
```

Теперь метод *LiveID* полностью готов и должен выглядеть, как следующий фрагмент кода. Приложение принимает сведения для аутентификации от Windows Live ID, подготавливает профиль ASP.NET Membership Service и создает маркер аутентификации ASP.NET Forms.

```
public ActionResult LiveId()
{
string action = Request.QueryString["action"];
switch (action)
{
```

```
        case "logout":
            this.formsAuthentication.SignOut();
            return Redirect("~/");

        case "clearcookie":
            this.formsAuthentication.SignOut();
            string type;
            byte[] content;
            this.windowsLogin.GetClearCookieResponse(out type, out content);
            return new FileStreamResult(new MemoryStream(content), type);

        default:
            // вход
            NameValueCollection tokenContext;
            if ((Request.HttpMethod ?? "GET").ToUpperInvariant() == "POST")
            {
                tokenContext = Request.Form;
             }
            else
            {
                tokenContext = new NameValueCollection(Request.QueryString);
                tokenContext["stoken"] =
                    System.Web.HttpUtility.UrlEncode(tokenContext["stoken"]);
            }
            var liveIdUser = this.windowsLogin.ProcessLogin(tokenContext);

            if (liveIdUser != null)
            {
                var returnUrl = liveIdUser.Context;
                var userId = new Guid(liveIdUser.Id).ToString();
                if (!this.membershipService.ValidateUser(userId, userId))
                {
                    this.formsAuthentication.SignIn(userId, false);
                    this.membershipService.CreateUser(userId, userId,
                                                      string.Empty);
                    var profile = this.membershipService.CreateProfile(userId);
                    profile.FullName = "New User";
                    profile.State = string.Empty;
                    profile.City = string.Empty;
                    profile.PreferredActivityTypeId = 0;
                    this.membershipService.UpdateProfile(profile);

                    if (string.IsNullOrEmpty(returnUrl)) returnUrl = null;
                    return RedirectToAction("Index",
                                            new { returnUrl = returnUrl });
                }
                else
                {
                    this.formsAuthentication.SignIn(userId, false);
                    if (string.IsNullOrEmpty(returnUrl)) returnUrl = "~/";
                    return Redirect(returnUrl);
                }
            }
        break;
    }
    return Redirect("~/");
    }
```

Конечно, пользователь должен иметь возможность перед регистрацией попасть на страницу входа Windows Live ID. На данный момент в приложении Plan My Night имеется кнопка входа с помощью Windows Live ID. Но возможны ситуации, когда понадобится перенаправить пользователя на страницу входа прямо из кода. Для реализации такого сценария добавим в контроллер небольшой метод под именем *Login* (Вход):

```
    public ActionResult Login(string returnUrl)
    {
        var redirect = HttpContext.Request.Browser.IsMobileDevice ?
                       this.windowsLogin.GetMobileLoginUrl(returnUrl) :
                       this.windowsLogin.GetLoginUrl(returnUrl);
        return Redirect(redirect);
    }
```

Данный метод просто извлекает URL-адрес входа для Windows Live и перенаправляет пользователя по этому адресу. Это также удовлетворяет конфигурации для аутентификации ASP.NET Forms в web.config с той точки зрения, что любой запрос, требующий аутентификации, будет перенаправлен в этот метод:

```
<authentication mode="Forms">
    <forms loginUrl="~/Account/Login" name="XAUTH" timeout="2880" path="~/" />
</authentication>
```

Извлечение профиля для текущего пользователя

Теперь, когда методы аутентификации описаны, что реализует первую цель данного контролера – обработка входа и выхода пользователей из приложения – можно переходить к извлечению данных текущего пользователя.

Метод *Index* является методом по умолчанию для контроллера на основании конфигурации отображения URL в Global.asax. Он будет располагаться там, где должны извлекаться данные текущего пользователя и возвращаться представление, отображающее эти данные. Метод *Index*, изначально описывавшийся при создании класса *AccountController*, необходимо заменить следующим:

```
 [Authorize()]
[AcceptVerbs(HttpVerbs.Get)]
public ActionResult Index(string returnUrl)
{
        var profile = this.membershipService.GetCurrentProfile();
        var model = new ProfileViewModel
        {
            Profile = profile,
            ReturnUrl = returnUrl ?? this.GetReturnUrl()
        };

        this.InjectStatesAndActivityTypes(model);

        return View("Index", model);
}
```

> **Visual Studio 2005** Атрибуты, такие как *[Authorize()]*, не имели такого широкого распространения в Visual Studio 2005, но в ASP.NET MVC они часто используются. Атрибуты позволяют описывать метаданные цели, к которой они относятся. Благодаря этому появляется возможность проверять данные во время выполнения (через отражение) и предпринимать действия в случае необходимости.

Очень удобен атрибут *Authorize* (Авторизовать). Он показывает, что данный метод может вызываться только для уже аутентифицированных http-запросов. Если запрос не аутентифицирован, он будет перенаправлен в заданную цель входа ASP.NET Forms Authentication, которая только что была настроена. Атрибут *AcceptVerbs* (Допустимые команды) также ограничивает способы вызова этого метода, задавая допустимые Http-команды. В данном случае этот метод вызывается по запросу HTTP GET. В сигнатуру метода добавлен строковый параметр *returnUrl*. Он обеспечивает возможность возвращения пользователя на исходный адрес по завершении просмотра или обновления его сведений.

> **Примечание** Это приводит нас к части инфраструктуры ASP.NET MVC под названием *Привязка модели* (*Model Binding*), подробное рассмотрение которой выходит за рамки данной книги. Однако необходимо знать, что она пытается найти источник *returnUrl* (поле формы, данные таблицы маршрутизации или параметр строки запроса с таким же именем) и выполняет его привязку к этому значению при вызове данного метода. Если средству привязки модели не удается найти подходящего источника, значение будет null. Такое поведение может обусловить проблемы для типов значений, которые не могут быть null, потому что приведет к формированию исключения *InvalidOperationException* (Недопустимая операция).

В целом, этот метод прост: он принимает возвращаемое значение метода *GetCurrentProfile* (Получить профиль текущего пользователя) интерфейса ASP.NET Membership Service и настраивает объект модели представления, который будет использоваться представлением. Вызов *GetReturnUrl* (Получить URL возврата) – это пример метода расширения, описанного в проекте PlanMyNight.Infrastructure. Он не является членом класса Controller, но в среде разработки обеспечивает намного более надежный код (рис. 6-5).

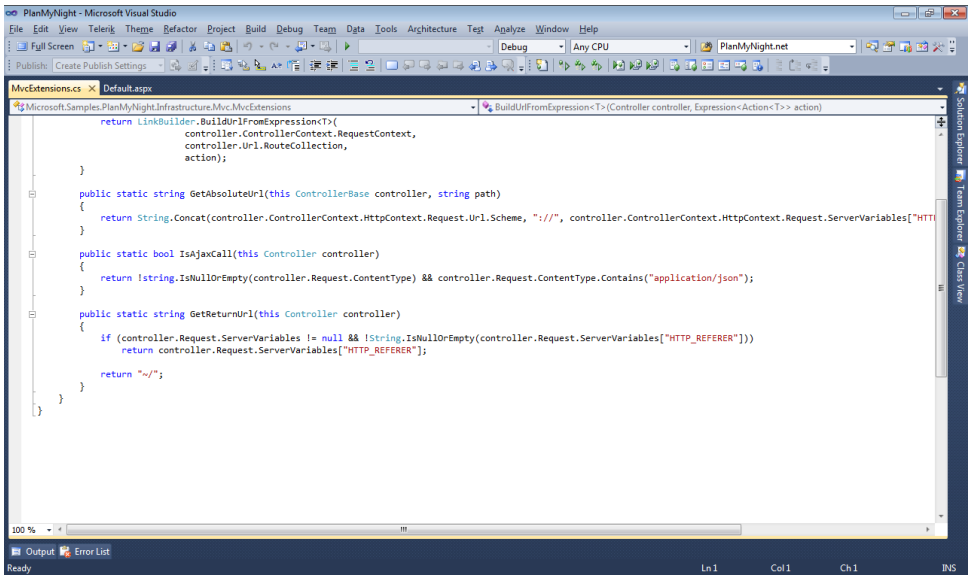


Рис. 6-5 Пример методов расширения в MvcExtensions.cs

> **Visual Studio 2005** В .NET Framework 2.0, используемой Visual Studio 2005, методом расширений не было. Сейчас мы просто вызываем *this.GetReturnUrl()*, и этот метод появляется в списке IntelliSense для этого объекта. Ранее же приходилось вводить **MvcExtensions.GetReturnUrl(this)**, передавая контроллер в качестве параметра. Безусловно, методы расширения делают код более удобным для восприятия, при этом от разработчика не требуется знания статического класса, в котором описан этот метод расширения. Для обеспечения работы IntelliSense это пространство имен просто должно быть указано в директивах *using*.

InjectStatesAndActivityTypes (Ввести состояния и типы действий) – метод, который требуется реализовать. Он собирает данные имен состояния из хранилища справочных ресурсов и данные о типах действий из хранилища действий. Он создает две коллекции *SelectListItem* (HTML-класс для MVC): одну для списка состояний, и другую для списка разных типов действий, доступных в приложении. Также он задает соответствующее значение.

```
private void InjectStatesAndActivityTypes(ProfileViewModel model)
{
    var profile = model.Profile;
    var types = this.activitiesRepository.RetrieveActivityTypes().Select(
                o => new SelectListItem {
                          Text = o.Name,
                          Value = o.Id.ToString(),
                          Selected = (profile != null && o.Id ==
                                      profile.PreferredActivityTypeId)
                }).ToList();

    types.Insert(0, new SelectListItem { Text = "Select...", Value = "0" });
    var states = this.referenceRepository.RetrieveStates().Select(
                 o => new SelectListItem {
                           Text = o.Name,
                           Value = o.Abbreviation,
                           Selected = (profile != null && o.Abbreviation ==
                                       profile.State)
                }).ToList();

    states.Insert(0, new SelectListItem {
                          Text = "Any state",
                          Value = string.Empty
                });

    model.PreferredActivityTypes = types;
    model.States = states;
}
```

> **Visual Studio 2005** В Visual Studio 2005 реализовать метод *InjectStatesAndActivities* сложнее, потому что разработчик не может использовать расширения LINQ (вызов *Select*) и лямбда выражения, которые являются формой анонимного делегата, применяемого методом *Select* к каждому перечисляемому члену коллекции. Вместо этого разработчику пришлось бы писать собственный цикл и выполнять перечисление всех элементов вручную.

Обновление данных профиля

Закончив с инфраструктурой для извлечения данных текущего профиля, перейдем к обновлению данных модели посредством формы, передаваемой пользователем. После этого можно создавать собственные страницы представления и видеть, как все это сочетается. Метод *Update* прост, однако, в нем появляются некоторые новые до сих пор не встречавшиеся нам возможности:

```
[Authorize()]
[AcceptVerbs(HttpVerbs.Post)]
[ValidateAntiForgeryToken()]
public ActionResult Update(UserProfile profile)
{
    var returnUrl = Request.Form["returnUrl"];
    if (!ModelState.IsValid)
    {
        // ошибка валидации
        return this.IsAjaxCall() ? new JsonResult { JsonRequestBehavior =
                    JsonRequestBehavior.AllowGet, Data = ModelState }
                    : this.Index(returnUrl);
    }

    this.membershipService.UpdateProfile(profile);
    if (this.IsAjaxCall())
    {
        return new JsonResult { JsonRequestBehavior =
                                JsonRequestBehavior.AllowGet,
                                Data = new { Update = true,
                                             Profile = profile,
                                             ReturnUrl = returnUrl } };
    }
    else
    {
        return RedirectToAction("UpdateSuccess", "Account", new { returnUrl =
                                 returnUrl });
    }
}
```

Атрибут *ValidateAntiForgeryToken* (Проверить маркер, препятствующий фальсификации) подтверждает то, что форма не была сфальсифицирована. Для использования этой возможности необходимо добавить *AntiForgeryToken* (Маркер, препятствующий фальсификации) в форму ввода представления. Проверка действительности *ModelState* (Состояние модели) – наш первый опыт валидации ввода. Это вализация на стороне сервера, и ASP.NET MVC предлагает простую в использовании возможность убедиться в том, что поступающие данные удовлетворяют некоторым правилам. Одно из свойств объекта *UserProfile*, созданного для обеспечения ввода в этот метод через привязку модели MVC, имеет атрибут *System.ComponentModel.DataAnnotations.Required*. При привязке модели инфраструктура MVC проверяет атрибуты *DataAnnotation* (Аннотация данных) и объявляет *ModelState* действительным только в случае выполнения всех правил.

В случае, когда *ModelState* недействительный, пользователь перенаправляется в метод *Index*, где *ModelState* используется для отображения формы ввода. Или, если запрос был AJAX-вызовом, возвращается *JsonResult* с прикрепленными к нему данными *ModelState*.

> **Visual Studio 2005** В ASP.NET MVC запросы маршрутизируются не через страницы, а через контроллеры. Поэтому один URL может обрабатывать множество разных запросов и отвечать на них соответствующим представлением. В Visual Studio 2005 для реализации этой функциональности разработчику пришлось бы создавать два разных URL и вызывать метод в третьем классе

Если *ModelState* действительный, сервис членства обновляет профиль, и для AJAX-запросов возвращается JSON-результат со сведениями об успешности операции. Для «обычных» запросов пользователь перенаправляется к действию *UpdateSuccess* (Успешное обновление) контроллера Account. Метод *UpdateSuccess* – последний метод, необходимый для завершения выполнения данного контроллера:

```
public ActionResult UpdateSuccess(string returnUrl)
{
    var model = new ProfileViewModel
    {
        Profile = this.membershipService.GetCurrentProfile(),
        ReturnUrl = returnUrl
    };
    return View(model);
}
```

Этот метод используется для возвращения представления успешного выполнения в браузер, отображения некоторых обновленных данных и обеспечения ссылки для возвращения пользователя туда, где он находился перед началом процесса обновления профиля.

Теперь, когда контроллер Account полностью реализован, полученный класс должен выглядеть следующим образом:

```
    using System;
    using System.Collections.Specialized;
    using System.IO;
    using System.Linq;
    using System.Web;
    using System.Web.Mvc;
    using Microsoft.Samples.PlanMyNight.Data;
    using Microsoft.Samples.PlanMyNight.Entities;
    using Microsoft.Samples.PlanMyNight.Infrastructure;
    using Microsoft.Samples.PlanMyNight.Infrastructure.Mvc;
    using Microsoft.Samples.PlanMyNight.Web.ViewModels;
    using WindowsLiveId;

namespace Microsoft.Samples.PlanMyNight.Web.Controllers
{
    [HandleErrorWithContentType()]
    [OutputCache(NoStore = true, Duration = 0, VaryByParam = "*")]
    public class AccountController : Controller
    {
        private readonly IWindowsLiveLogin windowsLogin;
        private readonly IMembershipService membershipService;
        private readonly IFormsAuthentication formsAuthentication;
        private readonly IReferenceRepository referenceRepository;
        private readonly IActivitiesRepository activitiesRepository;

        public AccountController() :
            this(
                new ServiceFactory().GetMembershipService(),
                new WindowsLiveLogin(true),
                new FormsAuthenticationService(),
                new ServiceFactory().GetReferenceRepositoryInstance(),
                new ServiceFactory().GetActivitiesRepositoryInstance())
        {
        }

        public AccountController(IMembershipService membershipService,
                                 IWindowsLiveLogin windowsLogin,
                                 IFormsAuthentication formsAuthentication,
                                 IReferenceRepository referenceRepository,
                                 IActivitiesRepository activitiesRepository)
        {
            this.membershipService = membershipService;
            this.windowsLogin = windowsLogin;
            this.formsAuthentication = formsAuthentication;
            this.referenceRepository = referenceRepository;
            this.activitiesRepository = activitiesRepository;
        }

        public ActionResult LiveId()
        {
            string action = Request.QueryString["action"];
            switch (action)
            {
                case "logout":
                    this.formsAuthentication.SignOut();
                    return Redirect("~/");
                case "clearcookie":
                    this.formsAuthentication.SignOut();
                    string type;
                    byte[] content;
                    this.windowsLogin.GetClearCookieResponse(out type, out
content);
                    return new FileStreamResult(new MemoryStream(content), type);
                default:
```

```
                // вход
                NameValueCollection tokenContext;
                if ((Request.HttpMethod ?? "GET").ToUpperInvariant() == "POST")
                {
                    tokenContext = Request.Form;
                }
                else
                {
                    tokenContext = new
                 NameValueCollection(Request.QueryString);
                    tokenContext["stoken"] =
                 System.Web.HttpUtility.UrlEncode(tokenContext["stoken"]);
                }

                var liveIdUser =
                        this.windowsLogin.ProcessLogin(tokenContext);
                if (liveIdUser != null)
                {
                    var returnUrl = liveIdUser.Context;
                    var userId = new Guid(liveIdUser.Id).ToString();
                    if (!this.membershipService.ValidateUser(userId, userId))
                    {
                        this.formsAuthentication.SignIn(userId, false);
                        this.membershipService.CreateUser( userId, userId,
                                                           string.Empty);
                        var profile =
                            this.membershipService.CreateProfile(userId);
                        profile.FullName = "New User";
                        profile.State = string.Empty;
                        profile.City = string.Empty;
                        profile.PreferredActivityTypeId = 0;
                        this.membershipService.UpdateProfile(profile);
                        if (string.IsNullOrEmpty(returnUrl)) returnUrl =
                                                             null;
                        return RedirectToAction("Index", new { returnUrl =
                                                             returnUrl });
                    }
                    else
                    {
                        this.formsAuthentication.SignIn(userId, false);
                        if (string.IsNullOrEmpty(returnUrl)) returnUrl =
                                                             "~/";
                        return Redirect(returnUrl);
                    }
                }
                break;
            }
            return Redirect("~/");
        }

        public ActionResult Login(string returnUrl)
        {
            var redirect = HttpContext.Request.Browser.IsMobileDevice ?
                      this.windowsLogin.GetMobileLoginUrl(returnUrl) :
                      this.windowsLogin.GetLoginUrl(returnUrl);
            return Redirect(redirect);
        }

        [Authorize()]
        [AcceptVerbs(HttpVerbs.Get)]
        public ActionResult Index(string returnUrl)
        {
            var profile = this.membershipService.GetCurrentProfile();
            var model = new ProfileViewModel
            {
                Profile = profile,
                ReturnUrl = returnUrl ?? this.GetReturnUrl()
            };

            this.InjectStatesAndActivityTypes(model);
```

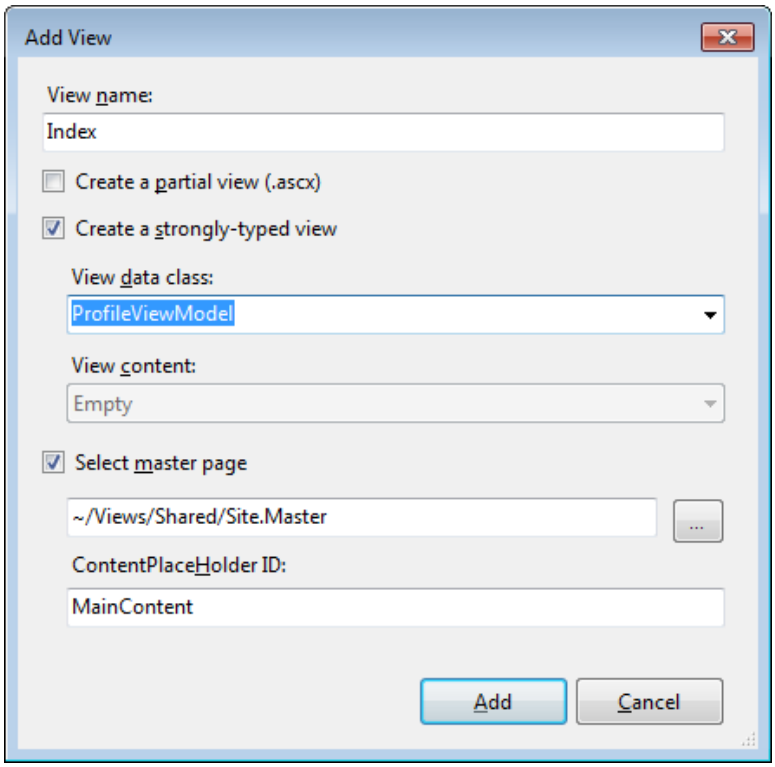
```
            return View("Index", model);
        }

        [Authorize()]
        [AcceptVerbs(HttpVerbs.Post)]
        [ValidateAntiForgeryToken()]
        public ActionResult Update(UserProfile profile)
        {
            var returnUrl = Request.Form["returnUrl"];
            if (!ModelState.IsValid)
            {
                // ошибка валидации
                return this.IsAjaxCall() ?
                new JsonResult { JsonRequestBehavior =
                JsonRequestBehavior.AllowGet, Data = ModelState }
                                    : this.Index(returnUrl);
            }
            this.membershipService.UpdateProfile(profile);
            if (this.IsAjaxCall())
            {
                return new JsonResult {
                       JsonRequestBehavior = JsonRequestBehavior.AllowGet,
                       Data = new {
                       Update = true,
                       Profile = profile,
                       ReturnUrl = returnUrl } };
            }
            else
            {
                return RedirectToAction("UpdateSuccess",
                        "Account", new { returnUrl = returnUrl });
            }
        }
        public ActionResult UpdateSuccess(string returnUrl)
        {
            var model = new ProfileViewModel
            {
                Profile = this.membershipService.GetCurrentProfile(),
                ReturnUrl = returnUrl
            };
            return View(model);
        }

        private void InjectStatesAndActivityTypes(ProfileViewModel model)
        {
            var profile = model.Profile;
            var types = this.activitiesRepository.RetrieveActivityTypes()
                        .Select(o => new SelectListItem { Text = o.Name,
                              Value = o.Id.ToString(),
                              Selected = (profile != null &&
                              o.Id == profile.PreferredActivityTypeId) })
                              .ToList();
            types.Insert(0, new SelectListItem { Text = "Select...",
                                                 Value = "0" });
            var states = this.referenceRepository.RetrieveStates().Select(
                              o => new SelectListItem {
                              Text = o.Name,
                              Value = o.Abbreviation,
                              Selected = (profile != null &&
                              o.Abbreviation == profile.State) })
                              .ToList();
            states.Insert(0, new SelectListItem { Text = "Any state",
                                                  Value = string.Empty });
            model.PreferredActivityTypes = types;
            model.States = states;
        }
    }
}
```

Создание представления учетной записи

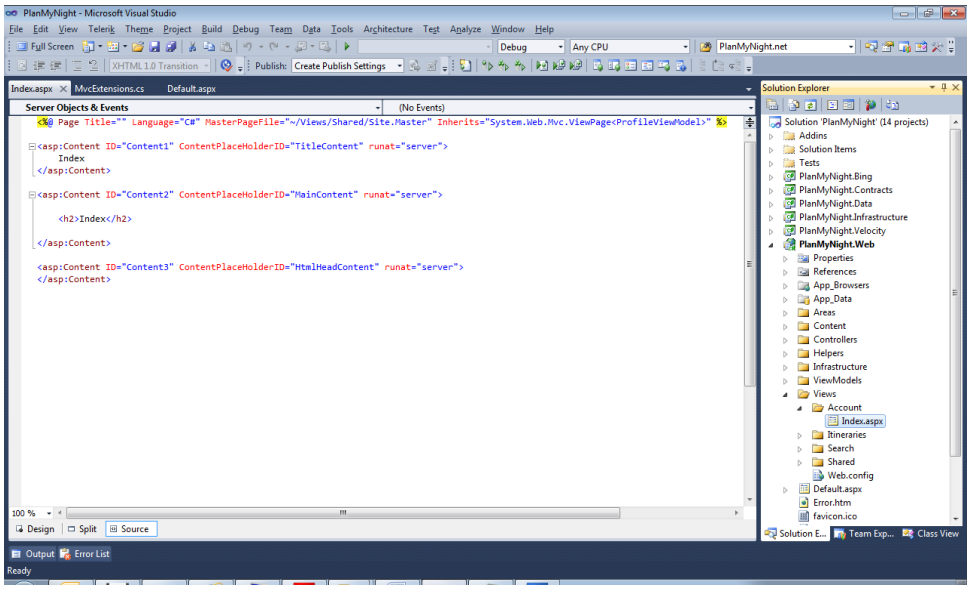
В предыдущем разделе был создан контроллер, реализующий функциональность для обновления и просмотра данных пользователя. В этом разделе мы поэтапно рассмотрим возможности Visual Studio 2010, позволяющие создавать представления, которые будут отображать эту функциональность пользователю.

Для создания представления Index контроллера Account:

76. Перейдем к папке Views проекта PlanMyNight.Web.
77. Щелкнем папку Views правой кнопкой мыши, раскроем подменю Add и выберем New Folder.
78. Назовем новую папку **Account**.
79. Щелкнем правой кнопкой мыши новую папку Account, раскроем подменю Add и выберем View.
80. Заполним поля диалогового окна Add View (Добавить представление), как показано на рисунке:



81. Щелкнем Add. Мы должны получить HTML-страницу с несколькими элементами управления *<asp:Content>*:



Можно заметить, здесь нет особых отличий от того, что мы привыкли видеть в Visual Studio 2005. По умолчанию ASP.NET MVC 2 использует механизм формирования представлений Веб-форм ASP.NET, поэтому страницы MVC и Веб-форм будут иметь некоторое сходство. Основное отличие на этом этапе в том, что класс *page* наследуется от *System.Web.Mvc.ViewPage<ProfileViewModel>*, и отсутствует файл

выделенного кода. MVC не использует файлы выделенного кода, в отличие от Веб-форм ASP.NET, чтобы обеспечить четкое разделение функциональных областей. Редактирование страниц MVC, как правило, осуществляется через разметку. Конструктор применяется преимущественно для приложений Веб-форм ASP.NET.

Чтобы этот каркас страницы стал основным представлением контроллера Account, необходимо изменить содержимое заголовка, обеспечив его единообразие с остальными представлениями:

```
<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Plan My Night – Profile
</asp:Content>
```

Далее требуется добавить клиентские сценарии, которые будут использоваться в содержимом *HtmlHeadContent* (Содержимое HTML-заголовка):

```
<asp:Content ID="Content3" ContentPlaceHolderID="HtmlHeadContent" runat="server">
    <% Ajax.RegisterClientScriptInclude(
        Url.Content("~/Content/Scripts/jquery-1.3.2.min.js"),
        "http://ajax.Microsoft.com/ajax/jQuery/jquery-1.3.2.min.js"); %>
    <% Ajax.RegisterClientScriptInclude(
        Url.Content("~/Content/Scripts/jquery.validate.js"),
"http://ajax.microsoft.com/ajax/jquery.validate/1.5.5/jquery.validate.min.js"); %>
    <% Ajax.RegisterCombinedScriptInclude(
        Url.Content("~/Content/Scripts/MicrosoftMvcJQueryValidation.js"), "pmn");
%>
    <% Ajax.RegisterCombinedScriptInclude(
        Url.Content("~/Content/Scripts/ajax.common.js"), "pmn"); %>
    <% Ajax.RegisterCombinedScriptInclude(
        Url.Content("~/Content/Scripts/ajax.profile.js"), "pmn"); %>
    <%= Ajax.RenderClientScripts() %>
</asp:Content>
```

Этот сценарий использует методы расширения *System.Web.Mvc.AjaxHelper*, описанные в проекте PlanMyNight.Infrastructure в папке MVC.

Когда содержимое заголовка настроено, можно заняться основным содержимым представления:

```
<asp:Content ContentPlaceHolderID="MainContent" runat="server">
<div class="panel" id="profileForm">
    <div class="innerPanel">
        <h2><span>My Profile</span></h2>
        <% Html.EnableClientValidation(); %>
        <% using (Html.BeginForm("Update", "Account")) %>
        <% { %>
        <%=Html.AntiForgeryToken()%>
        <div class="items">
            <fieldset>
                <p>
                    <label for="FullName">Name:</label>
                    <%=Html.EditorFor(m => m.Profile.FullName)%>
                    <%=Html.ValidationMessage("Profile.FullName",
                        new { @class = "field-validation-error-wrapper" })%>
                </p>
                <p>
                    <label for="State">State:</label>
                    <%=Html.DropDownListFor(m => m.Profile.State,
                                            Model.States)%>
                </p>
                <p>
                    <label for="City">City:</label>
                    <%=Html.EditorFor(m => m.Profile.City,
                                        Model.Profile.City)%>
                </p>
                <p>
                    <label for="PreferredActivityTypeId">Preferred
activity:</label>
                    <%=Html.DropDownListFor(m =>
                        m.Profile.PreferredActivityTypeId,
                        Model.PreferredActivityTypes)%>
                </p>
            </fieldset>
            <div class="submit">
```

```
                <%=Html.Hidden("returnUrl", Model.ReturnUrl)%>
                <%=Html.SubmitButton("submit", "Update")%>
            </div>
        </div>
        <div class="toolbox"></div>
        <% } %>
        </div>
</div>
</asp:Content>
```

Если не обращать внимания на некоторый встроенный код, все это выглядит как обычная HTML-разметка. Рассмотрим фрагменты встроенного кода и продемонстрируем, какую мощь они обеспечивают (и простоту при этом).

> **Visual Studio 2005** В Visual Studio 2005 более типичным для отображения данных было использовать серверные элементы управления и логику времени отображения. Но поскольку страницы представлений ASP.NET MVC не имеют файла выделенного кода, эта логика должна быть описана в одном файле с разметкой. По-прежнему могут применяться элементы управления ASP.NET Web Forms. В нашем примере используется элемент управления *<asp:Content>*. Однако, как правило, функциональность элементов управления ASP.NET Web Forms ограничена из-за отсутствия файла выделенного кода

В MVC широко применяются вспомогательные классы HTML. Методы, содержащиеся в *System.Web.Mvc.HtmlHelper*, генерируют компактные соответствующие стандартам HTML-теги для использования в различных целях. Для этого MVC-разработчику в некоторых случаях приходится писать больше кода разметки, чем разработчику Веб-форм, но он получает непосредственный контроль над формированием вывода. Строго типизированная версия этого класса расширения (*HtmlHelper<TModel>*) может использоваться в разметке представления через свойство *ViewPage<TModel>.Html*.

В данной форме используются такие *HTML*-методы (и это лишь часть того, что доступно по умолчанию):

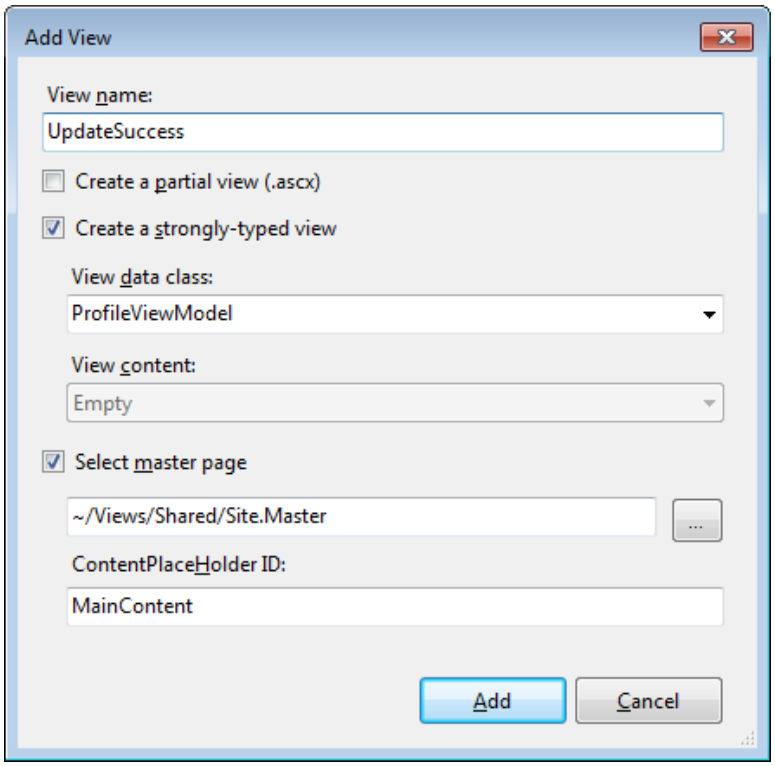
- *Html.EnableClientValidation* (Включить проверку на стороне клиента) обеспечивает проверку данных на стороне клиента на основании строго типизированного словаря ModelState
- *Html.BeginForm* (Начать форму) помещает в разметку тег *<form>* и закрывает форму в конце раздела директив *using*. Принимает различные параметры, но наиболее часто используемыми являются имя действия и контроллер, для которого это действие должно быть вызвано. Благодаря этому инфраструктура MVC может автоматически формировать URL конкретной формы во время выполнения, что избавляет от необходимости вводить строку URL в разметку.
- *Html.AntiForgeryToken* размещает в форме скрытое поле с проверочным значением, которое также сохраняется на стороне сервера и проверяется, если цель формы имеет атрибут *ValidateAntiForgeryToken*. Мы добавили этот атрибут в методе *Update* контроллера.
- *Html.EditorFor* (Редактор для) – перегруженный метод, обеспечивающий вставку текстового поля в разметку. Это строго типизированная версия метода *Html.Editor*.
- *Html.DropDownListFor* (Раскрывающийся список для) – перегруженный метод, обеспечивающий вставку раскрывающегося списка в разметку. Это строго типизированная версия метода *Html.DropDownList*.
- *Html.ValidationMessage* (Сообщение проверки) – вспомогательный метод, который обеспечит вывод на экран сообщения об ошибке проверки, если заданный ключ уже имеется в словаре ModelState.
- *Html.Hidden* (Скрытый) помещает в форму скрытое поле с переданными именем и значением.
- *Html.SubmitButton* (Кнопка Передать) создает в форме кнопку Submit (Передать).

> **Примечание** Когда разметка представления Index готова, остается лишь добавить представления для действия *UpdateSuccess*, и можно будет просматривать результаты.

Для создания представление UpdateSuccess:

82. Раскроем проект PlanMyNight.Web в Solution Explorer и затем раскроем папку Views.
83. Щелкнем правой кнопкой мыши папку Account.
84. Откроем подменю Add и щелкнем View.

85. Заполним поля диалогового окна Add View, чтобы оно выглядело следующим образом:



После создания страницы представления заполним содержимое заголовка следующим образом:

```
<asp:Content ContentPlaceHolderID="TitleContent" runat="server">Plan My Night -
Profile Updated</asp:Content>
```

И содержимое *MainContent* должно выглядеть так:

```
<asp:Content ContentPlaceHolderID="MainContent" runat="server">
<div class="panel" id="profileForm">
    <div class="innerPanel">
        <h2><span>My Profile</span></h2>
        <div class="items">
            <p>Your profile has been successfully updated.</p>
            <h3>» <a href="<%=Html.AttributeEncode(Model.ReturnUrl ??
                          Url.Content("~/"))%>">Continue</a></h3>
        </div>
        <div class="toolbox">
        </div>
    </div>
</div>
</asp:Content>
```

Чтобы увидеть созданные представления, необходимо отредактировать файл Site.Master (располагающийся в папке Views/Shared в корневом каталоге Веб-проекта). Раскомментируем строку 33, удалив теги комментариев. Код должен выглядеть следующим образом:

```
<%=Html.ActionLink<AccountController>(c => c.Index(null), "My Profile")%>
```

Теперь, когда это последнее представление создано, можно откомпилировать и выполнить приложение. Щелкнем кнопку Sign In (Вход), которая показана в верхнем правом углу формы на рис. 6-6, и выполним вход с помощью Windows Live ID.

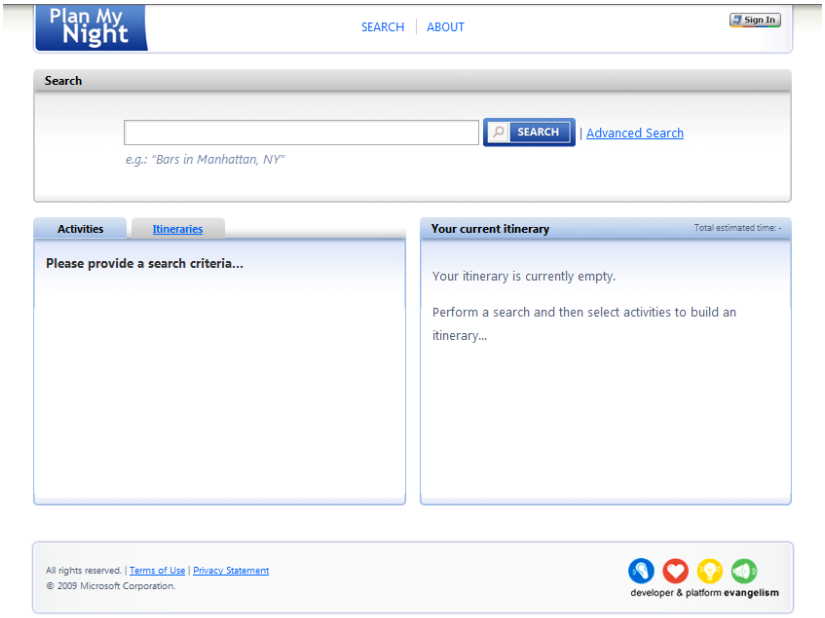


Рис. 6-6 Окно по умолчанию приложения Plan My Night

После того, как вход выполнен, вы должны быть перенаправлены к созданному представлению Index контроллера Account (рис. 6-7).

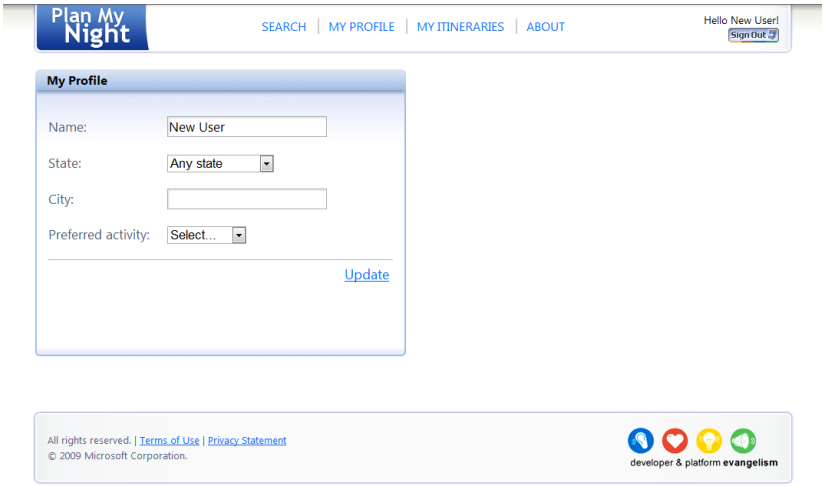


Рис. 6-7 Окно настроек профиля, возвращенное методом Index контроллера Account

Если вместо этого вы оказались на странице поиска, просто щелкните ссылку My Profile (Мой профиль), которая находится в центре вверху интерфейса рядом с остальными ссылками. Чтобы увидеть новые возможности проверки данных в действии, попробуйте сохранить форму, не заполнив поле Full Name (Полное имя). На экране должно появиться следующее сообщение (рис. 6-8).

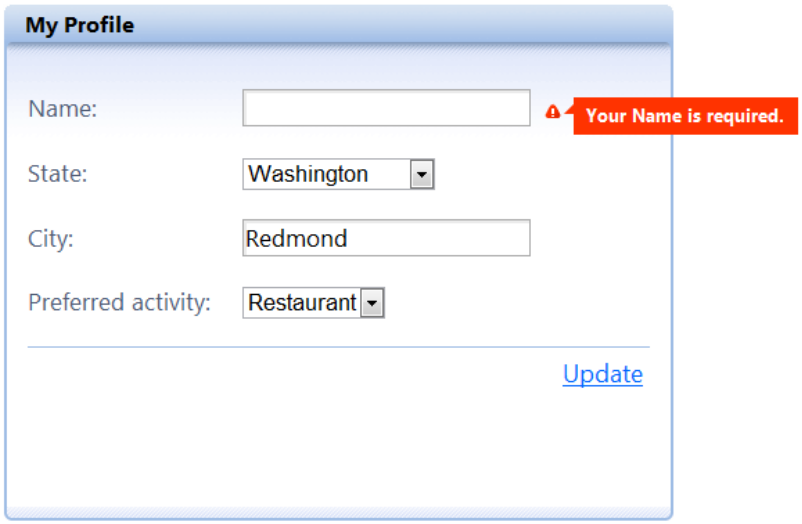


Рис. 6-8 Пример сбоя при проведении проверки Model Binding

Поскольку в приложении включена проверка на стороне клиента, обратного вызова не было. Чтобы увидеть, как работает проверка на стороне сервера, потребовалось бы внести изменения в файл Index.aspx в папке Account, закомментировав вызов *Html.EnableClientValidation*. Благодаря тесной интеграции и поддержке AJAX и JavaScript в приложениях MVC намного упрощается перенос на сторону клиента таких серверных операций, как проверка, по сравнению с тем, как это было ранее.

> **Visual Studio 2005** В приложениях ASP.NET MVC значение атрибута ID отдельного HTML-элемента не преобразовывается, как это происходит в ASP.NET Web Forms 2.0. В Visual Studio 2005 разработчик должен был сохранять *UniqueID* элемента управления/элемента в переменной JavaScript, чтобы обеспечить возможность доступа к нему внешнего JavaScript. Это делалось, чтобы гарантировать уникальность ID, но всегда вводило дополнительный уровень сложности во взаимодействие между элементами управления ASP.NET 2.0 Web Forms и JavaScript. В MVC такого преобразования нет, но обеспечение уникальности ID – сфера ответственности разработчиков. Также следует отметить, что теперь ASP.NET 4.0 Web Forms поддерживает отключение преобразования ID на уровне элемента управления по желанию разработчика

Контроллер Account и связанные представления и являются той недостающей «базовой» функциональностью Plan My Night. В ходе работы над ними мы рассмотрели некоторые новые возможности приложений Visual Studio 2010 и MVC 2.0. Но MVC не единственный доступный выбор для Веб-разработчиков. Веб-формы ASP.NET являются основным типом приложений ASP.NET с момента ее создания, и были улучшены в Visual Studio 2010. В следующем разделе Веб-форма ASP.NET для приложения MVC будет создана в визуальном дизайнере.

Использование дизайнера для создания Веб-формы

Все приложения рано или поздно сталкиваются с непредвиденными условиями, и наше приложение-пример не исключение. При возникновении непредвиденной ситуации оно возвращает сообщение об ошибке, такое как показано на рис. 6-9.

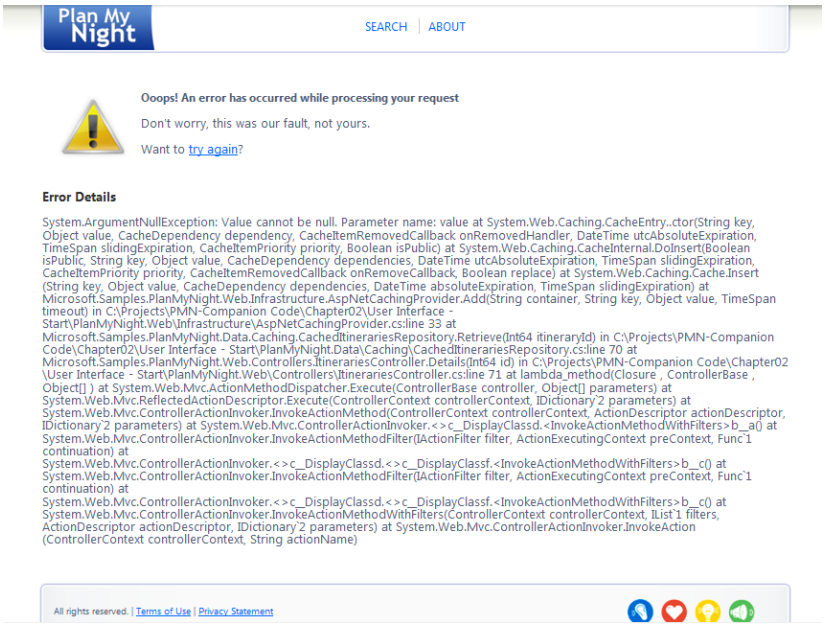
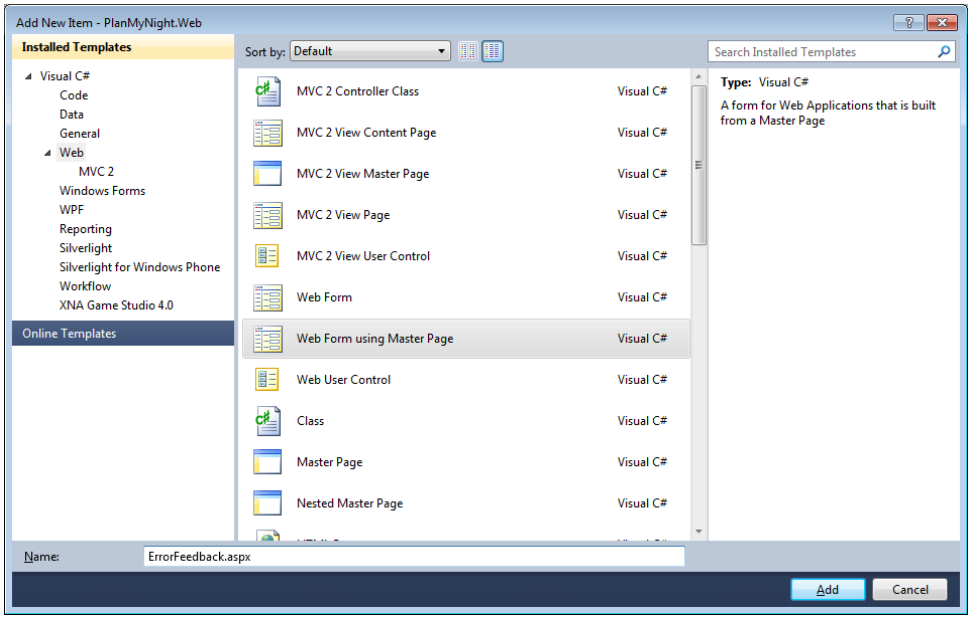


Рис. 6-9 Пример сообщения об ошибке приложения Plan My Night

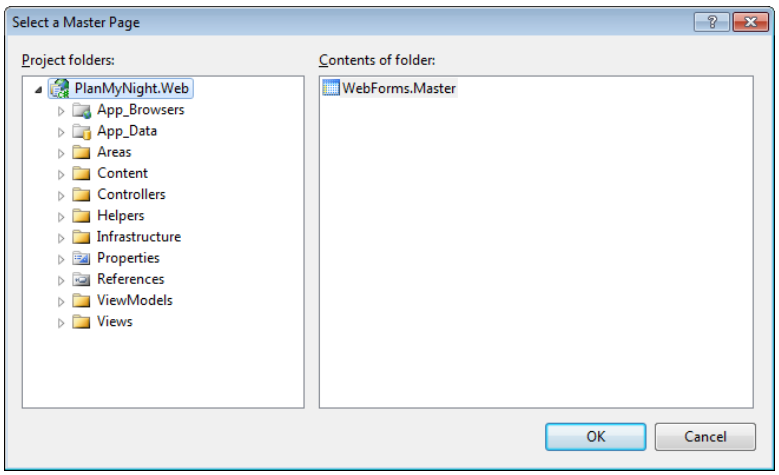
В настоящее время у пользователя, получившего такое сообщение, небогатый выбор: либо попытаться повторить это действие, либо использовать навигационные ссылки в верхней части окна приложения. (Конечно, это может привести к еще одной ошибке.) Предоставление пользователю возможности обратной связи позволит разработчикам получать сведения о ситуации, которых могут не обеспечивать стандартное сообщение об ошибке и трассировка стека. Чтобы рассмотреть другой способ создания компонента пользовательского интерфейса для приложения Plan My Night, спроектируем страницу обратной связи в случае возникновения ошибки как Веб-форму ASP.NET, преимущественно используя для этого дизайнер Visual Studio. Прежде чем приступать к дизайну формы, необходимо создать файл формы.

Для создания новой Веб-формы:

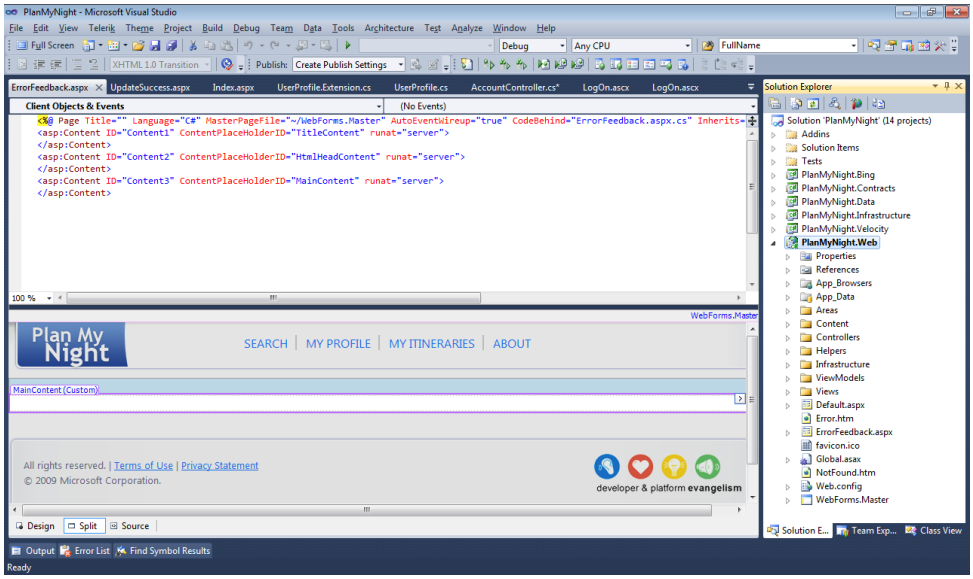
86. Откроем контекстное меню проекта PlanMyNight.Web (щелкнув его правой кнопкой мыши), откроем подменю Add и выберем New Item.
87. В диалоговом окне Add New Item выберем Web Form using Master Page (Веб-форма, использующая главную страницу) и в поле Name введем имя элемента: **ErrorFeedback.aspx**.



88. Появится диалоговое окно, которое позволяет ассоциировать главную страницу с этой Веб-формой. Убедитесь, что в части Project Folders (Папки проекта) окна выбрана основная папка PlanMyNight.Web, и затем выберите элемент WebForms.Master в правой части окна.



89. Результирующая страница может быть представлена не в режиме разделенного представления (Split view), а в режиме просмотра исходного кода (или в режиме дизайнера (Design view)). Перейдите к разделенному представлению (соответствующая кнопка располагается внизу окна, как и в предыдущих версиях Visual Studio). После этого экран должен выглядеть следующим образом:



> **Примечание** Рекомендуется использовать разделенное представление, поскольку оно позволяет видеть исходный код, формируемый дизайнером, и добавлять разметку в случае необходимости.

Раскройте панель элементов управления и оставьте ее на экране, поскольку в ходе работы над формой будете постоянно брать из нее элементы управления и элементы и переносить в область содержимого. Если эта панель еще не вынесена на экран, ее можно найти в меню View (Вид).

Начнем с переноса методом drag-and-drop элемента div (из группы элементов HTML) из панели элементов управления в раздел MainContent (Основное содержимое) дизайнера. При этом появится вкладка div, свидетельствующая о том, что добавленный новый элемент выбран в настоящий момент. Откроем контекстное меню div и выберем Properties (Свойства) (которые также можно открыть, нажав клавишу F4). В раскрывшемся окне Properties задаем свойству *(Id)* значение *profileForm* (Форма профиля) (регистр имеет значение). Также изменим значение свойства *Class* (Класс) на *panel* (панель). После редактирования этих значений размер области содержимого изменится, потому что к представлению дизайнера применены CSS.

> **Visual Studio 2005** Таким важным и необходимым усовершенствованием дизайнера Веб-форм, по сравнению с Visual Studio 2005, стало применение CSS. Благодаря этому разработчик получает возможность в реальном масштабе времени видеть эффекты от вносимых в стили изменений без необходимоысти выполнения приложения. В Visual Studio 2005 дизайнер для страницы search.aspx выглядит, как показано на рис. 6-10.

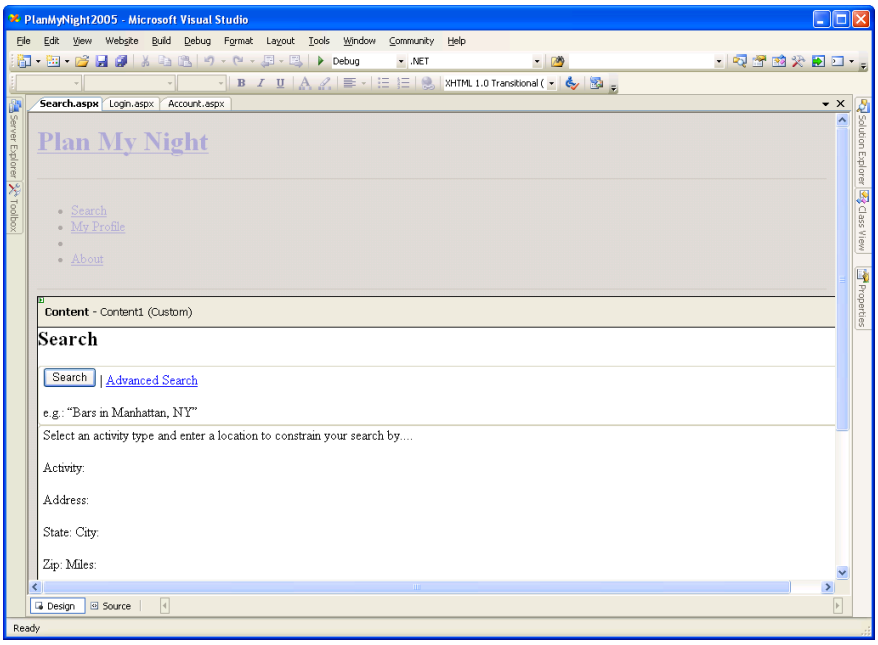


Рис. 6-10 Дизайнерское представление Веб-страницы ASP.NET в Visual Studio 2005

Поместим другой div внутрь первого и зададим его свойству *class* значение *innerPanel* (Внутренняя панель). На панели разметки добавим в *innerPanel* следующую разметку:

```
<h2><span>Error Feedback</span></h2>
```

Закрыв тег <h2>, добавим новую строку и откроем контекстное меню. Выберем Insert Snippet (Вставить фрагмент) и последовательно щелкнем ASP.NET > formr. Это обеспечит создание тега серверной формы, которая будет служить контейнером для Веб-элементов управления. Между тегами формы поместим тег div, атрибуту class которого присвоено значение *items* (элементы), и затем в тег тег div вставим тег fieldset (набор полей).

Далее перетягиваем в тег fieldset элемент управления TextBox (находится в разделе Standard (Стандартные) панели элементов управления). Задаем ID этого текстового поля значение **FullName**. В представлении разметки перед этим элементом управления добавляем тег <label> (метка), в качестве значения его свойства *for* задаем значение ID текстового поля, т.е. **Full Name:** (двоеточие обязательно). Значение тега <label> - это текст, размещаемый между тегами <label> и </label>. Поместите эти два элемента в тег <p>, и представление дизайнера должно выглядеть, как показано на рис. 6-11.

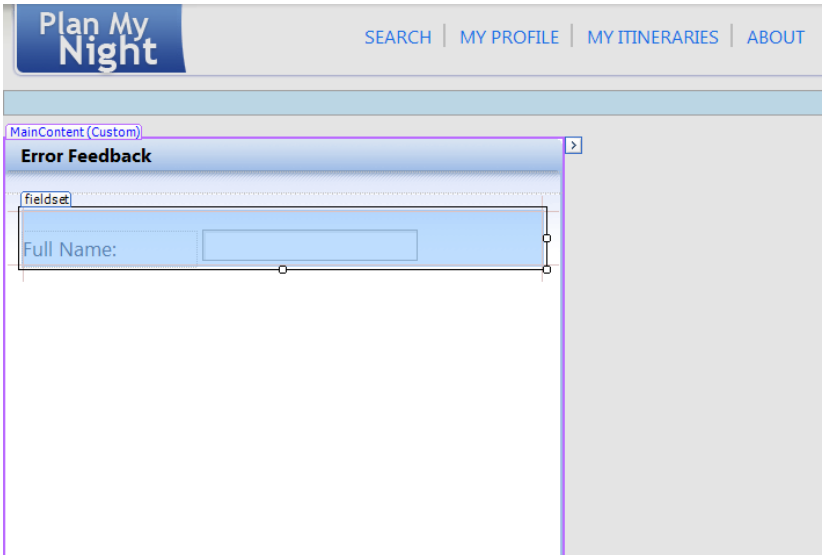


Рис. 6-11 Так на данный момен выглядит ErrorFeedback.aspx в дизайнере

Вышеописанным способом добавим еще одно текстовое поле и метку, но в качестве значения ID текстового поля зададим **EmailAddress** (Адрес электронной почты) и в качестве значения метки – **Email Address:** (не забывая о двоеточии). Повторим весь этот процесс третий раз и зададим TextBox ID и метку **Comments** (Комментарии). Теперь в дизайнере должно находиться три метки и три однострочных элемента управления TextBox. Для элемента управления Comments необходимо обеспечить многострочный ввод,

поэтому открываем его свойства и задаем *TextMode* (Текстовый режим) значение *Multiline* (Многострочный), *Rows* (Строки) значение *5* и *Columns* (Столбцы) значение *40*. После этого текстовое поле станет намного шире, что позволит пользователю вводить комментарии.

Снова воспользуемся возможностью Insert Snippet и после текстового поля Comments вставим тег «div with class» (HTML>divc). В качестве класса тега div зададим *submit*. Из панели инструментов перетащим в этот div элемент управления Button (Кнопка). Зададим свойству *Text* (Текст) кнопки значение *Send Feedback* (Отправить отзыв).

В дизайнере должно отображаться нечто похожее представленному на рис. 6-12. На данный момент мы получили страницу, которая будет передавать форму.

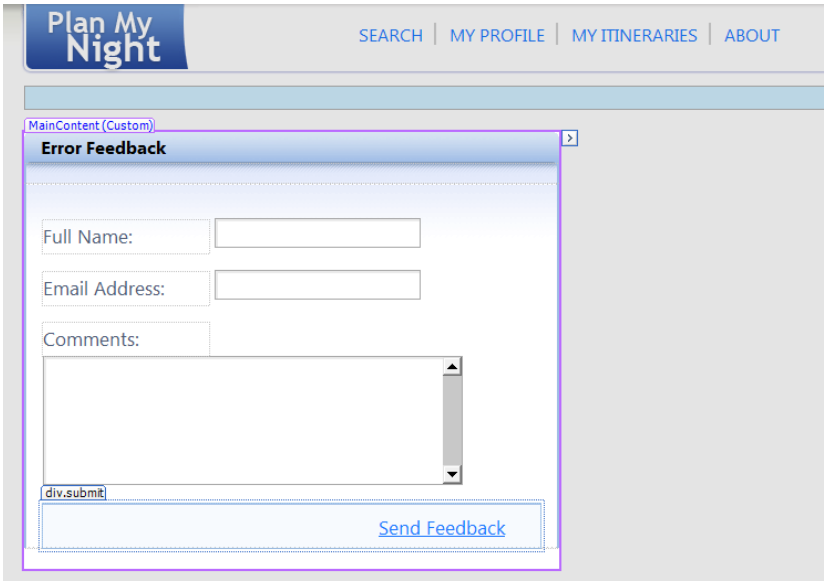


Рис. 6-12 Форма ErrorFeedback.aspx с полным набором полей

Но эта форма не выполняет никакой проверки передаваемых ею данных. Для реализации такой функциональности воспользуемся некоторыми элементами управления для проверки ввода пользователя, предлагаемыми ASP.NET. Сделаем поля Full Name и Comments обязательными для заполнения и осуществим проверку адреса электронной почты при помощи регулярных выражений, чтобы гарантировать его соответствие заданному шаблону.

В группе Validation (Проверка) панели инструментов имеются некоторые готовые элементы управления для проверки ввода пользователя. Возьмите на панели инструментов объект *RequiredFieldValidator* (Средство проверки обязательного поля) и поместите его справа от текстового поля Full Name. Откройте свойства этого элемента управления для проверки и задайте свойству *ControlToValidate* (Проверяемый элемент управления) значение *FullName*. (Для удобства предлагается раскрывающийся список элементов управления страницы.) Также задайте свойству *CssClass* (Класс CSS) значение *field-validation-error* (ошибка проверки поля). Это обусловит то, что все ошибки в приложении будут обозначаться красным треугольником. Наконец, свойству Error Message (Сообщение об ошибке) задаем значение **Name is Required** (Необходимо указать имя) (рис. 6-13).

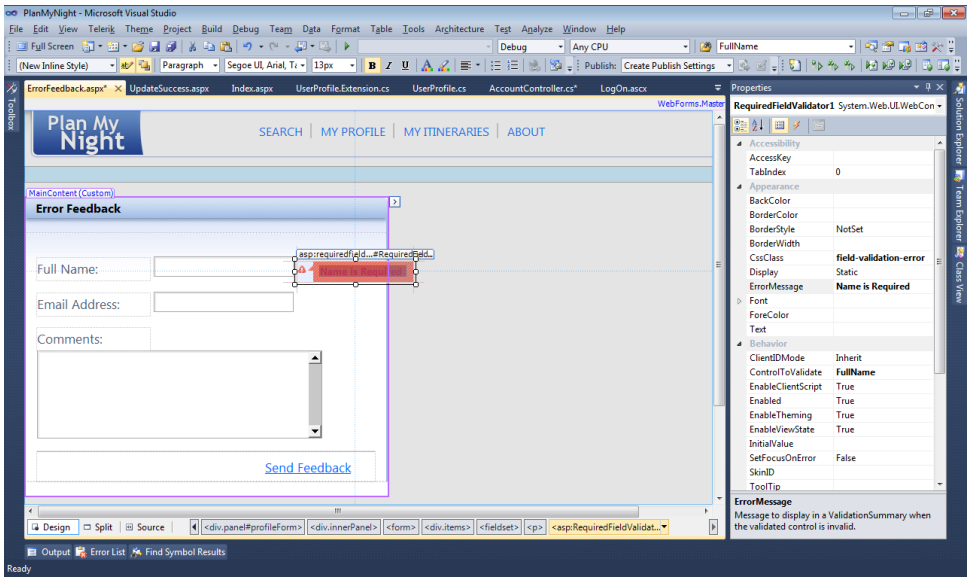


Рис. 6-13 Пример элемента управления для проверки ввода пользователя

Эти же шаги повторим для поля Comments, но зададим соответствующие значения свойствам *ErrorMessage* (Сообщение об ошибке) и *ControlToValidate*.

Пользователь должен гарантированно вводить действительный адрес электронной почты в поле Email Address, поэтому возьмем на панели инструментов элемент управления *RegularExpressionValidator* и поместим его рядом с этим текстовым полем. Значения свойствам этого элемента управления задаем по используемой ранее схеме, т.е. свойству *ControlToValidate* присваиваем значение *EmailAddress* и свойству *CssClass* – значение *field-validation-error*. Но для этого элемента управления также задается регулярное выражение, которое должно применяться к вводимым данным. Это осуществляется посредством свойства *ValidationExpression* (Выражение проверки), значение которого должно быть определено следующим образом:

```
[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}
```

Сообщение об ошибке этого средства проверки должно быть сформулировано примерно так: «Введите действительный адрес электронной почты».

Форма готова. Чтобы увидеть ее в приложении, необходимо добавить опцию обратной связи при возникновении ошибки для пользователя. В Solution Explorer перейдите по дереву проекта PlanMyNight.Web к папке Views и затем к подпапке Shared. Откройте файл Error.aspx в средстве просмотра разметки и перейдите к строке 35. В этой строке обрабатывается предложение пользователю сделать еще одну попытку выполнить действие, и сюда мы поместим опцию для отправки отзыва. После текста вопроса в тот же абзац добавьте следующую разметку:

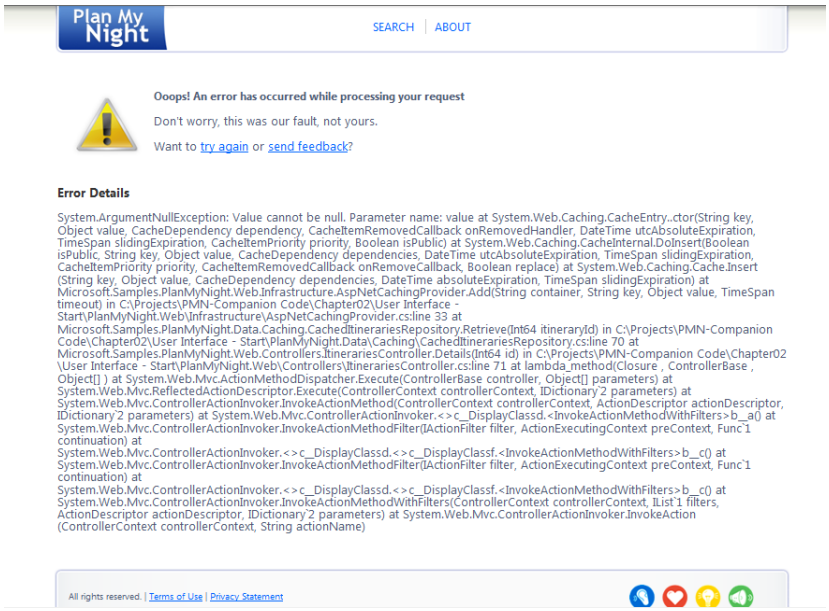
```
or <a href="/ErrorFeedback.aspx">send feedback</a>?
```

Это обеспечит добавление опции перехода к только что созданной нами форме при возникновении в приложении MVC любой ошибки общего характера. Чтобы увидеть эту форму, необходимо спровоцировать ошибку в приложении.

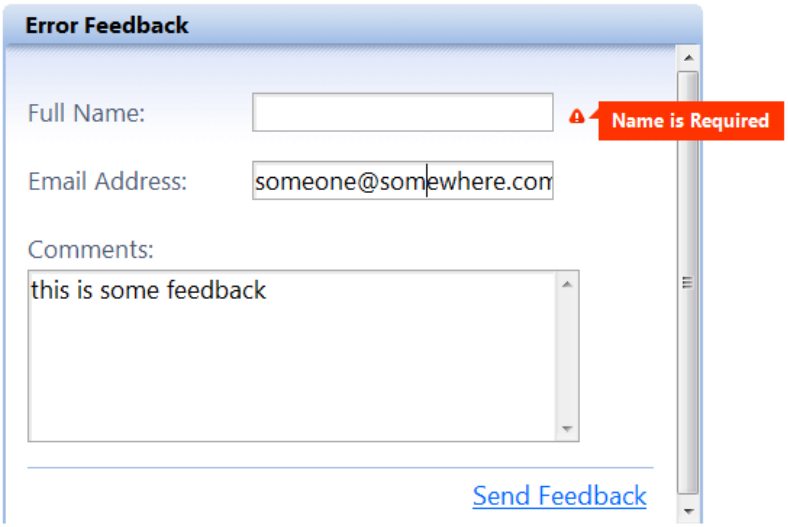
Чтобы спровоцировать возникновение ошибки в приложении Plan My Night:

90. Запустите приложение.
91. Увидев на экране стандартную страницу поиска, введите в адресную строку браузера следующий адрес: **http://www.planmynight.net:48580/Itineraries/Details/38923828**.

92. Очень маловероятно, что в базе данных имеется план мероприятий с таким ID, поэтому на экране появится страница с сообщением об ошибке.



93. На странице с сообщением об ошибке щелкните ссылку для перехода к форме обратной связи. Попробуйте передать форму, введя в нее недействительные данные.



Для осуществления проверки ASP.NET использует сценарий на стороне клиента (если браузер поддерживает это), поэтому пока данные вводятся, никаких обратных вызовов не выполняется. Когда же сервер получает введенные данные, разработчик может проконтролировать состояние проверки на стороне сервера с помощью свойства *Page.IsValid* в файле выделенного кода. Однако поскольку использовалась проверка на стороне клиента (по умолчанию), значение этого свойства всегда будет *true*. В файл выделенного кода необходимо добавить лишь код, обеспечивающий перенаправление пользователя при обратном вызове (и проверку свойства *Page.IsValid* в случае, если что-то было упущено при проверке на стороне клиента):

```
protected void Page_Load(object sender, EventArgs e)
    {
        if (this.IsPostBack && this.IsValid)
        {
            this.Response.Redirect("/", true);
        }
    }
```

В этом нет особого смысла для пользователя, но нашей целью в данном разделе было создание Веб-формы ASP.NET с помощью дизайнера. Поэтому в проекте PlanMyNight.Web появился новый интерфейс. Но что, если бы потребовалось обеспечить большую модульность функциональности, скажем, реализовать некую функцию, которую можно было бы добавлять или удалять без компиляции основного проекта приложения. Вот здесь-то и может продемонстрировать свои преимущества такая инфраструктура расширения как Managed Extensibility Framework (MEF).

Расширение приложения с помощью MEF

Visual Studio 2010 предлагает новую технологию в составе .NET Framework 4 – Managed Extensibility Framework (MEF). Managed Extensibility Framework обеспечивает разработчикам простой, но при этом мощный механизм, с помощью которого сторонние производители получают возможность расширять приложения уже после их поставки. Благодаря MEF даже в рамках одного приложения разработчики могут создавать полностью изолированные компоненты, что обеспечивает возможность их независимого обслуживания или замены. Для этого MEF использует контейнер разрешений, который позволяет сопоставлять компоненты, обеспечивающие конкретную функцию (экспортеры), и компоненты, нуждающиеся в этой функциональности (импортеры), и при этом двум конкретным компонентам даже не надо ничего знать друг о друге. Разрешения выполняются только на контрактной основе, что делает компоненты взаимозаменяемыми или позволяет вводить их в приложения с очень небольшими издержками.

> *Дополнительные сведения Веб-сайт сообщества разработчиков MEF, на котором представлена детальная информация об этой архитектуре, можно найти по адресу* http://mef.codeplex.com.

Приложение Plan My Night создавалось с учетом возможности расширения. В папке Addins (Надстройки) его решения имеется три проекта модулей «надстроек» (рис. 6-14).

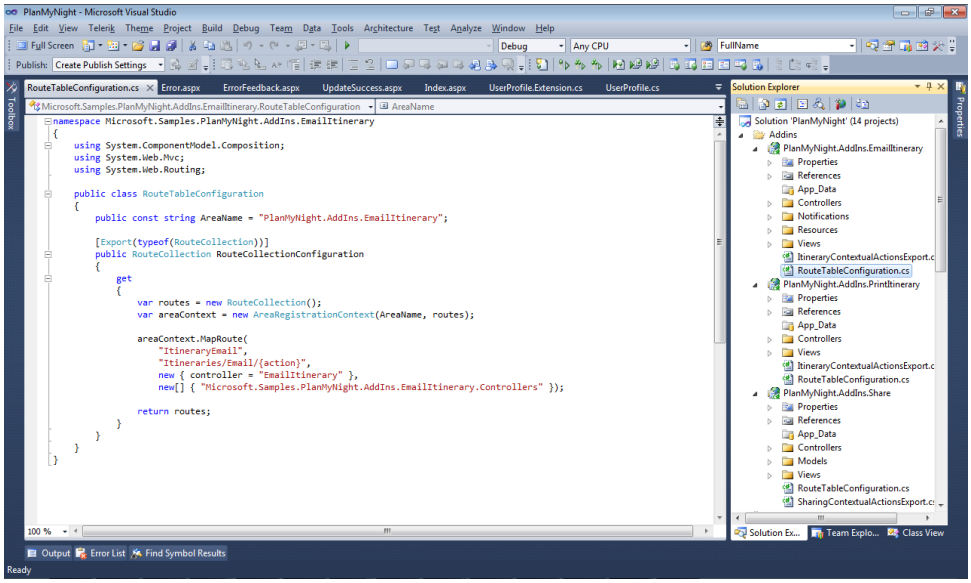


Рис. 6-14 Надстройки приложения Plan My Night

PlanMyNight.Addins.EmailItinerary добавляет возможность отправлять списки маршрутов всем, кому пожелает пользователь. PlanMyNight.Addins.PrintItinerary обеспечивает версию для печати маршрута. Наконец, PlanMyNight.Addins.Share добавляет функции публикации на социальных сайтах (что позволяет пользователю публиковать ссылку на маршрут), а также операции «обрезки» URL. Ни один из этих проектов не ссылается на основное приложение и не упоминается в нем. Да, в них имеются ссылки на проекты PlanMyNight.Contracts и PlanMyNight.Infrastructure, что позволяет экспортировать (и импортировать в некоторых случаях) соответствующие контракты через MEF, а также использовать любые специальные расширения проекта инфраструктуры.

> **Примечание** Если Веб-приложение еще не выполняется, прежде чем переходить к следующему этапу, запустите проект PlanMyNight.Web, чтобы видеть пользовательский интерфейс.

Чтобы добавить модули в выполняющееся приложение, запустим файл DeployAllAddins.bat, располагающийся в одной папке с файлом PlanMyNight.sln. Это обеспечит создание новых папок в разделе Areas проекта PlanMyNight.Web. Новые папки, по одной для каждой надстройки, будут включать файлы, необходимые для добавления их функциональности в основное Веб-приложение. Надстройки появляются в приложении как дополнительные опции на странице результатов поиска и на странице данных маршрута под разделом текущего маршрута. После того как командный файл закончит выполнение, перейдите к интерфейсу PlanMyNight, выполните поиск действия и добавьте его в текущий маршрут. Теперь под панелью маршрута, кроме New (Создать) и Save (Сохранить), должны появиться дополнительные опции (рис. 6-15).

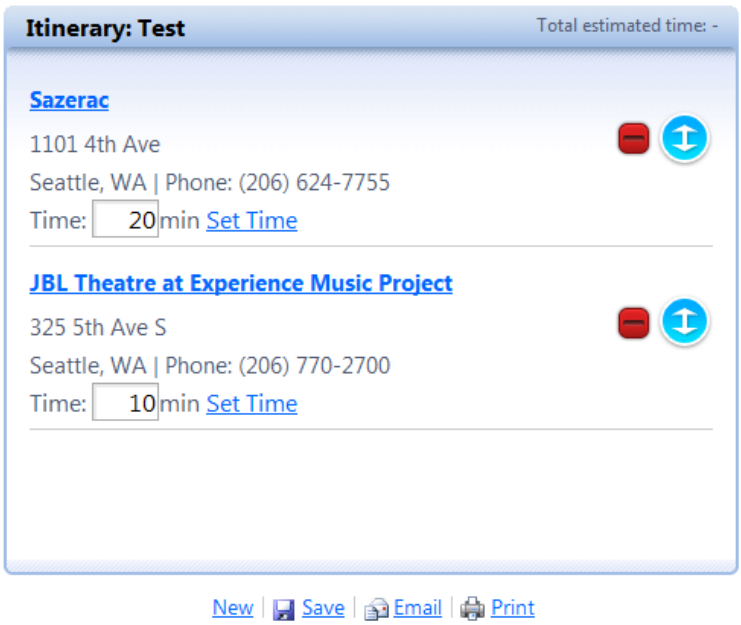


Рис. 6-15 Пользовательский интерфейс с надстройкой для отправки маршрута по электронной почте

Опции публикации на социальных сайтах начнут отображаться в интерфейсе только после того, как маршрут будет сохранен и отмечен как общедоступный (рис. 6-16).

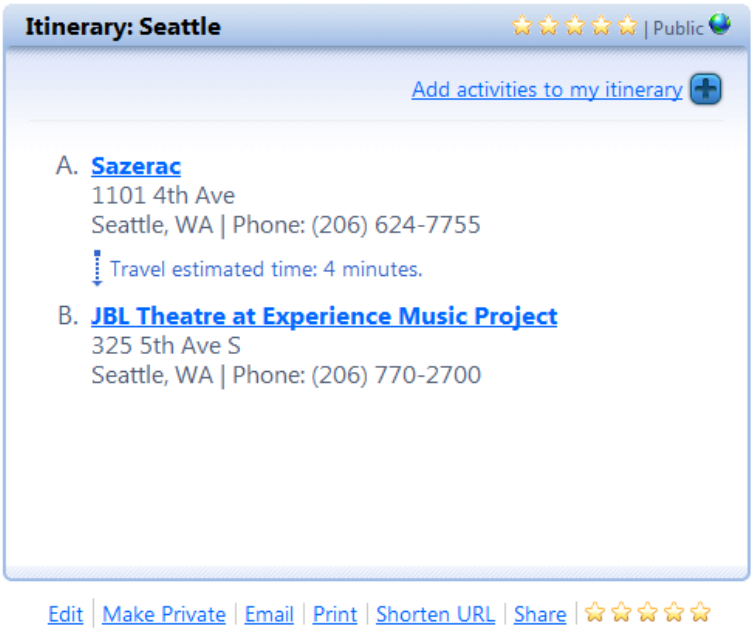


Рис. 6-16 Пользовательский интерфейс с надстройкой для публикации маршрута на социальных сайтах

> **Visual Studio 2005** В Visual Studio 2005 нет ничего похожего на MEF. Для обеспечения поддержки подключаемых модулей разработчику пришлось бы создавать инфраструктуру подключаемого модуля с нуля или приобретать коммерческий продукт. Любой из этих вариантов приводил к созданию собственных решений, в которых внешнему разработчику приходилось разбираться в случае необходимости реализации компонента для них. Добавление MEF в .NET Framework помогает убрать барьеры для перехода к разработке расширяемых приложений и подключаемых модулей для них.

Надстройка для создания печатной версии плана мероприятий

Рассмотрим использование подключаемых модулей в приложении на примере проекта PrintItinerary.Addin. Дерево проекта в развернутом виде должно быть похожим на структуру, представленную на рис. 6-17.

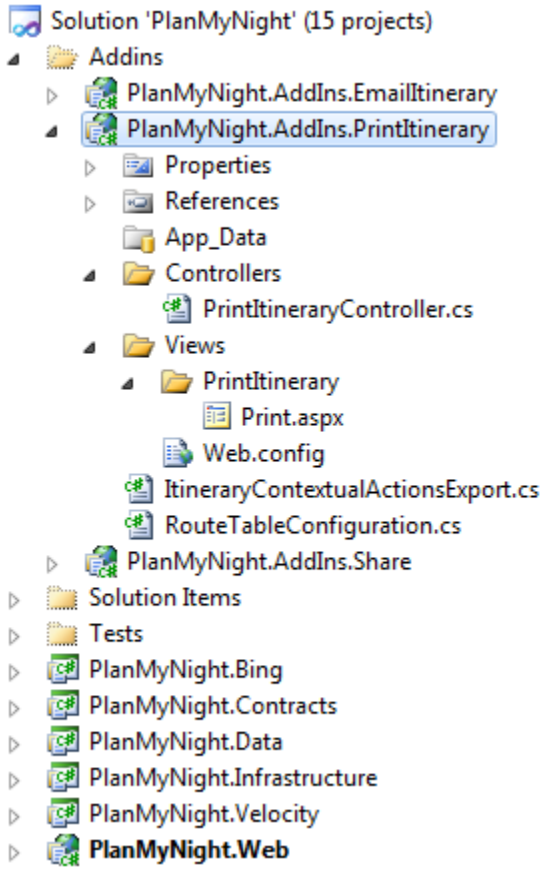


Рис. 6-17 Структура проекта PrintItinerary

Некоторые моменты данной структуры аналогичны проекту PlanMyNight.Web (Controllers и Views), и поэтому эта надстройка будет помещена в приложение MVC как Area. Если более внимательно посмотреть на файл PrintItineraryController.cs в папке Controller, можно заметить, что его структура очень похожа на структуру контроллера, созданного нами ранее в этой главе (и любого другого контроллера Веб-приложения), но при этом имеются и существенные отличия от контроллеров, компилируемых в основном приложении PlanMyNight.Web.

В описании класса присутствуют дополнительные атрибуты:

```
[Export("PrintItinerary", typeof(IController))]
[PartCreationPolicy(CreationPolicy.NonShared)]
```

Эти два атрибута описывают данный тип контейнера разрешений MEF. Первый атрибут, *Export* (Экспорт), помечает этот класс как предоставляющий *IController* с именем контракта *PrintItinerary*. Второй атрибут объявляет, что этот объект поддерживает только индивидуальное создание и не может создаваться как совместно используемый/singleton-объект. Задание этих двух атрибутов – это все что необходимо для создания типа, используемого MEF. На самом деле, *PartCreationPolicy* (Политика создания части) является необязательным атрибутом, но он должен быть определен на случай, если тип не может обрабатывать все типы политики создания.

Далее в файле PrintItineraryController.cs можно заметить, что конструктор снабжен атрибутом *ImportingConstructor* (Конструктор импорта):

```
[ImportingConstructor]
public PrintItineraryController(IServiceFactory serviceFactory) :
this(
    serviceFactory.GetItineraryContainerInstance(),
    serviceFactory.GetItinerariesRepositoryInstance(),
    serviceFactory.GetActivitiesRepositoryInstance())
    {
    }
```

Атрибут *ImportingConstructor* указывает MEF о необходимости предоставления параметров при создании этого объекта. В данном конкретном случае MEF обеспечивает экземпляр *IServiceFactory* для использования этим объектом. Классу *this* не важно, откуда поступает экземпляр, участвующий в создании

модульных приложений. Для наших целей оговоренный контрактом *IServiceFactory* экспортируется в проект PlanMyNight.Web файлом ServiceFactory.cs.

Файл RouteTableConfiguration.cs регистрирует сведения о маршруте URL, которые должны быть направлены в PrintItineraryController. Этот маршрут, и маршруты других надстроек, регистрируются в приложении в ходе выполнения метода *Application_Start*, описанного в файле Global.asax.cs приложения PlanMyNight.Web:

```
// Фабрика EF Controller
var controllerFactory = new MefControllerFactory(container);
ControllerBuilder.Current.SetControllerFactory(controllerFactory);

// Регистрация маршрутов из надстроек
foreach (RouteCollection routes in container.GetExportedValues<RouteCollection>())
{
    foreach (var route in routes)
    {
        RouteTable.Routes.Add(route);
    }
}
```

controllerFactory – это контейнер MEF, включающий путь к подпапке Areas (в которой находятся все надстройки). Он назначен фабрикой контроллера на весь срок жизни приложения. Благодаря этому контроллеры, импортированные через MEF, могут использоваться в любом месте приложения. Маршруты этих надстроек извлекаются из контейнера MEF и регистрируются в таблице маршрутизации MVC.

Файл ItineraryContextualActionsExport.cs экспортирует сведения для создания ссылки на этот подключаемый модуль, а также создания метаданных для отображения. Эти сведения используются в файле ViewModelExtensions.cs проекта PlanMyNight.Web при построении модели представления для отображения пользователю:

```
// получаем ссылки и панели инструментов надстроек
var addinBoxes = new List<RouteValueDictionary>();
var addinLinks = new List<ExtensionLink>();

addinBoxes.AddRange(AddinExtensions.GetActionsFor("ItineraryToolbox", model.Id == 0
? null : new { id = model.Id }));

addinLinks.AddRange(AddinExtensions.GetLinksFor("ItineraryLinks", model.Id == 0 ?
null : new { id = model.Id }));
```

Вызов *AddinExtensions.GetLinksFor* обеспечивает перечисление экспортированных элементов в поставщике экспорта MEF и возвращает их коллекцию, которая может быть добавлена в локальную коллекцию *addinLinks* (Ссылки на надстройки). Впоследствии эта коллекция используется в представлении для отображения большего числа опций, если они имеются.

Заключение

В данной главе мы обсудили лишь несколько возможностей из большого числа новых технологий, предлагаемых Visual Studio 2010. Поэтапно рассмотрели создание контроллера и его связанного представления, а также мощную возможность, предлагаемую Веб-разработчикам инфраструктурой ASP.NET MVC для создания Веб-приложений. Мы познакомились, как с помощью Managed Extensibility Framework можно создавать внешние модули надстроек и подключать их к приложению во время выполнения. В следующей главе будет показано, какие улучшения предлагает Visual Studio 2010 для отладки приложений.

Глава 7

От 2005 к 2010: отладка приложения

В данной главе рассматривается

- Использование новых возможностей отладки Microsoft Visual Studio 2010
- Создание модульных тестов и их выполнение в Visual Studio 2010
- Проводится сравнение возможностей Visual Studio 2005 и 2010

В ходе написания данной книги мы увидели, насколько сильно эволюционировали инструменты отладки и различные вспомогательные средства для разработчиков в трех последних версиях Visual Studio. В Visual Studio 2010 внимание акцентируется на отладке приложения и написании модульных тестов, что лишь расширяет предлагаемые возможности.

Возможности отладки в Visual Studio 2010

В данной главе рассматриваются различные возможности отладки. В качестве примера используется модифицированное приложение Plan My Night. Если сопроводительные материалы данной книги установлены в каталог по умолчанию, интересующее нас приложение находится по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 7\Code. Щелкните двойным щелчком файл PlanMyNight.sln.

Прежде чем переходить непосредственно к процессу отладки, необходимо выполнить некоторую настройку:

94. Проверьте в Solution Explorer, является ли PlanMyNight.Web автозагружаемым проектом. Если его имя не выделено жирным шрифтом, щелкните PlanMyNight.Web правой кнопкой и выберите Set As StartUp Project (Назначить автозагружаемым проектом).
95. Для подготовки к следующим этапам в решении PlanMyNight.Web откройте файл Global.asax.cs, щелкнув треугольник рядом с папкой Global.asax, и затем щелкните двойным щелчком файл Global.asax.cs, как показано на рис. 7-1.

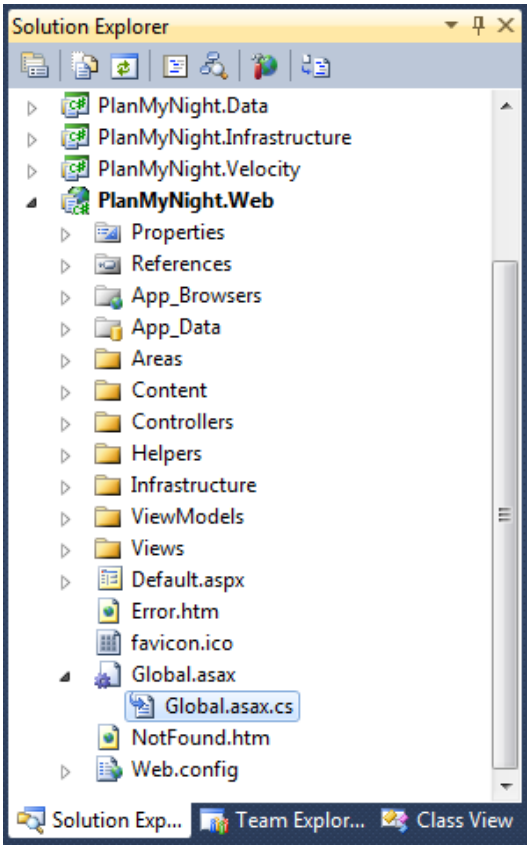


Рис. 7-1 Внешний вид Solution Explorer до открытия файла Global.asax.cs

Управление сеансом отладки

На примере приложения Plan My Night мы увидим, как может разработчик управлять точками останова и организовывать их совместное использование. Новые улучшенные возможности работы с точками останова позволяют значительно повысить скорость и эффективность проверки различных элементов данных в приложении. Также будут рассмотрены новые минидампы и новый интерпретатор промежуточного языка (intermediate language, IL), с помощью которого разработчик может видеть свойства и функции управляемого кода во время отладки минидампа.

Новые улучшенные возможности работы с точками останова

Итак, на данный момент в редакторе открыт файл Global.ascx.cs. Рассмотрим поэтапно некоторые из возможных путей управления точками останова и организации их совместного использования.

96. Перейдем к методу *Application_BeginRequest(object sender, EventArgs e)* и установим точку останова в строке *var url = HttpContext.Current.Request.Url;*, щелкнув поле слева или нажав F9. Рис. 7-2 демонстрирует, как это выглядит в редакторе.

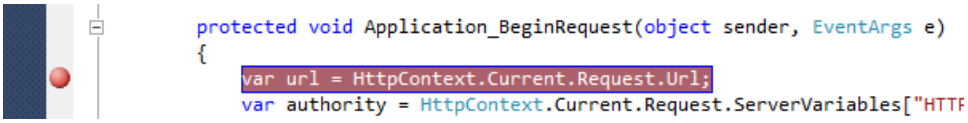


Рис. 7-2 Создание точки останова

97. Нажмем F5, чтобы запустить приложение в режиме отладки. В панели задач должен быть отражен запуск Веб-сервера разработчика, откроется новое окно браузера, и приложение немедленно прекратит выполнение в только что созданной точке останова. Если окно Breakpoints (Точки останова) не появилось на экране даже после запуска приложения в режиме отладки, в меню Debug (Отладка) выберите опцию Windows (Окна) и затем Breakpoints, или используйте комбинацию горячих клавиш Ctrl+D+B.

    Теперь на экран должно быть выведено окно Breakpoints, как показано на рис. 7-3.

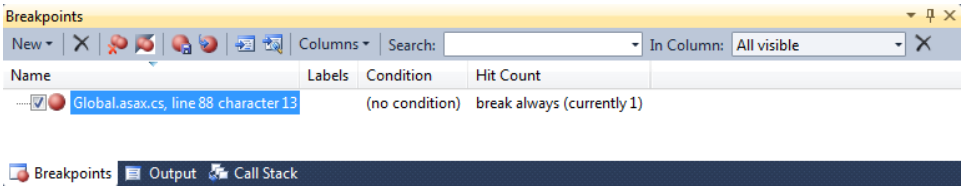


Рис. 7-3 Окно Breakpoints

98. В том же методе добавьте еще три точки останова. Редактор кода и окно Breakpoints должны выглядеть, как показано на рис. 7-4.

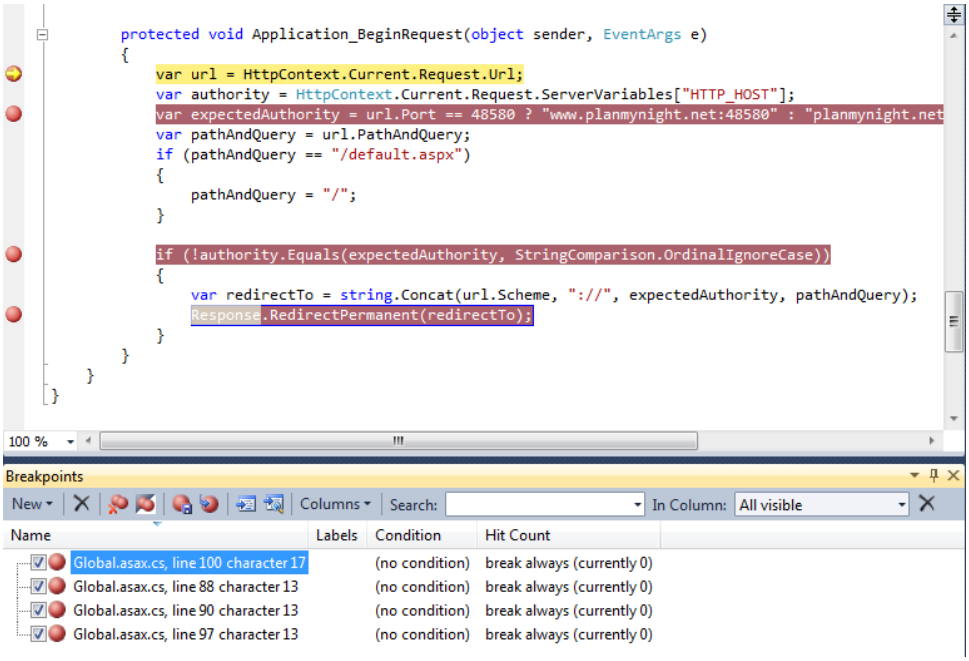


Рис. 7-4 Редактор кода и окно Breakpoints с тремя новыми точками останова

Visual Studio 2005 Внимательный читатель и профессиональный разработчик, часто использовавший Visual Studio 2005, вероятно, заметил в этом упражнении в окне Breakpoints ряд новых кнопок и полей. Напомним, как это окно выглядело в Visual Studio 2005 (рис. 7-5).

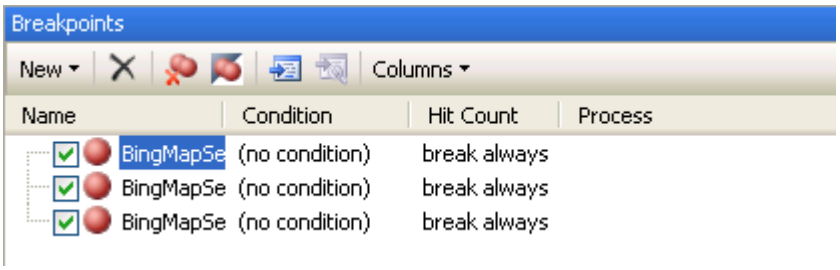


Рис. 7-5 Окно Breakpoints в Visual Studio 2005

99. Обратите внимание, в окне Breakpoints теперь имеется столбец Labels (Метки), что упрощает индексацию и поиск точек останова. Это на самом деле замечательная и очень полезная возможность, появившаяся в Visual Studio 2010. Чтобы воспользоваться ею, достаточно щелкнуть правой кнопкой точку останова в окне Breakpoints и выбрать Edit Labels (Редактировать метки) или использовать сочетание горячих клавиш Alt+F9, L. Это показано на рис. 7-6.

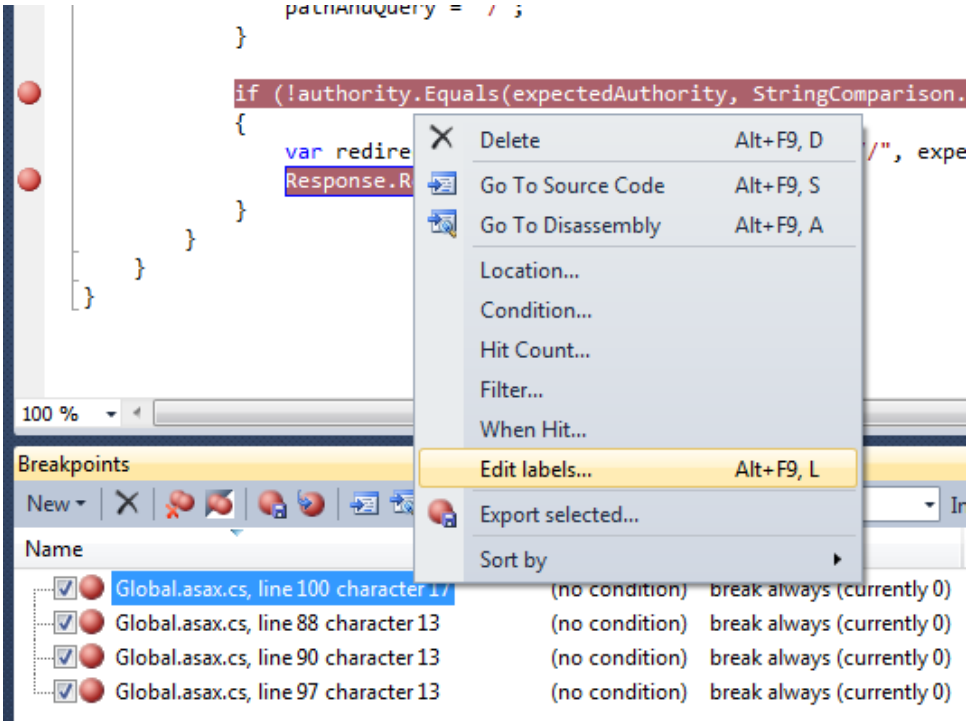


Рис. 7-6 Опция Edit Labels

100. В окне Edit Breakpoint Labels (Редактор меток точек останова) добавьте метки для выбранной точки останова (первой в окне Breakpoints). В текстовом поле Type a New Label (Введите новую метку) введите **ContextRequestUrl** и щелкните Add. Повторите эту операцию для следующей точки останова и введите имя метки **url**. Когда все готово, щелкните OK. На рис. 7-7 представлено, как выглядит окно Edit Breakpoint Labels в момент ввода меток, и окно Breakpoints после выполнения этих двух операций.

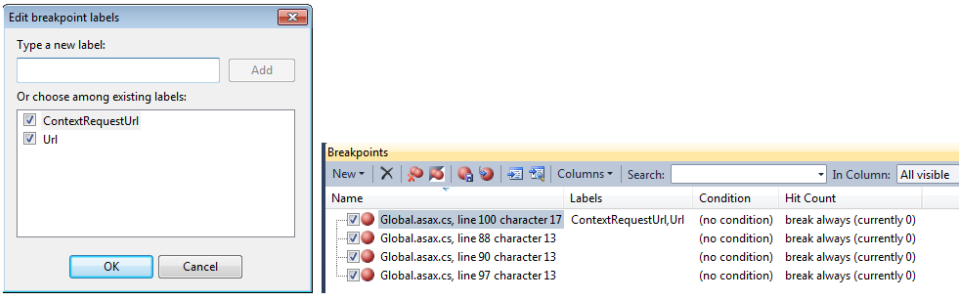


Рис. 7-7 Добавление меток, которые отображаются в окне Breakpoints

> **Примечание** Также вышеописанные задачи можно выполнить, щелкнув правой кнопкой точку останова в поле слева и выбрав в появившемся меню опцию Edit Labels.

> **Примечание** При добавлении меток для новой точки останова есть возможность выбрать любую из существующих меток, которые перечислены в области Or Choose Among Existing Labels (Или выбрать из существующих меток) диалогового окна Edit Breakpoint Labels (рис. 7-7).

101. Любым из рассмотренных способов добавляем метки для каждой точки останова. После этого окно Breakpoints должно выглядеть, как представлено на рис. 7-8.

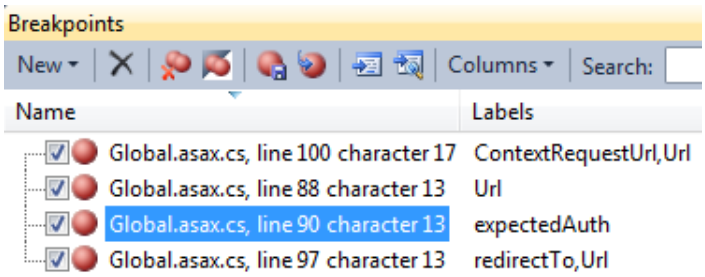


Рис. 7-8 Окно Breakpoints со всеми метками

В ходе отладки больших объемов кода очень пригодится возможность применения фильтра к отображаемому списку точек остановка. Именно это позволяет делать новая функция поиска в Visual Studio 2010.

102. Чтобы увидеть возможность поиска в действии, просто введите **url** в строку поиска и получите список, включающий только точки останова, в метках которых имеется строка *url*.

При групповой разработке, когда в одном проекте занято множество разработчиков и тестировщиков, над одними и теми же ошибками часто работают два человека одновременно. В Visual Studio 2005 этим двум сотрудникам приходилось бы сидеть рядом, обмениваться мгновенными снимками экрана или пересылать друг другу номера строк кода, в которых требуется поставить точки останова, чтобы уточнить, с каким фрагментом кода работать при отладке конкретной ошибки.

> **Важно** Еще одним замечательным дополнением к управлению точками останова в Visual Studio 2010 является появившаяся возможность экспортировать их в файл, который затем можно переслать коллеге. Из этого файла точки останова могут быть импортированы в другую среду. Эта возможность также пригодится, если требуется распространить точки останова на несколько компьютеров. Далее мы рассмотрим, как это делается.

103. В окне Breakpoints щелкните кнопку Export (Экспортировать) для экспорта точек останова в файл и сохраните этот файл на рабочем столе. Назовите файл **breakexports.xml**.

104. Удалите все точки останова, либо щелкнув кнопку Delete All Breakpoints Matching The Current Search Criteria (Удалить все точки останова, соответствующие текущему критерию поиска), либо выбрав все точки останова и нажав кнопку Delete The Selected Breakpoints (Удалить выбранные точки останова). Удаляем точки останова с целью моделировать их совместное использование двумя разработчиками или распространение их на два компьютера.

105. Теперь выполнител импорт точек останова, нажав кнопку Import (Импортировать) и загрузив их с рабочего стола. Обратите внимание, что все точки останова со всеми свойствами опять появились в среде разработки. Для целей данной главы удалите все точки останова.

Проверка данных

Всем известно, как много времени при отладке приложения может уйти на проверку содержимого переменных, аргументов и пр. Помните, еще совсем недавно, когда отладчики не были реальностью или находились в зачаточном состоянии, как много выражений *printf* или *WriteLn* приходилось писать для проверки содержимого различных элементов данных? Хотя, возможно вы и не помните, вероятно, вы не так стары, как мы.

> **Visual Studio 2005** Visual Studio 2005 предлагала настоящий отладчик с новой функциональностью, что являло собой значительный прогресс по сравнению с временами, когда приходилось писать все виды выражений в консоли. Новые средства визуализации данных позволяли видеть форматированный XML, а не просто длинную

строку кода. Более того, они представляли массивы в более удобном для восприятия виде – как список элементов и их индексов – и чтобы увидеть это, достаточно было навести курсор мыши на объект. Пример представлен на рис. 7-9.

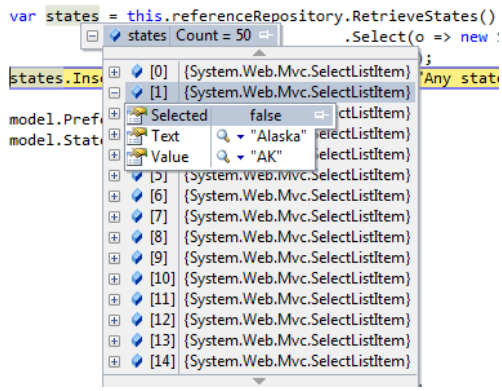


Рис. 7-9 Сравнение представления в виде коллекции и в виде массива в отладчике в Visual Studio 2010 и в Visual Studio 2005

Все эти техники визуализации данных в виде всплывающей подсказки по-прежнему доступны в Visual Studio 2010, но при этом предлагается ряд нововведений, которые повышают эффективность всплывающих подсказок по данным. В сочетании с улучшениями всплывающих подсказок в Visual Studio 2010 появилась еще одна возможность – поддержка одновременной работы с несколькими мониторами. Всплывающие подсказки по данным очень полезны для разработчика. А наличие возможности поместить их на второй экран может намного облегчить отладку, позволяя всегда иметь перед глазами данные, которые всегда должны быть в контексте, на втором мониторе. Рассмотрим поэтапно, как использовать эти возможности:

106. В файле Global.ascx.cs установите точки останова в строках 89 и 91 – строках, начинающихся с *var authority* и *var pathAndQuery*, соответственно.

107. Поэкспериментируем с новыми возможностями всплывающих подсказок по данным. Запустим отладчик, нажав F5. Когда отладчик встречает первую точку останова, наведите курсор на слово *url* и щелкните значок вешки, как показано на рис. 7-10.

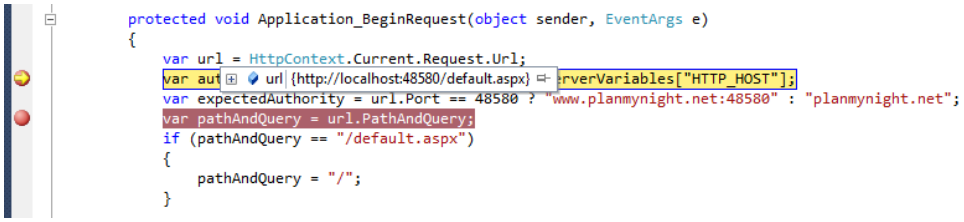


Рис. 7-10 Новая возможность всплывающих подсказок по данным – вешка

108. Справа от строки кода должен располагаться закрепленная всплывающая подсказка по данным (как показано на рис. 7-11 слева). Если навести курсором мыши на эту подсказку, появится панель управления всплывающей подсказки по данным (как показано на рис. 7-11 справа).

Рис. 7-11 Слева – закрепленная всплывающая подсказка по данным, и справа – ее панель управления

Примечание Также в поле точек останова должна отображаться голубая вешка, указывающая на то, что данная всплывающая подсказка закреплена. Вешка выглядит так: . Поскольку в этой строке располагается точка останова, фактически вешка находится под ней. Чтобы увидеть вешку, просто переключите точку останова, щелкнув ее в поле. Один щелчок снимет точку останова, второй – вернет ее назад.

Примечание По щелчку указывающей вниз двойной стрелки на панели управления всплывающей подсказки по данным появляется строка, в которую можно ввести комментарий к этой подсказке, как показано на рис. 7-12.

Также можно полностью удалить всплывающую подсказку, нажав кнопку X на панели управления всплывающей подсказки.

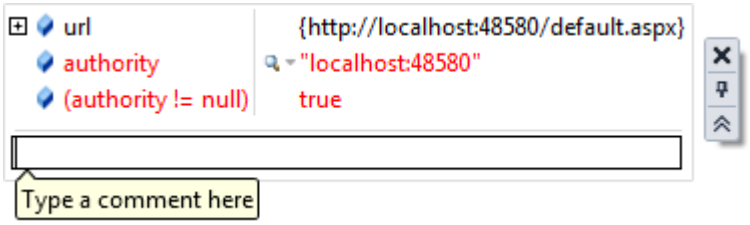


Рис. 7-12 Вставка комментария к всплывающей подсказке по данным

109. Еще одно замечательное свойство новой всплывающей подсказки по данным в том, что есть возможность вставить любое выражение, и оно будет вычислено прямо в ходе сеанса отладки. Например, щелкните правой кнопкой имя всплывающей подсказки, в данном случае это *url*, выберите Add Expression (Добавить выражение), введите **authority** и добавьте еще одно, например, **(authority != null)**. Выражение тут же вычисляется и будет вычисляться на протяжении всего сеанса отладки при каждой остановке отладчика на этих точках останова. На данном этапе сеанса отладки эти выражения должны быть равны *null* и *false*, соответственно.

110. Нажмите F10, чтобы выполнить строку, в которой остановился отладчик, и проверьте всплывающую подсказку для url и оба выражения. Они должны содержать значения, соответствующие текущему контексту, как показано на рис. 7-13.

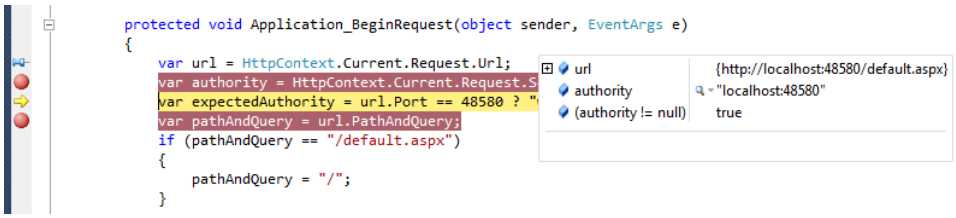


Рис. 7-13 Закрепленная всплывающая подсказка для url с двумя вычисленными выражениями

111. Замечательно иметь возможность просматривать мини-окно в том месте, где оно имеет значение – прямо там, где выполняется код – но оно перекрывает отлаживаемый исходный код. Окно всплывающей подсказки можно перетянуть в любую точку редактора кода, как показано на рис. 7-14.

Рис. 7-14 Убираем закрепленное окно подсказки с исходного кода

112. Закрепленное окно подсказки остается там, где оно закреплено, поэтому при переходе к другому файлу его не будет видно. Но в некоторых случаях окно всплывающей подсказки должно быть в поле зрения постоянно, например, для глобальных переменных, которые всегда в контексте, или для сценариев с использованием множества мониторов. Чтобы переместить окно подсказки, надо сначала открепить его, щелкнув вешку на панели управления всплывающей подсказки. Окно изменит цвет и станет желтым, что свидетельствует о возможности перемещать его в любое место, например, поверх Solution Explorer, на второй монитор, на рабочий стол или в любое другое окно. Пример можно увидеть на рис. 7-15.

Рис. 7-15 Открепленная всплывающая подсказка поверх Solution Explorer и рабочего стола Windows

Примечание Когда отладчик переходит к другому файлу или методу, и окно всплывающей подсказки не закреплено, оно может отображать элементы, не соответствующие текущему контексту. Тогда окно всплывающей подсказки будет выглядеть, как показано на рис. 7-16. Можно щелкнуть кнопку , чтобы отладчик повторил попытку вычислить значение элемента. Но если элемент не имеет значения в данном контексте, весьма вероятно, что это ни к чему не приведет.

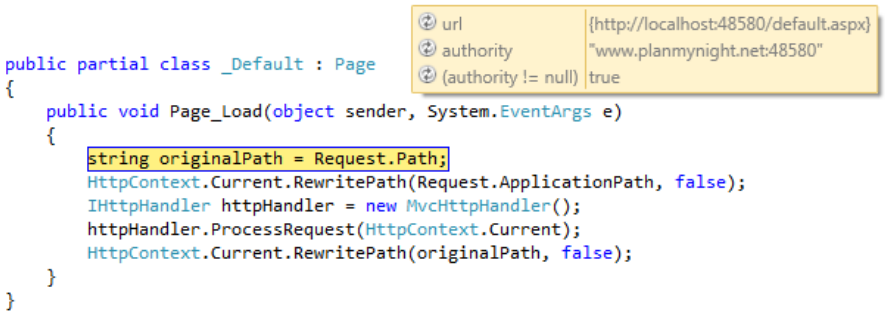


Рис. 7-16 Окно подсказки с элементами, не входящими в текущий контекст

Примечание При попытке закрепить окно всплывающей подсказки вне редактора будет выведено сообщение об ошибке, как показано на рис. 7-17.

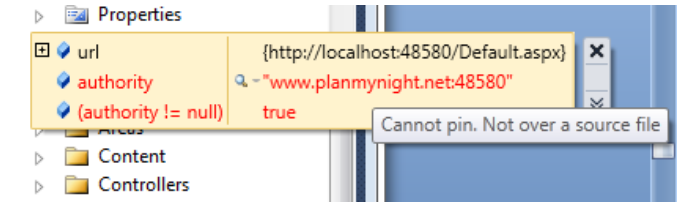


Рис. 7-17 Сообщение об ошибке, которое появляется при попытке закрепить окно всплывающей подсказки вне редактора кода

Примечание Используемый вами номер порта может отличаться от того, который представлен на приводимых здесь снимках экрана. Это нормально, поскольку Веб-сервер, включенный с Visual Studio, выбирает порт для использования случайным образом.

Примечание Закрепить можно и любой дочерний элемент закрепленного элемента. Например, разверните содержимое элемента url, нажав знак плюс (+). Вы увидите, что имеется возможность закрепить его дочерний элемент, как показано на рис. 7-18.

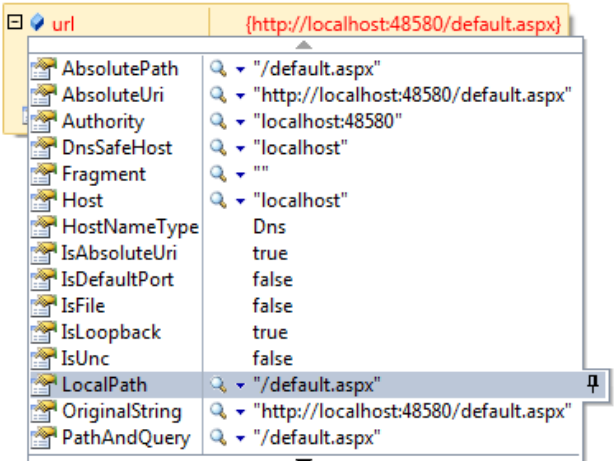
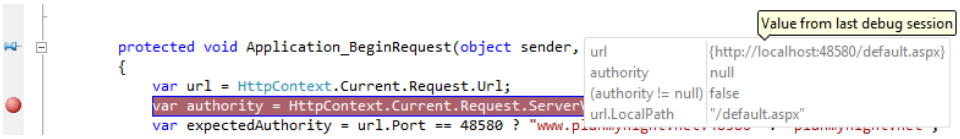


Рис. 7-18 Закрепленный дочерний элемент элемента url в окне всплывающей подсказки

113. Прежде чем остановить выполнение отладчика, вернитесь в Global.ascx.cs, если еще не сделали этого, и повторно закрепите окно всплывающей подсказки. После этого завершите сеанс отладки, щелкнув кнопку Stop Debugging (Остановить отладку) на панели инструментов отладки ( ) или нажав клавиши Shift+F5. Теперь если провести курсором мыши по голубой вешке в поле точек останова, можно увидеть значения последнего сеанса отладки. Такая функциональность является приятным улучшением по



сравнению с окном контрольных значений. На рис. 7-19 показано, что должно быть выведено на экран.

Рис. 7-19 Значения последнего сеанса отладки для закрепленной всплывающей подсказки

Примечание Как и точки останова, подсказки по данным могут экспортироваться или импортироваться. Для этого в меню Debug предусмотрены опции Export DataTips и Import DataTips, соответственно.

Использование отладчика минидампа

При эксплуатации реальных проектов часто возникают ситуации, когда группе поддержки продукта приходится работать с минидампом. Кроме описаний и шагов воспроизведения ошибок, это может быть единственным средством для отладки приложений заказчика. В Visual Studio 2010 отладка минидампа доработана и улучшена.

Visual Studio 2005 В Visual Studio 2005 для отладки управляемого кода или файлов минидампа приходилось использовать расширение – инструмент SOS, который загружался в отладчик с помощью окна интерпретации (Immediate). Отладчик подключался с разрешенной отладкой как управляемого, так и неуправляемого кода. При этом никакие данные не передавались ни в стек вызовов, ни в окно локальных переменных (Locals). При отладке файлов минидампа команды для инструмента SOS вводились в окне интерпретации. Отладка приложений, написанных неуправляемым кодом, выполнялась с использованием обычных окон и инструментов отладки. Познакомиться с этими вопросами более подробно или просто обновить их в памяти можно в колонке *Bug Slayer* (Убийца ошибок) журнала MSDN по адресу *http://msdn.microsoft.com/en-us/magazine/cc164138.aspx*.

Теперь рассмотрим, как был доработан отладчик минидампа. Для начала смоделируем сбой, на основании которого можно будет сформировать файл минидампа:

114. В Solution Explorer в проекте PlanMyNight.Web измените имя файла Default.aspx на **DefaultA.aspx** (обратите внимание на букву *A*, добавленную после слова «Default»).

115. Убедитесь, что убрали все точки останова из проекта. Для этого откройте окно Breakpoints и удалите все имеющиеся там точки останова любым из рассмотренных ранее в этой главе способом.

116. Нажмите F5, чтобы начать отладку приложения. В зависимости от производительности компьютера вскоре после завершения процесса сборки должно быть выведено необработанное исключение типа *HttpException*. Несмотря на то что в данном случае ошибка проста, последовательно пройдем все шаги создания и отладки файла минидампа. На рис. 7-20 показано, что должно отображаться на экране в данный момент.

Рис. 7-20 Необрабатываемое исключение, которое должно быть сформировано

117. Пора создавать файл минидампа для этого исключения. Перейдите в меню Debug и выберите Save Dump As (Сохранить дамп как), как показано на рис. 7-21. Вы должны видеть имя процесса, из которого было сформировано исключение. В данном случае это процесс Cassini или Personal Web Server в Visual Studio. Оставьте предлагаемое имя файла (WebDev.WebServer40.dmp) без изменений и сохраните файл на своем рабочем столе. Обратите внимание, что создание файла минидампа может занять некоторое время, потому что его размер будет около 300 MB.

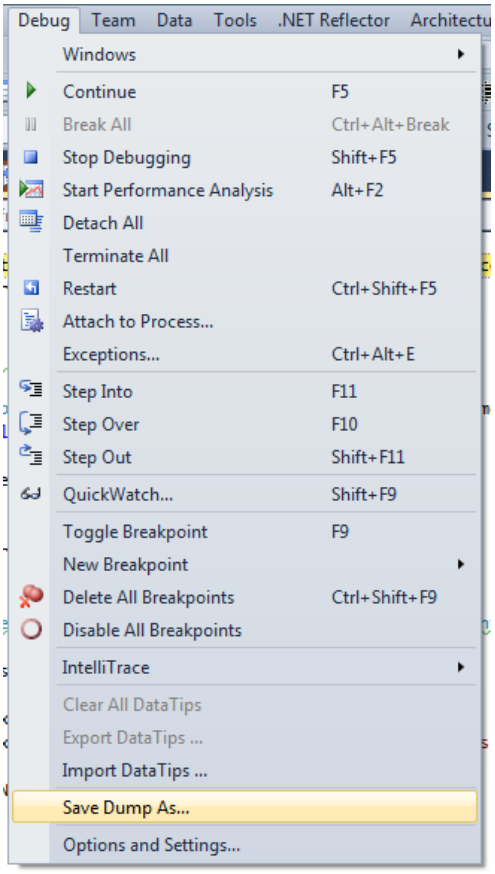


Рис. 7-21 Сохранение файла манидампа

118. Остановите отладку, нажав Shift+F5 или кнопку Stop Debugging.

119. Перейдите в меню File и закройте свое решение.

120. Чтобы загрузить свой файл манидампа WebDev.WebServer40.dmp, в меню File щелкните Open и выберите рабочий стол. При этом откроется страница Minidump File Summary (Сведения о файле минидампа), содержащая некоторые общие сведения об ошибке, которую мы пытаемся устранить (рис. 7-22). Еще до начала отладки эта страница предоставляет основные данные, такие как имя процесса, архитектура процесса, версия операционной системы, версия CLR, загруженные модули, а также некоторые действия, которые можно предпринять в этот момент. Прямо здесь можно задать пути к файлам символов. Очень удобно, что список модулей (Modules) включает версию и путь к модулю, что облегчает поиск символов и исходного кода. Используется CLR версия 4.0, т.е. отладку можно выполнять в Visual Studio 2010.

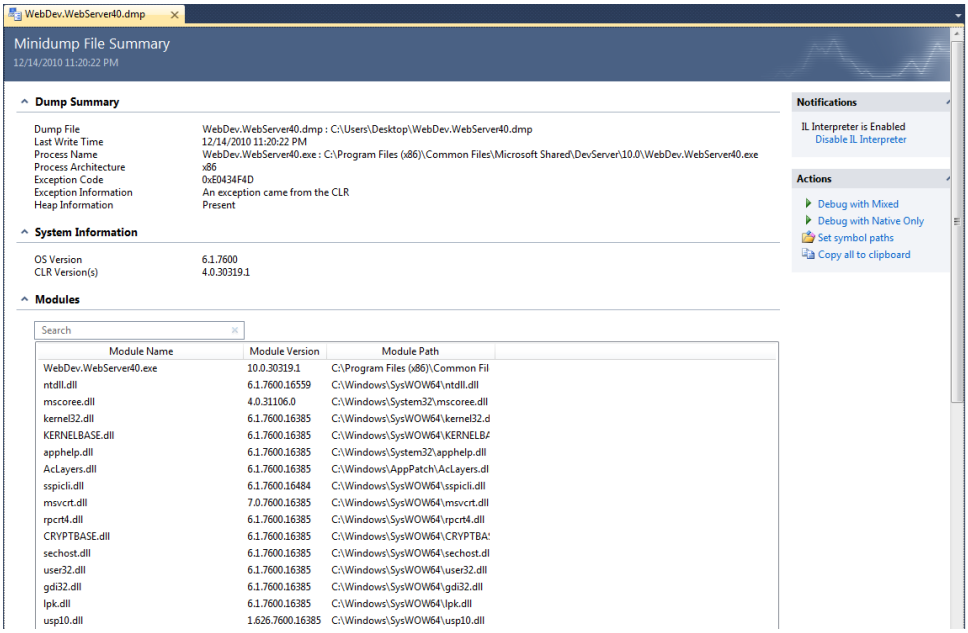


Рис. 7-22 Страница сведений о файле минидампа

121. Чтобы начать отладку, в списке Actions (Действия) в правой части страницы Minidump File Summary щелкните ссылку Debug With Mixed (Отладка в смешанном режиме).

122. Практически сию минуту на экран должен быть выведен первый этап обработки исключения, как показано на рис. 7-23. В данном случае сообщается, какая ошибка произошла, хотя это не всегда так. Продолжим, нажав кнопку Break (Прервать).

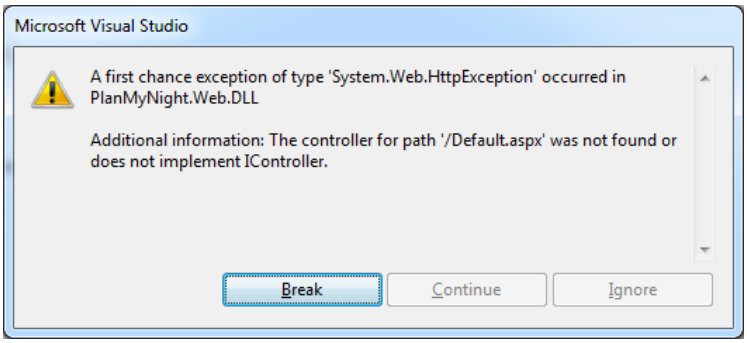


Рис. 7-23 Первый этап обработки исключения

123. Вы должны видеть подсвеченную зеленым строку, которая указывает на то, какая команда привела к формированию исключения. Если взглянуть на исходный код, в окне Autos (Автоматические значения) можно увидеть, что переменная *controllerExport* (Экспорт контроллера) имеет значение *null*. И прямо перед этим мы задали, что если эта переменная *null* и файл для загрузки не найден, должно формироваться исключение *HttpException*. В данном случае, как видно в окне Locals в переменной *controllerName* (Имя контроллера), должен загружаться файл Default.aspx. В окнах Locals и Autos можно также видеть множество других переменных, объектов и прочих элементов, отражающих текущий контекст. Здесь только один вызов относится к нашему коду, поэтому стек вызовов показывает, что весь код до и после этого вызова является внешним для нашего процесса. Если бы цепочка вызовов нашего кода была длиннее, мы могли бы перемещаться назад и вперед по коду и просматривать переменные. На рис. 7-24 показано сводное представление всего вышеописанного.

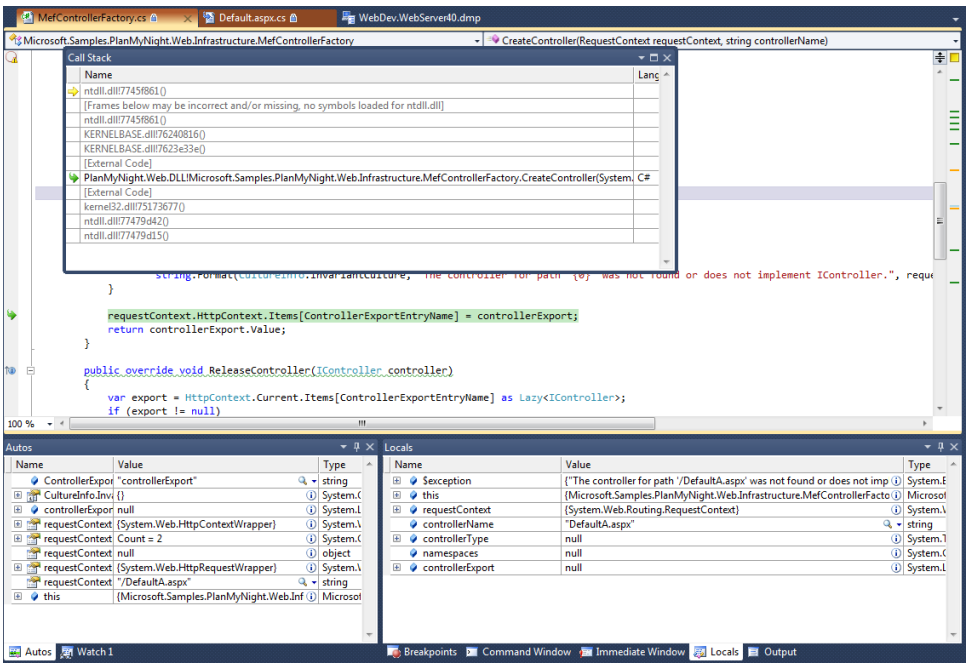


Рис. 7-24 Окна Autos, Locals и Call Stack и следующая команда к выполнению

124. Итак, ошибка найдена. Останавливаем отладку нажатием клавиш Shift+F5 или кнопки Stop Debugging. Чтобы исправить ошибку, опять загружаем решение PlanMyNight и исправляем имя файла **default.aspx**. Переходим в меню Build, выбираем Rebuild Solution (Выполнить повторную сборку решения) и выполняем повторную сборку решения. Затем нажимаем F5, и приложение должно опять исправно работать.

Преобразования Web.Config

Следующая возможность, хотя и незначительная, но она порадует многих разработчиков, потому что сбережет им время при отладке. Преобразования Web.Config позволяют сохранять файлы преобразований, отражающие разницу между средами отладки и производственной эксплуатации. Например, часто в разных средах используются разные строки подключения. Таким образом, создание файлов преобразований – ASP.NET предоставляет средства для изменения (преобразования) файлов web.config – обеспечит использование соответствующих строк для соответствующих сред. Более подробно о том, как это реализуется, рассказывает следующая статья на сайте MSDN: *http://go.microsoft.com/fwlink/?LinkId=125889*.

Разработка модульных тестов

Инфраструктура и средства модульного тестирования в Visual Studio 2010 Professional остались неизменными. В других редакциях Visual Studio 2010 в управление тестами и средства тестирования внесены действительно заметные изменения. В Visual Studio 2010 Premium и Visual Studio 2010 Ultimate предлагаются такие возможности, как модульные тесты пользовательского интерфейса, IntelliTrace и Microsoft Test Manager 2010. Все возможности Управления жизненным циклом приложения подробно рассматриваются на сайте MSDN в статье *http://msdn.microsoft.com/en-us/library/ee789810(VS.100).aspx*.

> **Visual Studio 2005** Чтобы создавать и выполнять тесты, поставляемые в комплекте с Visual Studio 2005, требовалось иметь Visual Studio 2005 Team System или Visual Studio 2005 Team Test. Альтернативным вариантом в те времена было использование продукта сторонних производителей, такого как Nunit.

В данной части главы мы покажем, как добавлять модульный тест для одного из классов приложения Plan My Night. Не будем тратить время на определение того, что такое модульный тест или что он должен включать. Лучше рассмотрим, как добавлять и выполнять тесты в Visual Studio 2010.

Добавим в приложение Plan My Night модульные тесты для надстройки Print Itinerary. Для создания модульных тестов откроем решение, находящееся в папке сопроводительных материалов книги. Если не помните, как это делается, можете вернуться к первой странице данной главы за инструкциями. Когда решение открыто, просто последовательно выполните следующие шаги:

125. В Solution Explorer разверните проект PlanMyNight.Web и папку Helpers. Щелкните двойным щелчком файл ViewHelper.cs, чтобы открыть его в редакторе коды. Взгляните на рис. 7-25 и убедитесь, что делаете все правильно.

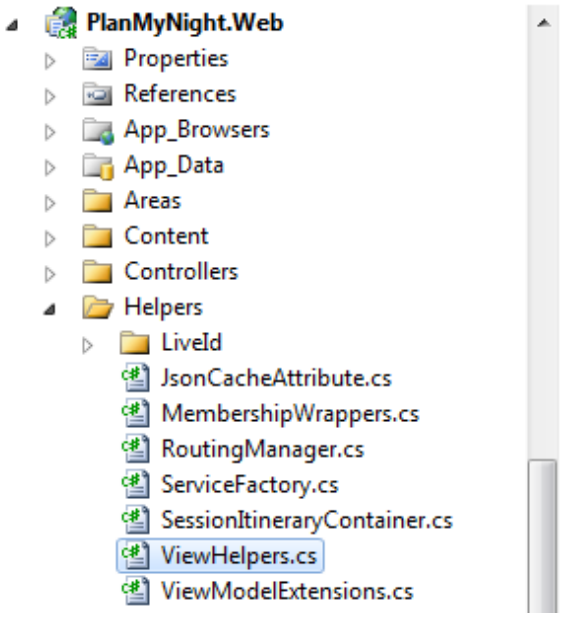


Рис. 7-25 Проект PlanMyNight.Web и файл ViewHelper.cs в Solution Explorer

126. Редактор кода позволяет добавлять модульные тесты двумя разными способами. Можно щелкнуть правой кнопкой имя класса или имя метода и выбрать в появившемся меню Create Unit Tests (Создать модульные тесты). Также можно перейти в меню Test (Тест) и выбрать New Test (Новый тест). Рассмотрим первый способ создания модульных тестов, при котором Visual Studio автоматически формирует некоторый исходный код. Щелкаем правой кнопкой метод *GetFriendlyTime* (Представить время в удобном для восприятия виде) и выбираем Create Unit Tests (рис. 7-26).

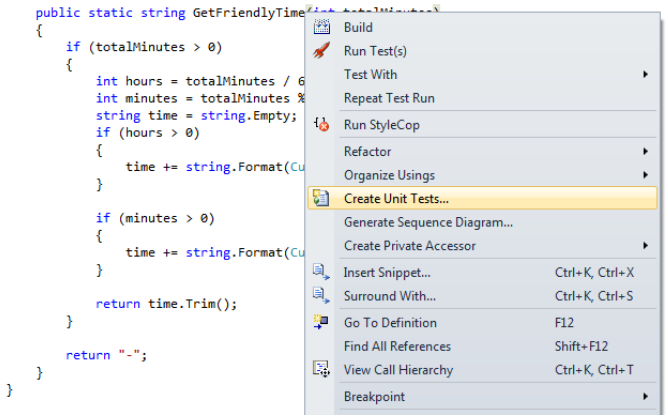


Рис. 7-26 Контекстное меню для создания модульных тестов

127. По нажатию Create Unit Tests на экран выводится диалоговое окно, в котором по умолчанию предлагается метод, выбранный нами в этом классе. Перейдем к полю с раскрывающимся списком внизу диалогового окна и выберем в нем, где будут создаваться модульные тесты. В данном случае это PlanMyNight.Web.Tests. Если есть необходимость создать новый проект для тестов, просто выбираем в этом списке Create A New Visual C# Test Project (Создать новый тестовый проект Visual C#). На рис. 7-27 представлено, как все это должно выглядеть.

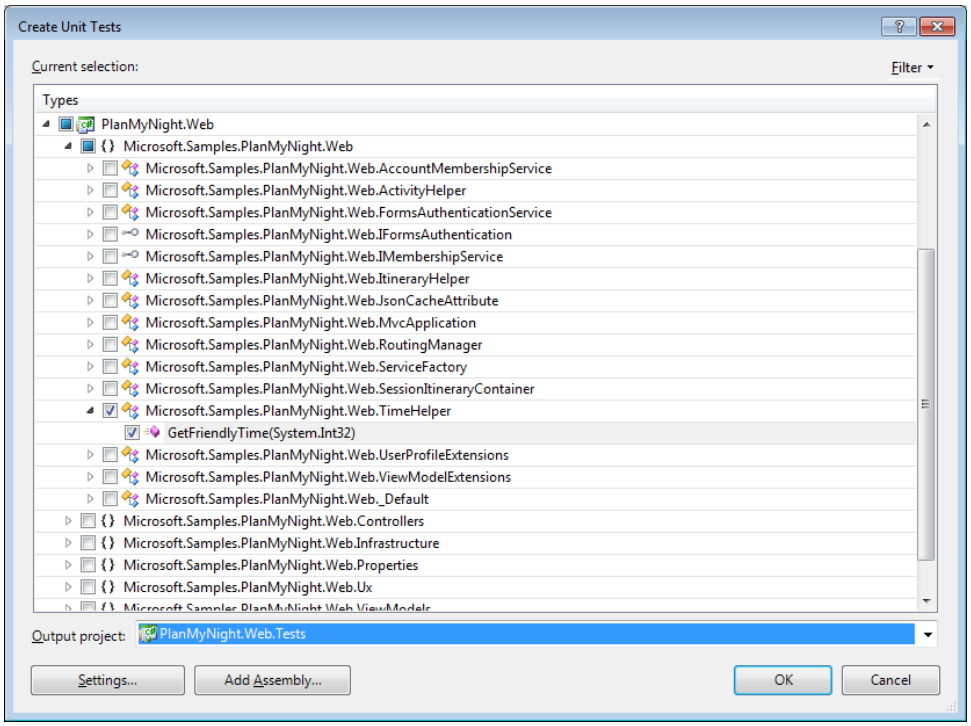


Рис. 7-27 Выбор метода, для которого требуется создать модульный тест

128. По щелчку OK диалоговое окно переключается в режим формирования тестового случая, отображается индикатор хода выполнения. По завершении этого процесса создается файл *TimeHelperTest.cs*, содержащий автоматически сформированные заглушки кода, которые мы будем дорабатывать.

129. Удалите этот метод и его атрибуты, потому что для него мы создадим три тестовых случая. Удаляем следующий код:

```
/// <summary>
/// Тест для GetFriendlyTime
///</summary>
// TODO: Гарантированно обеспечить, что атрибут UrlToTest содержит URL
// страницы ASP.NET (например, http://.../Default.aspx).
// Это необходимо, чтобы модульный тест выполнялся на Веб-сервере
// независимо от того, выполняется ли тестирование страницы,
// Веб-сервиса или WCF-сервиса.
[TestMethod()]
[HostType("ASP.NET")]
[AspNetDevelopmentServerHost("C:\\Users\\Patrice\\Documents\\Chapter
7\\code\\PlanMyNight.Web", "/")]
[UrlToTest("http://localhost:48580/")]
public void GetFriendlyTimeTest()
{
    int totalMinutes = 0; // TODO: Инициализировать с соответствующим значением
    string expected = string.Empty; // TODO: Инициализировать с соответствующим
значением
    string actual; actual = TimeHelper.GetFriendlyTime(totalMinutes);
    Assert.AreEqual(expected, actual);
    Assert.Inconclusive("Verify the correctness of this test method[1].");
}
```

130. Добавляем три простых тестовых случая для проверки ключевых сценариев, используемых в Plan My Night. Для этого вставляем следующий исходный код прямо под атрибутами метода, оставшимися после удаления блока кода в шаге 5:

```
[TestMethod]
public void ZeroReturnsSlash()
{
    Assert.AreEqual("-", TimeHelper.GetFriendlyTime(0));
}
```

[1] Проверяем правильность этого тестового метода (*прим. переводчика*).

```
[TestMethod]
public void LessThan60MinutesReturnsValueInMinutes()
{
    Assert.AreEqual("10m", TimeHelper.GetFriendlyTime(10));
}

[TestMethod()]
public void MoreThan60MinutesReturnsValueInHoursAndMinutes()
{
    Assert.AreEqual("2h 3m", TimeHelper.GetFriendlyTime(123));
}
```

131. В проекте PlanMyNight.Web.Tests создадим папку для решения и назовем ее **Helpers**. Перенесем файл TimeHelperTests.cs в эту папку. После этого проект должен выглядеть, как показано на рис. 7-28.

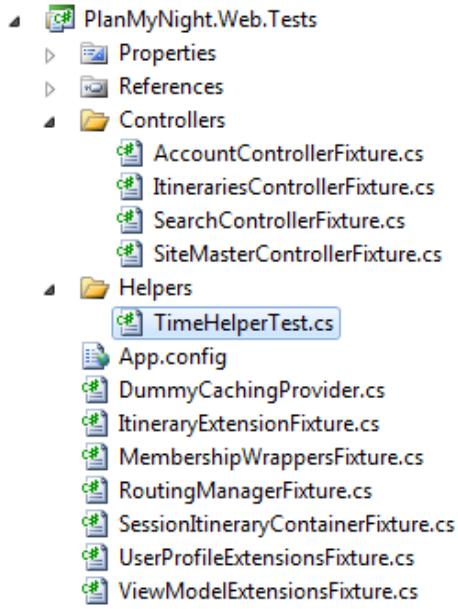


Рис. 7-28 TimeHelperTest.cs в папке Helpers

132. Пора выполнить только что созданные тесты. Чтобы выполнить только те тесты, которые мы только что создали, перейдем в редактор кода и наведем курсором на класс *public class TimeHelperTest*. Теперь можно либо зайти в меню Test, выбрать Run и затем Tests In Current Context (Тесты в текущем контексте); либо выполнить все то же самое посредством сочетания горячих клавиш CTRL+R, T (рис. 7-29).

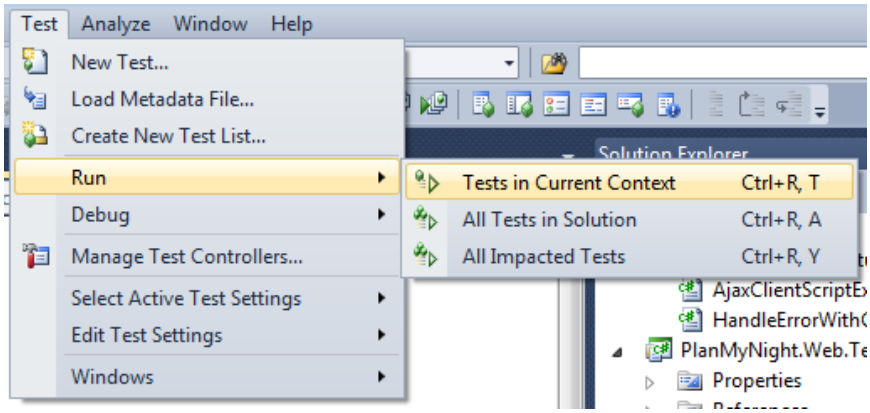


Рис. 7-29 Меню выполнения теста

133. Это действие обеспечит выполнение только трех созданных нами тестов. Внизу редактора должно появиться окно Test Results (Результаты тестирования) с результатами выполнения этих тестов (рис. 7-30).

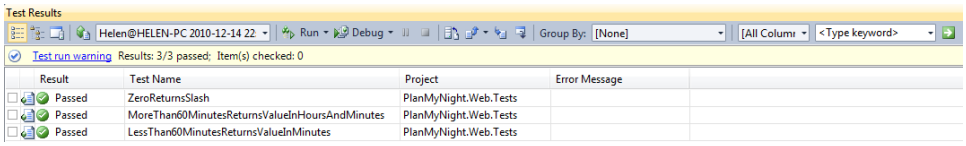


Рис. 7-30 Окно Test Results с результатами выполнения вновь созданных тестов

> **Дополнительные сведения** В зависимости от заданных параметров поведение при выборе опции Tests In Current Context может быть разным. Например, если выбран тестовый метод, такой как *ZeroReturnsSlash,* будет выполнен только этот тестовый случай. Но если щелкнуть вне класса теста, будут выполнены все тестовые случаи, что эквивалентно выбору опции All Tests In Solution (Все тесты решения).

Новое окно Threads

Появление многоядерных процессоров и многочисленных инструментов, предоставляемых языками программирования для работы с ними, обусловило новую проблему: отладка многопоточных приложений. Новое окно Threads (Потоки) обеспечивает разработчику возможность приостанавливать потоки и выполнять поиск артефактов по стеку вызовов, аналогично тому, как это происходило при использовании знаменитого Process Monitor от SysInternals (*http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx*). Окно Threads можно открыть во время отладки приложения из меню Debug, нажав опцию Windows And Threads (Окна и потоки). На рис. 7-31 представлено окно Threads во время отладки приложения Plan My Night.

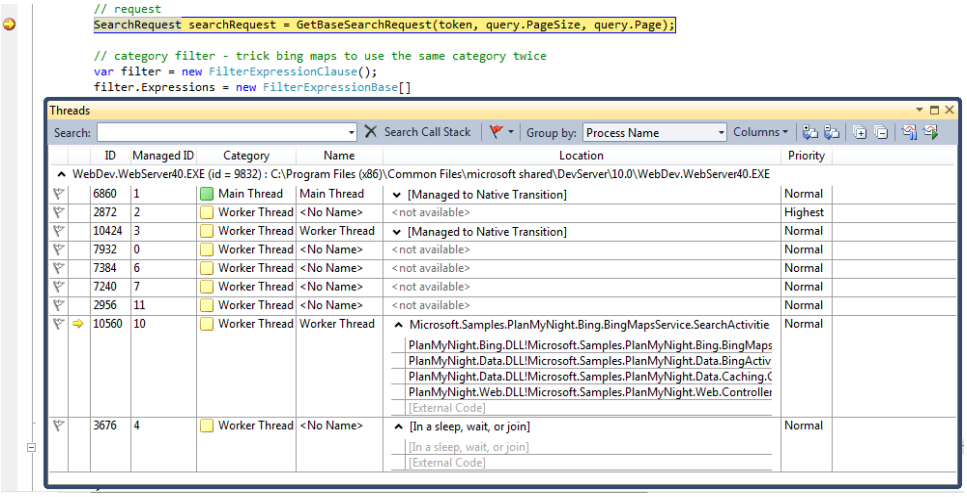


Рис. 7-31 Окно Threads во время отладки Plan My Night

Окно Threads позволяет приостанавливать потоки и вновь запускать их, когда требуется. Это может быть действительно полезным, когда необходимо изолировать конкретные эффекты. Можно выполнять отладку как управляемого, так и неуправляемого кода. Эта новая возможность отладчика в Visual Studio 2010, безусловно, понравится разработчикам, которые столкнуться с необходимостью отладки приложений, использующих потоки.

> **Visual Studio 2005** В Visual Studio 2005 приходилось использовать множество не интегрированных в Visual Studio инструментов или инструментов сторонних производителей. И все равно в большинстве случаев для выявления ошибок многопоточной обработки разработчику по-прежнему приходилось полагаться на свое чутье и опыт.

Заключение

В данной главе мы научились управлять сеансами отладки с помощью улучшенных возможностей работы с точками останова и использования новых техник проверки и визуализации данных. Также мы узнали, как с помощью отладки минидампа и новых инструментов решать проблемы, возникающие при работе приложений, не выводя их из эксплуатации. В этой главе было рассмотрено, как повысить качество кода посредством модульных тестов и как сделать это в Visual Studio 2010 Professional. Компьютеры с множеством процессоров или с многоядерными процессорами в наши дни стали нормой, как и многопоточные приложения. Поэтому особенно приятным является тот факт, что Visual Studio 2010 предлагает специальные инструменты отладки для выявления проблем в многопоточных приложениях.

Наконец, в ходе главы было показано, насколько в Visual Studio 2010 Professional усовершенствован процесс отладки приложений в целом и какие инструменты получили в свое распоряжение профессиональные разработчики для отладки современных насыщенных разнообразнейшей

функциональностью приложений. Прогресс по сравнению с Visual Studio 2005 очевиден. Примеры, рассмотренные в данной главе, лишь вкратце продемонстрировали, насколько выгоден с точки зрения экономии времени и средств переход к новой инфраструктуре отладки, но даже из них видно, Visual Studio 2010 не просто очередной этап, а значительная веха в развитии Visual Studio. Переход на Visual Studio 2010 обеспечит значительное повышение производительности разработки.

Различные редакции Visual Studio 2010 предоставляют целый ряд замечательных улучшений, касающихся отладки и тестирования. Особо из них выделяется возможность IntelliTrace – *http://msdn.microsoft.com/en-us/library/dd264915(VS.100).aspx* – которая доступна только в Visual Studio 2010 Ultimate и Microsoft Test Manager. IntelliTrace намного улучшает условия работы групп тестирования при использовании Visual Studio 2010 и Visual Studio 2010 Team Foundation Server – *http://msdn.microsoft.com/en-us/library/bb385901(VS.100).aspx*.

Часть III

Переход от Microsoft Visual Studio 2008 к Visual Studio 2010

Авторы Патрис Пелланд, Кен Хайнс и Паскаль Паре

В данной части:

От 2008 к 2010: Бизнес-логика и данные (Паскаль)

От 2008 к 2010: Проектирование восприятия и поведения (Кен)

От 2008 к 2010: отладка приложения (Патрис)

Глава 8

От 2008 к 2010: бизнес-логика и данные

В данной главе рассматривается

- Применение Entity Framework (EF) для создания слоя доступа к данным с использованием существующей базы данных или модель-ориентированного подхода
- Создание типов сущностей в дизайнере Entity Data Model (EDM) с использованием POCO шаблонов в ADO.NET Entity Framework
- Кэширование данных с использованием Microsoft Windows Server AppFabric (ранее известной под кодовым названием «Velocity»)

Архитектура приложения

С помощью приложения Plan My Night (PMN) пользователь может составлять программы своих мероприятий и делиться ими с остальными. Данные хранятся в базе данных Microsoft SQL Server. Программы составляются на основании обращений к Веб-сервисам Bing Maps.

Рассмотрим исходную блок-схему модели данных приложения (рис. 8-1).

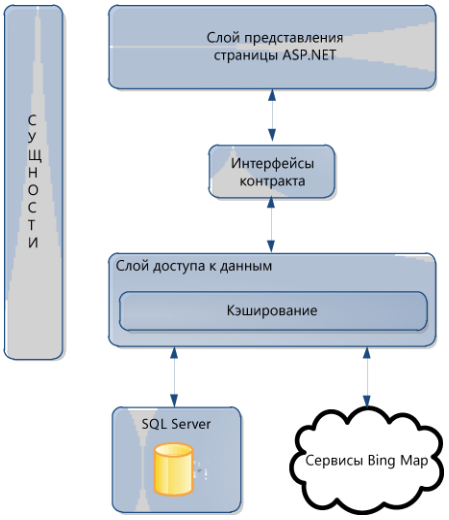


Рис. 8-1 Архитектурная диаграмма приложения PlanMyNight

Описание контрактов и классов сущностей, свободных от каких-либо ограничений, налагаемых методом хранения, позволяет объединять их в сборки, не зависящие от метода хранения. Это обеспечивает четкое разделение слоев представления и доступа к данным.

Опишем интерфейсы контрактов основных компонентов приложения PMN:

- *IItinerariesRepository* (Хранилище маршрутов) будет интерфейсом хранилища данных поездки (база данных Microsoft SQL Server).
- *IActivitiesRepository* (Хранилище действий) позволит выполнять поиск мероприятий (Веб-сервисы Bing Map).
- *ICachingProvider* (Поставщик кэширования) обеспечивает интерфейс кэширования данных (кэширование ASP.NET или кэширование AppFabric Windows Server).

Примечание Это неполный список контрактов, реализованных в приложении PMN.

PMN сохраняет программы мероприятий пользователя в базе данных SQL. Все пользователи смогут комментировать и рейтинговать программы друг друга. На рис. 8-2 представлены таблицы, используемые приложением PMN.

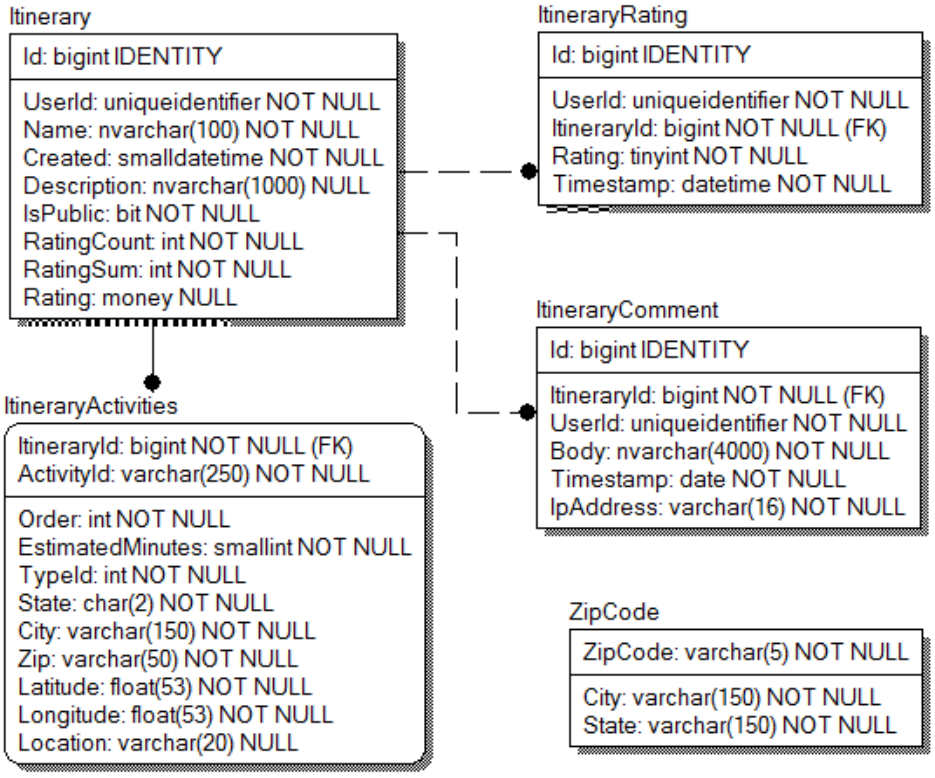


Рис. 8-2 Схема базы данных PlanMyNight

> **Важно** В приложении Plan My Night безопасное хранение учетных данных пользователей обеспечивается посредством возможностей членства ASP.NET. Таблицы хранилища пользователя на рис. 8-2 не показаны. Более подробную информацию об этих возможностях можно найти на сайте MSDN в разделе ASP.NET 4 – Introduction to Membership (ASP.NET 4 – введение в членство) по адресу *http://msdn.microsoft.com/en-us/library/yh26yfzy(VS.100).aspx*.

> **Примечание** Таблица ZipCode (Почтовый индекс) используется как хранилище справочных ресурсов и предоставляет список доступных почтовых индексов и городов. Благодаря этому обеспечивается функциональность автозаполнения запроса поиска пользователя в приложении.

Данные PlanMyNight в Microsoft Visual Studio 2008

Visual Studio 2008 предоставляет все необходимые инструменты для создания приложения, поэтому написание PlanMyNight не составило бы никакого труда. Однако некоторые применяемые тогда технологии требовали написания намного большего объема кода для достижения тех же целей.

Рассмотрим, как создавался бы слой доступа к данным в Visual Studio 2008. Один из подходов – непосредственное использование DataSet (Множество данных) или DataReader (Средство чтения данных) ADO.NET (рис. 8-3). Такое решение предлагает большую гибкость, поскольку обеспечивает полный контроль над доступом к базе данных. С другой стороны, оно имеет определенные недостатки:

- Необходимо знать синтаксис SQL.
- Все запросы специализированы. Изменение в запросе или в таблицах приводит к необходимости обновления всех соответствующих запросов в коде.
- Приходится сопоставлять свойства классов сущностей, используя имя столбца, что довольно утомительно и чревато многочисленными ошибками.
- Приходится самостоятельно управлять отношениями между таблицами.

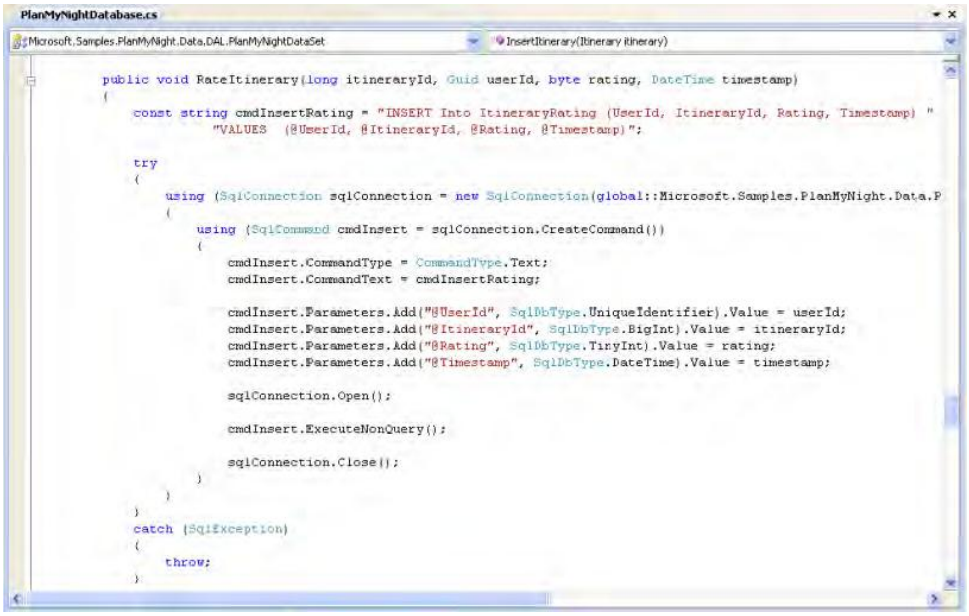


Рис. 8-3 Запрос Insert ADO.NET Insert

Другой подход – использование дизайнера DataSet, предоставляемого Visual Studio 2008. При наличии базы данных с таблицами PMN, можно было бы использовать TableAdapter Configuration Wizard (Мастер настройки TableAdapter) для импорта таблиц базы данных, как показано на рис. 8-4. Автоматически сформированный код предлагает типизированный DataSet. К преимуществам относится контроль типов во время разработки, что обеспечивает автоматическое завершение выражений средствами редактора исходного кода. Но, тем не менее, данный подход имеет и недостатки:

- По-прежнему требуется знание синтаксиса SQL, хотя прямо из дизайнера DataSet имеется возможность доступа к построителю запросов.
- По-прежнему приходится создавать специализированные SQL-запросы, чтобы выполнять все требования контрактов данных.
- Нет возможности управления автоматически сформированными классами. Например, добавление или удаление запроса к таблице из *DataSet* приведет к повторной сборке автоматически сформированных классов *TableAdapter* и может изменить индекс, используемый для запроса. Это усложняет задачу по написанию предсказуемого кода с использованием таких автоматически формируемых классов.
- Автоматически формируемые классы, ассоциированные с таблицами, включают код, зависящий от метода хранения, поэтому придется создавать другое множество простых сущностей и копировать данные туда. Это означает больший объем обработки и более активное использование памяти.

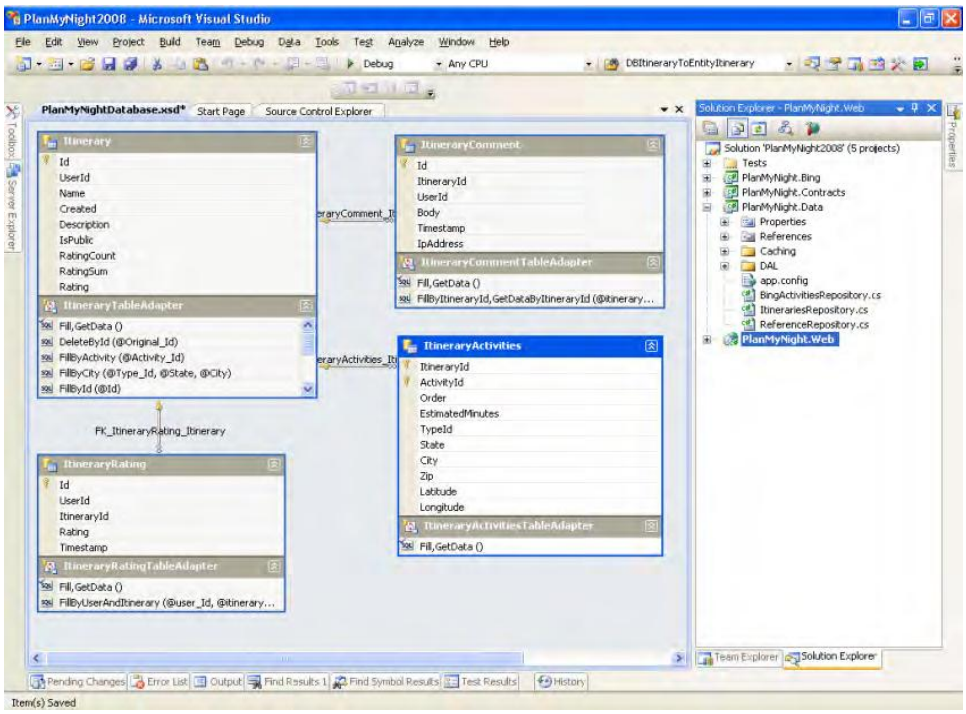


Рис. 8-4 Дизайнер DataSet в Visual Studio 2008

Еще одна технология, доступная в Visual Studio 2008 – LINQ to SQL (L2S). Объектно-реляционный дизайнер (Object Relational Designer) для L2S позволяет без труда добавлять привязки для необходимых приложению таблиц базы данных. Этот подход обеспечивает доступ к строго типизированным объектам и к LINQ для создания запросов для доступа к данным, т.е. знание синтаксиса SQL не требуется. Недостатки такого подхода:

- LINQ to SQL применим только для СУБД SQL Server.
- Ограниченные возможности управления создаваемыми сущностями. Не обеспечивается возможность простого обновления базы данных при изменении ее схемы.
- Автоматически создаваемые сущности включают код, зависящий от метода хранения.

> **Примечание** Для .NET 4.0 Майкрософт рекомендует Entity Framework в качестве решения для сценариев использования LINQ с реляционными базами данных.

В следующих разделах данной главы будут рассмотрены некоторые новые возможности, предоставляемые Visual Studio 2010. Благодаря им создание слоя доступа к данным связано с меньшим объемом написания кода, предоставляется больший контроль над автоматически формируемым кодом, обслуживать и расширять его становится намного проще.

Работа с данными в Visual Studio 2008 с использованием Entity Framework

ADO.NET Entity Framework (EF) позволяет без труда создавать слой доступа к данным приложения через абстрагирование работы с данными от базы данных и создание модели, приближенной к бизнес-требованиям приложения. EF была существенно улучшена в выпущенной версии .NET Framework 4.

Будем использовать приложение PlanMyNight в качестве примера использования некоторых возможностей EF. В следующих двух разделах демонстрируются два разных подхода автоматического формирования модели данных PMN. В первом случае EF создаст Entity Data Model (EDM) из существующей базы данных. Во втором – используется модель-ориентированный подход, при котором сначала в дизайнере EF создаются сущности, а затем для создания базы данных, в которой может храниться EDM, автоматически формируются сценарии на языке описания данных (Data Definition Language, DDL).

Visual Studio 2008 Первая версия Entity Framework была выпущена с Visual Studio 2008 Service Pack 1. Вторая версия EF, вошедшая в состав .NET Framework 4.0, предлагает множество новых возможностей, которые облегчат создание приложений для работы с данными. Назовем некоторые из них:

- Шаблоны формирования кода T4, которые можно настроить соответственно собственным требованиям.
- Возможность определения собственных POCO (Plain Old CLR Objects[1]), что гарантирует разделение сущностей и технологии хранения.
- Модель-ориентированная разработка, т.е. сначала создается модель сущностей, на базе которой Visual Studio 2010 формирует базу данных.
- Поддержка подхода «только код», что позволяет работать с Entity Framework, используя сущности POCO, и без файла EDMX.
- Отложенная загрузка для связанных сущностей, что обепечивает их загрузку из базы данных только по мере надобности.
- Самоотслеживающиеся сущности, которые регистрируют собственные изменения на клиенте и передают их в хранилище базы данных.

В следующих разделах мы рассмотрим некоторые из этих новых возможностей.

Дополнительные сведения Большое количество ресурсов по ADO.NET Entity Framework в .NET 4 также предлагается на сайте MSDN в разделе Data Developer Center (Центр решений по работе с данными) по адресу http://msdn.microsoft.com/en-us/data/aa937723.aspx.

EF: импорт существующей базы данных

Мы будем работать с уже существующим решением, в котором описаны основные проекты приложения PMN. Если сопроводительные материалы данной книги установлены в каталог по умолчанию, интересующее нас решение находится по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 8\Code\ExistingDatabase. Щелкните двойным щелчком файл PlanMyNight.sln.

Это решение включает все проекты, показанные на рис. 8-5.

- PlanMyNight.Data: слой доступа к данным приложения.
- PlanMyNight.Contracts: сущности и контракты.
- PlanMyNight.Bing: сервисы Bing Map.
- PlanMyNight.Web: слой представления.
- PlanMyNight.AppFabricCaching: кэширование AppFabric.

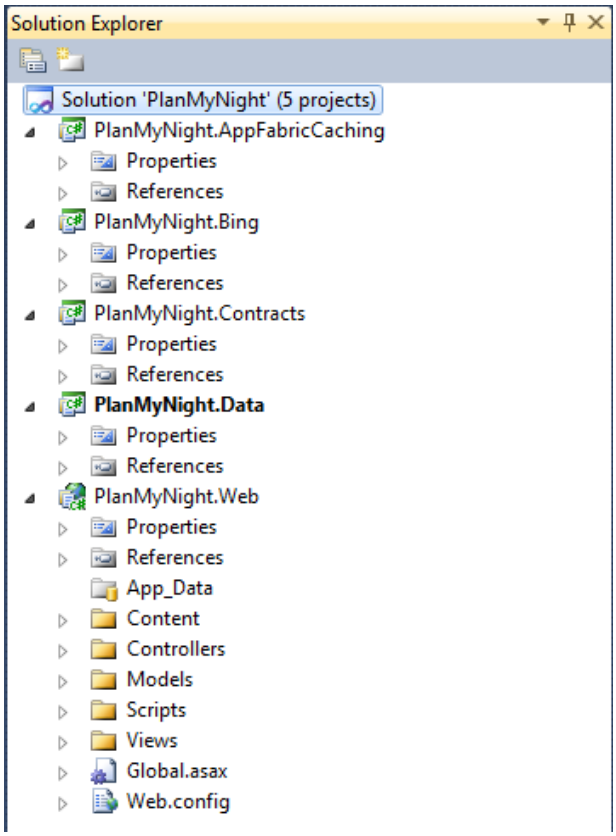


Рис. 8-5 Решение PlanMyNight

EF позволяет без труда импортировать существующую базу данных. Рассмотрим этот процесс поэтапно.

Первый шаг – добавление EDM в проект PlanMyNight.Data. Щелкаем правой кнопкой мыши проект PlanMyNight.Data, выбираем Add (Добавить) и New Item (Новый элемент). Выбираем элемент ADO.NET Entity Data Model и меняем его имя на **PlanMyNight.edmx**, как показано на рис. 8-6.

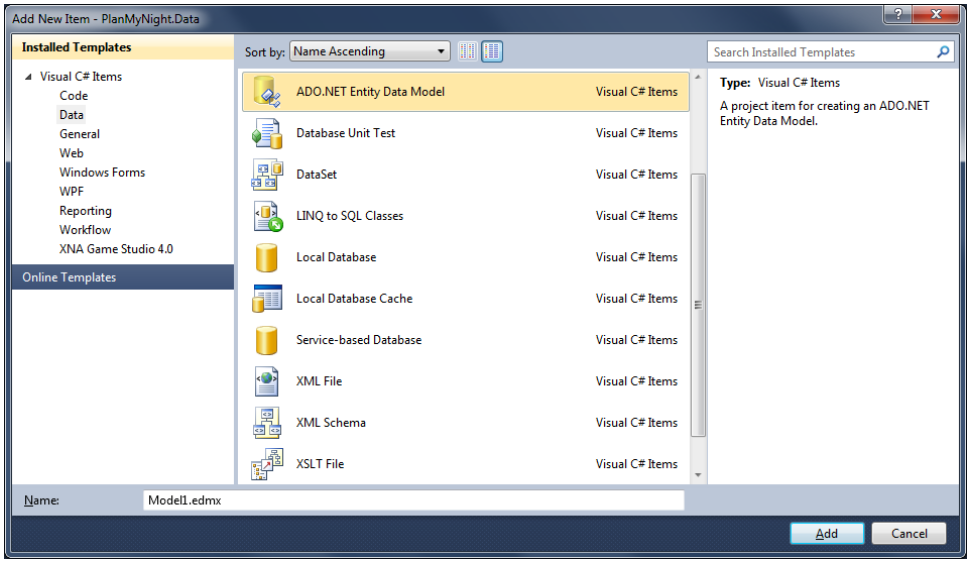


Рис. 8-6 Диалоговое окно Add New Item с ADO.NET Entity Data Model

Первое диалоговое окно мастера Entity Data Model Wizard позволяется выбрать содержимое модели. Мы собираемся создать модель из существующей базы данных. Выбираем Generate From Database (Создать из базы данных) и щелкаем Next (Далее).

Необходимо подключиться к существующему файлу базы данных. Щелкаем New Connection (Создать подключение). В диалоговом окне Choose Data Source (Выбрать источник данных) выбираем файл базы данных Microsoft SQL Server и щелкаем Continue (Продолжить). Выбираем файл %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 8\ExistingDatabase\PlanMyNight.Web\App_Data\PlanMyNight.mdf (рис. 8-7)

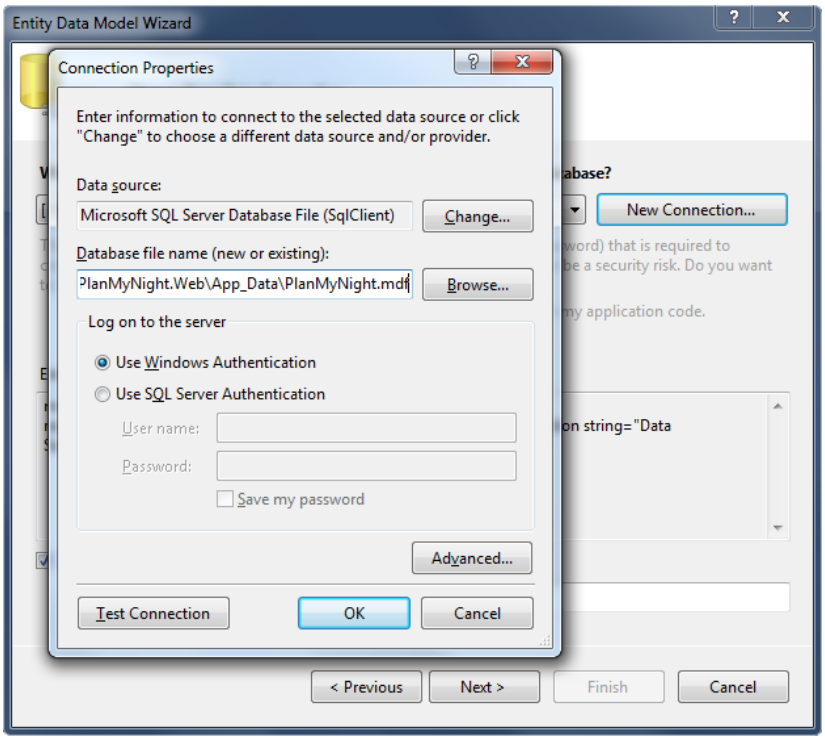


Рис. 8-7 Мастер EDM, диалог создания подключения к базе данных

Все остальные поля формы пока оставляем без изменений и щелкаем Next.

> **Примечание** На экран будет выведено предупреждение о том, что локальный файл данных не входит в состав текущего проекта. Поскольку мы не хотим копировать файл базы данных в текущий проект, щелкните No (Нет), чтобы закрыть этот диалог.

В диалоговом окне Choose Your Database Objects (Выберите объекты своей базы данных) выберите таблицы Itinerary (План мероприятий), ItineraryActivities (Мероприятия), ItineraryComment (Комментарий к плану мероприятий), ItineraryRating (Рейтинг плана мероприятий) и ZipCode (Почтовый индекс), а также представление UserProfile (Профиль пользователя). Выберите хранимую процедуру RetrieveItinerariesWithinArea (Извлечение плана мероприятий в определенном районе). В поле Model Namespace (Пространство имен модели) зададим **Entities** (Сущности), как показано на рис. 8-8.

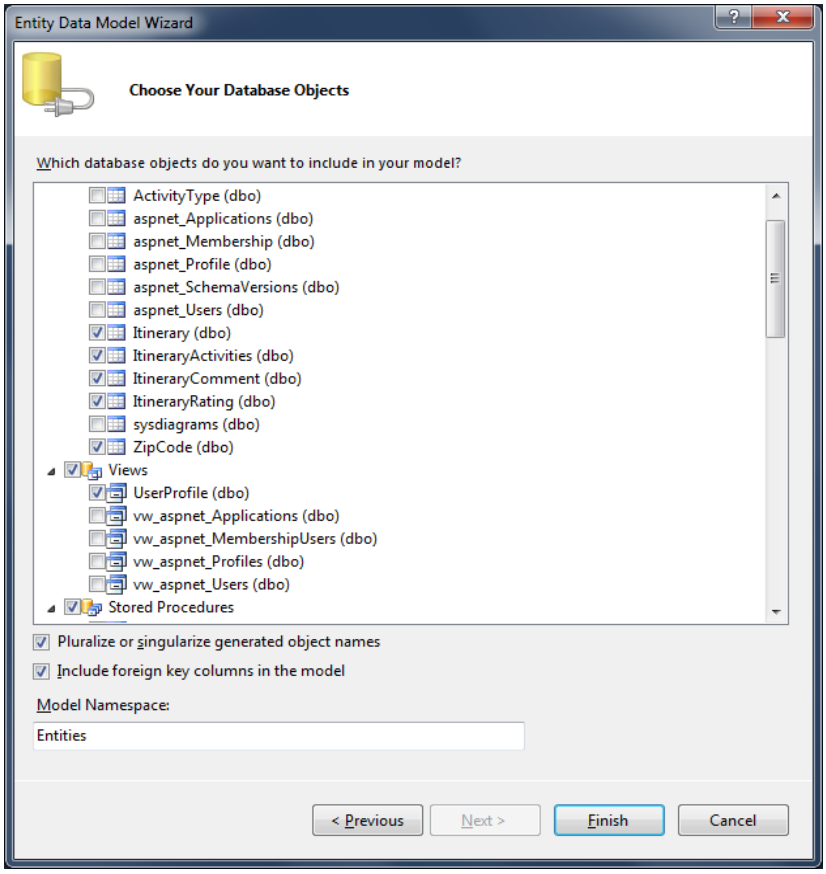


Рис. 8-8 Мастер EDM, выбор объектов базы данных

Щелкаем Finish (Готово), чтобы сформировать EDM.

> **Visual Studio 2008** В первой версии EF при создании модели из базы данных имена, ассоциированные с *EntityType* (Тип сущности), *EntitySet* (Множество сущностей) и *NavigationProperty* (Свойство навигации), часто были неверными, потому что они создавались на основании имени таблицы базы данных. Наверняка, нам не нужен экземпляр сущности с именем во множественном числе, *ItineraryActivities*, скорее всего, имя должно быть в единственном числе: *ItineraryActivity*. Кнопка-флажок *Pluralize or singularize generated object names* (Образовывать форму единственного или множественного числа для формируемых имен объектов), показанная на рис. 8-8, позволяет управлять тем, в какой форме будут создаваться имена объектов.

Корректировка сформированной модели данных

Теперь мы имеем модель, которая представляет собой множество сущностей, соответствующее используемой базе данных. Мастер сформировал все навигационные свойства, ассоциированные с внешними ключами базы данных.

Приложению PMN необходимо только навигационное свойство *ItineraryActivies* из таблицы Itinerary, поэтому все остальные навигационные свойства можно смело удалить. Также придется переименовать навигационное свойство *ItineraryActivities* в **Activities**. Обновленная модель представлена на рис. 8-9.

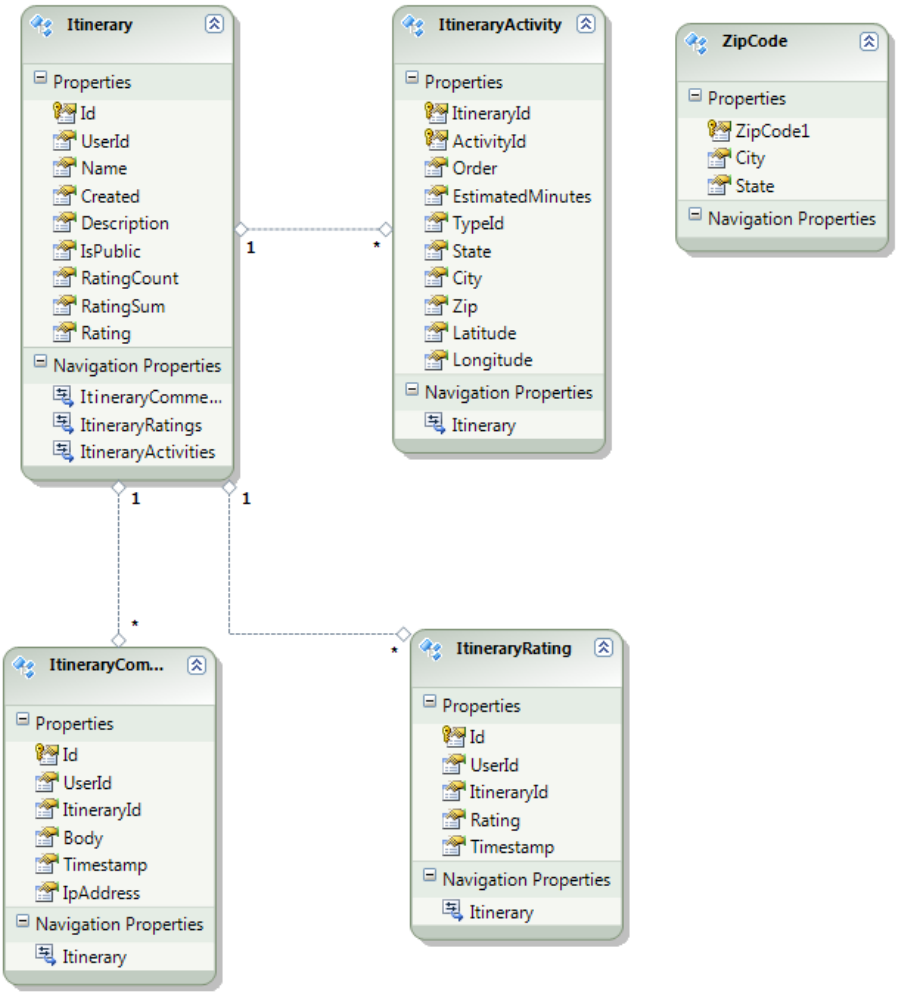


Рис. 8-9 Модель, импортированная из базы данных PlanMyNight

Можно заметить, что одному из свойств сущности ZipCode присвоено имя *ZipCode1*, потому что сама таблица уже названа ZipCode, а имя должно быть уникальным. Чтобы исправить это имя, щелкнем его двойным щелчком. Зададим имя **Code**, как показано на рис. 8-10.

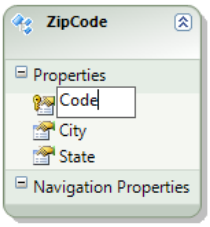


Рис. 8-10 Сущность ZipCode

Выполним сборку решения, нажав Ctrl+Shift+B. В окне вывода можно увидеть два сообщения о сформированной EDM. Первое можно сразу удалить, поскольку столбец Location (Местоположение) в PMN не нужен. Второе сообщение гласит:

> *Для таблицы/представления 'dbo.UserProfile' первичный ключ не задан и не может быть сформирован. Данная таблица/представление не включена в модель. Для использования этой сущности необходимо извлечь схему, добавить соответствующие ключи и раскомментировать ее.*

Взглянув на представление UserProfile можно заметить, что первичный ключ не задан явно, даже несмотря на то что значения столбца UserName (Имя пользователя) уникальны.

Необходимо вручную подкорректировать EDM, исправив сопоставление для представления UserProfile, что позволит выполнять доступ к данным UserProfile из приложения.

В обозревателе проекта щелкните правой кнопкой мыши файл PlanMyNight.edmx и выберите Open With (Открыть в). В диалоговом окне Open With выберите XML (Text) Editor ((Текстовый) редактор XML), как показано на рис. 8-11. Щелкните OK, чтобы открыть XML-файл, связанный с нашей моделью[1].

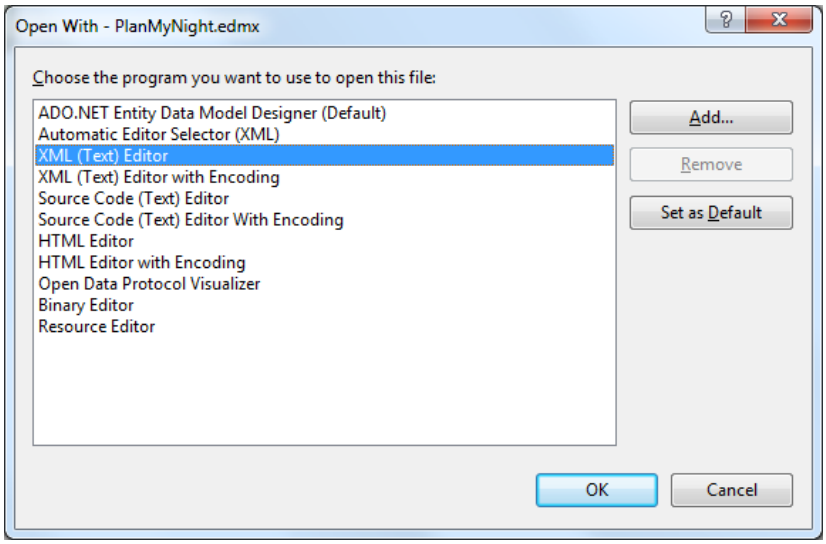


Рис. 8-11 Открываем PlanMyNight.emdx в редакторе XML

> **Примечание** На экран будет выведено сообщение о том, что файл PlanMyNight.edmx уже открыт. Щелкните Yes, чтобы закрыть его.

Механизм формирования кода закомментировал созданный код из-за отсутствия первичного ключа. Чтобы использовать представление UserProfile из дизайнера, необходимо раскомментировать тип сущности UserProfile и добавить в него тег Key (Ключ). Найдите UserProfile в файле. Раскомментируйте тип сущности, добавьте тег ключа, задайте для него имя **UserName** и сделайте свойство *UserName* не допускающим значение null. Обновленный тип сущности представлен в листинге 8-1.

Листинг 8-1 XML-описание типа сущности UserProfile

```
<EntityType Name="UserProfile">
    <Key>
        <PropertyRef Name="UserName"/>
    </Key>
    <Property Name="UserName" Type="uniqueidentifier" Nullable="false" />
    <Property Name="FullName" Type="varchar" MaxLength="500" />
    <Property Name="City" Type="varchar" MaxLength="500" />
    <Property Name="State" Type="varchar" MaxLength="500" />
    <Property Name="PreferredActivityTypeId" Type="int" />
</EntityType>
```

Если закрыть XML-файл и попытаться открыть дизайнер EDM, в нем будет выведена такая ошибка: «The Entity Data Model Designer is unable to display the file you requested. You can edit the model using the XML Editor[2]».

На панели Error List (Список ошибок) выводится следующее предупреждение, более подробно раскрывающее суть этой ошибки:

Error 11002: Entity type 'UserProfile' has no entity set.[3]

Необходимо задать множество сущностей для типа UserProfile, чтобы тип сущности мог проецироваться на схему хранилища. Для этого открываем файл PlanMyNight.edmx в редакторе XML. Вверху файла, прямо над множеством сущностей Itinerary, добавляем XML-код, представленный в листинге 8-2.

Листинг 8-2 XML-описание множества сущностей для UserProfile

```
<EntitySet Name="UserProfile" EntityType="Entities.Store.UserProfile"
store:Type="Views" store:Schema="dbo" store:Name="UserProfile">
    <DefiningQuery>
```

[1] Этот XML-файл содержит описание всей созданной модели (*прим. технического редактора*)

[2] Дизайнер Entity Data Model не может вывести на экран запрашиваемый файл. Скорректируйте модель в редакторе XML (*прим. переводчика*).

[3] Для типа сущности 'UserProfile' не задано множество сущностей (*прим. переводчика*).

```
        SELECT
      [UserProfile].[UserName] AS [UserName],
      [UserProfile].[FullName] AS [FullName],
             [UserProfile].[City] AS [City],
          [UserProfile].[State] AS [State],
          [UserProfile].[PreferredActivityTypeId] as [PreferredActivityTypeId]
          FROM [dbo].[UserProfile] AS [UserProfile]
    </DefiningQuery>
</EntitySet>
```

Сохраним XML-файл EDM и повторно откроем дизайнер EDM. На рис. 8-12 показано представление UserProfile в разделе Entities.Store браузера модели (Model Browser).

> **Подсказка** Model Browser можно открыть из меню View (Вид), щелкнув Other Windows (Другие окна) и выбрав Entity Data Model Browser.

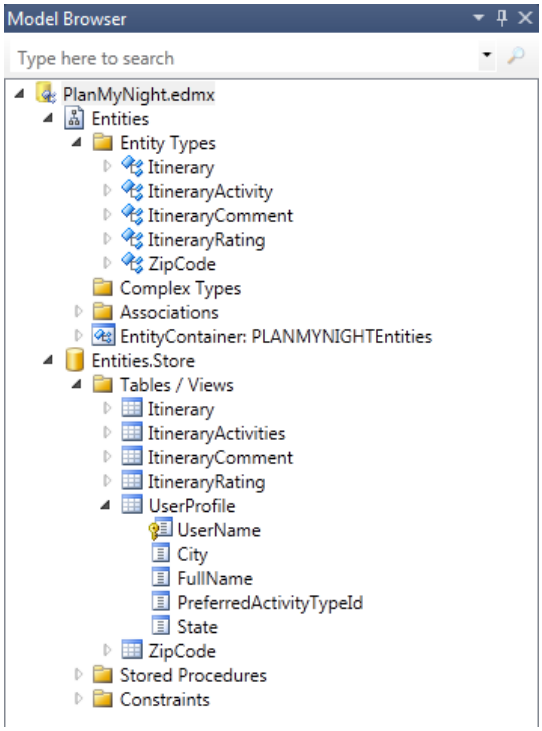


Рис. 8-12 Представление UserProfile в Model Browser

Теперь, когда представление доступно в метаданных хранилища, добавим сущность UserProfile и сопоставим ее с представлением UserProfile. Щелкаем правой кнопкой мыши фон дизайнера EDM, выбираем Add и затем Entity. На экран будет выведено диалоговое окно, показанное на рис. 8-13.

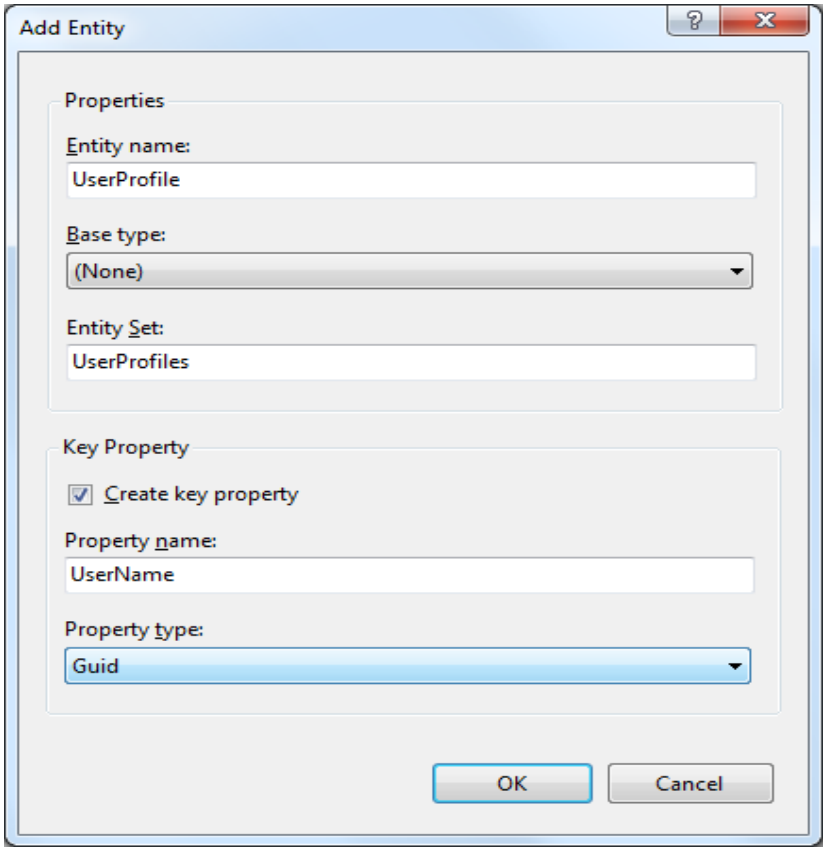


Рис. 8-13 Диалоговое окно Add Entity

Заполните диалоговое окно, как показано на рис. 8-13, и щелкните OK, чтобы создать сущность.

После этого понадобиться добавить остальные свойства: *City* (Город), *State* (Страна) и *PreferedActiviyTypeId* (Идентификатор предпочтительного типа мероприятий). Для этого щелкаем правой кнопкой мыши сущность UserProfile, выбираем Add и Scalar Property (Скалярное свойство). Как только свойство добавлено, задаем значения полей *Type* (Тип), *Max Length* (Максимальная длина) и *Unicode*. В табл. 8-1 представлены ожидаемые значения каждого из полей.

Таблица 8-1 Свойства сущности UserProfile

Name	Type	Max Length	Unicode
FullName	String	500	False
City	String	500	False
State	String	500	False
PreferedActivityTypeId	Int32	Недоступно	Недоступно

Теперь, когда сущность UserProfile создана, ее требуется сопоставить с представлением UserProfile. Щелкаем правой кнопкой мыши сущность UserProfile и выбираем Table Mapping (Сопоставление таблиц), как показано на рис. 8-14.

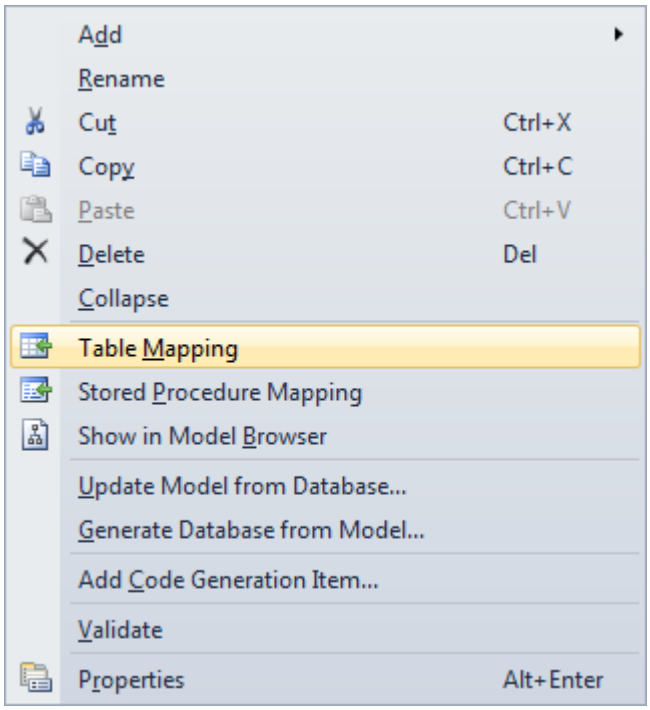


Рис. 8-14 Пункт меню Table Mapping

Затем выбираем представление UserProfile из раскрывающегося списка, как показано на рис. 8-15. Убедитесь, что все столбцы и свойства сущности сопоставлены правильно. Теперь представление UserProfile нашего хранилища доступно из кода через сущность UserProfile.

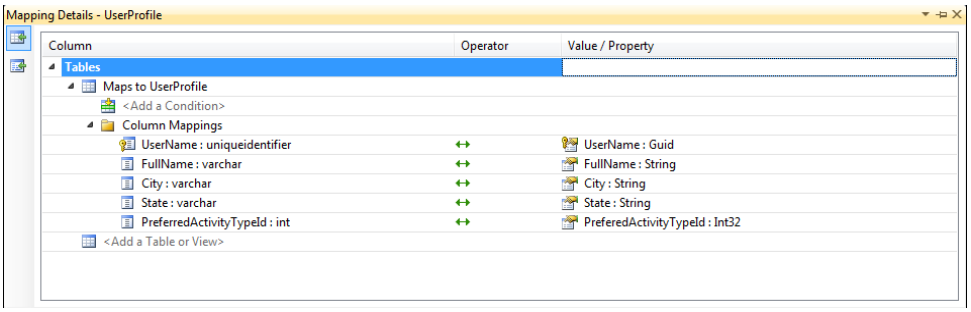


Рис. 8-15 Детали сопоставления UserProfile

Хранимая процедура и импортированные функции

Мастер Entity Data Model Wizard (Мастер модели сущность-данные) создал в модели хранилища запись для хранимой процедуры *RetrieveItinerariesWithinArea*, которая была выбрана в мастере в заключительном шаге. Необходимо создать соответствующую запись в концептуальной модели, добавив Function Import (Импортированная функция).

В разделе Entities.Store браузера модели откройте папку Stored Procedures (Хранимые процедуры). Щелкните правой кнопкой мыши *RetrieveItineraryWithinArea* и выберите Add Function Import… (Добавить импортированную функцию). Диалоговое окно Add Function Import представлено на рис. 8-16. Задайте возвращаемый тип, выбрав Entities, и элемент Itinerary в окне раскрывающегося списка. Щелкните OK.

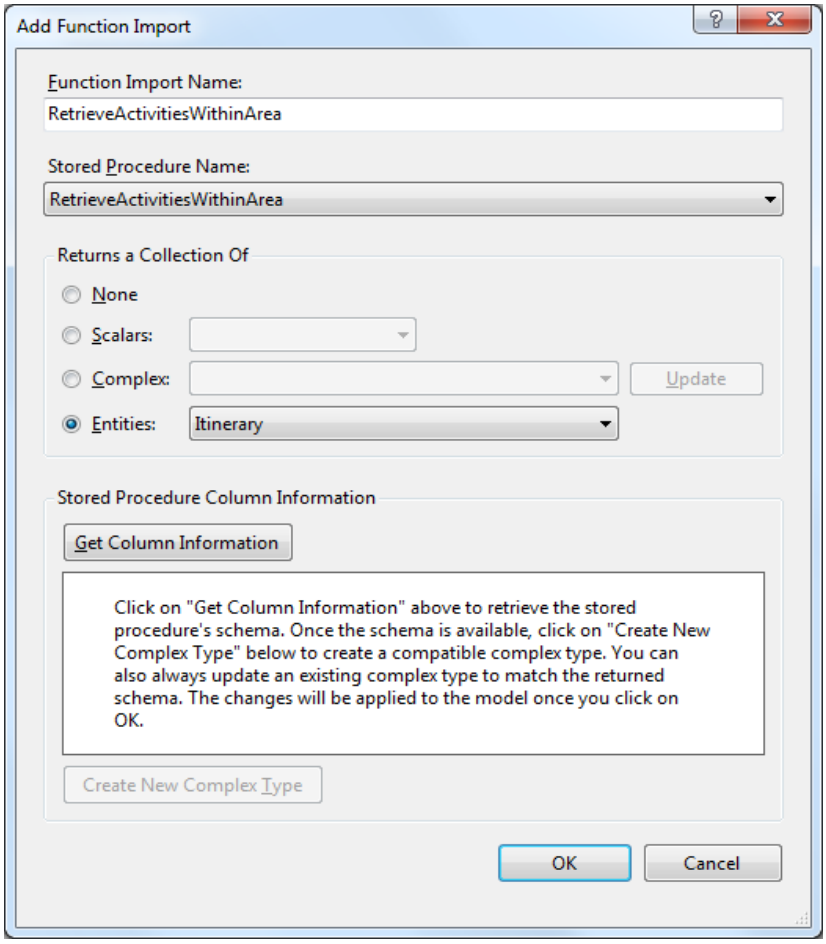


Рис. 8-16 Диалоговое окно Add Function Import

В браузер модели добавлена импортированная функция *RetrieveItinerariesWithinArea*, как показано на рис. 8-17.

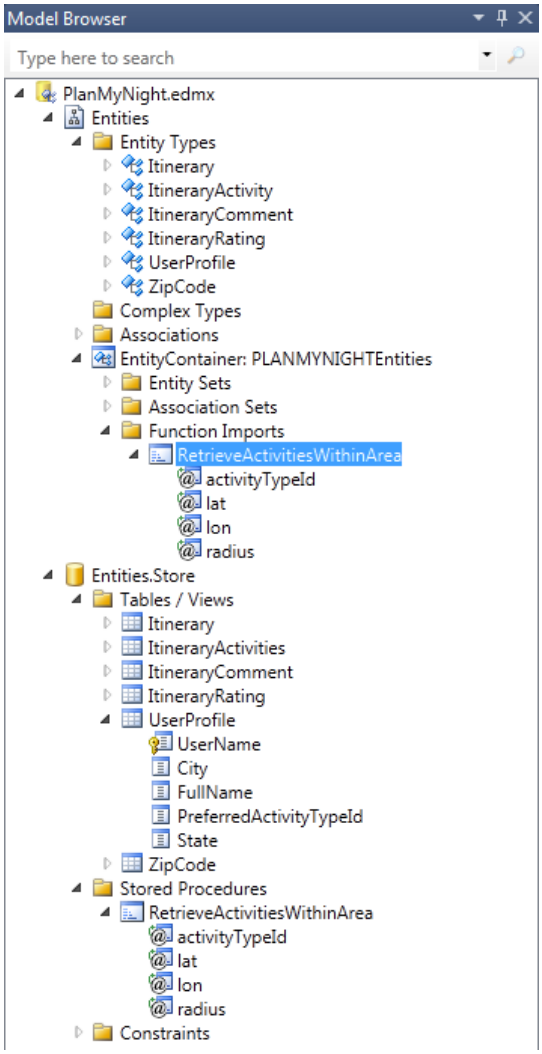


Рис. 8-17 Импортированные функции в браузере модели

Теперь можно проверить правильность EDM, щелкнув правой кнопкой мыши рабочую область дизайнера и выбрав Validate (Проверить). При этом не должно появиться никаких ошибок или предупреждений.

EF: сначала модель[1]

В предыдущем разделе мы увидели, как использовать дизайнер EF для создания модели путем импорта существующей базы данных. Дизайнер EF в Visual Studio 2010 также поддерживает возможность автоматического формирования файла Data Definition Language (DDL), который позволит создать базу данных из модели сущностей. В этом разделе мы используем новое решение, чтобы научиться автоматически формировать сценарий базы данных из модели.

Можно начать с пустой модели. Для этого в мастере Entity Data Model Wizard выбираем опцию Empty model (Пустая модель).

> **Примечание** Чтобы открыть мастер, щелкните правой кнопкой проект PlanMyNight.Data, выберите Add и затем New Item. Выберите элемент ADO.NET Entity Data Model.

[1] Модель-ориентированный подход (*прим. технического редактора*)

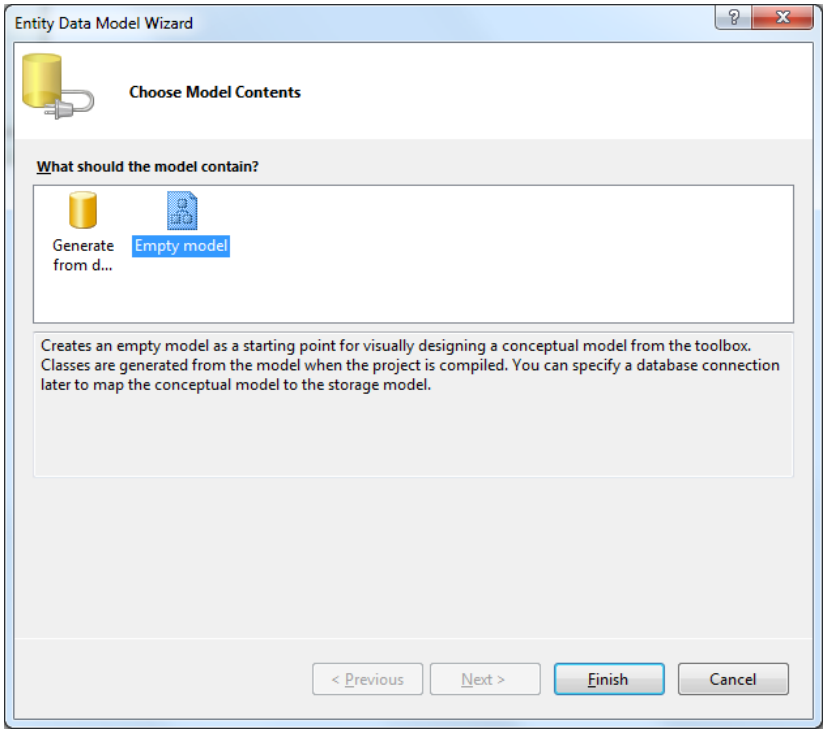


Рис. 8-18 Мастер EDM: Empty model

Открываем решение PMN по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 8\Code\ModelFirst, щелкнув двойным щелчком файл PlanMyNight.sln.

Проект PlanMyNight.Data этого решения уже содержит EDM-файл PlanMyNight.edmx с некоторыми уже созданными сущностями. Эти сущности соответствуют схеме данных, представленной на рис. 8-2.

Дизайнер Entity Model позволяет без труда добавлять сущности в модель данных. Добавим в нашу модель недостающую сущность ZipCode. Из панели инструментов перенесите в дизайнер элемент Entity, как показано на рис. 8-19. Присвойте этой сущности имя **ZipCode**, свойству *Id* – имя **Code** и задайте для него тип *String*.

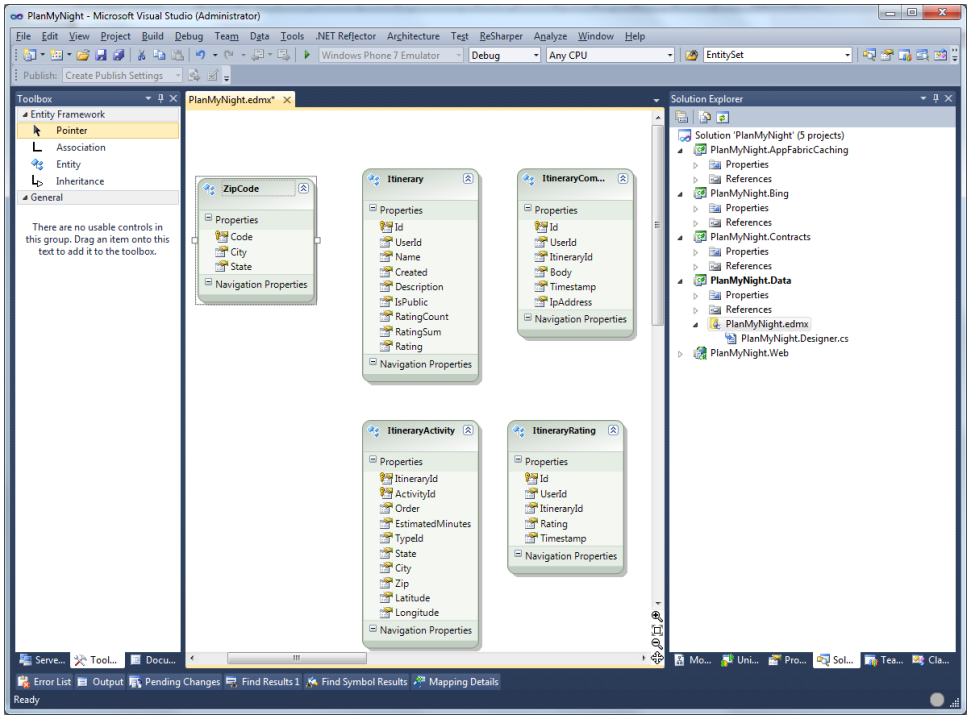


Рис. 8-19 Дизайнер модели сущностей

Теперь необходимо добавить в сущность свойства *City* и *State*. Щелкаем правой кнопкой мыши сущность ZipCode, выбираем Add и затем Scalar Property. Убеждаемся, что значения всех свойств соответствуют приведенным в табл. 8-2.

Таблица 8-2 Свойства сущности ZipCode

Name	Type	Fixed Length	Max length	Unicode
Code	*String*	False	5	False
City	*String*	False	150	False
State	*String*	False	150	False

Добавим отношения между ItineraryComment и сущностями Itinerary. Щелкаем правой кнопкой мыши поверхность дизайнера, выбираем Add и затем Association (Связь).

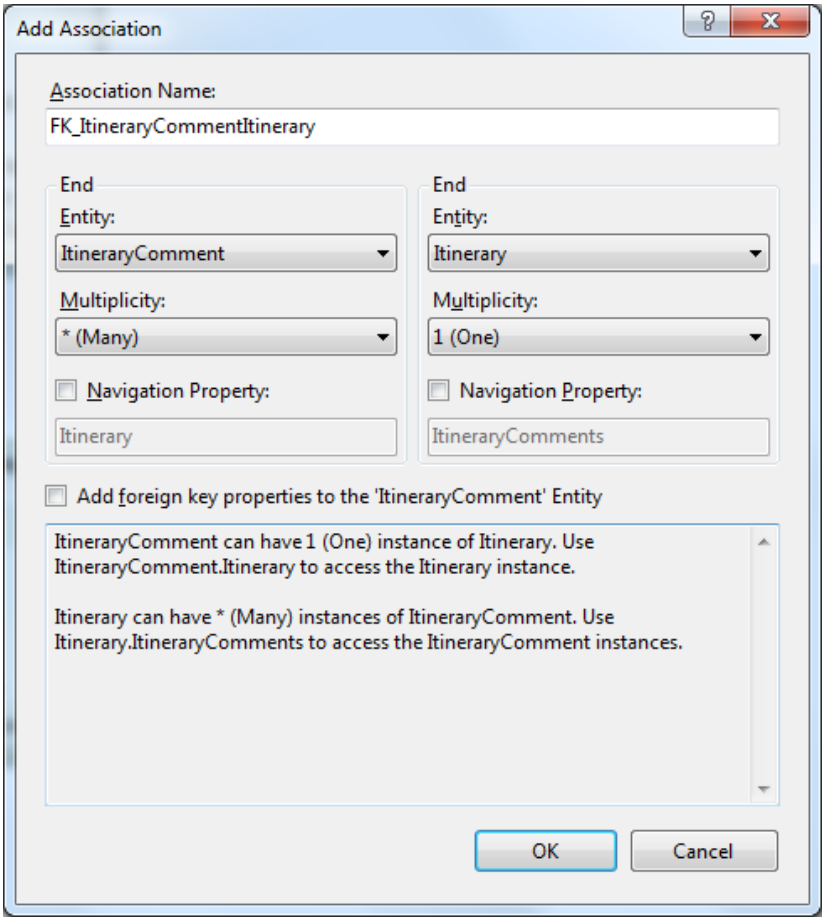


Рис. 8-20 Диалоговое окно Add Association для FK_ItineraryCommentItinerary

> **Visual Studio 2008** В версию Entity Framework .NET 4.0 включены Foreign Key Associations (Связи внешних ключей). Это позволяет задавать для сущностей внешние свойства. Foreign Key Associations теперь является типом по умолчанию для связи, но поддерживаемый в .NET 3.5 тип Independent Associations (Независимые связи) по-прежнему доступен.

Задайте имя связи **FK_ItineraryCommentItinerary**, выберите сущность и кратность для каждого конца связи, как показано на рис. 8-20. Как только связь создана, щелкните двойным щелчком строку связи, чтобы задать Referential Constraint (Справочное ограничение), как показано на рис. 8-21.

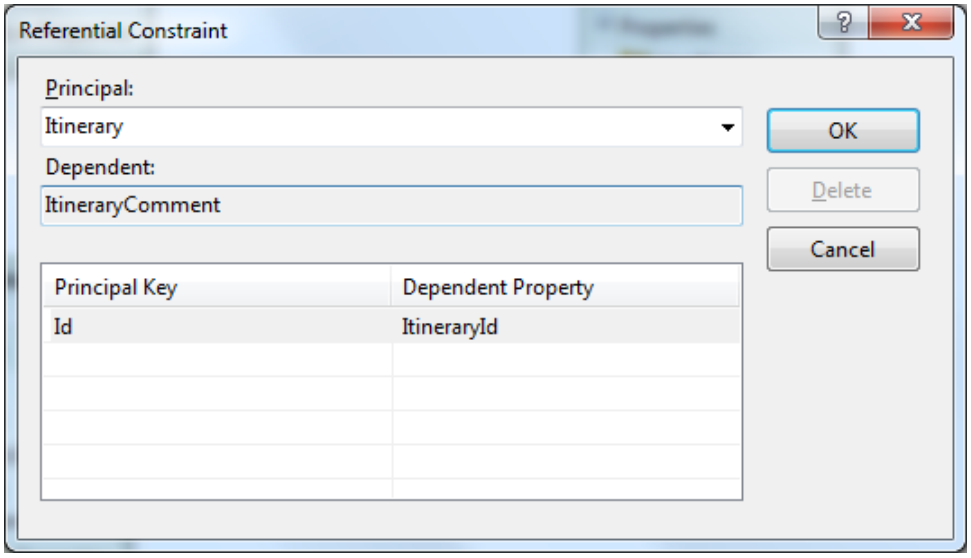


Рис. 8-21 Диалоговое окно Association Referential Constraint

Добавьте связь между сущностями ItineraryRating и Itinerary. Щелкните правой кнопкой фон дизайнера, выберите Add и затем Association. Задайте имя связи **FK_ItineraryItineraryRating**. После этого выберите сущность и кратность для каждого конца связи, как делали это в предыдущем случае, только в качестве первого конца связи задайте **ItineraryRating**. Щелкните двойным щелчком линию связи и задайте Referential Constraint, как показано на рис. 8-21. Обратите внимание, что в поле Dependent (Зависимый) отображается не ItineraryComment, а ItineraryRating. Создайте новую связь между сущностями ItineraryActivity и Itinerary. Также для связи *FK_ItineraryItineraryActivity* потребуется создать навигационное свойство и назвать его **Activities**, как показано на рис. 8-22. Когда связь создана, задайте для нее Referential Constraint, щелкнув двойным щелчком линию связи.

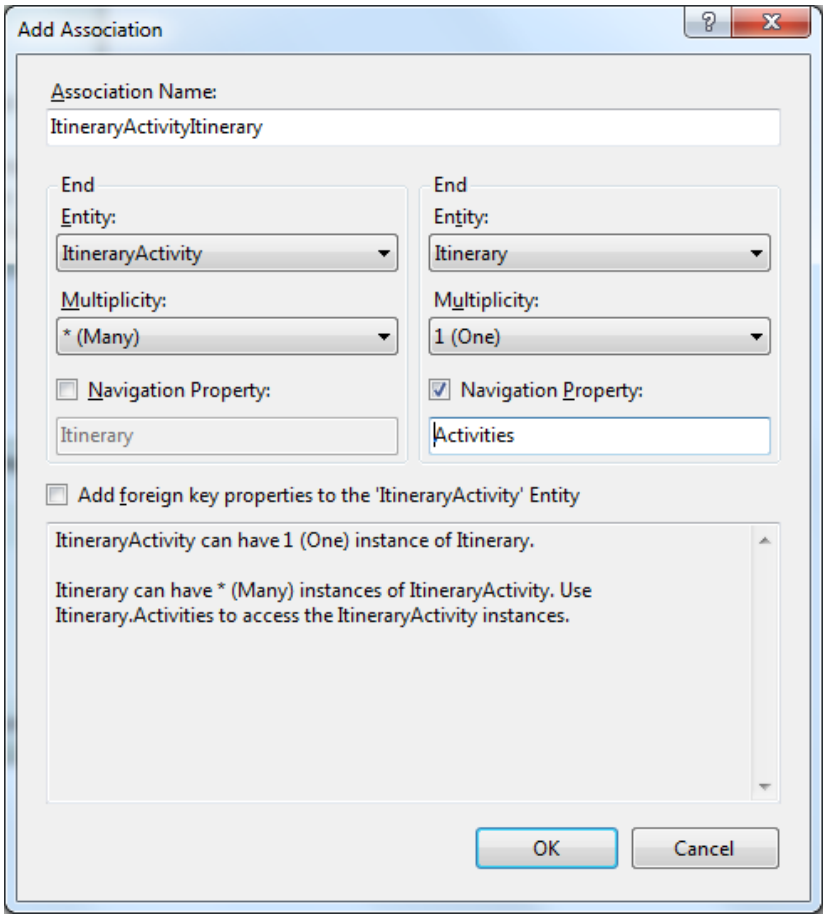


Рис. 8-22 Диалоговое окно Add Association для FK_ItineraryActivityItinerary

Автоматическое формирование сценария базы данных из модели

Модель данных готова, но с ней не ассоциировано никакое хранилище или сопоставление. Дизайнер EF предлагает возможность автоматического формирования сценария базы данных из модели.

Щелкните правой кнопкой мыши дизайнер и выберите пункт меню Generate Database From Model (Сформировать базу данных из модели), как показано на рис. 8-23.

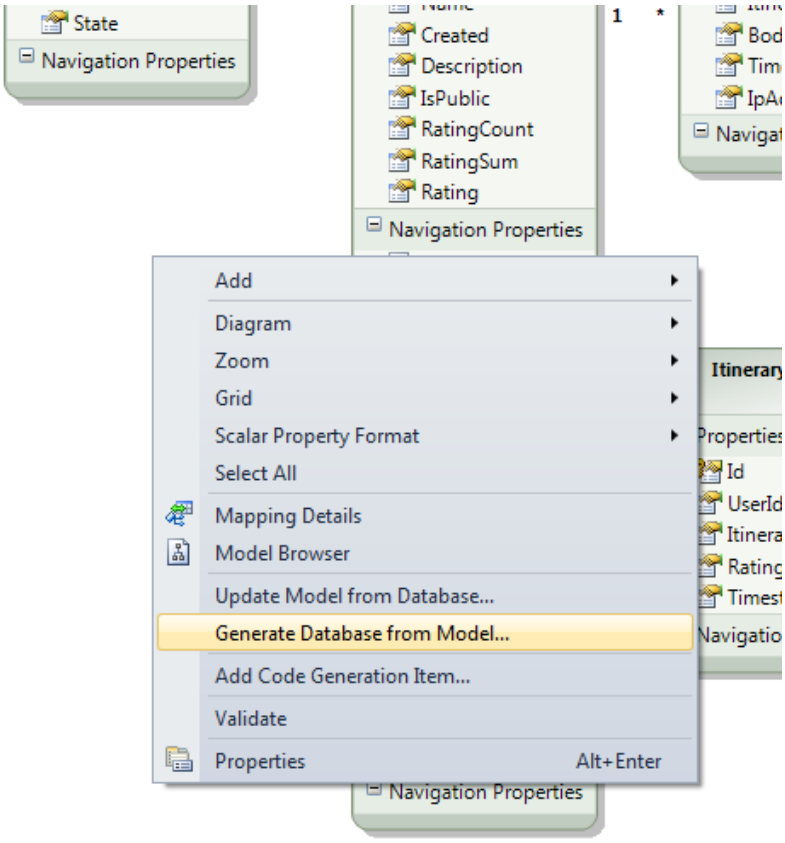


Рис. 8-23 Пункт меню Generate Database from Model

Мастер Generate Database Wizard требует подключения к данным. Этот мастер будет использовать данные подключения для трансляции типов модели в тип базы данных и для формирования DDL-сценария для этой базы данных.

Выберите New Connection. В диалоговом окне Choose Data Source выберите Microsoft SQL Server Database File и щелкните Continue. Выберите файл базы данных, находящийся по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 8\Code\ModelFirst\Data\PlanMyNight.mdf (рис. 8-24).

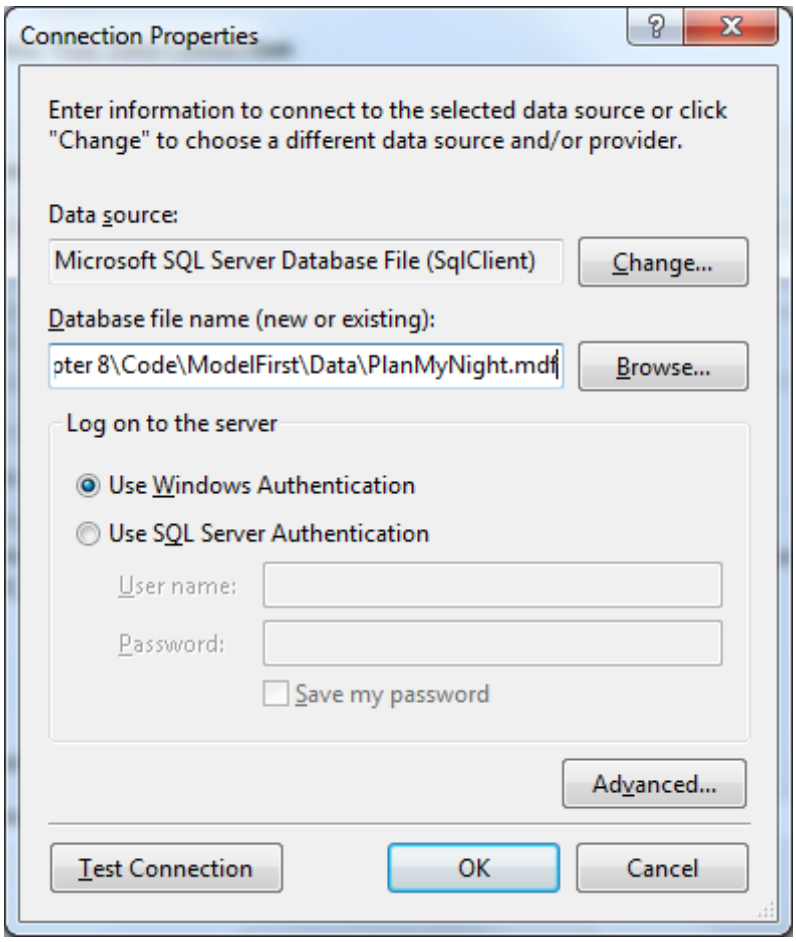


Рис. 8-24 Подключение базы данных для формирования сценария

Как только подключение настроено, щелкните Next, чтобы перейти к последнему окну мастера, как показано на рис. 8-24. По нажатию Finish сформированный файл T-SQL PlanMyNight.edmx.sql добавляется в проект. DDL-сценарий сформирует ограничения первичного и внешнего ключей для модели.

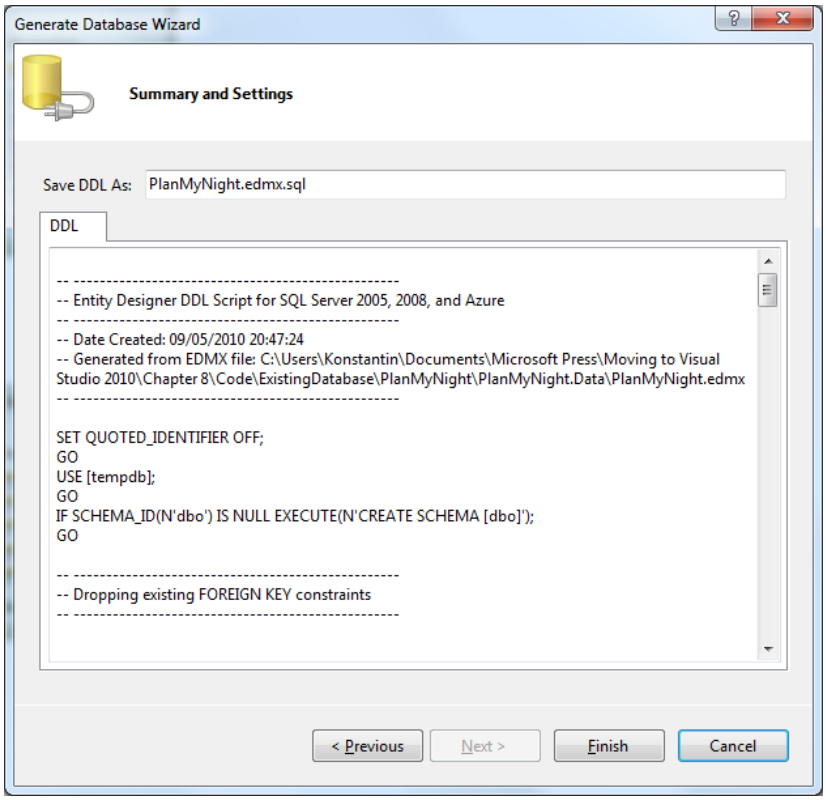


Рис. 8-25 Сформированный файл T-SQL

EDM также обновляется, чтобы вновь созданное хранилище гарантированно сопоставлялось с сущностями. Теперь сформированный DDL-сценарий может использоваться для добавления таблиц в базу данных, и мы получили слой доступа к данным, предоставляющий строго типизированные сущности, которые могут использоваться в приложении.

> **Важно** Для создания полной базы данных PMN потребовалось бы добавить оставшиеся таблицы, хранимые процедуры и триггеры, используемые приложением. Вместо того чтобы осуществлять все эти операции, вернемся к решению, каким оно было в конце раздела «EF: импорт существующей базы данных».

POCO-шаблоны

Для формирования кода сущностей дизайнер EDM использует шаблоны T4. До сих пор мы позволяли дизайнеру создавать сущности на базе шаблонов по умолчанию. Автоматически сформированный код можно увидеть в файле PlanMyNight.Designer.cs, ассоциированном с PlanMyNight.edmx. Созданные в нем сущности типа EntityObject и снабжены атрибутами, которые обеспечивают возможность EF управлять ими во время выполнения.

> **Примечание** T4 расшифровывается как *Text Template Transformation Toolkit* (Набор инструментов для преобразования текстовых шаблонов). Поддержка T4 в Visual Studio 2010 позволяет без труда создавать собственные шаблоны и формировать любые текстовые файлы (Веб, ресурсы или источники). Более подробные сведения об автоматическом формировании кода в Visual Studio 2010 можно найти в статье Code Generation and Text Template (Автоматическое формирование кода и текстовые шаблоны) по адресу *http://msdn.microsoft.com/en-us/library/bb126445(VS.100).aspx.*

EF также поддерживает типы сущностей POCO. POCO-классы – это простые объекты без атрибутов или базового класса инфраструктуры. (Листинг 8-3 в следующем разделе представляет POCO-класс для сущности ZipCode.) EF использует имена типов и свойства этих объектов для их сопоставления с моделью во время выполнения.

> **Примечание** POCO расшифровывается как *Plain-Old CLR Objects*.

Шаблон ADO.NET POCO Entity Generator

Повторно откроем файл %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 8\Code\ExistingDatabase\PlanMyNight.sln.

Выберем файл PlanMyNight.edmx, щелкнем правой кнопкой мыши дизайнер и выберем Add Code Generation Item (Добавить элемент формирования кода). При этом откроется диалоговое окно, показанное на рис. 8-26, которое предоставляет возможность выбрать необходимый шаблон. Выберем шаблон ADO.NET POCO Entity Generator[1] и назовем его **PlanMyNight.tt**. После этого щелкнем кнопку Add.

> **Примечание** Возможно, на экран будет выведено предупреждение об угрозе безопасности в случае выполнения этого текстового шаблона. Источник данного шаблона доверенный, поэтому просто щелкните OK, чтобы закрыть это диалоговое окно.

[1] По умолчанию данный шаблон не установлен. Если у Вас его нет, выберите страницу Online Templates (он-лайн шаблоны), а далее введите слово POCO в поле Search Online Templates (Поиск он-лайн шаблонов) и нажмите Enter, через некоторое время шаблон будет найден, его можно будет установить и использовать (*прим. технического редактора*)

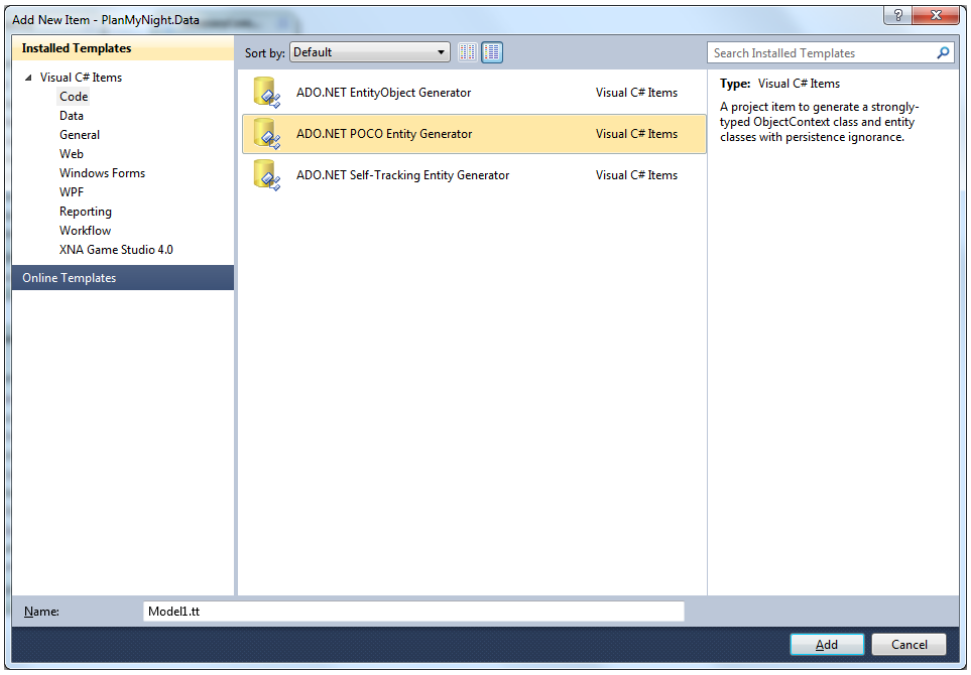


Рис. 8-26 Диалоговое окно Add New Item

В проект были добавлены два файла, PlanMyNight.tt и PlanMyNight.Context.tt (рис. 8-27). Эти файлы заменяют шаблон формирования кода по умолчанию, и код больше не формируется в файле PlanMyNight.Designer.cs.

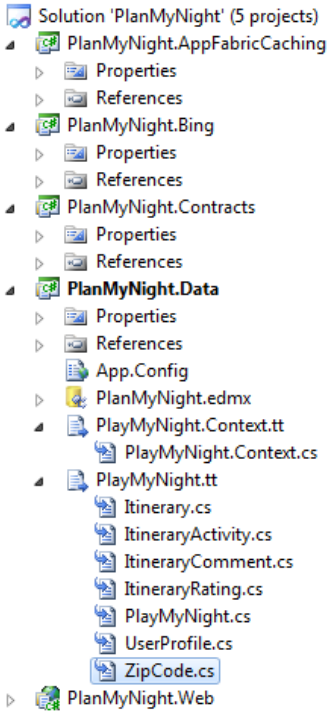


Рис. 8-27 Добавленные шаблоны

Шаблон PlanMyNight.tt создает по файлу класса для каждой сущности в модели. Листинг 8-3 представляет POCO-версию класса *ZipCode*.

Листинг 8-3 POCO-версия класса ZipCode

```
namespace Microsoft.Samples.PlanMyNight.Data
{
    public partial class ZipCode
    {
        #region Primitive Properties
        public virtual string Code
        {
            get;
```

```
            set;
        }
        public virtual string City
        {
            get;
            set;
        }
        public virtual string State
        {
            get;
            set;
        }
        #endregion
    }
}
```

Другой файл, PlanMyNight.Context.cs, формирует объект ObjectContext (Контекст объекта) для модели PlanMyNight.edmx. Этот объект мы будем использовать для взаимодействия с базой данных.

> **Подсказка** Шаблоны POCO будут автоматически обновлять сформированные классы соответственно внесенным в модель изменениям при сохранении файла .edmx.

Перенос классов сущностей в проект контрактов

Архитектура приложения PMN спроектирована таким образом, что слой представления гарантированно безразличен к методу хранения. Это реализовано путем перемещения контрактов и классов сущностей в сборку, которая не имеет ссылки на хранилище.

> **В Visual Studio 2008** существовала возможность расширения обработки XSD с помощью инструментов формирования кода, но это было связано с определенными сложностями и требовало обслуживания этих инструментов. EF использует шаблоны T4 для формирования и схемы базы данных, и кода. Эти шаблоны можно без труда настроить соответственно конкретным требованиям.

Шаблоны POCO ADO.NET выносят формирование классов сущностей в отдельный шаблон, что дает нам возможность с легкостью переносить эти сущности в другой проект.

Перенесем файл PlanMyNight.tt в проект PlanMyNight.Contracts. Щелкнув правой кнопкой мыши файл PlanMyNight.tt, выбираем Cut (Вырезать). Щелкаем правой кнопкой мыши папку Entities проекта PlanMyNight.Contracts и выбираем Paste (Вставить). Результат представлен на рис. 8-28.

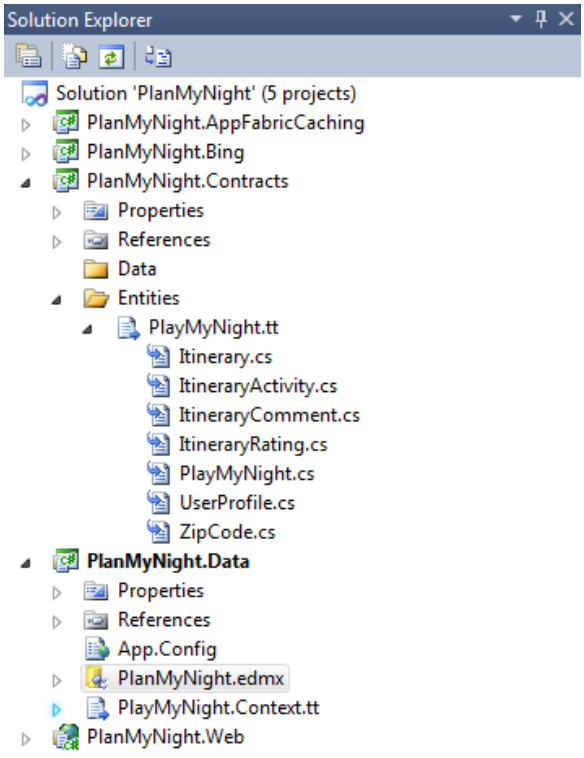


Рис. 8-28 Шаблон POCO, перенесенный в проект Contracts

Для формирования кода типа сущности шаблон PlanMyNight.tt использует метаданные модели EDM. Необходимо скорректировать относительный путь, используемый шаблоном для доступа к файлу EDMX.

Откройте шаблон PlanMyNight.tt и найдите следующую строку:

```
string inputFile = @"PlanMyNight.edmx";
```

Исправьте путь к файлу, чтобы он указывал на файл PlanMyNight.edmx в проекте PlanMyNight.Data:

```
string inputFile = @"..\..\PlanMyNight.Data\PlanMyNight.edmx";
```

Классы сущностей будут сформированы повторно, как только вы сохраните шаблон.

Также необходимо обновить шаблон PlanMyNight.Context.tt в проекте PlanMyNight.Contracts, поскольку теперь классы сущностей находятся в пространстве имен *Microsoft.Samples.PlanMyNight.Entities*, а не в *Microsoft.Samples.PlanMyNight.Data*. Откройте файл PlanMyNight.Context.tt и включите новое пространство имен в раздел директив *using*.

```
using System;
using System.Data.Objects;
using System.Data.EntityClient;
using Microsoft.Samples.PlanMyNight.Entities;
```

Нажав Ctrl+Shift+B, выполните сборку решения. Теперь компиляция проекта должна пройти успешно.

Сведение воедино

Универсальный код для взаимодействия с базой данных SQL создан, так что все готово для реализации специальной функциональности приложения PMN. В следующих разделах мы поэтапно рассмотрим этот процесс, вкратце остановимся на получении данных с сервисов Bing Maps и пройдем небольшой вводный курс в возможность кэширования AppFabric Microsoft Windows Server, используемую в PMN.

Чтобы свести все это воедино, требуется довольно большой объем вспомогательного кода. Упростить процесс поможет использование обновленного решения, в котором уже написана большая часть кода для контрактов и сущностей, а также для работы с сервисами Bing Maps. Решение также будет включать проект PlanMyNight.Data.Test для проверки кода проекта PlanMyNight.Data.

> **Примечание** Тестирование в Visual Studio 2010 будет рассмотрено в главе 10.

Получение данных из базы данных

В начале этой главы нами было принято решение сгруппировать операции над сущностью Itinerary в интерфейс хранилища *IItinerariesRepository*. Вот некоторые из этих операций:

- Поиск плана мероприятий по мероприятию
- Поиск плана мероприятий по почтовому индексу
- Поиск плана мероприятий по радиусу
- Добавление нового плана мероприятий

Посмотрим на соответствующие методы интерфейса *IItinerariesRepository*:

- *SearchByActivity* обеспечит поиск планов мероприятий по мероприятию и возвращение страницы данных.
- *SearchByZipCode* обеспечит поиск планов мероприятий по почтовому индексу и возвращение страницы данных.
- *SearchByRadius* обеспечит поиск планов мероприятий из определенного пункта и возвращение страницы данных.
- *Add* обеспечит добавление плана мероприятий в базу данных.

Откроем решение PMN, располагающееся по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 8\Code\Final, щелкнув двойным щелчком файл PlanMyNight.sln.

Выберем проект PlanMyNight.Data и откроем файл ItinerariesRepository.cs. Это реализация интерфейса *IItinerariesRepository*. Используя сформированный ранее объект контекста для работы с базой данных PlanMyNightEntities, мы можем создать запросы LINQ к модели, и EF транслирует их в обычный T-SQL, который будет применяться для работы с базой данных.

Перейдем к описанию *SearchByActivity*. Этот метод должен возвращать набор планов мероприятий, обозначенных как общедоступные (флаг *IsPublic* имеет значение true), и *activId* одного из мероприятий которых соответствует переданному в аргументе функции значению. Полученные в результате планы мероприятий должны быть сортированы по полю рейтинга.

SearchByActivity реализуется с применением стандартных операторов LINQ, как показано в листинге 8-4. Добавим выделенный код в тело метода *SearchByActivity*.

Листинг 8-4 Реализация SearchByActivity

```
public PagingResult<Itinerary> SearchByActivity(string activityId, int pageSize,
int pageNumber)
{
    using (var ctx = new PlanMyNightEntities())
    {
        ctx.ContextOptions.ProxyCreationEnabled = false;

        var query = from itinerary in ctx.Itineraries.Include("Activities")
                    where itinerary.Activities.Any(t => t.ActivityId ==
                    activityId) && itinerary.IsPublic
                    orderby itinerary.Rating
                    select itinerary;

        return PageResults(query, pageNumber, pageSize);
    }
}
```

Примечание Разбиение результатов на страницы реализовано в методе *PageResults*.

```
private static PagingResult<Itinerary> PageResults(IQueryable<Itinerary> query,
int page, int pageSize)
{
int rowCount = rowCount = query.Count();
if (pageSize > 0)
{
query = query.Skip((page - 1) * pageSize)
.Take(pageSize);
}
var result = new PagingResult<Itinerary>(query.ToArray())
{
PageSize = pageSize,
CurrentPage = page,
TotalItems = rowCount
};
return result;
}
```

В этот метод передается *IQueryable<Itinerary>*, что позволяет добавить разбиение на страницы в состав базового запроса. Передача *IQueryable* вместо *IEnumerable* гарантирует, что T-SQL для запроса к хранилищу будет формироваться только при вызове *query.ToArray()*.

Метод *SearchByZipCode* аналогичен методу *SearchByActivity*, но также добавляет фильтр по почтовому индексу места выполнения действия. Опять же, поддержка LINQ упрощает реализацию, как показано в листинге 8-5. Добавим выделенный код в тело метода *SearchByZipCode*.

Листинг 8-5 Реализация SearchByZipCode

```
public PagingResult<Itinerary> SearchByZipCode(int activityTypeId, string zip, int
pageSize, int pageNumber)
{
    using (var ctx = new PlanMyNightEntities())
    {
        ctx.ContextOptions.ProxyCreationEnabled = false;

        var query = from itinerary in ctx.Itineraries.Include("Activities")
                    where itinerary.Activities.Any(t => t.TypeId == activityTypeId
&& t.Zip == zip)
                            && itinerary.IsPublic
                    orderby itinerary.Rating
```

```
                    select itinerary;

        return PageResults(query, pageNumber, pageSize);
    }
}
```

Функция *SearchByRadius* вызывает импортированную функцию *RetrieveItinerariesWithinArea*, которая была сопоставлена с хранимой процедурой. После этого загружаются действия для каждого найденного маршрута. Вы можете скопировать выделенный код листинга 8-6 в тело метода *SearchByRadius* в файле ItinerariesRepository.cs.

Листинг 8-6 Реализация SearchByRadius

```
public PagingResult<Itinerary> SearchByRadius(int activityTypeId, double longitude,
double latitude, double radius, int pageSize, int pageNumber)
{
    using (var ctx = new PlanMyNightEntities())
    {
        ctx.ContextOptions.ProxyCreationEnabled = false;

        // Хранимая процедура с выходным параметром
        var totalOutput = new ObjectParameter("total", typeof(int));
        var items = ctx.RetrieveItinerariesWithinArea(activityTypeId, latitude,
longitude, radius, pageSize, pageNumber, totalOutput).ToArray();

        foreach (var item in items)
        {
            item.Activities.AddRange(this.Retrieve(item.Id).Activities);
        }

        int total = totalOutput.Value == DBNull.Value ? 0 : (int)totalOutput.Value;

        return new PagingResult<Itinerary>(items)
        {
            TotalItems = total,
            PageSize = pageSize,
            CurrentPage = pageNumber
        };
    }
}
```

С помощью метода *Add* планы мероприятий добавляются в хранилище данных. Реализация этой функциональности упростилась, поскольку наш контракт и объект контекста используют один и тот же объект сущности. Вставьте выделенный код листинга 8-7 в тело метода *Add*.

Листинг 8-7 Реализация Add

```
public void Add(Itinerary itinerary)
{
    using (var ctx = new PlanMyNightEntities())
    {
        ctx.Itineraries.AddObject(itinerary);
        ctx.SaveChanges();
    }
}
```

Готово! Мы завершили реализацию *ItinerariesRepository*, применив объект контекста, сформированный дизайнером EF. Выполните все тесты решения, нажав CTRL+R, A. Все тесты, касающиеся *ItinerariesRepository*, должны быть пройдены успешно.

Разработка многопоточных приложений

С развитием многопроцессорной и многоядерной обработки данных все большее и большее значение приобретает обеспечение разработчикам возможностей для создания многопоточных приложений. Visual Studio 2010 и .NET Framework 4.0 предлагают новые пути реализации многопоточности в приложениях. Библиотека Task Parallel Library (TPL)[1] стала частью библиотеки Base Class Library (BCL)[2] для .NET Framework. Это означает, что теперь любое .NET-приложение может использовать TPL без всяких ссылок на сборки.

[1] Библиотека для параллельного выполнения задач (*прим. переводчика*).

[2] Библиотека базовых классов (*прим. переводчика*).

Для каждого ItineraryActivity PMN сохраняет в базу данных только идентификатор действия в Bing (Bing Activity Id). Когда приходит время извлечь весь объект Bing Activity (Действие в Bing), для заполнения сущности Bing Activity данными Веб-сервиса Bing Maps используется функция, которая выбирает все ItineraryActivity текущего Itinerary.

Один из способов выполнения этой операции – последовательное обращение к сервису для каждого действия Itinerary, как показано в листинге 8-8. Эта функция, прежде чем выполнять последующий вызов *RetrieveActivity* (Извлечь мероприятие), будет ожидать завершения предыдущего вызова, что делает ее выполнение линейным по времени.

Листинг 8-8 Последовательное извлечение мероприятий

```
public void PopulateItineraryActivities(Itinerary itinerary)
{
    foreach (var item in itinerary.Activities.Where(i => i.Activity == null))
    {
        item.Activity = this.RetrieveActivity(item.ActivityId);
    }
}
```

В прошлом для организации многопоточной обработки этой задачи приходилось использовать потоки и выполнять огромный объем работы. Теперь, с появлением TPL, для этого достаточно использовать статический метод Parallel.ForEach, который позаботиться обо всем, что связано с многопоточной обработкой, как показано в листинге 8-9.

Листинг 8-9 Параллельное извлечение мероприятий

```
public void PopulateItineraryActivities(Itinerary itinerary)
{
    Parallel.ForEach(itinerary.Activities.Where(i => i.Activity == null),
        item =>
    {
        item.Activity = this.RetrieveActivity(item.ActivityId);
    });
}
```

Дополнительные сведения .NET Framework 4.0 теперь включает библиотеки Parallel LINQ (в System.Core.dll). PLINQ представляет расширение .AsParallel для реализации многопоточной обработки в запросах LINQ. Благодаря расширениям .AsOrdered не составляет труда инициировать обработку источника данных, как будто данные были изначально отсортированы. Также в пространство имен System.Collections.Concurrent добавлены новые потокобезопасные коллекции. Более подробные сведения об этих новых возможностях можно найти в разделе Parallel Computing (Многопоточная обработка данных) на сайте MSDN по адресу http://msdn.microsoft.com/en-us/concurrency/default.aspx.

Кэширование AppFabric

PMN – это управляемое данными приложение, которое получает данные из базы данных приложения и от Веб-сервисов Bing Maps. Одна из возможных сложностей при создании Веб-приложения – реализация поддержки большого числа пользователей с обеспечением необходимой производительности и времени отклика. Обращения к хранилищу данных и к сервисам для поиска действий могут значительно повысить использование ресурсов сервера для элементов, совместно используемых множеством пользователей. Например, многие пользователи имеют доступ к общедоступным планам мероприятий. Просмотр этих планов мероприятий обусловит многочисленные обращения к одним и тем же элементам в базе данных. Реализация кэширования на Веб-уровне приведет к сокращению использования ресурсов хранилища данных и снизит задержку при выполнении повторных обращений к Веб-сервисам Bing Maps. На рис. 8-29 представлена архитектура приложения, реализующего кэширование на уровне Веб-сервера.

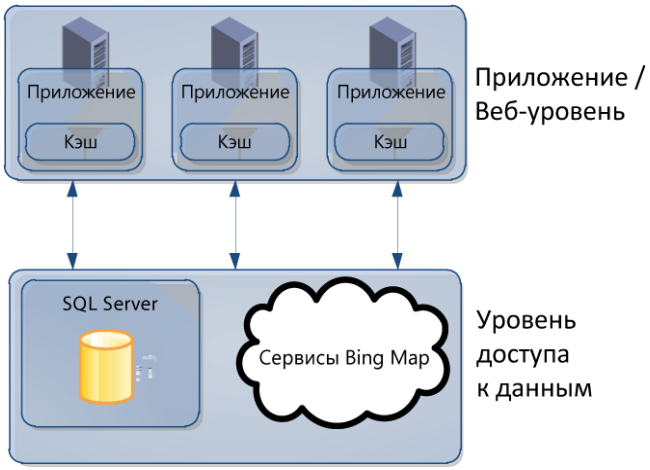


Рис. 8-29 Типовая архитектура Веб-приложения

Применение этого подхода сократит нагрузку на слой доступа к данным, но кэширование по-прежнему связано с конкретным сервером, обслуживающим запрос. Каждый сервер Веб-уровня будет иметь собственный кэш, и по-прежнему существует вероятность неравномерного распределения нагрузки между этими серверами.

Кэширование AppFabric Windows Server предлагает платформу распределенного кэширования в памяти. Клиентская библиотека AppFabric обеспечивает приложению возможность доступа к кэшу как к унифицированному событию просмотра, если кэш распределен на несколько компьютеров, как показано на рис. 8-30. API предоставляет простые методы *get* и *set*, что позволяет без труда извлекать и сохранять любые сериализуемые объекты общеязыковой среды выполнения (CLR). С кэшем AppFabric вы можете добавлять компьютер для размещения кэша по мере надобности, т.е. масштабирование становится прозрачным для клиента. Другое преимущество – кэш может также разделять копии данных между кластерами, обеспечивая высокую доступность данных даже в случае сбоев.

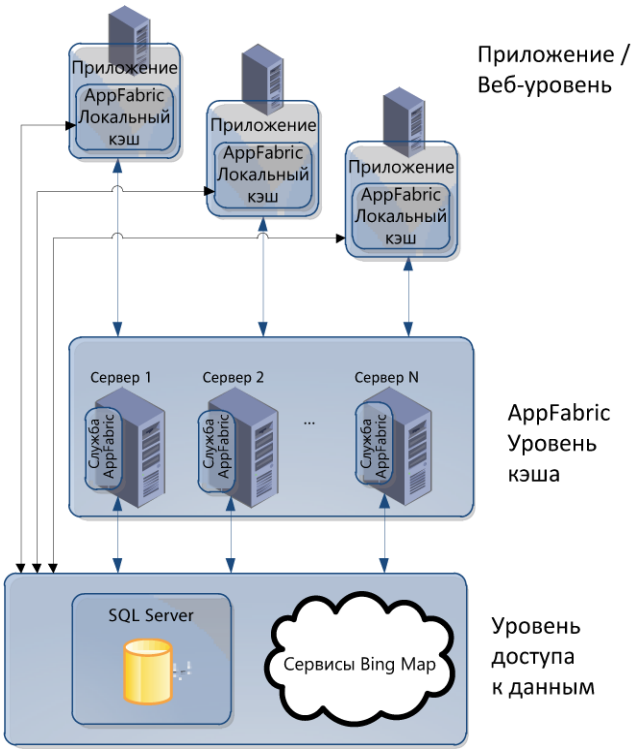


Рис. 8-30 Веб-приложение, использующее кэширование AppFabric Windows Server

> *Дополнительные сведения Кэширование AppFabric Windows Server предлагается как набор расширений .NET Framework 4.0. Подробные сведения о том, как найти, установить и настроить Windows Server AppFabric, можно найти на сайте Windows Server AppFabric по адресу* http://msdn.microsoft.com/en-us/windowsserver/ee695849.aspx.

> *Дополнительные сведения PMN может быть конфигурировано на использование либо кэширования ASP.NET, либо кэширования AppFabric Windows Server. Подробное пошаговое руководство с описанием того, как добавить кэширование AppFabric Windows Server в PMN предлагается в разделе* PMN: Adding

Caching using Velocity (Добавление кэширования с использованием Velocity) по адресу http://channel9.msdn.com/learn/courses/VS2010/ASPNET/EnhancingAspNetMvcPlanMyNight/Exercise-1-Adding-Caching-using-Velocity/.

Заключение

В данной главе мы рассмотрели ряд новых возможностей Visual Studio 2010 для структурирования слоя доступа к данным приложения Plan My Night и использовали Entity Framework версию 4.0 для организации доступа к базе данных. Также были представлены автоматическое формирование сущностей с помощью POCO-шаблонов ADO.NET Entity Framework и расширения кэширования AppFabric Windows Server.

В следующей главе мы займемся созданием замечательных Веб-приложений с помощью инфраструктуры ASP.NET MVC и Managed Extensibility Framework (Инфраструктура расширения возможностей).

Глава 9

От 2008 к 2010: проектирование внешнего вида и поведения

В данной главе рассматривается

- Создание контроллера ASP.NET MVC, взаимодействующего с моделью данных
- Создание представления ASP.NET MVC для отображения данных контроллера и проверки пользовательского ввода
- Расширение приложения внешним подключаемым модулем с помощью Managed Extensibility Framework

За годы, прошедшие с момента выхода ASP.NET 1.0, разработка Веб-приложений в Microsoft Visual Studio, безусловно, существенно улучшилась. В Visual Studio 2008 появилась официальная поддержка Веб-страниц с поддержкой AJAX, Language Integrated Query (LINQ)[34] и многие другие дополнения, благодаря которым разработчики получили возможность создавать эффективные простые в обслуживании приложения.

Жажда улучшений, призванных помочь разработчикам в создании высококлассных приложений, не иссякла и в Visual Studio 2010. В этой главе будут рассмотрены некоторые новые возможности на примере расширения функциональности используемого нами для примера приложения Plan My Night.

> **Примечание** Приложение-пример – это проект ASP.NET MVC 2, но в Visual Studio 2010 у Веб-разработчика есть выбор использовать либо новую форму приложения ASP.NET, либо более традиционную ASP.NET (которую в сообществе разработчиков *для отличия называют Веб-формами* (Web Forms)). ASP.NET 4.0 была существенно дополнена, чтобы облегчить жизнь разработчиков и при этом обеспечить крайне эффективный подход к созданию Веб-приложений.

В данной главе будет использоваться модифицированная форма решения приложения-примера. Если сопроводительные материалы данной книги установлены в каталог по умолчанию, интересующее нас решение находится в Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 9\ в папке UserInterface-Start.

Введение в проект PlanMyNight.Web

> **Примечание** Инфраструктура ASP.NET MVC 1.0 Framework предлагается как расширение Visual Studio 2008. Однако данная глава написана, исходя из предположения о том, что у пользователя имеется стандартная установка Visual Studio 2008, которая поддерживает только проекты ASP.NET Web Forms 3.5.

Пользовательский интерфейс Plan My Night в Visual Studio 2010 реализован как ASP.NET MVC-приложение, компоновка которого отличается от того, к чему, возможно, привыкли разработчики, создавая приложения Веб-форм ASP.NET в Visual Studio 2008. Некоторые элементы проекта (как видно на рис. 9-1) будут знакомыми (такие как Global.asax), но все остальные абсолютно новые. Некоторые из них являются обязательными для инфраструктуры ASP.NET MVC.

[34] Язык интегрированных запросов (*прим. переводчика*).

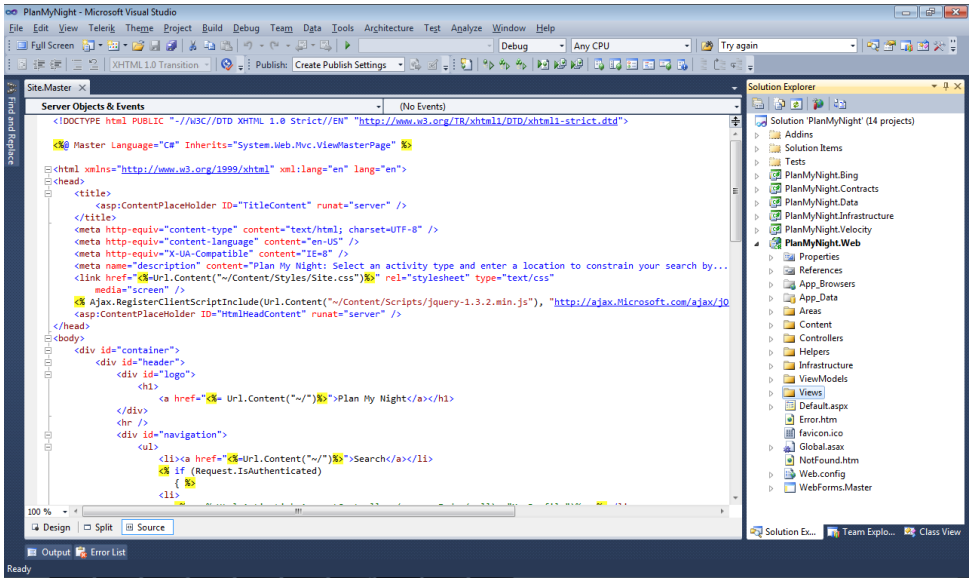


Рис. 9-1 Внешний вид проекта PlanMyNight.Web

Обязательные элементы ASP.NET MVC:

- **Areas (Области)** Эта папка используется инфраструктурой ASP.NET MVC для организации больших Веб-приложений в небольшие компоненты без разделения решений или проектов. Эта возможность не используется в приложении Plan My Night, но представлена здесь, потому что данная папка создается шаблоном MVC-проекта.
- **Controllers (Контроллеры)** Во время обработки запроса инфраструктура ASP.NET MVC ищет контроллеры для обработки запроса в этой папке.
- **Views (Представления)** Папка Views на самом деле является структурой каталогов. Его подпапки носят имена соответствующие классам в папке Controllers. Также имеется подпапка Shared (Общие). Она предназначена для представлений, частичных представлений, главных страниц и любых других ресурсов, которые должны быть доступны всем котроллерам.

Дополнительные сведения Более подробные сведения о компонентах ASP.NET MVC, а также отличия обработки запросов от того, как это происходит в Веб-формах ASP.NET, представлены по адресу http://asp.net/mvc.

В большинстве случаев web.config является последним файлом в корневой папке проекта. В Visual Studio 2010 для этого файла предлагается существенное обновление: Преобразование Web.config. Эта возможность позволяет создавать на базе основного web.config файлы web.config для конкретных сборок, переопределяющие настройки базового файла во время сборки, развертывания и выполнения. Эти файлы отображаются под базовым web.config, как показано на рис. 9-2.

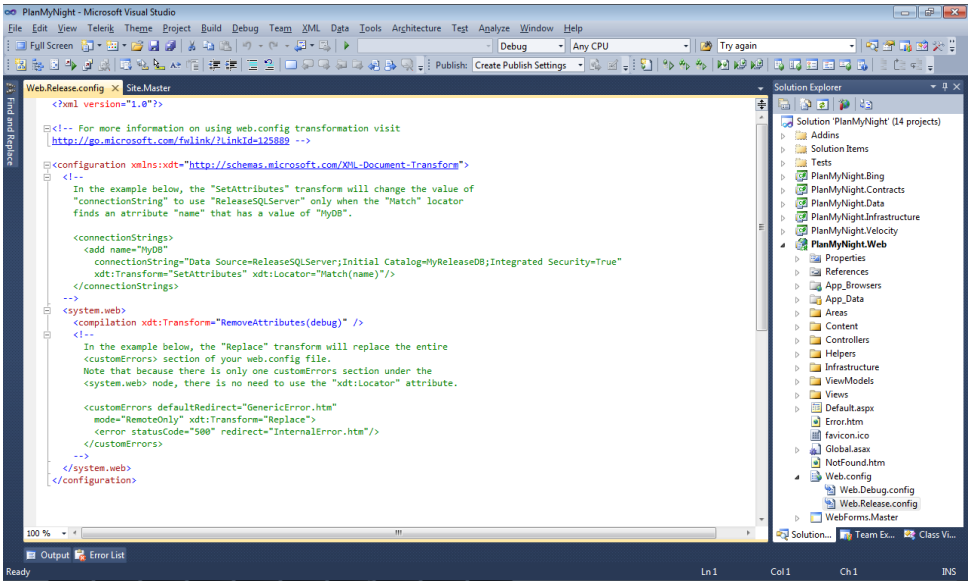


Рис. 9-2 Файл Web.config и специальные файлы конфигурации для конкретных сборок

> **Visual Studio 2008** В Visual Studio 2008 приходилось всегда помнить о том, что параметры в web.config не должны переопределяться параметрами отладки. Или нельзя было забывать вносить в web.config правильные настройки после его публикации для окончательной сборки. В Visual Studio 2010 таких проблем нет. Для переопределения значений в web.config при окончательных сборках используется набор параметров файла web.Release.config.retail; при отладочных сборках используется файл web.Debug.config.

Проект также включает разделы:

- **Content (Содержимое)** Набор папок, содержащих изображения, сценарии и файлы стилей.
- **Helpers (Вспомогательные классы)** Различные классы, включающие ряд методов расширения, которые расширяют функциональность типов, используемых в проекте.
- **Infrastructure (Инфраструктура)** В данную папку включены элементы, обеспечивающие взаимодействие с более низкоуровневой инфраструктурой ASP.NET MVC (например, фабрики кэширования и контроллеров).
- **ViewModels (Модели представлений)** Объекты данных, заполненные классами контроллеров (Controller) и используемые представлениями (Views) для отображения данных.

Запуск проекта

Если скомпилировать и запустить проект, на экране должно быть выведено следующее (рис. 9-3).

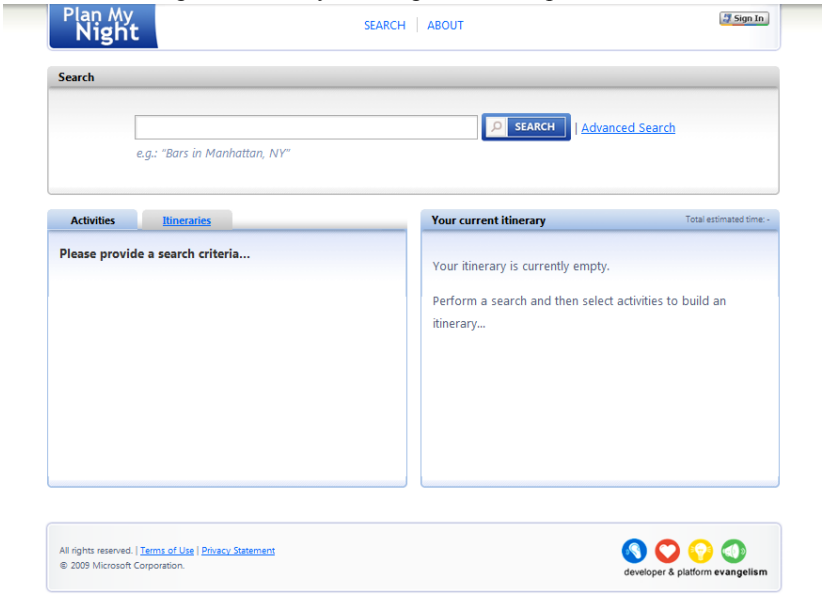


Рис. 9-3 Страница по умолчанию приложения Plan My Night

Поиск и организация исходного списка элементов плана мероприятий функционируют, но если попытаться сохранить разрабатываемый план мероприятий или выполнить вход с использованием Windows Live ID, приложение возвратит ошибку 404 Not Found (Не найден) (как показано на рис. 9-4).

Рис. 9-4 Ошибка, возвращаемая приложением Plan My Night при попытке регистрации

Это происходит потому, что на данный момент в проекте нет контроллера учетной записи для обработки таких запросов.

Создание контроллера учетной записи

Класс *AccountController* (Контроллер учетной записи) обеспечивает приложение Plan My Night критически важными функциями:

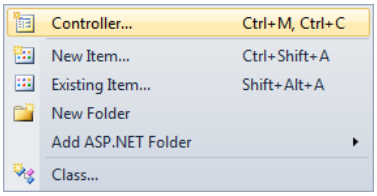
- Он обрабатывает вход и выход пользователей из приложения (с использованием Windows Live ID).

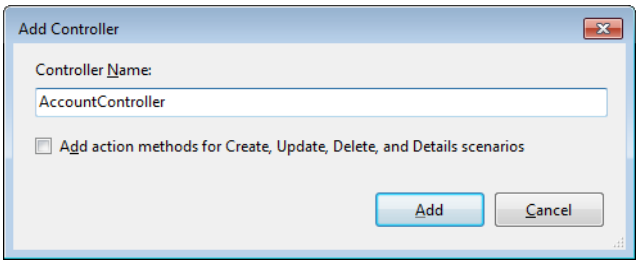
- Обеспечивает действия для отображения и обновления данных профиля пользователя.

Для создания нового контроллера ASP.NET MVC:

134. В Solution Explorer (Обозреватель решений) перейдите к папке Controllers проекта PlanMyNight.Web и щелкните ее правой кнопкой мыши.

135. Откройте подменю Add и выберите пункт Controller.





136. Введите имя контроллера, **AccountController**.

> **Примечание** Не устанавливайте флажок Add Action Methods For Create, Update, And Delete Scenarios (Добавить методы для сценариев создания, обновления и удаления). Установка этого флажка обеспечит вставку некоторых методов-«заглушек», но поскольку мы не будем использовать методы по умолчанию, в их создании нет необходимости

По щелчку кнопки Add в диалоговом окне Add Controller откроется базовый класс *AccountController* с единственным методом *Index*:

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

namespace Microsoft.Samples.PlanMyNight.Web.Controllers
{

    public class AccountController : Controller
    {
        //
        // GET: /Account/

        public ActionResult Index()
        {
            return View();
        }

    }
}
```

> **Visual Studio 2008** Следует отметить отличие от разработки приложений ASP.NET Web Forms в Visual Studio 2008: в приложениях ASP.NET MVC для файлов .aspx не создаются файлы выделенного кода. Котроллеры, созданием которого мы только что занимались, осуществляют логику, необходимую для обработки ввода и подготовки вывода. Такой подход обеспечивает четкое разделение логики отображения и бизнес-логики, и это является ключевым аспектом ASP.NET MVC.

Реализация функциональности

Для взаимодействия с любым слоем доступа к данным и сервисами (Моделью) потребуется создать и инициализировать некоторые поля экземпляров. Но перед этим следует добавить ряд пространств имен в блок директив *using*:

```
using System.IO;
using Microsoft.Samples.PlanMyNight.Data;
using Microsoft.Samples.PlanMyNight.Entities;
using Microsoft.Samples.PlanMyNight.Infrastructure;
using Microsoft.Samples.PlanMyNight.Infrastructure.Mvc;
using Microsoft.Samples.PlanMyNight.Web.ViewModels;
using System.Collections.Specialized;
using WindowsLiveId;
```

Вот теперь добавим поля экземпляров. Эти поля являются интерфейсами различных разделов Модели:

```
public class AccountController : Controller
{
    private readonly IWindowsLiveLogin windowsLogin;
    private readonly IMembershipService membershipService;
    private readonly IFormsAuthentication formsAuthentication;
    private readonly IReferenceRepository referenceRepository;
    private readonly IActivitiesRepository activitiesRepository;
.
.
.
```

Примечание Использование интерфейсов для взаимодействия со всеми внешними зависимостями обеспечивает лучшую портируемость кода на различные платформы. Также интерфейсы более эффективно изолируют отдельные компоненты, что облегчает моделирование зависимостей при тестировании

Как уже упоминалось, эти поля представляют части Модели, с которой будет взаимодействовать данный контроллер для реализации функциональных требований. Рассмотрим общие описания каждого из этих интерфейсов:

- **IWindowsLiveLogin (Регистрация WindowsLive)** Описывает контракт функциональности для взаимодействия с сервисом Windows Live ID.
- **IMembershipService (Сервис членства)** Описывает данные профиля пользователя и методы авторизации. В используемом приложении-примере это абстракция ASP.NET Membership Service (Сервис членства).
- **IFormsAuthentication (Аутентификация с помощью форм)** Описывает фукнциональность для взаимодействия с абстракцией ASP.NET Forms Authentication (Аутентификация с помощью форм).
- **IReferenceRepository (Хранилище справочных ресурсов)** Описывает справочные ресурсы, такие как списки состояний и другие характерные для модели данные.
- **IActivitiesRepository (Хранилище действий)** Интерфейс для извлечения и обновления данных действий.

В этот класс добавим два конструктора: один – общий для времени выполнения, который использует класс *ServiceFactory* (Фабрика сервисов) для получения ссылок на необходимые интерфейсы, и другой – для обеспечения возможности вводить конкретные экземпляры интерфейсов при тестировании.

```
public AccountController() :
    this(
        new ServiceFactory().GetMembershipService(),
        new WindowsLiveLogin(true),
        new FormsAuthenticationService(),
        new ServiceFactory().GetReferenceRepositoryInstance(),
        new ServiceFactory().GetActivitiesRepositoryInstance())
{
}
public AccountController(
    IMembershipService membershipService,
    IWindowsLiveLogin windowsLogin,
    IFormsAuthentication formsAuthentication,
    IReferenceRepository referenceRepository,
    IActivitiesRepository activitiesRepository)
{
    this.membershipService = membershipService;
    this.windowsLogin = windowsLogin;
    this.formsAuthentication = formsAuthentication;
    this.referenceRepository = referenceRepository;
    this.activitiesRepository = activitiesRepository;
}
```

Аутентификация пользователя

Первой настоящей функциональностью, реализованной нами в этом контроллере, будет вход и выход из приложения. Большинство методов, которые будут реализовываться в дальнейшем, требуют аутентификации, поэтому именно с нее мы и начнем.

В нашем приложении для обеспечения аутентификации используется одновременно несколько технологий: Windows Live ID, ASP.NET Forms Authentication и ASP.NET Membership Services. Эти три технологии применяются в действии LiveID, которое будет реализовано следующим.

Начнем с создания в классе *AccountController* следующего метода:

```
public ActionResult LiveId()
{
    return Redirect("~/");
}
```

Это основной метод для взаимодействия с сервисами Windows Live ID. Если вызвать его прямо сейчас, он просто перенаправит пользователя к корню приложения.

> **Примечание** Вызов *Redirect* (Перенаправить) возвращает *RedirectResult* (Результат перенаправления). В данном примере для определения цели перенаправления используется строка, но в разных ситуациях могут использоваться различные перегрузки этого метода

Когда Windows Live ID возвращает пользователя в приложение, может быть предпринято несколько разных типов действий. Пользователь может выполнить вход в Windows Live ID, выполнить выход или очистить «cookies» Windows Live ID. Когда Windows Live ID возвращает пользователя, в его URL присутствует строковый параметр запроса *action*, поэтому переключение ветвей логики выполняется на основании значения этого параметра.

Добавим следующий код в метод *LiveId* над выражением *return*:

```
string action = Request.QueryString["action"];
switch (action)
{
    case "logout":
        this.formsAuthentication.SignOut();
        return Redirect("~/");

    case "clearcookie":
        this.formsAuthentication.SignOut();
        string type;
        byte[] content;
        this.windowsLogin.GetClearCookieResponse(out type, out content);
        return new FileStreamResult(new MemoryStream(content), type);
}
```

Дополнительные сведения Полную документацию по системе Windows Live ID можно найти по адресу http://dev.live.com/.

В только что добавленном коде обрабатываются два действия выхода для Windows Live ID. В обоих случаях используется интерфейс *IFormsAuthentication* для удаления файла «cookie» ASP.NET Forms Authentication, чтобы все будущие http-запросы (до момента повторной регистрации пользователя) не считались аутентифицированными. Во втором случае выполняется на одну операцию больше, и очищаются файлы «cookies» Windows Live ID (те, в которых сохраняется регистрационное имя, но не пароль).

Сценарий входа включает немного больший объем кода, поскольку требуется проверка присутствия аутентифицируемого пользователя в базе данных сервиса членства (Membership Database). Если этого пользователя там нет, для него создается новый профиль. Однако перед этим должны быть переданы данные, которые Windows Live ID переслал в интерфейс Windows Live ID, что позволит проверить их и предоставить объект *WindowsLiveLogin.User*:

```
default:
    // вход
    NameValueCollection tokenContext;
    if ((Request.HttpMethod ?? "GET").ToUpperInvariant() == "POST")
    {
        tokenContext = Request.Form;
    }
    else
    {
        tokenContext = new NameValueCollection(Request.QueryString);
        tokenContext["stoken"] =
```

```
                System.Web.HttpUtility.UrlEncode(tokenContext["stoken"]);
        }

        var liveIdUser = this.windowsLogin.ProcessLogin(tokenContext);
```

На данном этапе, если пользователь выполнит вход, *liveIdUser* будет либо ссылаться на аутентифицированный объект *WindowsLiveLogin.User*, либо будет null. Исходя из этого, добавим следующий раздел кода, который обеспечивает выполнение некоторых действий, если значение *liveIdUser* не *null*:

```
        if (liveIdUser != null)
        {
            var returnUrl = liveIdUser.Context;
            var userId = new Guid(liveIdUser.Id).ToString();
            if (!this.membershipService.ValidateUser(userId, userId))
            {
                this.formsAuthentication.SignIn(userId, false);
                this.membershipService.CreateUser(userId, userId, string.Empty);
                var profile = this.membershipService.CreateProfile(userId);
                profile.FullName = "New User";
                profile.State = string.Empty;
                profile.City = string.Empty;
                profile.PreferredActivityTypeId = 0;
                this.membershipService.UpdateProfile(profile);

                if (string.IsNullOrEmpty(returnUrl)) returnUrl = null;
                return RedirectToAction("Index", new { returnUrl = returnUrl });
            }
            else
            {
                this.formsAuthentication.SignIn(userId, false);
                if (string.IsNullOrEmpty(returnUrl)) returnUrl = "~/";
                return Redirect(returnUrl);
            }
        }
    break;
```

Вызов метода *ValidateUser* (Проверить пользователя) в *IMembershipService* позволяет приложению проверить, посещал ли данный пользователь этот сайт ранее и существует ли его профиль. Поскольку аутентификация пользователя выполняется по Windows Live ID, значение его ID (которое является GUID) используется и как имя пользователя, и как пароль для ASP.NET Membership Service.

Если в приложении нет записи для данного пользователя, она создается путем вызова метода *CreateUser* (Добавить пользователя). Затем посредством *CreateProfile* (Создать профиль) создается и профиль параметров пользователя. Профиль заполняется некоторыми значениями по умолчанию и сохраняется в хранилище. Пользователь перенаправляется на основную страницу ввода, где может обновить сведения.

> **Примечание** На основании сочетания входных параметров *Controller.RedirectToAction* определяет, какой URL должен быть создан. В данном случае требуется перенаправить пользователя к действию *Index* (Корень) этого контроллера и передать текущее значение URL-адреса возврата

Также в данном коде реализована регистрация пользователя в сервисе аутентификации ASP.NET Forms, т.е. обеспечивается создание файла «cookie» с идентификационными данными для использования в запросах, требующих аутентификации, в будущем.

Параметры профиля также управляются сервисами ASP.NET Membership Services и объявляются в файле web.config приложения:

```
<system.web>
…
<profile enabled="true">
    <properties>
        <add name="FullName" type="string" />
        <add name="State" type="string" />
        <add name="City" type="string" />
        <add name="PreferredActivityTypeId" type="int" />
    </properties>

    <providers>
        <clear />
        <add name="AspNetSqlProfileProvider"
type="System.Web.Profile.SqlProfileProvider,
```

```
            System.Web, Version=4.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a"
        connectionStringName="ApplicationServices"
        applicationName="/" />
    </providers>
</profile>
…
</system.web>
```

Теперь метод *LiveID* полностью готов и должен выглядеть, как следующий фрагмент кода. Приложение принимает сведения для аутентификации от Windows Live ID, подготавливает профиль ASP.NET Membership Service и создает маркер аутентификации ASP.NET Forms.

```
public ActionResult LiveId()
{
string action = Request.QueryString["action"];
switch (action)
{
    case "logout":
        this.formsAuthentication.SignOut();
        return Redirect("~/");

    case "clearcookie":
        this.formsAuthentication.SignOut();
        string type;
        byte[] content;
        this.windowsLogin.GetClearCookieResponse(out type, out content);
        return new FileStreamResult(new MemoryStream(content), type);

    default:
        // вход
        NameValueCollection tokenContext;
        if ((Request.HttpMethod ?? "GET").ToUpperInvariant() == "POST")
        {
            tokenContext = Request.Form;
         }
        else
        {
            tokenContext = new NameValueCollection(Request.QueryString);
            tokenContext["stoken"] =
                System.Web.HttpUtility.UrlEncode(tokenContext["stoken"]);
        }
        var liveIdUser = this.windowsLogin.ProcessLogin(tokenContext);

        if (liveIdUser != null)
        {
            var returnUrl = liveIdUser.Context;
            var userId = new Guid(liveIdUser.Id).ToString();
            if (!this.membershipService.ValidateUser(userId, userId))
            {
                this.formsAuthentication.SignIn(userId, false);
                this.membershipService.CreateUser(userId, userId,
                                                  string.Empty);
                var profile = this.membershipService.CreateProfile(userId);
                profile.FullName = "New User";
                profile.State = string.Empty;
                profile.City = string.Empty;
                profile.PreferredActivityTypeId = 0;
                this.membershipService.UpdateProfile(profile);

                if (string.IsNullOrEmpty(returnUrl)) returnUrl = null;
                return RedirectToAction("Index",
                                        new { returnUrl = returnUrl });
            }
            else
            {
                this.formsAuthentication.SignIn(userId, false);
                if (string.IsNullOrEmpty(returnUrl)) returnUrl = "~/";
                return Redirect(returnUrl);
            }
        }
```

```
        break;
    }
    return Redirect("~/");
    }
```

Конечно, пользователь должен иметь возможность перед регистрацией попасть на страницу входа Windows Live ID. На данный момент в приложении Plan My Night имеется кнопка входа с помощью Windows Live ID. Но возможны ситуации, когда понадобится перенаправить пользователя на страницу входа прямо из кода. Для реализации такого сценария добавим в контроллер небольшой метод под именем *Login* (Вход):

```
public ActionResult Login(string returnUrl)
{
    var redirect = HttpContext.Request.Browser.IsMobileDevice ?
                   this.windowsLogin.GetMobileLoginUrl(returnUrl) :
                   this.windowsLogin.GetLoginUrl(returnUrl);
    return Redirect(redirect);
}
```

Данный метод просто извлекает URL-адрес входа для Windows Live и перенаправляет пользователя по этому адресу. Это также удовлетворяет конфигурации для аутентификации ASP.NET Forms в web.config с той точки зрения, что любой запрос, требующий аутентификации, будет перенаправлен в этот метод:

```
<authentication mode="Forms">
    <forms loginUrl="~/Account/Login" name="XAUTH" timeout="2880" path="~/" />
</authentication>
```

Извлечение профиля для текущего пользователя

Теперь, когда методы аутентификации описаны, что реализует первую цель данного контролера – обработка входа и выхода пользователей из приложения – можно переходить к извлечению данных текущего пользователя.

Метод *Index* является методом по умолчанию для контроллера на основании конфигурации отображения URL в Global.asax. Он будет располагаться там, где должны извлекаться данные текущего пользователя и возвращаться представление, отображающее эти данные. Метод *Index*, изначально описывавшийся при создании класса *AccountController*, необходимо заменить следующим:

```
[Authorize()]
[AcceptVerbs(HttpVerbs.Get)]
public ActionResult Index(string returnUrl)
{
        var profile = this.membershipService.GetCurrentProfile();
        var model = new ProfileViewModel
        {
            Profile = profile,
            ReturnUrl = returnUrl ?? this.GetReturnUrl()
        };

        this.InjectStatesAndActivityTypes(model);

        return View("Index", model);
}
```

> **Visual Studio 2008** Атрибуты, такие как *[Authorize()]*, не имели такого широкого распространения в Visual Studio 2008, но в ASP.NET MVC они часто используются. Атрибуты позволяют описывать метаданные цели, к которой они относятся. Благодаря этому появляется возможность проверять данные во время выполнения (через отражение) и предпринимать действия в случае необходимости.

Очень удобен атрибут *Authorize* (Авторизовать). Он показывает, что данный метод может вызываться только для уже аутентифицированных http-запросов. Если запрос не аутентифицирован, он будет перенаправлен в заданную цель входа ASP.NET Forms Authentication, которая только что была настроена. Атрибут *AcceptVerbs* (Допустимые команды) также ограничивает способы вызова этого метода, задавая допустимые http-команды. В данном случае этот метод вызывается по запросу HTTP GET. В сигнатуру метода добавлен строковый параметр *returnUrl*. Он обеспечивает возможность возвращения пользователя на исходный адрес по завершении просмотра или обновления его сведений.

> **Примечание** Это приводит нас к части инфраструктуры ASP.NET MVC под названием *Привязка модели* (*Model Binding*), подробное рассмотрение которой выходит за рамки данной книги. Однако необходимо знать, что она пытается найти источник *returnUrl* (поле формы, данные таблицы маршрутизации или параметр строки запроса с таким же именем) и выполняет его привязку к этому значению при вызове данного метода. Если средству привязки модели не удается найти подходящего источника, значение будет null. Такое поведение может

обусловить проблемы для типов значений, которые не могут быть null, потому что приведет к формированию исключения *InvalidOperationException* (Недопустимая операция).

В целом, этот метод прост: он принимает возвращаемое значение метода *GetCurrentProfile* (Получить профиль текущего пользователя) интерфейса ASP.NET Membership Service и настраивает объект модели представления, который будет использоваться представлением. Вызов *GetReturnUrl* (Получить URL возврата) – это пример метода расширения, описанного в проекте PlanMyNight.Infrastructure. Он не является членом класса Controller, но в среде разработки обеспечивает намного более надежный код (рис. 9-5).

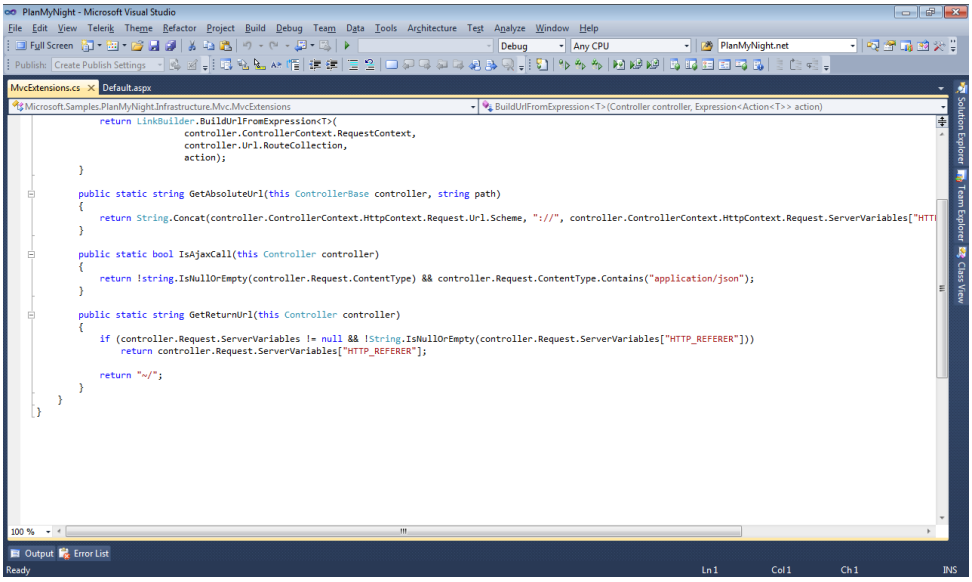


Рис. 9-5 Пример методов расширения в MvcExtensions.cs

InjectStatesAndActivityTypes (Ввести состояния и типы действий) – метод, который требуется реализовать в классе *AccountController*. Он собирает данные имен состояния из хранилища справочных ресурсов и данные о типах действий из хранилища действий. Он создает две коллекции *SelectListItem* (HTML-класс для MVC): одну для списка состояний, и другую для списка разных типов действий, доступных в приложении. Также он задает соответствующее значение.

```
private void InjectStatesAndActivityTypes(ProfileViewModel model)
{
    var profile = model.Profile;
    var types = this.activitiesRepository.RetrieveActivityTypes().Select(
                o => new SelectListItem {
                          Text = o.Name,
                          Value = o.Id.ToString(),
                          Selected = (profile != null && o.Id ==
                                      profile.PreferredActivityTypeId)
                }).ToList();

    types.Insert(0, new SelectListItem { Text = "Select...", Value = "0" });
    var states = this.referenceRepository.RetrieveStates().Select(
                 o => new SelectListItem {
                           Text = o.Name,
                           Value = o.Abbreviation,
                           Selected = (profile != null && o.Abbreviation ==
                                       profile.State)
                }).ToList();

    states.Insert(0, new SelectListItem {
                          Text = "Any state",
                          Value = string.Empty
                });

    model.PreferredActivityTypes = types;
    model.States = states;
}
```

Обновление данных профиля

Закончив с инфраструктурой для извлечения данных текущего профиля, перейдем к обновлению данных модели посредством формы, передаваемой пользователем. После этого можно создавать собственные

страницы представления и видеть, как все это сочетается. Метод *Update* прост, однако, в нем появляются некоторые новые до сих пор не встречавшиеся нам возможности:

```
[Authorize()]
[AcceptVerbs(HttpVerbs.Post)]
[ValidateAntiForgeryToken()]
public ActionResult Update(UserProfile profile)
{
    var returnUrl = Request.Form["returnUrl"];
    if (!ModelState.IsValid)
    {
        // ошибка валидации
        return this.IsAjaxCall() ? new JsonResult { JsonRequestBehavior =
                    JsonRequestBehavior.AllowGet, Data = ModelState }
                    : this.Index(returnUrl);
    }

    this.membershipService.UpdateProfile(profile);
    if (this.IsAjaxCall())
    {
        return new JsonResult { JsonRequestBehavior =
                                JsonRequestBehavior.AllowGet,
                                Data = new { Update = true,
                                             Profile = profile,
                                             ReturnUrl = returnUrl } };
    }
    else
    {
        return RedirectToAction("UpdateSuccess", "Account", new { returnUrl =
                                 returnUrl });
    }
}
```

Атрибут *ValidateAntiForgeryToken* (Проверить маркер, препятствующий фальсификации) подтверждает то, что форма не была сфальсифицирована. Для использования этой возможности необходимо добавить *AntiForgeryToken* (Маркер, препятствующий фальсификации) в форму ввода представления. Проверка действительности *ModelState* (Состояние модели) – наш первый опыт валидации ввода. Это валидация на стороне сервера, и ASP.NET MVC предлагает простую в использовании возможность убедиться в том, что поступающие данные удовлетворяют некоторым правилам. Одно из свойств объекта *UserProfile*, созданного для обеспечения ввода в этот метод через привязку модели MVC, имеет атрибут *System.ComponentModel.DataAnnotations.Required*. При привязке модели инфраструктура MVC проверяет атрибуты *DataAnnotation* (Аннотация данных) и объявляет *ModelState* действительным только в случае выполнения всех правил.

В случае, когда *ModelState* недействительный, пользователь перенаправляется в метод *Index*, где *ModelState* используется для отображения формы ввода. Или, если запрос был AJAX-вызовом, возвращается *JsonResult* с прикрепленными к нему данными *ModelState*.

> **Visual Studio 2008** В ASP.NET MVC запросы маршрутизируются не через страницы, а через контроллеры. Поэтому один URL может обрабатывать множество разных запросов и отвечать на них соответствующим представлением. В Visual Studio 2008 для реализации этой функциональности разработчику пришлось бы создавать два разных URL и вызывать метод в третьем классе

Если *ModelState* действительный, сервис членства обновляет профиль, и для AJAX-запросов возвращается JSON-результат со сведениями об успешности операции. Для «обычных» запросов пользователь перенаправляется к действию *UpdateSuccess* (Успешное обновление) контроллера Account. Метод *UpdateSuccess* – последний метод, необходимый для завершения выполнения данного контроллера:

```
public ActionResult UpdateSuccess(string returnUrl)
{
    var model = new ProfileViewModel
    {
        Profile = this.membershipService.GetCurrentProfile(),
        ReturnUrl = returnUrl
    };
    return View(model);
}
```

Этот метод используется для возвращения представления успешного выполнения в браузер, отображения некоторых обновленных данных и обеспечения ссылки для возвращения пользователя туда, где он находился перед началом процесса обновления профиля.

Теперь, когда контроллер Account полностью реализован, полученный класс должен выглядеть следующим образом:

```
    using System;
    using System.Collections.Specialized;
    using System.IO;
    using System.Linq;
    using System.Web;
    using System.Web.Mvc;
    using Microsoft.Samples.PlanMyNight.Data;
    using Microsoft.Samples.PlanMyNight.Entities;
    using Microsoft.Samples.PlanMyNight.Infrastructure;
    using Microsoft.Samples.PlanMyNight.Infrastructure.Mvc;
    using Microsoft.Samples.PlanMyNight.Web.ViewModels;
    using WindowsLiveId;

namespace Microsoft.Samples.PlanMyNight.Web.Controllers
{
    [HandleErrorWithContentType()]
    [OutputCache(NoStore = true, Duration = 0, VaryByParam = "*")]
    public class AccountController : Controller
    {
        private readonly IWindowsLiveLogin windowsLogin;
        private readonly IMembershipService membershipService;
        private readonly IFormsAuthentication formsAuthentication;
        private readonly IReferenceRepository referenceRepository;
        private readonly IActivitiesRepository activitiesRepository;

        public AccountController() :
            this(
                new ServiceFactory().GetMembershipService(),
                new WindowsLiveLogin(true),
                new FormsAuthenticationService(),
                new ServiceFactory().GetReferenceRepositoryInstance(),
                new ServiceFactory().GetActivitiesRepositoryInstance())
        {
        }

        public AccountController(IMembershipService membershipService,
                                 IWindowsLiveLogin windowsLogin,
                                 IFormsAuthentication formsAuthentication,
                                 IReferenceRepository referenceRepository,
                                 IActivitiesRepository activitiesRepository)
        {
            this.membershipService = membershipService;
            this.windowsLogin = windowsLogin;
            this.formsAuthentication = formsAuthentication;
            this.referenceRepository = referenceRepository;
            this.activitiesRepository = activitiesRepository;
        }

        public ActionResult LiveId()
        {
            string action = Request.QueryString["action"];
            switch (action)
            {
                case "logout":
                    this.formsAuthentication.SignOut();
                    return Redirect("~/");
                case "clearcookie":
                    this.formsAuthentication.SignOut();
                    string type;
                    byte[] content;
                    this.windowsLogin.GetClearCookieResponse(out type, out
content);
                    return new FileStreamResult(new MemoryStream(content), type);
                default:
                    // вход
                    NameValueCollection tokenContext;
                    if ((Request.HttpMethod ?? "GET").ToUpperInvariant() == "POST")
```

```
            {
                tokenContext = Request.Form;
            }
            else
            {
                tokenContext = new
             NameValueCollection(Request.QueryString);
                tokenContext["stoken"] =
             System.Web.HttpUtility.UrlEncode(tokenContext["stoken"]);
            }

            var liveIdUser =
                    this.windowsLogin.ProcessLogin(tokenContext);
            if (liveIdUser != null)
            {
                var returnUrl = liveIdUser.Context;
                var userId = new Guid(liveIdUser.Id).ToString();
                if (!this.membershipService.ValidateUser(userId, userId))
                {
                    this.formsAuthentication.SignIn(userId, false);
                    this.membershipService.CreateUser( userId, userId,
                                                       string.Empty);
                    var profile =
                        this.membershipService.CreateProfile(userId);
                    profile.FullName = "New User";
                    profile.State = string.Empty;
                    profile.City = string.Empty;
                    profile.PreferredActivityTypeId = 0;
                    this.membershipService.UpdateProfile(profile);
                    if (string.IsNullOrEmpty(returnUrl)) returnUrl =
                                                         null;
                    return RedirectToAction("Index", new { returnUrl =
                                                         returnUrl });
                }
                else
                {
                    this.formsAuthentication.SignIn(userId, false);
                    if (string.IsNullOrEmpty(returnUrl)) returnUrl =
                                                         "~/";
                    return Redirect(returnUrl);
                }
            }
            break;
        }
        return Redirect("~/");
    }

    public ActionResult Login(string returnUrl)
    {
        var redirect = HttpContext.Request.Browser.IsMobileDevice ?
                  this.windowsLogin.GetMobileLoginUrl(returnUrl) :
                  this.windowsLogin.GetLoginUrl(returnUrl);
        return Redirect(redirect);
    }

    [Authorize()]
    [AcceptVerbs(HttpVerbs.Get)]
    public ActionResult Index(string returnUrl)
    {
        var profile = this.membershipService.GetCurrentProfile();
        var model = new ProfileViewModel
        {
            Profile = profile,
            ReturnUrl = returnUrl ?? this.GetReturnUrl()
        };

        this.InjectStatesAndActivityTypes(model);
        return View("Index", model);
    }
```

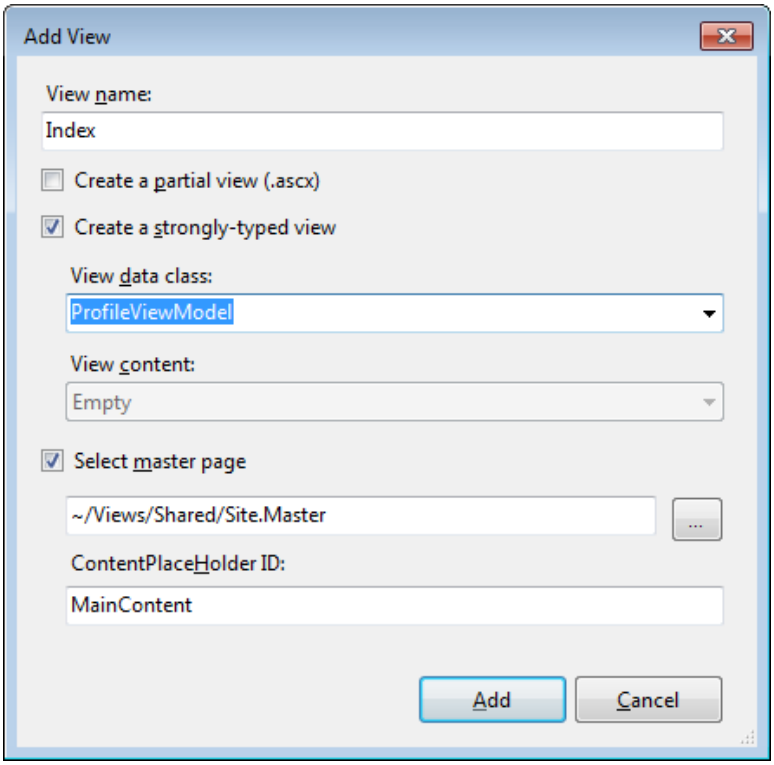
```
        [Authorize()]
        [AcceptVerbs(HttpVerbs.Post)]
        [ValidateAntiForgeryToken()]
        public ActionResult Update(UserProfile profile)
        {
            var returnUrl = Request.Form["returnUrl"];
            if (!ModelState.IsValid)
            {
                // ошибка валидации
                return this.IsAjaxCall() ?
                new JsonResult { JsonRequestBehavior =
                JsonRequestBehavior.AllowGet, Data = ModelState }
                                    : this.Index(returnUrl);
            }
            this.membershipService.UpdateProfile(profile);
            if (this.IsAjaxCall())
            {
                return new JsonResult {
                       JsonRequestBehavior = JsonRequestBehavior.AllowGet,
                       Data = new {
                       Update = true,
                       Profile = profile,
                       ReturnUrl = returnUrl } };
            }
            else
            {
                return RedirectToAction("UpdateSuccess",
                        "Account", new { returnUrl = returnUrl });
            }
        }
        public ActionResult UpdateSuccess(string returnUrl)
        {
            var model = new ProfileViewModel
            {
                Profile = this.membershipService.GetCurrentProfile(),
                ReturnUrl = returnUrl
            };
            return View(model);
        }

        private void InjectStatesAndActivityTypes(ProfileViewModel model)
        {
            var profile = model.Profile;
            var types = this.activitiesRepository.RetrieveActivityTypes()
                        .Select(o => new SelectListItem { Text = o.Name,
                              Value = o.Id.ToString(),
                              Selected = (profile != null &&
                              o.Id == profile.PreferredActivityTypeId) })
                              .ToList();
            types.Insert(0, new SelectListItem { Text = "Select...",
                                                 Value = "0" });
            var states = this.referenceRepository.RetrieveStates().Select(
                              o => new SelectListItem {
                              Text = o.Name,
                              Value = o.Abbreviation,
                              Selected = (profile != null &&
                              o.Abbreviation == profile.State) })
                              .ToList();
            states.Insert(0, new SelectListItem { Text = "Any state",
                                                  Value = string.Empty });
            model.PreferredActivityTypes = types;
            model.States = states;
        }
    }
}
```

Создание представления учетной записи

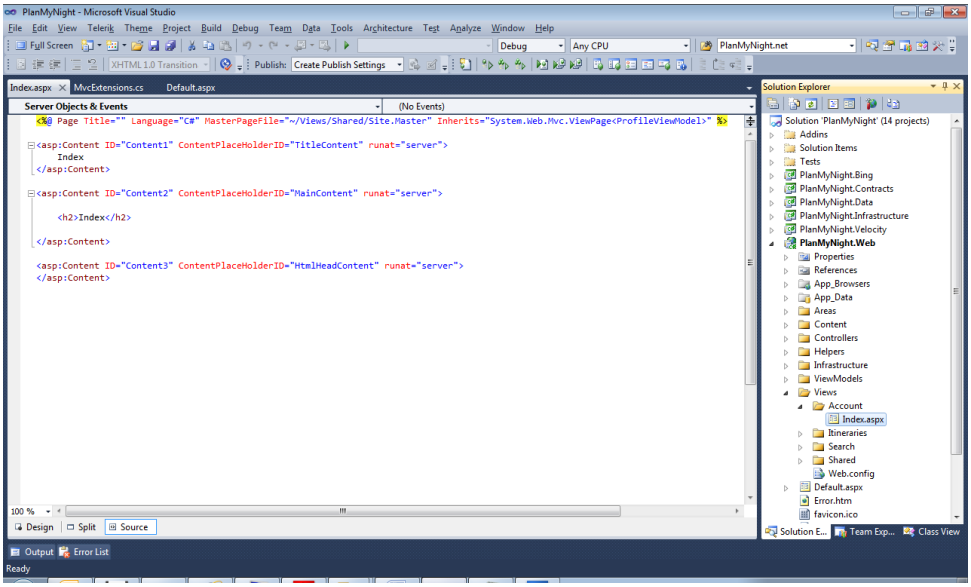
В предыдущем разделе был создан контроллер, реализующий функциональность для обновления и просмотра данных пользователя. В этом разделе мы поэтапно рассмотрим возможности Visual Studio 2010, позволяющие создавать представления, которые будут отображать эту функциональность пользователю.

Для создания представления Index контроллера Account:

137. Перейдем к папке Views проекта PlanMyNight.Web.
138. Щелкнем папку Views правой кнопкой мыши, раскроем подменю Add и выберем New Folder.
139. Назовем новую папку **Account**.
140. Щелкнем правой кнопкой мыши новую папку Account, раскроем подменю Add и выберем View.
141. Заполним поля диалогового окна Add View (Добавить представление), как показано на рисунке:



142. Щелкнем Add. Мы должны получить HTML-страницу с несколькими элементами управления *<asp:Content>*:



Можно заметить, здесь нет особых отличий от того, что мы привыкли видеть в Visual Studio 2008. По умолчанию ASP.NET MVC 2 использует механизм формирования представлений Веб-форм ASP.NET, поэтому страницы MVC и Веб-форм будут иметь некоторое сходство. Основное отличие на этом этапе в том, что класс *page* наследуется от *System.Web.Mvc.ViewPage<ProfileViewModel>*, и отсутствует файл

выделенного кода. MVC не использует файлы выделенного кода, в отличие от Веб-форм ASP.NET, чтобы обеспечить четкое разделение функциональных областей. Редактирование страниц MVC, как правило, осуществляется через разметку. Конструктор применяется преимущественно для приложений Веб-форм ASP.NET.

Чтобы этот каркас страницы стал основным представлением контроллера Account, необходимо изменить содержимое заголовка, обеспечив его единообразие с остальными представлениями:

```
<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Plan My Night – Profile
</asp:Content>
```

Далее требуется добавить клиентские сценарии, которые будут использоваться в содержимом *HtmlHeadContent* (Содержимое HTML-заголовка):

```
<asp:Content ID="Content3" ContentPlaceHolderID="HtmlHeadContent" runat="server">
    <% Ajax.RegisterClientScriptInclude(
      Url.Content("~/Content/Scripts/jquery-1.4.1.min.js")); %>
    <% Ajax.RegisterCombinedScriptInclude(
      Url.Content("~/Content/Scripts/MicrosoftAjax.js"), "pmn"); %>
    <% Ajax.RegisterCombinedScriptInclude(
      Url.Content("~/Content/Scripts/MicrosoftAjaxMvc.js"), "pmn"); %>
    <% Ajax.RegisterCombinedScriptInclude(
      Url.Content("~/Content/Scripts/MicrosoftMvcValidation.js"), "pmn"); %>
    <% Ajax.RegisterCombinedScriptInclude(
        Url.Content("~/Content/Scripts/ajax.common.js"), "pmn"); %>
    <% Ajax.RegisterCombinedScriptInclude(
        Url.Content("~/Content/Scripts/ajax.profile.js"), "pmn"); %>
    <%= Ajax.RenderClientScripts() %>
</asp:Content>
```

Этот сценарий использует методы расширения *System.Web.Mvc.AjaxHelper*, описанные в проекте PlanMyNight.Infrastructure в папке MVC.

Когда содержимое заголовка настроено, можно заняться основным содержимым представления:

```
<asp:Content ContentPlaceHolderID="MainContent" runat="server">
<div class="panel" id="profileForm">
    <div class="innerPanel">
        <h2><span>My Profile</span></h2>
        <% Html.EnableClientValidation(); %>
        <% using (Html.BeginForm("Update", "Account")) %>
        <% { %>
        <%=Html.AntiForgeryToken()%>
        <div class="items">
            <fieldset>
                <p>
                    <label for="FullName">Name:</label>
                    <%=Html.EditorFor(m => m.Profile.FullName)%>
                    <%=Html.ValidationMessage("Profile.FullName",
                        new { @class = "field-validation-error-wrapper" })%>
                </p>
                <p>
                    <label for="State">State:</label>
                    <%=Html.DropDownListFor(m => m.Profile.State,
                                            Model.States)%>
                </p>
                <p>
                    <label for="City">City:</label>
                    <%=Html.EditorFor(m => m.Profile.City,
                                      Model.Profile.City)%>
                </p>
                <p>
                    <label for="PreferredActivityTypeId">Preferred
activity:</label>
                    <%=Html.DropDownListFor(m =>
                        m.Profile.PreferredActivityTypeId,
                        Model.PreferredActivityTypes)%>
                </p>
            </fieldset>
            <div class="submit">
                <%=Html.Hidden("returnUrl", Model.ReturnUrl)%>
```

```
                <%=Html.SubmitButton("submit", "Update")%>
            </div>
        </div>
        <div class="toolbox"></div>
        <% } %>
        </div>
</div>
</asp:Content>
```

Если не обращать внимания на некоторый встроенный код, все это выглядит как обычная HTML-разметка. Рассмотрим фрагменты встроенного кода и продемонстрируем, какую мощь они обеспечивают (и простоту при этом).

> **Visual Studio 2008** В Visual Studio 2008 более типичным для отображения данных было использовать серверные элементы управления и логику времени отображения. Но поскольку страницы представлений ASP.NET MVC не имеют файла выделенного кода, эта логика должна быть описана в одном файле с разметкой. По-прежнему могут применяться элементы управления ASP.NET Web Forms. В нашем примере используется элемент управления *<asp:Content>*. Однако, как правило, функциональность элементов управления ASP.NET Web Forms ограничена из-за отсутствия файла выделенного кода

В MVC широко применяются вспомогательные классы HTML. Методы, содержащиеся в *System.Web.Mvc.HtmlHelper*, генерируют компактные соответствующие стандартам HTML-теги для использования в различных целях. Для этого MVC-разработчику в некоторых случаях приходится писать больше кода разметки, чем разработчику Веб-форм, но он получает непосредственный контроль над формированием вывода. Строго типизированная версия этого класса расширения (*HtmlHelper<TModel>*) может использоваться в разметке представления через свойство *ViewPage<TModel>.Html*.

В данной форме используются такие *HTML*-методы (и это лишь часть того, что доступно по умолчанию):

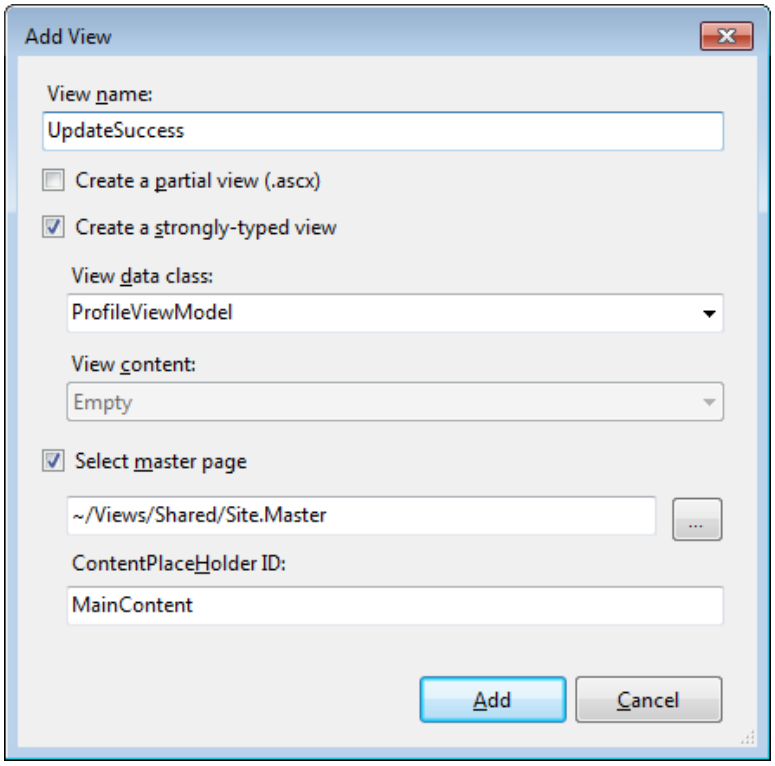
- *Html.EnableClientValidation* (Включить проверку на стороне клиента) обеспечивает проверку данных на стороне клиента на основании строго типизированного словаря ModelState
- *Html.BeginForm* (Начать форму) помещает в разметку тег *<form>* и закрывает форму в конце раздела директив *using*. Принимает различные параметры, но наиболее часто используемыми являются имя действия и контроллер, для которого это действие должно быть вызвано. Благодаря этому инфраструктура MVC может автоматически формировать URL конкретной формы во время выполнения, что избавляет от необходимости вводить строку URL в разметку.
- *Html.AntiForgeryToken* размещает в форме скрытое поле с проверочным значением, которое также сохраняется на стороне сервера и проверяется, если цель формы имеет атрибут *ValidateAntiForgeryToken*. Мы добавили этот атрибут в методе *Update* контроллера.
- *Html.EditorFor* (Редактор для) – перегруженный метод, обеспечивающий вставку текстового поля в разметку. Это строго типизированная версия метода *Html.Editor*.
- *Html.DropDownListFor* (Раскрывающийся список для) – перегруженный метод, обеспечивающий вставку раскрывающегося списка в разметку. Это строго типизированная версия метода *Html.DropDownList*.
- *Html.ValidationMessage* (Сообщение проверки) – вспомогательный метод, который обеспечит вывод на экран сообщения об ошибке проверки, если заданный ключ уже имеется в словаре ModelState.
- *Html.Hidden* (Скрытый) помещает в форму скрытое поле с переданными именем и значением.
- *Html.SubmitButton* (Кнопка Передать) создает в форме кнопку Submit (Передать).

> **Примечание** Когда разметка представления Index готова, остается лишь добавить представления для действия *UpdateSuccess*, и можно будет просматривать результаты.

Для создания представление UpdateSuccess:

143. Раскроем проект PlanMyNight.Web в Solution Explorer и затем раскроем папку Views.
144. Щелкнем правой кнопкой мыши папку Account.
145. Откроем подменю Add и щелкнем View.

146.Заполним поля диалогового окна Add View, чтобы оно выглядело следующим образом:



После создания страницы представления заполним содержимое заголовка следующим образом:

```
<asp:Content ContentPlaceHolderID="TitleContent" runat="server">Plan My Night -
Profile Updated</asp:Content>
```

И содержимое *MainContent* должно выглядеть так:

```
<asp:Content ContentPlaceHolderID="MainContent" runat="server">
<div class="panel" id="profileForm">
    <div class="innerPanel">
        <h2><span>My Profile</span></h2>
        <div class="items">
            <p>Your profile has been successfully updated.</p>
            <h3>» <a href="<%=Html.AttributeEncode(Model.ReturnUrl ??
                          Url.Content("~/"))%>">Continue</a></h3>
        </div>
        <div class="toolbox">
        </div>
    </div>
</div>
</asp:Content>
```

Чтобы увидеть созданные представления, необходимо отредактировать файл Site.Master (располагающийся в папке Views/Shared в корневом каталоге Веб-проекта). Раскомментируем строку 33, удалив теги комментариев. Код должен выглядеть следующим образом:

```
<%=Html.ActionLink<AccountController>(c => c.Index(null), "My Profile")%>
```

Теперь, когда это последнее представление создано, можно откомпилировать и выполнить приложение. Щелкнем кнопку Sign In (Вход), которая показана в верхнем правом углу формы на рис. 9-6, и выполним вход с помощью Windows Live ID.

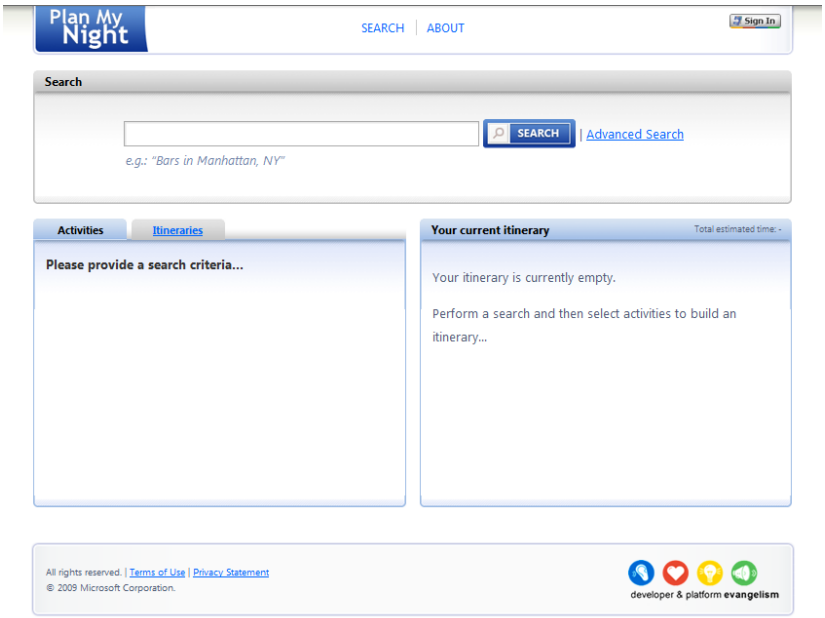


Рис. 9-6 Окно по умолчанию приложения Plan My Night

После того, как вход выполнен, вы должны быть перенаправлены к созданному представлению Index контроллера Account (рис. 9-7).

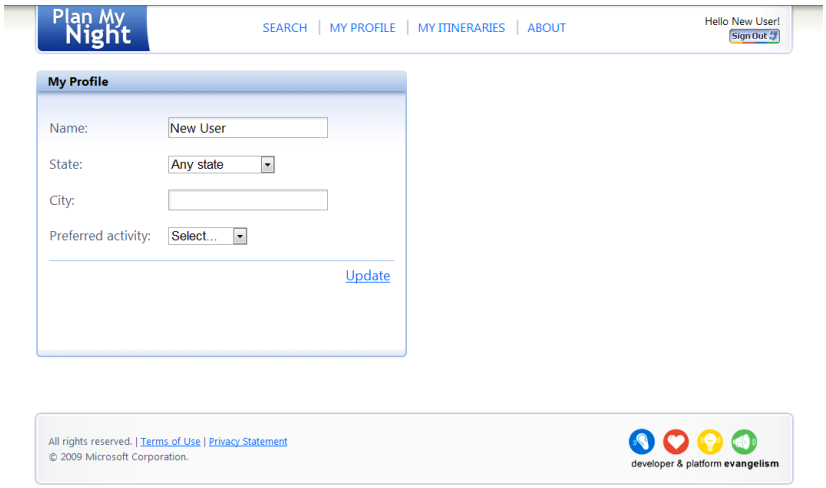


Рис. 9-7 Окно настроек профиля, возвращенное методом Index контроллера Account

Если вместо этого вы оказались на странице поиска, просто щелкните ссылку My Profile (Мой профиль), которая находится в центре вверху интерфейса рядом с остальными ссылками. Чтобы увидеть новые возможности проверки данных в действии, попробуйте сохранить форму, не заполнив поле Full Name (Полное имя). На экране должно появиться следующее сообщение (рис. 9-8).

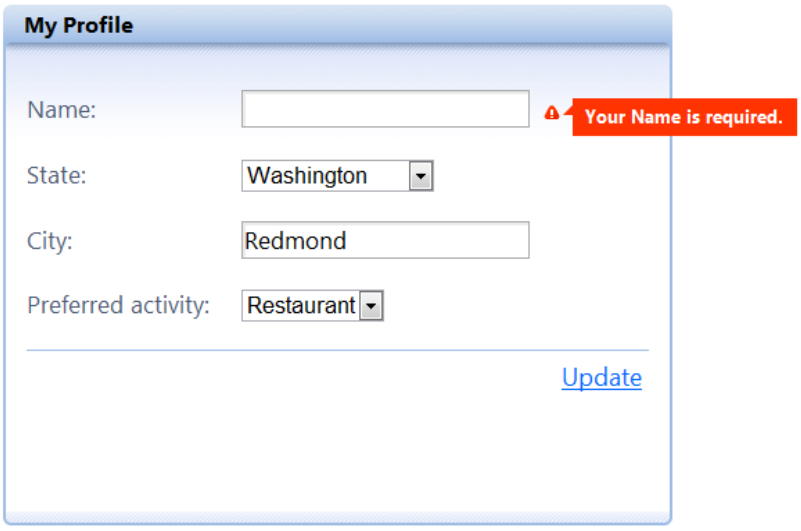


Рис. 9-8 Пример сбоя при проведении проверки Model Binding

Поскольку в приложении включена проверка на стороне клиента, обратного вызова не было. Чтобы увидеть, как работает проверка на стороне сервера, потребовалось бы внести изменения в файл Index.aspx в папке Account, закомментировав вызов *Html.EnableClientValidation*. Благодаря тесной интеграции и поддержке AJAX и JavaScript в приложениях MVC намного упрощается перенос на сторону клиента таких серверных операций, как проверка, по сравнению с тем, как это было ранее.

> **Visual Studio 2008** В приложениях ASP.NET MVC значение атрибута ID отдельного HTML-элемента не преобразовывается, как это происходит в ASP.NET Web Forms 3.5. В Visual Studio 2008 разработчик должен был сохранять *UniqueID* элемента управления/элемента в переменной JavaScript, чтобы обеспечить возможность доступа к нему внешнего JavaScript. Это делалось, чтобы гарантировать уникальность ID, но всегда вводило дополнительный уровень сложности во взаимодействие между элементами управления ASP.NET 3.5 Web Forms и JavaScript. В MVC такого преобразования нет, но обеспечение уникальности ID – сфера ответственности разработчиков. Также следует отметить, что теперь ASP.NET 4.0 Web Forms поддерживает отключение преобразования ID на уровне элемента управления по желанию разработчика

Контроллер Account и связанные представления и являются той недостающей «базовой» функциональностью Plan My Night. В ходе работы над ними мы рассмотрели некоторые новые возможности приложений Visual Studio 2010 и MVC 2.0. Но MVC не единственный доступный выбор для Веб-разработчиков. Веб-формы ASP.NET являются основным типом приложений ASP.NET с момента ее создания, и были улучшены в Visual Studio 2010. В следующем разделе Веб-форма ASP.NET для приложения MVC будет создана в визуальном дизайнере.

Использование дизайнера для создания Веб-формы

Все приложения рано или поздно сталкиваются с непредвиденными условиями, и наше приложение-пример не исключение. При возникновении непредвиденной ситуации оно возвращает сообщение об ошибке, такое как показано на рис. 9-9.

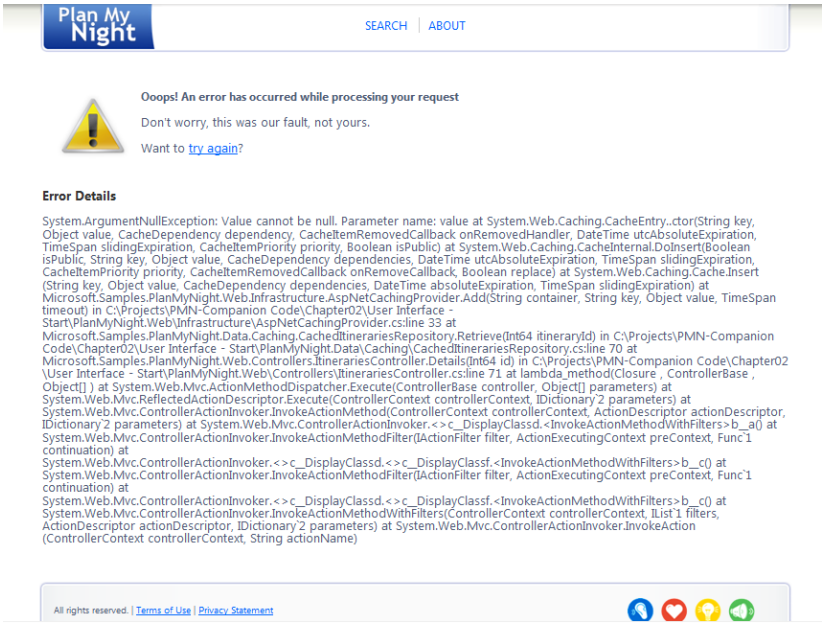


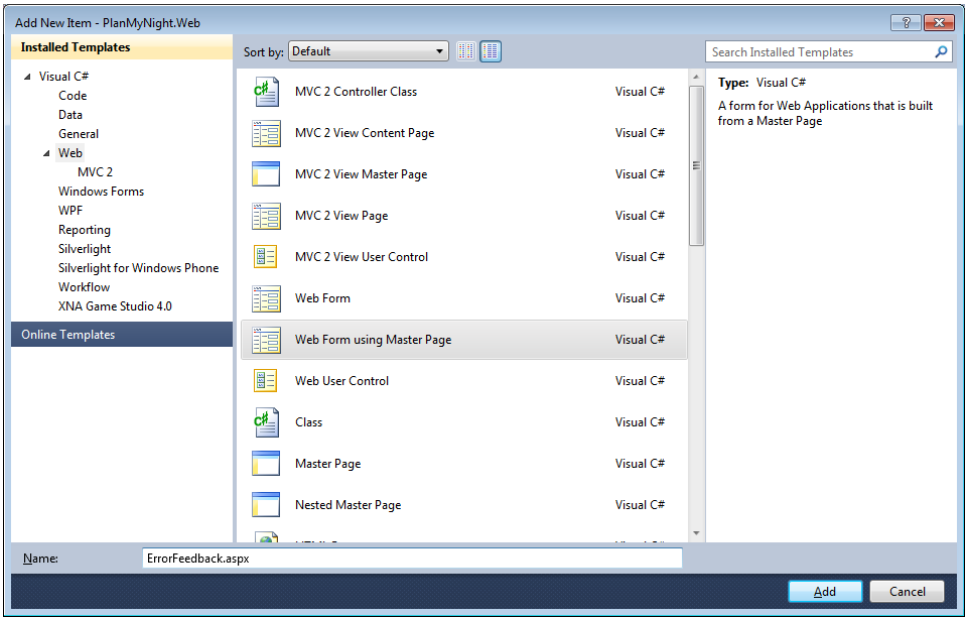
Рис. 9-9 Пример сообщения об ошибке приложения Plan My Night

В настоящее время у пользователя, получившего такое сообщение, небогатый выбор: либо попытаться повторить это действие, либо использовать навигационные ссылки в верхней части окна приложения. (Конечно, это может привести к еще одной ошибке.) Предоставление пользователю возможности обратной связи позволит разработчикам получать сведения о ситуации, которых могут не обеспечивать стандартное сообщение об ошибке и трассировка стека. Чтобы рассмотреть другой способ создания компонента пользовательского интерфейса для приложения Plan My Night, спроектируем страницу обратной связи в случае возникновения ошибки как Веб-форму ASP.NET, преимущественно используя для этого дизайнер Visual Studio. Прежде чем приступать к дизайну формы, необходимо создать файл формы.

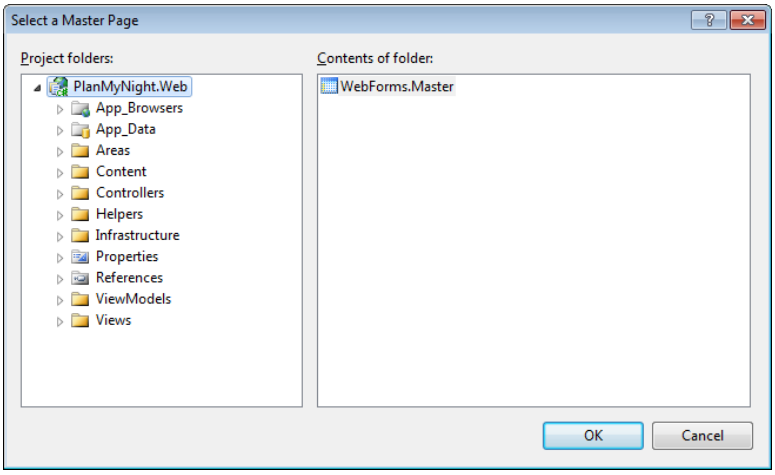
Для создания новой Веб-формы:

147. Откроем контекстное меню проекта PlanMyNight.Web (щелкнув его правой кнопкой мыши), откроем подменю Add и выберем New Item.

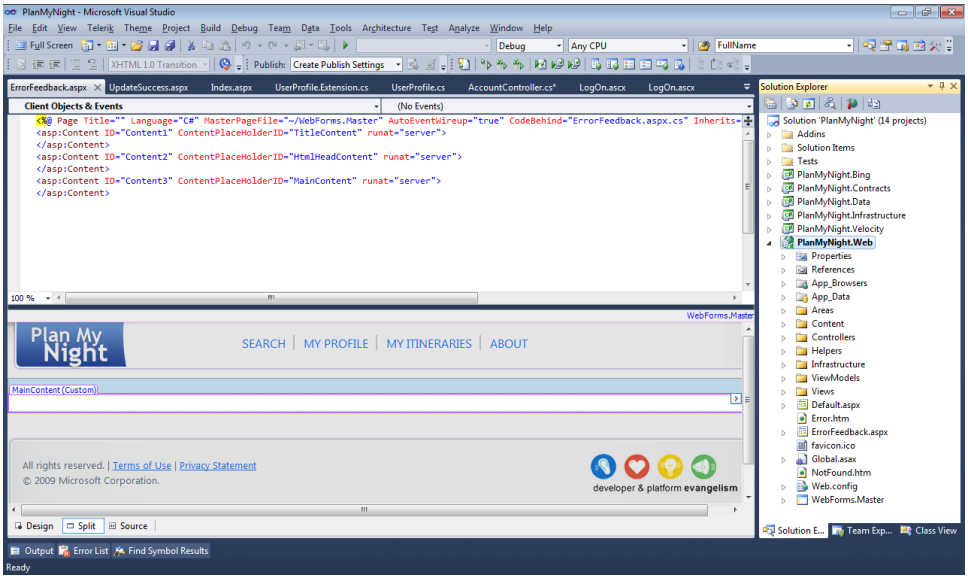
148. В диалоговом окне Add New Item выберем Web Form using Master Page (Веб-форма, использующая главную страницу) и в поле Name введем имя элемента: **ErrorFeedback.aspx**.



149. Появится диалоговое окно, которое позволяет ассоциировать главную страницу с этой Веб-формой. Убедитесь, что в части Project Folders (Папки проекта) окна выбрана основная папка PlanMyNight.Web, и затем выберите элемент WebForms.Master в правой части окна.



150. Результирующая страница может быть представлена не в режиме разделенного представления (Split view), а в режиме просмотра исходного кода (или в режиме дизайнера (Design view)). Перейдите к разделенному представлению (соответствующая кнопка располагается внизу окна, как и в предыдущих версиях Visual Studio). После этого экран должен выглядеть следующим образом:



> **Примечание** Рекомендуется использовать разделенное представление, поскольку оно позволяет видеть исходный код, формируемый дизайнером, и добавлять разметку в случае необходимости.

Раскройте панель элементов управления и оставьте ее на экране, поскольку в ходе работы над формой будете постоянно брать из нее элементы управления и элементы и переносить в область содержимого. Если эта панель еще не вынесена на экран, ее можно найти в меню View (Вид).

Начнем с переноса методом drag-and-drop элемента div (из группы элементов HTML) из панели элементов управления в раздел MainContent (Основное содержимое) дизайнера. При этом появится вкладка div, свидетельствующая о том, что добавленный новый элемент выбран в настоящий момент. Откроем контекстное меню div и выберем Properties (Свойства) (которые также можно открыть, нажав клавишу F4). В раскрывшемся окне Properties задаем свойству *(Id)* значение *profileForm* (Форма профиля) (регистр имеет значение). Также изменим значение свойства *Class* (Класс) на *panel* (панель). После редактирования этих значений размер области содержимого изменится, потому что к представлению дизайнера применены CSS.

Поместим другой div внутрь первого и зададим его свойству *class* значение *innerPanel* (Внутренняя панель). На панели разметки добавим в *innerPanel* следующую разметку:

```
<h2><span>Error Feedback</span></h2>
```

Закрыв тег <h2>, добавим новую строку и откроем контекстное меню. Выберем Insert Snippet (Вставить фрагмент) и последовательно щелкнем ASP.NET > formr. Это обеспечит создание тега серверной формы, которая будет служить контейнером для Веб-элементов управления. Между тегами формы поместим тег div, атрибуту class которого присвоено значение *items* (элементы), и затем в тег тег div вставим тег fieldset (набор полей).

Далее перетягиваем в тег fieldset элемент управления TextBox (находится в разделе Standard (Стандартные) панели элементов управления). Задаем ID этого текстового поля значение **FullName**. В представлении разметки перед этим элементом управления добавляем тег <label> (метка), в качестве значения его свойства *for* задаем значение ID текстового поля, т.е. **Full Name:** (двоеточие обязательно). Значение тега <label> - это текст, размещаемый между тегами <label> и </label>. Поместите эти два элемента в тег <p>, и представление дизайнера должно выглядеть, как показано на рис. 9-10.

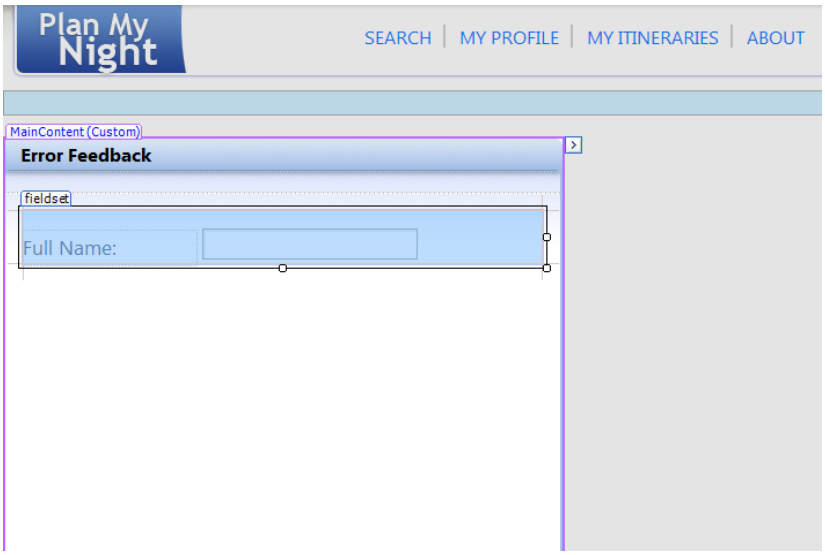


Рис. 9-10 Так на данный момен выглядит ErrorFeedback.aspx в дизайнере

Вышеописанным способом добавим еще одно текстовое поле и метку, но в качестве значения ID текстового поля зададим **EmailAddress** (Адрес электронной почты) и в качестве значения метки – **Email Address:** (не забывая о двоеточии). Повторим весь этот процесс третий раз и зададим TextBox ID и метку **Comments** (Комментарии). Теперь в дизайнере должно находиться три метки и три однострочных элемента управления TextBox. Для элемента управления Comments необходимо обеспечить многострочный ввод, поэтому открываем его свойства и задаем *TextMode* (Текстовый режим) значение *Multiline* (Многострочный), *Rows* (Строки) значение *5* и *Columns* (Столбцы) значение *40*. После этого текстовое поле станет намного шире, что позволит пользователю вводить комментарии.

Снова воспользуемся возможностью Insert Snippet и после текстового поля Comments вставим тег «div with class» (HTML>divc). В качестве класса тега div зададим *submit*. Из панели инструментов перетащим в этот div элемент управления Button (Кнопка). Зададим свойству *Text* (Текст) кнопки значение *Send Feedback* (Отправить отзыв).

В дизайнере должно отображаться нечто похожее представленному на рис. 9-11. На данный момент мы получили страницу, которая будет передавать форму.

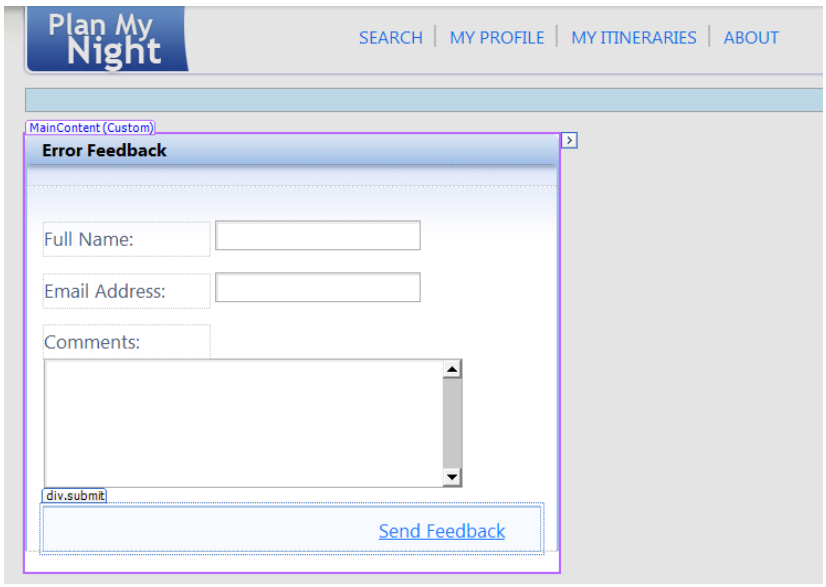


Рис. 9-11 Форма ErrorFeedback.aspx с полным набором полей

Но эта форма не выполняет никакой проверки передаваемых ею данных. Для реализации такой функциональности воспользуемся некоторыми элементами управления для проверки ввода пользователя,

предлагаемыми ASP.NET. Сделаем поля Full Name и Comments обязательными для заполнения и осуществим проверку адреса электронной почты при помощи регулярных выражений, чтобы гарантировать его соответствие заданному шаблону.

В группе Validation (Проверка) панели инструментов имеются некоторые готовые элементы управления для проверки ввода пользователя. Возьмите на панели инструментов объект *RequiredFieldValidator* (Средство проверки обязательного поля) и поместите его справа от текстового поля Full Name. Откройте свойства этого элемента управления для проверки и задайте свойству *ControlToValidate* (Проверяемый элемент управления) значение *FullName*. (Для удобства предлагается поле со списком элементов управления страницы.) Также задайте свойству *CssClass* (Класс CSS) значение *field-validation-error* (ошибка проверки поля). Это обусловит то, что все ошибки в приложении будут обозначаться красным треугольником. Наконец, свойству Error Message (Сообщение об ошибке) задаем значение **Name is Required** (Необходимо указать имя) (рис. 9-12).

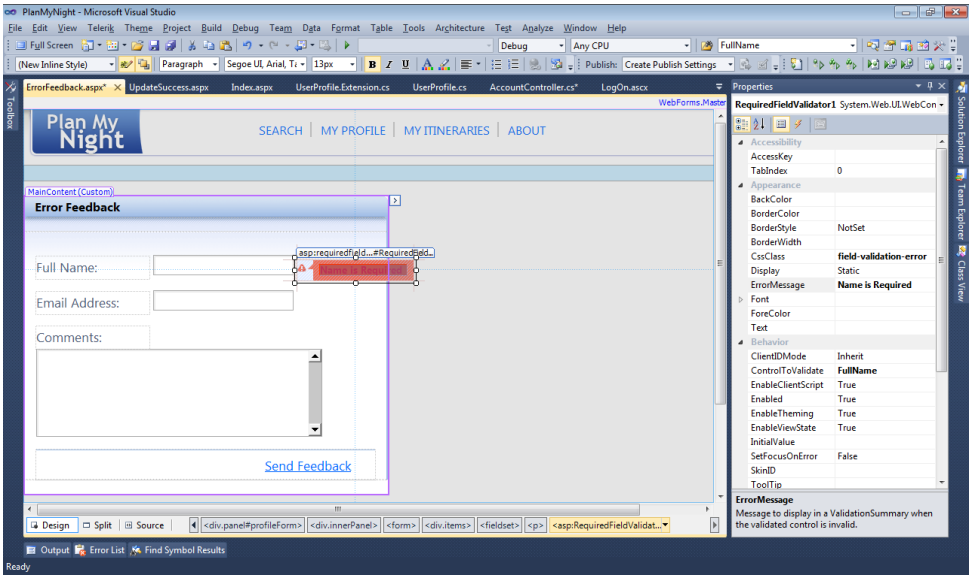


Рис. 9-12 Пример элемента управления для проверки ввода пользователя

Эти же шаги повторим для поля Comments, но зададим соответствующие значения свойствам *ErrorMessage* (Сообщение об ошибке) и *ControlToValidate*.

Пользователь должен гарантированно вводить действительный адрес электронной почты в поле Email Address, поэтому возьмем на панели инструментов элемент управления *RegularExpressionValidator* и поместим его рядом с этим текстовым полем. Значения свойствам этого элемента управления задаем по используемой ранее схеме, т.е. свойству *ControlToValidate* присваиваем значение *EmailAddress* и свойству *CssClass* – значение *field-validation-error*. Но для этого элемента управления также задается регулярное выражение, которое должно применяться к вводимым данным. Это осуществляется посредством свойства *ValidationExpression* (Выражение проверки), значение которого должно быть определено следующим образом:

```
[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}
```

Сообщение об ошибке этого средства проверки должно быть сформулировано примерно так: «Введите действительный адрес электронной почты».

Форма готова. Чтобы увидеть ее в приложении, необходимо добавить опцию обратной связи при возникновении ошибки для пользователя. В Solution Explorer перейдите по дереву проекта PlanMyNight.Web к папке Views и затем к подпапке Shared. Откройте файл Error.aspx в средстве просмотра разметки и перейдите к строке 35. В этой строке обрабатывается предложение пользователю сделать еще одну попытку выполнить действие, и сюда мы поместим опцию для отправки отзыва. После текста вопроса в тот же абзац добавьте следующую разметку:

```
or <a href="/ErrorFeedback.aspx">send feedback</a>?
```

Это обеспечит добавление опции перехода к только что созданной нами форме при возникновении в приложении MVC любой ошибки общего характера. Чтобы увидеть эту форму, необходимо спровоцировать ошибку в приложении.

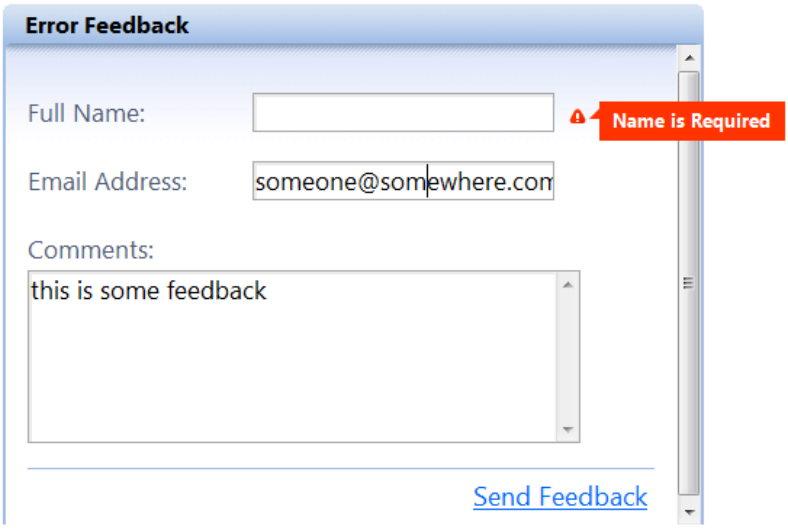
Чтобы спровоцировать возникновение ошибки в приложении Plan My Night:

151. Запустите приложение.

152. Увидев на экране стандартную страницу поиска, введите в адресную строку браузера следующий адрес: **http://www.planmynight.net:48580/Itineraries/Details/38923828**.

153. Очень маловероятно, что в базе данных имеется план мероприятий с таким ID, поэтому на экране появится страница с сообщением об ошибке. Если приложение выполняется с подключенным отладчиком, выполнение прервется на точке останова при возникновении исключительной ситуации. Продолжите выполнение приложения (по умолчанию для этого используется клавиша F5), и на экране появится следующее окно:

154. На странице с сообщением об ошибке щелкните ссылку для перехода к форме обратной связи. Попробуйте передать форму, введя в нее недействительные данные.



Для осуществления проверки ASP.NET использует сценарий на стороне клиента (если браузер поддерживает это), поэтому пока данные вводятся, никаких обратных вызовов не выполняется. Когда же сервер получает введенные данные, разработчик может проконтролировать состояние проверки на стороне сервера с помощью свойства *Page.IsValid* в файле выделенного кода. Однако поскольку использовалась проверка на стороне клиента (по умолчанию), значение этого свойства всегда будет *true*. В файл выделенного кода необходимо добавить лишь код, обеспечивающий перенаправление пользователя при обратном вызове (и проверку свойства *Page.IsValid* в случае, если что-то было упущено при проверке на стороне клиента):

```
protected void Page_Load(object sender, EventArgs e)
    {
        if (this.IsPostBack && this.IsValid)
        {
            this.Response.Redirect("/", true);
        }
    }
```

В этом нет особого смысла для пользователя, но нашей целью в данном разделе было создание Веб-формы ASP.NET с помощью дизайнера. Поэтому в проекте PlanMyNight.Web появился новый интерфейс. Но что,

если бы потребовалось обеспечить большую модульность функциональности, скажем, реализовать некую функцию, которую можно было бы добавлять или удалять без компиляции основного проекта приложения. Вот здесь-то и может продемонстрировать свои преимущества такая инфраструктура расширения как Managed Extensibility Framework (MEF).

Расширение приложения с помощью MEF

Visual Studio 2010 предлагает новую технологию в составе .NET Framework 4 – Managed Extensibility Framework (MEF). Managed Extensibility Framework обеспечивает разработчикам простой, но при этом мощный механизм, с помощью которого сторонние производители получают возможность расширять приложения уже после их поставки. Благодаря MEF даже в рамках одного приложения разработчики могут создавать полностью изолированные компоненты, что обеспечивает возможность их независимого обслуживания или замены. Для этого MEF использует контейнер разрешений, который позволяет сопоставлять компоненты, обеспечивающие конкретную функцию (экспортеры), и компоненты, нуждающиеся в этой функциональности (импортеры), и при этом двум конкретным компонентам даже не надо ничего знать друг о друге. Разрешения выполняются только на контрактной основе, что делает компоненты взаимозаменяемыми или позволяет вводить их в приложения с очень небольшими издержками.

> *Дополнительные сведения Веб-сайт сообщества разработчиков MEF, на котором представлена детальная информация об этой архитектуре, можно найти по адресу* http://mef.codeplex.com.

Приложение Plan My Night создавалось с учетом возможности расширения. В папке Addins (Надстройки) его решения имеется три проекта модулей «надстроек» (рис. 9-13).

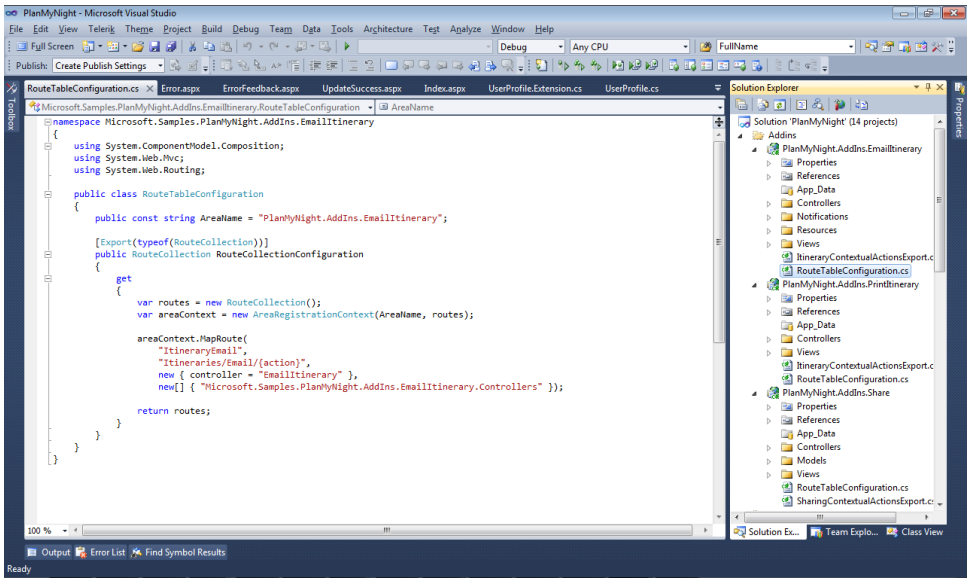


Рис. 9-13 Надстройки приложения Plan My Night

PlanMyNight.Addins.EmailItinerary добавляет возможность отправлять списки маршрутов всем, кому пожелает пользователь. PlanMyNight.Addins.PrintItinerary обеспечивает версию для печати маршрута. Наконец, PlanMyNight.Addins.Share добавляет функции публикации на социальных сайтах (что позволяет пользователю публиковать ссылку на маршрут), а также операции «обрезки» URL. Ни один из этих проектов не ссылается на основное приложение и не упоминается в нем. Да, в них имеются ссылки на проекты PlanMyNight.Contracts и PlanMyNight.Infrastructure, что позволяет экспортировать (и импортировать в некоторых случаях) соответствующие контракты через MEF, а также использовать любые специальные расширения проекта инфраструктуры.

> **Примечание** Если Веб-приложение еще не выполняется, прежде чем переходить к следующему этапу, запустите проект PlanMyNight.Web, чтобы видеть пользовательский интерфейс.

Чтобы добавить модули в выполняющееся приложение, запустим файл DeployAllAddins.bat, располагающийся в одной папке с файлом PlanMyNight.sln. Это обеспечит создание новых папок в разделе Areas проекта PlanMyNight.Web. Новые папки, по одной для каждой надстройки, будут включать файлы, необходимые для добавления их функциональности в основное Веб-приложение. Надстройки появляются в приложении как дополнительные опции на странице результатов поиска и на странице данных маршрута под разделом текущего маршрута. После того как командный файл закончит выполнение, перейдите к интерфейсу PlanMyNight, выполните поиск действия и добавьте его в текущий маршрут. Теперь под панелью маршрута, кроме New (Создать) и Save (Сохранить), должны появиться дополнительные опции (рис. 9-14).

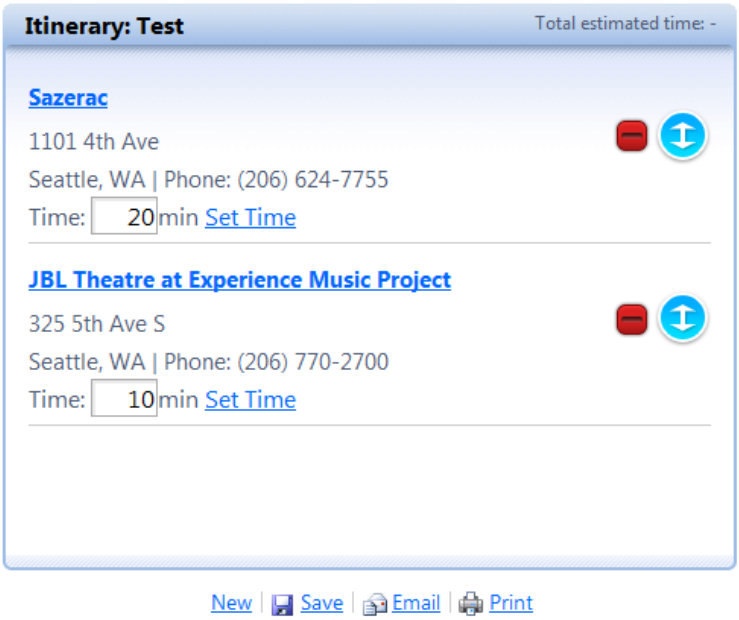


Рис. 9-14 Пользовательский интерфейс с надстройкой для отправки маршрута по электронной почте

Опции публикации на социальных сайтах начнут отображаться в интерфейсе только после того, как маршрут будет сохранен и отмечен как общедоступный (рис. 9-15).

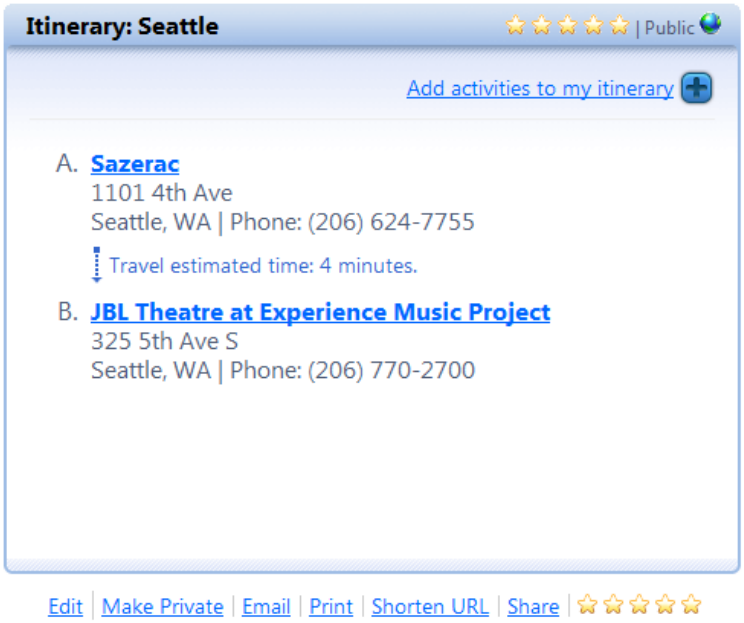


Рис. 9-15 Пользовательский интерфейс с надстройкой для публикации маршрута на социальных сайтах

Visual Studio 2008 В Visual Studio 2008 нет ничего похожего на MEF. Для обеспечения поддержки подключаемых модулей разработчику пришлось бы создавать инфраструктуру подключаемого модуля с нуля или приобретать коммерческий продукт. Любой из этих вариантов приводил к созданию собственных решений, в которых внешнему разработчику приходилось разбираться в случае необходимости реализации компонента для них. Добавление MEF в .NET Framework помогает убрать барьеры для перехода к разработке расширяемых приложений и подключаемых модулей для них.

Надстройка для создания печатной версии плана мероприятий

Рассмотрим использование подключаемых модулей в приложении на примере проекта PrintItinerary.Addin. Дерево проекта в развернутом виде должно быть похожим на структуру, представленную на рис. 9-16.

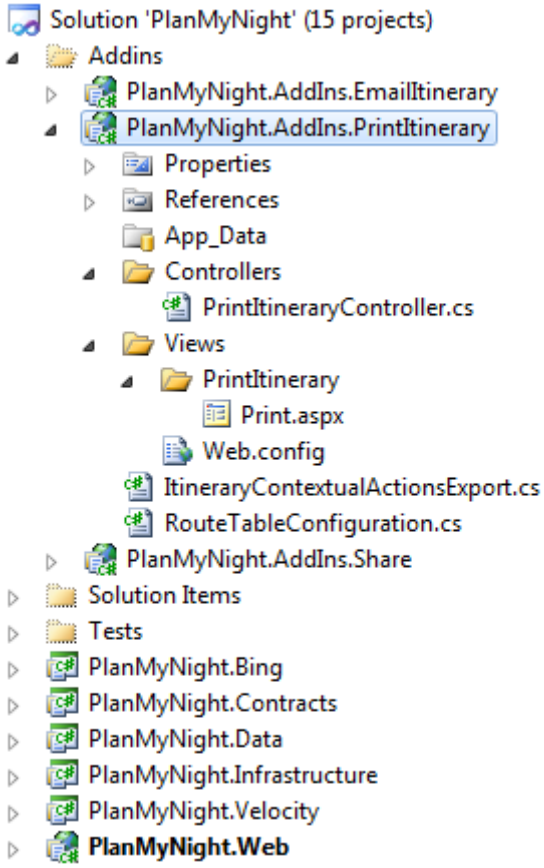


Рис. 9-16 Структура проекта PrintItinerary

Некоторые моменты данной структуры аналогичны проекту PlanMyNight.Web (Controllers и Views), и поэтому эта надстройка будет помещена в приложение MVC как Area. Если более внимательно посмотреть на файл PrintItineraryController.cs в папке Controller, можно заметить, что его структура очень похожа на структуру контроллера, созданного нами ранее в этой главе (и любого другого контроллера Веб-приложения), но при этом имеются и существенные отличия от контроллеров, компилируемых в основном приложении PlanMyNight.Web.

В описании класса присутствуют дополнительные атрибуты:

```
[Export("PrintItinerary", typeof(IController))]
[PartCreationPolicy(CreationPolicy.NonShared)]
```

Эти два атрибута описывают данный тип контейнера разрешений MEF. Первый атрибут, *Export* (Экспорт), помечает этот класс как предоставляющий *IController* с именем контракта *PrintItinerary*. Второй атрибут объявляет, что этот объект поддерживает только индивидуальное создание и не может создаваться как совместно используемый/singleton-объект. Задание этих двух атрибутов – это все что необходимо для создания типа, используемого MEF. На самом деле, *PartCreationPolicy* (Политика создания части) является необязательным атрибутом, но он должен быть определен на случай, если тип не может обрабатывать все типы политики создания.

Далее в файле PrintItineraryController.cs можно заметить, что конструктор снабжен атрибутом *ImportingConstructor* (Конструктор импорта):

```
[ImportingConstructor]
public PrintItineraryController(IServiceFactory serviceFactory) :
this(
    serviceFactory.GetItineraryContainerInstance(),
    serviceFactory.GetItinerariesRepositoryInstance(),
    serviceFactory.GetActivitiesRepositoryInstance())
    {
    }
```

Атрибут *ImportingConstructor* указывает MEF о необходимости предоставления параметров при создании этого объекта. В данном конкретном случае MEF обеспечивает экземпляр *IServiceFactory* для использования этим объектом. Классу *this* не важно, откуда поступает экземпляр, участвующий в создании модульных приложений. Для наших целей оговоренный контрактом *IServiceFactory* экспортируется в проект PlanMyNight.Web файлом ServiceFactory.cs.

Файл RouteTableConfiguration.cs регистрирует сведения о маршруте URL, которые должны быть направлены в PrintItineraryController. Этот маршрут, и маршруты других надстроек, регистрируются в приложении в ходе выполнения метода *Application_Start*, описанного в файле Global.asax.cs приложения PlanMyNight.Web:

```
// Фабрика EF Controller
var controllerFactory = new MefControllerFactory(container);
ControllerBuilder.Current.SetControllerFactory(controllerFactory);

// Регистрация маршрутов из надстроек
foreach (RouteCollection routes in container.GetExportedValues<RouteCollection>())
{
    foreach (var route in routes)
    {
        RouteTable.Routes.Add(route);
    }
}
```

controllerFactory – это контейнер MEF, включающий путь к подпапке Areas (в которой находятся все надстройки). Он назначен фабрикой контроллера на весь срок жизни приложения. Благодаря этому контроллеры, импортированные через MEF, могут использоваться в любом месте приложения. Маршруты этих надстроек извлекаются из контейнера MEF и регистрируются в таблице маршрутизации MVC.

Файл ItineraryContextualActionsExport.cs экспортирует сведения для создания ссылки на этот подключаемый модуль, а также создания метаданных для отображения. Эти сведения используются в файле ViewModelExtensions.cs проекта PlanMyNight.Web при построении модели представления для отображения пользователю:

```
// получаем ссылки и панели инструментов надстроек
var addinBoxes = new List<RouteValueDictionary>();
var addinLinks = new List<ExtensionLink>();

addinBoxes.AddRange(AddinExtensions.GetActionsFor("ItineraryToolbox", model.Id == 0
? null : new { id = model.Id }));

addinLinks.AddRange(AddinExtensions.GetLinksFor("ItineraryLinks", model.Id == 0 ?
null : new { id = model.Id }));
```

Вызов *AddinExtensions.GetLinksFor* обеспечивает перечисление экспортированных элементов в поставщике экспорта MEF и возвращает их коллекцию, которая может быть добавлена в локальную коллекцию *addinLinks* (Ссылки на надстройки). Впоследствии эта коллекция используется в представлении для отображения большего числа опций, если они имеются.

Заключение

В данной главе мы обсудили лишь несколько возможностей из большого числа новых технологий, предлагаемых Visual Studio 2010. Поэтапно рассмотрели создание контроллера и его связанного представления, а также мощную возможность, предлагаемую Веб-разработчикам инфраструктурой ASP.NET MVC для создания Веб-приложений. Мы познакомились, как с помощью Managed Extensibility Framework можно создавать внешние модули надстроек и подключать их к приложению во время выполнения. В следующей главе будет показано, какие улучшения предлагает Visual Studio 2010 для отладки приложений.

Глава 10

От 2008 к 2010: отладка приложения

В данной главе рассматривается

- Использование новых возможностей отладки Microsoft Visual Studio 2010
- Создание модульных тестов и их выполнение в Visual Studio 2010
- Проводится сравнение возможностей и отличий Visual Studio 2008 и 2010

В ходе написания данной книги мы увидели, насколько сильно эволюционировали инструменты отладки и различные вспомогательные средства для разработчиков в трех последних версиях Visual Studio. В Visual Studio 2010 внимание акцентируется на отладке приложения и написании модульных тестов, что лишь расширяет предлагаемые возможности.

Возможности отладки в Visual Studio 2010

В данной главе рассматриваются различные возможности отладки. В качестве примера используется модифицированное приложение Plan My Night. Если сопроводительные материалы данной книги установлены в каталог по умолчанию, интересующее нас приложение находится по адресу %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 10\Code. Щелкните двойным щелчком файл PlanMyNight.sln.

Прежде чем переходить непосредственно к процессу отладки, необходимо выполнить некоторую настройку:

155. Проверьте в Solution Explorer, является ли PlanMyNight.Web автозагружаемым проектом. Если его имя не выделено жирным шрифтом, щелкните PlanMyNight.Web правой кнопкой и выберите Set As StartUp Project (Назначить автозагружаемым проектом).

156. Для подготовки к следующим этапам в решении PlanMyNight.Web откройте файл Global.asax.cs, щелкнув треугольник рядом с папкой Global.asax, и затем щелкните двойным щелчком файл Global.asax.cs, как показано на рис. 10-1.

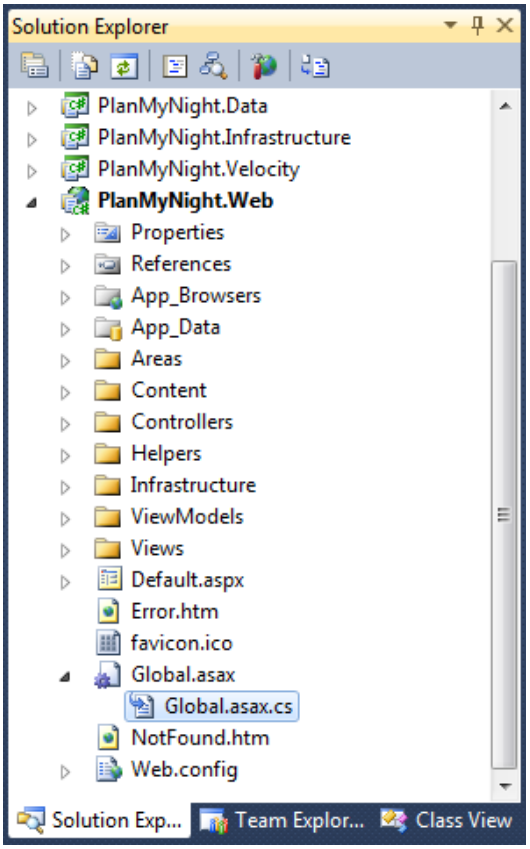


Рис. 10-1 Внешний вид Solution Explorer до открытия файла Global.asax.cs

Управление сеансом отладки

На примере приложения Plan My Night мы увидим, как может разработчик управлять точками останова и организовывать их совместное использование. Новые улучшенные возможности работы с точками останова позволяют значительно повысить скорость и эффективность проверки различных элементов данных в приложении. Также будут рассмотрены новые минидампы и новый интерпретатор промежуточного языка (intermediate language, IL), с помощью которого разработчик может видеть свойства и функции управляемого кода во время отладки минидампа.

Новые улучшенные возможности работы с точками останова

Итак, на данный момент в редакторе открыт файл Global.ascx.cs. Рассмотрим поэтапно некоторые из возможных путей управления точками останова и организации их совместного использования.

157. Перейдем к методу *Application_BeginRequest(object sender, EventArgs e)* и установим точку останова в строке *var url = HttpContext.Current.Request.Url;*, щелкнув поле слева или нажав F9. Рис. 10-2 демонстрирует, как это выглядит в редакторе.

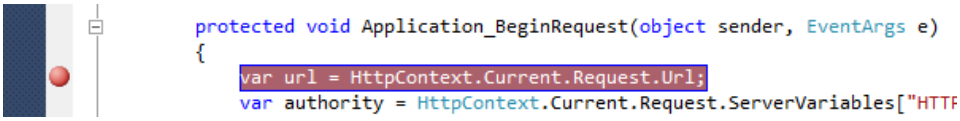


Рис. 10-2 Создание точки останова

158. Нажмем F5, чтобы запустить приложение в режиме отладки. В панели задач должен быть отражен запуск Веб-сервера разработчика, откроется новое окно браузера, и приложение немедленно прекратит выполнение в только что созданной точке останова. Если окно Breakpoints (Точки останова) не появилось на экране даже после запуска приложения в режиме отладки, в меню Debug (Отладка) выберите опцию Windows (Окна) и затем Breakpoints, или используйте комбинацию горячих клавиш Ctrl+D+B.

Теперь на экран должно быть выведено окно Breakpoints, как показано на рис. 10-3.

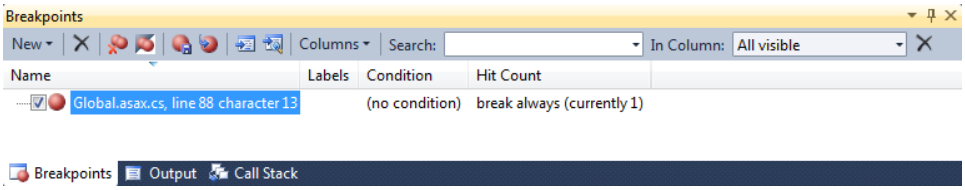


Рис. 10-3 Окно Breakpoints

159. В том же методе добавьте еще три точки останова. Редактор кода и окно Breakpoints должны выглядеть, как показано на рис. 10-4.

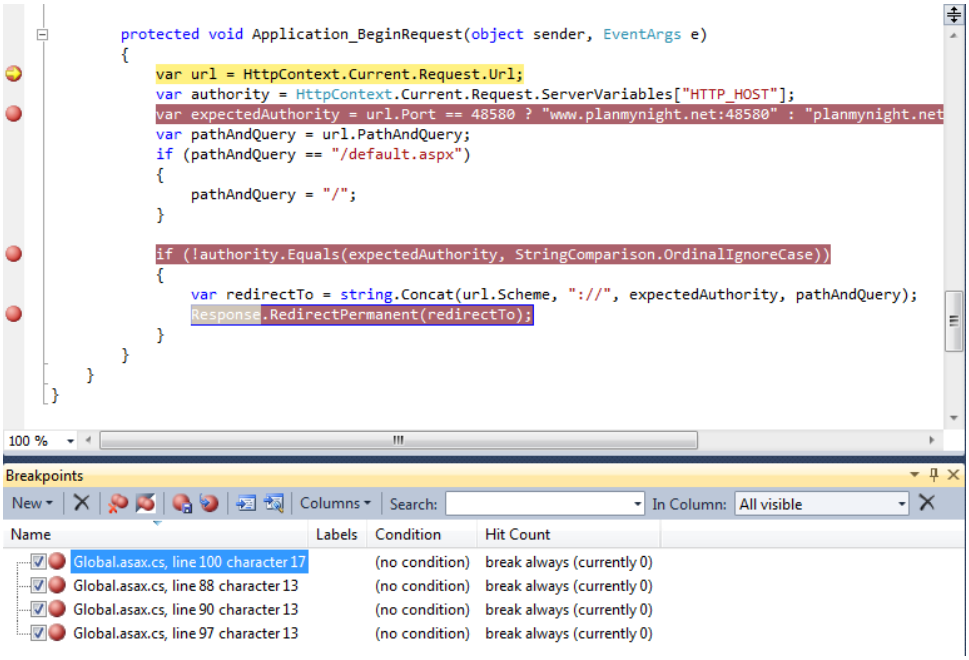


Рис. 10-4 Редактор кода и окно Breakpoints с тремя новыми точками останова

Visual Studio 2008 Внимательный читатель и профессиональный разработчик, часто использовавший Visual Studio 2008, вероятно, заметил в этом упражнении в окне Breakpoints ряд новых кнопок и полей. Напомним, как это окно выглядело в Visual Studio 2008 (рис. 10-5).

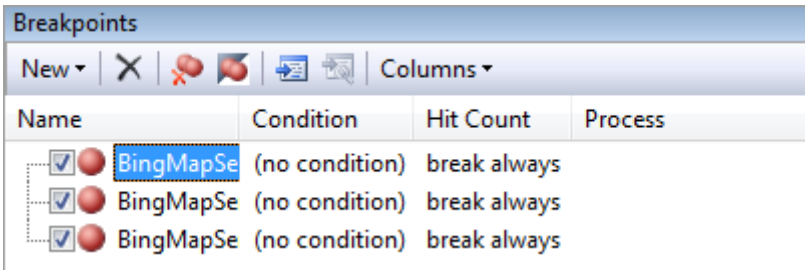


Рис. 10-5 Окно Breakpoints в Visual Studio 2008

160. Обратите внимание, в окне Breakpoints теперь имеется столбец Labels (Метки), что упрощает индексацию и поиск точек останова. Это на самом деле замечательная и очень полезная возможность, появившаяся в Visual Studio 2010. Чтобы воспользоваться ею, достаточно щелкнуть правой кнопкой точку останова в окне Breakpoints и выбрать Edit Labels (Редактировать метки) или использовать сочетание горячих клавиш Alt+F9, L. Это показано на рис. 10-6.

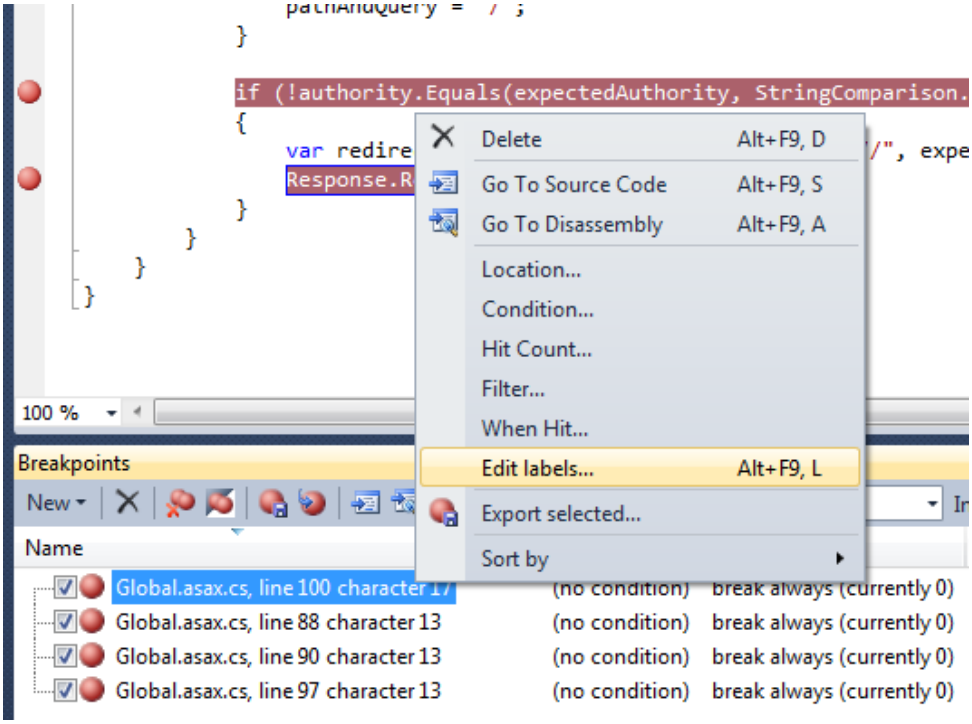


Рис. 10-6 Опция Edit Labels

161. В окне Edit Breakpoint Labels (Редактор меток точек останова) добавьте метки для выбранной точки останова (первой в окне Breakpoints). В текстовом поле Type a New Label (Введите новую метку) введите **ContextRequestUrl** и щелкните Add. Повторите эту операцию для следующей точки останова и введите имя метки **url**. Когда все готово, щелкните OK. На рис. 10-7 представлено, как выглядит окно Edit Breakpoint Labels в момент ввода меток, и окно Breakpoints после выполнения этих двух операций.

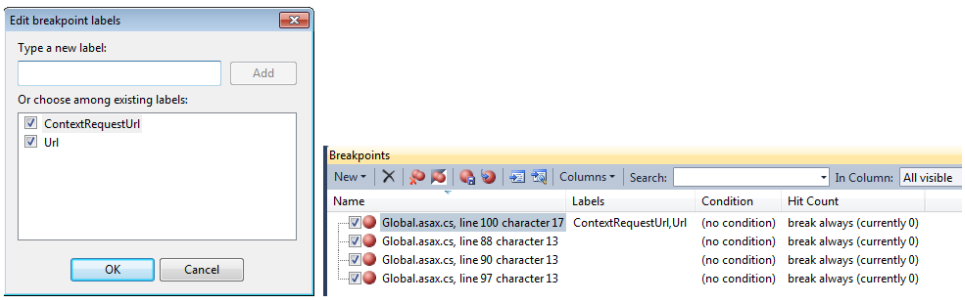


Рис. 10-7 Добавление меток, которые отображаются в окне Breakpoints

> **Примечание** Также вышеописанные задачи можно выполнить, щелкнув правой кнопкой точку останова в поле слева и выбрав в появившемся меню опцию Edit Labels.

> **Примечание** При добавлении меток для новой точки останова есть возможность выбрать любую из существующих меток, которые перечислены в области Or Choose Among Existing Labels (Или выбрать из существующих меток) диалогового окна Edit Breakpoint Labels (рис. 10-7).

162. Любым из рассмотренных способов добавляем метки для каждой точки останова. После этого окно Breakpoints должно выглядеть, как представлено на рис. 10-8.

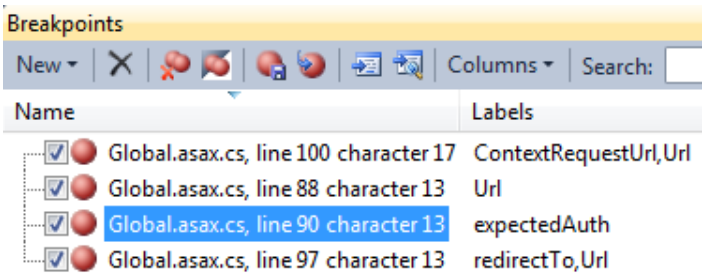


Рис. 10-8 Окно Breakpoints со всеми метками

В ходе отладки больших объемов кода очень пригодится возможность применения фильтра к отображаемому списку точек остановка. Именно это позволяет делать новая функция поиска в Visual Studio 2010.

163. Чтобы увидеть возможность поиска в действии, просто введите **url** в строку поиска и получите список, включающий только точки останова, в метках которых имеется строка *url*.

    При групповой разработке, когда в одном проекте занято множество разработчиков и тестировщиков, над одними и теми же ошибками часто работают два человека одновременно. В Visual Studio 2008 этим двум сотрудникам приходилось бы сидеть рядом, обмениваться мгновенными снимками экрана или пересылать друг другу номера строк кода, в которых требуется поставить точки останова, чтобы уточнить, с каким фрагментом кода работать при отладке конкретной ошибки.

> **Важно** Еще одним замечательным дополнением к управлению точками останова в Visual Studio 2010 является появившаяся возможность экспортировать их в файл, который затем можно переслать коллеге. Из этого файла точки останова могут быть импортированы в другую среду. Эта возможность также пригодится, если требуется распространить точки останова на несколько компьютеров. Далее мы рассмотрим, как это делается.

164. В окне Breakpoints щелкните кнопку Export (Экспортировать) для экспорта точек останова в файл и сохраните этот файл на рабочем столе. Назовите файл **breakexports.xml**.

165. Удалите все точки останова, либо щелкнув кнопку Delete All Breakpoints Matching The Current Search Criteria (Удалить все точки останова, соответствующие текущему критерию поиска), либо выбрав все точки останова и нажав кнопку Delete The Selected Breakpoints (Удалить выбранные точки останова). Удаляем точки останова с целью моделировать их совместное использование двумя разработчиками или распространение их на два компьютера.

166. Теперь выполнител импорт точек останова, нажав кнопку Import (Импортировать) и загрузив их с рабочего стола. Обратите внимание, что все точки останова со всеми свойствами опять появились в среде разработки. Для целей данной главы удалите все точки останова.

> **В Visual Studio 2008** Уже в Visual Studio 2008 была предложена, а в Visual Studio 2010 доработана, поддержка JavaScript и jQuery. Уже в Visual Studio 2008 все было замечательно, но интеграция в Visual Studio 2010 намного более производительная, и разработчик может просто воспользоваться ею, без всяких дополнительных операций.

Проверка данных

Всем известно, как много времени при отладке приложения может уйти на проверку содержимого переменных, аргументов и пр. Помните, еще совсем недавно, когда отладчики не были реальностью или

находились в зачаточном состоянии, как много выражений *printf* или *WriteLn* приходилось писать для проверки содержимого различных элементов данных? Хотя, возможно вы и не помните, вероятно, вы не так стары, как мы.

Visual Studio 2005 Уже Visual Studio 2005 предлагала настоящий отладчик с новой функциональностью, который был еще более доработан в Visual Studio 2008, что являло собой значительный прогресс по сравнению с временами, когда приходилось писать все виды выражений в консоли. Новые средства визуализации данных позволяли видеть форматированный XML, а не просто длинную строку кода. Более того, они представляли массивы в более удобном для восприятия виде – как список элементов и их индексов – и чтобы увидеть это, достаточно было навести курсор мыши на объект. Пример представлен на рис. 10-9.

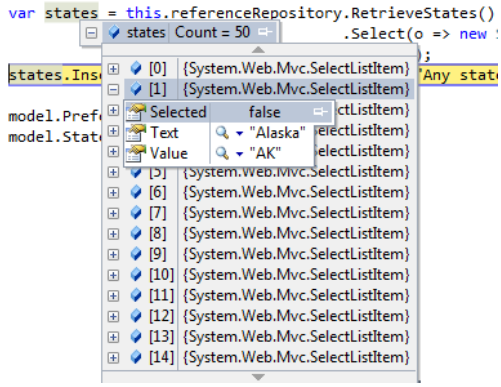


Рис. 10-9 Представление в виде коллекции в отладчике в Visual Studio 2008

Visual Studio 2008 Visual Studio 2008 впервые была предложена визуализация новых типов элементов данных. Самой замечательной и заметной стала возможность просматривать результаты выражения LINQ с помощью таких элементов отладчика, как всплывающие подсказки, окно Locals (Локальные переменные) и окна Watch (Контрольные значения) или QuickWatch (Контрольное значение). Как и для любого другого элемента – но иметь такую возможность для запроса LINQ особенно приятно – переменную LINQ можно скопировать и вставить в окно отладчика. Не забывайте, что чтобы вывести на экран результаты запроса, отладчик должен обработать запрос. Обратите внимание на такие вещи, как побочные эффекты или разница в производительности при раскрытии вложенных объектов.

Все эти техники визуализации данных в виде всплывающей подсказки по-прежнему доступны в Visual Studio 2010, но при этом предлагается ряд нововведений, которые повышают эффективность всплывающих подсказок по данным. В сочетании с улучшениями всплывающих подсказок в Visual Studio 2010 появилась еще одна возможность – поддержка одновременной работы с несколькими мониторами. Всплывающие подсказки по данным очень полезны для разработчика. А наличие возможности поместить их на второй экран может намного облегчить отладку, позволяя всегда иметь перед глазами данные, которые всегда должны быть в контексте, на втором мониторе. Рассмотрим поэтапно, как использовать эти возможности:

167. В файле Global.ascx.cs установите точки останова в строках 89 и 91 – строках, начинающихся с *var authority* и *var pathAndQuery*, соответственно.

168. Поэкспериментируем с новыми возможностями всплывающих подсказок по данным. Запустим отладчик, нажав F5. Когда отладчик встречает первую точку останова, наведите курсор на слово *url* и щелкните значок вешки, как показано на рис. 10-10.

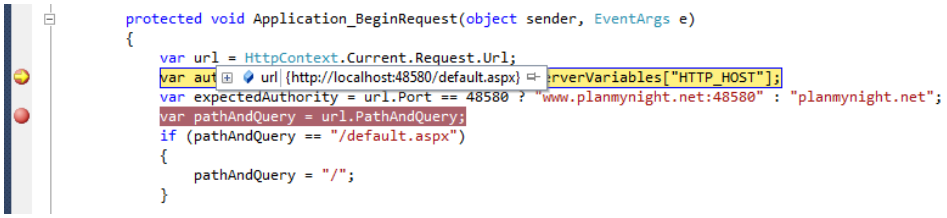
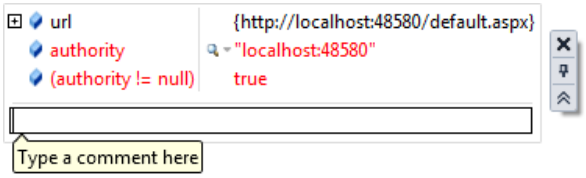


Рис. 10-10 Новая возможность всплывающих подсказок по данным – вешка

169. Справа от строки кода должен располагаться закрепленная всплывающая подсказка по данным (как показано на рис. 10-11 слева). Если навести курсором мыши на эту подсказку, появится панель управления всплывающей подсказки по данным (как показано на рис. 10-11 справа).

Рис. 10-11 Слева – закрепленная всплывающая подсказка по данным, и справа – ее панель управления

> **Примечание** Также в поле точек останова должна отображаться голубая вешка, указывающая на то, что данная всплывающая подсказка закреплена. Вешка выглядит так: . Поскольку в этой строке располагается точка останова, фактически вешка находится под ней. Чтобы увидеть вешку, просто переключите точку останова, щелкнув ее в поле. Один щелчок снимет точку останова, второй – вернет ее назад.

> **Примечание** По щелчку указывающей вниз двойной стрелки на панели управления всплывающей подсказки по данным появляется строка, в которую можно ввести комментарий к этой подсказке, как показано на рис. 10-12. Также можно полностью удалить всплывающую подсказку, нажав кнопку X на панели управления всплывающей подсказки.
>
> 
>
>
> *Рис. 10-12 Вставка комментария к всплывающей подсказке по данным*

170. Еще одно замечательное свойство новой всплывающей подсказки по данным в том, что есть возможность вставить любое выражение, и оно будет вычислено прямо в ходе сеанса отладки. Например, щелкните правой кнопкой имя всплывающей подсказки, в данном случае это *url*, выберите Add Expression (Добавить выражение), введите **authority** и добавьте еще одно, например, **(authority != null)**. Выражение тут же вычисляется и будет вычисляться на протяжении всего сеанса отладки при каждой остановке отладчика на этих точках останова. На данном этапе сеанса отладки эти выражения должны быть равны *null* и *false*, соответственно.

171. Нажмите F10, чтобы выполнить строку, в которой остановился отладчик, и проверьте всплывающую подсказку для url и оба выражения. Они должны содержать значения, соответствующие текущему контексту. На рис. 10-13 представлено, как все это выглядит в действии.

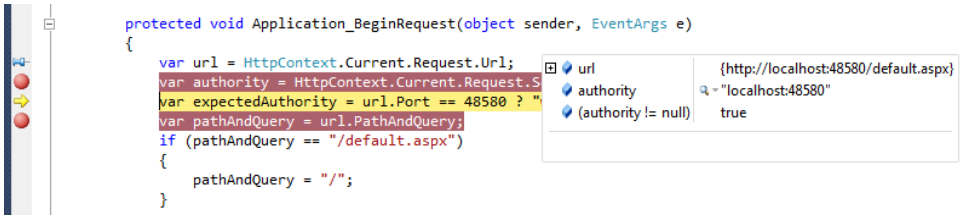


Рис. 10-13 Закрепленная всплывающая подсказка для url с двумя вычисленными выражениями

172. Замечательно иметь возможность просматривать мини-окно в том месте, где оно имеет значение – прямо там, где выполняется код – но оно перекрывает отлаживаемый исходный код. Окно всплывающей подсказки можно перетянуть в любую точку редактора кода. Пример показан на рис. 10-14.

Рис. 10-14 Убираем закрепленное окно подсказки с исходного кода

173. Закрепленное окно подсказки остается там, где оно закреплено, поэтому при переходе к другому файлу его не будет видно. Но в некоторых случаях окно всплывающей подсказки должно быть в поле зрения постоянно, например, для глобальных переменных, которые всегда в контексте, или для сценариев с использованием множества мониторов. Чтобы переместить окно подсказки, надо сначала открепить его, щелкнув вешку на панели управления всплывающей подсказки. Окно изменит цвет и станет желтым, что

свидетельствует о возможности перемещать его в любое место, например, поверх Solution Explorer, на второй монитор, на рабочий стол или в любое другое окно. Пример можно увидеть на рис. 10-15.

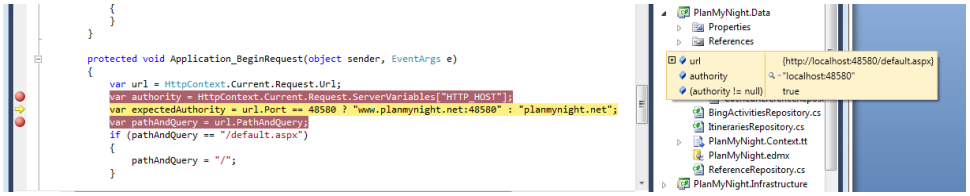


Рис. 10-15 Откреплённая всплывающая подсказка поверх Solution Explorer и рабочего стола Windows

Примечание Когда отладчик переходит к другому файлу или методу, и окно всплывающей подсказки не закреплено, оно может отображать элементы, не соответствующие текущему контексту. Тогда окно всплывающей подсказки будет выглядеть, как показано на рис. 10-16. Можно щелкнуть кнопку  , чтобы отладчик повторил попытку вычислить значение элемента. Но если элемент не имеет значения в данном контексте, весьма вероятно, что это ни к чему не приведет.

Рис. 10-16 Незакрепленное окно подсказки с элементами, не входящими в текущий контекст

Примечание При попытке закрепить окно всплывающей подсказки вне редактора будет выведено сообщение об ошибке, как показано на рис. 10-17.

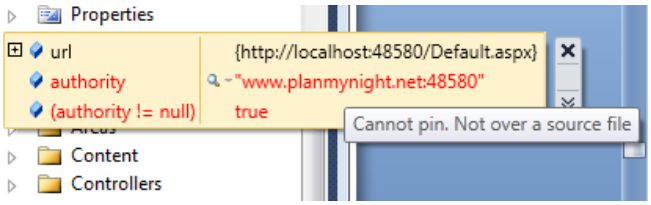


Рис. 10-17 Сообщение об ошибке, которое появляется при попытке закрепить окно всплывающей подсказки вне редактора кода

Примечание Используемый вами номер порта может отличаться от того, который представлен на приводимых здесь снимках экрана. Это нормально, поскольку Веб-сервер, включенный с Visual Studio, выбирает порт для использования случайным образом.

Примечание Закрепить можно и любой дочерний элемент закрепленного элемента. Например, разверните содержимое элемента url, нажав знак плюс (+). Вы увидите, что имеется возможность закрепить его дочерний элемент, как показано на рис. 10-18.

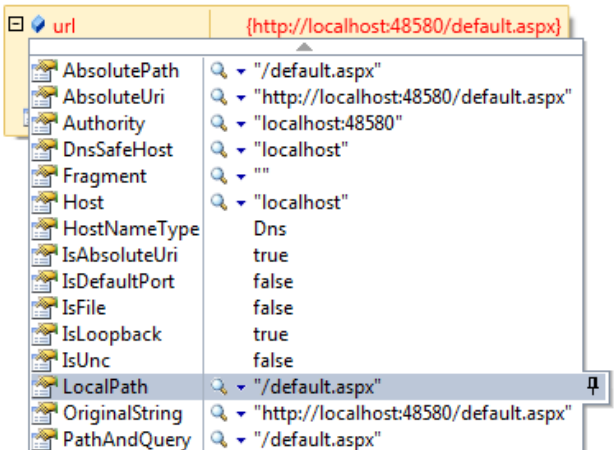


Рис. 10-18 Закрепленный дочерний элемент элемента url в окне всплывающей подсказки

174. Прежде чем остановить выполнение отладчика, вернитесь в Global.ascx.cs, если еще не сделали этого, и повторно закрепите окно всплывающей подсказки. После этого завершите сеанс отладки, щелкнув кнопку Stop Debugging (Остановить отладку) на панели инструментов отладки ( ) или нажав клавиши Shift+F5. Теперь если провести курсором мыши по голубой вешке в поле точек останова, можно увидеть значения последнего сеанса отладки. Такая функциональность является приятным улучшением по сравнению с окном контрольных значений. На рис. 10-19 показано, что должно быть выведено на экран.

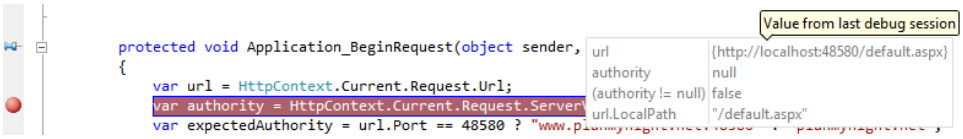


Рис. 10-19 Значения последнего сеанса отладки для закрепленной всплывающей подсказки

Примечание Как и точки останова, подсказки по данным могут экспортироваться или импортироваться. Для этого в меню Debug предусмотрены опции Export DataTips и Import DataTips, соответственно.

Использование отладчика минидампа

При эксплуатации реальных проектов часто возникают ситуации, когда группе поддержки продукта приходится работать с минидампом. Кроме описаний и шагов воспроизведения ошибок, это может быть единственным средством для отладки приложений заказчика. В Visual Studio 2010 отладка минидампа доработана и улучшена.

Visual Studio 2008 В Visual Studio 2008 для отладки управляемого кода или файлов минидампа приходилось использовать расширение – инструмент SOS, который загружался в отладчик с помощью окна интерпретации (Immediate). Отладчик подключался с разрешенной отладкой как управляемого, так и неуправляемого кода. При этом никакие данные не передавались ни в стек вызовов, ни в окно локальных переменных (Locals). При отладке файлов минидампа команды для инструмента SOS вводились в окне интерпретации. Отладка приложений, написанных неуправляемым кодом, выполнялась с использованием обычных окон и инструментов отладки. Познакомиться с этими вопросами более подробно или просто обновить их в памяти можно в колонке *Bug Slayer* (Убийца ошибок) журнала MSDN по адресу *http://msdn.microsoft.com/en-us/magazine/cc164138.aspx*.

Теперь рассмотрим, как был доработан отладчик минидампа. Для начала смоделируем сбой, на основании которого можно будет сформировать файл минидампа:

175. В Solution Explorer в проекте PlanMyNight.Web измените имя файла Default.aspx на **DefaultA.aspx** (обратите внимание на букву *A*, добавленную после слова «Default»).

176. Убедитесь, что убрали все точки останова из проекта. Для этого откройте окно Breakpoints и удалите все имеющиеся там точки останова любым из рассмотренных ранее в этой главе способом.

177. Нажмите F5, чтобы начать отладку приложения. В зависимости от производительности компьютера вскоре после завершения процесса сборки должно быть выведено необработанное исключение типа *HttpException*. Несмотря на то что в данном случае ошибка проста, последовательно пройдем все шаги создания и отладки файла минидампа. На рис. 10-20 показано, что должно отображаться на экране в данный момент.

Рис. 10-20 Необрабатываемое исключение, которое должно быть сформировано

178. Пора создавать файл минидампа для этого исключения. Перейдите в меню Debug и выберите Save Dump As (Сохранить дамп как), как показано на рис. 10-21. Вы должны видеть имя процесса, из которого было сформировано исключение. В данном случае это процесс Cassini или Personal Web Server в Visual Studio. Оставьте предлагаемое имя файла (WebDev.WebServer40.dmp) без изменений и сохраните файл на своем рабочем столе. Обратите внимание, что создание файла минидампа может занять некоторое время, потому что его размер будет около 300 MB.

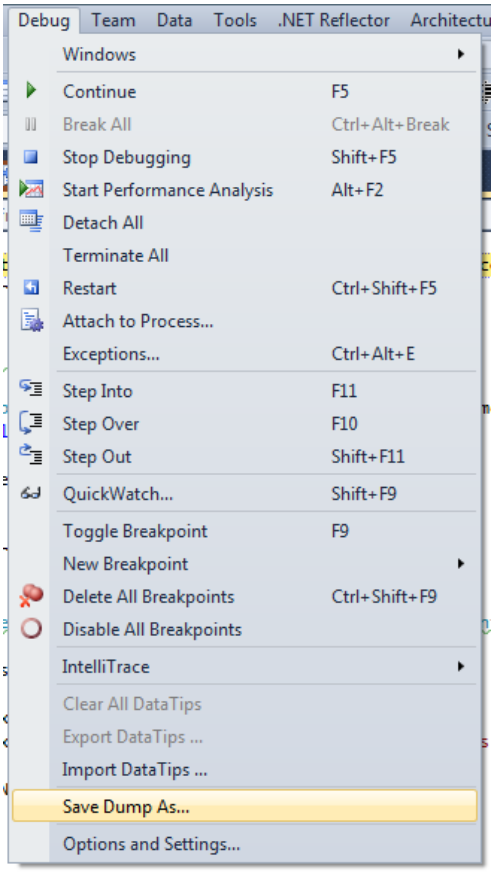


Рис. 10-21 Сохранение файла манидампа

179. Остановите отладку, нажав Shift+F5 или кнопку Stop Debugging.

180. Перейдите в меню File и закройте свое решение.

181. Чтобы загрузить свой файл манидампа WebDev.WebServer40.dmp, в меню File щелкните Open и выберите рабочий стол. При этом откроется страница Minidump File Summary (Сведения о файле минидампа), содержащая некоторые общие сведения об ошибке, которую мы пытаемся устранить (рис. 10-22). Еще до начала отладки эта страница предоставляет основные данные, такие как имя процесса, архитектура процесса, версия операционной системы, версия CLR, загруженные модули, а также некоторые действия, которые можно предпринять в этот момент. Прямо здесь можно задать пути к файлам символов. Очень удобно, что список модулей (Modules) включает версию и путь к модулю, что облегчает поиск символов и исходного кода. Используется CLR версия 4.0, т.е. отладку можно выполнять в Visual Studio 2010.

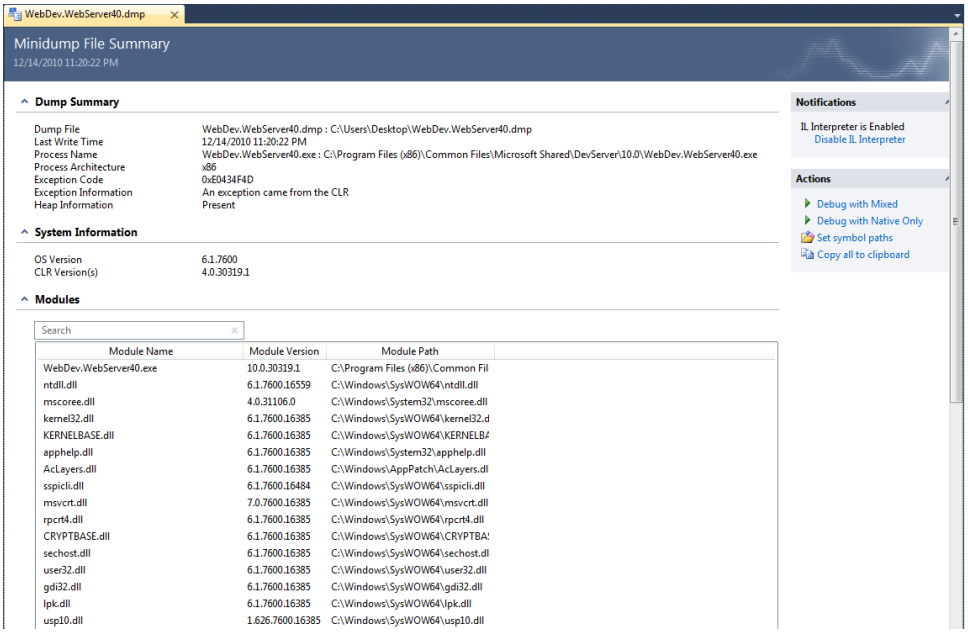


Рис. 10-22 Страница сведений о файле минидампа

182. Чтобы начать отладку, в списке Actions (Действия) в правой части страницы Minidump File Summary щелкните ссылку Debug With Mixed (Отладка в смешанном режиме).

183. Практически сию минуту на экран должен быть выведен первый этап обработки исключения, как показано на рис. 10-23. В данном случае сообщается, какая ошибка произошла, хотя это не всегда так. Продолжим, нажав кнопку Break (Прервать).

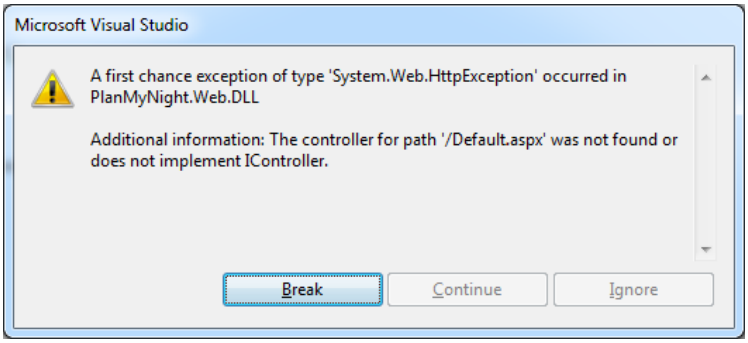


Рис. 10-23 Первый этап обработки исключения

184. Вы должны видеть подсвеченную зеленым строку, которая указывает на то, какая команда привела к формированию исключения. Если взглянуть на исходный код, в окне Autos (Автоматические значения) можно увидеть, что переменная *controllerExport* (Экспорт контроллера) имеет значение *null*. И прямо перед этим мы задали, что если эта переменная *null* и файл для загрузки не найден, должно формироваться исключение *HttpException*. В данном случае, как видно в окне Locals в переменной *controllerName* (Имя контроллера), должен загружаться файл Default.aspx. В окнах Locals и Autos можно также видеть множество других переменных, объектов и прочих элементов, отражающих текущий контекст. Здесь только один вызов относится к нашему коду, поэтому стек вызовов показывает, что весь код до и после этого вызова является внешним для нашего процесса. Если бы цепочка вызовов нашего кода была длиннее, мы могли бы перемещаться назад и вперед по коду и просматривать переменные. На рис. 10-24 показано сводное представление всего вышеописанного.

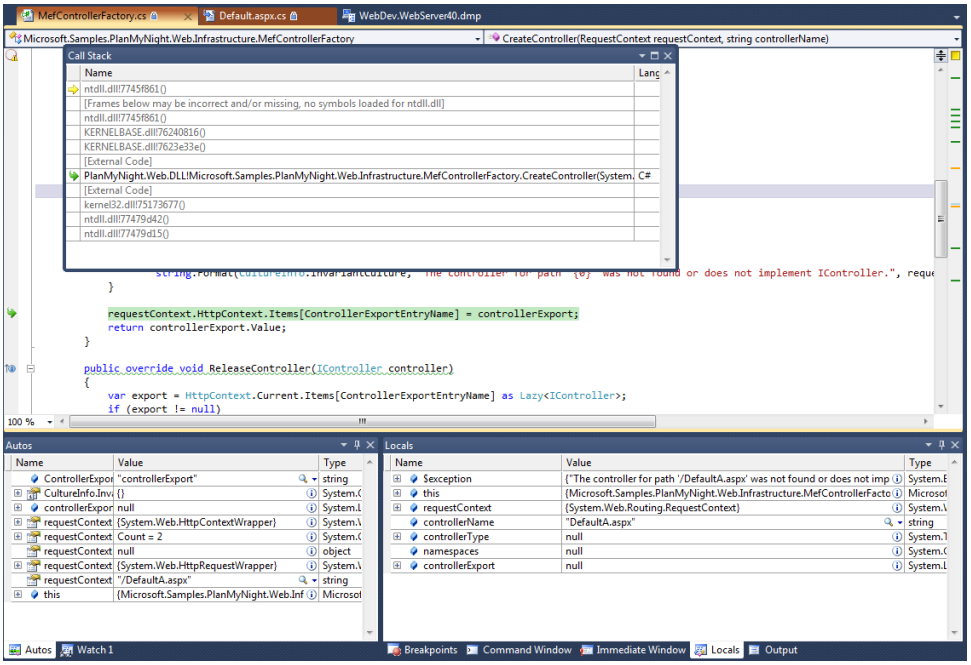


Рис. 10-24 Окна Autos, Locals и Call Stack и следующая команда к выполнению

185. Итак, ошибка найдена. Останавливаем отладку нажатием клавиш Shift+F5 или кнопки Stop Debugging. Чтобы исправить ошибку, опять загружаем решение PlanMyNight и исправляем имя файла **default.aspx**. Переходим в меню Build, выбираем Rebuild Solution (Выполнить повторную сборку решения) и выполняем повторную сборку решения. Затем нажимаем F5, и приложение должно опять исправно работать.

Преобразования Web.Config

Следующая возможность, хотя и незначительная, но она порадует многих разработчиков, потому что сбережет им время при отладке. Преобразования Web.Config позволяют сохранять файлы преобразований, отражающие разницу между средами отладки и производственной эксплуатации. Например, часто в разных средах используются разные строки подключения. Таким образом, создание файлов преобразований – ASP.NET предоставляет средства для изменения (преобразования) файлов web.config – обеспечит использование соответствующих строк для соответствующих сред. Более подробно о том, как это реализуется, рассказывает следующая статья на сайте MSDN: *http://go.microsoft.com/fwlink/?LinkId=125889*.

Разработка модульных тестов

Инфраструктура и средства модульного тестирования в Visual Studio 2010 Professional остались неизменными. В других редакциях Visual Studio 2010 в управление тестами и средства тестирования внесены действительно заметные изменения. В Visual Studio 2010 Premium и Visual Studio 2010 Ultimate предлагаются такие возможности, как модульные тесты пользовательского интерфейса, IntelliTrace и Microsoft Test Manager 2010. Все возможности Управления жизненным циклом приложения подробно рассматриваются на сайте MSDN в статье *http://msdn.microsoft.com/en-us/library/ee789810(VS.100).aspx*.

> **Visual Studio 2008** Чтобы создавать и выполнять тесты, поставляемые в комплекте с Visual Studio 2008, требовалось иметь Visual Studio 2008 Team System или Visual Studio 2008 Team Test. Альтернативным вариантом в те времена было использование продукта сторонних производителей, такого как Nunit.

В данной части главы мы просто покажем, как добавлять модульный тест для одного из классов приложения Plan My Night. Не будем тратить время на определение того, что такое модульный тест или что он должен включать. Лучше рассмотрим, как добавлять и выполнять тесты в Visual Studio 2010.

Добавим в приложение Plan My Night модульные тесты для надстройки Print Itinerary. Для создания модульных тестов откроем решение, находящееся в папке сопроводительных материалов книги. Если не помните, как это делается, можете вернуться к первой странице данной главы за инструкциями. Когда решение открыто, просто последовательно выполните следующие шаги:

186. В Solution Explorer разверните проект PlanMyNight.Web и папку Helpers. Щелкните двойным щелчком файл ViewHelper.cs, чтобы открыть его в редакторе коды. Взгляните на рис. 10-25 и убедитесь, что делаете все правильно.

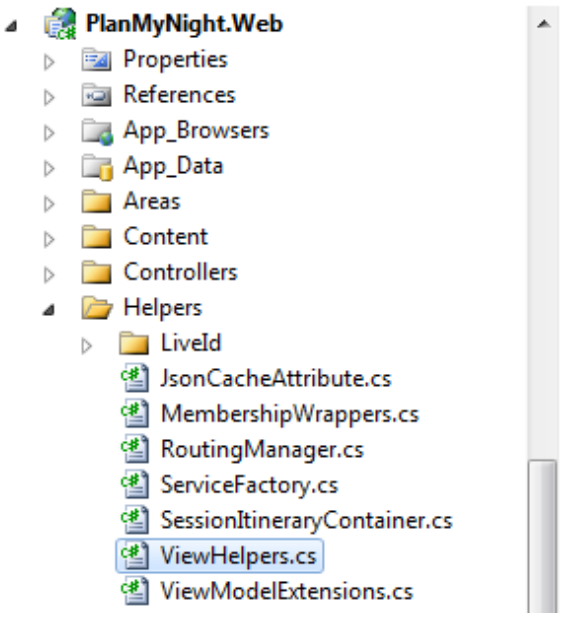


Рис. 10-25 Проект PlanMyNight.Web и файл ViewHelper.cs в Solution Explorer

187. Редактор кода позволяет добавлять модульные тесты двумя разными способами. Можно щелкнуть правой кнопкой имя класса или имя метода и выбрать в появившемся меню Create Unit Tests (Создать модульные тесты). Также можно перейти в меню Test (Тест) и выбрать New Test (Новый тест). Рассмотрим первый способ создания модульных тестов, при котором Visual Studio автоматически формирует некоторый исходный код. Щелкаем правой кнопкой метод *GetFriendlyTime* (Представить время в удобном для восприятия виде) и выбираем Create Unit Tests (рис. 10-26).

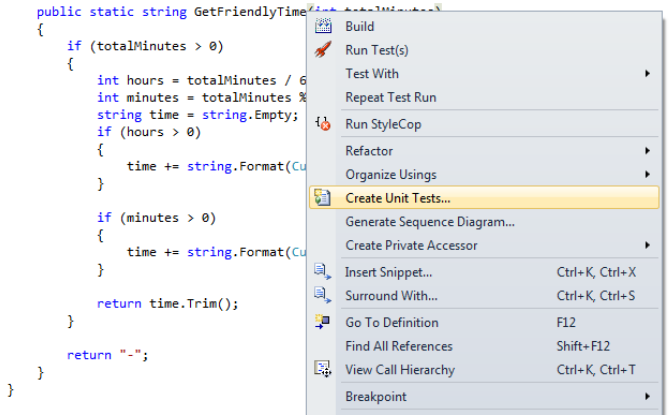


Рис. 10-26 Контекстное меню для создания модульных тестов

188. По нажатию Create Unit Tests на экран выводится диалоговое окно, в котором по умолчанию предлагается метод, выбранный нами в этом классе. Перейдем к полю с раскрывающимся списком внизу диалогового окна и выберем в нем, где будут создаваться модульные тесты. В данном случае это PlanMyNight.Web.Tests. Если есть необходимость создать новый проект для тестов, просто выбираем в этом списке Create A New Visual C# Test Project (Создать новый тестовый проект Visual C#). На рис. 10-27 представлено, как все это должно выглядеть.

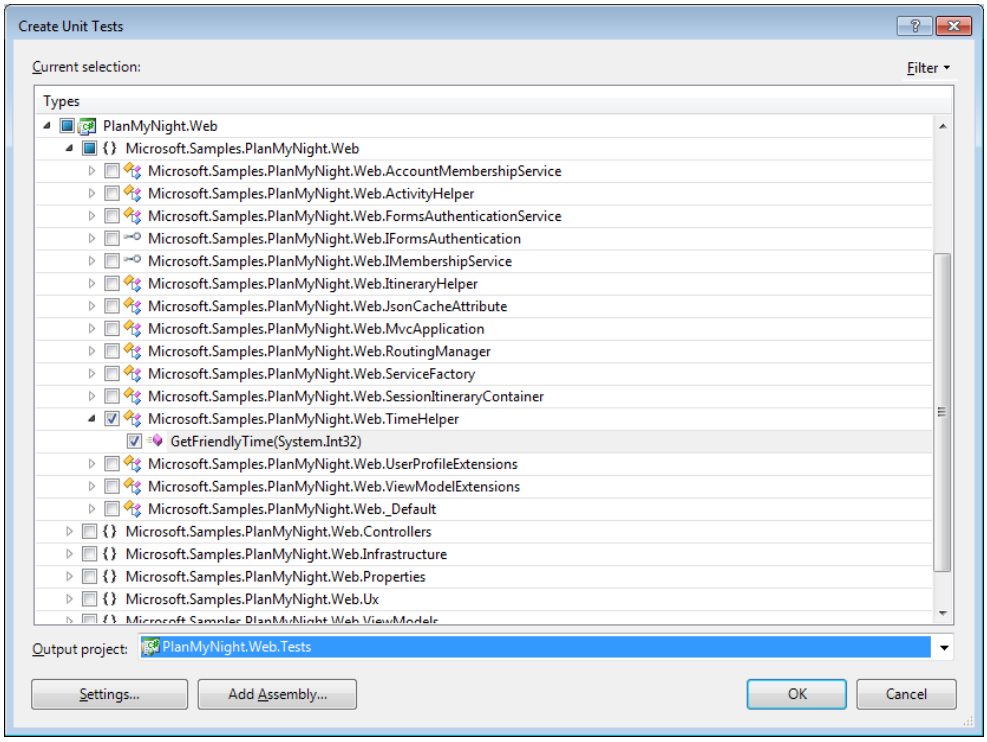


Рис. 10-27 Выбор метода, для которого требуется создать модульный тест

189. По щелчку OK диалоговое окно переключается в режим формирования тестового случая, отображается индикатор хода выполнения. По завершении этого процесса создается файл *TimeHelperTest.cs*, содержащий автоматически сформированные заглушки кода, которые мы будем дорабатывать.

190. Удалите этот метод и его атрибуты, потому что для него мы создадим три тестовых случая. Удаляем следующий код:

```
/// <summary>
/// Тест для GetFriendlyTime
///</summary>
// TODO: Гарантированно обеспечить, что атрибут UrlToTest содержит URL
// страницы ASP.NET (например, http://.../Default.aspx).
// Это необходимо, чтобы модульный тест выполнялся на Веб-сервере
// независимо от того, выполняется ли тестирование страницы,
// Веб-сервиса или WCF-сервиса.
 [TestMethod()]
[HostType("ASP.NET")]
[AspNetDevelopmentServerHost("C:\\Users\\Patrice\\Documents\\Chapter
10\\code\\PlanMyNight.Web", "/")]
[UrlToTest("http://localhost:48580/")]
public void GetFriendlyTimeTest()
{
    int totalMinutes = 0; // TODO: Инициализировать с соответствующим значением
    string expected = string.Empty; // TODO: Инициализировать с соответствующим
значением
    string actual; actual = TimeHelper.GetFriendlyTime(totalMinutes);
    Assert.AreEqual(expected, actual);
    Assert.Inconclusive("Verify the correctness of this test method[1].");
}
```

191. Добавляем три простых тестовых случая для проверки ключевых сценариев, используемых в Plan My Night. Для этого вставляем следующий исходный код прямо под атрибутами метода, оставшимися после удаления блока кода в шаге 5:

```
[TestMethod]
public void ZeroReturnsSlash()
{
    Assert.AreEqual("-", TimeHelper.GetFriendlyTime(0));
}
```

[1] Проверяем правильность этого тестового метода (*прим. переводчика*).

```
[TestMethod]
public void LessThan60MinutesReturnsValueInMinutes()
{
    Assert.AreEqual("10m", TimeHelper.GetFriendlyTime(10));
}

[TestMethod()]
public void MoreThan60MinutesReturnsValueInHoursAndMinutes()
{
    Assert.AreEqual("2h 3m", TimeHelper.GetFriendlyTime(123));
}
```

192. В проекте PlanMyNight.Web.Tests создадим папку для решения и назовем ее **Helpers**. Перенесем файл TimeHelperTests.cs в эту папку. После этого проект должен выглядеть, как показано на рис. 10-28.

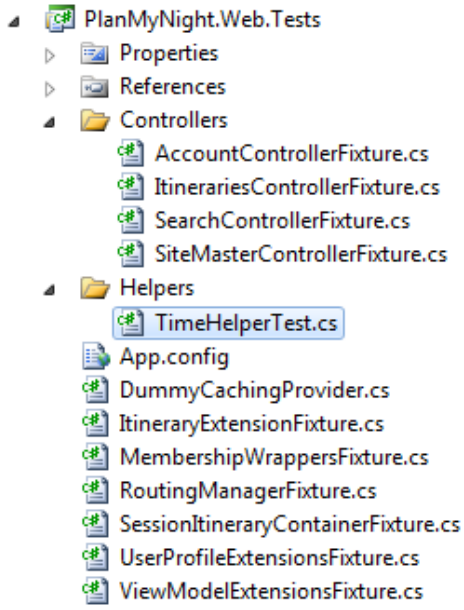


Рис. 10-28 TimeHelperTest.cs в папке Helpers

193. Пора выполнить только что созданные тесты. Чтобы выполнить только те тесты, которые мы только что создали, перейдем в редактор кода и наведем курсором на класс *public class TimeHelperTest*. Теперь можно либо зайти в меню Test, выбрать Run и затем Tests In Current Context (Тесты в текущем контексте); либо выполнить все то же самое посредством сочетания горячих клавиш CTRL+R, T (рис. 10-29).

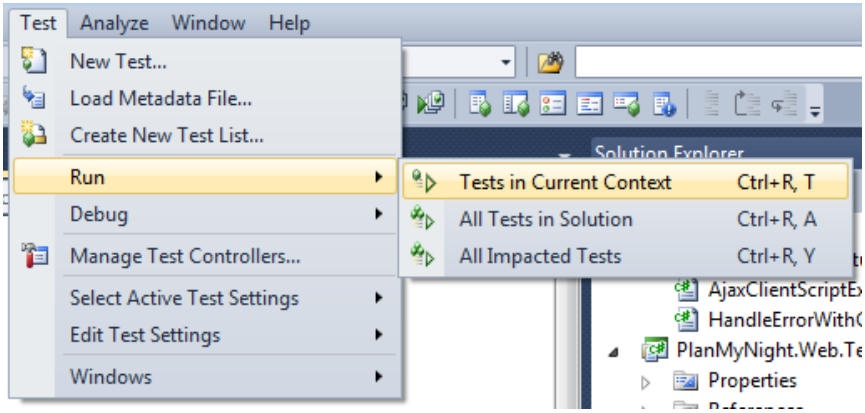


Рис. 10-29 Меню выполнения теста

194. Это действие обеспечит выполнение только трех созданных нами тестов. Внизу редактора должно появиться окно Test Results (Результаты тестирования) с результатами выполнения этих тестов (рис. 10-30).

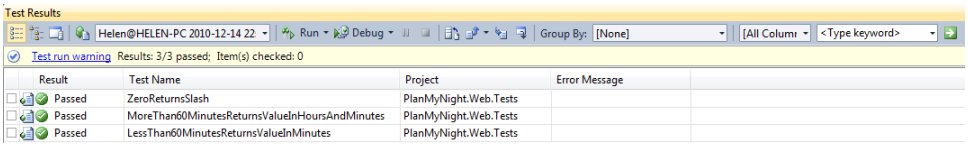


Рис. 10-30 Окно Test Results с результатами выполнения вновь созданных тестов

> **Дополнительные сведения** В зависимости от заданных параметров поведение при выборе опции Tests In Current Context может быть разным. Например, если выбран тестовый метод, такой как *ZeroReturnsSlash*, будет выполнен только этот тестовый случай. Но если щелкнуть вне класса теста, будут выполнены все тестовые случаи, что эквивалентно выбору опции All Tests In Solution (Все тесты решения).

Новое окно Threads

Появление многоядерных процессоров и многочисленных инструментов, предоставляемых языками программирования для работы с ними, обусловило новую проблему: отладка многопоточных приложений. Новое окно Threads (Потоки) обеспечивает разработчику возможность приостанавливать потоки и выполнять поиск артефактов по стеку вызовов, аналогично тому, как это происходило при использовании знаменитого Process Monitor от SysInternals (*http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx*). Окно Threads можно открыть во время отладки приложения из меню Debug, нажав опцию Windows And Threads (Окна и потоки). На рис. 10-31 представлено окно Threads во время отладки приложения Plan My Night.

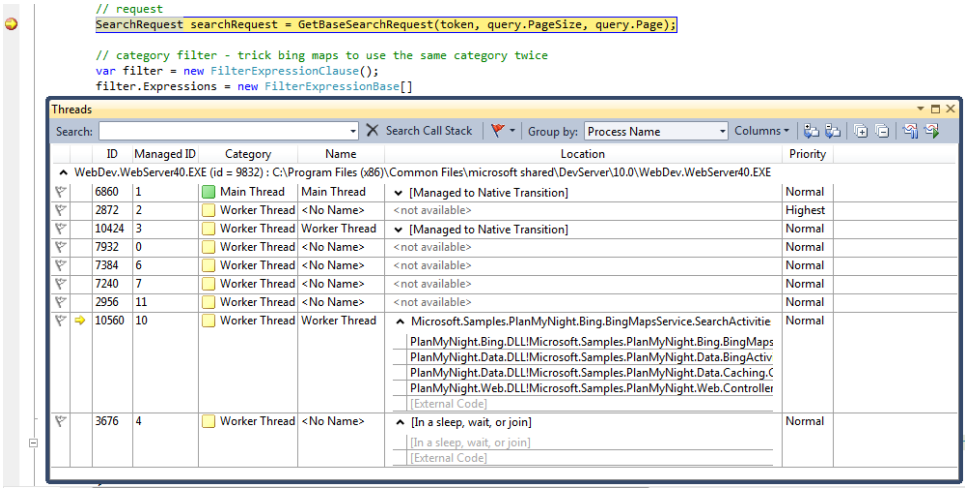


Рис. 10-31 Окно Threads во время отладки Plan My Night

Окно Threads позволяет приостанавливать потоки и вновь запускать их, когда требуется. Это может быть действительно полезным, когда необходимо изолировать конкретные эффекты. Можно выполнять отладку как управляемого, так и неуправляемого кода. Эта новая возможность отладчика в Visual Studio 2010, безусловно, понравится разработчикам, которые столкнуться с необходимостью отладки приложений, использующих потоки.

> **Visual Studio 2008** В Visual Studio 2008 окно для отладки потоков, наконец, соответствовало своему названию. Но не было возможности фильтрации, группировки, поиска по стеку вызовов и расширения. Порядок столбцов был фиксирован. Получение масок сходства, имен процессов и управляемых ID было невозможно без специального инструмента.

Заключение

В данной главе мы научились управлять сеансами отладки с помощью улучшенных возможностей работы с точками останова и использования новых техник проверки и визуализации данных. Также мы узнали, как с помощью отладки минидампа и новых инструментов решать проблемы, возникающие при работе приложений, не выводя их из эксплуатации. В этой главе было рассмотрено, как повысить качество кода посредством модульных тестов и как сделать это в Visual Studio 2010 Professional. Компьютеры с множеством процессоров или с многоядерными процессорами в наши дни стали нормой, как и многопоточные приложения. Поэтому особенно приятным является тот факт, что Visual Studio 2010 предлагает специальные инструменты отладки для выявления проблем в многопоточных приложениях.

Наконец, в ходе главы было показано, насколько в Visual Studio 2010 Professional усовершенствован процесс отладки приложений в целом и какие инструменты получили в свое распоряжение профессиональные разработчики для отладки современных насыщенных разнообразнейшей функциональностью приложений. Прогресс по сравнению с Visual Studio 2008 очевиден. Примеры, рассмотренные в данной главе, лишь вкратце продемонстрировали, насколько выгоден с точки зрения экономии времени и средств переход к новой инфраструктуре отладки, но даже из них видно, Visual Studio 2010 не просто очередной этап, а значительная веха в развитии Visual Studio. Переход на Visual Studio 2010 обеспечит значительное повышение производительности разработки. С точки зрения возможностей отладки пропасть между Visual Studio 2008 и Visual Studio 2010 не так велика, чем между более ранними версиями.

Различные редакции Visual Studio 2010 предоставляют целый ряд замечательных улучшений, касающихся отладки и тестирования. Особо из них выделяется возможность IntelliTrace – *http://msdn.microsoft.com/en-us/library/dd264915(VS.100).aspx* – которая доступна только в Visual Studio 2010 Ultimate и Microsoft Test Manager. IntelliTrace намного улучшает условия работы групп тестирования при использовании Visual Studio 2010 и Visual Studio 2010 Team Foundation Server – *http://msdn.microsoft.com/en-us/library/bb385901(VS.100).aspx*.

Индекс

Об авторах

Кен Хайнс (Ken Haines) – специалист по разработке ПО, сотрудник отдела Разработки решений для конечных потребителей и онлайн (Consumer and Online Division) компании Майкрософт. Кен занимается распределенными приложениями в облаке и всегда рад помочь пользователям и партнерам найти правильное решение, отвечающее их нуждам.

За последние 12 лет он попробовал себя во многих ролях в различных отраслях, включая поставку Интернет-сервисов, спутниковые телекоммуникации, мониторинг сетей и Веб-аналитика. В связи с профессиональной деятельностью подчас ему приходилось забираться в очень удаленные точки земли, такие как Нуванут и Северно-Западные территории Канады.

Когда Кен не разрабатывает ПО, он занимается туризмом, катается на горных велосипедах, фотографирует природу, читает и проводит время с семьей. Проживает в Монро, штат Вашингтон.

Паскаль Паре (Pascal Paré) работает в Майкрософт с 2006 года в качестве специалиста по разработке ПО в группах разработки и тестирования. В настоящее время является специалистом по разработке ПО отдела Разработки решений для конечных потребителей и онлайн.

13 лет назад закончил Университет Лаваль в Квебеке, Канада, по специальности инженер по вычислительной технике. До Майкрософт работал тестировщиком и разработчиком в различных компаниях и отраслях (в том числе производство контрольно-измерительной аппаратуры волоконной оптики, телекоммуникации и разработка ПО для медицинского оборудования).

В свободное время занимается туризмом, катается на велосипеде и лыжах на Тихоокеанском Северо-Западе. Также любит готовить и путешествовать. Больше всего он любит сесть за руль своего Lotus Elise и погонять по проселочным дорогам вокруг Пьюджет-Саунда. Паскаль женат, в настоящее время проживает в Сиэтле.

Патрис Пелланд (Patrice Pelland) заведующий отделом разработки подразделения Разработки решений для конечных потребителей и онлайн Майкрософт. Руководит группой разработки, которая занимается внедрением новых технологий и пользовательских продуктов Майкрософт. Его страстью являются сложные распределенные системы и разработки для мобильных устройств; он помогает пользователям и партнерам по всему миру в полном объеме использовать преимущества, предлагаемые продуктами Майкрософт.

В течение последних 17 лет занимается разработкой ПО и как индивидуальный разработчик, и в качестве руководителя проектов в различных областях, включая Веб-разработку, создание инструментов разработки, оптико-волоконные телекоммуникации, авиацию, пищевую промышленность. Также он посвятил три года преподаванию теории вычислительных машин и разработки ПО в колледже в Канаде.

Когда не занят разработкой ПО, Патрис любит проводить время с семьей и друзьями, устраивает замечательные ужины с хорошей едой и напитками, путешествует, готовит, читает книги, интересуется автомобилями Porsche, смотрит хоккей и футбол, ходит в спорзал. Проживает с семьей в Саммамише, Вашингтон.