Moving to
Microsoft®
Visual Studio® 2010

PREVIEW CONTENT

Patrice Pelland,
Pascal Pare, and Ken Haines

PREVIEW CONTENT
This excerpt provides early content from a book currently in development, and is still in draft, unedited format. See additional notice below.

Introduction

Every time we get close to a new release of Microsoft Visual Studio we can feel the excitement in the developer community. This release of Visual Studio is certainly no different, but at the same time we can feel a different vibe. In November 2009, at Microsoft Professional Developer Conference in Los Angeles, participants had the chance to get their hands on the latest beta of this Visual Studio incarnation. The developer community started to see how different this release is compared to any of its predecessors. This might sound familiar, but Visual Studio 2010 constitutes, in our opinion, a big leap and is a true game changer in that it has been designed and developed from the core up.

Looking at posts in the MSDN forums and many other popular developer communities also reveals that many of you—professional developers—are still working in previous versions of Visual Studio. This book will show you how to move to Visual Studio 2010 and will try to explain why it's a great time to make this move.

Who Is This Book for?

This book is for professional developers who are working with previous versions of Visual Studio and are looking to make the move to Visual Studio 2010 Professional.

What Is the Book About?

The book is not a language primer, a language reference, or a single technology book. It's a book that will help professional developers move from previous versions of Visual Studio (starting with 2003 and on up). It will cover the features of Visual Studio 2010 through an application. It will go through a lot of the exciting new language features and new versions of the most popular technologies without putting the emphasis on the technologies themselves. It will instead put the emphasis on how you would get to those new tools and features from Visual Studio 2010. If you are expecting this book to thoroughly cover the new Entity Framework or ASP.NET MVC 2, this is not the book for you. If you want to read a book where the focus is on Visual Studio 2010 and on the reasons for moving to Visual Studio 2010, this is the book for you.

How Will This Book Help Me Move to Visual Studio 2010?

This book will try to answer this question by using a practical approach and by going through the new features and characteristics of Visual Studio 2010 from your point of view—that is, from the view of someone using Visual Studio 2005, for example. To be consistent for all points of view and to cover the same topics from all points of view, we decided to build and

use a real application that covers many areas of the product rather than show you many disjointed little samples. This application is named *Plan My Night*, and we'll describe it in detail in this Introduction.

To help as many developers as we can, we decided to divide this book into three parts:

- Part I will be for developers moving from Visual Studio 2003

- Part II will be for developers moving from Visual Studio 2005

- Part III will be for developers moving from Visual Studio 2008

Each part will help developers understand how to use Visual Studio 2010 to create many different types of applications and unlock their creativity independently of the version they are using today. This book will be focusing on Visual Studio, but we'll also cover many language features that make the move even more interesting.

Each part will follow a similar approach and will include these chapters:

- "Business Logic and Data"

- "Designing the Look and Feel"

- "Debugging the Application"

For example, Part I, "Moving from Microsoft Visual Studio 2003 to Visual Studio 2010," includes a chapter called "From 2003 to 2010: Debugging the Application." Likewise, Part II, "Moving from Microsoft Visual Studio 2005 to Visual Studio 2010," includes a chapter called "From 2005 to 2010: Debugging the Application."

Designing the Look and Feel

These chapters will focus on comparing how the creation of the user interface has evolved through the versions of Visual Studio. They pay attention to the design surface, the code editor, the tools, and the different controls, as well as compare UI validation methods. These chapters also tackle the topic of application extensibility.

Business Logic and Data

These chapters tackle how the application is structured and demonstrate the evolution of the tools and language features available to manage data. They describe the different application layers. They also show how the middle-tier is created across versions and how the application will manage caching the data as well as how to manage getting data in and from the database.

Debugging the Application

These chapters showcase the evolution of all developer aids and debugger tools as well as compare the different ways to improve the performance of an application. They also discuss the important task of unit-testing your code.

Deploying Plan My Night

Part I, for developers using Visual Studio 2003, also includes one extra chapter, "From 2003 to 2010: Deploying Plan My Night." This chapter goes through the different ways to package, deploy, and deliver your application to your end users. The topic of updating and sending new bits to your customers is also discussed. We feel that Parts II and III, for developers using Visual Studio 2005 and Visual Studio 2008, didn't require a chapter on deployment.

What Is Plan My Night?

Plan My Night (PMN) is an application that is self-describing, but just to make sure we're on the same page, here's the elevator pitch about PMN:

> *Plan My Night is designed and developed to help its users plan and manage their evening activities. It allows the user to create events, search for activities and venues, gather information about the activities and the venues, and finally share or produce information about them.*

As the saying goes, a picture is worth a thousand words, so take a look at Plan My Night user's interface in Figure I-1.
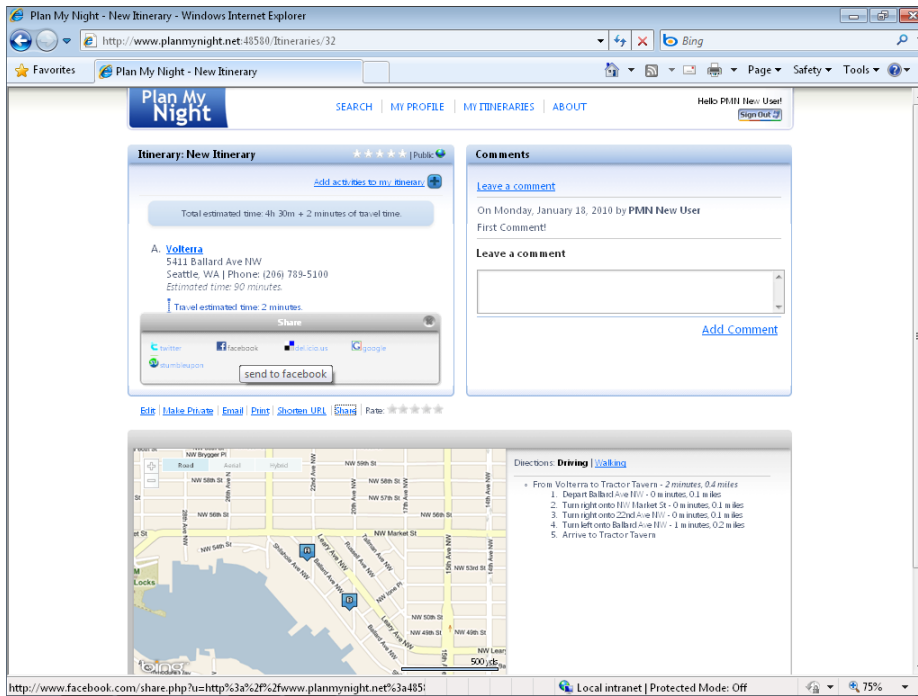
Figure I-1 PMN's user interface.

In its Visual Studio 2010 version, Plan My Night is built with ASP.NET MVC 2.0 using jQuery and Ajax for UI validation and animation. It uses the Managed Extensibility Framework (MEF) for extending the capabilities of the application by building plug-ins: for sharing to social networks, printing, emailing, etc. We have used the Entity Framework to create the data layer and the Windows Server App Fabric (formerly known as codename Velocity) to cache data in memory sent and obtained from the SQL Server 2008 database.

We figure that three pictures are better than one, so take a look at Figure I-2 for a diagram displaying the different parts and how they interact with each other and at Figure I-3 to see the different technologies used in building Plan My Night.

Figure I-2 Plan My Night components and interactions.

Plan My Night 1.0

Add-In Modules

MEF

IIS 7.5

ASP.NET 4.0

ASP.NET MVC 2.0

Windows Server App Fabric (previously codenamed Velocity)

SQL SERVER 2008

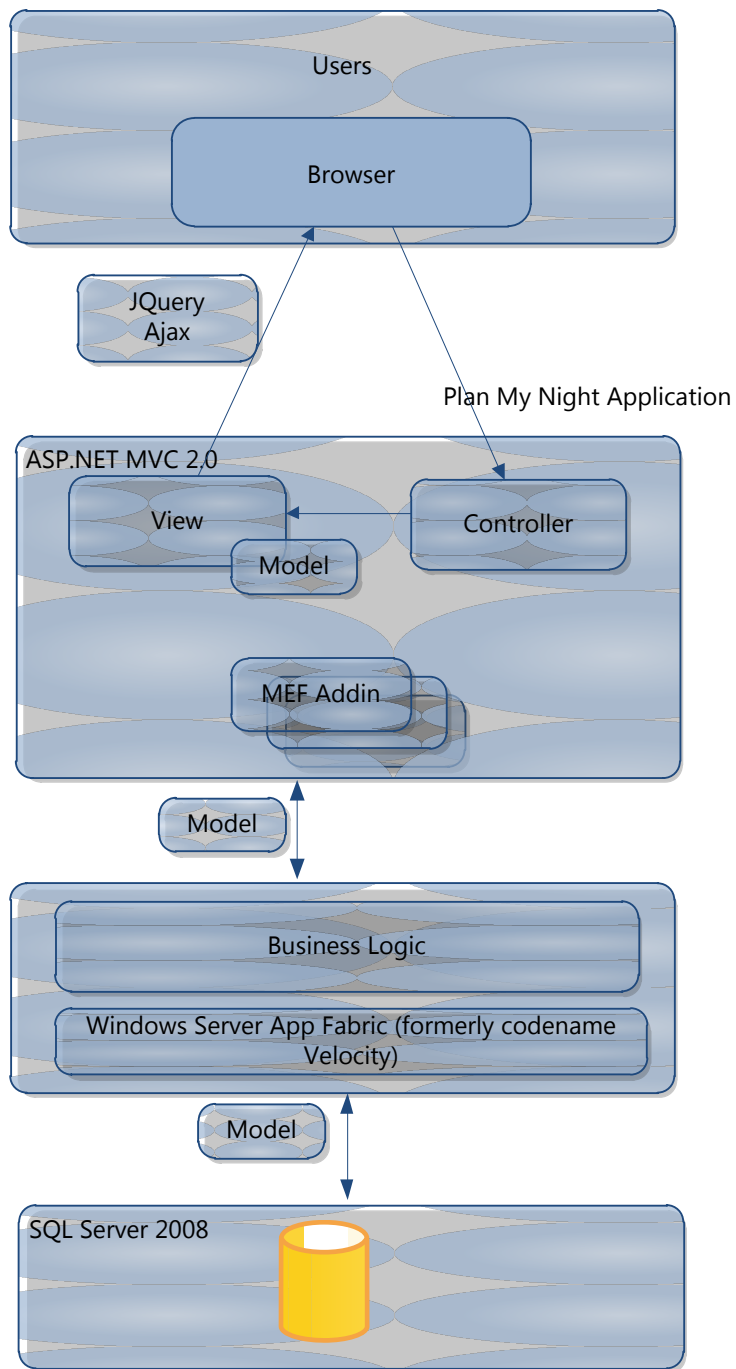Windows LiveID

Bing
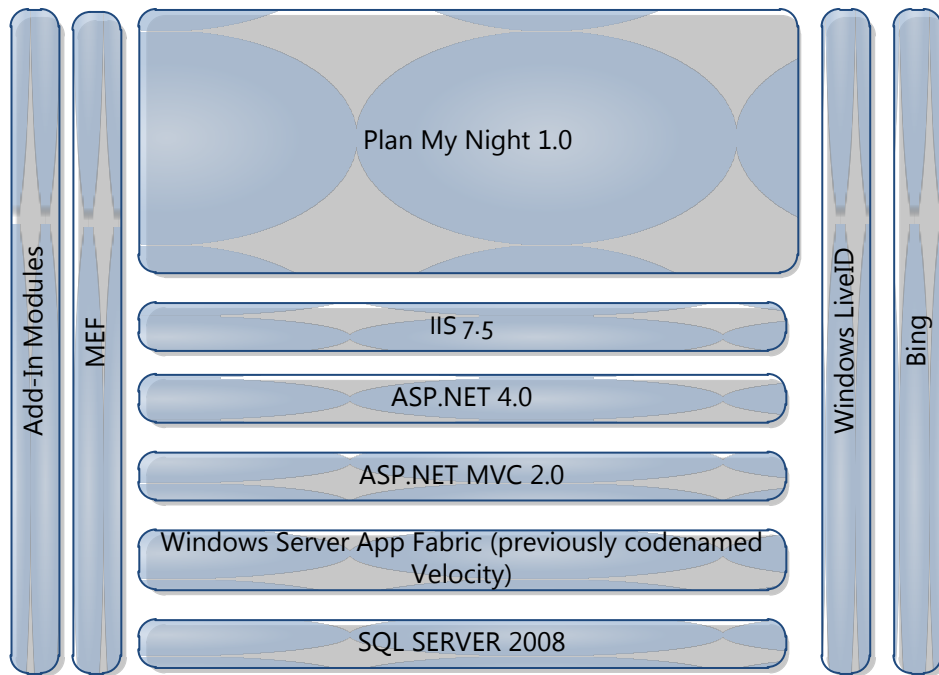
Figure I-3 PMN 1.0 and the different technologies used in building it.

Why Should You Move to Visual Studio 2010?

There are numerous reasons to move to Visual Studio 2010 Professional, and before we dive in into the book parts to examine them, we thought it would be good to list a few from a high-level perspective (presented without any priority ordering).

- Built-in tools for Windows 7, including multi-touch and "ribbon" UI components.

- Rich new editor built in WPF that you can highly customize to suit how you work. Look below this list at Figure I-4 for a sneak peek.

- Multi-monitor support.

- New Quick Search helping to find relevant results just by quickly typing the first few letters of any method, class, or property.

- Great support for developing and deploying Microsoft Office 2010, SharePoint 2010 and Windows Azure applications.

- Multicore development support allows you to parallelize your applications, and a new specialized debugger to help you track the tasks and threads.

- Improvements to the ASP.NET AJAX framework, core JavaScript IntelliSense support, and the inclusion in Visual Studio 2010 of jQuery, the open-source library for DOM interactions.

- Multi-targeting/multi-framework support. Read Scott Guthrie's blog post to get an understanding of this great feature: http://weblogs.asp.net/scottgu/archive/2009/08/27/multi-targeting-support-vs-2010-and-net-4-series.aspx.

- Support for developing WPF and Silverlight applications with enhanced drag and drop support and data binding. Great new enhancements to the designers, enabling a higher fidelity in rendering your controls, which in turn enables you to discover bugs in rendering before they happen at run time (which is a great improvement from previous versions of Visual Studio). New WPF and Silverlight tools will help you to navigate the visual tree and inspect objects in your rich WPF and Silverlight applications.

- Great support for TFS 2010 (and previous versions) using Team Explorer. This enables you to use the data and reports that are automatically collected by Visual Studio 2010 and track and analyze the health of your projects with the integrated reports as well as maintaining your bugs and tasks up-to-date.

- Integrated support for Test-First Development. Automatic test stub generation and a rich unit test framework are two nice test features that developers can take advantage of for creating and executing unit tests. Visual Studio 2010 has great extensibility points that will enable you to also use common third-party or open source unit test frameworks directly within Visual Studio 2010.
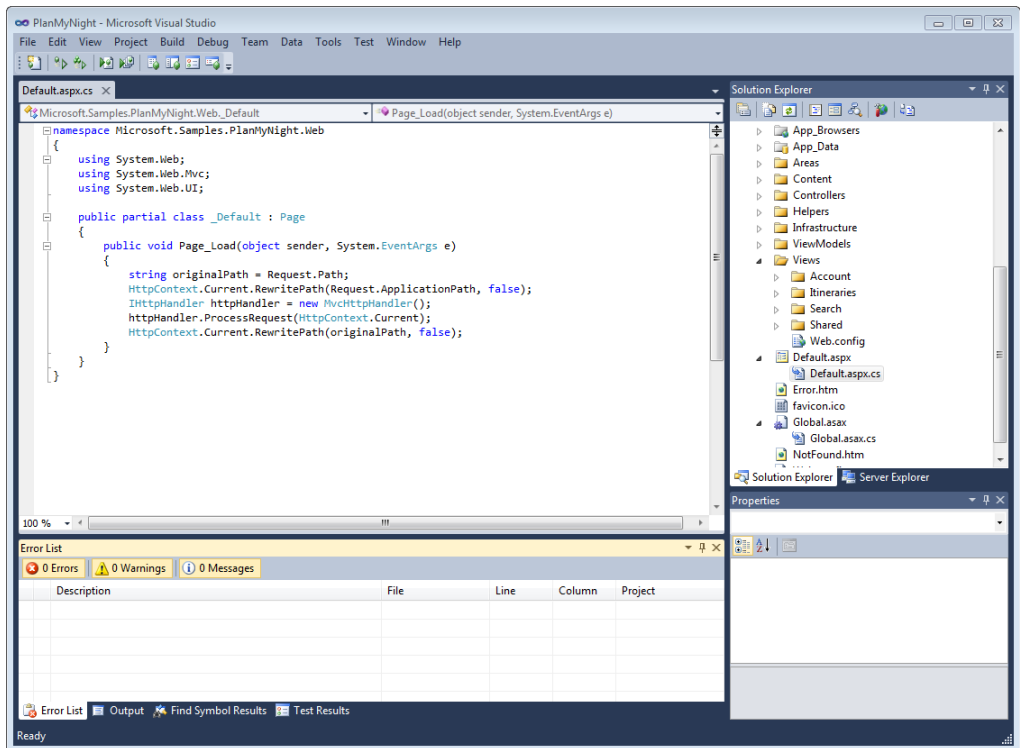
Figure I-4 Visual Studio New WPF Code Editor.

This is just a short list of all the new features of Visual Studio 2010 Professional; you'll experience some of them firsthand in this book. You can get the complete list of new features by reading the information presented in those two locations: http://msdn.microsoft.com/en-us/library/dd547188(VS.100).aspx and http://msdn.microsoft.com/en-us/library/bb386063(VS.100).aspx.

But the most important reason for many developers and enterprises to make the move is to be able to concentrate on the real problems you're facing rather than spend your time interpreting code. You'll see that with Visual Studio 2010 you can solve those problems faster. Visual Studio 2010 provides you with new powerful design surfaces and powerful tools that help you write less code, write it faster, and deliver it with higher quality.

Chapter 5
From 2005 to 2010: Business Logic and Data

After reading this chapter, you will be able to

- Use Entity Framework to build a data access layer using an existing database or with the model first approach

- Generate entity types from the Entity Data Model designer using the ADO.NET Entity Framework POCO templates

- Get data from Web services

- Learn about data caching using the Windows Server AppFabric (formerly known as codename Velocity)

Application Architecture

The PlanMyNight application allows the user to manage his itinerary activities and share them with others. The data stored in a SQL Server database. Activities are gathered from searches to the Bing Maps Web services.

Let's have a look at the high-level block model of the data model for the application, which is shown in Figure 5-1.
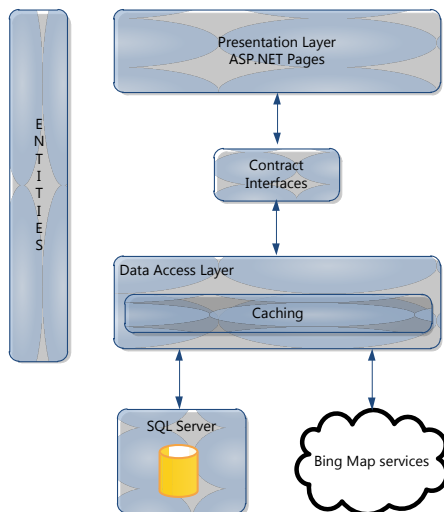

Figure 5-1 PlanMyNight application architecture diagram

Defining contracts and entities classes that are cleared of any persistence-related code constraints will allow us to put them in an assembly that has no persistence aware code. This will ensure a clean separation between the Presentation and Data layers.

Let's identify the contract interfaces for the major components of the PMN application:

- IItinerariesRepository will be the interface to our data store (Microsoft SQL Server database).

- IActivitiesRepository will allow us to search for activities (Bing Map Web services).

- ICachingProvider will provide us our data caching interface (ASP.NET Caching or Windows Server AppFabric Caching).

> **Note**   This is not an exhaustive list of the contracts implemented in the PMN application.

PMN stores the user's itineraries into a SQL database. Other users will be able to comment and rate each other itineraries. **Error! Reference source not found.**Figure 5-2 shows the tables used by the PMN application.
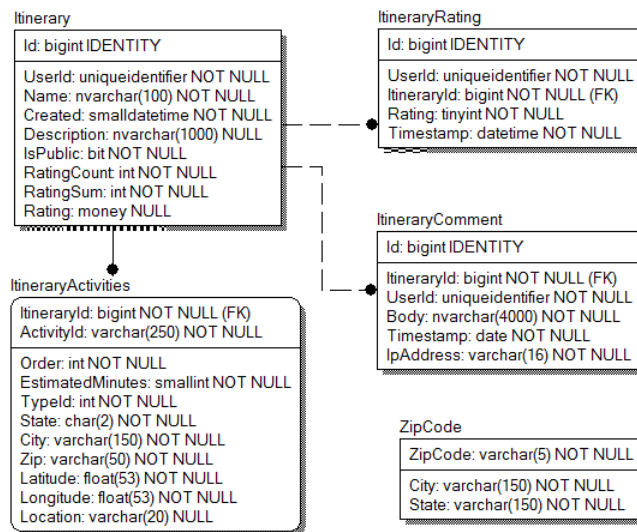

Figure 5-2 PlanMyNight datase schema

> **Important**   The PlanMyNight application uses the ASP.NET Membership feature to provide secure credential storage for the users. The user store tables are not shown in the **Error! Reference source not found.**Figure 5-2. You can learn more about this feature on MSDN: ASP.NET 4 - Introduction to Membership.

PlanMyNight Data in Microsoft Visual Studio 2005

It would be straightforward to create the PlanMyNight in Visual Studio 2005 since it offers all the required tools to help you to code the application. However, some of the technologies used back then required you to write a lot more code to achieve the same goals.

Let's take a look at how you could create the required data layer in Visual Studio 2005. One approach would have been to write the data layer using ADO.NET DataSet or DataReader directly. This solution offers you a great flexibility since you have complete control over the access to the database. On the other hand, it also has some drawbacks:

- You need to know the SQL syntax.

- All queries are specialized. A change in requirement or in the tables will force you to update all the related queries in the code.

- You need to map the properties of your entity classes using the column name which is tedious and error prone.

- You have to manage the relations between tables yourself.
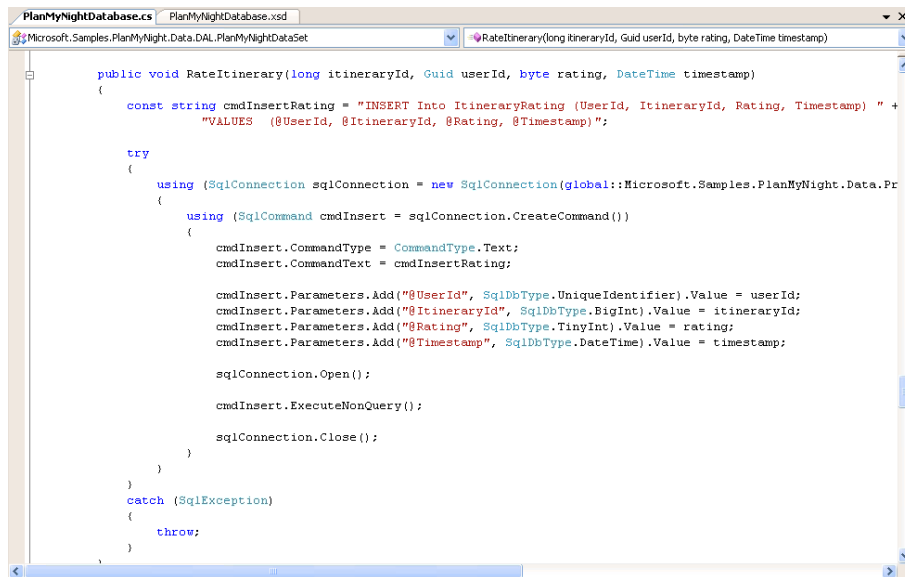
Figure 5-3   ADO.NET Insert query

Another approach would be to use the DataSet designer available in Visual Studio 2005. Starting from a database with the PMN tables, you could use the TableAdapter Configuration Wizard to import the database tables as show in Figure 5-4. The generated code offers you a typed DataSet. One of the benefits includes type checking at design time which gives the advantage of statement completion. There are still some pain points with this approach:

- You still need to know the SQL syntax although you have access to the Query builder directly from the DataSet designer.

- You still need to write specialized SQL queries to match each of the requirements of your data contracts.

- You have no control on the generated classes. For example, changing the DataSet to add or remove a query for a table will rebuild the generated TableAdapter classes and may change the index used for a query. This makes it difficult to write predictable code using these generated items.

- The generated classes associated with the tables are persistence aware so you will have to create another set of simple entities and copy the data from one to the other. This means more processing and memory usage.
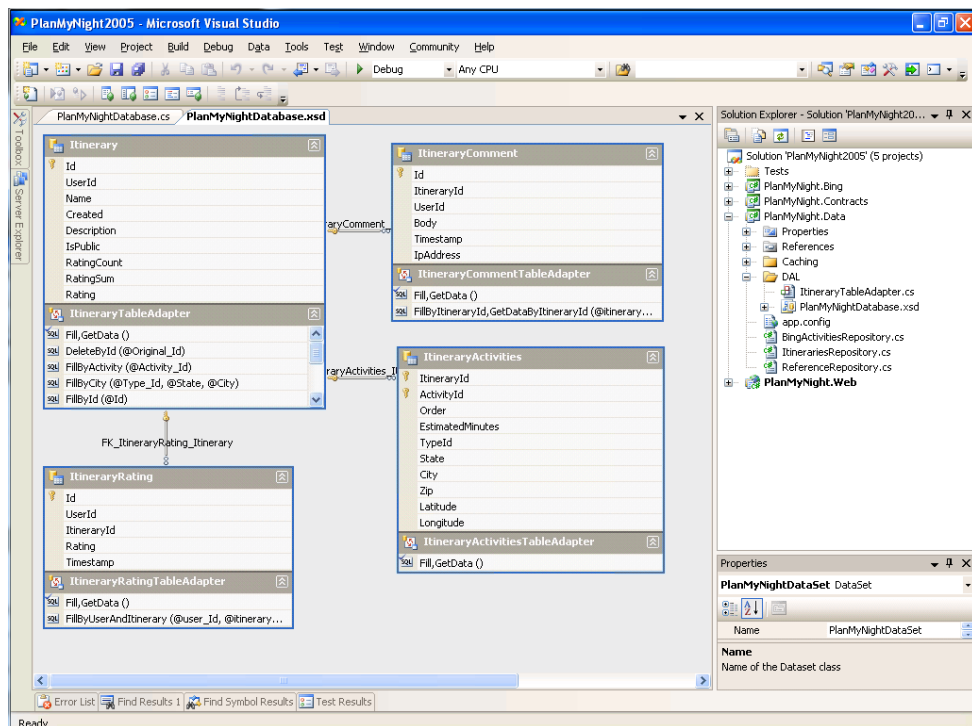

Figure 5-4 Dataset designer in Visual Studio 2005

In the next sections of this chapter, you are going to explore some of the new features of Visual Studio 2010 that will help you create the PMN data layer with less code, give you more control on the generated code and allow to easily maintain and expand it.

Data with the Entity Framework in Visual Studio 2010

The ADO.NET Entity Framework (EF) allows you to easily create the data access layer for an application by abstracting the data from the database and exposing a model closer to business requirement of the application. The EF has been considerably enhanced in the .NET Framework 4 release.

> See Also   The MSDN Data Developer Center offers a lot of resources about the ADO.NET Entity Framework in .NET 4.

You are going to use the PlanMyNight project as an example of how to build an application using some of the features of the EF. The next two sections demonstrate two different approaches to generate the data model of PMN. In the first one, you are going to let the EF generate the Entity Data Model (EDM) from an existing database. In the second part, you will use a Model First approach where you first create the entities from the EF designer and generate the Data Definition Language (DDL) scripts to create a database that can store your EDM.

EF: Importing an Existing Database

You are going to start with an existing solution that already defines the main projects of the PMN application. If you have installed the companion content at the default location you will find the solution at this location: %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 5\Code\ExistingDatabase. Double-click the PlanMyNight.sln file.

This solution includes all the projects you can see in Figure 5-5.

- PlanMyNight.Data: Application data layer.

- PlanMyNight.Contracts: Entities and Contracts.

- PlanMyNight.Bing: Bing Map Services

- PlanMyNight.Web: Presentation Layer

- PlanMyNight.AppFabricCaching: AppFabric Caching
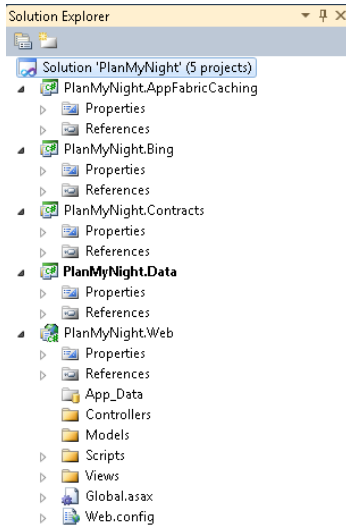
Figure 5-5 PlanMyNight Solution

The EF allows you to easily import an existing database. Let's walk through this process.

The first step is to add an EDM to the PlanMyNight.Data project. Right-click the PlanMyNight.Data project, select Add, and then New Item... Select the ADO.NET Entity Data Model item and change its name to PlanMyNight.edmx, as shown in Figure 5-6.
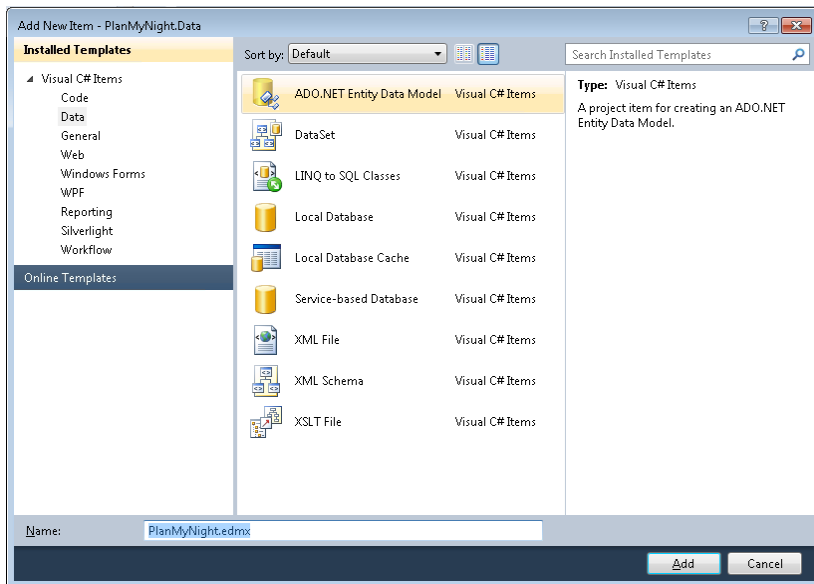

Figure 5-6 Add New item dialog with Add ADO.NET Entity Data Model

The first dialog of the Entity Data Model Wizard allows you to choose the model content. You are going to generate the model from an existing database. Select Generate From Database then click Next.

You need to connect to an existing database file. Click New Connection... and select the %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 5\ExistingDatabase\PlanMyNight.Web\App_Data\PlanMyNight.mdf file.

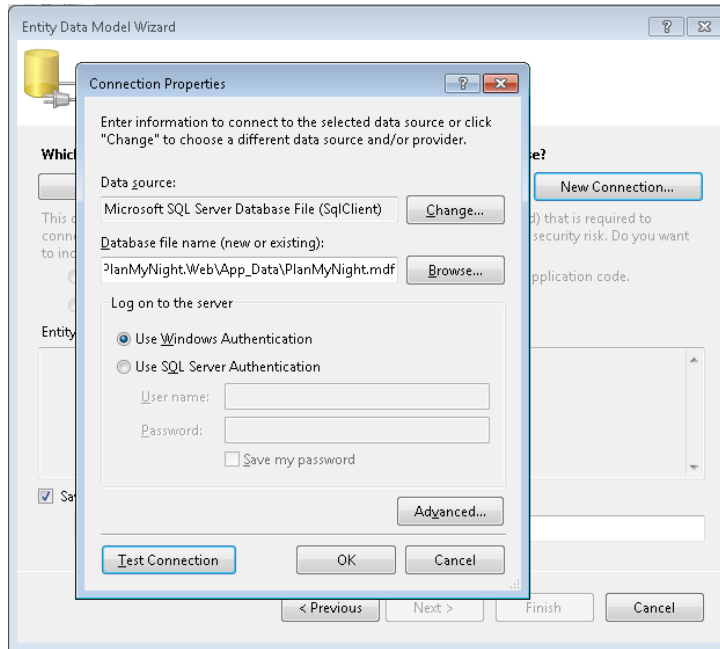Figure 5-7 EDM Wizard Database Connection

Leave the other fields in the form as is for now and click Next.

From the Choose Your Database Objects dialog, select the Itinerary, ItineraryActivities, ItineraryComment, ItineraryRating, and ZipCode tables and the UserProfile view. Select the RetrieveItinerariesWithinArea stored procedure. Change the Model Namespace to Entities as shown in Figure 5-8.
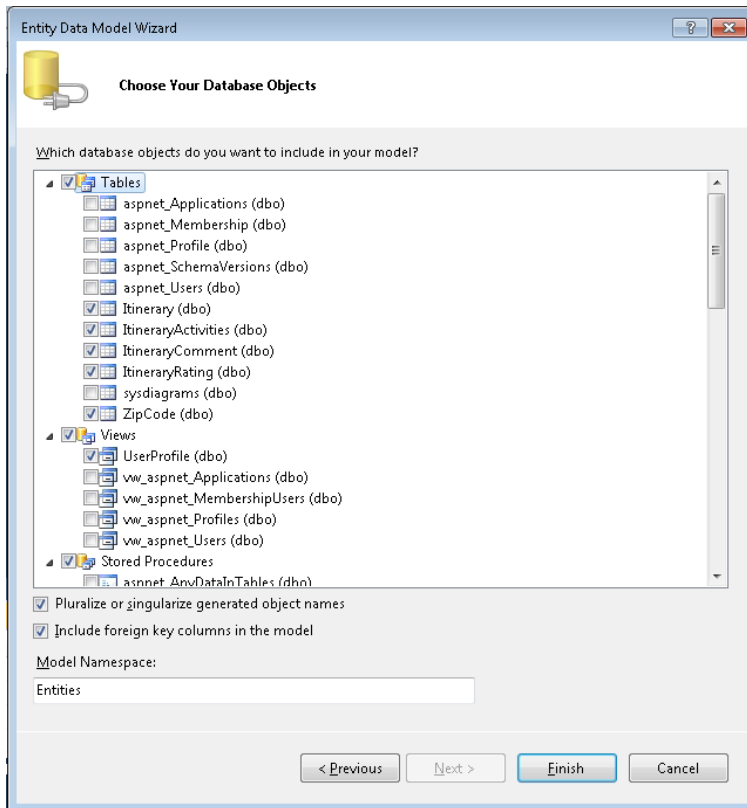
Figure 5-8 EDM Wizard: Choose the database objects

Click Finish to generate your EDM.

Fixing the Generated Data Model

You now have a model representing a set of entities matching your database. The wizard has generated all the navigation properties associated with the foreign keys from the database.

The PMN application only requires the navigation property associated with the ItineraryActivies table so you can go ahead and delete all the other navigation properties. You will also need to rename the *ItineraryActivities* navigation property to Activities. Refer to Figure 5-9 for the updated model.
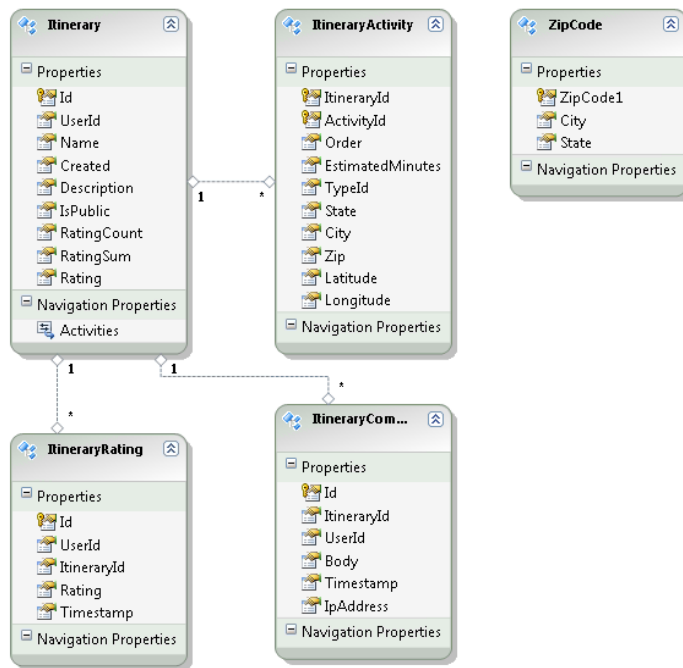
Figure 5-9 Model imported from the PlanMyNight database

You will notice that one of the properties of the ZipCode entity has been generated with the name *ZipCode1*. Let's fix the property name by double-clicking it. Change the name to Code, as shown in Figure 5-10.
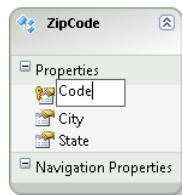
Figure 5-10 ZipCode entity

Build the solution by pressing Ctrl+Shift+B. When looking at the output window, you will notice two messages from the generated EDM. You can discard the first one since the Location column is not required in PMN. The second message reads:

> *The table/view 'dbo.UserProfile' does not have a primary key defined and no valid primary key could be inferred. This table/view has been excluded. To use the entity, you will need to review your schema, add the correct keys, and uncomment it.*

By looking at the UserProfile view, you will notice that it does not explicitly define a primary key even though the UserName column is unique.

You will have to modify the EDM manually fix the UserProfile view mapping

From the project explorer, right-click the PlanMyNight.edmx file and then select Open With...
Choose XML (Text) Editor from the Open With dialog as shown in Figure 5-11. This will open
the XML file associated with your model.
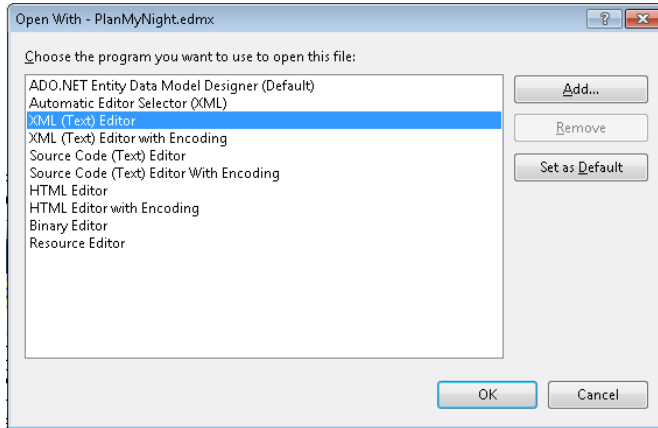
Figure 5-11 Open PlanMyNight.emdx in the XML Editor

> **Note**   You will get a warning stating that the PlanMyNight.edmx file is already open. Click Yes to
> close it.

The generated code was commented out by the code generation tool because there was no
primary key defined. To allow you to use the UserProfile view from the designer, you need to
uncomment the UserProfile entity type and add the Key tag to it. Search for UserProfile in the
file. Uncomment the entity type, add a key tag and set its name to **UserName** and make the
UserName property not nullable. Refer to Listing 5-1 to see the updated EntityType.

Listing 5-1 UserProfile Entity Type XML Definition

```
<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>
```

If you close the XML file and try to open the EDM Designer, you will get this error:

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

You will need to define an entity set for the UserProfile type so that it can map the entity type to the store schema. Open the PlanMyNight.edmx file in the XML editor so that you can define an entity set for UserProfile. At the top of the file, just above the Itinerary entity set, add the XML code shown in Listing 5-2.

Listing 5-2 UserProfile EntitySet XML Definition

```
  <EntitySet Name="UserProfile" EntityType="Entities.Store.UserProfile"
store:Type="Views" store:Schema="dbo" store:Name="UserProfile">
  <DefiningQuery>
     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>
```

Save the EDM XML file and reopen the EDM Designer. The Figure 5-12 shows the UserProfile view in the Entities.Store section of the Model Browser.

Tip You can open the Model Browser from the View menu by clicking Other Windows and selecting the Entity Data Model Browser item.
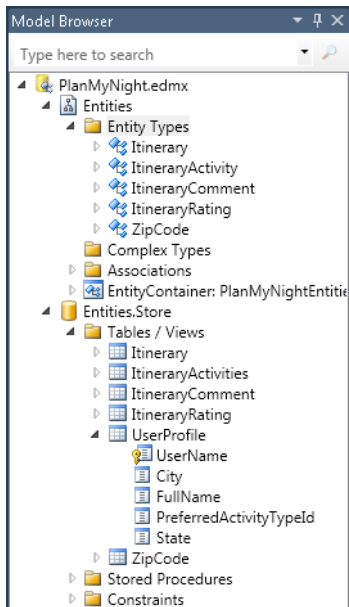
Figure 5-12 Model Browser with the UserProfile view

Now that the view is available in the store metadata, you are going to add the UserProfile entity and map it to the UserProfile view. Right-click in the background of the EDM Designer, select Add and then Entity...
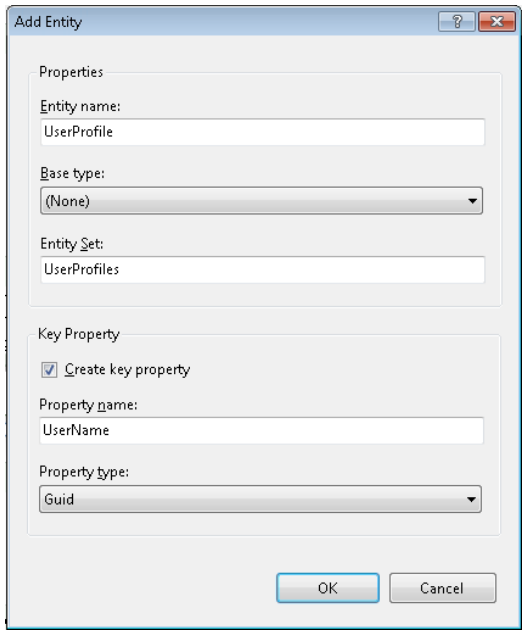
Figure 5-13 Add UserProfile entity dialog

Complete the dialog as shown in Figure 5-13 and click OK to generate the entity.

You will need to add the remaining properties: *City*, *State,* and *PreferredActiviyTypeId*. To do so, right-click the UserProfile entity, select Add, then Scalar Property. Once the property is added, set the *Type*, *Max Length,* and *Unicode* field value. Table 5-1 shows the expected values for each of the field.

Table 5-1 UserProfile Entity Properties

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

Now that you have created the UserProfile entity, you need to map it to the UserProfile view. By right-clicking the UserProfile entity, select Table Mapping as shown in Figure 5-14.
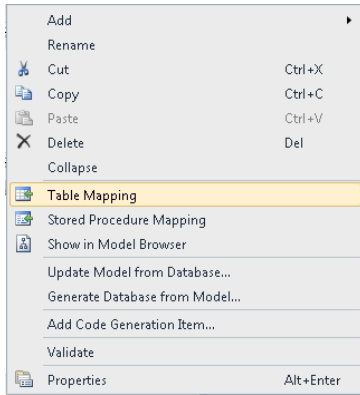
Figure 5-14 Table Mapping menu item

Then select the UserProfile view from the dropdown box as shown in Figure 5-15. Ensure that all the columns are correctly mapped to the entity properties. The UserProfile view of our store is now accessible from the code through the UserProfile entity.
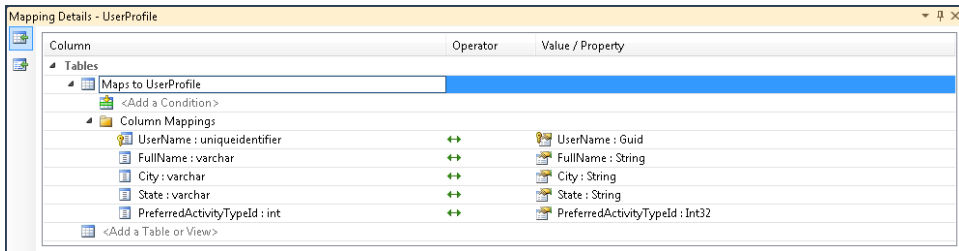
Figure 5-15 UserProfile Mapping details

Stored Procedure and Function Imports

The Entity Data Model Wizard has created an entry in the storage model for the *RetrieveItinerariesWithinArea* stored procedure you selected in last step of the wizard. You need to create a corresponding entry to the conceptual model by adding a Function Import.

From the Model Browser, open the Stored Procedures folder in the Entities.Store section. Right-click *RetrieveItineraryWithinArea*, and then select Add Function Import... The Add Function Import dialog appears as shown in Figure 5-16. Specify the return type by selecting Entities and then the item Itinerary from the drop-down box. Click OK.
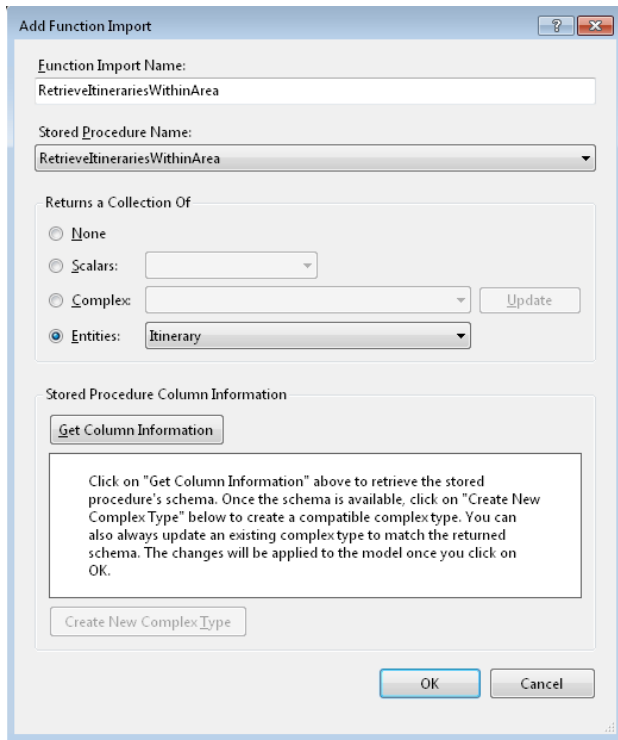
Figure 5-16 Add Function Import dialog

The *RetrieveItinerariesWithinArea* function import was added to the Model Browser as shown in Figure 5-17.
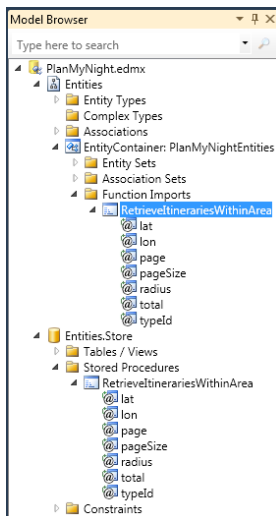
Figure 5-17 Function Imports in the Model Browser

You can now validate the EDM by right-clicking in the design surface and selecting Validate. There should be no error or warning.

EF: Model First

In the last section, we saw how to use the EF designer to generate the model by importing an existing database. The EF Designer in Visual Studio 2010 also supports the ability to generate the Data Definition Language (DDL) file that will allow you to create a database based on your entity model.

You can start from empty model by selecting the Empty model option from the Entity Data Model Wizard.
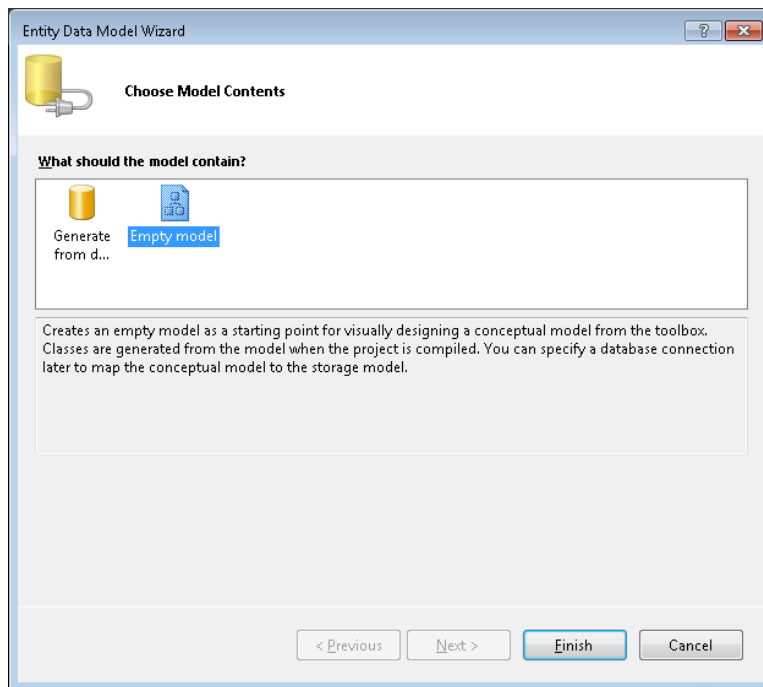
Figure 5-18 EDM Wizard: Empty Model

Open the PMN solution at this location: %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 5\Code\ModelFirst by double-clicking the PlanMyNight.sln file.

The PlanMyNight.Data project from this solution already contains an EDM file named PlanMyNight.edmx with some entities already created. These entities are matching the data schema from Figure 5-2.

The Entity Model designer lets you easily add an entity to your data model. Let's add the missing ZipCode entity to the model. From the Toolbox, drag an Entity item into the designer,

as shown in Figure 5-19. Rename the entity to ZipCode. Rename the *Id* property to Code and change its type to String.
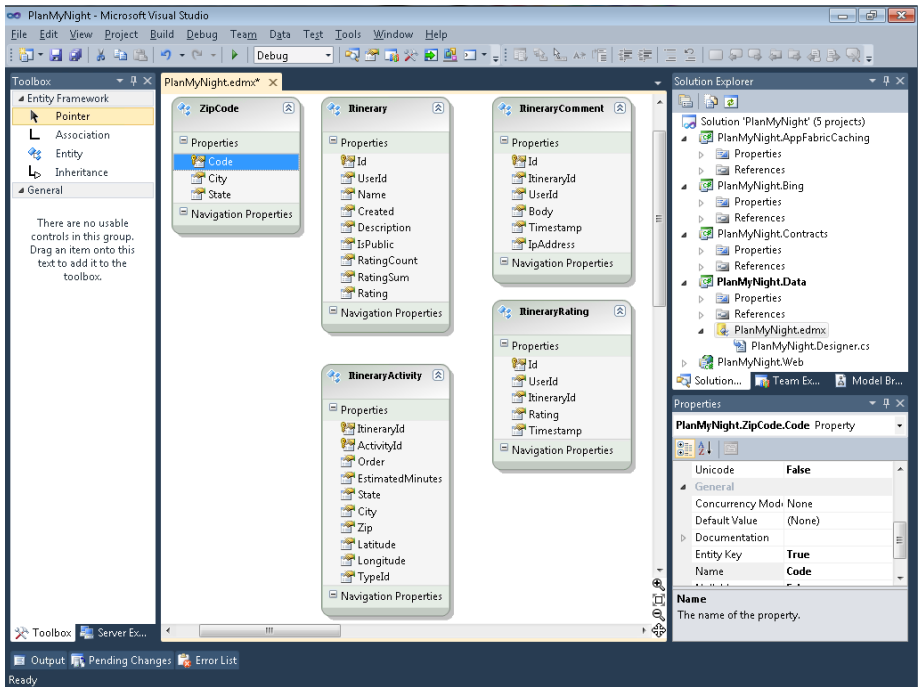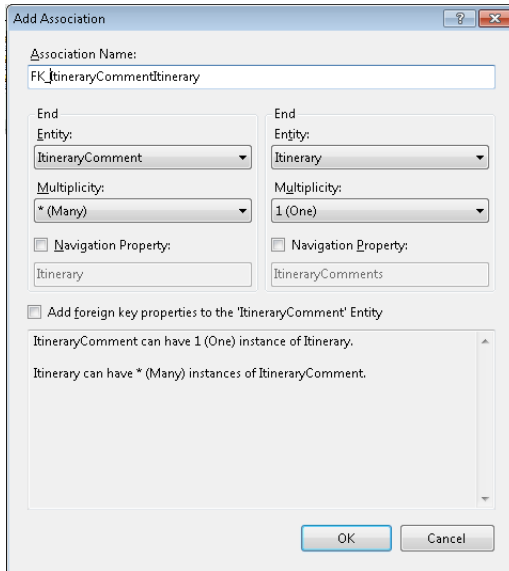
Figure 5-19 Entity Model designer

You need to add the *City* and *State* properties to the entity. Right-click the ZipCode entity, select Add and then Scalar Property. Ensure that each property has the values shown in Table 5-2.

Table 5-2 ZipCode Entity Properties

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

Add the relations between the ItineraryComment and the Itinerary entities. Right-click the designer background, select Add and then Association...
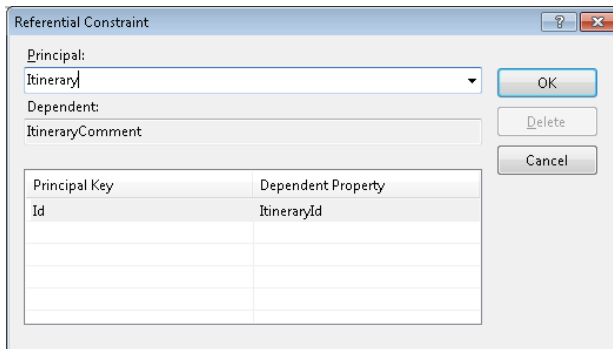
Figure 5-20 Add Association dialog for FK_ItineraryCommentItinerary

Set the association name to FK_ItineraryCommentItinerary and the select the entity and the multiplicity for each end, as shown in Figure 5-20. Once the association is created, double-click association line to set the Referential Constraint as shown in Figure 5-21.

Figure 5-21   Association Referential Constraint dialog

Repeat the same operations for the FK_ItineraryItineraryRating association by setting the first end to ItineraryRating.

For the FK_ItineraryItineraryActivity association, you want to also create a navigation property and name it Activities, as shown in Figure 5-22.
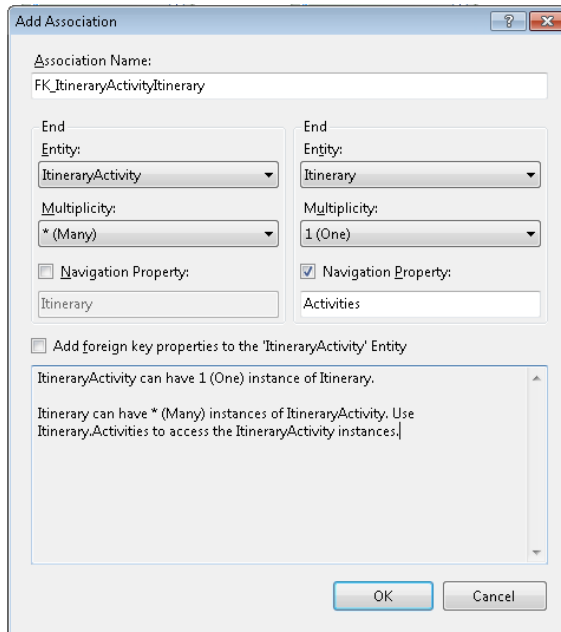
Figure 5-22 Add Association dialog for FK_ItineraryActivityItinerary

Generating the Database Script from the Model

Your data model is now completed but there is no mapping or store associate to it. The EF designer offers the possibility to generate a database script from our model.

Right-click in the designer surface and choose Generate Database From Model... as shown in Figure 5-23.
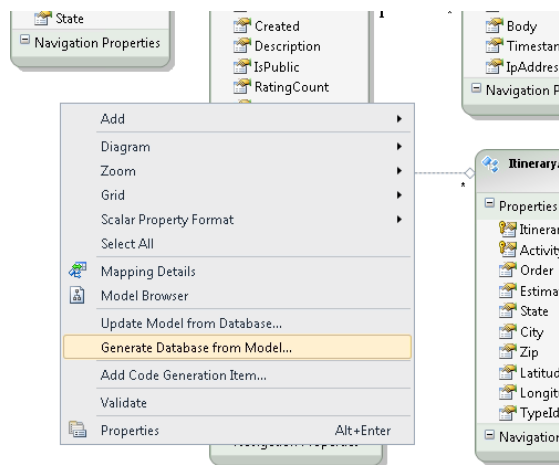
Figure 5-23 Generate Database from Model menu item

The Generate Database Wizard requires a data connection. The wizard will use the connection information to translate the model types to the database type and to generate a DDL script targeting this database.

Select New Connection... and make sure the Data Source is set to Microsoft SQL Server File. Click Browse... and select the database file located %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 5\Code\ModelFirst\Data\PlanMyNight.mdf.
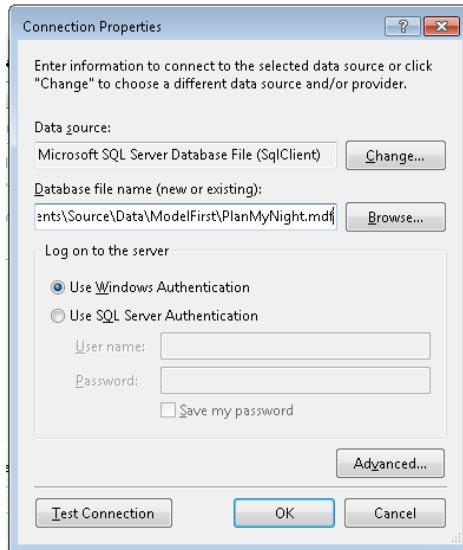
Figure 5-24 Generate script database connection

Once your connection is configured, click Next to get to wizard last screen. When you click Finish, the generated T-SQL PlanMyNight.edmx.sql file is added to your project. The DDL script will generate the primary and foreign key constraints for your model.
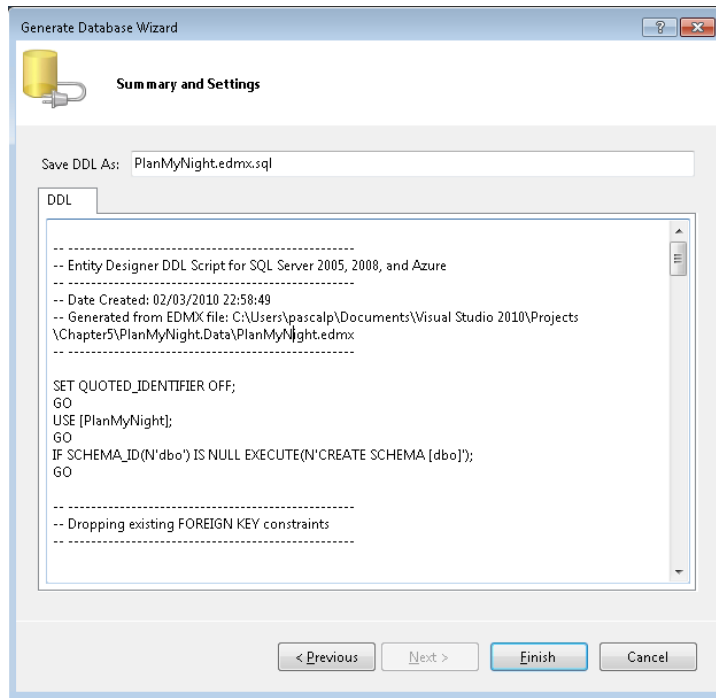
Figure 5-25 Generated T-SQL file

The EDM is also updated to ensure your newly created store is mapped to the entities. You could now use the generated DDL script to add the tables to the database. You now have a data layer that expose strongly typed entities that you can use in your application.

> **Important**   Generating the complete PMN database would require adding the remaining tables, stored procedure and triggers used by the application. Instead of performing all these operations, we will go back to the solution we had at the end of the "EF: Importing an Existing Database" section.

POCO Templates

The EDM Designer uses T4 templates to generate the entities code. So far, we have let the designer create the entities using the default templates. You can take a look at the code generated by opening the PlanMyNight.Designer.cs file associated to PlanMyNight.edmx. The generated entities are based on the EntityObject type and decorated with attributes to allow the EF to manage them at run time.

> **Note**   T4 stands for Text Template Transformation Toolkit. T4 support in Visual Studio 2010 allows you to easily create your own templates and generate any type of text file (Web, resource or source). To learn more about the code generation in Visual Studio 2010, visit Code Generation and Text Templates.

The EF also supports POCO entity types. POCO classes are simple objects with no attributes or base class related to the framework. (Listing 5-3, in the next section, shows the POCO class for the ZipCode entity.) The EF uses the names of the types and the properties of these objects to map them to the model at run time.

Note POCO stands for Plain-Old CLR Objects.

ADO.NET POCO Entity Generator

Let's re-open the %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 5\Code\ExistingDatabase\PlanMyNight.sln file.

Select the PlanMyNight.edmx file, right-click in the design surface and choose Add Code Generation Item... This will open a dialog shown in Figure 5-26 where you can select the template you want to use. Select the ADO.NET POCO Entity Generator template and name it PlanMyNight.tt. Then click the Add button.
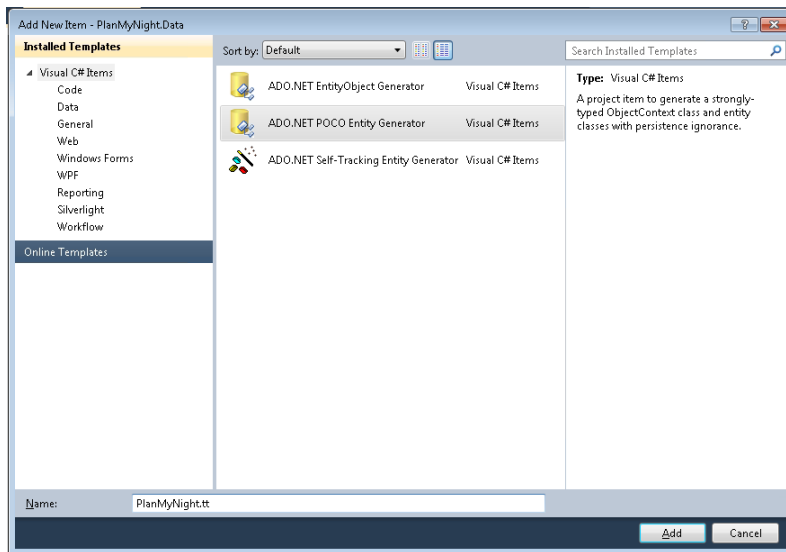
Figure 5-26 Add New Item dialog

Two files have been added to your project, as shown in Figure 5-27. These files replace the default code-generation template and the code is no longer generated in the PlanMyNight.Designer.cs file.
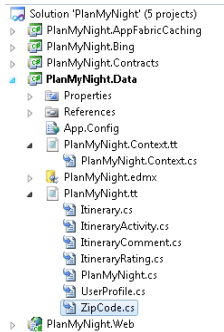
Figure 5-27 Added templates

The PlanMyNight.tt template produces a class file for each entity in the model. Listing 5-3 shows the POCO version of the *ZipCode* class.

Listing 5-3 POCO version of the *ZipCode* Class

```
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
    }
}
```

Tip   C# 3.0 introduced a new feature called "automatic properties". The backing field is created at compile time if the compiler finds empty get or set blocks.

The other file, PlanMyNight.Context.cs, generates the ObjectContext object for the PlanMyNight.edmx model. This is the object we are going to use to interact with the database.

> **Tip**   The POCO templates will automatically update the generated classes to reflect the changes to your model when you save the .edmx file.

Moving the Entity Classes to the Contracts Project

We have designed the PMN application architecture to ensure that the presentation layer was persistence ignorant by moving the contracts and entity classes to an assembly that has no reference to the storage.

> **Visual Studio 2005**   Even though it was possible to extend the XSD processing with code generator tools, it was not easy and you had to maintain these tools. The EF uses T4 templates to generate both the database schema and the code. These templates can easily be customized to your needs.

The ADO.NET POCO templates split the generation of the entity classes to a separate template allowing us to easily move these entities to a different project.

You are going to move the PlanMyNight.tt file to the PlanMyNight.Contracts project. By right-clicking PlanMyNight.tt file, select Cut. Right-click the Entities folder in the PlanMyNight.Contracts project and select Paste.
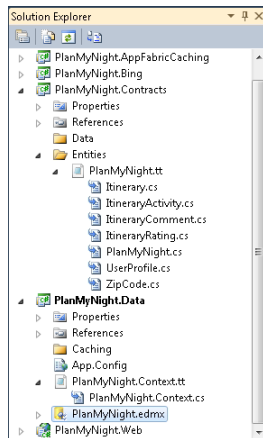
Figure 5-28 POCO Template moved to the Contracts Project

The PlanMyNight.tt template relies on the metadata from the EDM model to generate the entity type's code. You need to fix the relative path used by the template to access the EDMX file.

Open the PlanMyNight.tt template and locate the line:

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

Fix the file location so it points to the PlanMyNight.edmx file in the PlanMyNight.Data project:

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

The entity classes are regenerated once you save the template.

You also need to update the PlanMyNight.Context.tt template since the entity classes are now in the *Microsoft.Samples.PlanMyNight.Entities* namespace instead of the *Microsoft.Samples.PlanMyNight.Data* namespace. Open the PlanMyNight.Context.tt file and update the *using* section to include the new namespace:

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

Build the solution with Ctrl+Shift+B. The project should now compile successfully.

Getting It All Together

Now that you have created the generic code layer to interact with your SQL database, you are ready to start implementing the functionalities specific to the PMN application. In the next sections, you are going to walk through this process, briefly look at the getting data from the Bing Maps services and get a quick introduction to the Windows Server AppFabric Caching feature used in PMN.

There is a lot of plumbing pieces of code required to get this all together. To simplify the process, you are going to use an updated solution where the contracts, entities and most of the connecting pieces to the Bing Maps services have been coded. The solution will also include the PlanMyNight.Data.Test project to help you validate the code from the PlanMyNight.Data project.

> **Note** Testing in Visual Studio 2010 will be covered in Chapter 7.

Getting Data from the Database

At the beginning of this chapter, we have decided to group the operations on the Itinerary entity into the IItinerariesRepository repository interface. Some of these operations are:

- Searching for Itinerary by Activity

- Searching for Itinerary by ZipCode

- Searching for Itinerary by Radius

- Adding a new Itinerary

Let's take a look at the corresponding methods in the IItinerariesRepository interface:

- *SearchByActivity* will allow searching for itineraries by activity and returning a page of data.

- *SearchByZipCode* will allow searching for itineraries by zip code and returning a page of data.

- *SearchByRadius* will allow searching for itineraries from a specific location and returning a page of data.

- *Add* will allow to add itinerary to the database.

Open the PMN solution at this location: %userprofile%\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 5\Code\Final by double-clicking the PlanMyNight.sln file.

Select the PlanMyNight.Data project and open the ItinerariesRepository.cs file. This is the IItinerariesRepository interface implementation. Using the PlanMyNightEntities Object Context you have generated earlier, you will be able to write LINQ queries against your model and the EF will translate these queries to native T-SQL that will be executed against the database.

> **Note** LINQ stands for Language Integrated Query and was introduced in the .Net Framework 3.5. It adds native data querying capability to the .NET Framework so that you don't have to worry about learning or maintaining custom SQL queries. LINQ allows you to use strongly typed objects and the Visual Studio IntelliSense lets you select the properties or methods that are in the current context as shown in Figure 5-29. To learn more about LINQ, visit the .NET Framework Developer Center.
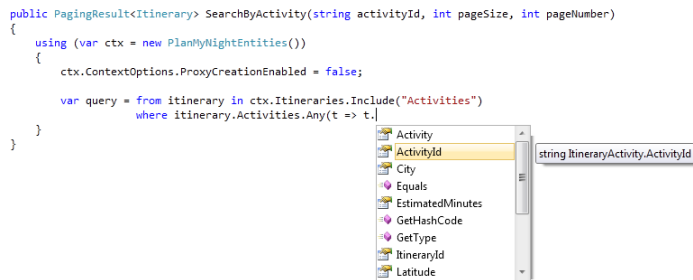
Figure 5-29 IntelliSense support for LINQ queries

Navigate to the *SearchByActivity* function definition. This method must return a set of itineraries marked as public where one of their activities has the specified activity ID. You also need to order the result itinerary list by the rating field.

Visual Studio 2005   Implementing each methods to retrieve the itinerary in Visual Studio 2005 would have required writing tailored SQL. With the EF and LINQ, any query becomes trivial and changes can be easily implemented at the code level!

Using standard LINQ operators, you can implement the *SearchByActivity* as shown in Listing 5-4. Add the highlighted code to the *SearchByActivity* method body.

Listing 5-4 *SearchByActivity* Implementation

```
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);
  }
}
```

Note The result paging is implemented in the *PageResults* method.

```
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;
}
```

An *IQueryable<Itinerary>* is passed to this function so it can add the paging to the base query composition. Passing an *IQueryable* instead on *IEnumerable* ensure that T-SQL created for the query against the repository will only be generated when *query.ToArray()* is called.

The *SearchByZipCode* function method is similar to the *SearchByActivity* method but it also adds a filter on the Zip Code of the activity. Here again, LINQ support makes it easy to implement as shown in Listing 5-5. Add the highlighted code to the *SearchByZipCode* method body.

Listing 5-5 *SearchByZipCode* Implementation

```
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);
  }
}
```

The *SearchByRadius* function calls the *RetrieveItinerariesWithinArea* import function that was mapped to a stored procedure. It then loads the activities for each itinerary found. You can copy the highlighted code in Listing 5-6 to the *SearchByRadius* method body in the ItinerariesRepository.cs file.

Listing 5-6 *SearchByRadius* Implementation

```
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;

    // Stored Procedure with output parameter
    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
    };
  }
}
```

The *Add* method allows adding Itinerary to the data store. Implementing this functionality becomes trivial since our contract and our Context Object are using the same entity object. Copy and paste the highlighted code in Listing 5-7 to the *Add* method body.

Listing 5-7 *Add* Implementation

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

Here you have it! You have been able to complete the *ItinerariesRepository* implementation using the Context Object generated using the EF designer. Run all the tests in the solution by pressing CTRL+R, A. The tests related to the *ItinerariesRepository* should all succeed.

Getting Data from the Bing Maps Web Services

PMN relies on the Bing Maps services to allow the user to search for activities to add to its itineraries. To get a Bing Maps Key to use in the PMN application, you need to create a Bing Maps Developer Account. You can create a free developer account on the Bing Maps Account Center.

See Also   Microsoft Bing Maps Web Services is a set of programmable SOAP services that allow you to match addresses to the map, search for points of interest, integrate maps and imagery, return driving directions, and incorporate other location intelligence into your Web application. You can learn more about these services by visiting the site for the Bing Maps Web Services SDK.

Visual Studio 2005   In Visual Studio 2005, if you had to connect to reference a Web service to a project, you would have selected the Add Web Service Reference from the contextual menu to

bring up the Add Web Reference Dialog and then add a reference to an ASMX Web service to your project.
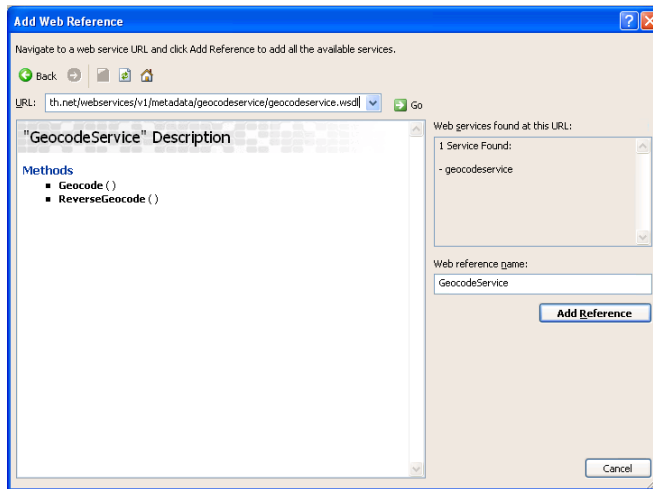
Figure 5-30 Visual Studio 2005 Add Web Reference dialog

Introduced in the .NET Framework 3.0, the Windows Communication Foundation (WCF) Services brought the ASMX Web services and other communication technologies to a unified programming model.

Visual Studio 2010 provides tools for working with Windows Communication Foundation (WCF) Services. You can bring the new Add Service Reference dialog by right-clicking on a project node and selecting Add Service Reference... as shown in Figure 5-31. In this dialog, you first need to specify the service metadata address in the Address field and then click Go to view the available service endpoints. You can then specify a Namespace for the generate code and then click OK to add the proxy to your project.
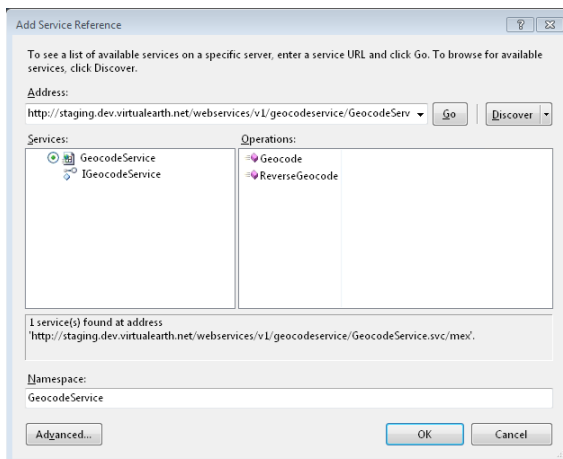
Figure 5-31 Add Service Reference dialog

Tip   The Discover button will look for WCF services in the current solution.

See Also   The Advanced... button brings the Service Reference Settings dialog. This dialog lets you tweak the configuration of the WCF service proxy. You can add the .Net Framework 2.0 style reference by using the Add Web Service button. To learn more about these settings, visit the [MSDN - Configure Service Reference Dialog Box](#)*.*

The generated WCF proxy can be used in the same way you will have used the ASMX style proxy as shown in Listing 5-8.

Listing 5-8 Using a Web Service Proxy

```
public BingCoordinate GeocodeAddress(ActivityAddress address, string token)
{
  ...
  Microsoft.Samples.PlanMyNight.Bing.VEGeocodingService.GeocodeResponse
geocodeResponse = null;
  // Make the geocode request
  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);
}
```

Parallel Programming

With the advance in the multi-core and many cores computing, it is becoming more and more important for today's developer to be able to write parallel applications. Visual Studio

2010 and the .NET Framework 4.0 are bringing new ways to express concurrency in applications. The Task Parallel Library (TPL) is now part of the Base Class Library (BCL) for the .NET Framework. This means that every .NET application can now access the TPL without adding any assembly reference.

PMN stores only the Bing Activity Id for each ItineraryActivity to the database. When it's time to retrieve the entire Bing Activity object, a function that iterates through each of the ItineraryActivity for the current Itinerary is used to populate the Bing Activity entity from the Bing Maps Web service.

One way of performing this operation would be to sequentially call the service for each activity in the Itinerary as shown in LISTING 5-9. This function will wait for each call to RetrieveActivity() to complete before making another call making it's execution time linear.

Listing 5-9 Activity Sequential Retrieval

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

In the past if you wanted to parallelize this task, you would have had to use threads and hand off work to them. With the TPL, all you have to do now is to use a Parallel.ForEach that will take care of the threading for you as seen in Listing 5-10.

Listing 5-10 Activity Parallel retrieval

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

See Also   The .NET Framework 4.0 now includes the Parallel Linq libraries (in System.Core.dll). PLinq introduce the .AsParallel() extension to perform parallel operations in Linq queries. You can also easily enforce the treatment of a data source as if it was ordered by using the .AsOrdered() extensions. Some new thread safe collections have also been added in the System.Collections.Concurrent namespace. You can learn more about these new features from Parallel Computing on MSDN.

AppFabric Caching

PMN is a data-driven application getting its data from the application database and the Bing Maps Web services. One of the challenges that could be faced when building a Web application is to manage the needs the support a large number of users, performance and response time. The operations to the data store and to the services to search for activities can increase the server resources usage dramatically for items that are shared across many users. For example, many users have access to the public itineraries so displaying these will generate numerous the calls to the database for the same items. Implementing caching at the Web tier will help reduce the resources utilization at the data store and help mitigate latency for recurring searches to the Bing Maps Web services. Figure 5-32 shows the architecture for an application implementing a caching solution at the front end server.
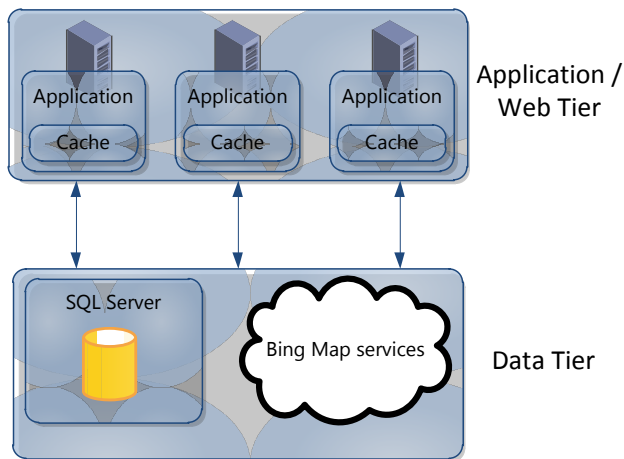
Figure 5-32 Typical Web application architecture

Using this approach would reduce the pressure on the data layer but the caching is still coupled to a specific server serving the request. Each Web tier server will have its own cache and it is still possible to end up with an uneven distribution of the processing to these servers.

Windows Server AppFabric Caching offers a distributed, in-memory cache platform. The AppFabric client library allows the application to access the cache as a unified view event if the cache is distributed across multiple computers as shown in Figure 5-33. The API provides simple get and set methods to retrieve and store any serializable CLR objects easily. The AppFabric cache allows adding cache computer on demand thus making it possible to scale in a manner that is transparent to the client. Another benefit is that the cache can also share copies of the data across the cluster protecting data against failure.
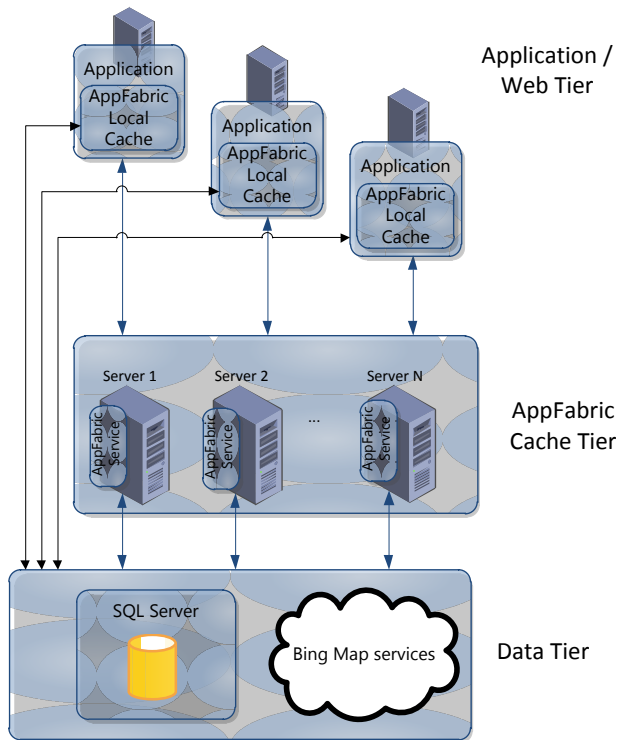
Figure 5-33   Web application using Windows Server AppFabric Caching

See Also   Windows Server AppFabric caching is available as a set of extensions to the .NET Framework 4.0. For more information about how to get, install and configure Windows Server AppFabric, please visit Windows Server AppFabric*.*

See Also   PMN can be configured to use either ASP.NET caching or Windows Server AppFabric caching. A complete walkthrough describing how to add Windows Server AppFabric caching to PMN is available here: PMN: Adding Caching using Velocity

Summary

In this chapter, you have use a few of the new Visual Studio 2010 features to structure the data layer of the PlanMyNight application using the Entity Framework v4.0 to access a database. You have also been introduced to the automated entity generation using the ADO.NET Entity Framework POCO templates and to the Windows Server AppFabric caching extensions. In the next chapter, you are going to explore how the ASP.NET MVC framework and the Managed Extensibility Framework can help you build great Web applications.

Chapter 6
From 2005 to 2010: Designing the Look and Feel

After reading this chapter, you will be able to

- Create an ASP.NET MVC controller that interacts with the data model

- Create an ASP.NET MVC View that displays data from the controller and validates user input

- Extend the application with an external plug-in using the Managed Extensibility Framework

Web application development in Microsoft Visual Studio has certainly made significant improvements over the years since ASP.NET 1.0 was released. Visual Studio 2005 brought things like more efficient View State, partial classes, and generic types (plus many others) to help developers create efficient applications that were easy to manage.

The spirit of improvement to assist developers in creating world-class applications is very much alive in Visual Studio 2010. In this chapter we will explore some of the new features as we add functionality to the Plan My Night companion application.

> **Note** The companion application is an ASP.NET MVC 2 project, but a Web developer has a choice in Visual Studio 2010 to use this new form of ASP.NET application, or the more traditional ASP.NET (referred to in the community as Web Forms for distinction). ASP.NET 4.0 has gotten many improvements to help developers and is still a very viable approach to creating Web applications.

We will be using a modified version of the companion application's solution to work our way through this chapter. If you installed the companion content in the default location, the correct solution can be found at: Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 6\ and look for a folder called UserInterface-Start.

Introducing the PlanMyNight.Web Project

The user interface portion of Plan My Night in Visual Studio 2010 was developed as an ASP.NET MVC application, the layout of which will differ from what a developer might be accustomed to when developing an ASP.NET Web Forms application in Visual Studio 2005. Some items in the project (as seen in Figure 6-1) will look familiar (like Global.asax), but others are completely new, and some of the structure is required by the ASP.NET MVC Framework.
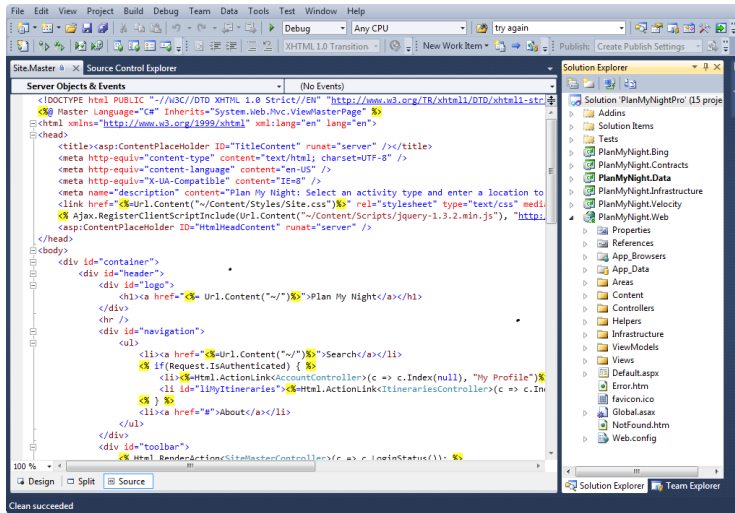
Figure 6-1 PlanMyNight.Web project view.

Items required by ASP.NET MVC are:

- **Areas**   This folder is used by the ASP.NET MVC framework to organize large Web applications into smaller components, without using separate solutions or projects. This feature is not used in the Plan My Night application but is called out because this folder is created by the MVC project template.

- **Controllers**   During request processing, the ASP.NET MVC framework looks for controllers in this folder to handle the request.

- **Views**   The Views folder is actually a structure of folders. The layer immediately inside the Views folder is named for each of the classes found in the Controllers folder, plus a Shared folder. The Shared subfolder is for common Views, Partial Views, Master Pages, and anything else to be available to all controllers.

See Also   More information about ASP.NET MVC's components, as well as how its request processing differs from ASP.NET Web Forms, can be found at http://asp.net/mvc.

In most cases, the web.config is the last file in a project's root folder. However, it has gotten a much needed update in Visual Studio 2010: Web.config Transformation. This feature allows for a base web.config file to be created but then have build-specific web.config files override the settings of the base at build, deployment, and run time. These files visually appear under the base web.config file, as seen in Figure 6-2.
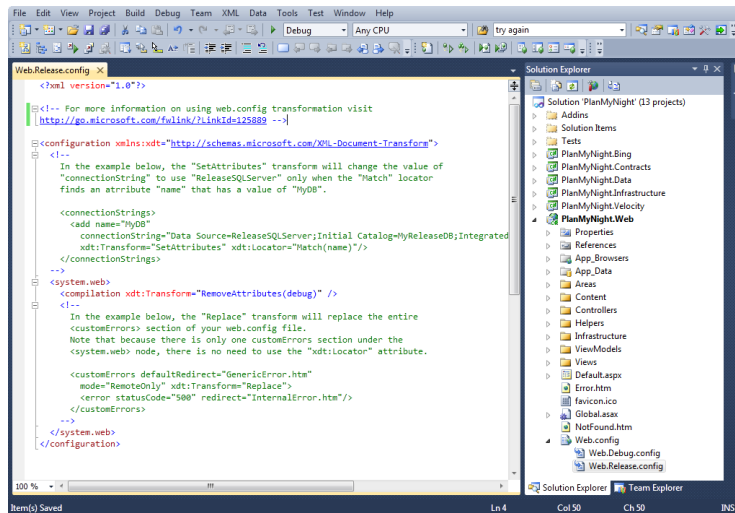
Figure 6-2 Web.config file with build-specific files expanded.

When working on a project in Visual Studio 2005, remember needing to remember not to overwrite the web.config with your debug settings? Or needing to remember to update the web.config when it was published for retail build with the correct settings? This is no longer an issue in Visual Studio 2010. Settings set in the web.config.retail file will be used during retail builds to override the values in web.config, and the same goes for the web.config.debug in debug builds.

Other sections of the project include

- **Content**   A collection of folders containing images, scripts and style files.

- **Helpers**   Miscellaneous classes, containing a number of Extension methods, adding functionality to types used in the project.

- **Infrastructure**   Items related to dealing with the lower level infrastructure of ASP.NET MVC (for example: Caching and Controller Factories, are contained in this folder.

- **ViewModels**   Data Entities filed out by Controller classes and used by Views to display data.

Running the Project

If you compile and run the project, you should see a screen similar to Figure 6-3:
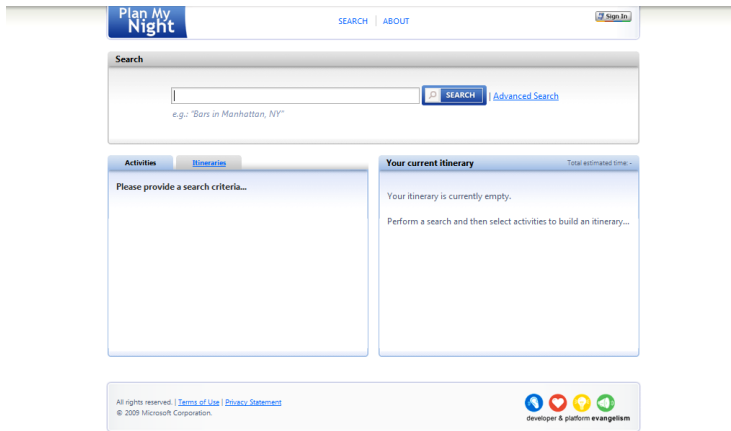
Figure 6-3   Default page of Plan My Night application.

The searching functionality and the ability to organize an initial list of itinerary items all works, but if you attempt to save the itinerary you are working on, or if you log in with Windows Live ID, the application will return a 404 Not Found error screen (as shown in Figure 6-4).

Figure 6-4 Error screen returned when logging into the Plan My Night application.

This is because currently the project does not include an account controller to handle these requests.

Creating the Account Controller

The *AccountController* class provides some critical functionality to the companion Plan My Night application:

- It handles signing users in and out of the application (via Windows Live ID).

- It provides actions for displaying and updating user profile information.

To create a new ASP.NET MVC controller:

1. Navigate the Solution explorer to the Controllers folder in the PlanMyNight.Web project, and click the right mouse button.

2. Open the Add submenu and select the Controller item.
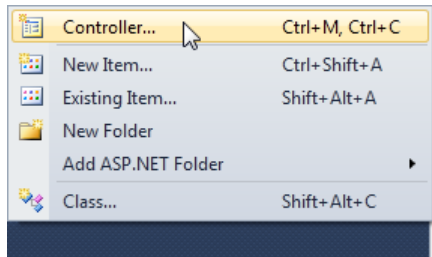
3. Fill in the name of the controller as **AccountController**.
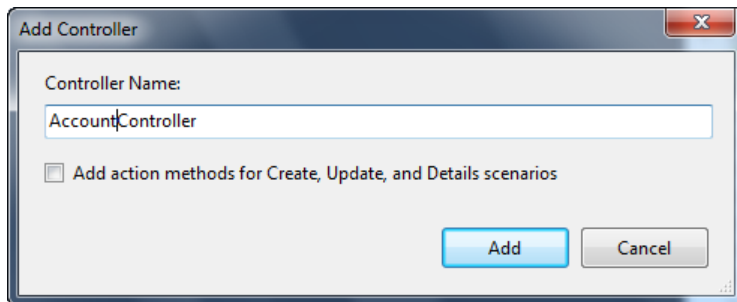
Note Leave the check box for "Add action methods for Create, Update and Delete scenarios" blank. Checking the box inserts some "starter" action methods, but because we will not be using the default methods, there is no reason to create them.

Once you have clicked the Add button in the Add Controller dialog box, you should have a basic *AccountController* class open, with a single *Index()* method in its body:

```
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   A difference to be noted from developing ASP,NET Web Forms applications
in Visual Studio 2005, is that ASP.NET MVC applications do not have a companion code behind file
for each of their aspx files. Controllers like the one we are currently creating perform the logic
required to process input and prepare output. This allows for a clear separation of display and
business logic, and is a key aspect of ASP.NET MVC.

Implementing the Functionality

To communicate with any of the data layer and services (the Model), we will need to add
some instance fields and initialize them. Before that, we need to add some namespaces to our
using block:

```
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 WindowsLiveId;
```

Now, let's add the instance fields. These fields are interfaces to the various section of our
Model.

```
public class AccountController : Controller
{
    private readonly IWindowsLiveLogin windowsLogin;
    private readonly IMembershipService membershipService;
    private readonly IFormsAuthentication formsAuthentication;
    private readonly IReferenceRepository referenceRepository;
    private readonly IActivitiesRepository activitiesRepository;
.
.
.
```

Note   Using interfaces to interact with all external dependencies allows for better portability of the code to various platforms. Also, during testing, dependencies can be mocked much easier, making for more efficient isolation of a specific component.

As mentioned, these fields represent parts of the Model this controller will interact with in order to meet its functional needs. The general descriptions for each of the interfaces are:

- **IWindowsLiveLogin**   Provides functionality for interacting with Windows Live ID service.

- **IMembershipService**   User Profile information and authorization methods. In our companion application, it is an abstraction of the ASP.NET Membership Service.

- **IFormsAuthentication**   ASP.NET Forms Authentication abstraction.

- **IReferenceRespository**   Reference resources, like lists of states and other model specific information.

- **IActivitiesRespository**   Interface for retrieving and updating activity information.

We are going to add two constructors to this class: one for general run time use, which utilizes the *ServiceFactory* class to get references to the needed interfaces, and one to enable tests to inject specific instances of the interfaces to use.

```
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;
}
```

Authenticating the User

The first real functionality we are going to implement in this controller is that of signing in and out of the application. Most of the methods that we'll implement later require authentication, so it makes a good place to start.

The companion application uses a few technologies together at the same time to give the user a smooth authentication experience: Windows Live ID, ASP.NET Forms Authentication, and ASP.NET Membership Services. These three technologies are utilized in the LiveID action we are going to implement next.

Start by creating the following method, in the *AccountController* class:

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

This method will be the primary action invoked when interacting with the Windows Live ID services. Right now, if it is invoked, it will just redirect the user to the root of the application.

> **Note**   The call to Redirect returns a *RedirectResult*, and while this example uses a string to define the target of the redirection, a number of overloads can be used for different situations.

A few different types of actions can be taken when Windows Live ID returns a user to our application. The user could be signing into Windows Live ID, could be signing out, or could be clearing the Windows Live ID cookies. Windows Live ID uses a query string parameter called *action* on the URL when it returns a user, so we will use a switch to branch the logic depending on the value of the parameter.

Add the following to the *LiveId* method:

```
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);
}
```

See also   Full documentation of the Windows Live ID system can be found on the http://dev.live.com/ Web site.

The code we just added handles the two sign-out actions for Windows Live ID. In both cases we use our *IFormsAuthentication* interface to remove the ASP.NET Forms Authentication cookie so that any future http requests (until they sign in again) will not be considered authenticated. In the second case we went one step further to clear the Windows Live ID cookies (the ones that remember your login name, but not your password).

Handling the sign-in scenario requires a bit more code, because we have to check whether the authenticating user is in our Membership Database and, if not, create a profile for them. However, before that, we must pass the data Windows Live ID sent us to our Windows Live ID interface so that it can validate the information and give us a *WindowsLiveLogin.User* object:

```
default:
    // login
    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);
```

At this point in the case for logging in, either the *liveIdUser* will be a reference to an authenticated *WindowsLiveLogin.User* object or it will be null. With this in mind we can add our next section of the code, which will take action when the *liveIdUser* value is not 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;
```

The call to the *ValidateUser* method on the *IMembershipService* reference allows the application to check whether the user has been to this site before and if there will be a profile for them. Because the user is authenticated with Windows Live ID, we are using their ID value (which is a guid) as both the user name and password to the ASP.NET Membership Service.

If the user does not have a user record with the application, we create one by calling the *CreateUser* method and then also have a user settings profile created via the *CreateProfile*. The profile is filled with some defaults, saved back to its store, and the user is redirected to the primary input page so that they can update the information.

> **Note** *Controller.RedirectToAction* determines which URL to create based off of the combination of input parameters. In this case we want to redirect the user to the *Index* action of this controller, along with passing the current return URL value.

The other action that takes place in this code is that the user is signed into ASP.NET Forms authentication so that a cookie will be created, providing identity information on future requests that require authentication.

The settings profile is managed by ASP.NET Membership Services as well and is declared in the web.config file of the application:

```
<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>
```

At this point the *LiveID* method is complete and should look like what follows below. The application can now take authentication information from Windows Live ID, prepare an ASP.NET MembershipService profile, and create an ASP.NET Forms Authentication ticket.

```
public ActionResult LiveId()
{
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:
        // login
        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("~/");
}
```

Of course, the user has to be able to get to the Windows Live ID login page in the first place before logging in. Currently in the Plan My Night application, there is a Windows Live ID login button. However, there are cases where the application will want the user to be redirected to the login page from code. To cover this scenario, we need to add a small method called *Login* to our controller:

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

This method simply retrieves the login URL for Windows Live and redirects the user to that location. This also satisfies a configuration value in our web.config file for ASP.NET Forms Authentication, in that any request requiring authentication will be redirected to this method:

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

Retrieving the Profile for the Current User

Now with the authentication methods defined, which satisfies our first goal for this controller—signing users in and out in the application—we can move on to retrieving data for the current user.

The *Index* method, which is the default method for the controller, will be where we retrieve the current user's data and return a view displaying that data. The *Index* method that was initially created when the *AccoutController* class was created should be replaced with the following:

```
[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**   Attributes may not have been in common use in Visual Studio 2005, however ASP.NET MVC makes use of them often. Attributes allow for metadata to be defined about the target they decorate. This allows for the information to be examined at run time (via reflection), and action taken if deemed necessary.

The *Authorize* attribute is very handy, because it declares that this method can be invoked only for http requests that are already authenticated. If the request is not authenticated, it will redirect to the ASP.NET Forms Authentication configured login target, which we just finished setting up. The *AcceptVerbs* attribute also restricts how this method can be invoked, by specifying which Http verbs can be used. In this case, we are restricting this method to HTTP GET verb requests. We've added a string parameter, *returnUrl*, to the method signature so that when the user is finished viewing or updating her information, she can be returned to what she was looking at previously.

> **Note**   This highlights a part of the ASP.NET MVC framework called Model Binding, details of which are out of scope for this book. However, you should know that it attempts to find a source for *returnUrl* (a form field, routing table data, or query string parameter with the same name) and binds it to this value when invoking the method. If the Model Binder cannot find a suitable source, the value will be null. This can cause problems for value types that cannot be null, because it will throw an *InvalidOperationException*.

The main portion of this method is fairly straightforward: it takes the return of the *GetCurrentProfile* method on the ASP.NET Membership Service interface and sets up a view model object for the view to use. The call to *GetReturnUrl* is an example of an extension method defined in the PlanMyNight.Infrastructure project. It's not a member of the Controller class, but in the development environment it makes for much more readable code.
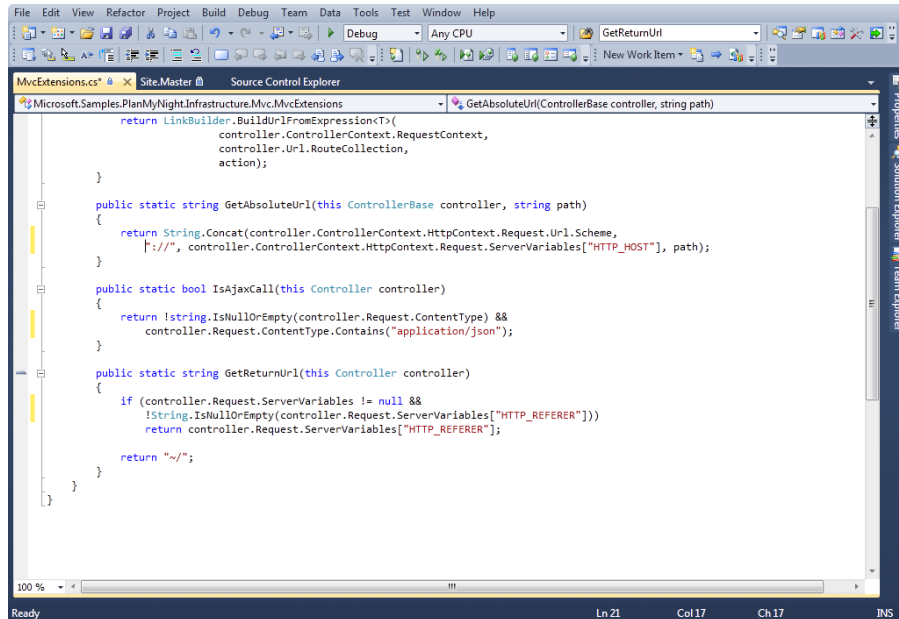
Visual Studio 2005   In Visual Studio 2005, extension methods did not exist, and rather than calling *this.GetReturnUrl()* and also having the method appear in the Intellisense for this object, you would have to type **MvcExtensions.GetReturnUrl(this)**, passing in the controller as a parameter. Extension methods certainly make the code more readable and do not require the developer to know the static class the extension method exists under. For the Intellisense to work, the name space does need to be listed in the *using* clauses.

InjectStatesAndActivityTypes is a method we do need to implement. It gathers data from the reference respository for names of stats and the activity repository. It makes two collections of *SelectListItem* (an HTML class for MVC): one for the list of States, and the other for the list of different activity types available in the application. It also sets the respective value.

```
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   In Visual Studio 2005, the *InjectStatesAndActivities* method takes longer to implement because a developer cannot utilize the LINQ extensions (the call to *Select()*) and Lambda expressions, which are a form of anonymous delegate the *Select* method applies to each member of the collection being enumerated. Instead the developer would have to write out his own loop and enumerate each item manually.

Updating the Profile Data

Having completed the infrastructure needed to retrieve data for the current profile, we can move on to updating the data in the model from a form submission by the user. After this we can create our View pages and see how all this ties together. The *Update* method is simple; however, it does introduce some new features not seen yet.

```
[Authorize()]
[AcceptVerbs(HttpVerbs.Post)]
[ValidateAntiForgeryToken()]
public ActionResult Update(UserProfile profile)
{
    var returnUrl = Request.Form["returnUrl"];
    if (!ModelState.IsValid)
    {
        // validation error
        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 });
```

```
        }
}
```

The *ValidateAntiForgeryToken* attribute ensures that the form has not been tampered with. To utilize this feature, we will need to add an *AntiForgeryToken* to our View's input form. The check on the *ModelState* to see if it is valid is our first look at input validation. This is a look at the server-side validation, and ASP.NET MVC offers a very easy-to-use feature to make sure that incoming data meets some rules. The *UserProfile* object that is created for input to this method, via MVC Model Binding, has had one of its properties decorated with a *System.ComponentModel.DataAnnotations.Required* attribute. During Model Binding, the MVC framework will evaluate *DataAnnotation* attributes and mark the *ModelState* as valid only when all of the rules pass.

In the case where the *ModelState* is not valid, the user is redirected to the *Index* method where the *ModelState* will be used in the display of the input form. Or, if the request was an AJAX call, a *JsonResult* is returned with the *ModelState* data attached to it.

> **Visual Studio 2005**   Because in ASP.NET MVC requests are routed through controllers rather than pages, the same URL can handle a number of different requests and respond with the appropriate view. In Visual Studio 2005, a developer would have to create two different URLs and call a method in a third class to perform the functionality.

When the *ModelState* is valid, the profile is updated in the membership service and a JSON result is returned for AJAX requests with the success data, or in the case of "normal" requests, the user is redirected to the *UpdateSuccess* action on the Account controller. The *UpdateSuccess* method is the final method we need to implement to finish off this controller:

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

The method will be used to return a success view to the browser, display some of the updated data, and provide a link to return the user to where she was when she started the profile update process.

Now that we've reached the end of the Account controller implementation, you should have a class that resembles this listing:

```
namespace Microsoft.Samples.PlanMyNight.Web.Controllers
{
    using System;
```

```csharp
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;

[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();
```

```csharp
                    string type;
                    byte[] content;
                    this.windowsLogin.GetClearCookieResponse(out type, out content);
                    return new FileStreamResult(new MemoryStream(content), type);
                default:
                    // login
                    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)
    {
        // validation error
        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;
        }
    }
}
```

Creating the Account View

In the previous section, we created a controller with functionality that allows a user to update his or her information and view it. In this section we're going to walk through the Visual Studio 2010 features that enable us to create the Views that display this functionality to the user.

To create the Index view for the Account controller:

1. Navigate to the Views folder in the PlanMyNight.web project.

2. Click the right mouse button on the Views folder, expand the Add submenu, and select New Folder.

3. Name the new folder Account.

4. Click the right mouse button on the new Account folder, expand the Add submenu, and select View.

5. Fill out the Add View dialog box as shown here:
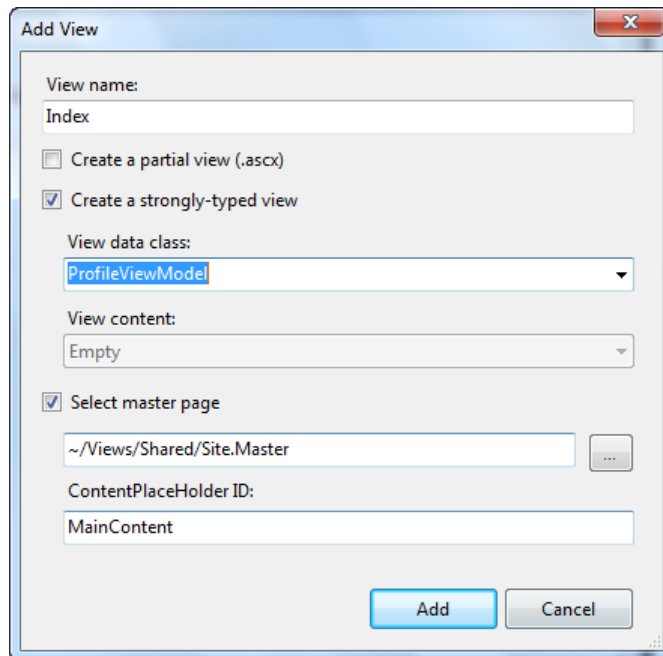
6. Click OK, and now you should be looking at an HTML page with some *<asp:Content>* controls in the markup:
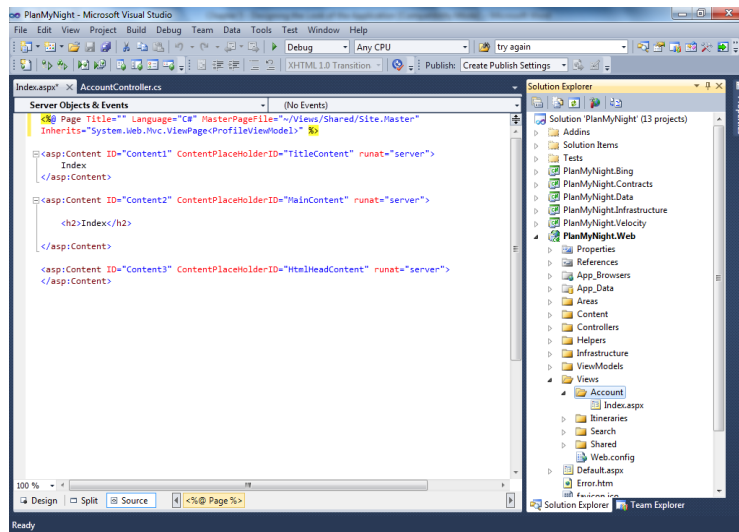
You might notice that it doesn't look much different from what you are used to in Visual Studio 2005. By default, ASP.NET MVC 2 uses the ASP.NET Web Forms view engine, so there will be some commonality between MVC and Web Forms pages. The primary differences at this point are that the *page* class derives from *System.Web.Mvc.ViewPage<ProfileViewModel>* and there is no code behind file. MVC does not utilize code-behind files, like ASP.NET Web

Forms does, to enforce a strict separation of concerns. MVC pages are generally edited in markup view; the designer view is primarily for ASP.NET Web Forms applications.

In order for this page skeleton to become the main view for the Account controller, we should change the title content to be more in line with the other views:

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

Next we need to add the client scripts we are going to use in the content placeholder for the *HtmlHeadContent*:

```
<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>
```

This script makes use of extension methods for the *System.Web.Mvc.AjaxHelper*, which are found in the PlanMyNight.Infrastructure project, under the MVC folder.

With the head content setup, we can look at main content of the view:

```
<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>
```

Aside from some inline code, this looks to be fairly normal HTML markup. We're going to focus our attention on the inline code pieces to demonstrate the power they bring (as well as the simplicity).

> **Visual Studio2005**   In Visual Studio 2005, it was more common place to use server side controls to display data, and other display time logic. However since ASP.NET MVC view pages do not have a code behind file, it must be done in the same file with the markup. ASP.NET Web Forms controls can still be used, our example makes use of the <asp:Content> control, however their functionality is generally limited because there is no code behind file.

MVC makes a lot of use of what is known as HTML helpers. The methods contained under *System.Web.Mvc.HtmlHelper* emit small, standards-compliant HTML tags for various uses. This requires the MVC developer to type more markup than a Web Forms developer in some cases, but they have more direct control over the output. The strongly typed version of this extension class (*HtmlHelper<TModel>*), can be referenced in the view markup via the *ViewPage<TModel>.Html* property.

These are the *HTML* methods used in this form, which are only a fraction of what is available by default:

- *Html.EnableClientValidation()* enables data validation to be performed on the client side based on the strongly typed ModelState dictionary.

- *Html.BeginForm* places a *<form>* tag in the markup and closes the form at the end of the *using* section. It takes various parameters for options, but the most common is the name of the Action and the controller to invoke that action on. This allows the MVC Framework

to generic the specific url to target the form to at run time, versus having to input a string url into the markup.

- *Html.AntiForgeryToken* places a hidden field in the form with a check value that is also stored on the server side and validated when the target of the form has the *ValidateAntiForgeryToken* attribute. Remember that we added this attribute to the *Update* method in the controller.

- *Html.EditorFor* is an overloaded method that inserts a text box into the markup. This is the strongly typed version of the *Html.Editor* method.

- *Html.DropDownListFor* is an overloaded method that places a drop-down list into the markup. This is the strongly typed version of the *Html.DropDownList* method.

- *Html.ValidationMessage* is a helper that will display a validation error message when a given key is present in the ModelState dictionary.

- *Html.Hidden* places a hidden field in the form, with the name and value that is passed in.

- *Html.Submit* creates a Submit button for the form.

> **Note**   With the Index view markup complete, we only need to add the view for the UpdateSucess action before we can see our results.

To Create the UpdateSuccess view:

1. Expand the PlanMyNight.Web project in the Solution explorer, and then expand the Views folder.
2. Click the right mouse button on the Account folder.
3. Open the Add submenu, and click View.
4. Fill out the View creation dialog box so that it looks like this:
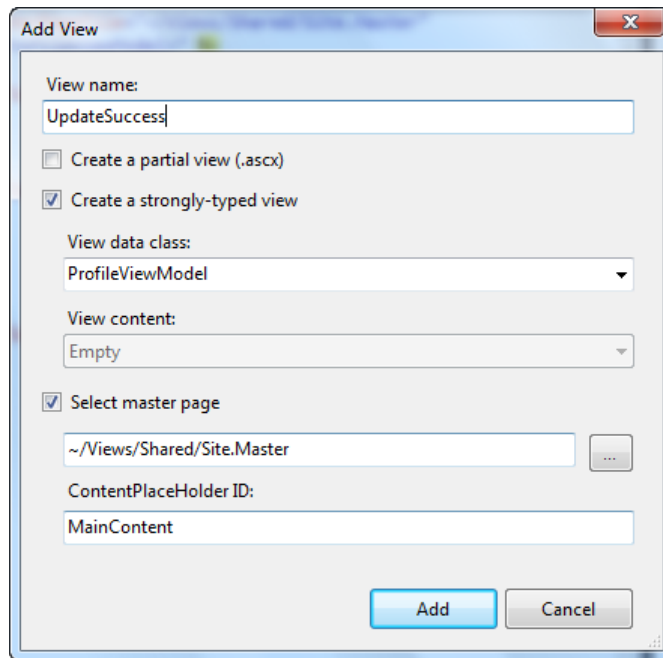
Once the view page is created, fill in the title content so that it looks like this:

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

And the placeholder for *MainContent* should look like this:

```
<asp:Content ContentPlaceHolderID="MainContent" runat="server">
<div class="panel" id="profileForm">
    <div class="innerPanel">
        <h2><span>My Profile</span></h2>
        <div class="items">
            <p>You 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>
```

With this last view created, we can now compile and launch the application. Click the Sign In button, as seen in the top right corner of Figure 6-6, and sign in to Windows Live ID.
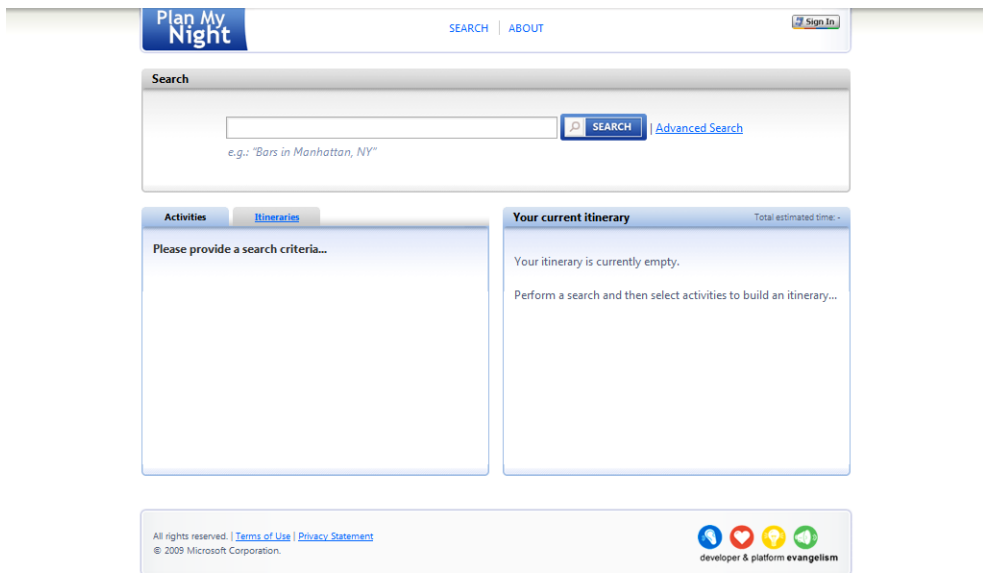
Figure 6-6   Plan My Night default screen.

Once you've signed in, you should be redirected to the Index view of the Account controller we created, shown in Figure 6-7.
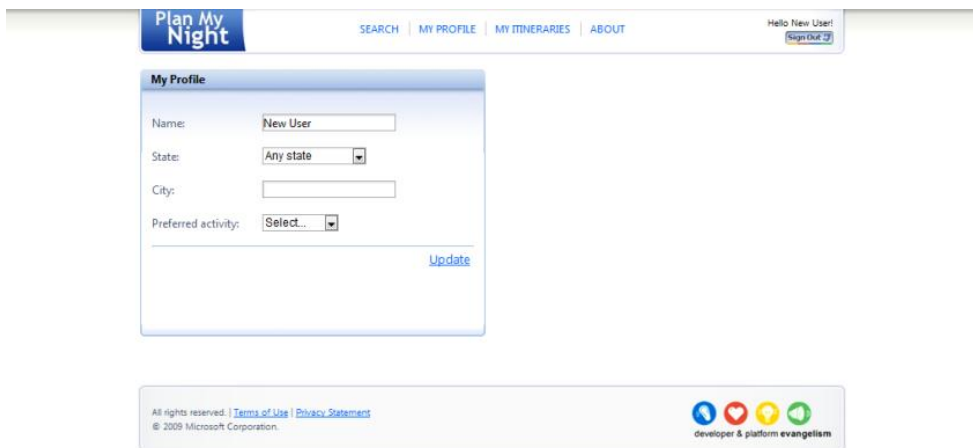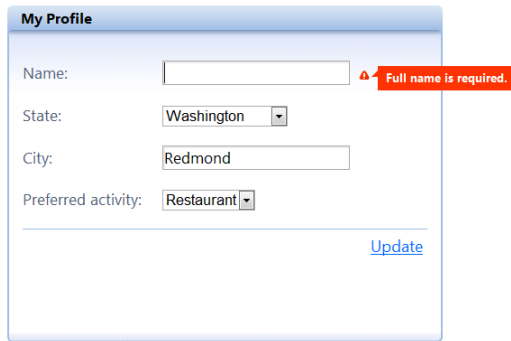
Figure 6-7   Profile settings screen returned from the *Index* method of the Account controller.

If instead it returned you to the search page, just click the My Profile link, located in the links at the center and top of the interface. To see the new data validation features at work, try to

save the form without filling in the Full Name field. You should get a result that looks like Figure 6-8:

Figure 6-8   Example of failed validation during Model Binding checks.

Because we enabled client-side validation, there was no post back. To see the server-side validation work, we would have to edit the Index.aspx file in the Account folder and comment out the call to *Html.EnableClientValidation*. The tight integration and support of AJAX and other JavaScript in MVC applications allows for server-side operations like validation to be moved to the client side much more easily than they were previously.

Visual Studio 2005   In ASP.NET MVC applications, the value of the ID attribute for a particular HTML element are not transformed, like they are in ASP.NET Web Forms 2.0. In Visual Studio 2005, a developer would have to make sure to set the *UniqueID* of a control/element into a JavaScript variable so that it could be accessed by external JavaScript. This was done to make sure the ID was unique. However, it was always an extra layer of complexity to the interaction between ASP.NET 2.0 Web Forms controls and JavaScript. In MVC this transformation does not happen, but it is up to the developers to ensure uniqueness of the ID. It should also be noted that ASP.NET 4.0 Web Forms now supports disabling the ID transformation on a per-control basis, if the developer so wishes.

With the completed Account controller and related views, we have filled in the missing "core" functionality of Plan My Night, while taking a brief tour of some of the new features in Visual Studio 2010 and MVC 2.0 applications. But MVC is not the only choice for Web developers. ASP.NET Web Forms has been the primary application type for ASP.NET since it was released, and it continues to be improved ipon in Visual Studio 2010. In the next section we'll explore creating an ASP.NET Web form with the Visual Designer to be used in the MVC application.

Using the Designer View to Create a Web Form

Applications will encounter an unexpected condition at some point in their lifetime of use. The companion application is no different, and when it does encounter an unexpected condition, it returns an error screen like that shown in Figure 6-9.
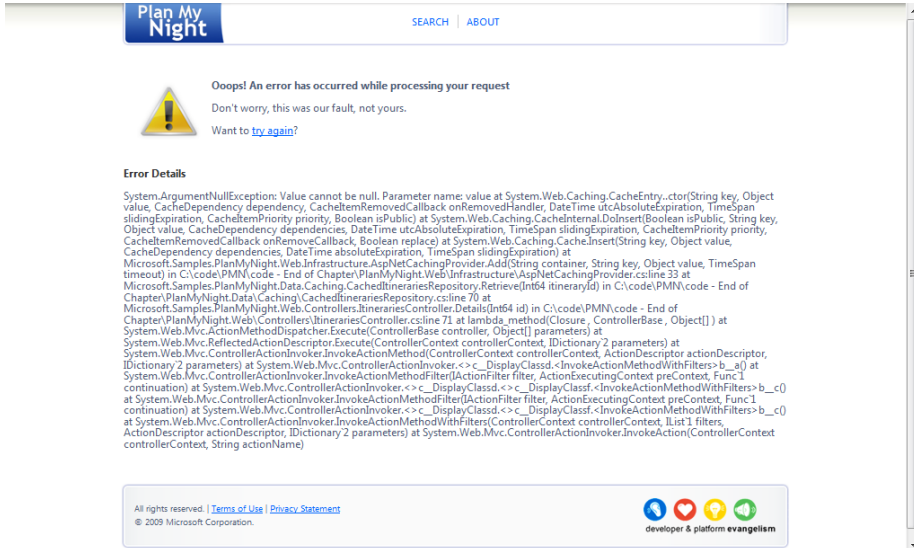

Figure 6-9 Example of an error screen in the Plan My Night Application.

Currently, when a user sees this screen, they really have only the option of trying their action again or using the navigation links along the top area of the application. (Of course, that might also cause another error.) Adding an option for the user to provide feedback would allow the developers to gain information about the situation that might not be apparent by the standard exception message and stack trace. To show a different way to create a user interface component for Plan My Night, the error feedback page is going to be created as an ASP.NET Web Form using primarily the Designer view in Visual Studio. Before we can begin designing the form, we need to create a base form file to work from.

To create a new Web form:

1. Open the context menu on the PlanMyNight.Web project (via clicking the right mouse button), open the Add submenu, and select New Item.

2. In the Add New Item dialog box, select Web Form Using Master Page and call the item **ErrorFeedback.aspx** in the Name field.
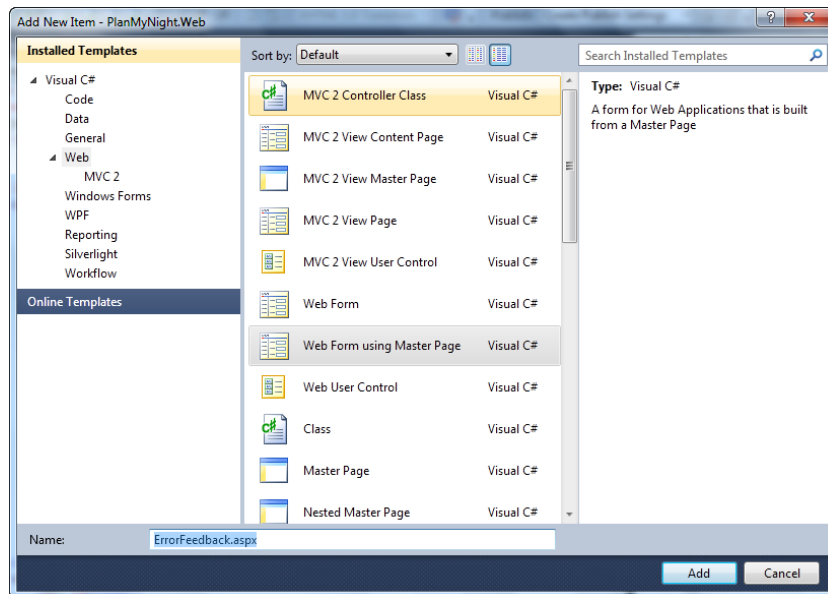
3. The dialog screen to associate a master page with this Web form will appear. On the Project Folders side, ensure that the main PlanMyNight.web folder is selected and then select the WebForms.Master item on the right.
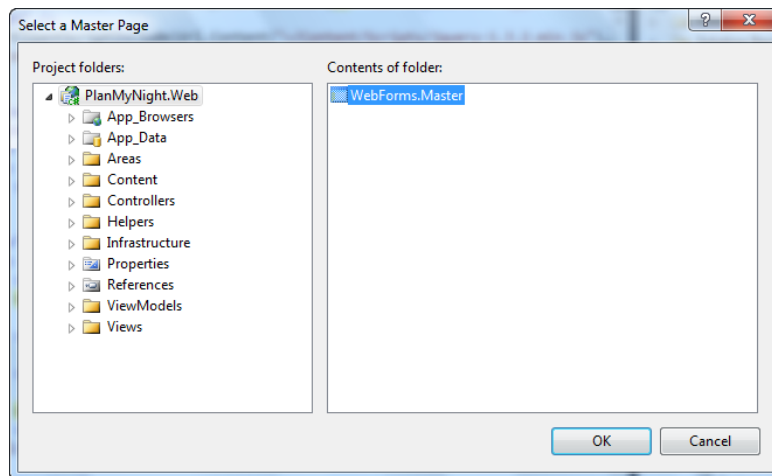
4. The resulting page may be shown in source mode (or Design) instead of Split. Switch the view to Split (bottom of the window, just like in previous Visual Studio versions). At the end the screen should look similar to this:
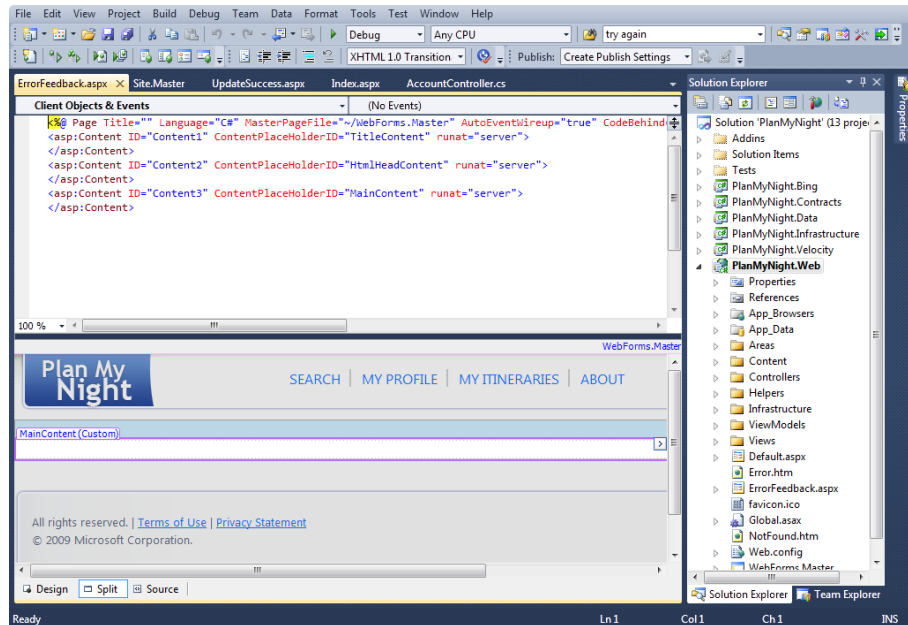
It's a good idea to pin the control Toolbox open on the screen because we will be dragging controls and elements to the content area during this section. The Toolbox, if not present already, can be found under the View menu.

Start by dragging a div element (under the HTML group) from the tool box into the MainContent section of the designer. A div tab will appear, indicating that the new element we added is the currently selected element. Open the context menu for the div, and choose Properties (which can also be opened with the F4 key). With the Properties window open, edit the *(Id)* property to have a value of *profileForm***.** (Casing is important.) Also, change the *Class* property to have a value of *panel*. After editing the values, the size of our content area will have changed, because css is applied in the designer view.

Visual Studio 2005   A much needed update to the Web Forms designer surface from Visual Studio 2005, is the application of CSS. This allows the developer to see in real-time how the style changes are applied, without having to run the application. When viewed in Visual Studio 2005, the designer for the search.aspx page will appear similar to the following image.
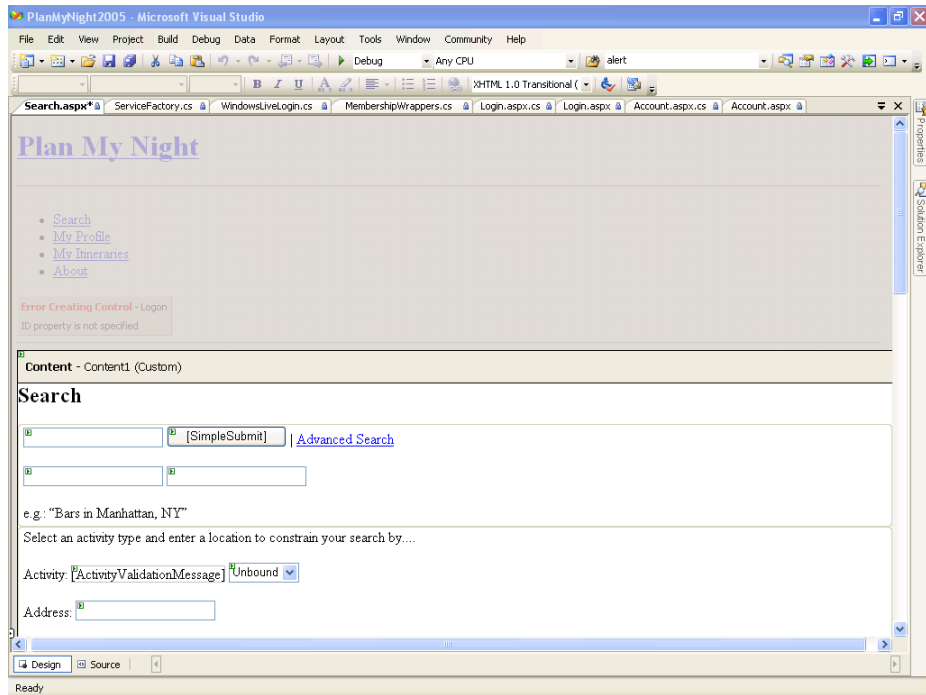
Drag another div inside in the first, and set its *class* property to *innerPanel*. In the markup panel add the following markup to the *innerPanel*:

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

After the close of the <h2> tag, add a new line and open the context menu. Choose Insert Snippet, and follow the click path of ASP.NET > formr. This will create a server-side form tag for us to insert Web controls into. Inside the form, place a div with class of *items* and a fieldset tag.

Next drag a TextBox control (found under Standard) from the Toolbox and drop it inside the fieldset tag. Set the ID of the text box to *FullName*. Add a <label> tag before this control in the markup view, set its *for* property to the ID of the textbox and its value to *Full Name:* (note the colon). Surround these two elements with a <p>, and you should have something like Figure 6-11 in the design view.
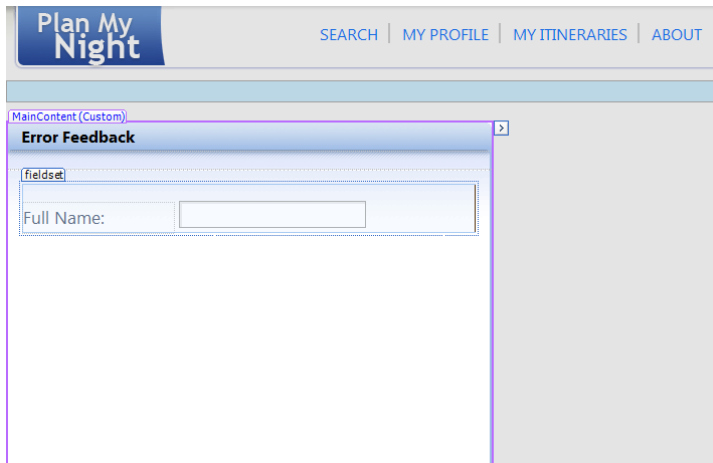
Figure 6-11 Current state of ErrorFeedback.aspx in the designer view.

Add another text box and label in a similar manner as the first, but set the ID of the text box to *EmailAddress* and the label value to *Email Address:*. Repeat the process a third time, setting the TextBox ID and label value to *Comments*. There should now be three labels and three single line TextBox controls in the Design view. The Comments control needs multiline input, so open its property page and set *TextMode* to *Multiline*, *Rows* to *5*, and *Columns* to *40*. This should create a much wider text box in which the user can enter comments

Use the Insert Snippet feature again, after the Comments Text box, and insert a "div with class" tag (HTML>divc). Set the class of the div tag to *submit*, and drag a Button control from the toolbox into this div. Set the Button's *Text* property to *Send Feedback*.

The designer should show something similar to what is in Figure 6-12, and at this point we have a page that will submit a form.

Figure 6-12 ErrorFeedback.aspx form with complete field set.

However, it does not perform any validation on the data being submitted. To do this, we are going to take advantage of some of the Validation controls present in ASP.NET. We are going to make the Full Name and Comments boxes required fields and perform a regex validation of the email address to ensure that it matches the right pattern.

Under the Validation group of the toolbox, there are some premade validation controls we will use. Drag a RequiredFieldValidator object from the toolbox and drop it to the right of the Full Name text box. Open the properties for the validation control and set the *ControltoValidate* property to *FullName*. (It's a drop-down list of controls on the page.) Also, set the *CssClass* to *errorShort*. This will change the display of the error to a red triangle used elsewhere in the application. Finally, change the Error Message to say "Name is required".
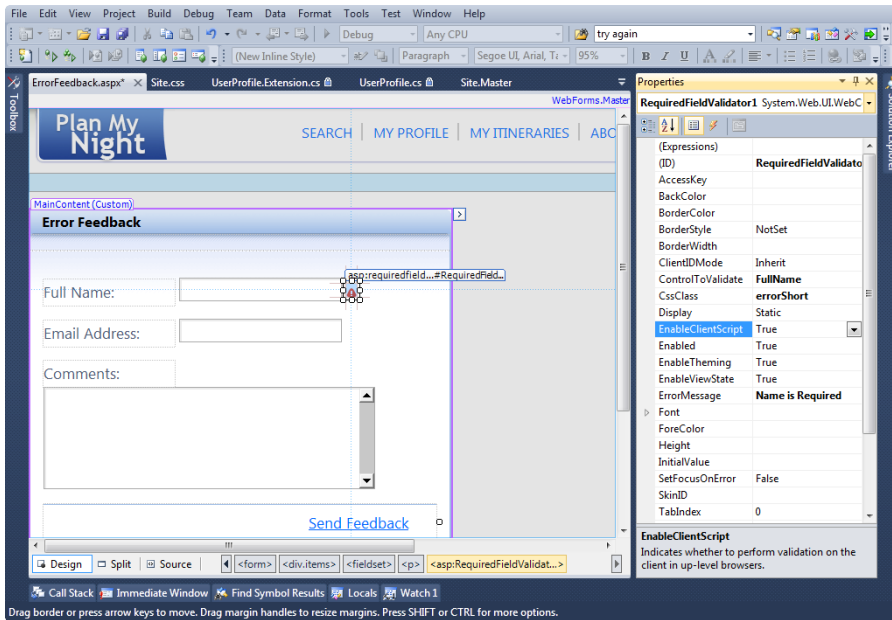
Repeat these steps for the Comments box, but substitute the *language* and *property* values as appropriate.

For the email address field, we want to make sure the user types in a valid email address, so for this field drag a RegularExpressionValidator control from the toolbox and drop it next to the Email Address text box. The property values are similar for this control in that we set the *ControlToValidate* property to *EmailAddress* and the *CssClass* property to *errorShort*, but with this control we define the regular expression to be applied to the input data. This is done with the *ValidationExpression* property, and it should be set like this:

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

The error message for this validator should say something like "Must enter a valid email address".

Finally, we need a place to display the error messages encountered during validation of the form. The ValidationSummary control takes care of this automatically once it is placed in the form. Drag one from the Toolbox, and drop it at the beginning of the form.

The form is complete. To see it in the application, we need to add the option of providing feedback to a user when they encounter an error. In the solution explorer, navigate the PlanMyNight.web project tree into the Views folder and then into the Shared subfolder. Open the Error.aspx file in the markup viewer, and go to line 35. This is the line of the error screen where we ask the user if they want to try their action again and where we will put the option

for sending the feedback. After the question text in the same paragraph, add the following markup:

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

This will add an option to go to the form we just created whenever there is a general error in the MVC application. To see our form, we are going to have to cause an error in our application.

To cause an error in the Plan My Night application:

1. Start the application.

2. Once the default search page is up, type the following into the browser address bar: http://www.planmynight.net:48580/Itineraries/Details/38923828

3. Since it is highly unlikely such an itinerary id exists in the database, an error screen will be shown.
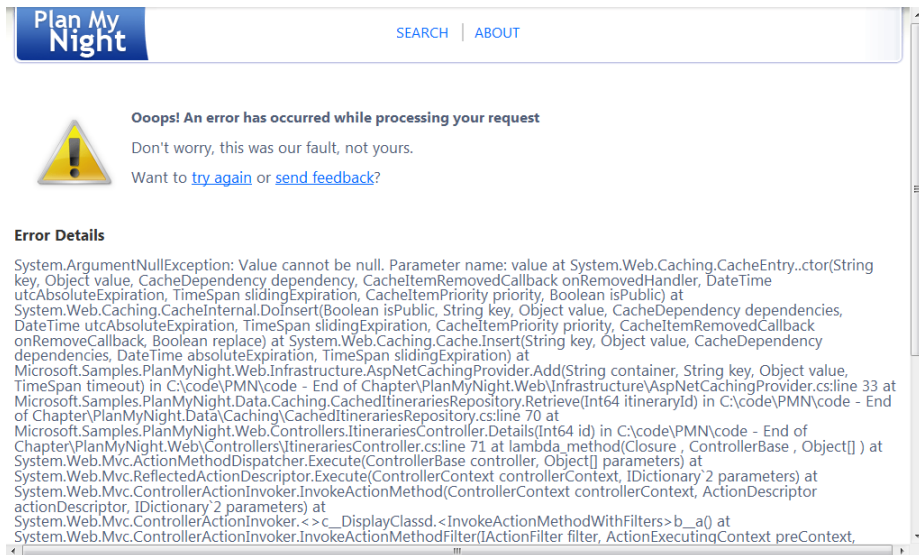
4. With the error screen visible, click the link to go to our feedback form. Try to submit the form with invalid data.

ASP.NET uses client-side script (when the browser supports it) to perform the validation, so no post backs are occurring until the data passes. On the server side, once it does receive a post back, a developer can check the validation state with the *Page.IsValid* property in the code behind. However, because we used client-side validation (on by default), this will always be true. The only code in the code behind that needs to be added would be to redirect the user on a post back (and checking the *Page.IsValid* property, in case client validation missed something).

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

This really isn't very useful to the user, but our goal in this section was to work with the designer to create an ASP.NET Web form. This added a new interface to the PlanMyNight.Web project, but what if we wanted to add new functionality to the application in a more modular sense, some degree of functionality that could be added or removed without having to compile the main application project. This is where an extensibility framework like the Managed Extensibility Framework (MEF) can show the benefits it brings.

Extending the Application with MEF

A new technology available in Visual Studio 2010 as part of the .NET Framework 4 is the Managed Extensibility Framework. The Managed Extensibility Framework provides developers with a simple (yet powerful) mechanism to allow their applications to be extended by 3rd parties after the application has been shipped. Even within the same application, MEF allows developers to create applications, which completely isolate components, allowing them to be managed or changed independently. It utilizes a resolution container to map components that provide a particular function (Exporters) and components that require that functionality (importers), without the two concrete components having to know about each other directly. Resolutions are done on a contract basis only, which easily allows components to be interchange, or introduced to an application with very little overhead.

See also   *MEF's community Web site, containing in-depth details about the architecture, can be found at http://mef.codeplex.com.*

The companion Plan My Night application has been designed with extendibility in mind, and it has three "add-in" module projects in the solution, under the Addins solution folder.
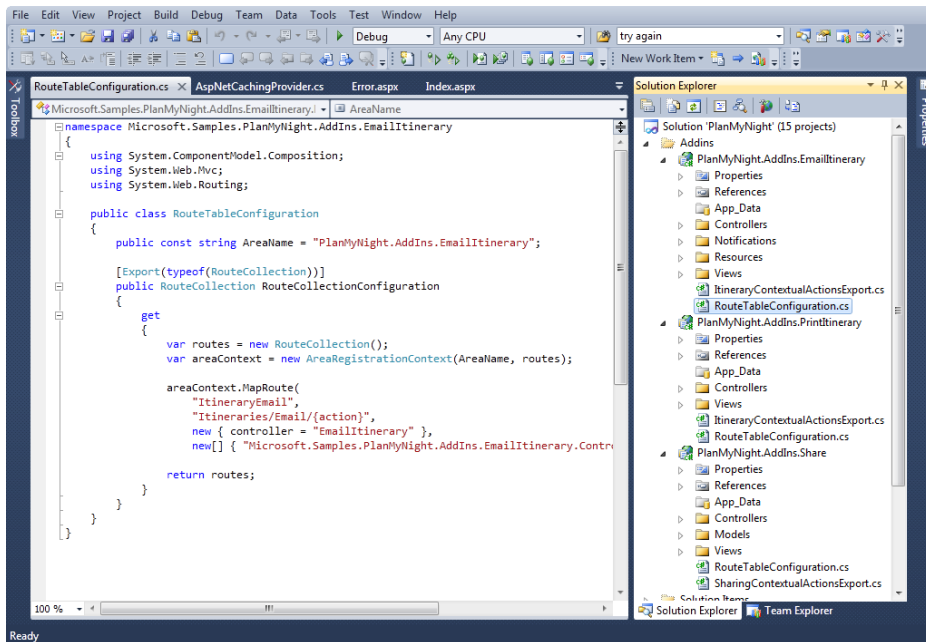
PlanMyNight.Addins.EmailItinerary adds the ability to email itinerary lists to anyone the user sees fit. PlanMyNight.Addins.PrintItinerary provides a printer-friendly view of the itinerary. Lastly, PlanMyNight.Addins.Share adds in social media sharing functions (so that the user can post a link to an itinerary) as well as url-shortening operations. None of these projects reference the main PlanMyNight.Web application or are referenced by it. They do have references to the PlanMyNight.Contracts and PlanMyNight.Infrastructure projects, so they can export (and import in some cases) the correct contracts via MEF as well as utilize any of the custom extensions in the infrastructure project.

> **Note**   Before doing the next step, if the Web application is not already running, launch the PlanMyNight.web project so that the UI is visible to you.

To add the modules to our running application, run the DeployAllAddins.bat file, found in the same folder as the PlanMyNight.sln file. This will create new folders under the Areas section of the PlanMyNight.Web project. These new folders, one for each plug in, will contain the files needed to add their functionality to the main web application. The plug ins appear in the application as extra options under the current itinerary section of the search results page and on the itinerary details page. Once the batch file is finished running, go to the interface for PlanMyNight, search for an activity, and add it to the current itinerary. You should notice some extra options under the itinerary panel other than just New and Save.
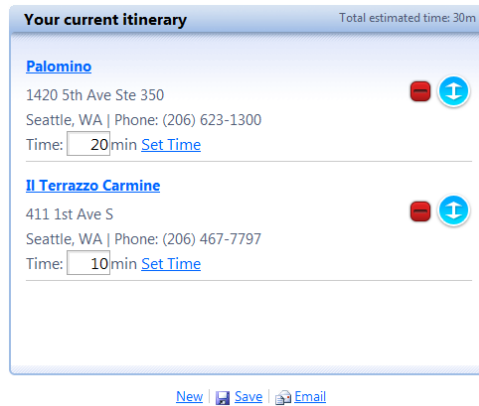
The social sharing options will show in the interface only after the itinerary is saved and marked public.

Visual Studio 2005   Visual Studio 2005 does not have anything that compares to MEF. In order to support "plug-ins," a developer would have to either write the plug in framework from scratch or purchase a commercial package. Either of two options led to proprietary solutions in which an external developer would have to understand in order to create a component for them. Adding MEF to the .NET Framework, it helps to cut down the entry barriers to producing extendible applications and the "plug in" modules for them.

Print Itinerary Addin Explained

To demonstrate how these plug-ins wire into the application, let's have a look at the PrintIntinerary.Addin project. When you expand the project, you should see something like the following structure:
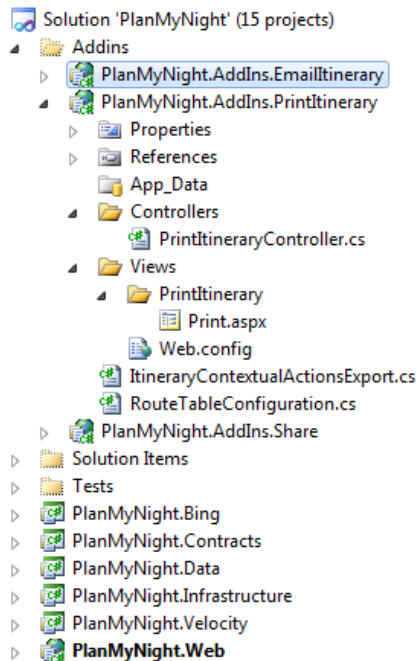
Some of this structure is similar to the PlanMyNight.Web project (Controllers and Views), and that's because this add-in is going to be placed in an MVC application as an Area. If we have a closer look at PrintItineraryController.cs file in the Controller folder, we can see that it is extremely similar in structure to the controller we created earlier in this chapter (and any of the other controllers in the Web application). However, some key differences set it apart from the controllers that are compiled in the primary PlanMyNight .web application.

Focusing on the class definition, we will notice some extra attributes:

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

These two attributes describe this type to the MEF resolution container. The first attribute, *Export*, marks this class as providing an IController under the contract name of PrintItinerary. The second attribute declares that this object only supports nonshared creation and cannot be created as a shared/singleton object. Defining these two attributes are all you need to do to have the type used by MEF. In fact, *PartCreationPolicy* is an optional attribute, but it should be defined if the type cannot handle all the creation policy kinds.

Further into the PrintItineraryController.cs file, the constructor is decorated with an ImportingConstructor attribute:

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

The *ImportingConstructor* attribute informs MEF to provide the parameters when creating this object, . In this particular case, MEF will provide an instance of *IServiceFactory* for this object to use. Where the instance comes from is of no concern to the *this* class and really assists with creating modular applications. For our purposes, the *IServiceFactory* contracted is being exported by the ServiceFactory.cs file in the PlanMyNight.web project.

The RouteTableConfiguration.cs file registers the url route information that should be directed to the PrintItineraryController. This route, and the routes of the other add-ins, are registered into the application during the *Application_Start* method in the Global.asax.cs file of PlanMyNight.Web:

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

// Register routes from Addins
foreach (RouteCollection routes in container.GetExportedValues<RouteCollection>())
{
    foreach (var route in routes)
    {
        RouteTable.Routes.Add(route);
    }
}
```

The *controllerFactory*, which was initialized with a MEF container containing path information to the Areas subfolder (so that it enumerated all the plug-ins), is assigned to be the controller factory for the lifetime of the application. This will allow controllers imported via MEF to be usable anywhere in the application. The routes these plug-ins repond to are then retrieved from the MEF container and registered into the MVC routing table.

The ItineraryContextualActionsExport.cs file exports information to create the link to this plug-in, as well as metadata for displaying it. This information is used in the ViewModelExtensions.cs, in the PlanMyNight.web project, when building a view model for display to the user:

```
// get addin links and toolboxes
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 }));
```

The call to *AddinExtentions.GetLinksFor* will enumerate over exports in the MEF Export provider and return a collection of them to be added to the local *addinLinks* collection. These are then used in the View to display more options when they are present.

Summary

In this chapter we've explored a few of the many new features and technologies found in Visual Studio 2010 that were used to create the companion Plan My Night application. We've walked through creating a controller and its associated view and how the ASP.NET MVC framework offers Web developers a very powerful option for creating Web applications. We've also explored how using the Managed Extensibility Framework in application design can allow plug-in modules to be developed external to the application and loaded at run time. In the next chapter we will explore how debugging applications has been improved in Visual Studio 2010.

Chapter 7

From 2005 to 2010: Debugging an Application

After reading this chapter, you will be able to

- Use the new debugger features coming with Microsoft Visual Studio 2010

- Create unit tests and execute them in Visual Studio 2010

- Compare what was available to you as a developer in Visual Studio 2005

As we are writing this book we have realized how much the debugging tools and developer aids have evolved over the last three versions of Visual Studio. Focusing on debugging an application and writing unit tests just reinforce the chance we have to be able to work with Visual Studio 2010.

Visual Studio 2010 Debugging Features

In this chapter you will go through the different debugging features using a modified Plan My Night application. If you installed the companion content at the default location you will find the modified Plan My Night application at the following location: *%userprofile%*\Documents\Microsoft Press\Moving to Visual Studio 2010\Chapter 7\DebuggerStart\Code and double-click the **PlanMyNight.sln** file.

First of all before diving into the debugging session itself you will need to setup a few things.

1. In **Solution Explorer**, ensure that **PlanMyNight.Web** is the startup project. If the project name is not in bold, right click on **PlanMyNight.Web** and select **Set as StartUp Project.**

2. In the **PlanMyNight.Web** solution open the **Global.asax.cs** file by clicking the triangle beside the **Global.asax** folder and then double-clicking the **Global.asax.cs** file as shown in Figure 7-1.
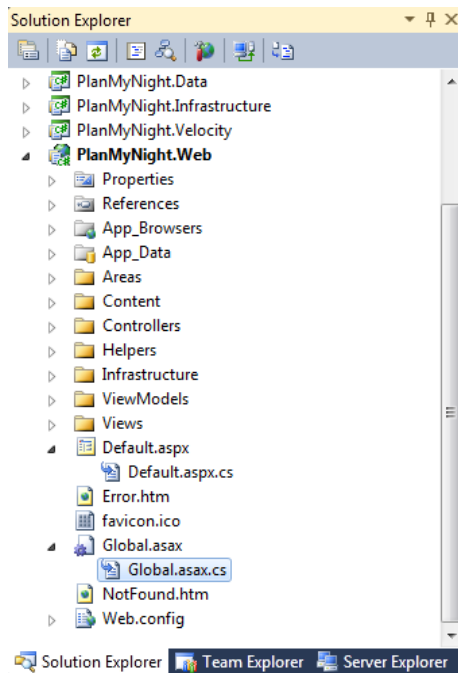
Figure 7-1 Solution Explorer before opening the file Global.asax.cs

Managing Your Debugging Session

Using the Plan My Night application you will examine how a developer can manage and share breakpoints. And with the use of new breakpoints enhancements you will learn how to inspect the different data elements in the application in a much faster and efficient way. You will also look at new minidumps and the addition of a new IL interpreter that allows the evaluation of managed code properties and functions during minidump debugging.

New Breakpoints Enhancements

Now you should have the Global.ascx.cs file opened in your editor and using the application you will examine how a developer can manage and share breakpoints. And with the use of new breakpoints enhancements you will learn how to inspect the different data elements in the application in a much faster and efficient way.

1. Navigate to the Application_BeginRequest(object sender, EventArgs e) method and set a breakpoint on the line that reads var url = HttpContext.Current.Request.Url; by clicking in the left hand margin or by pressing F9. Look at Figure 7-2 to see this in action.
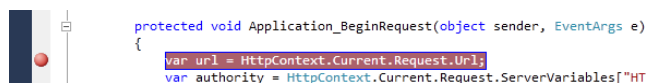
Figure 7-2 Creating a breakpoint

2. Press F5 to start the application in debug mode. You should see the developer web server starting in the system tray and a new browser window opening. The application should immediately stop at the breakpoint you just created. It is possible that the **Breakpoints** window is not visible even after starting the application in debug mode. If it is the case you can make it visible by going to the **Debug** menu and selecting **Windows** and then **Breakpoints** or you can use the keyboard shortcut: **Ctrl-Alt-B**.

3. You should now see the Breakpoints window as shown in Figure 7-3.
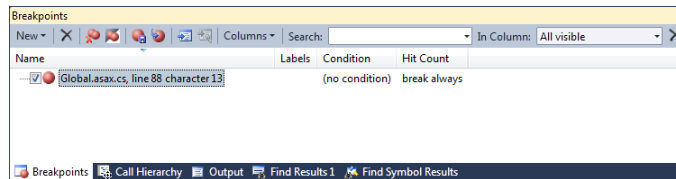
Figure 7-3 Breakpoints Window.

4. In the same method add three more breakpoints so that the editor and the Breakpoints window look like in Figure 7-4.
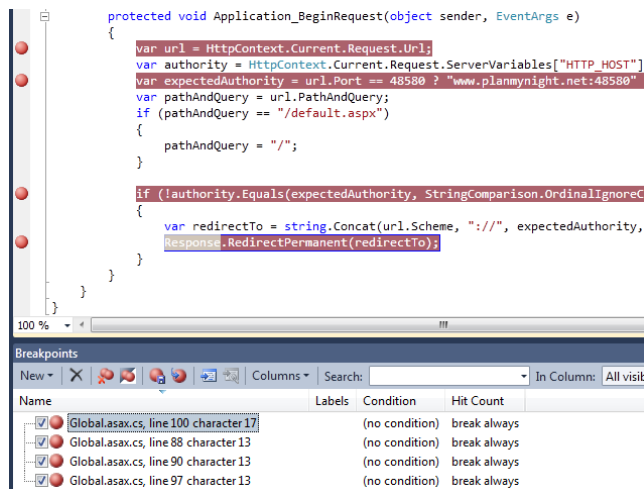
Figure 7-4 Code editor and Breakpoints window with three new breakpoints.

Visual Studio 2005   As a reader and a professional developer using Visual Studio 2005 every day I am sure you noticed a series of new buttons as well as new fields in the Breakpoints window. As a reminder pay attention to Figure 7-5 for a quick comparison of what it looks like in Visual Studio 2005.
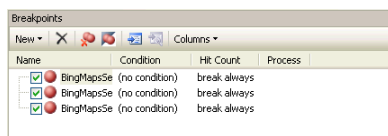
Figure 7-5 Visual Studio 2005 Breakpoints window.

6. The first thing I would like you to pay attention to is that the Labels column is now available for you to help in indexing and in searching breakpoints. It is a really nice and useful feature that Visual Studio 2010 is bringing to the table. To utilize this feature you simply have to right-click on a breakpoint and select Edit Labels or the keyboard shortcut Alt+F9, L. Take a look at Figure 7-6 as a reference.
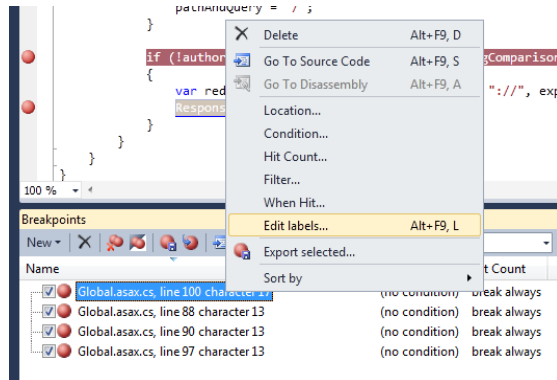

Figure 7-6 Edit Labels

7. You should see a window named **Edit Breakpoints labels** in that window you will add labels for the selected breakpoint. Add the word **ContextRequestUrl** in the **Type a new label:** text box and click on **Add**. Repeat this operation on the first breakpoint and add a label **Url**. When you are done click on **OK** You should see a window that looks like Figure 7-7 while you are entering them and to the right the **Breakpoints** window after you are done with those 2 operations.
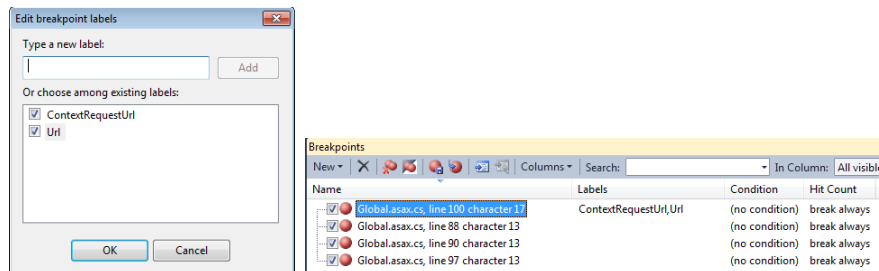
Figure 7-7 Adding Labels and seeing them in the Breakpoints Window

Note   You can also right-click on the breakpoint in the left hand side margin and select Edit Labels to accomplish the same tasks as above.

Note   You will see that when adding labels to a new breakpoint you can choose any of the existing labels that you have already entered. Look at the left figure in Figure 7-7 in the window labeled **Or choose among existing labels:**

8. Using any of the ways learned just now and for the purpose of this exercise please add labels for each of the breakpoints and make sure your Breakpoints window looks like Figure 7-8 after you're done.
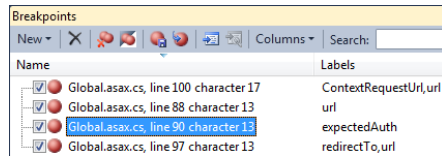

Figure 7-8 Breakpoints window with all labels entered

When you have a lot of code and you are in the midst of a debugging session it would be great to filter the displayed list of breakpoints well that's exactly what the new Search feature in Visual Studio 2010 allows you to do.

9. To see this in action just enter the word **url** in the search text box and you will see the list of breakpoint is filtered down to breakpoints containing **url** in one of their labels.

In a team environment when you have many developers and testers working together it is not rare that two people at any given point in time are working on the same bugs. In Visual Studio 2005 you would have to sit with them, send them a screenshot or line numbers of where to put breakpoints to refine where they should look at while debugging that bug.

> **Important** One of the great new addition to breakpoint management In Visual Studio 2010 is that you can now export breakpoints to a file and then send them to a colleague that can then import them in their environment. Another scenario that this feature is useful is to share breakpoints between machines. Let's see how to do that.

10. In the **Breakpoints** window click on the export button ![export] to export your breakpoints to a file and save it on your desktop. Name the file: **breakexports.xml**

11. Now delete all the breakpoints by clicking either on the **Delete all breakpoints matching the current search criteria** button ![delete matching] **or** by selecting all the breakpoints and clicking the **Delete the selected breakpoints** button ![delete selected] . The only purpose of deleting them is to simulate two developers sharing them or one developer sharing breakpoints between 2 machines.

12. You will now import your breakpoints by clicking on the import button ![import] and loading them from your desktop. Notice that all of your breakpoints with all of their properties are back and loaded in your environment.

Inspecting the Data

When you are debugging your applications you know how much time one can spend stepping into the code and inspecting the content of variables, arguments, etc. Maybe you can remember when you were learning to write code, a while ago, when a debugger wasn't a reality or when it was really rudimentary. Do you remember (maybe not – I am old) how many **printf** or **WriteLn** statements you had to write to inspect the content of different data elements.

> **Visual Studio 2005** In Visual Studio 2005 things were already a big improvement from your days of writing to the console with all kinds of statements because we had a real debugger with new functionalities. New data visualizers allowed you to see XML as a well formed XML snippet and not as a long string. Furthermore with those data visualizers you were able to view arrays in a more visual way with the list of elements and their indices and accomplishing that by simply hovering your mouse over the object. Take a look at Figure 7-9 for an example.
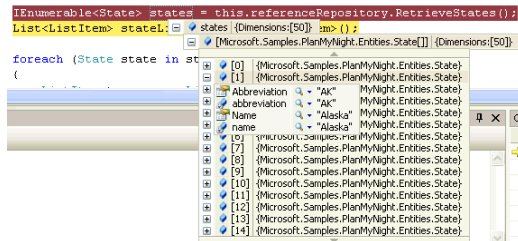
Figure 7-9 Collection view in the debugger in Visual Studio 2005

Now while those DataTip data visualization techniques are still available in Visual Studio 2010, a few great enhancements have been added that makes DataTips even more useful. The DataTip enhancements have been added in conjunction with another new feature of Visual Studio 2010 namely multi-monitor support. Floating DataTips can be very valuable to you as a developer and having the ability to put them on a second monitor is a great new feature that can make your life a lot easier while debugging as it keeps the data that always need to be in context right there on the second monitor.

1. In the global.ascs.cs file you will insert two breakpoints on line 89 and 91, lines starting respectively with the source code `var authority` and `var pathAndQuery.`

2. You are now going to experiment with the new DataTip features. First of all start the debugger by pressing **F5** and then when the debugger hits the first breakpoint you will move you mouse over the word url and click on the push pin as seen in Figure 7-10.
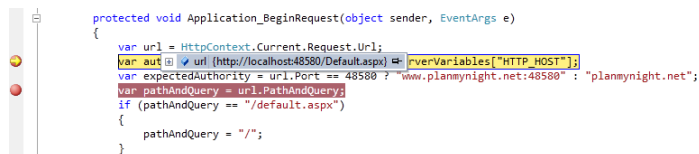
Figure 7-10 The new DataTip pushpin feature

3. You should now see to the right of the line of code the pinned DataTip (as seen in Figure 7-11 on the left) and if you hover your mouse over the DataTip you will get the DataTip management bar (as seen on Figure 7-11 in the right).

Figure 7-11 On the left the pinned DataTip and on the right the DataTip management bar.

> **Note**   You should also see in the breakpoint gutter a blue pushpin indicating that the DataTip is pinned. The push pin should look like this:

> **Note**   If you click the double-arrow pointing down in the DataTip management bar, you can insert a comment for this DataTip, as shown in Figure 7-12. You can also remove the DataTip altogether by clicking on the **X** button in the DataTip management bar.
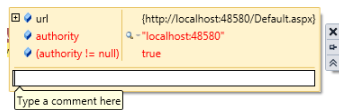
Figure 7-12 Inserting a comment for a DataTip.

5. One nice feature of the new DataTip is that you can insert any expression to be evaluated right there in your debugging session. For instance if you **right-click** on the DataTip name, in this case on **url**, and select **Add Expression** and type in **authority** and then add another one like as this: **(authority != null).** You will see that the expressions are evaluated immediately and will continue to be evaluated for the rest of the debugging session every time your debugger stops on those breakpoints. At this point in the debugging session the expression should respectively evaluate to **null** and **false**.

6. Now press F10 to execute the line where the debugger stopped and look at the url DataTip as well as both expressions. They should contain values based on the current s context. Take a look at Figure 7-13 to see this in action.
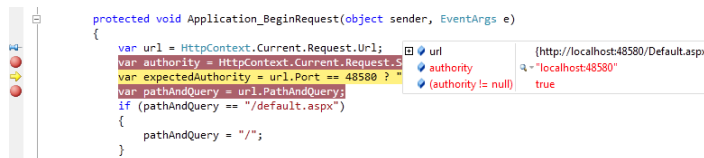

Figure 7-13 The url pinned DataTip with the two evaluated expressions.

7. While it is nice to be able to have a mini watch window where it matters right there where the code is executing you can also see that it is superposed over the source code

being debugged. To that effect you can move the DataTip window anywhere you want in the code editor by simply dragging it. Take a look at Figure 7-14 for an example.
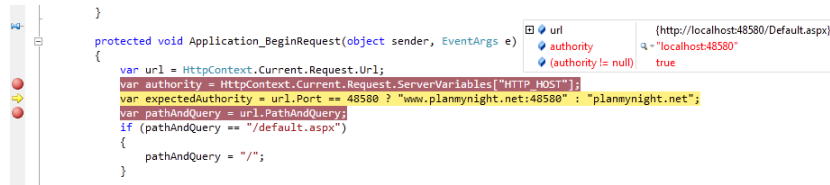

Figure 7-14 Moved the pinned DataTip away from the source code

8. Being pinned the DataTip window will stick where you have pinned which means that it will not be in view anymore if you trace into another file. But in some cases it is important for the DataTip window to be visible at all time. For instance it is interesting for global variables that are always in context or for multi-monitor scenarios. To move a DataTip you have to first unpin it by clicking the pushpin in the DataTip management bar then you will see that it will turn yellow. That is your indication that it is now movable wherever you want for instance: over the Solution Explorer, a second monitor, over your desktop, or any other window. Take a look at Figure 7-15 for an example.

Figure 7-15 Unpinned DataTip over the solution Explorer and the Windows Desktop.

Note   If the DataTip is not pinned and if the debugger stops in another file and method and that the DataTip contains items that are out of context, the DataTip windows will look like Figure 7-16. You can retry to have the debugger evaluate the value of an element by clicking on this button: but it is possible that if that element has no meaning in this context that nothing happens.
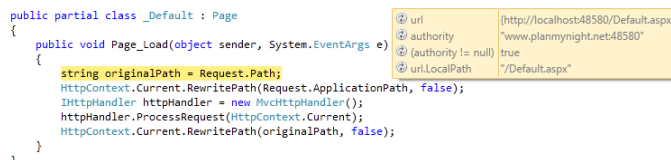

Figure 7-16

Note   You will get an error message if you try to pin outside the editor as seen in Figure 7-17.
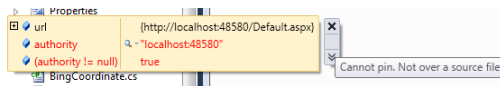
Figure 7-17 Error message trying to pin a DataTip outside the code editor.

> **Note**   You can also pin any child of a pinned item. For instance if you look at url and
> expand its content by pressing the + you will see that you can also pin a child element as
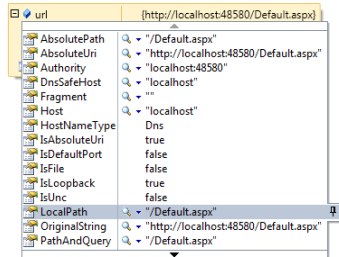> seen in Figure 7-18.

Figure 7-18 Pinned child element within the url DataTip.

9. Before stopping the debugger go back to the global.ascx.cs if you are not already in
   there and re-pin the DataTip window. Now stop the debugging session by pressing the
   Stop Debugging button in the debug toolbar ( ) or by pressing **Shift+F5**. Now if
   you hover your mouse over the blue pushpin in the breakpoint gutter you will see the
   values from the last debug session which is a nice enhancement over the watch window.
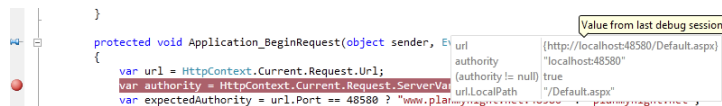   Take a look at Figure 7-19 for what you should see.

Figure 7-19 Values from the last debug session for a pinned DataTip.

> **Note**   Similarly as with the breakpoints you can export/import the DataTips by going to the
> Debug Menu and selecting Export DataTips and Import DataTips.

Using the Minidump Debugger

Many times in real world situation you will have access to a minidump from your product
support team and apart from their bug descriptions and repro steps it might be the only
thing you have to help debug a customer bug. Visual Studio 2010 enables a few
enhancements to the minidump debugging experience.

> **Visual Studio 2005**   In Visual Studio 2005 you could debug managed application or minidump
> files but you had to use an extension if your code was written in managed code. You had to use a
> tool called SOS and load it in the debugger using the Immediate Window. You had to attach the

debugger both in native and managed mode and you couldn't expect to have information in the Call Stack or Locals Window. You had to use commands for SOS in the immediate to help you go through minidump files. With application written in native code you were using normal debugging windows and tools. To read more about this or just to refresh on the topic you can read the *Bug Slayer* column in MSDN magazine here: http://msdn.microsoft.com/en-us/magazine/cc164138.aspx.

Let's see the new enhancements to the minidump debugger but first we'll need to create a crash from which we will be able to generate a minidump file.

1. In the Solution Explorer in the PlanMyNight.Web project rename the file **Default.aspx** to **DefaultA.aspx**. Note the A appended to the end of the word Default.

2. Make sure you have no breakpoints left in your project and to do that look in the breakpoints window and delete any breakpoints left there with any of the ways learned earlier in the chapter.

3. Press F5 to start debugging the application and depending on your machine speed you should see pretty soon after the build process is complete an unhandled exception of type HttpException. While the bug is really simple in this case you will go through the steps of creating the minidump file and debugging it. Take a look at Figure 7-20 to see what you should see at this point.
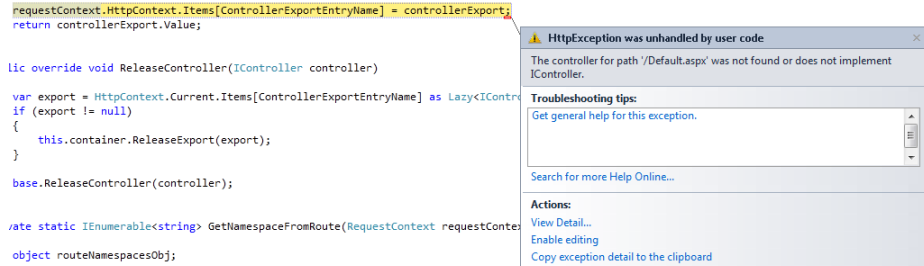

Figure 7-20 Unhandled exception you should expect.

4. It is time to create the minidump file for this exception. Go to the **Debug** menu and select **Save Dump As…** as seen in Figure 7-21. You should see the name of the process from which the exception was thrown. In your case the process from which the exception was thrown was Cassini or the Personal Web Server in Visual Studio. Keep the file name proposed (WebDev.WebServer40.dmp) and save the file on your desktop. Note that it might take some time to create the file as the minidump file size will be close to 300 MB.
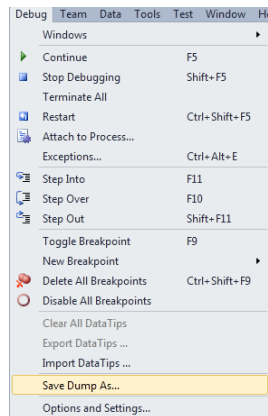
Figure 7-21 Saving the minidump file.

5. Stop Debugging by hitting **Shift+F5** or the **Stop Debugging** button.

6. Next go to the File menu and close your solution.

7. In the File menu now click Open and point it to the desktop to load your minidump file named WebDev.WebServer40.dmp. Doing so will open the minidump Summary Page. It will give you some summary information about the bug you are trying to fix. You can take a look at Figure 7-22 for what you should see. Out of that page and before you really start to debug you will get basic information like: Process name, process architecture, OS version and CLR version, modules loaded as well as some actions you can take from that point. From this place you can set the paths to the symbol files. Luckily you have in the modules list the version and the path on disk of your module so finding the symbols and source code should be fairly easy. The CLR version is 4.0 therefore it is possible to debug right here in Visual Studio 2010.
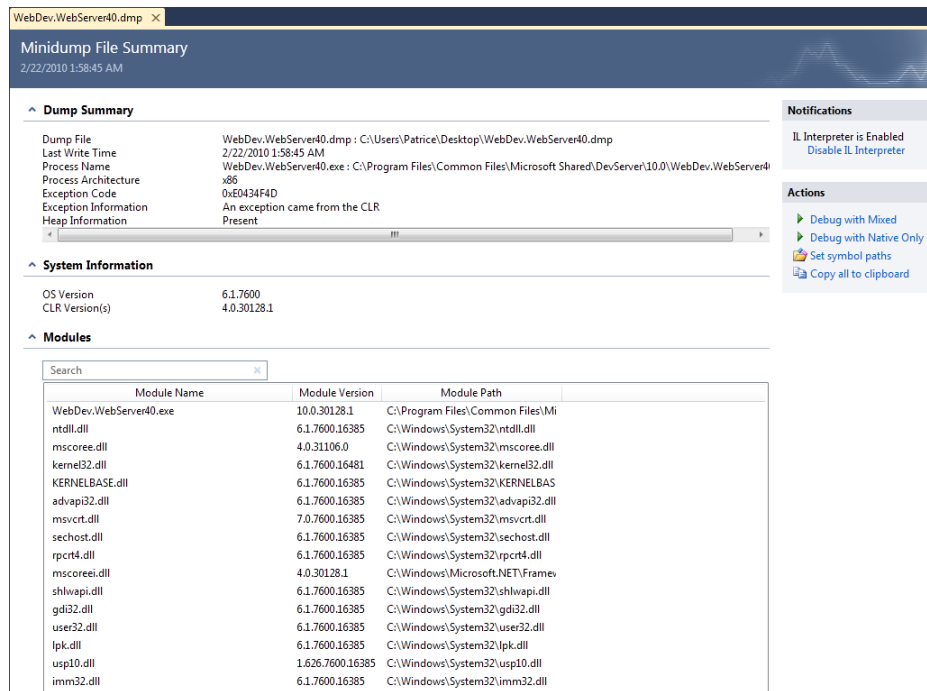
Figure 7-22 Minidump summary page.

8. To start debugging locate the **Actions** list on the right hand side of the summary page and click on **Debug with Mixed**.

9. You should almost see immediately a first chance exception like in Figure 7-23. In this case it tells you what the bug is but it won't be always the case so let's continue and click the **Break** button.
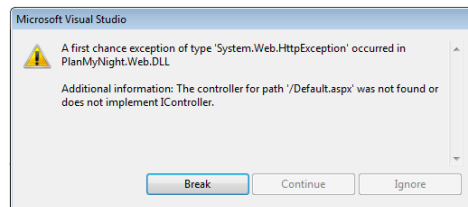

Figure 7-23 First chance exception.

10. You should see a green line indicating the next instruction to execute which means that the exception was thrown just before that. If you look at the source code you will see in your **Autos** window that the **controllerExport** variable is null and that just before that if that variable was null we would throw an **HttpException** if the file to load was not found. In this case the file to look for was Default.aspx as you can see in the **Locals** window in the **controllerName** variable. You can glance at many other variables, objects, etc. in the Locals and Autos windows containing the current context. In this case

we only have one call that is belonging to our code so the call stack indicates that that the code before and after is external to our process. If it would be a deeper chain of calls in our code you could step in the code back and forth before and after and look at the variables. Look at Figure 7-24 for a summary view of all of that.
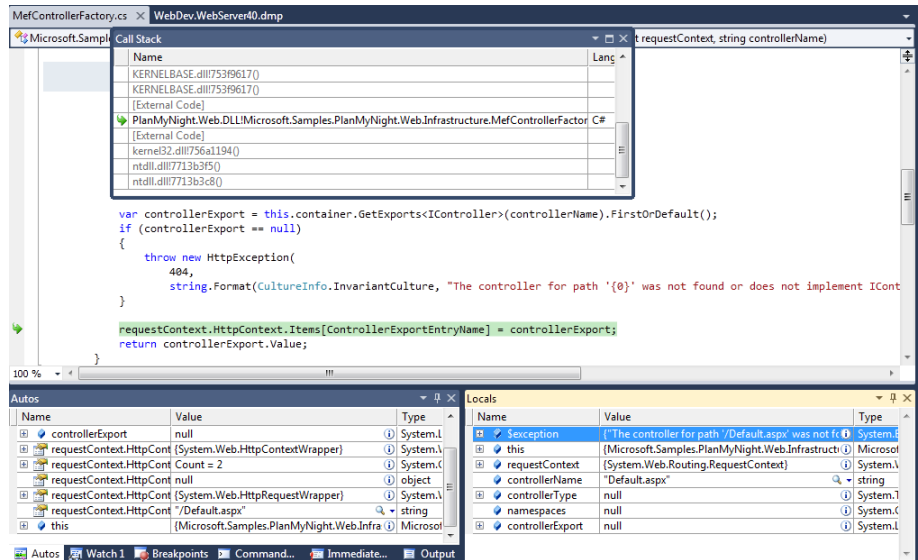

Figure 7-24 Autos, Locals, Call Stack and Next Instruction to execute.

11. Ok good job you have found the bug so stop the debugging by hitting **Shift+F5** or click on the Stop Debugging button and fix it by reloading the PlanMyNight solution and by renaming the file back to default.aspx. Then rebuild the solution by going to the **Build** menu and selecting Rebuild Solution. Now press **F5** and the application should be working again.

Web.Config Transformations

This next new feature while small is one that I believe will delight many developers because it will just save some time while debugging. The feature is the Web.Config transformations that allow you to have transform files with the differences between debug and release environments. As an example, connection strings are often different from an environment to the other and therefore by creating transform files with the different connection strings and because ASP.NET provides tools to change (transform) web.config files you will always end up with the right connection strings for the right environment. To learn more about how to do it take a look at this article on MSDN: http://go.microsoft.com/fwlink/?LinkId=125889.

Creating Unit Tests

Most of the unit test framework and tools are unchanged in the Visual Studio 2010 Professional. It is in other versions of Visual Studio 2010 that the change in test management and test tools is really apparent. Features like UI Unit Tests, IntelliTrace, and Microsoft Test Manager 2010 are available in other product versions like Visual Studio 2010 Premium, Visual Studio 2010 Ultimate. To see which features are covered in the Application Lifecycle Management and more specifically please refer to this article on MSDN: http://msdn.microsoft.com/en-us/library/ee789810(VS.100).aspx

> **Visual Studio 2005**   With Visual Studio 2005 you had to own either Visual Studio 2005 Team System or Visual Studio 2005 Team Test in order to have the ability to create and execute tests out of the box within Visual Studio 2005. Another option back then was to go with a third party option like Nunit.

In this part of the chapter I am going to simply show you how to add a unit test for a class you will find in the Plan My Night application. I will not spend time on defining what is a unit test or what it should contain but rather show you within Visual Studio 2010 how to add tests and execute them.

You will add unit tests to the Plan My Night application for the Print Itinerary Add-in. To create unit tests you will open the Solution from the companion content in the DebuggerStart folder. If you do not remember how you can look at the first page of this chapter on how to do it. Once you have the solution open just follow the next steps.

1. From the Solution Explorer expand the project **PlanMyNight.Web** and then expand the **Helpers** folder. Then double-click on the file **ViewHelper.cs** to open it in the code editor. Take a look at Figure 7-25 to make sure you are at the right place.
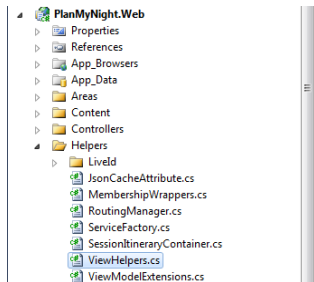

Figure 7-25 PlanMyNight.Web project and ViewHelper.cs file in the Solution Explorer.

2. In the code editor you can add unit tests in two different ways. You can right-click on a class name or on a method name and select **Create Unit Tests...** You can also go to the **Test** menu and select **New Test...** We are going to explore the first way of creating unit tests. This way Visual Studio automatically generates some source code for you. Right-

click on the method name **GetFriendlyTime** and select **Create Unit Tests…** Take a look at Figure 7-26 to see what it looks like.
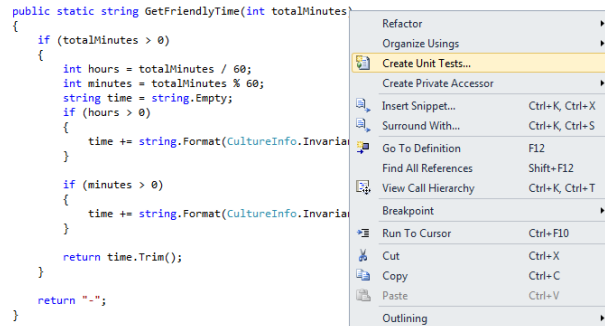

Figure 7-26 Contextual menu to create unit tests from right-clicking on a class name.

3. After selecting Create Unit Tests… you will be presented with a dialog that will by default have selected the method your selected from that class. You will then need to select where you want to create those unit tests and to do so click on the drop-down combo box at the bottom of this dialogue and select **PlanMyNight.Web.Tests**. If you didn't have an existing location you would have simply selected **Create a new Visual C# test project…** from the list. Take a look at Figure 7-27 for what you should be seeing.
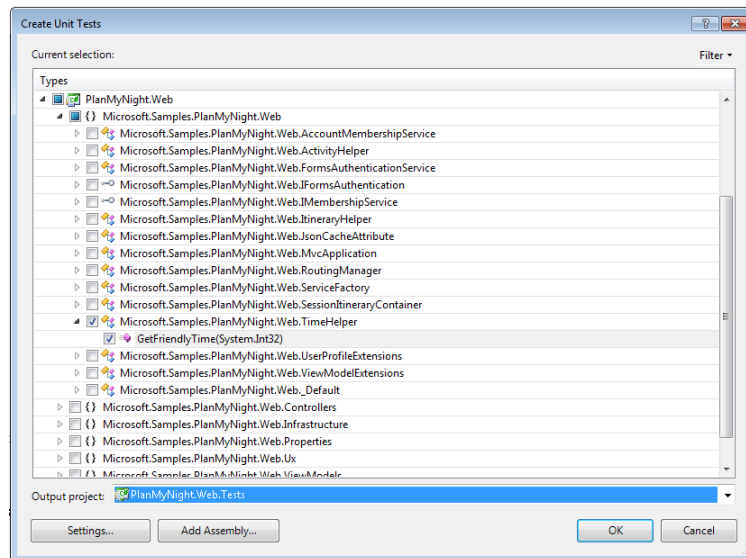

Figure 7-27 Selecting the method you want to create a unit test against.

4. The dialog window will then switch to a test case generation mode and you will see a progress bar. Once this is complete you will have a new file created named **TimeHelperTest.cs** with auto-generated code stubs for you to modify.

5. You will now remove the method and its attributes because we will create three new test cases for that method. Remove the following code.

```
/// <summary>
///A test for GetFriendlyTime
///</summary>
// TODO: Ensure that the UrlToTest attribute specifies a URL to an ASP.NET page (for example,
// http://.../Default.aspx). This is necessary for the unit test to be executed on the web server,
// whether you are testing a page, web service, or a WCF service.
[TestMethod()]
[HostType("ASP.NET")]
[AspNetDevelopmentServerHost("C:\\Users\\Patrice\\Documents\\Chapter 7\\DebuggerStart\\code\\PlanMyNight.Web", "/")]
[UrlToTest("http://localhost:48580/")]
public void GetFriendlyTimeTest()
{
    int totalMinutes = 0; // TODO: Initialize to an appropriate value
    string expected = string.Empty; // TODO: Initialize to an appropriate value
    string actual;
    actual = TimeHelper.GetFriendlyTime(totalMinutes);
    Assert.AreEqual(expected, actual);
    Assert.Inconclusive("Verify the correctness of this test method.");
}
```

6. You will now add the three simple test cases validating three key scenarios used by PlanMyNight. To do that insert the following source code right below the method attributes that were left behind when you deleted the block of code in step 5.

```
[TestMethod]
public void ZeroReturnsSlash()
{
    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));
}
```

7. Now in the PlanMyNight.Web.Tests project create a solution folder called Helpers. Then move your TimeHelperTests.cs file to that folder so that your project looks like Figure 7-28 where you are done.
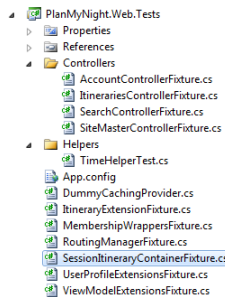

Figure 7-28 TimeHelperTest.cs in its Helpers folder.

8. Now it is time to execute your newly created tests. To only execute your newly created tests you need to go in the code editor and place your cursor on the class name `public class TimeHelperTest.  Then you` can either go to the **Test** menu, select **Run** and finally select **Test in Current Context** or using the keyboard shortcut **CTRL+R, T**. Look at Figure 7-29 for a reference.
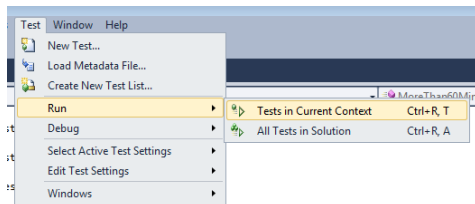

Figure 7-29 Tests Execution Menu

9. Performing this action will only execute your three tests and you should see the following Test Results Window appear at the bottom of your editor with the test results. Look at Figure 7-30 for the Test Results window.

Figure 7-30 Test Results window for your newly created tests.

More Info   Depending on what you select you will have a different behavior when you select to execute **Tests in Current Context**. For instance if you select a test method like **ZeroReturnsSlash** you will only execute this test case but if you click outside the test class you could end up executing every single test cases which will be the equivalent of selecting **All Tests in Solution**.

New Threads Window

The immergence of computers with multiple cores and the fact that the language features are giving developers many tools to take advantage of those cores creates a new problem: the difficulty to debug concurrency in applications. The new threads window enables you the developer to pause threads and searching the calling stack and see similar artifacts that you would see using the famous SysInternals Process Monitor (http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx). The Threads window can be displayed by going to Debug then selecting Windows and Threads while debugging an application. Take a look at Figure 7-31 for a peek at the Threads Window while debugging Plan My Night.
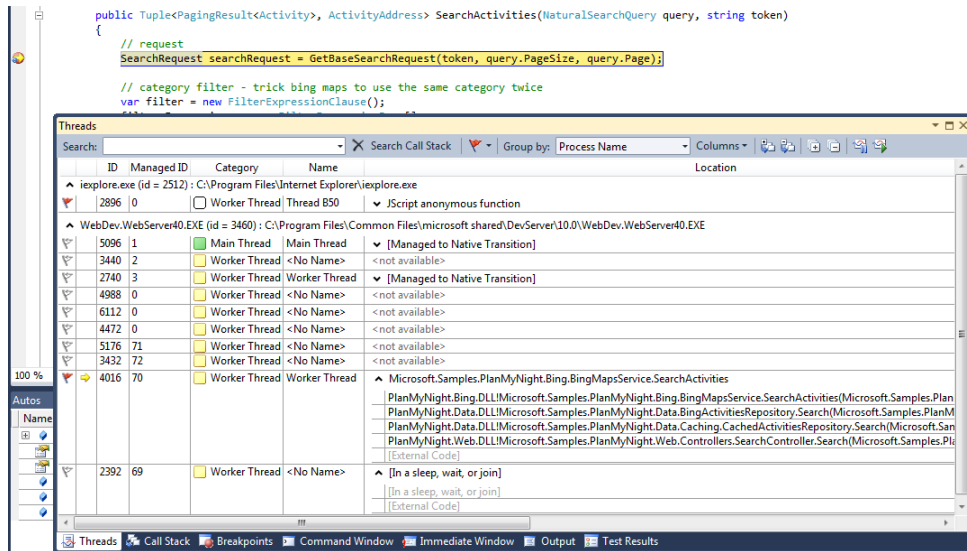

Figure 7-31 Displaying the Threads Window while debugging Plan My Night.

It allows you to freeze threads and thaw them whenever you are ready to let them continue. It can be really useful when you are trying to isolate particular effects... You can debug both manage code and unmanaged code. If you are application uses thread you will definitely love this new feature of the debugger in Visual Studio 2010.

In Visual Studio 2005 you had to uses many tools not integrated in Visual Studio or third-party tools and still in most cases you almost had rely on your instinct and experience to find concurrency bugs.

Summary

In this chapter, you have learned about new ways of managing your debugging session through new breakpoints enhancements, new data inspection and data visualization techniques as well as how the new minidump debugger and tools can help you solve real

customer problems from the field. You have also seen how to raise the quality of your code by writing unit tests and how Visual Studio 2010 Professional can help you doing this. Multi-cores machines are now the norm and so are multi-threaded applications. Therefore new debugger enhancements around finding issues in multi-threaded applications with specific debugger tools is great news.

Finally throughout this chapter you also saw how Visual Studio 2010 Professional has raised the bar in terms of debugging applications and giving you the professional developers the tools to debug today's feature rich experiences. You have seen that it is a clear improvement over what was available in Visual Studio 2005 and how you will be able to save time and money by moving to this new debugging environment and that Visual Studio 2010 is more than a small iteration it is a huge leap in productivity for developers. Different versions of Visual Studio 2010 give you a great list of improvements around the debugger and testing. My personal favorites are IntelliTrace (http://msdn.microsoft.com/en-us/library/dd264915(VS.100).aspx) available only in Visual Studio 2010 Ultimate and Microsoft Test Manager which will enable test teams to have a much better stories using Visual Studio 2010 and Visual Studio 2010 Team Foundation Server. (http://msdn.microsoft.com/en-us/library/bb385901(VS.100).aspx).