Deploying an ASP.NET Web Application to a Hosting Provider using Visual Studio or Visual Web

Contents

Developer: Introduction - 1 of 12

DOWNLOAD ASSETS: Starter Project

This series of tutorials shows you how to deploy (publish) an ASP.NET web application project by using the one-click publish feature in Visual Studio 2012 RC or Visual Studio Express 2012 RC for Web. You can also use Visual Studio 2010 or Visual Web Developer Express 2010 if you install the Web Publish update that was released with Visual Studio 2012 RC.

E-Book Gallery

The original version of this content (for Visual Studio 2010 or Visual Web Developer 2010 without the Web Publish Update and without using Code First Migrations) is available in the following e-book formats:
EPUB | MOBI | PDF

These tutorials assume you know how to work with ASP.NET in Visual Studio. If you don't, a good place to start is a basic ASP.NET Web Forms Tutorial or a basic ASP.NET MVC Tutorial.

Before you start, make sure you have the following software installed on your computer:

- Windows 7
- Visual Studio 2010 SP1 or Visual Web Developer Express 2010 SP1 (If you use one of these links, the following items will be installed automatically.)
- Microsoft SQL Server Compact 4.0
- Microsoft Visual Studio 2010 SP1 Tools for SQL Server Compact 4.0

If you have questions that are not directly related to the tutorial, you can post them to the ASP.NET Deployment forum.

Overview

These tutorials guide you through deploying first to IIS on your local development computer for testing, and then to a third-party hosting provider. The application that you'll deploy uses an application database and an

ASP.NET membership database. You start off using SQL Server Compact and deploying to SQL Server Compact, and later tutorials show you how to deploy database changes and how to migrate to SQL Server.

The number of tutorials – 11 in all plus a troubleshooting page – might make the deployment process seem daunting. In fact, the basic procedures for deploying a site make up a relatively small part of the tutorial set. However, in real-world situations, you often need information about some small but important extra aspect of deployment — for example, setting folder permissions on the target server. We've included many of these additional techniques in the tutorials, with the hope that the tutorials don't leave out information that might prevent you from successfully deploying a real application.

The tutorials are designed to run in sequence, and each part builds on the previous part. However, you can skip parts that aren't relevant to your situation. (Skipping parts might require you to adjust the procedures in later tutorials.)

Intended Audience

The tutorials are aimed at ASP.NET developers who work in small organizations or other environments where:

- A continuous integration process (automated builds and deployment) is not used.
- The production environment is a third-party hosting provider.
- One person typically fills multiple roles (the same person develops, tests, and deploys).

In enterprise environments, it's more typical to implement continuous integration processes, and the production environment is usually hosted by the company's own servers. Different people also typically perform different roles. For information about enterprise deployment, see Deploying Web Applications in Enterprise Scenarios.

Organizations of all sizes can also deploy web applications to Windows Azure, and most of the procedures shown in these tutorials apply also to Windows Azure Web Sites. For an introduction to Windows Azure, see http://windowsazure.com.

The Hosting Provider Shown in the Tutorials

The tutorials take you through the process of setting up an account with a hosting company and deploying the application to that hosting provider. A specific hosting company was chosen so that the tutorials could illustrate the complete experience of deploying to a live website. Each hosting company provides different features, and the experience of deploying to their servers varies somewhat; however, the process described in this tutorial is typical for the overall process.

The hosting provider used for this tutorial, Cytanium.com, is one of many that are available, and its use in this tutorial does not constitute an endorsement or recommendation.

Deploying Web Site Projects

Contoso University is a Visual Studio web application project. Most of the deployment methods and tools demonstrated in this tutorial do not apply to Web Site Projects. For information about how to deploy web site projects, see ASP.NET Deployment Content Map.

Deploying ASP.NET MVC Projects

For this tutorial you deploy an ASP.NET Web Forms project, but everything you learn how to do is applicable to ASP.NET MVC as well. A Visual Studio MVC project is just another form of web application project. The only difference is that if you're deploying to a hosting provider that does not support ASP.NET MVC or your target version of it, you must make sure that you have installed the appropriate (MVC 3 or MVC 4) NuGet package in your project.

Programming Language

The sample application uses C# but the tutorials do not require knowledge of C#, and the deployment techniques shown by the tutorials are not language-specific.

Troubleshooting During this Tutorial

When an error happens during deployment, or if the deployed site does not run correctly, the error messages don't always provide a solution. To help you with some common problem scenarios, a troubleshooting reference page is available. If you get an error message or something doesn't work as you go through the tutorials, be sure to check the troubleshooting page.

Comments Welcome

Comments on the tutorials are welcome, and when the tutorial is updated every effort will be made to take into account corrections or suggestions for improvements that are provided in tutorial comments.

Prerequisites

Before you start, make sure that you have Windows 7 or later and one of the following products installed on your computer:

- Visual Studio 2010 SP1
- Visual Web Developer Express 2010 SP1
- Visual Studio 2012 RC or Visual Studio Express 2012 RC for Web

If you have Visual Studio 2010 SP1 or Visual Web Developer Express 2010 SP1, install the following products also:

- Windows Azure SDK for .NET (VS 2010 SP1) (includes the Web Publish Update)
- Microsoft Visual Studio 2010 SP1 Tools for SQL Server Compact 4.0

Some other software is required in order to complete the tutorial, but you don't have to have that loaded yet. The tutorial will walk you through the steps for installing it when you need it.

Downloading the Sample Application

The application you'll deploy is named Contoso University and has already been created for you. It's a simplified version of a university web site, based loosely on the Contoso University application described in the Entity Framework tutorials on the ASP.NET site.

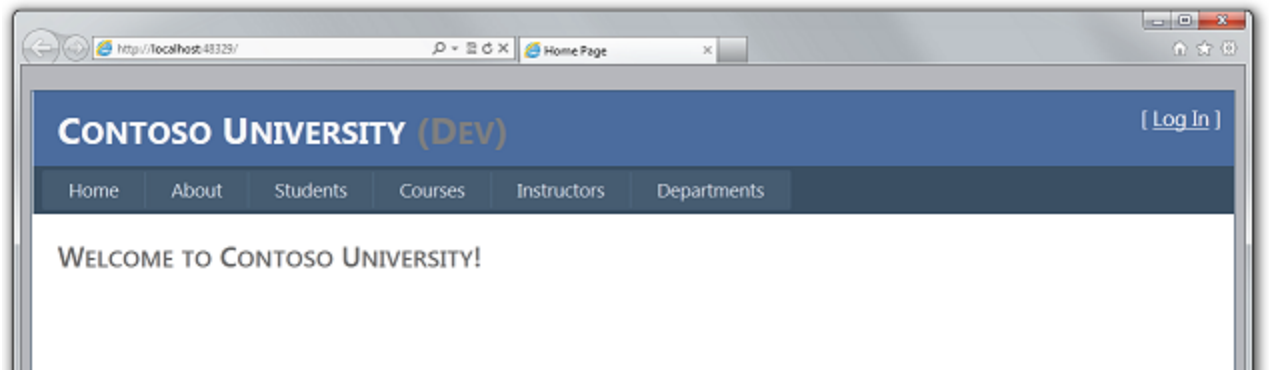
When you have the prerequisites installed, download the Contoso University web application. The *.zip* file contains multiple versions of the project and a PDF file that contains all 12 tutorials. To work through the steps of the tutorial, start with ContosoUniversity-Begin. To see what the project looks like at the end of the tutorials, open ContosoUniversity-End. To see what the project looks like before the migration to full SQL Server in tutorial 10, open ContosoUniversity-AfterTutorial09.

To prepare to work through the tutorial steps, save ContosoUniversity-Begin to whatever folder you use for working with Visual Studio projects. By default this is the following folder:

```
C:\Users\<username>\Documents\Visual Studio 2012\Projects
```

(For the screen shots in this tutorial, the project folder is located in the root directory on the C: drive.)

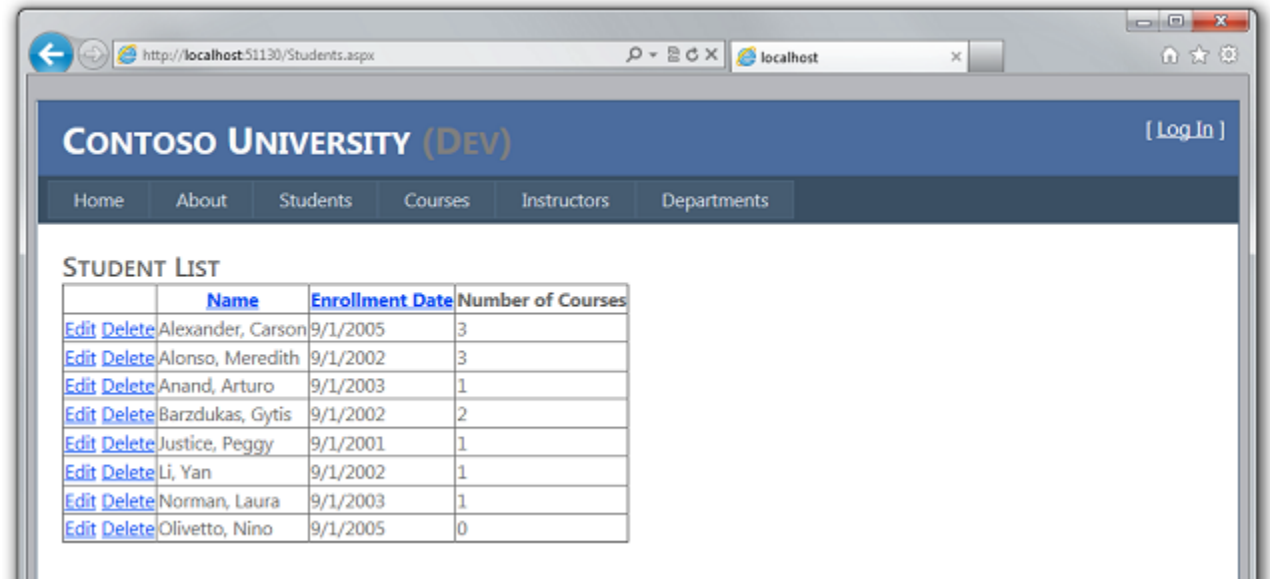
Start Visual Studio, open the project, and press CTRL-F5 to run it.

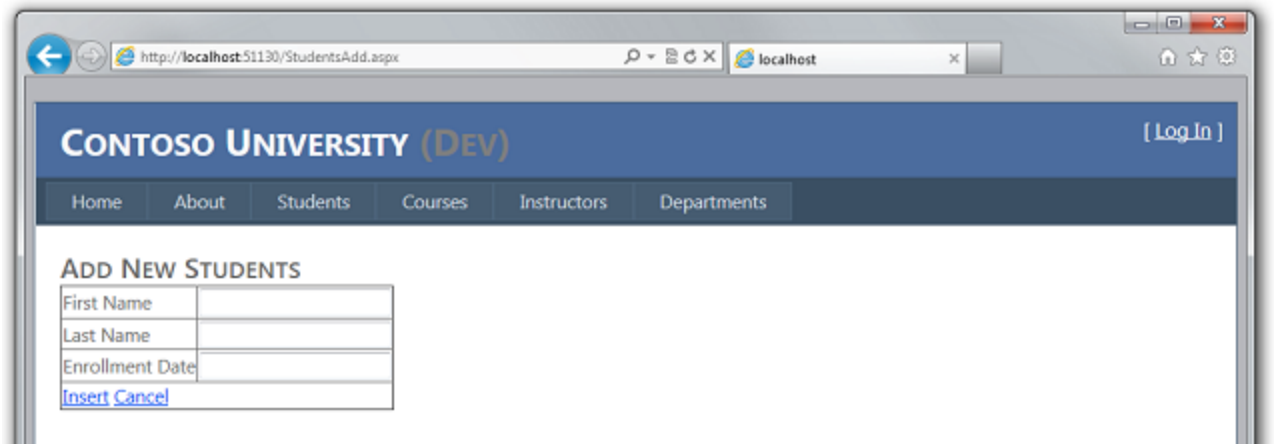


The website pages are accessible from the menu bar and let you perform the following functions:

- Display student statistics (the About page).
- Display, edit, delete, and add students.
- Display and edit courses.
- Display and edit instructors.
- Display and edit departments.

Following are screen shots of a few representative pages.





Reviewing Application Features that Affect Deployment

The following features of the application affect how you deploy it or what you have to do to deploy it. Each of these is explained in more detail in the following tutorials in the series.

- Contoso University uses a SQL Server Compact database to store application data such as student and instructor names. The database contains a mix of test data and production data, and when you deploy to production you need to exclude the test data. Later in the tutorial series you'll migrate from SQL Server Compact to SQL Server.
- The application uses the ASP.NET membership system, which stores user account information in a SQL Server Compact database. The application defines an administrator user who has access to some restricted information. You need to deploy the membership database without test accounts but with one administrator account.
- Because the application database and the membership database use SQL Server Compact as the database engine, you have to deploy the database engine to the hosting provider, as well as the databases themselves.
- The application uses ASP.NET universal membership providers so that the membership system can store its data in a SQL Server Compact database. The assembly that contains the universal membership providers must be deployed with the application.
- The application uses the Entity Framework 5.0 to access data in the application database. The assembly that contains Entity Framework 5.0 must be deployed with the application.
- The application uses a third-party error logging and reporting utility. This utility is provided in an assembly which must be deployed with the application.
- The error logging utility writes error information in XML files to a file folder. You have to make sure that the account that ASP.NET runs under in the deployed site has write permission to this folder, and you have

to exclude this folder from deployment. (Otherwise, error log data from the test environment might be deployed to production and/or production error log files might be deleted.)

- The application includes some settings that must be changed in in the deployed *Web.config* file depending on the destination environment (test or production), and other settings that must be changed depending on the build configuration (Debug or Release).
- The Visual Studio solution includes a class library project. Only the assembly that this project generates should be deployed, not the project itself.

In this first tutorial in the series, you have downloaded the sample Visual Studio project and reviewed site features that affect how you deploy the application. In the following tutorials, you prepare for deployment by setting up some of these things to be handled automatically. Others you take care of manually.

Deploying SQL Server Compact Databases - 2 of 12

Overview

This tutorial shows how to set up two SQL Server Compact databases and the database engine for deployment.

For database access, the Contoso University application requires the following software that must be deployed with the application because it is not included in the .NET Framework:

- SQL Server Compact (the database engine).
- ASP.NET Universal Providers (which enable the ASP.NET membership system to use SQL Server Compact)
- Entity Framework 5.0 (Code First with Migrations).

The database structure and some (not all) of the data in the application's two databases must also be deployed. Typically, as you develop an application, you enter test data into a database that you don't want to deploy to a live site. However, you might also enter some production data that you do want to deploy. In this tutorial you'll configure the Contoso University project so that the required software and the correct data are included when you deploy.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

SQL Server Compact versus SQL Server Express

The sample application uses SQL Server Compact 4.0. This database engine is a relatively new option for websites; earlier versions of SQL Server Compact do not work in a web hosting environment. SQL Server Compact offers a few benefits compared to the more common scenario of developing with SQL Server Express and deploying to full SQL Server. Depending on the hosting provider you choose, SQL Server Compact might be cheaper to deploy, because some providers charge extra to support a full SQL Server database. There is no extra charge for SQL Server Compact because you can deploy the database engine itself as part of your web application.

However, you should also be aware of its limitations. SQL Server Compact does not support stored procedures, triggers, views, or replication. (For a complete list of SQL Server features that are not supported by SQL Server Compact, see Differences Between SQL Server Compact and SQL Server.) Also, some of the tools that you can use to manipulate schemas and data in SQL Server Express and SQL Server databases do not work with SQL Server Compact. For example, you cannot use SQL Server Management Studio or SQL Server Data Tools in Visual

Studio with SQL Server Compact databases. You do have other options for working with SQL Server Compact databases:

- You can use Server Explorer in Visual Studio, which offers limited database manipulation functionality for SQL Server Compact.
- You can use the database manipulation feature of WebMatrix, which has more features than Server Explorer.
- You can use relatively full-featured third-party or open source tools, such as the SQL Server Compact Toolbox and SQL Compact data and schema script utility.
- You can write and run your own DDL (data definition language) scripts to manipulate the database schema.

You can start with SQL Server Compact and then upgrade later as your needs evolve. Later tutorials in this series show you how to migrate from SQL Server Compact to SQL Server Express and to SQL Server. However, if you're creating a new application and expect to need SQL Server in the near future, it's probably best to start with SQL Server or SQL Server Express.

Configuring the SQL Server Compact Database Engine for Deployment

The software required for data access in the Contoso University application was added by installing the following NuGet packages:

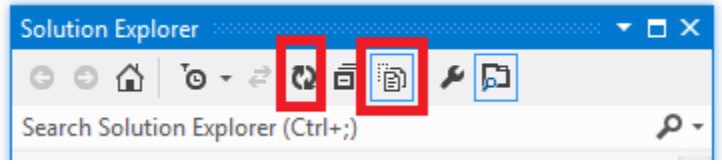
- SqlServerCompact
- System.Web.Providers (ASP.NET universal providers)
- EntityFramework
- EntityFramework.SqlServerCompact

The links point to the current versions of these packages, which might be newer than what is installed in the starter project that you downloaded for this tutorial. For deployment to a hosting provider, make sure that you use Entity Framework 5.0 or later. Earlier versions of Code First Migrations require Full Trust, and at many hosting providers your application will run in Medium Trust. For more information about Medium Trust, see the Deploying to IIS as a Test Environment tutorial.

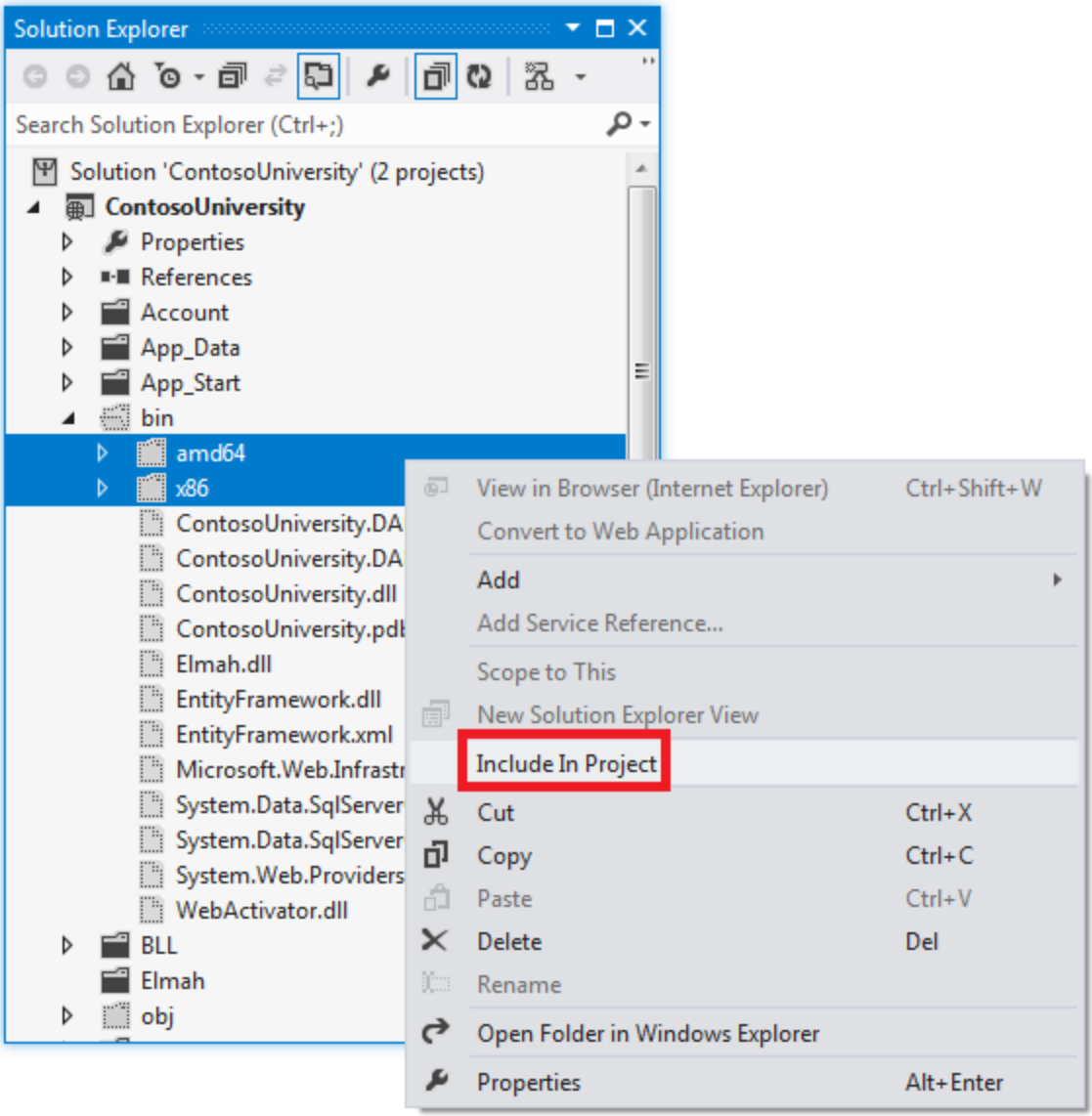
NuGet package installation generally takes care of everything that you need in order to deploy this software with the application. In some cases, this involves tasks such as changing the Web.config file and adding PowerShell scripts that run whenever you build the solution. **If you want to add support for any of these features (such as SQL Server Compact and Entity Framework) without using NuGet, make sure that you know what NuGet package installation does so that you can do the same work manually.**

There is one exception where NuGet doesn't take care of everything you have to do in order to ensure successful deployment. The SqlServerCompact NuGet package adds a post-build script to your project that copies the native assemblies to *x86* and *amd64* subfolders under the project *bin* folder, but the script does not include those folders in the project. As a result, Web Deploy will not copy them to the destination web site unless you manually include them in the project. (This behavior results from the default deployment configuration; another option, which you won't use in these tutorials, is to change the setting that controls this behavior. The setting that you can change is **Only files needed to run the application** under **Items to deploy** on the **Package/Publish Web** tab of the **Project Properties** window. Changing this setting is not generally recommended because it can result in the deployment of many more files to the production environment than are needed there. For more information about the alternatives, see the Configuring Project Properties tutorial.)

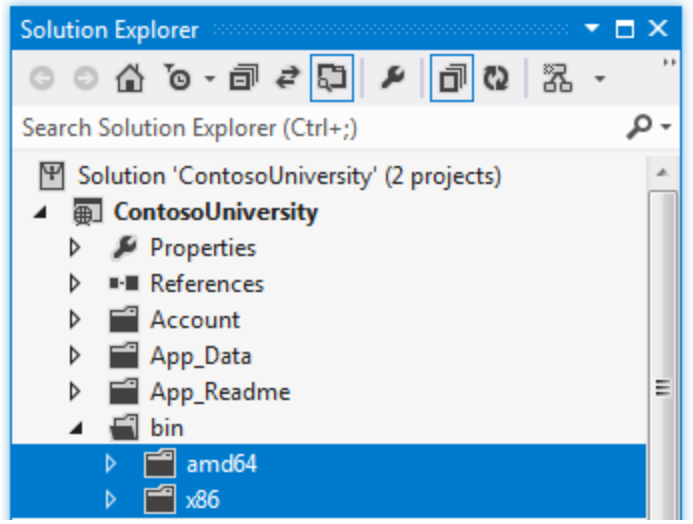
Build the project, and then in **Solution Explorer** click **Show all Files** if you have not already done so. You might also have to click **Refresh**.



Expand the **bin** folder to see the **amd64** and **x86** folders, and then select those folders, right-click, and select **Include in Project**.



The folder icons change to show that the folder has been included in the project.



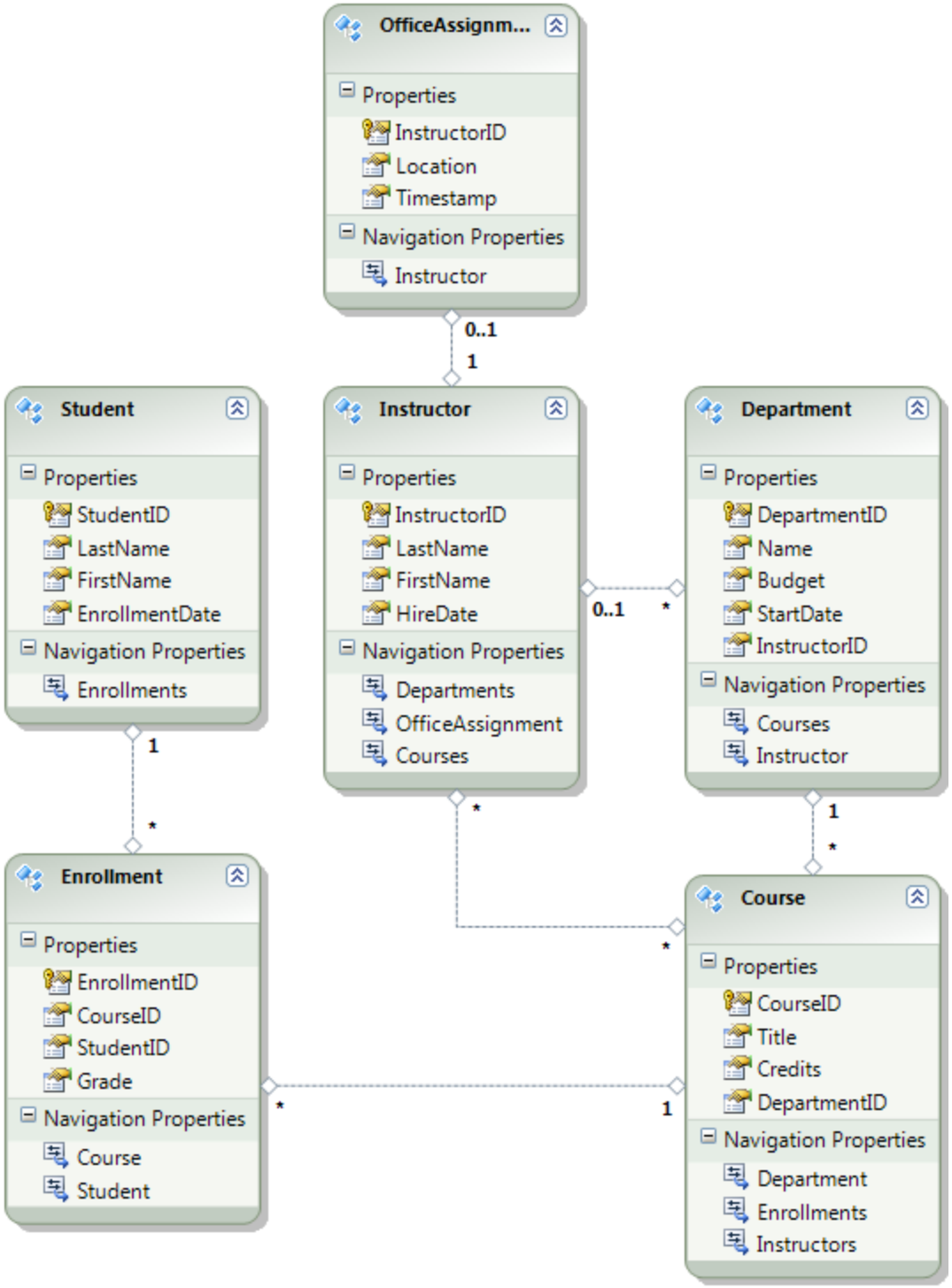
Configuring Code First Migrations for Application Database Deployment

When you deploy an application database, typically you don't simply deploy your development database with all of the data in it to production, because much of the data in it is probably there only for testing purposes. For example, the student names in a test database are fictional. On the other hand, you often can't deploy just the database structure with no data in it at all. Some of the data in your test database might be real data and must be there when users begin to use the application. For example, your database might have a table that contains valid grade values or real department names.

To simulate this common scenario, you'll configure a Code First Migrations Seed method that inserts into the database only the data that you want to be there in production. This Seed method won't insert test data because it will run in production after Code First creates the database in production.

In earlier versions of Code First before Migrations was released, it was common for Seed methods to insert test data also, because with every model change during development the database had to be completely deleted and re-created from scratch. With Code First Migrations, test data is retained after database changes, so including test data in the Seed method is not necessary. The project you downloaded uses the pre-Migrations method of including all data in the Seed method of an initializer class. In this tutorial you'll disable the initializer class and enable Migrations. Then you'll update the Seed method in the Migrations configuration class so that it inserts only data that you want to be inserted in production.

The following diagram illustrates the schema of the application database:



For these tutorials, you'll assume that the `Student` and `Enrollment` tables should be empty when the site is first deployed. The other tables contain data that has to be preloaded when the application goes live.

Since you will be using Code First Migrations, you no longer have to use the **DropCreateDatabaseIfModelChanges** Code First initializer. The code for this initializer is in the

SchoolInitializer.cs file in the ContosoUniversity.DAL project. A setting in the **appSettings** element of the Web.config file causes this initializer to run whenever the application tries to access the database for the first time:

```
 <appSettings>
  <add key="Environment" value="Dev" />
  <add key="DatabaseInitializerForType ContosoUniversity.DAL.SchoolContext,
ContosoUniversity.DAL" value="ContosoUniversity.DAL.SchoolInitializer,
ContosoUniversity.DAL" />
</appSettings>
```

Open the application Web.config file and remove the element that specifies the Code First initializer class from the appSettings element. The appSettings element now looks like this:

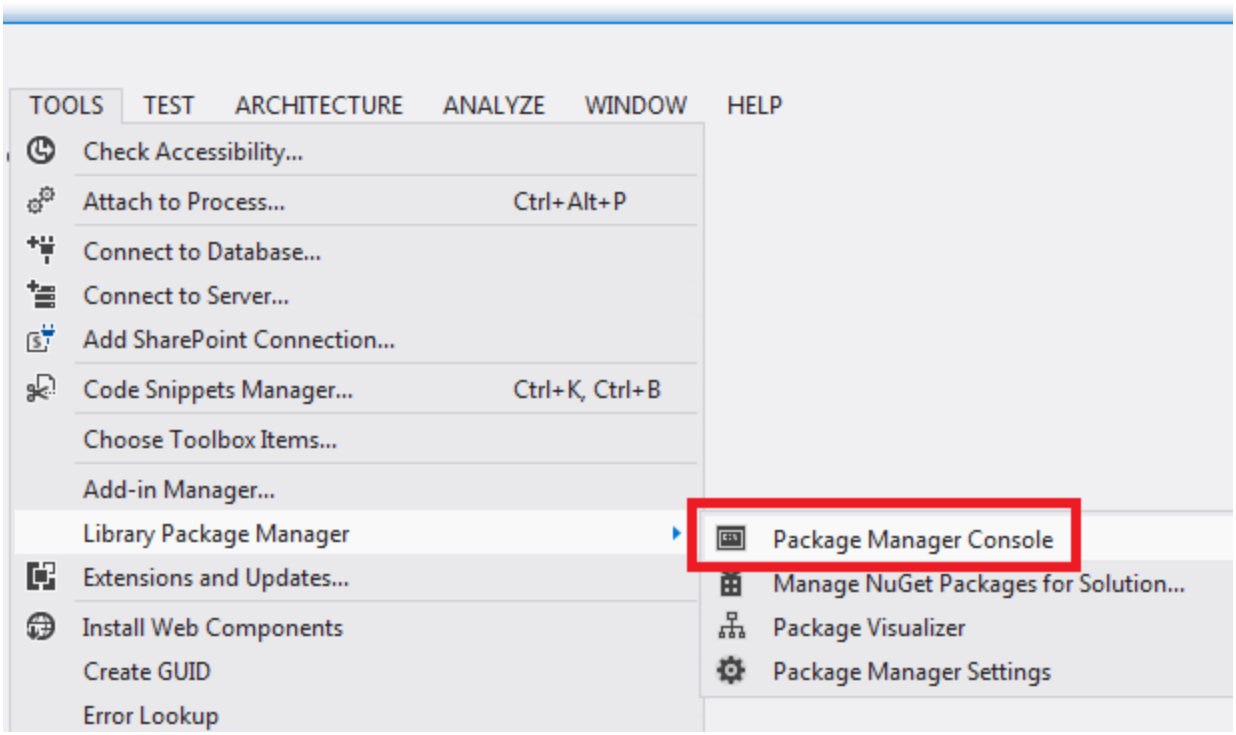
```
 <appSettings>
  <add key="Environment" value="Dev" />
</appSettings>
```

Note Another way to specify an initializer class is do it by calling `Database.SetInitializer` in the `Application_Start` method in the *Global.asax* file. If you are enabling Migrations in a project that uses that method to specify the initializer, remove that line of code.

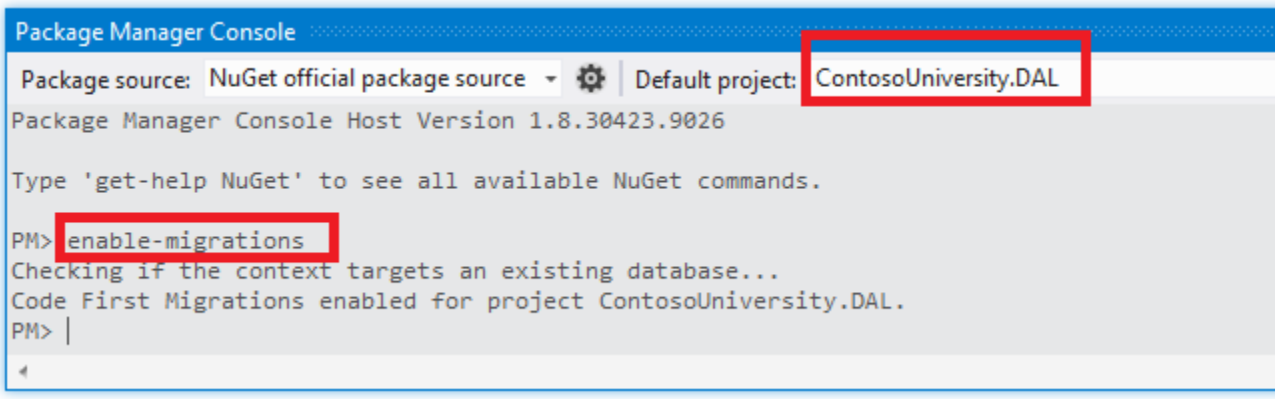
Next, enable Code First Migrations.

The first step is to make sure that the ContosoUniversity project is set as the startup project. In **Solution Explorer**, right-click the ContosoUniversity project and select **Set as Startup Project**. Code First Migrations will look in the startup project to find the database connection string.

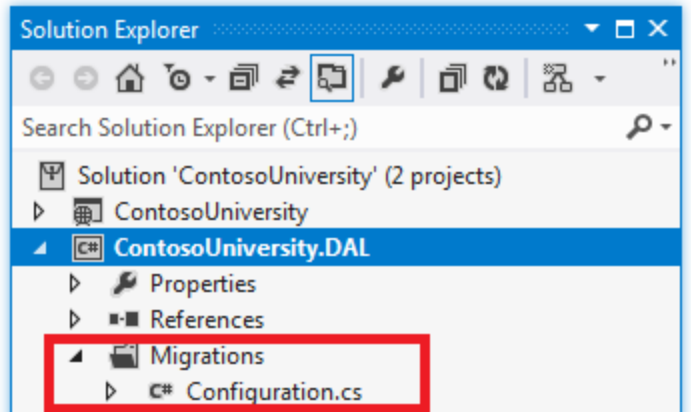
From the **Tools** menu, click **Library Package Manager** and then **Package Manager Console**.



At the top of the **Package Manager Console** window select ContosoUniversity.DAL as the default project and then at the `PM>` prompt enter "enable-migrations".



This command creates a *Configuration.cs* file in a new *Migrations* folder in the ContosoUniversity.DAL project.



You selected the DAL project because the "enable-migrations" command must be executed in the project that contains the Code First context class. When that class is in a class library project, Code First Migrations looks for the database connection string in the startup project for the solution. In the ContosoUniversity solution, the web project has been set as the startup project. (If you did not want to designate the project that has the connection string as the startup project in Visual Studio, you can specify the startup project in the PowerShell command. To see the command syntax for the enable-migrations command, you can enter the command "get-help enable-migrations".)

Open the Configuration.cs file and replace the comments in the `Seed` method with the following code:

```
var instructors = new List<Instructor>
{
    new Instructor { FirstMidName = "Kim",     LastName = "Abercrombie", HireDate =
DateTime.Parse("1995-03-11"), OfficeAssignment = new OfficeAssignment { Location =
"Smith 17" } },
    new Instructor { FirstMidName = "Fadi",    LastName = "Fakhouri",    HireDate =
DateTime.Parse("2002-07-06"), OfficeAssignment = new OfficeAssignment { Location =
"Gowan 27" } },
    new Instructor { FirstMidName = "Roger",   LastName = "Harui",       HireDate =
DateTime.Parse("1998-07-01"), OfficeAssignment = new OfficeAssignment { Location =
"Thompson 304" } },
    new Instructor { FirstMidName = "Candace", LastName = "Kapoor",      HireDate =
DateTime.Parse("2001-01-15") },
    new Instructor { FirstMidName = "Roger",   LastName = "Zheng",       HireDate =
DateTime.Parse("2004-02-12") }
};
instructors.ForEach(s => context.Instructors.AddOrUpdate(i => i.LastName, s));
context.SaveChanges();
```

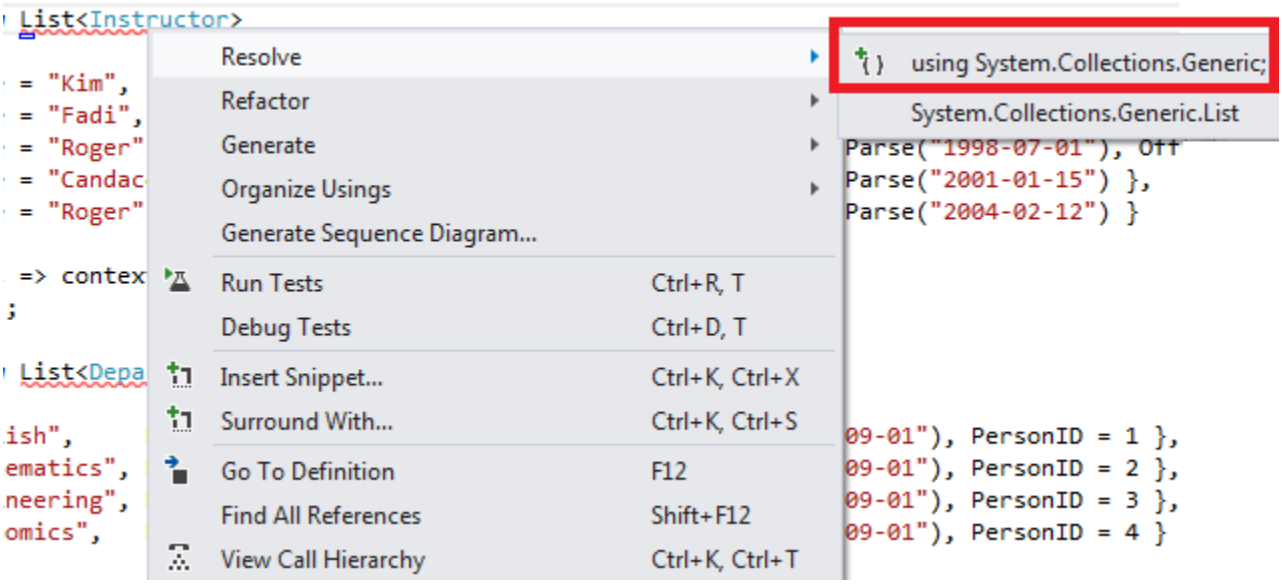
```
var departments = new List<Department>
{
    new Department { Name = "English",     Budget = 350000, StartDate =
DateTime.Parse("2007-09-01"), PersonID = 1 },
    new Department { Name = "Mathematics", Budget = 100000, StartDate =
DateTime.Parse("2007-09-01"), PersonID = 2 },
    new Department { Name = "Engineering", Budget = 350000, StartDate =
DateTime.Parse("2007-09-01"), PersonID = 3 },
    new Department { Name = "Economics",   Budget = 100000, StartDate =
DateTime.Parse("2007-09-01"), PersonID = 4 }
};
departments.ForEach(s => context.Departments.AddOrUpdate(d => d.Name, s));
context.SaveChanges();

var courses = new List<Course>
{
    new Course { CourseID = 1050, Title = "Chemistry",      Credits = 3, DepartmentID
= 3, Instructors = new List<Instructor>() },
    new Course { CourseID = 4022, Title = "Microeconomics", Credits = 3, DepartmentID
= 4, Instructors = new List<Instructor>() },
    new Course { CourseID = 4041, Title = "Macroeconomics", Credits = 3, DepartmentID
= 4, Instructors = new List<Instructor>() },
    new Course { CourseID = 1045, Title = "Calculus",       Credits = 4, DepartmentID
= 2, Instructors = new List<Instructor>() },
    new Course { CourseID = 3141, Title = "Trigonometry",   Credits = 4, DepartmentID
= 2, Instructors = new List<Instructor>() },
    new Course { CourseID = 2021, Title = "Composition",    Credits = 3, DepartmentID
= 1, Instructors = new List<Instructor>() },
    new Course { CourseID = 2042, Title = "Literature",     Credits = 4, DepartmentID
= 1, Instructors = new List<Instructor>() }
};
courses.ForEach(s => context.Courses.AddOrUpdate(s));
context.SaveChanges();

courses[0].Instructors.Add(instructors[0]);
courses[0].Instructors.Add(instructors[1]);
courses[1].Instructors.Add(instructors[2]);
```

```
courses[2].Instructors.Add(instructors[2]);
courses[3].Instructors.Add(instructors[3]);
courses[4].Instructors.Add(instructors[3]);
courses[5].Instructors.Add(instructors[3]);
courses[6].Instructors.Add(instructors[3]);
context.SaveChanges();
```

The references to `List` have red squiggly lines under them because you don't have a `using` statement for its namespace yet. Right-click one of the instances of `List` and click **Resolve**, and then click **using System.Collections.Generic**.



This menu selection adds the following code to the `using` statements near the top of the file.

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

Note Adding code to the `Seed` method is one of many ways that you can insert fixed data into the database. An alternative is to add code to the `Up` and `Down` methods of each migration class. The `Up` and `Down` methods contain code that implements database changes. You'll see examples of them in the Deploying a Database Update tutorial.

You can also write code that executes SQL statements by using the `Sql` method. For example, if you were adding a Budget column to the Department table and wanted to initialize all department budgets to $1,000.00 as part of a migration, you could add the folllowing line of code to the `Up` method for that migration:

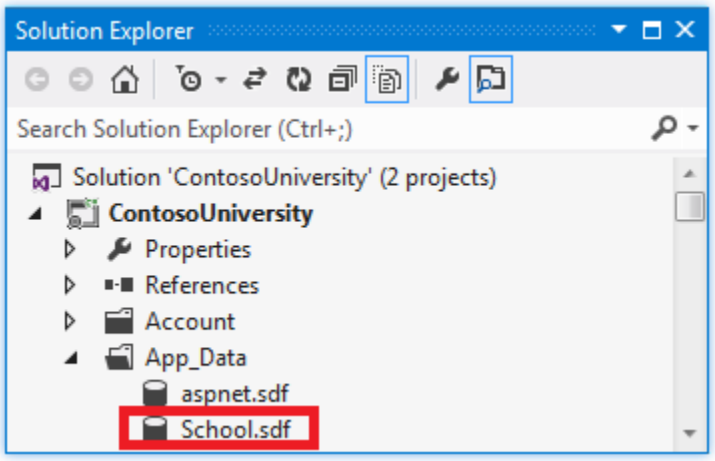
```
Sql("UPDATE Department SET Budget = 1000");
```

This example shown for this tutorial uses the `AddOrUpdate` method in the `Seed` method of the Code First Migrations `Configuration` class. Code First Migrations calls the `Seed` method after every migration, and this method updates rows that have already been inserted, or inserts them if they don't exist yet. The `AddOrUpdate` method might not be the best choice for your scenario. For more information, see Take care with EF 4.3 AddOrUpdate Method on Julie Lerman's blog.

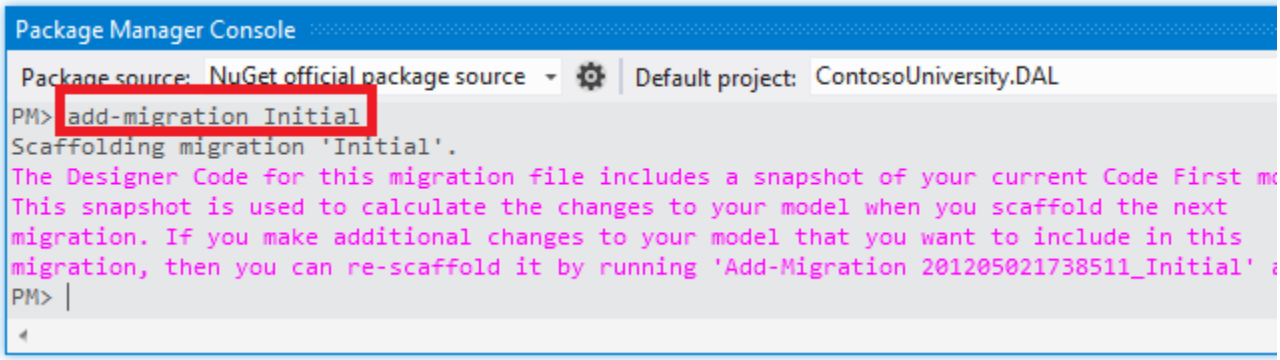
Press CTRL-SHIFT-B to build the project.

The next step is to create a `DbMigration` class for the initial migration. You want this migration to create a new database, so you have to delete the database that already exists. SQL Server Compact databases are contained in *.sdf* files in the *App_Data* folder. In **Solution Explorer**, expand *App_Data* in the ContosoUniversity project to see the two SQL Server Compact databases, which are represented by *.sdf* files.

Right-click the *School.sdf* file and click **Delete**.

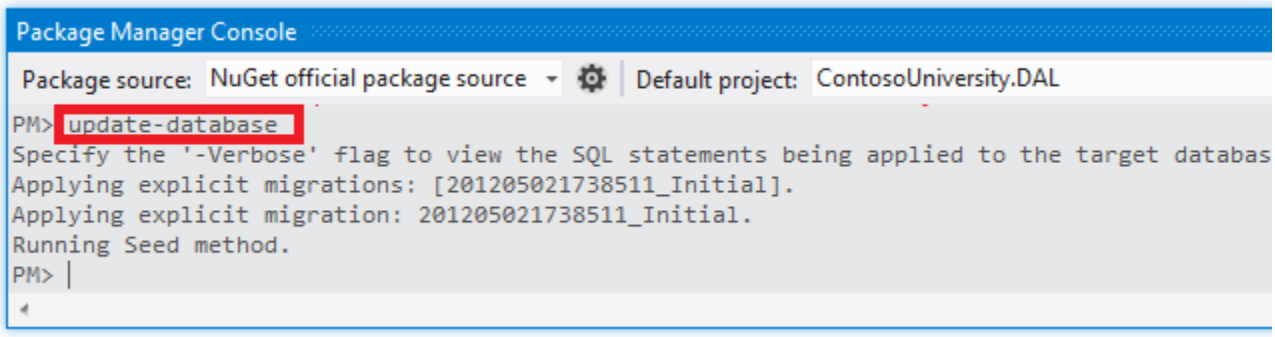


In the **Package Manager Console** window, enter the command "add-migration Initial" to create the initial migration and name it "Initial".



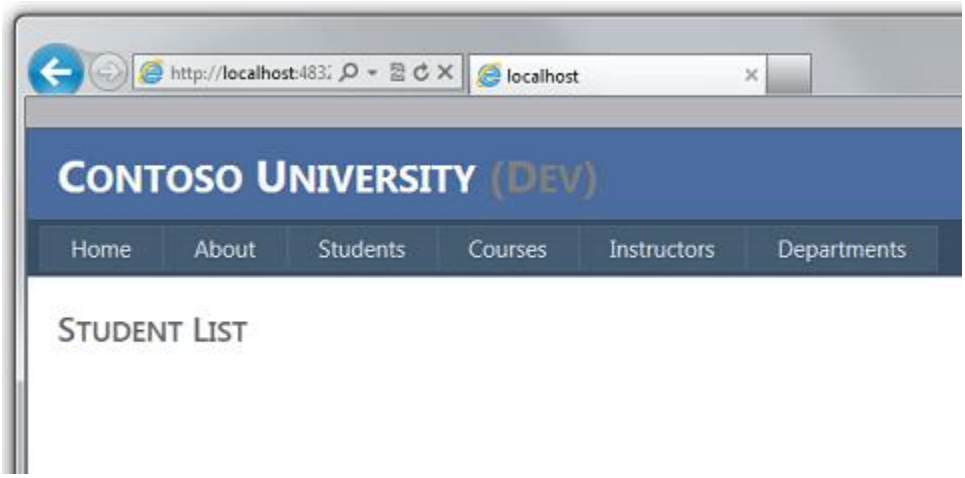
Code First Migrations creates another class file in the *Migrations* folder, and this class contains code that creates the database schema.

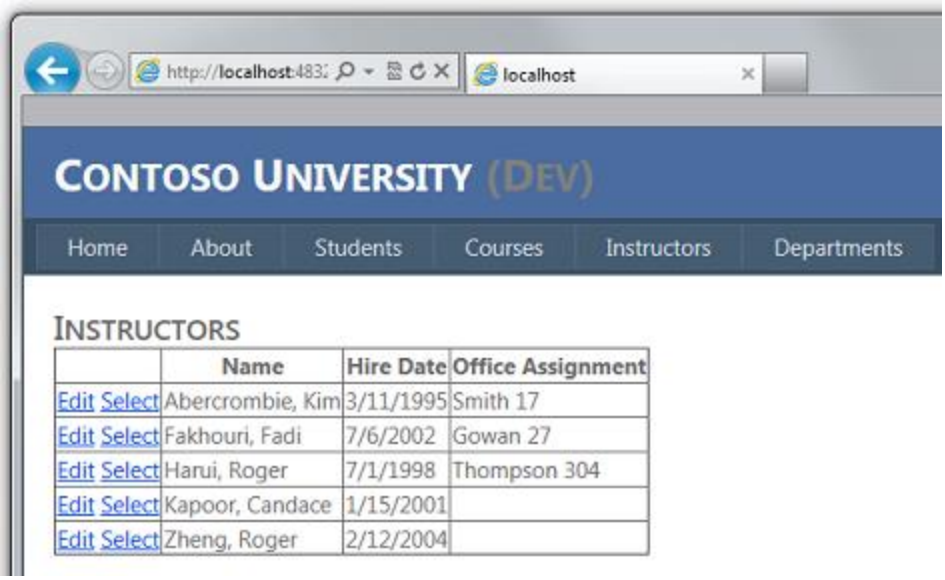
In the **Package Manager Console**, enter the command "update-database" to create the database and run the **Seed** method.



(If you get an error that indicates a table already exists and can't be created, it is probably because you ran the application after you deleted the database and before you executed `update-database`. In that case, delete the *School.sdf* file again and retry the `update-database` command.)

Run the application. Now the Students page is empty but the Instructors page contains instructors. This is what you will get in production after you deploy the application.

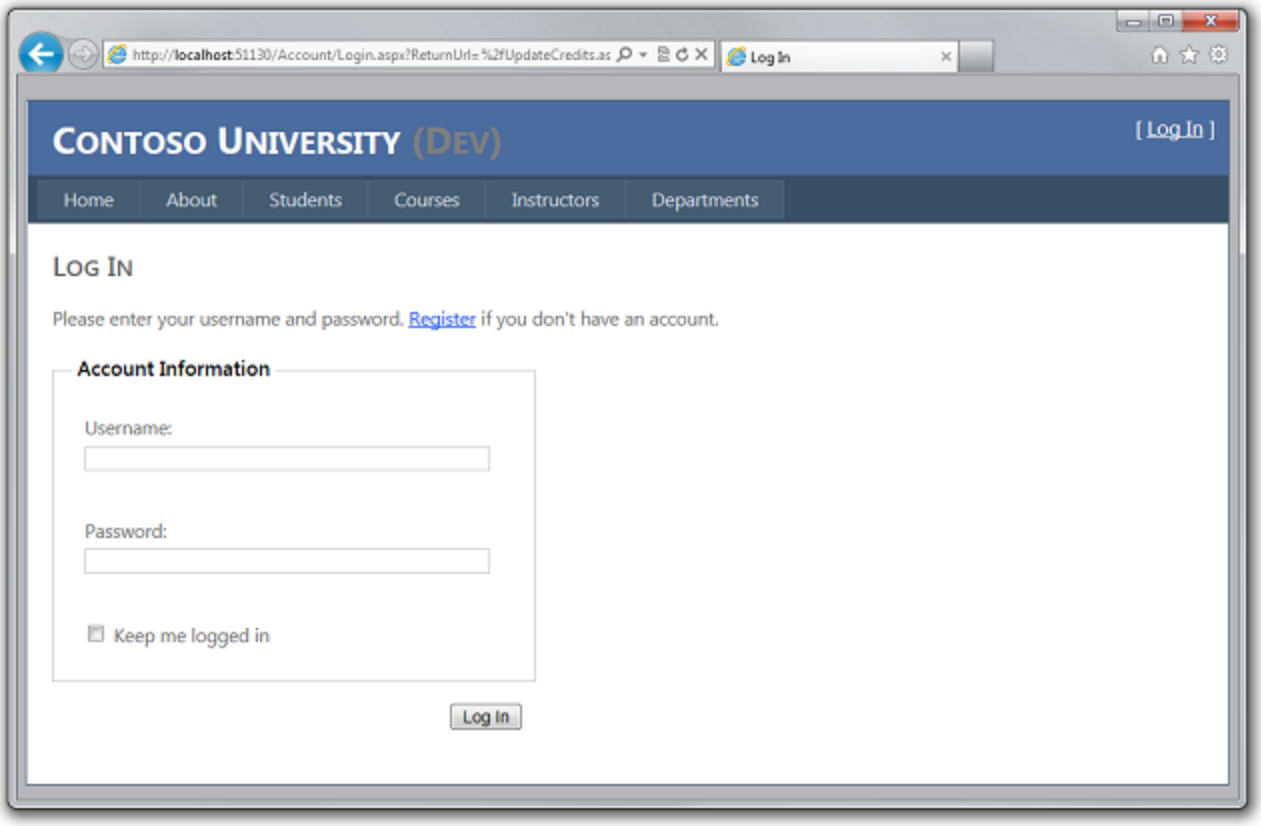




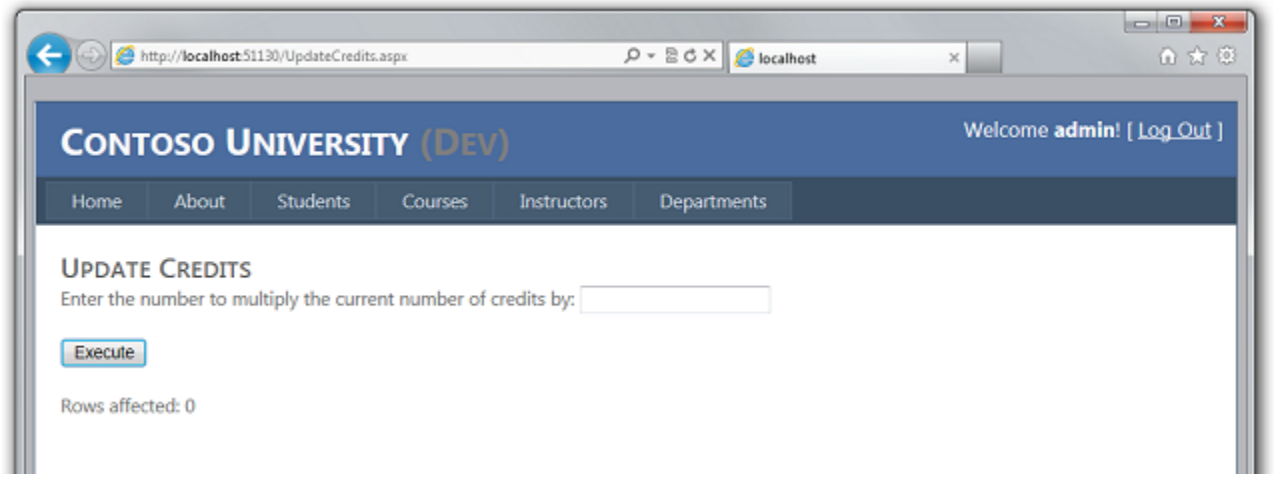
The project is now ready to deploy the *School* database.

Creating a Membership Database for Deployment

The Contoso University application uses the ASP.NET membership system and forms authentication to authenticate and authorize users. One of its pages is accessible only to administrators. To see this page, run the application and select **Update Credits** from the flyout menu under **Courses**. The application displays the **Log In** page, because only administrators are authorized to use the **Update Credits** page.



Log in as "admin" using the password "Pas$w0rd" (notice the number zero in place of the letter "o" in "w0rd"). After you log in, the **Update Credits** page is displayed.



When you deploy a site for the first time, it is common to exclude most or all of the user accounts you create for testing. In this case, you'll deploy an administrator account and no user accounts. Rather than manually deleting test accounts, you'll create a new membership database that has only the one administrator user account that you need in production.

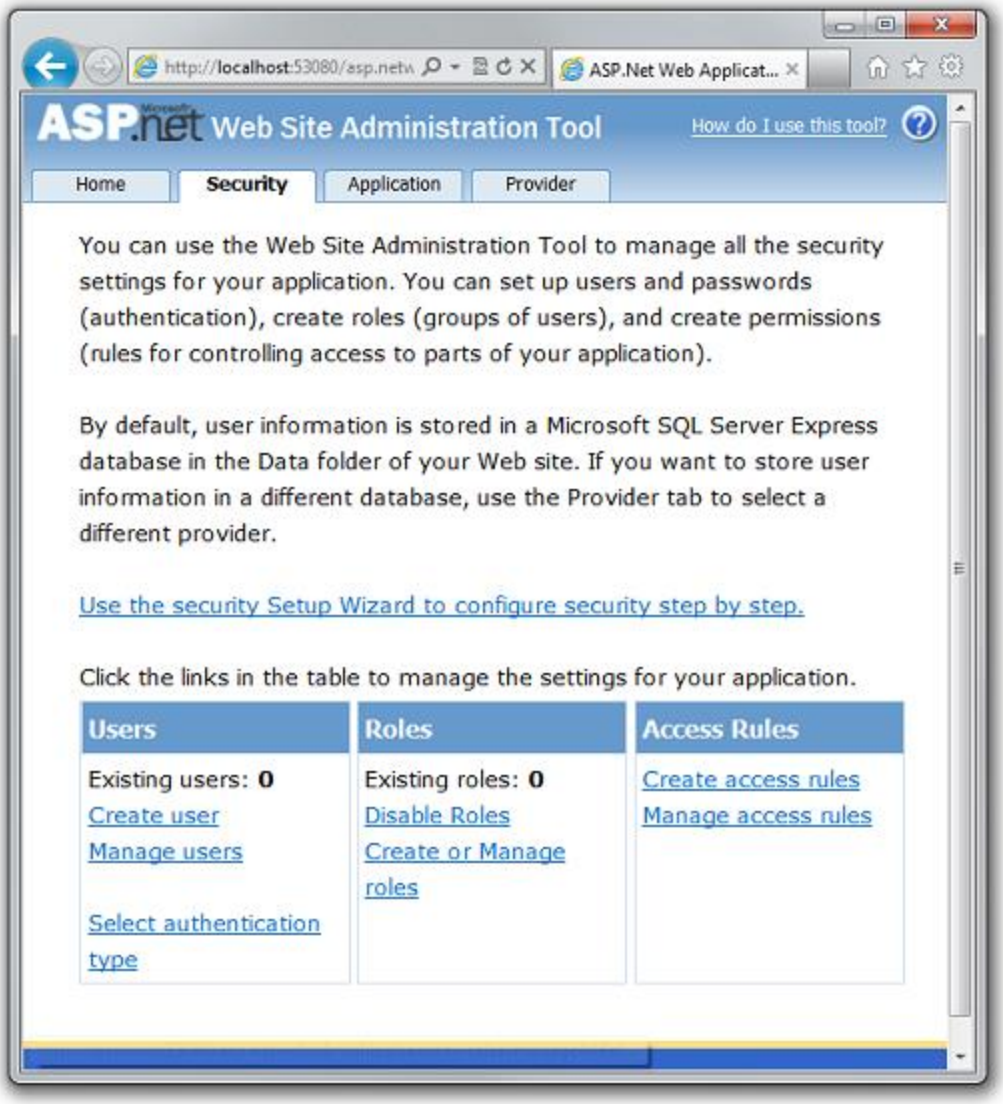
Note The membership database stores a hash of account passwords. In order to deploy accounts from one machine to another, you must make sure that hashing routines don't generate different hashes on the destination server than they do on the source computer. They will generate the same hashes when you use the ASP.NET Universal Providers, as long as you don't change the default algorithm. The default algorithm is HMACSHA256 and is specified in the **validation** attribute of the **machineKey** element in the Web.config file.

The membership database is not maintained by Code First Migrations, and there is no automatic initializer that seeds the database with test accounts (as there is for the School database). Therefore, to keep test data available you'll make a copy of the test database before you create a new one.

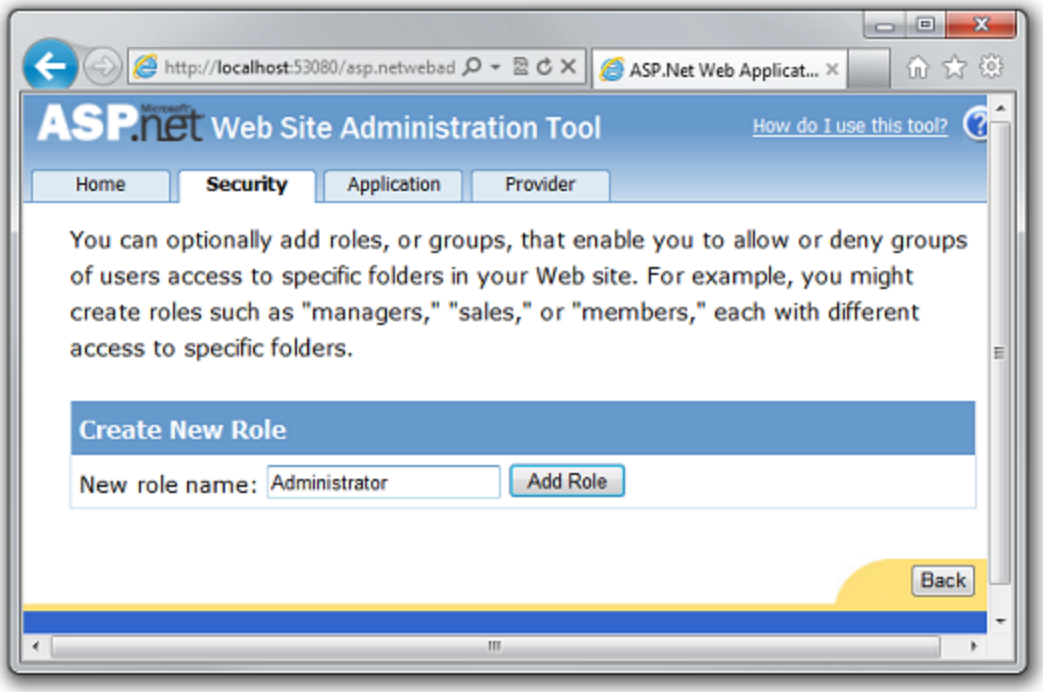
In **Solution Explorer**, rename the *aspnet.sdf* file in the *App_Data* folder to *aspnet-Dev.sdf*. (Don't make a copy, just rename it — you'll create a new database in a moment.)

In **Solution Explorer**, make sure that the web project (ContosoUniversity, not ContosoUniversity.DAL) is selected. Then in the **Project** menu, select **ASP.NET Configuration** to run the **Web Site Administration Tool** (WAT).

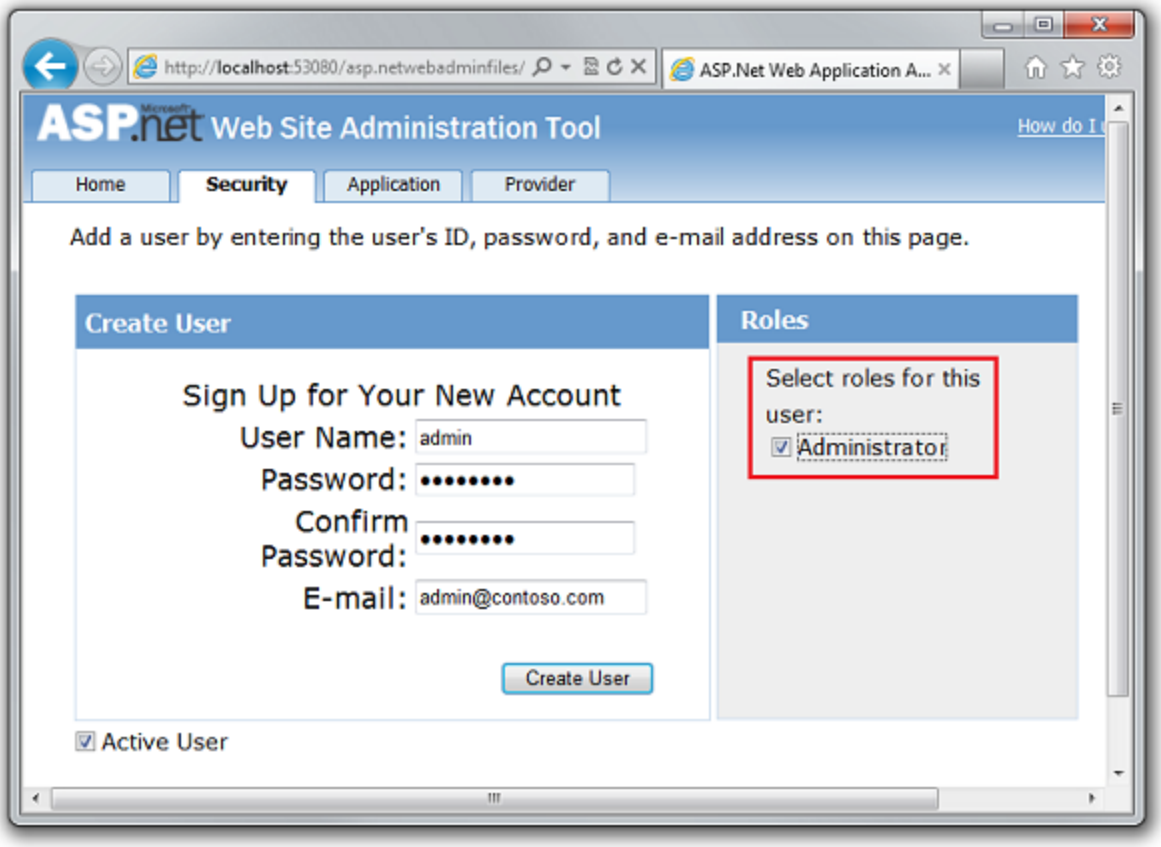
Select the **Security** tab.



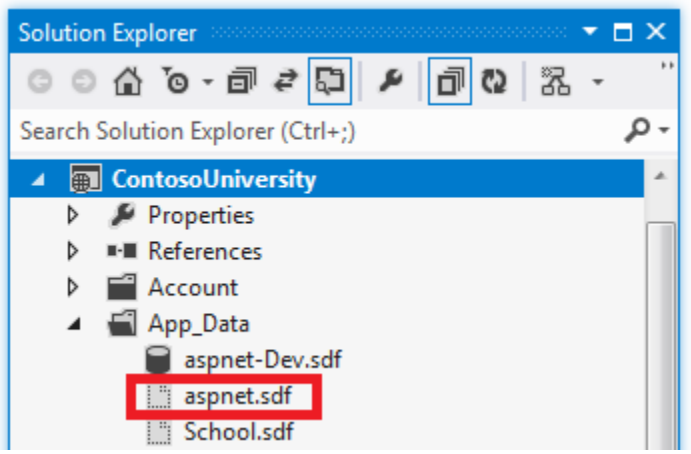
Click **Create or Manage Roles** and add an **Administrator** role.



Navigate back to the **Security** tab, click **Create User**, and add user "admin" as an administrator. Before you click the **Create User** button on the **Create User** page, make sure that you select the **Administrator** check box. The password used in this tutorial is "Pas$w0rd", and you can enter any email address.



Close the browser. In **Solution Explorer**, click the refresh button to see the new *aspnet.sdf* file.

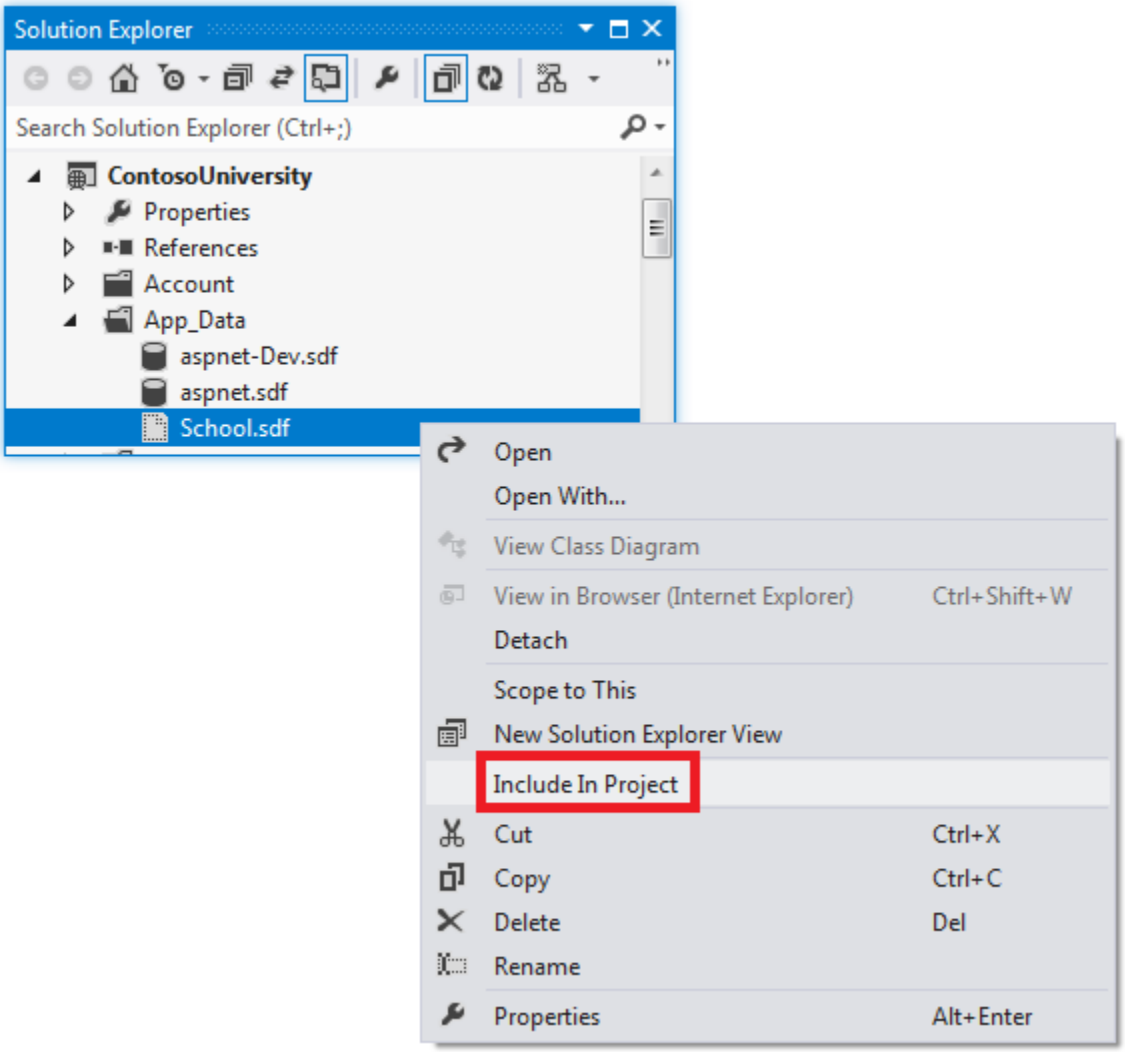


Right-click **aspnet.sdf** and select **Include in Project**.

Distinguishing Development from Production Databases

In this section, you'll rename the databases so that the development versions are School-Dev.sdf and aspnet-Dev.sdf and the production versions are School-Prod.sdf and aspnet-Prod.sdf. This isn't necessary, but doing so will help keep you from getting test and production versions of the databases confused.

In **Solution Explorer**, click **Refresh** and expand the App_Data folder to see the School database that you created earlier; right-click it and select **Include in project**.



Rename *aspnet.sdf* to *aspnet-Prod.sdf*.

Rename *School.sdf* to *School-Dev.sdf*.

When you run the application in Visual Studio you don't want to use the *-Prod* versions of the database files, you want to use *-Dev* versions. Therefore you have to change the connection strings in the Web.config file so

that they point to the *-Dev* versions of the databases. (You haven't created a School-Prod.sdf file, but that's OK because Code First will create that database in production the first time you run your app there.)

Open the application Web.config file, and locate the connection strings:

```
<configuration>
  <!-- Settings -->
  <connectionStrings>
    <add name="DefaultConnection" connectionString="Data
Source=|DataDirectory|aspnet.sdf" providerName="System.Data.SqlServerCe.4.0" />
    <add name="SchoolContext" connectionString="Data
Source=|DataDirectory|School.sdf" providerName="System.Data.SqlServerCe.4.0" />
  </connectionStrings>
  <!-- Settings -->
</configuration>
```

Change "aspnet.sdf" to "aspnet-Dev.sdf", and change "School.sdf" to "School-Dev.sdf":

```
<configuration>
  <!-- Settings -->
  <connectionStrings>
    <add name="DefaultConnection" connectionString="Data
Source=|DataDirectory|aspnet-Dev.sdf" providerName="System.Data.SqlServerCe.4.0" />
    <add name="SchoolContext" connectionString="Data Source=|DataDirectory|School-
Dev.sdf" providerName="System.Data.SqlServerCe.4.0" />
  </connectionStrings>
  <!-- Settings -->
</configuration>
```

The SQL Server Compact database engine and both databases are now ready to be deployed. In the following tutorial you set up automatic *Web.config* file transformations for settings that must be different in the development, test, and production environments. (Among the settings that must be changed are the connection strings, but you'll set up those changes later when you create a publish profile.)

More Information

For more information on NuGet, see Manage Project Libraries with NuGet and NuGet Documentation. If you don't want to use NuGet, you'll need to learn how to analyze a NuGet package to determine what it does when it is installed. (For example, it might configure *Web.config* transformations, configure PowerShell scripts to run at build time, etc.) To learn more about how NuGet works, see especially Creating and Publishing a Package and Configuration File and Source Code Transformations.

Web.Config File Transformations - 3 of 12

Overview

This tutorial shows you how to automate the process of changing the *Web.config* file when you deploy it to different destination environments. Most applications have settings in the *Web.config* file that must be different when the application is deployed. Automating the process of making these changes keeps you from having to do them manually every time you deploy, which would be tedious and error prone.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Web.config Transformations versus Web Deploy Parameters

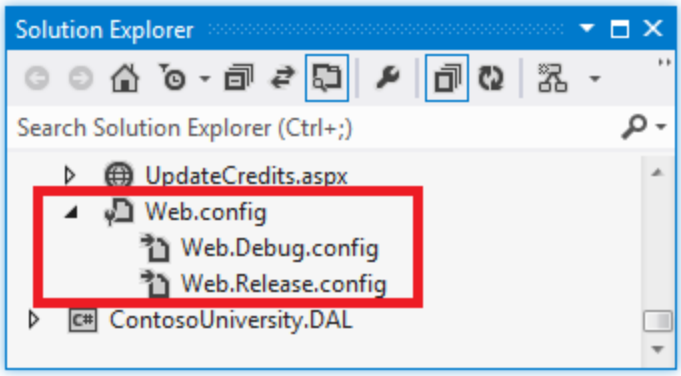
There are two ways to automate the process of changing *Web.config* file settings: Web.config transformations and Web Deploy parameters. A *Web.config* transformation file contains XML markup that specifies how to change the *Web.config* file when it is deployed. You can specify different changes for specific build configurations and for specific publish profiles. The default build configurations are Debug and Release, and you can create custom build configurations. A publish profile typically corresponds to a destination environment. (You'll learn more about publish profiles in the Deploying to IIS as a Test Environment tutorial.)

Web Deploy parameters can be used to specify many different kinds of settings that must be configured during deployment, including settings that are found in *Web.config* files. When used to specify *Web.config* file changes, Web Deploy parameters are more complex to set up, but they are useful when you do not know the value to be set until you deploy. For example, in an enterprise environment, you might create a *deployment package* and give it to a person in the IT department to install in production, and that person has to be able to enter connection strings or passwords that you do not know.

For the scenario that this tutorial covers, you know everything that has to be done to the *Web.config* file, so you do not need to use Web Deploy parameters. You'll configure some transformations that differ depending on the build configuration used, and some that differ depending on the publish profile used.

Creating Transformation Files for Publish Profiles

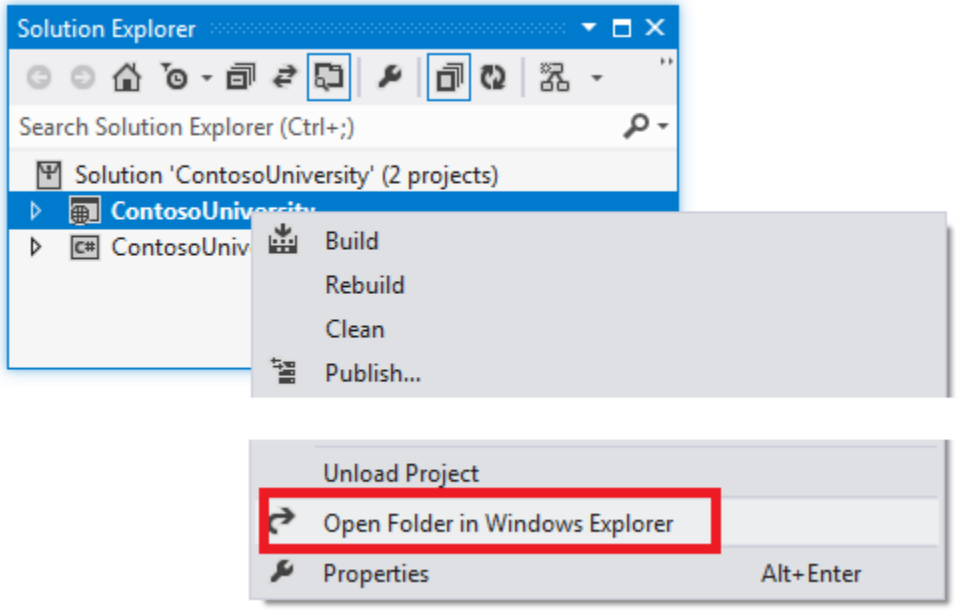
In **Solution Explorer**, expand *Web.config* to see the *Web.Debug.config* and *Web.Release.config* transformation files that are created by default for the two default build configurations.



You can create transformation files for custom build configurations by right-clicking the Web.config file and choosing **Add Config Transforms** from the context menu, but for this tutorial you don't need to do that.

You do need two more transformation files, for configuring changes that are related to the deployment destination rather than to the build configuration. A typical example of this kind of setting is a WCF endpoint that is different for test versus production. In later tutorials you'll create publish profiles named Test and Production, so you need a *Web.Test.config* file and a *Web.Production.config* file.

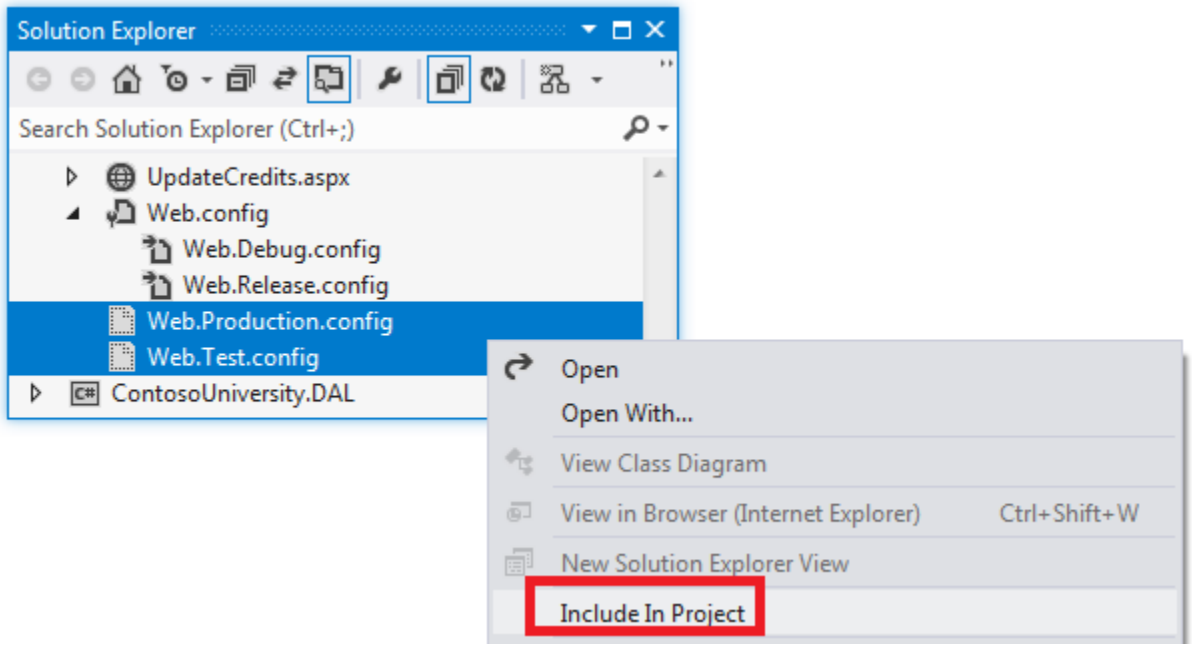
Transformation files that are tied to publish profiles must be created manually. In **Solution Explorer**, right-click the ContosoUniversity project and select **Open Folder in Windows Explorer**.



In **Windows Explorer**, select the *Web.Release.config* file, copy the file, and then paste two copies of it. Rename these copies *Web.Production.config* and *Web.Test.config*, then close **Windows Explorer**.

In **Solution Explorer**, click **Refresh** to see the new files.

Select the new files, right-click, and then click **Include in Project** in the context menu.



To prevent these files from being deployed, select them in **Solution Explorer**, and then in the **Properties** window change the **Build Action** property from **Content** to **None**. (The transformation files that are based on build configurations are automatically prevented from deploying.)

You are now ready to enter *Web.config* transformations into the *Web.config* transformation files.

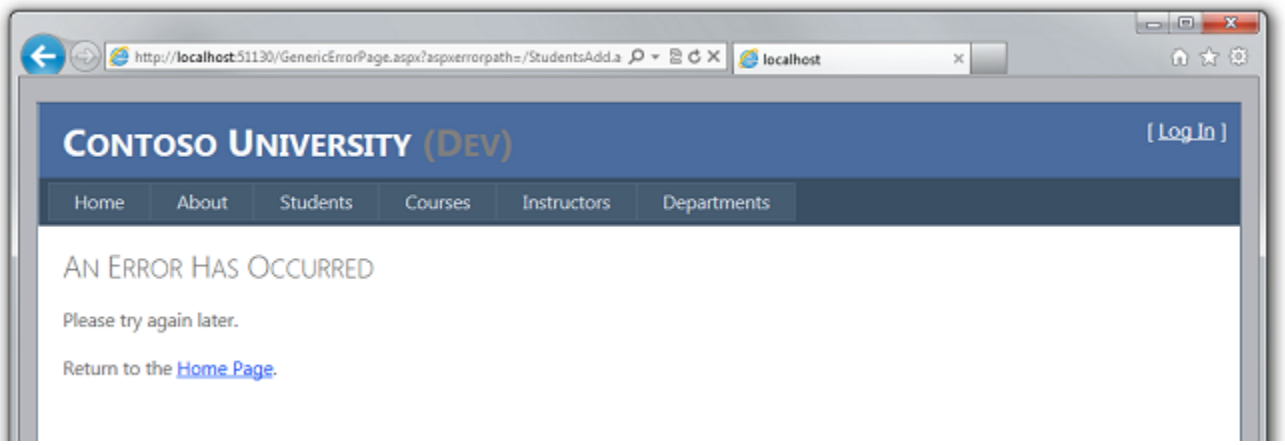
Limiting Error Log Access to Administrators

If there's an error while the application runs, the application displays a generic error page in place of the system-generated error page, and it uses Elmah NuGet package for error logging and reporting. The `customErrors` element in the *Web.config* file specifies the error page:

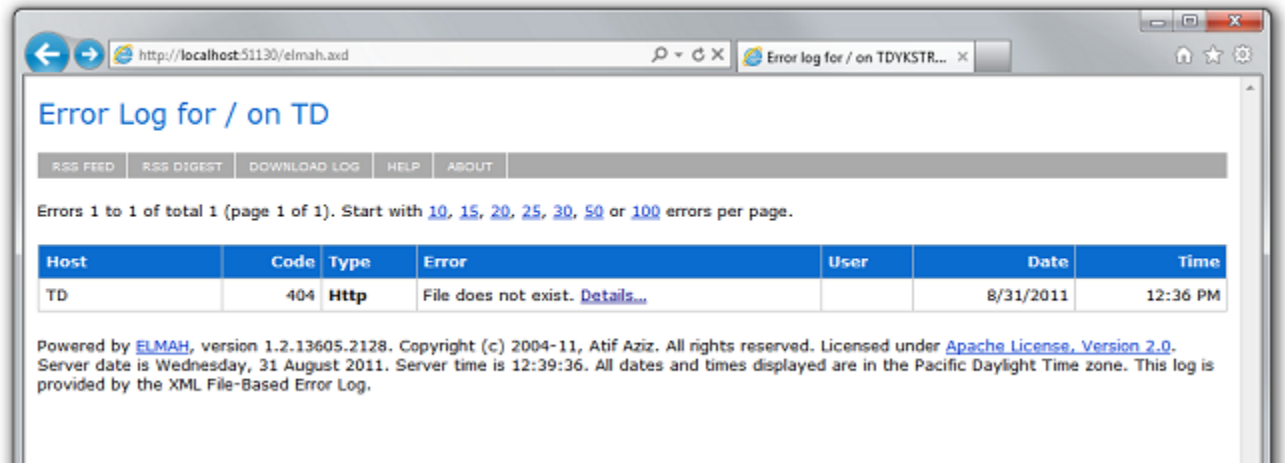
```
<customErrors mode="RemoteOnly" defaultRedirect="~/GenericErrorPage.aspx">
  <error statusCode="404" redirect="~/GenericErrorPage.aspx" />
</customErrors>
```

To see the error page, temporarily change the `mode` attribute of the `customErrors` element from "RemoteOnly" to "On" and run the application from Visual Studio. Cause an error by requesting an invalid URL,

such as *Studentsxxx.aspx*. Instead of an IIS-generated "page not found" error page, you see the *GenericErrorPage.aspx* page.



To see the error log, replace everything in the URL after the port number with *elmah.axd* (for the example in the screen shot, *http://localhost:51130/elmah.axd*) and press Enter:



Don't forget to set the `customErrors` element back to "RemoteOnly" mode when you're done.

On your development computer it's convenient to allow free access to the error log page, but in production that would be a security risk. For the production site, you can add an authorization rule that restricts error log access just to administrators by configuring a transform in the *Web.Production.config* file.

Open *Web.Production.config* and add a new `location` element immediately after the opening `configuration` tag, as shown here. (Make sure that you add only the `location` element and not the surrounding markup which is shown here only to provide some context.)

```
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
    <location path="elmah.axd" xdt:Transform="Insert">
      <system.web>
        <authorization>
          <allow roles="Administrator" />
          <deny users="*" />
        </authorization>
      </system.web>
    </location>
</configuration>
```

The `Transform` attribute value of "Insert" causes this `location` element to be added as a sibling to any existing `location` elements in the *Web.config* file. (There is already one `location` element that specifies authorization rules for the **Update Credits** page.) When you test the production site after deployment, you'll test to verify that this authorization rule is effective.

You don't have to restrict error log access in the test environment, so you don't have to add this code to the *Web.Test.config* file.

Security Note Never display error details to the public in a production application, or store that information in a public location. Attackers can use error information to discover vulnerabilities in a site. If you use ELMAH in your own application, be sure to investigate ways in which ELMAH can be configured to minimize security risks. The ELMAH example in this tutorial should not be considered a recommended configuration. It is an example that was chosen in order to illustrate how to handle a folder that the application must be able to create files in.

Setting an Environment Indicator

A common scenario is to have *Web.config* file settings that must be different in each environment that you deploy to. For example, an application that calls a WCF service might need a different endpoint in test and production environments. The Contoso University application includes a setting of this kind also. This setting controls a visible indicator on a site's pages that tells you which environment you are in, such as development, test, or production. The setting value determines whether the application will append "(Dev)" or "(Test)" to the main heading in the *Site.Master* master page:

The environment indicator is omitted when the application is running in production.

The Contoso University web pages read a value that is set in `appSettings` in the *Web.config* file in order to determine what environment the application is running in:

```
<appSettings>
    <add key="Environment" value="Dev" />
</appSettings>
```

The value should be "Test" in the test environment, and "Prod" in the production environment.

Open *Web.Production.config* and add an `appSettings` element immediately before the opening tag of the `location` element you added earlier:

```
<appSettings>
    <add key="Environment" value="Prod" xdt:Transform="SetAttributes"
xdt:Locator="Match(key)"/>
</appSettings>
```

The `xdt:Transform` attribute value "SetAttributes" indicates that the purpose of this transform is to change attribute values of an existing element in the *Web.config* file. The `xdt:Locator` attribute value "Match(key)" indicates that the element to be modified is the one whose `key` attribute matches the `key` attribute specified here. The only other attribute of the `add` element is `value`, and that is what will be changed in the deployed *Web.config* file. This code causes the `value` attribute of the `Environment appSettings` element to be set to "Prod" in the *Web.config* file that is deployed to production.

Next, apply the same change to *Web.Test.config* file, except set the `value` to "Test" instead of "Prod". When you are done, the `appSettings` section in *Web.Test.config* will look like the following example:

```
<appSettings>
    <add key="Environment" value="Test" xdt:Transform="SetAttributes"
xdt:Locator="Match(key)"/>
</appSettings>
```

Disabling Debug Mode

For a Release build, you do not want debugging enabled regardless of which environment you are deploying to. By default the *Web.Release.config* transform file is automatically created with code that removes the `debug` attribute from the `compilation` element:

```
<system.web>
  <compilation xdt:Transform="RemoveAttributes(debug)" />
</system.web>
```

The `Transform` attribute causes the `debug` attribute to be omitted from the deployed *Web.config* file whenever you deploy a Release build.

This same transformation is in Test and Production transform files because you created them by copying the Release transform file. You don't need it duplicated there, so open each of those files, remove the **compilation** element, and save and close each file.

Setting Connection Strings

In most cases you do not need to set up connection string transformations, because you can specify connection strings in the publish profile. But there is an exception when you're deploying a SQL Server Compact database and you're using Entity Framework Code First Migrations to update the database on the destination server. For this scenario, you have to specify an additional connection string that will be used on the server for updating the database schema. To set up this transformation, add a **<connectionStrings>** element immediately after the opening **<configuration>** tag in both the *Web.Test.config* and the *Web.Production.config* transform files:

```
<connectionStrings>
  <add name="SchoolContext_DatabasePublish" connectionString="Data
Source=|DataDirectory|School-Prod.sdf" providerName="System.Data.SqlServerCe.4.0"
xdt:Transform="Insert"/>
</connectionStrings>
```

The `Transform` attribute specifies that this connection string will be added to the *connectionStrings* element in the deployed *Web.config* file. (The publish process creates this additional connection string automatically for you if it doesn't exist, but by default the **providerName** attribute gets set to `System.Data.SqlClient`, which does not work not for SQL Server Compact. By adding the connection string manually, you keep the deployment process from creating a connection string element with the wrong provider name.)

You have now specified all of the *Web.config* transformations that you need for deploying the Contoso University application to test and production. In the following tutorial, you'll take care of deployment set-up tasks that require setting project properties.

More Information

For more information about topics covered by this tutorial, see the Web.config transformation scenario in ASP.NET Deployment Content Map.

Configuring Project Properties - 4 of 12

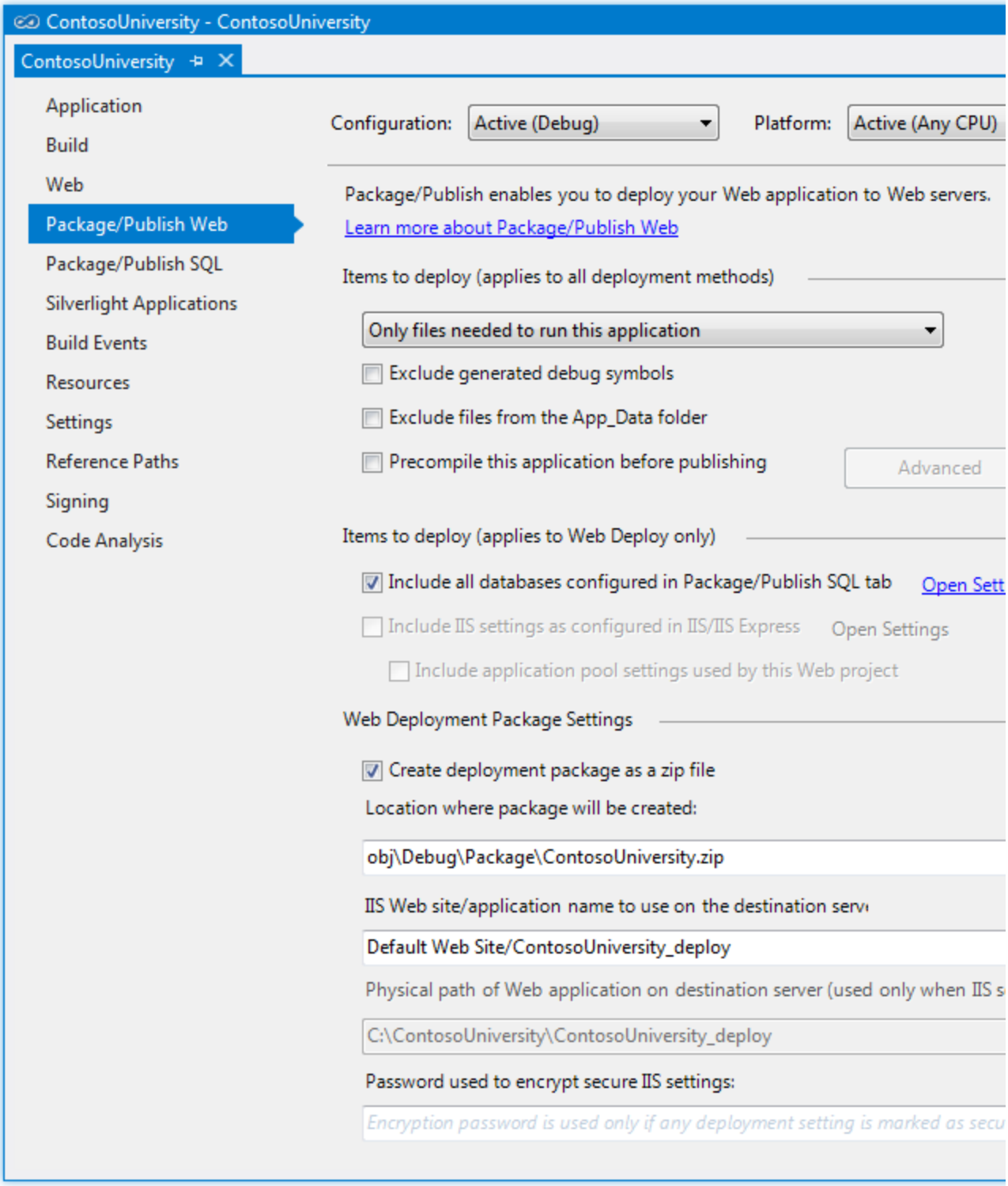
Overview

Some deployment options are configured in project properties that are stored in the project file (the *.csproj* or *.vbproj* file). In most cases, the default values of these settings are what you want, but you can use the **Project Properties** UI built into Visual Studio to work with these settings if you have to change them. In this tutorial you review the deployment settings in **Project Properties**. You also create a placeholder file that causes an empty folder to be deployed.

Configuring Deployment Settings in the Project Properties Window

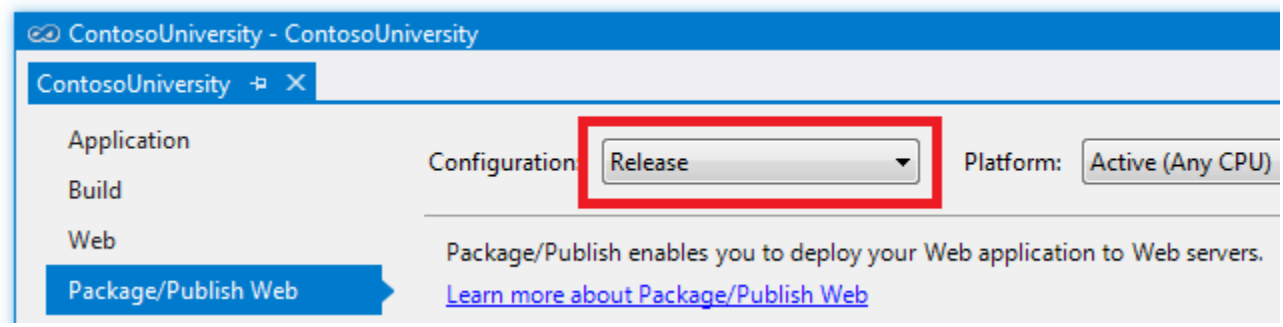
Most settings that affect what happens during deployment are included in the publish profile, as you'll see in the following tutorials. A few settings that you should be aware of are located in the **Package/Publish** tabs of the **Project Properties** window. These settings are specified for each build configuration — that is, you can have different settings for a Release build than you have for a Debug build.

In **Solution Explorer**, right-click the **ContosoUniversity** project, select **Properties**, and then select the **Package/Publish Web** tab.



When the window is displayed, it defaults to showing settings for whichever build configuration is currently active for the solution. If the **Configuration** box does not indicate **Active (Release)**, select **Release** in order to

display settings for the Release build configuration. You'll deploy Release builds to both your test and production environments.



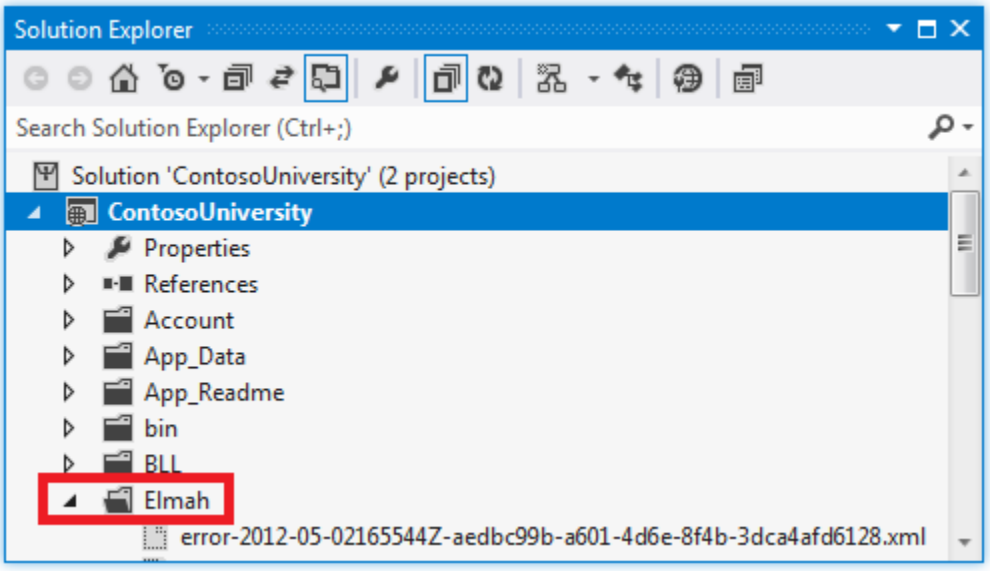
With **Active (Release)** or **Release** selected, you see the values that are effective when you deploy using the Release build configuration:

- In the **Items to deploy** box, **Only files needed to run the application** is selected. Other options are **All files in this project** or **All files in this project folder**. By leaving the default selection unchanged you avoid deploying source code files, for example. This setting is the reason why the folders that contain the SQL Server Compact binary files had to be included in the project. For more information about this setting, see **Why don't all of the files in my project folder get deployed?** in ASP.NET Web Application Project Deployment FAQ.
- **Exclude generated debug symbols** is selected. You won't be debugging when you use this build configuration.
- **Exclude files from the App_Data folder** is not selected. Your SQL Server Compact file for the membership database is in that folder and you have to deploy it. When you deploy updates that do not include database changes, you'll select this checkbox.
- **Precompile this application before publishing** is not selected. In most scenarios, there's no need to precompile web application projects. For more information about this option, see Package/Publish Web Tab, Project Properties and Advanced Precompile Settings Dialog.
- **Include all databases configured in Package/Publish SQL tab** is selected, but this option has no effect now because you aren't configuring the **Package/Publish SQL** tab. That tab is for a legacy database deployment method that used to be the only option for deploying SQL Server databases. You'll use the **Package/Publish SQL** tab in the Migrating to SQL Server tutorial.
- The **Web Deployment Package Settings** section does not apply because you're using one-click publish in these tutorials.

Change the **Configuration** drop-down box to Debug to see the default settings for Debug builds. The values are the same, except **Exclude generated debug symbols** is cleared so that you can debug when you deploy a Debug build.

Making Sure that the Elmah Folder gets Deployed

As you saw in the previous tutorial, the Elmah NuGet package provides functionality for error logging and reporting. In the Contoso University application Elmah has been configured to store error details in a folder named *Elmah*:



Excluding specific files or folders from deployment is a common requirement; another example would be a folder that users can upload files to. You don't want log files or uploaded files that were created in your development environment to be deployed to production. And if you are deploying an update to production you don't want the deployment process to delete files that exist in production. (Depending on how you set a deployment option, if a file exists in the destination site but not the source site when you deploy, Web Deploy deletes it from the destination.)

As you saw earlier in this tutorial, the **Items to deploy** option in the **Package/Publish Web** tab is set to **Only Files Needed to run this application**. As a result, log files that are created by Elmah in development will not be deployed, which is what you want to happen. (To be deployed, they would have to be included in the project and their **Build Action** property would have to be set to **Content**. For more information, see **Why don't all of the files in my project folder get deployed?** in ASP.NET Web Application Project Deployment FAQ). However, Web Deploy will not create a folder in the destination site unless there's at least one file to copy to it. Therefore, you'll add a *.txt* file to the folder to act as a placeholder so that the folder will be copied.

In **Solution Explorer**, right-click the *Elmah* folder, select **Add New Item**, and create a file named *Placeholder.txt*. Put the following text in it: "This is a placeholder file to ensure that the folder gets deployed." and save the file. That's all you have to do in order to make sure that Visual Studio deploys this file and the folder it's in, because the **Build Action** property of *.txt* files is set to **Content** by default.

You have now completed all of the deployment set-up tasks. In the next tutorial, you'll deploy the Contoso University site to the test environment and test it there.

Deploying to IIS as a Test Environment - 5 of 12

Overview

This tutorial shows how to deploy an ASP.NET web application to IIS on the local computer.

When you develop an application, you generally test by running it in Visual Studio. By default, this means you're using the Visual Studio Development Server (also known as Cassini). The Visual Studio Development Server makes it easy to test during development in Visual Studio, but it doesn't work exactly like IIS. As a result, it's possible that an application will run correctly when you test it in Visual Studio, but fail when it's deployed to IIS in a hosting environment.

You can test your application more reliably in these ways:

1. Use IIS Express or full IIS instead of the Visual Studio Development Server when you test in Visual Studio during development. This method generally emulates more accurately how your site will run under IIS. However, this method does not test your deployment process or validate that the result of the deployment process will run correctly.

2. Deploy the application to IIS on your development computer by using the same process that you'll use later to deploy it to your production environment. This method validates your deployment process in addition to validating that your application will run correctly under IIS.

3. Deploy the application to a test environment that is as close as possible to your production environment. Since the production environment for these tutorials is a third-party hosting provider, the ideal test environment would be a second account with the hosting provider. You would use this second account only for testing, but it would be set up the same way as the production account.

This tutorial shows the steps for option 2. Guidance for option 3 is provided at the end of the Deploying to the Production Environment tutorial, and at the end of this tutorial there are links to resources for option 1.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Configuring the Application to Run in Medium Trust

Before installing IIS and deploying to it, you'll change a Web.config file setting in order to make the site run more like it will in a typical shared hosting environment.

Hosting providers typically run your web site in *medium trust*, which means there are some things it is not allowed to do. For example, application code can't access the Windows registry and can't read or write files that are outside of your application's folder hierarchy. By default your application runs in *high trust* on your local computer, which means that the application might be able to do things that would fail when you deploy it to production. Therefore, to make the test environment more accurately reflect the production environment, you'll configure the application to run in medium trust.

In the application Web.config file, add a **trust** element in the **system.web** element, as shown in this example.

```
<configuration>
  <!-- Settings -->
  <system.web>
    <trust level="Medium" />
    <!-- Settings -->
  </system.web>
</configuration>
```

The application will now run in medium trust in IIS even on your local computer. This setting enables you to catch as early as possible any attempts by application code to do something that would fail in production.

Note If you are using Entity Framework Code First Migrations, make sure that you have version 5.0 or later installed. In Entity Framework version 4.3, Migrations requires full trust in order to update the database schema.

Installing IIS and Web Deploy

To deploy to IIS on your development computer, you must have IIS and Web Deploy installed. These are not included in the default Windows 7 configuration. If you have already installed both IIS and Web Deploy, skip to the next section.

Using the Web Platform Installer is the preferred way to install IIS and Web Deploy, because the Web Platform Installer installs a recommended configuration for IIS and it automatically installs the prerequisites for IIS and Web Deploy if necessary.

To run Web Platform Installer to install IIS and Web Deploy, use the following link. If you already have installed IIS, Web Deploy or any of their required components, the Web Platform Installer installs only what is missing.

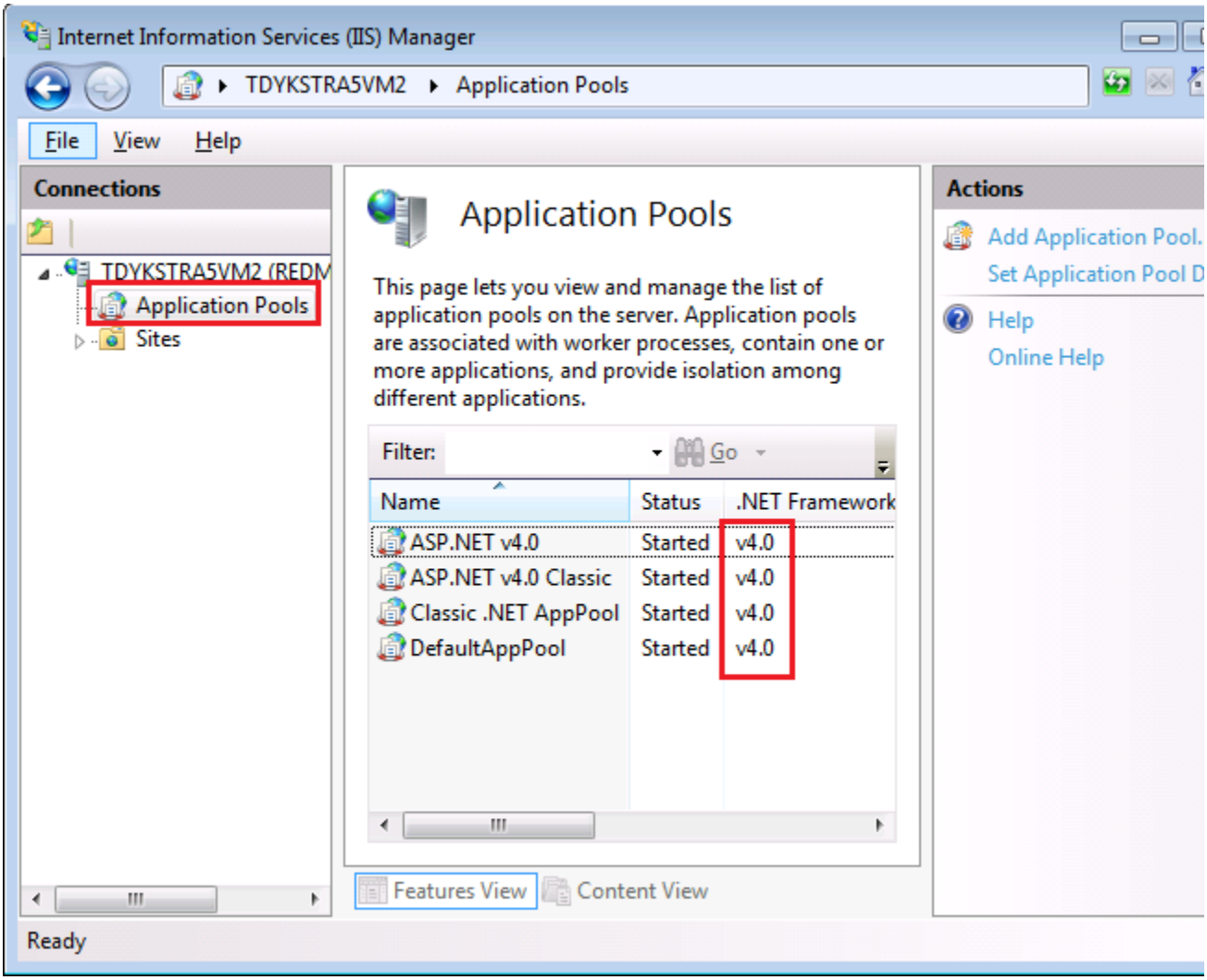
- Install IIS and Web Deploy using WebPI

Setting the Default Application Pool to .NET 4

After installing IIS, run **IIS Manager** to make sure that the .NET Framework version 4 is assigned to the default application pool.

From the Windows **Start** menu, select **Run**, enter "inetmgr", and then click **OK**. (If the **Run** command is not in your **Start** menu, you can press the Windows Key and R to open it. Or right-click the taskbar, click **Properties**, select the **Start Menu** tab, click **Customize**, and select **Run command**.)

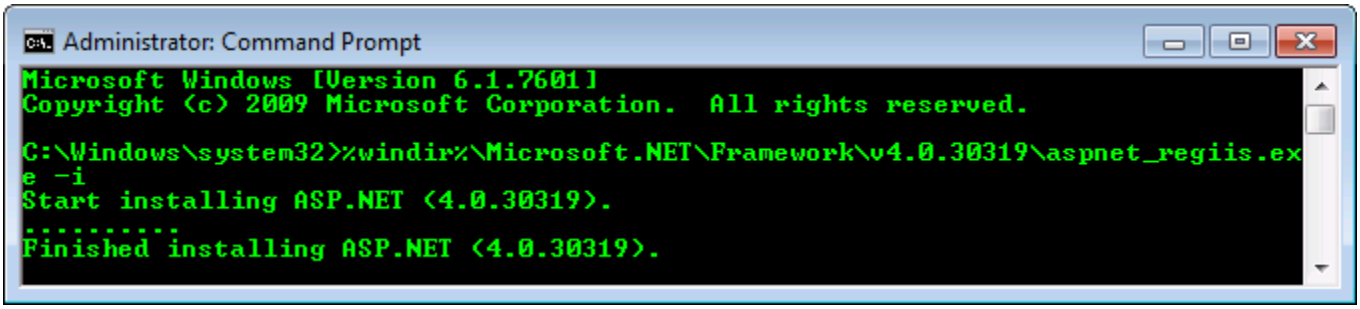
In the **Connections** pane, expand the server node and select **Application Pools**. In the **Application Pools** pane, if **DefaultAppPool** is assigned to the .NET framework version 4 as in the following illustration, skip to the next section.



If you see only two application pools and both of them are set to the .NET Framework 2.0, you have to install ASP.NET 4 in IIS:

- Open a command prompt window by right-clicking **Command Prompt** in the Windows **Start** menu and selecting **Run as Administrator**. Then run aspnet_regiis.exe to install ASP.NET 4 in IIS, using the following commands. (In 64-bit systems, replace "Framework" with "Framework64".)

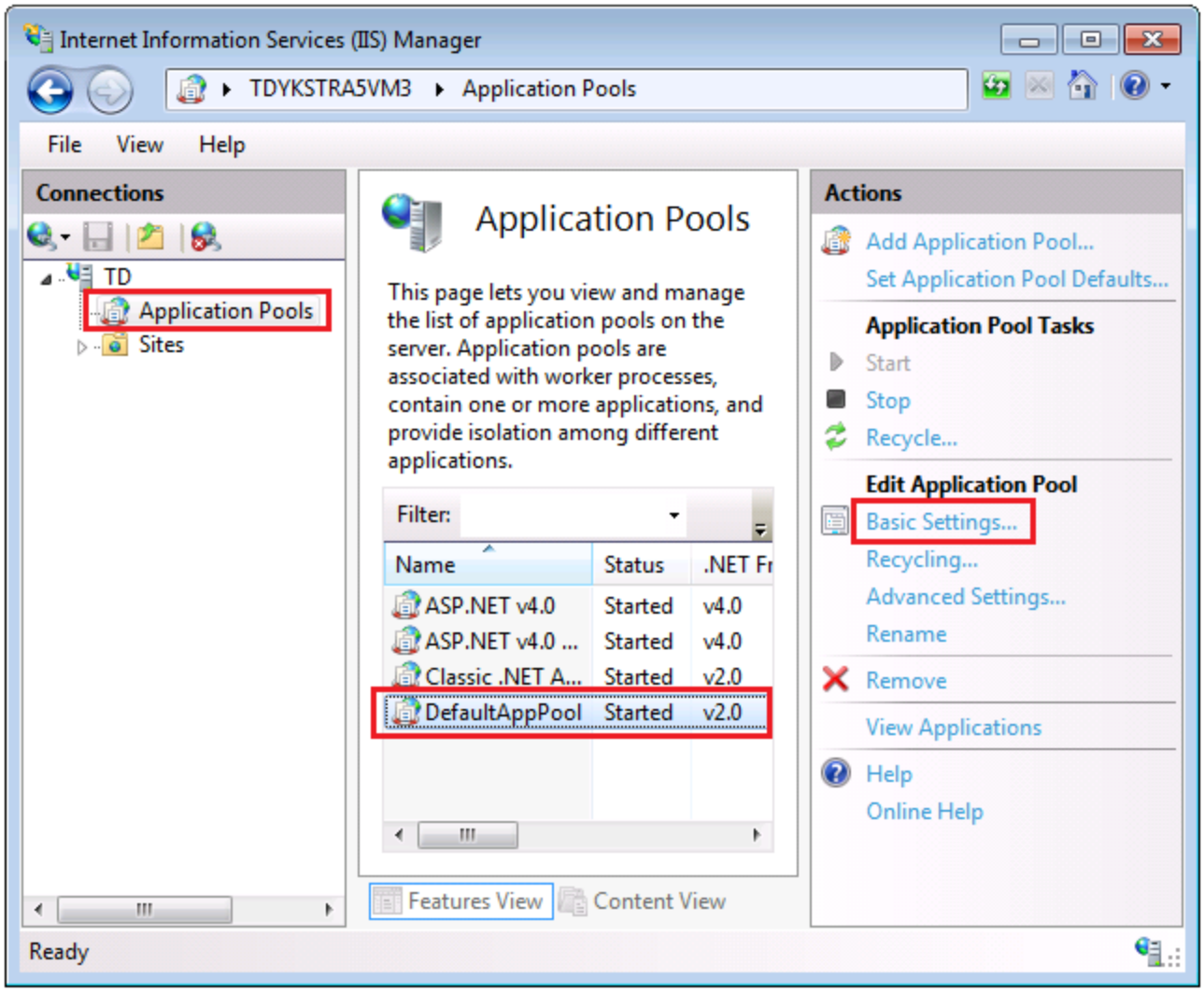
```
cd %windir%\Microsoft.NET\Framework\v4.0.30319
aspnet_regiis.exe -iru
```



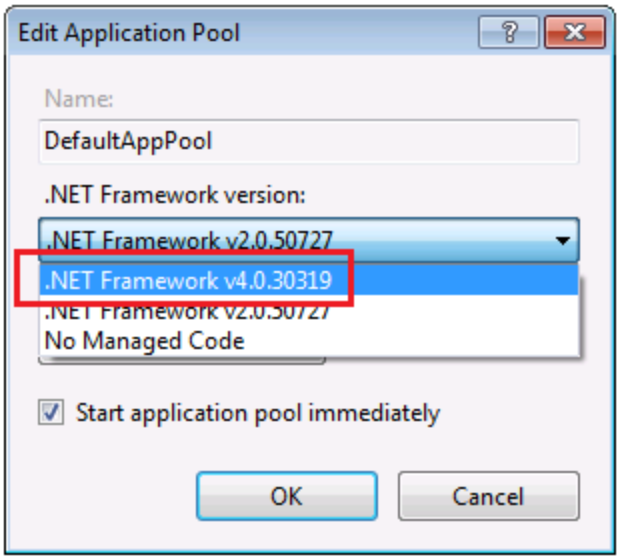
  This command creates new application pools for the .NET Framework 4, but the default application pool will still be set to 2.0. You'll be deploying an application that targets .NET 4 to that application pool, so you have to change the application pool to .NET 4.

If you closed **IIS Manager**, run it again, expand the server node, and click **Application Pools** to display the **Application Pools** pane again.

In the **Application Pools** pane, click **DefaultAppPool**, and then in the **Actions** pane click **Basic Settings**.



In the **Edit Application Pool** dialog box, change **.NET Framework version** to **.NET Framework v4.0.30319** and click **OK**.



You are now ready to publish to IIS.

Publishing to IIS

There are several ways you can deploy using Visual Studio 2010 and Web Deploy:

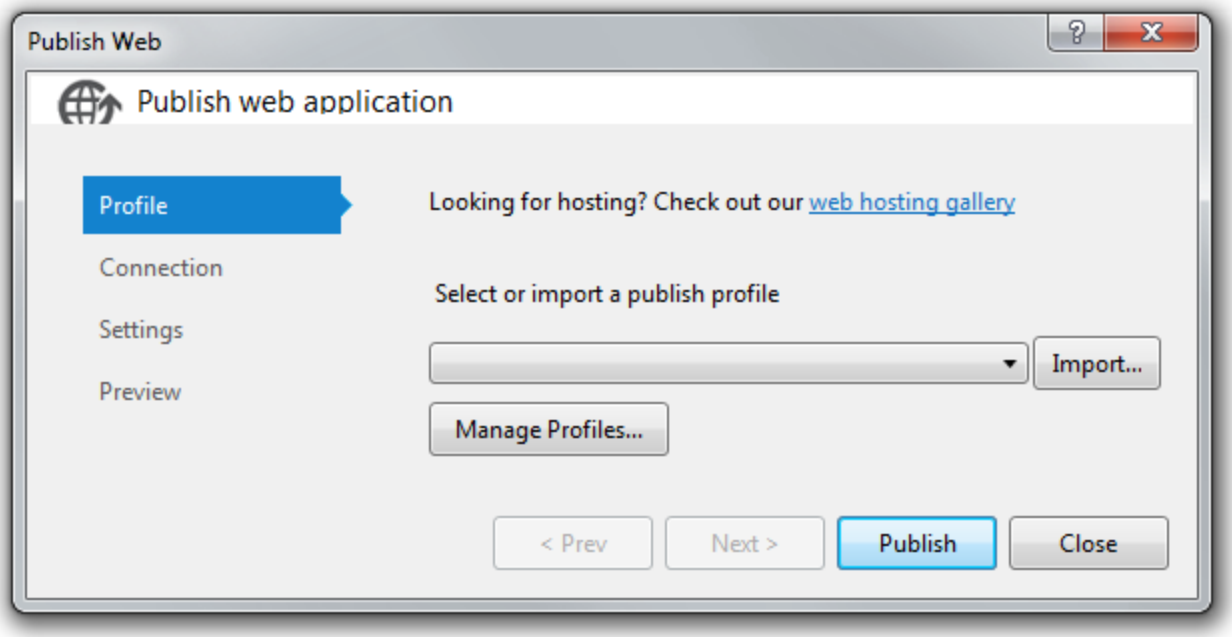
- Use Visual Studio one-click publish.
- Create a *deployment package* and install it using the IIS Manager UI. The deployment package consists of a *.zip* file that contains all the files and metadata needed to install a site in IIS.
- Create a deployment package and install it using the command line.

The process you went through in the previous tutorials to set up Visual Studio to automate deployment tasks applies to all of these three methods. In these tutorials you'll use the first of these methods. For information about using deployment packages, see ASP.NET Deployment Content Map.

Before publishing, make sure that you are running Visual Studio in administrator mode. (In the Windows 7 **Start** menu, right-click the icon for the version of Visual Studio you're using and select **Run as Administrator**.) Administrator mode is required for publishing only when you are publishing to IIS on the local computer.

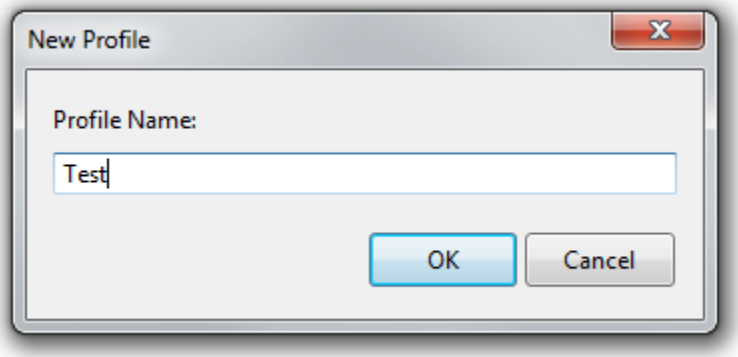
In **Solution Explorer**, right-click the ContosoUniversity project (not the ContosoUniversity.DAL project) and select **Publish**.

The **Publish Web** wizard appears.



In the drop-down list, select **<New...>**.

In the **New Profile** dialog box, enter "Test", and then click **OK**.



This name is the same as the middle node of the Web.Test.config transform file that you created earlier. This correspondence is what causes the Web.Test.config transformations to be applied when you publish by using this profile.

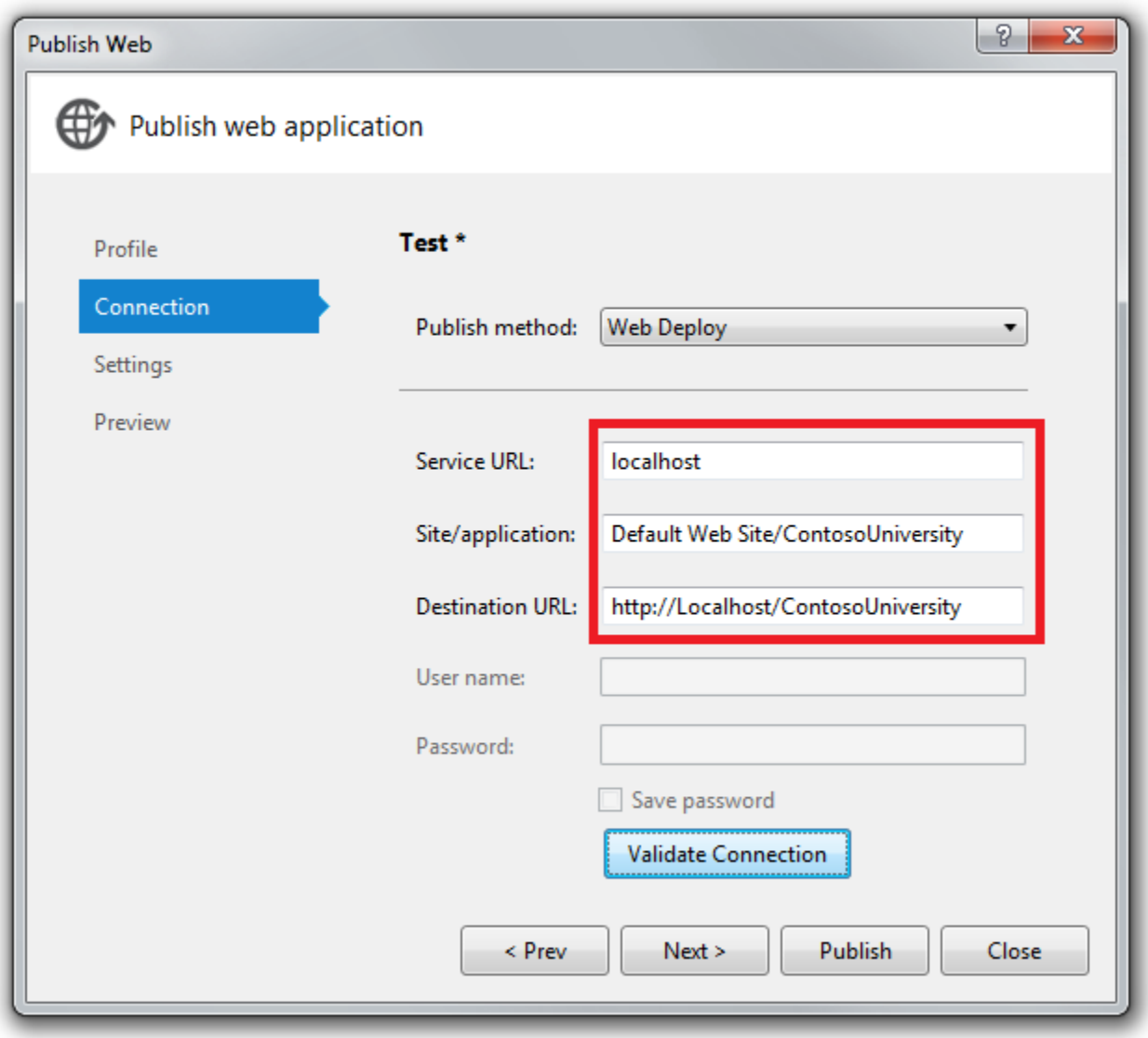
The wizard automatically advances to the **Connection** tab.

In the **Service URL** box, enter *localhost*.

In the **Site/application** box, enter *Default Web Site/ContosoUniversity*.

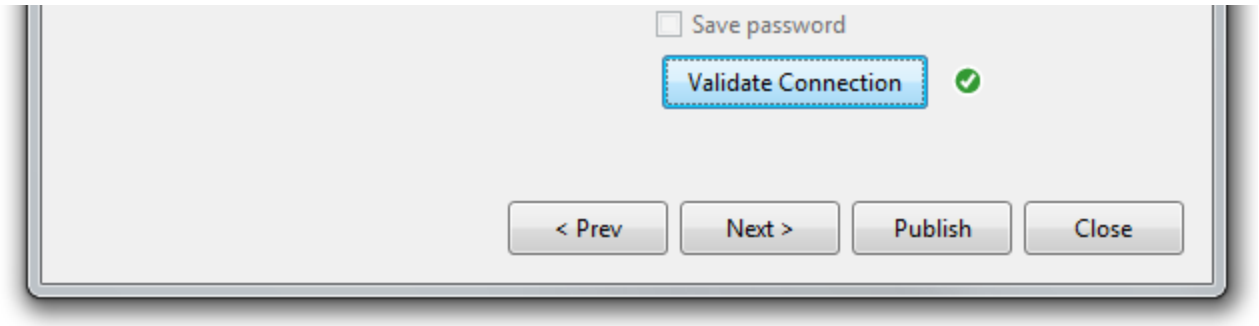
In the **Destination URL** box, enter *http://localhost/ContosoUniversity*.

The **Destination URL** setting isn't required. When Visual Studio finishes deploying the application, it automatically opens your default browser to this URL. If you don't want the browser to open automatically after deployment, leave this box blank.



Click **Validate Connection** to verify that the settings are correct and you can connect to IIS on the local computer.

A green check mark verifies that the connection is successful.



Click **Next** to advance to the **Settings** tab.

The **Configuration** drop-down box specifies the build configuration to deploy. The default value is Release, which is what you want.

Leave the **Remove additional files at destination** check box cleared. Since this is your first deployment, there won't be any files in the destination folder yet.

In the **Databases** section, enter the following value in the connection string box for **SchoolContext**:

```
Data Source=|DataDirectory|School-Prod.sdf
```

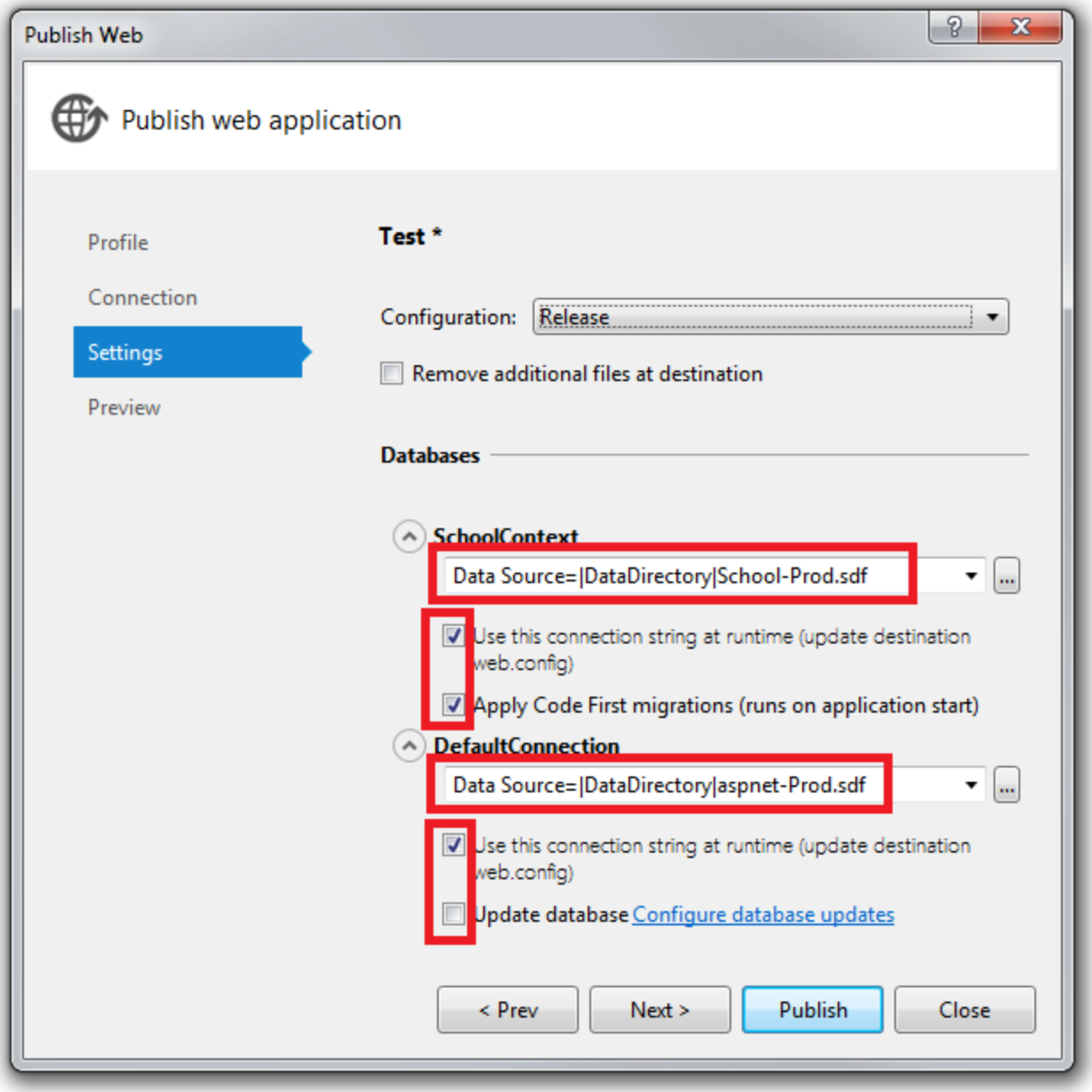
The deployment process will put this connection string in the deployed Web.config file because **Use this connection string at runtime** is selected.

Also under **SchoolContext**, select **Apply Code First Migrations**. This option causes the deployment process to configure the deployed Web.config file to specify the `MigrateDatabaseToLatestVersion` initializer. This initializer automatically updates the database to the latest version when the application accesses the database for the first time after deployment.

In the connection string box for **DefaultConnection**, enter the following value:

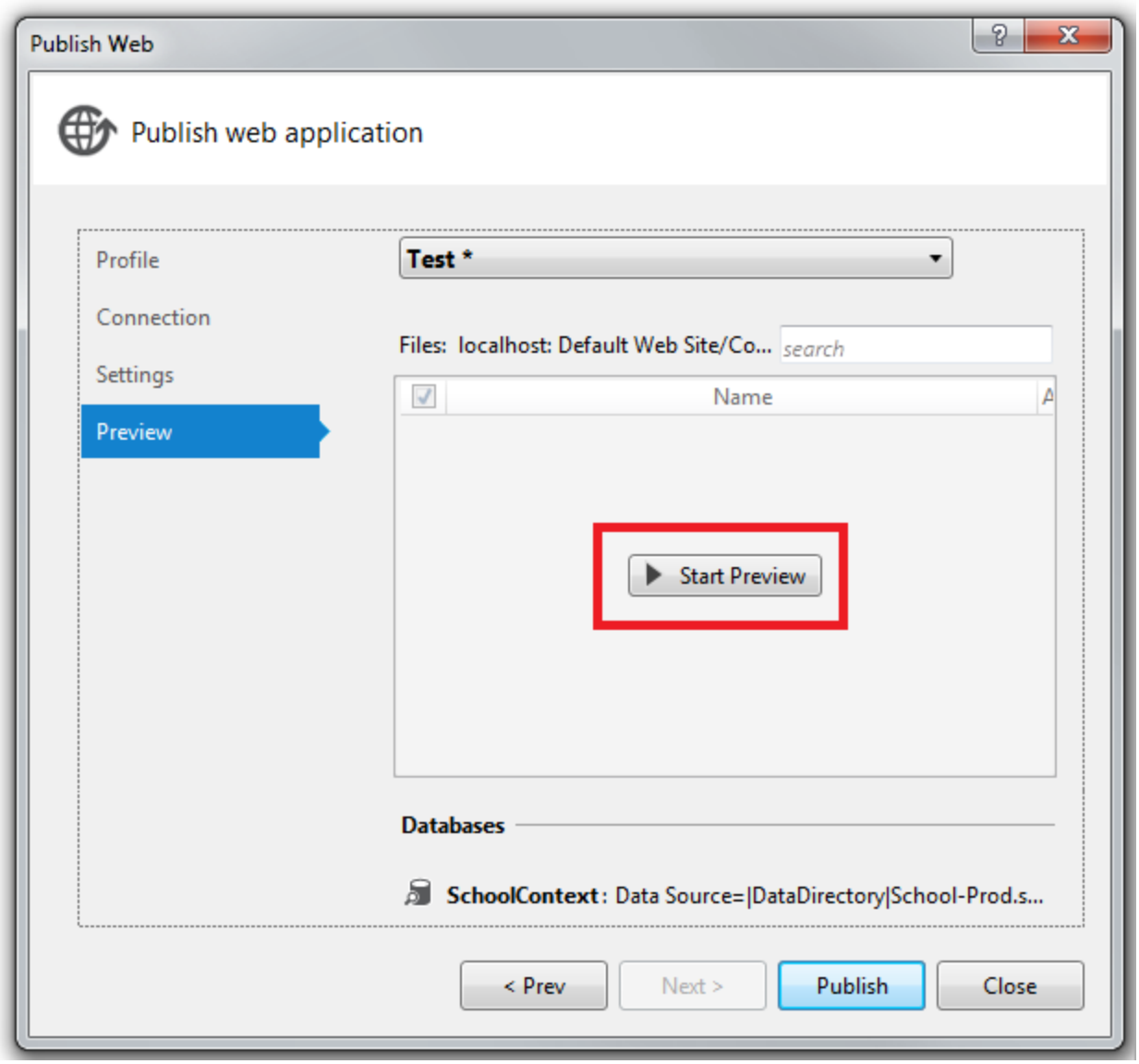
```
Data Source=|DataDirectory|aspnet-Prod.sdf
```

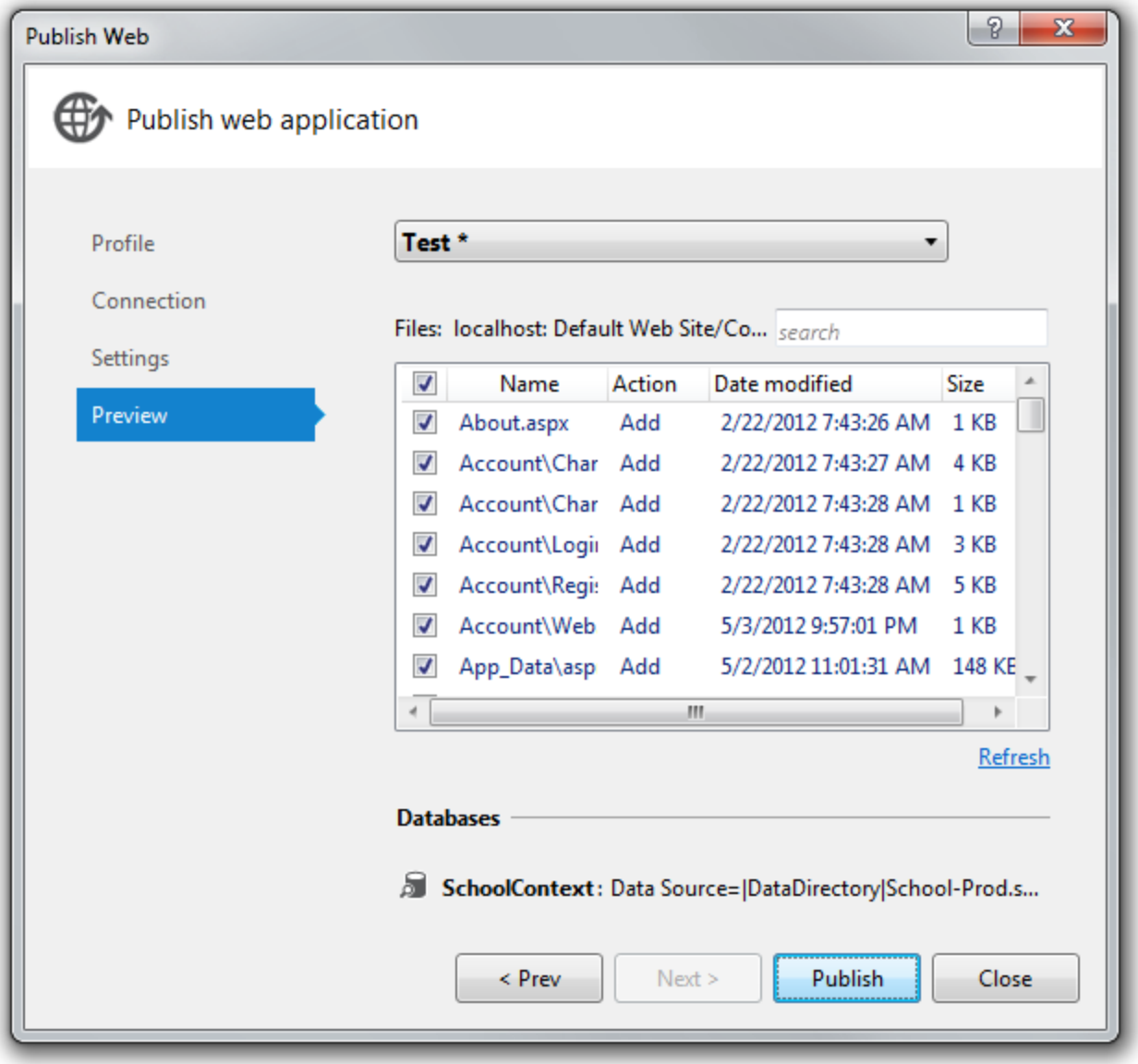
Leave **Update database** cleared. The membership database will be deployed by copying the .sdf file in App_Data, and you don't want the deployment process to do anything else with this database.



Click **Next** to advance to the **Preview** tab.

In the **Preview** tab, click **Start Preview** to see a list of the files that will be copied.

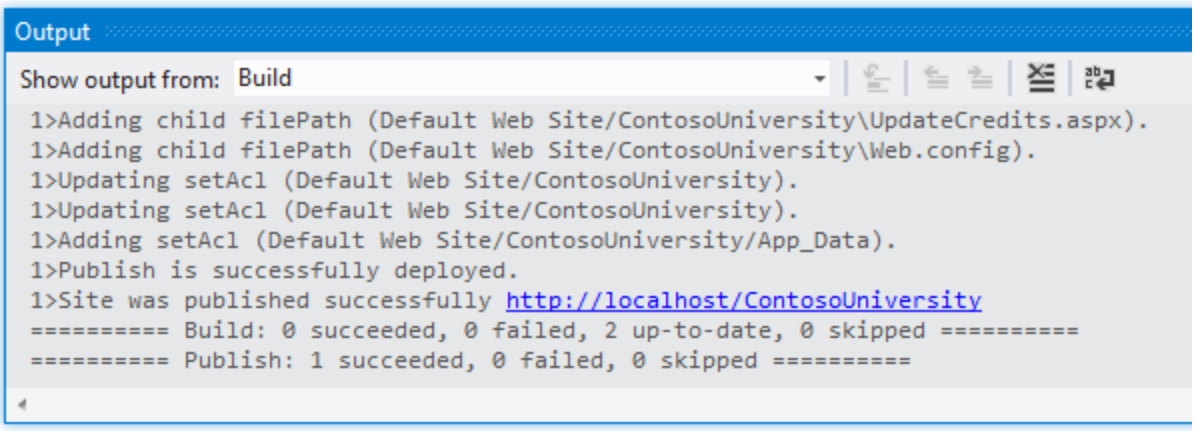

Publish Web
Publish web application
Profile
Connection
Settings
Preview
Test *
Files: localhost: Default Web Site/Co...
search
Name
Start Preview
Databases
SchoolContext: Data Source=|DataDirectory|School-Prod.s...
< Prev
Next >
Publish
Close



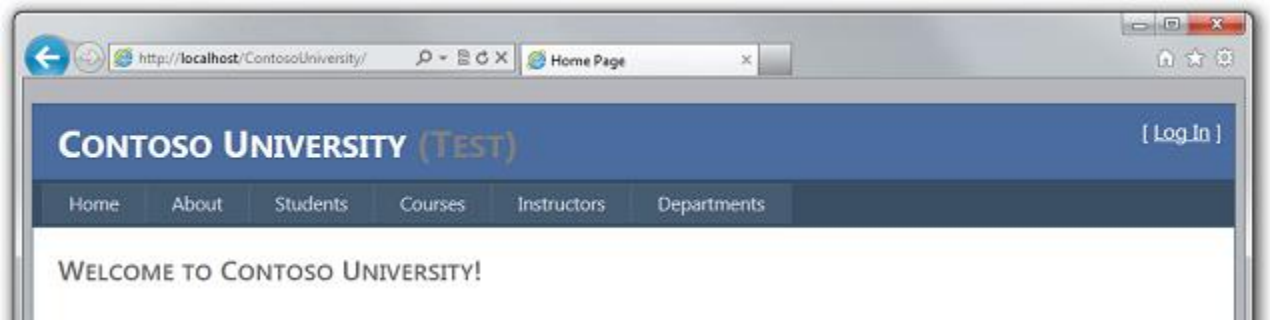
Click **Publish**.

If Visual Studio is not in administrator mode, you might get an error message that indicates a permissions error. In that case, close Visual Studio, open it in administrator mode, and try to publish again.

If Visual Studio is in administrator mode, the **Output** window reports successful build and publish.

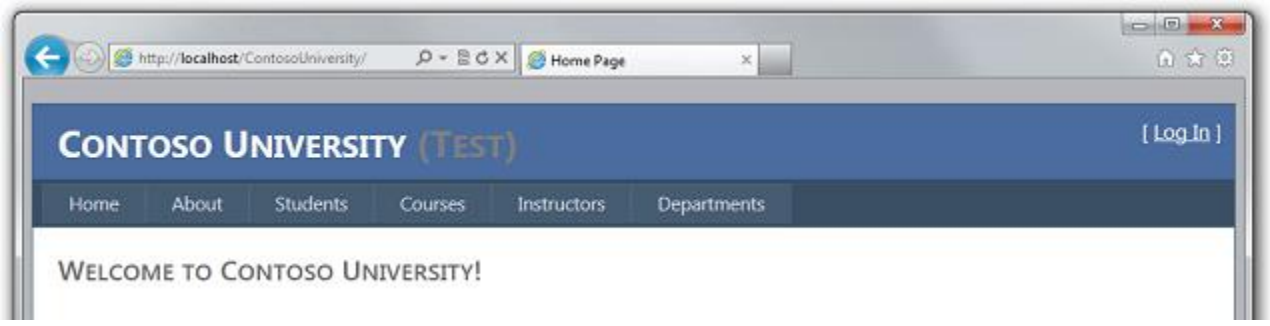


The browser automatically opens to the Contoso University Home page running in IIS on the local computer.

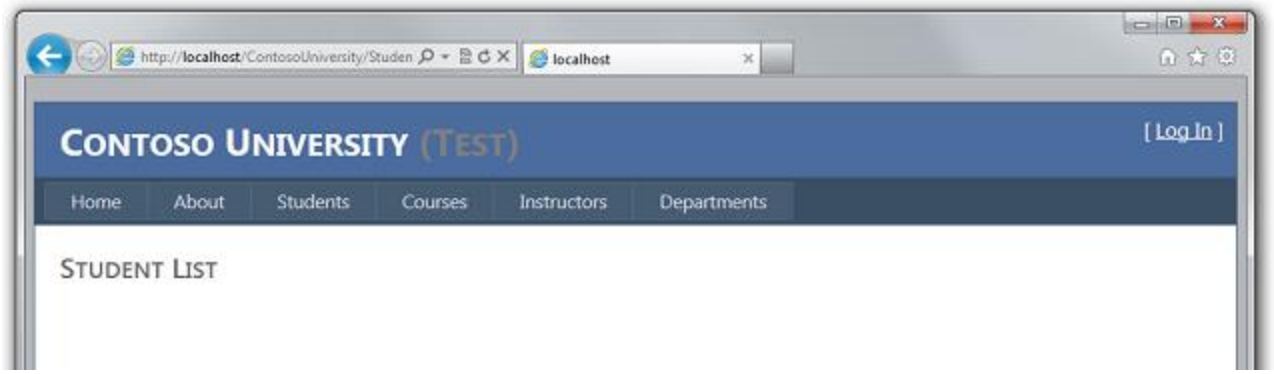


Testing in the Test Environment

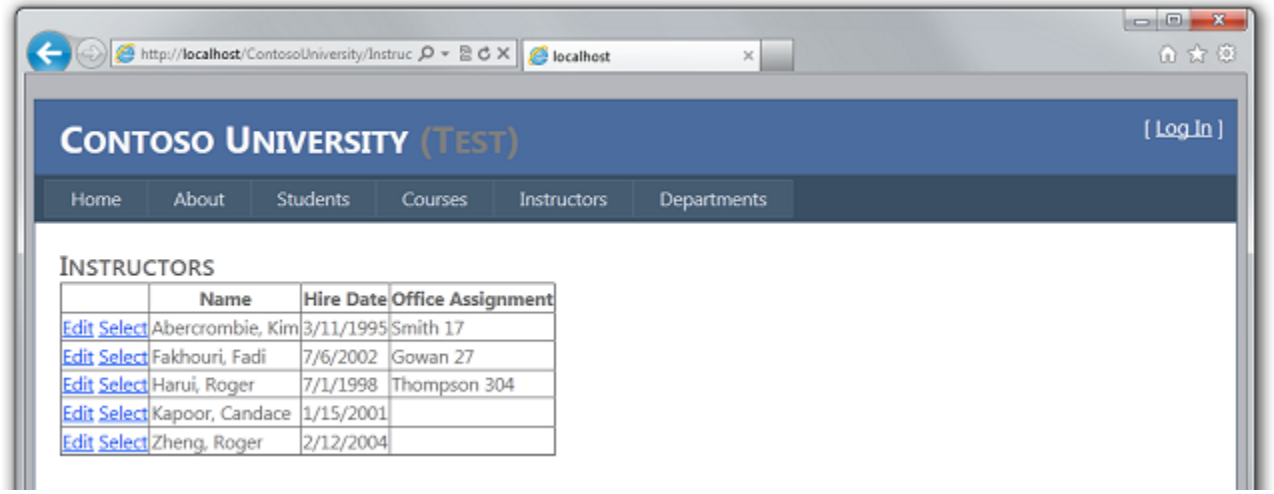
Notice that the environment indicator shows "(Test)" instead of "(Dev)", which shows that the *Web.config* transformation for the environment indicator was successful.



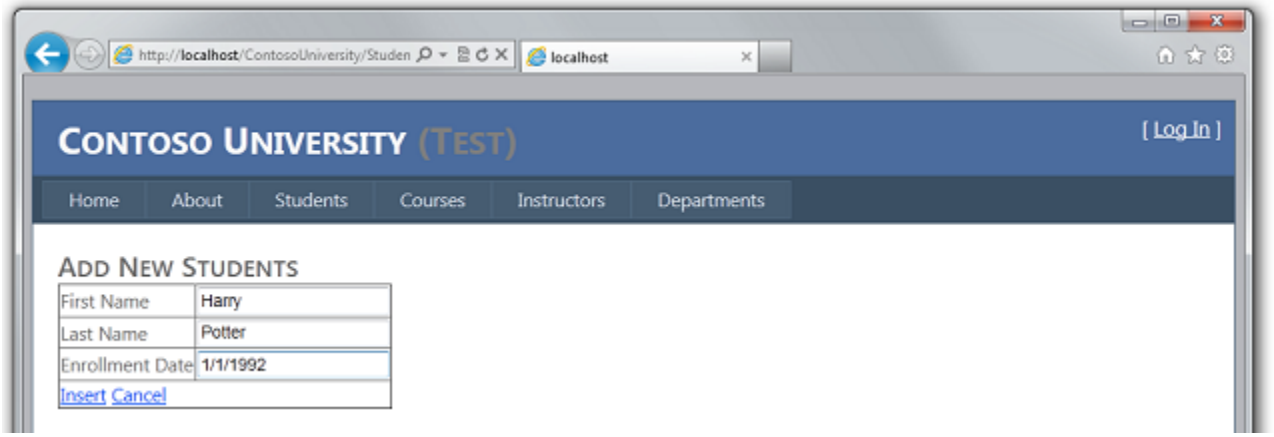
Run the **Students** page to verify that the deployed database has no students. When you select this page it may take a few minutes to load because Code First creates the database and then runs the `Seed` method. (It didn't do that when you were on the home page because the application didn't try to access the database yet.)

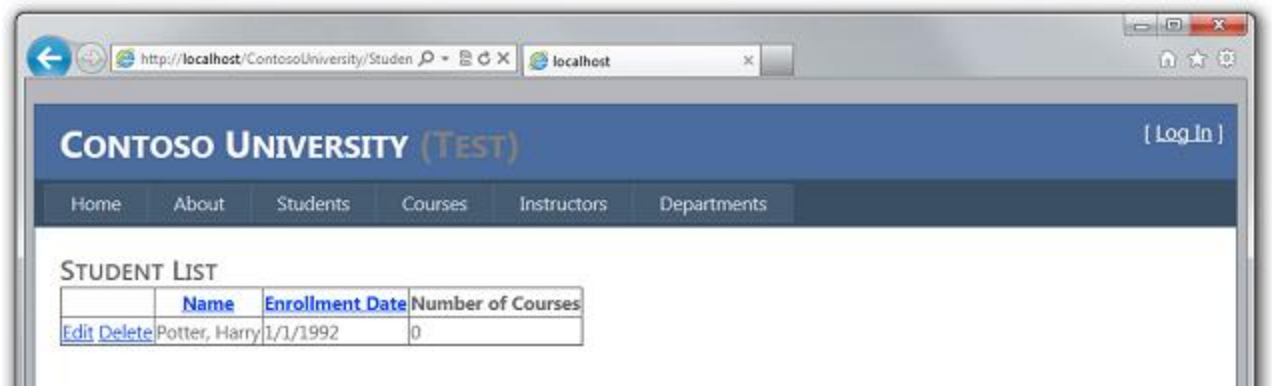


Run the **Instructors** page to verify that Code First seeded the database with instructor data:

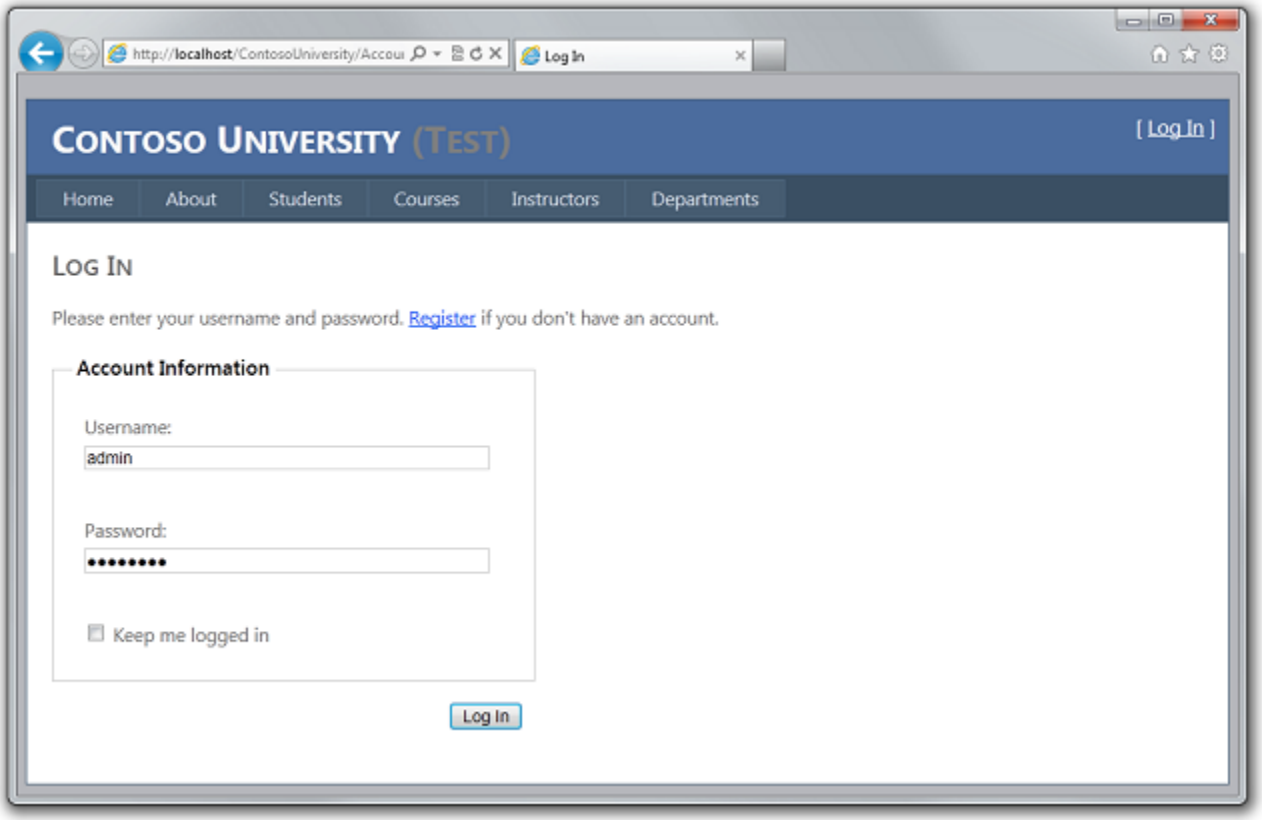


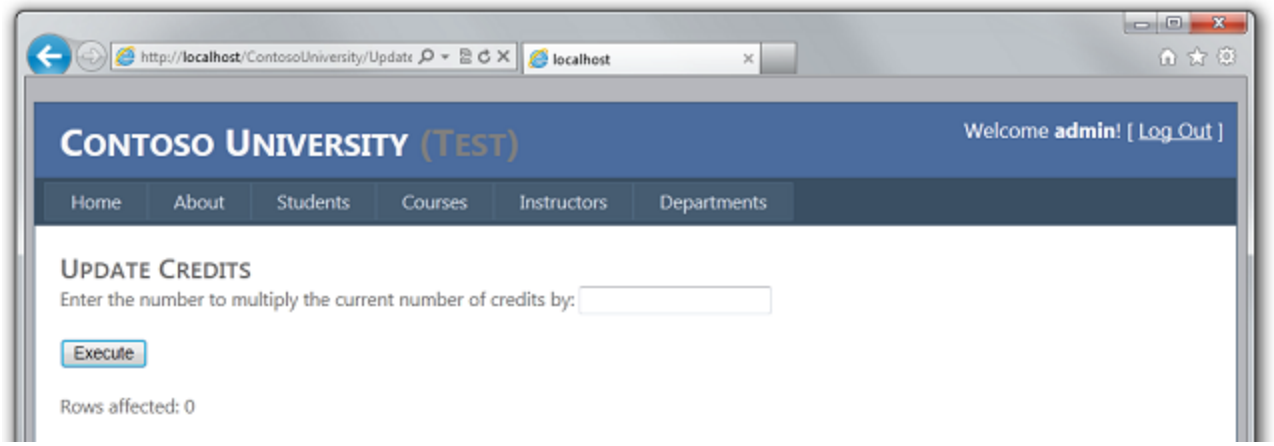
Select **Add Students** from the **Students** menu, add a student, and then view the new student in the **Students** page to verify that you can successfully write to the database:



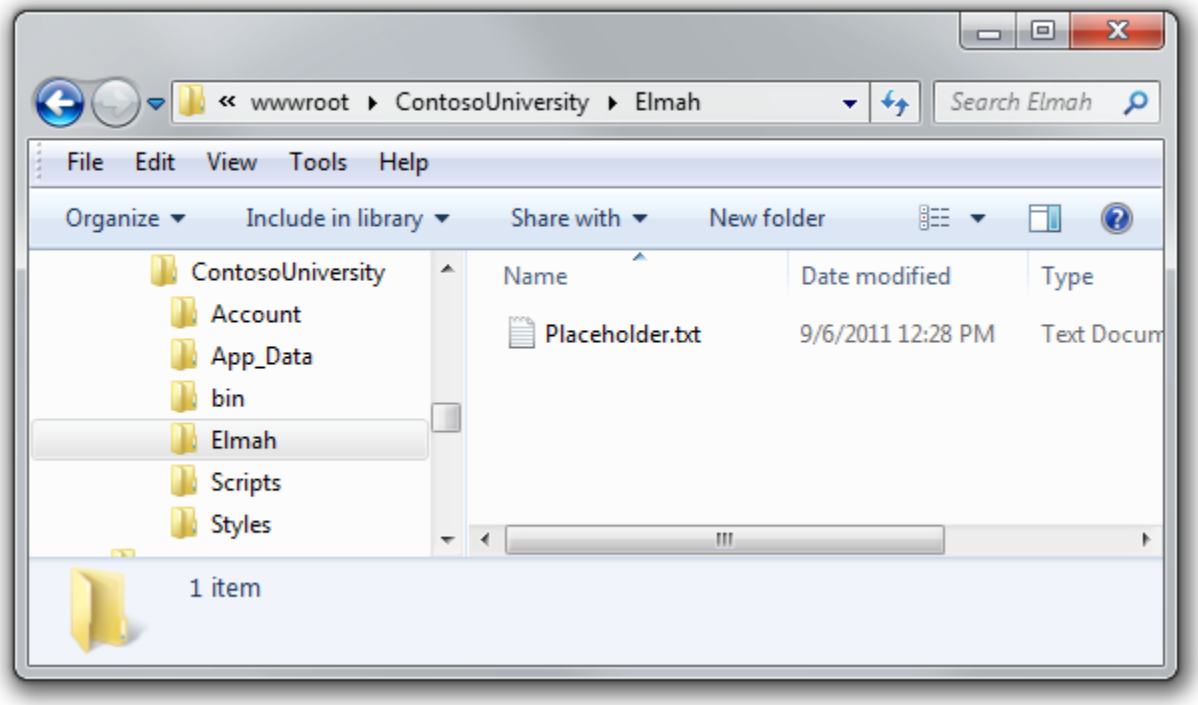


From the **Courses** menu, select **Update Credits**. The **Update Credits** page requires administrator permissions, so the **Log In** page is displayed. Enter the administrator account credentials that you created earlier ("admin" and "Pas$w0rd"). The **Update Credits** page is displayed, which verifies that the administrator account that you created in the previous tutorial was correctly deployed to the test environment.





Verify that an *Elmah* folder exists with only the placeholder file in it.



Reviewing the Automatic Web.config Changes for Code First Migrations

Open the *Web.config* file in the deployed application at *C:\inetpub\wwwroot\ContosoUniversity* and you can see where the deployment process configured Code First Migrations to automatically update the database to the latest version.

```
<entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlCeConnectionFactor
        <parameters>
            <parameter value="System.Data.SqlServerCe.4.0" />
        </parameters>
    </defaultConnectionFactory>
    <contexts>
        <context type="ContosoUniversity.DAL.SchoolContext, ContosoUniversity.DAL">
            <databaseInitializer type="System.Data.Entity.MigrateDatabaseToLatestVersion`2[
                <parameters>
                    <parameter value="SchoolContext_DatabasePublish" />
                </parameters>
            </databaseInitializer>
        </context>
    </contexts>
</entityFramework>
```

The deployment process also created a new connection string for Code First Migrations to use exclusively for updating the database schema:

```
<connectionStrings>
  <add name="DefaultConnection" connectionString="|DataDirectory|aspnet.sdf" providerName=
  <add name="SchoolContext" connectionString="Data Source=|DataDirectory|School.sdf" provi
  <add name="SchoolContext_DatabasePublish" connectionString="Data Source=|DataDirectory|S
</connectionStrings>
```

This additional connection string enables you to specify one user account for database schema updates, and a different user account for application data access. For example, you could assign the db_owner role to Code First Migrations, and db_datareader and db_datawriter roles to the application. This is a common defense-in-depth pattern that prevents potentially malicious code in the application from changing the database schema. (For example, this might happen in a successful SQL injection attack.) This pattern is not used by these tutorials. It does not apply to SQL Server Compact, and it does not apply when you migrate to SQL Server in a later tutorial in this series. The Cytanium site offers just one user account for accessing the SQL Server database that you create at Cytanium. If you are able to implement this pattern in your scenario, you can do it by performing the following steps:

1. In the **Settings** tab of the **Publish Web** wizard, enter the connection string that specifies a user with full database schema update permissions, and clear the **Use this connection string at runtime** check box. In the deployed Web.config file, this becomes the `DatabasePublish` connection string.
2. Create a Web.config file transformation for the connection string that you want the application to use at run time.

You have now deployed your application to IIS on your development computer and tested it there. This verifies that the deployment process copied the application's content to the right location (excluding the files that you did not want to deploy), and also that Web Deploy configured IIS correctly during deployment. In the next tutorial, you'll run one more test that finds a deployment task that has not yet been done: setting folder permissions on the *Elmah* folder.

More Information

For information about running IIS or IIS Express in Visual Studio, see the following resources:

- IIS Express Overview on the IIS.net site.
- Introducing IIS Express on Scott Guthrie's blog.
- How to: Specify the Web Server for Web Projects in Visual Studio.
- Core Differences Between IIS and the ASP.NET Development Server on the ASP.NET site.
- Test your ASP.NET MVC or Web Forms Application on IIS 7 in 30 seconds on Rick Anderson's blog. This entry provides examples of why testing with the Visual Studio Development Server (Cassini) is not as reliable as testing in IIS Express, and why testing in IIS Express is not as reliable as testing in IIS.

For information about what issues might arise when your application runs in medium trust, see Hosting ASP.NET Applications in Medium Trust on the 4 Guys from Rolla site.

Setting Folder Permissions - 6 of 12

Overview

In this tutorial, you set folder permissions for the *Elmah* folder in the deployed web site so that the application can create log files in that folder.

When you test a web application in Visual Studio using the Visual Studio Development Server (Cassini), the application runs under your identity. You are most likely an administrator on your development computer and have full authority to do anything to any file in any folder. But when an application runs under IIS, it runs under the identity defined for the application pool that the site is assigned to. This is typically a system-defined account that has limited permissions. By default it has read and execute permissions on your web application's files and folders, but it doesn't have write access.

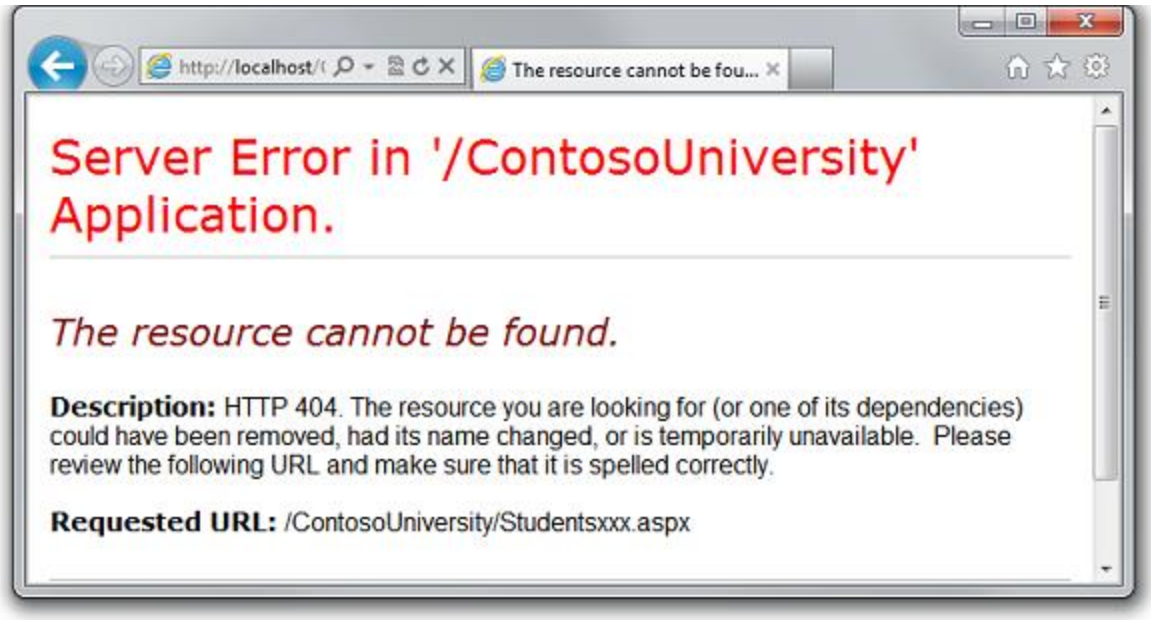
This becomes an issue if your application creates or updates files, which is a common need in web applications. In the Contoso University application, Elmah creates XML files in the *Elmah* folder in order to save details about errors. Even if you don't use something like Elmah, your site might let users upload files or perform other tasks that write data to a folder in your site.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

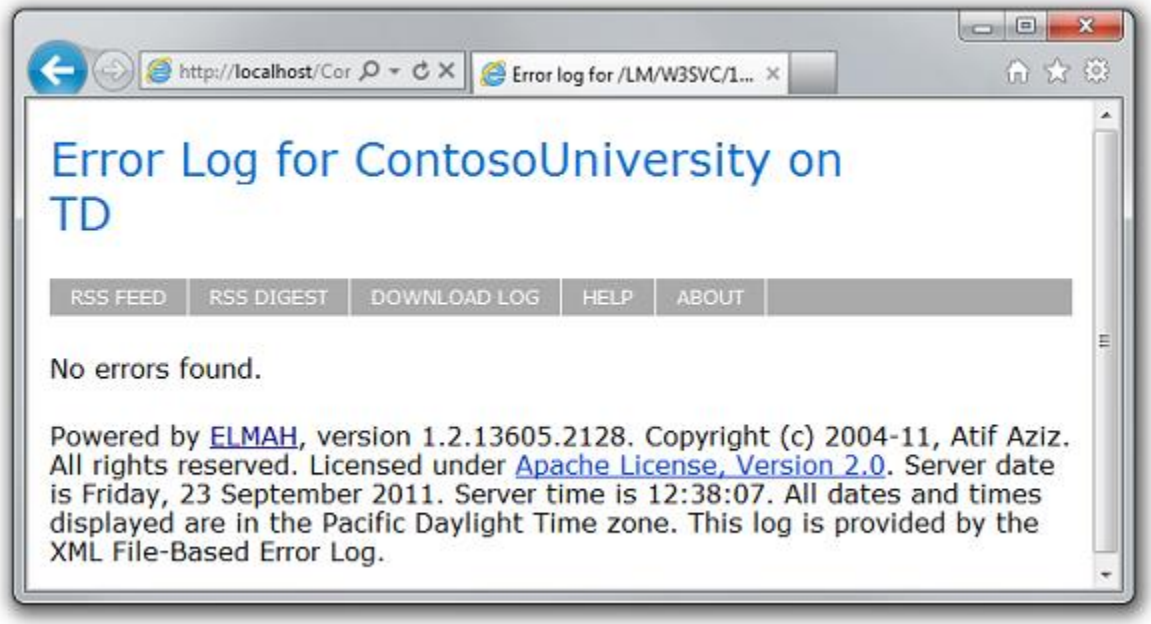
Testing Error Logging and Reporting

To see how the application doesn't work correctly in IIS (although it did when you tested it in Visual Studio), you can cause an error that would normally be logged by Elmah, and then open the Elmah error log to see the details. If Elmah was unable to create an XML file and store the error details, you see an empty error report.

Open a browser and go to *http://localhost/ContosoUniversity*, and then request an invalid URL like *Studentsxxx.aspx*. You see a system-generated error page instead of the *GenericErrorPage.aspx* page because the `customErrors` setting in the Web.config file is "RemoteOnly" and you are running IIS locally:



Now run *Elmah.axd* to see the error report. You see an empty error log page because Elmah was unable to create an XML file in the *Elmah* folder:

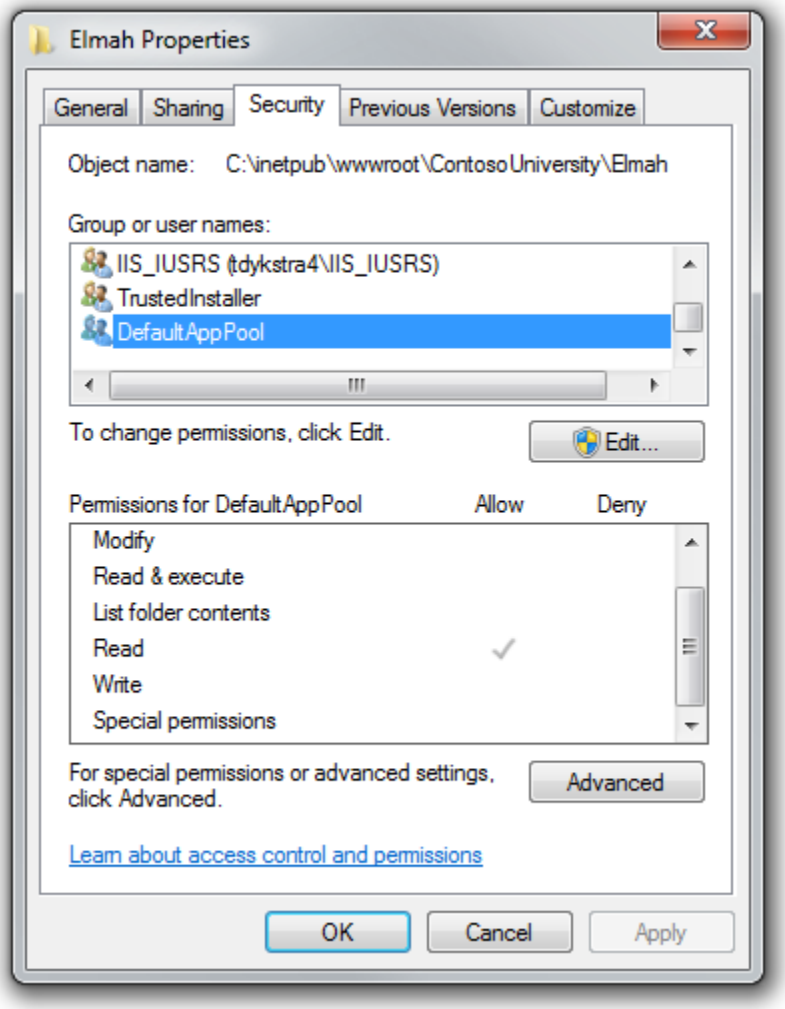


Setting Write Permission on the Elmah Folder

You can set folder permissions manually or you can make it an automatic part of the deployment process. Making it automatic requires complex MSBuild code, and since you only have to do this the first time you

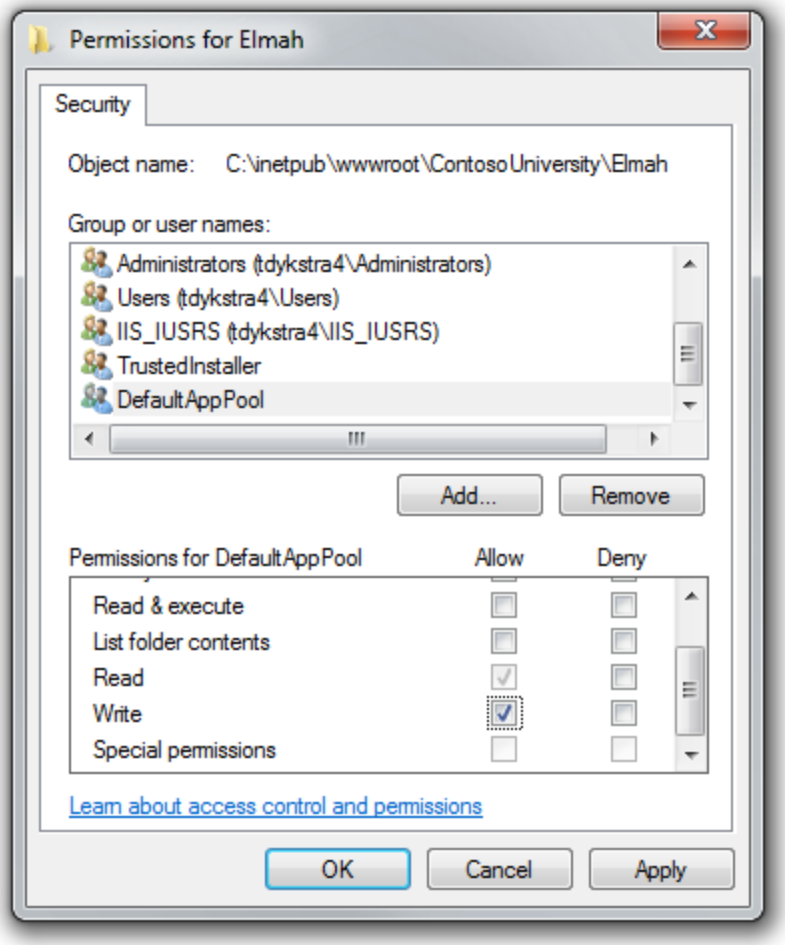
deploy, this tutorial only shows how to do it manually. (For information about how to make this part of the deployment process, see Setting Folder Permissions on Web Publish on Sayed Hashimi's blog.)

In **Windows Explorer**, navigate to *C:\inetpub\wwwroot\ContosoUniversity*. Right-click the *Elmah* folder, select **Properties**, and then select the **Security** tab.



(If you don't see **DefaultAppPool** in the **Group or user names** list, you probably used some other method than the one specified in this tutorial to set up IIS and ASP.NET 4 on your computer. In that case, find out what identity is used by the application pool assigned to the Contoso University application, and grant write permission to that identity. See the links about application pool identities at the end of this tutorial.)

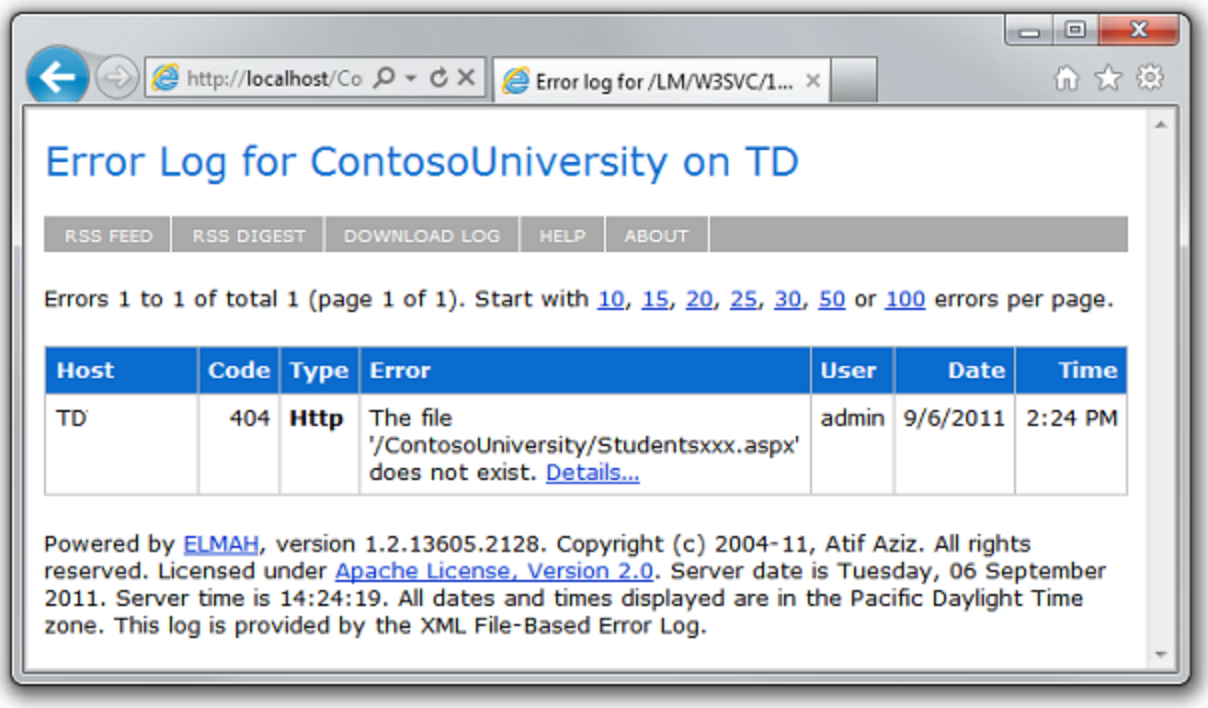
Click **Edit**. In the **Permissions for Elmah** dialog box, select **DefaultAppPool**, and then select the **Write** check box in the **Allow** column.



Click **OK** in both dialog boxes.

Retesting Error Logging and Reporting

Test by causing an error again in the same way (request a bad URL) and run the **Error Log** page. This time the error appears on the page.



You also need write permission on the *App_Data* folder because you have SQL Server Compact database files in that folder, and you want to be able to update data in those databases. In that case, however, you don't have to do anything extra because the deployment process automatically sets write permission on the *App_Data* folder.

You have now completed all of the tasks necessary to get Contoso University working correctly in IIS on your local computer. In the next tutorial, you will make the site publicly available by deploying it to a hosting provider.

More Information

In this example, the reason why Elmah was unable to save log files was fairly obvious. You can use IIS tracing in cases where the cause of the problem is not so obvious; see Troubleshooting Failed Requests Using Tracing in IIS 7 on the IIS.net site.

For more information about how to grant permissions to application pool identities, see Application Pool Identities and Secure Content in IIS Through File System ACLs on the IIS.net site.

Deploying to the Production Environment - 7 of 12

Overview

In this tutorial, you set up an account with a hosting provider and deploy your ASP.NET web application to the production environment by using the Visual Studio one-click publish feature.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Selecting a Hosting Provider

For the Contoso University application and this tutorial series, you need a provider that supports ASP.NET 4 and Web Deploy. A specific hosting company was chosen so that the tutorials could illustrate the complete experience of deploying to a live website. Each hosting company provides different features, and the experience of deploying to their servers varies somewhat. However, the process described in this tutorial is typical for the overall process. The hosting provider used for this tutorial, Cytanium.com, is one of many that are available, and its use in this tutorial does not constitute an endorsement or recommendation.

When you are ready to select your own hosting provider, you can compare features and prices in the gallery of providers on the Microsoft.com/web site.

Creating an Account

Create an account at your selected provider. If support for a full SQL Server database is an added extra, you do not need to select it for this tutorial, but you'll need it for the Migrating to SQL Server tutorial later in this series.

For these tutorials, you don't have to register a new domain name. You can test to verify successful deployment by using the temporary URL assigned to the site by the provider.

After the account has been created, you typically receive a welcome email that contains all the information you need in order to deploy and manage your site. The information that your hosting provider sends you will be similar to what is shown here. The Cytanium welcome email that's sent to new account owners includes the following information:

- The URL to the provider's control panel site, where you can manage settings for your site. The ID and password you specified are included in this part of the welcome email for easy reference. (Both have been changed to a demo value for this illustration.)

Control Panel URL

Control Panel URL	Username	Password
http://panel.cytanium.com	contoso	contoso

- The default .NET Framework version and information about how to change it. Many hosting sites default to 2.0, which works with ASP.NET applications that target the .NET Framework 2.0, 3.0, or 3.5. However Contoso University is a .NET Framework 4 application, so you have to change this setting. (For an ASP.NET 4.5 application you would use the .NET 4.0 setting.)

Web

Web Settings

This account is setup for ASP.NET 2.0 (3.5 SP1) in IIS 7 Integrated mode.
Instructions on how to change framework version can be found here.

Need help on selecting the correct .NET version? See guidance below:

- **.NET 3.5 – works with Web Sites built using "Site from Gallery" option (e.g. – DotNetNuke, Umbraco, etc...)**
- **.NET 4.0 – works with applications built using "Site from Template" option (e.g. – Starter Site, Bakery, etc...)**

- The temporary URL that you can use to access your web site. When this account was created, "contosouniversity.com" was entered as the existing domain name. Therefore the temporary URL is `http://contosouniversity.com.vserver01.cytanium.com`.

Temporary URL

You can access your web sites right now using their respective temporary URLs (instant aliases). Temporary URL is a sub-domain of the form http://<yourdomain.com>.vserver01.cytanium.com where <yourdomain.com> is your domain.

- Information about how to set up databases, and the connection strings that you need in order to access them:

Databases

SQL CE databases are automatically available on any hosting plan. You can create/upload any number of SQL CE databases from File Manager in Control Panel.

If you purchased a MySQL or MS SQL database, you can create the database via the Control Panel. Further instructions can be found here.

After creating a SQL Server or MySQL database you can access it using one of the following methods:

- **Database Manager using IIS Manager**
- **Remotely via SQL Management Stuido or MySQL Workbench**
- **directly from code**

We've provided two example connection strings below:

- **SQL Server: Data Source=vserver01.cytanium.com;Initial Catalog={myDataBase};User Id= {myUsername};Password={myPassword};**
- **MySQL: Server=vserver01.cytanium.com;Database= {myDataBase};Uid={myUsername};Pwd= {myPassword};**

- Information about tools and settings for deploying your site. (The email from Cytanium also mentions WebMatrix, which is omitted here.)

Deploy and Manage Your Site

You can deploy and manage your site using:

- **WebMatrix**
- **Visual Studio 2010**
- **IIS 7 Manager 32-bit**
- **IIS 7 Manager 64-bit**
- **WebDeploy (MSDeploy)**
- **Cytanium Control Panel**
- **FTP**

Following are field instructions for WebMatrix and Visual Studio 2010:

Visual Studio 2010
Publish method: Web Deploy
Service URL: https://vserver01.cytanium.com:8172/MsDeploy.axd
Site/application: <yourdomain.com>
Allow untrusted certificate (check)
User name: contoso
Password: contoso

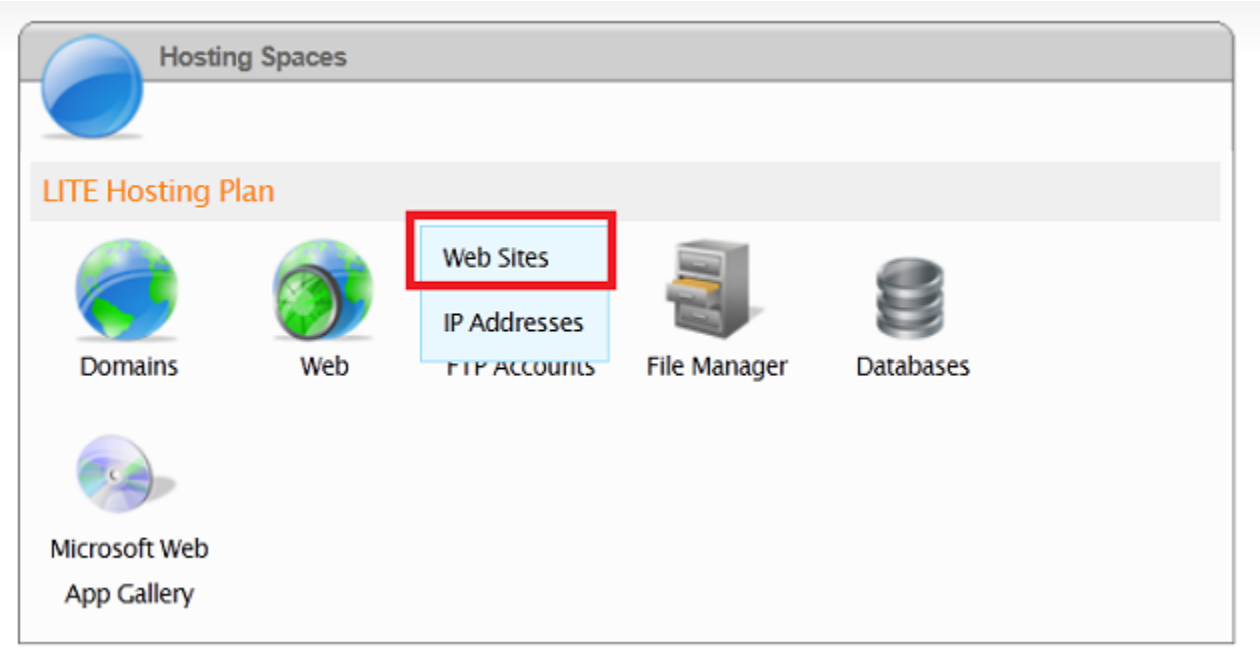
FTP Standalone client (FileZilla example)
Host: vserver01.cytanium.com
Server Type: FTPES - FTP over explicit TLS/SSL
Logon Type: Normal
User: contoso
Password: contoso
Transfer Settings tab: Set Transfer mode to Passive

Setting the .NET Framework Version

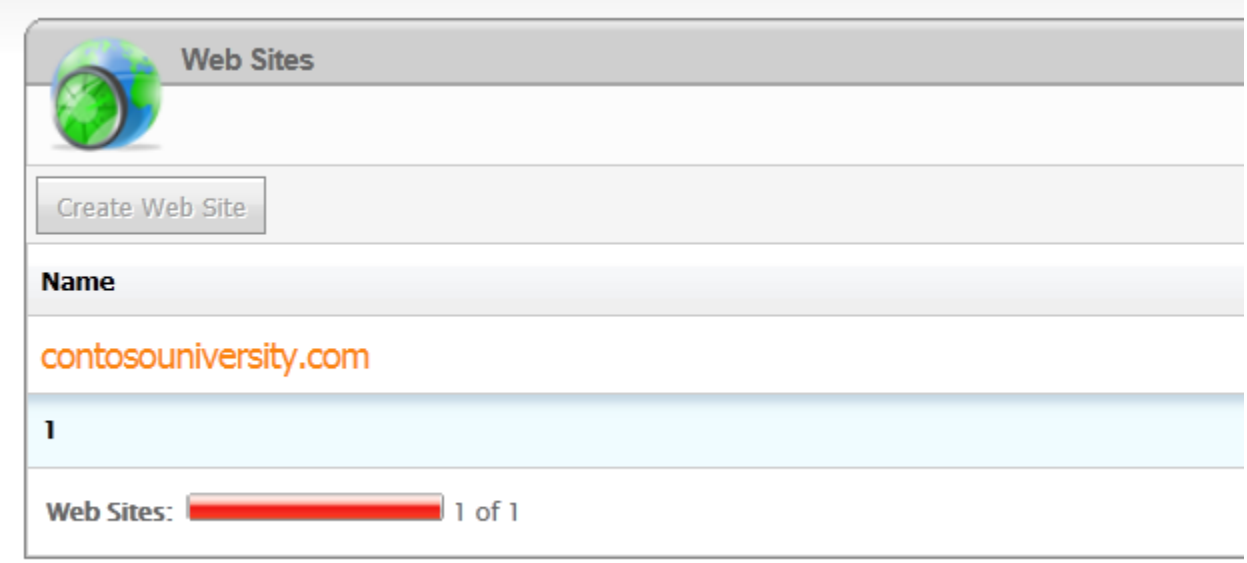
The Cytanium welcome email includes a link to instructions on how to change the version of the .NET Framework. These instructions explain that this can be done through the Cytanium control panel. Other providers have control panel sites that look different, or they may instruct you to do this in a different way.

Go to the control panel URL. After logging in with your user name and password, you see the control panel.

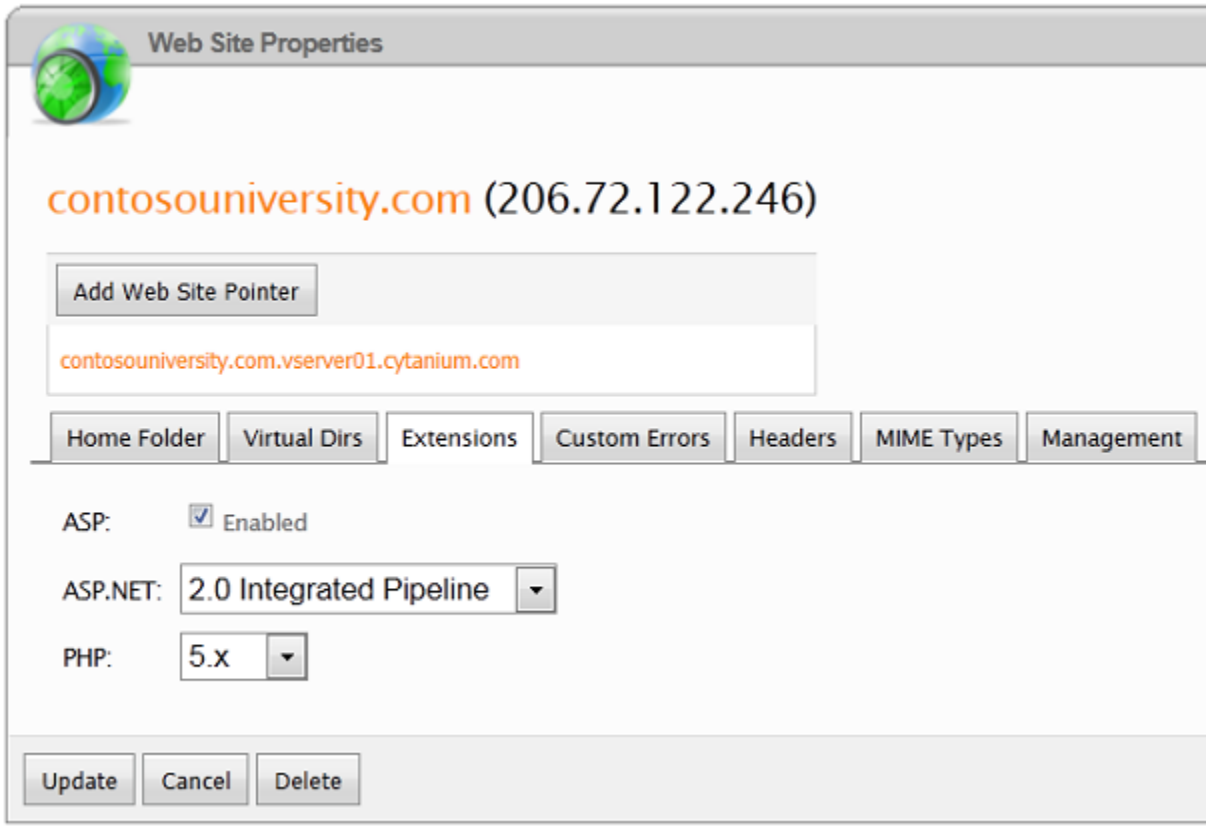
In the **Hosting Spaces** box, hold the pointer over the Web icon and select **Web Sites** from the menu.



In the **Web Sites** box, click **contosouniversity.com** (the name of the site that you used when you created the account).



In the **Web Site Properties** box, select the **Extensions** tab.

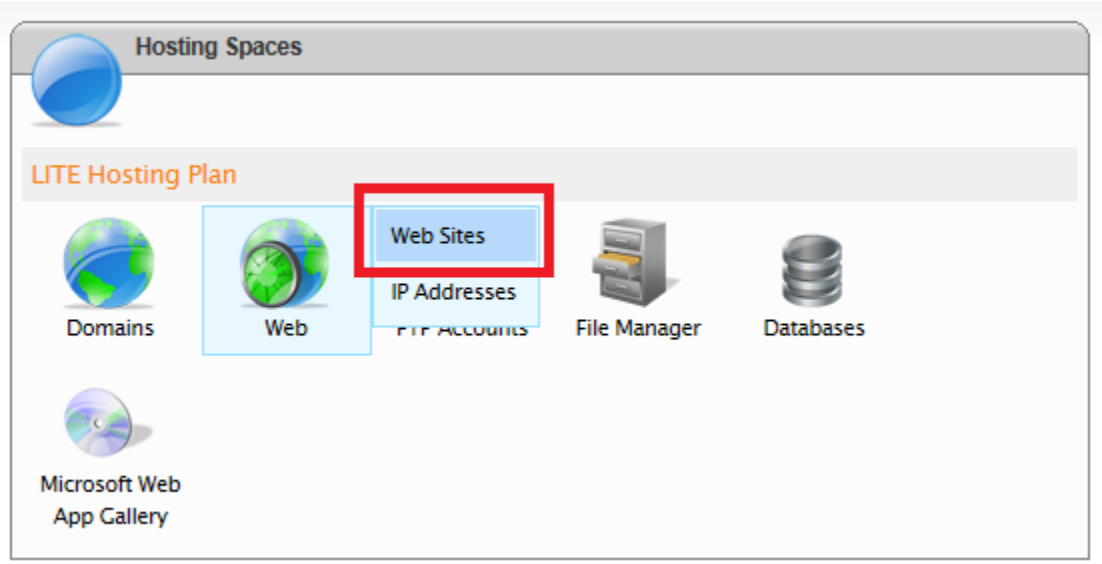


Change ASP.NET from **2.0 Integrated Pipeline** to **4.0 (Integrated Pipeline)**, and then click **Update**.

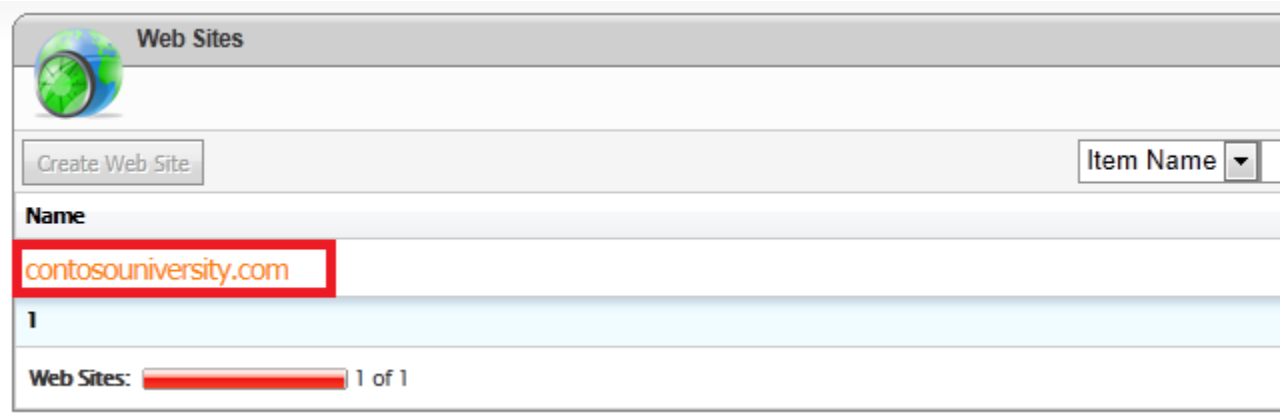
Publishing to the Hosting Provider

The welcome email from the hosting provider includes all of the settings you need in order to publish the project, and you can enter that information manually into a publish profile. But you'll use an easier and less error-prone method to configure deployment to the provider: you'll download a *.publishsettings* file and import it into a publish profile.

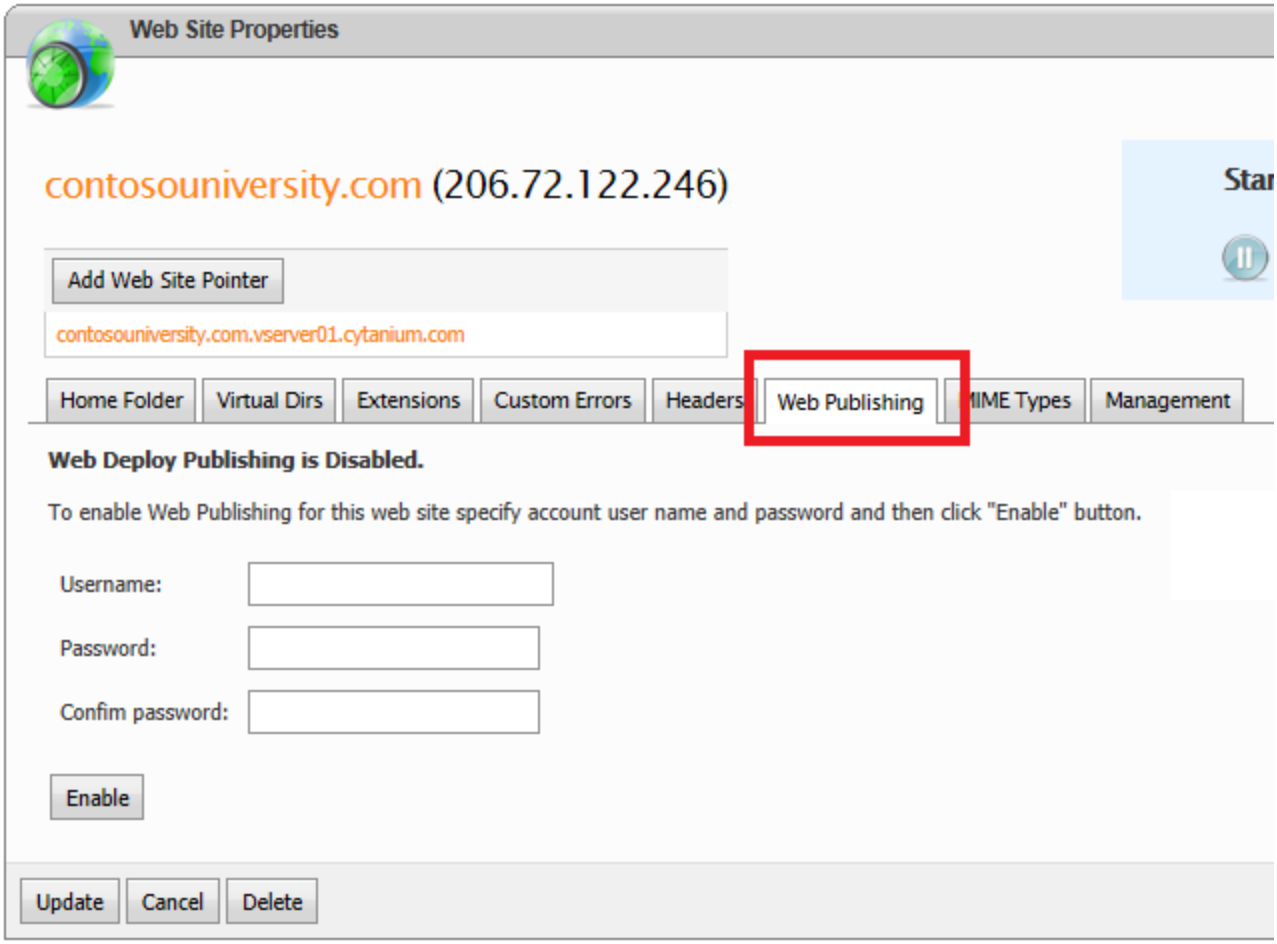
In your browser, go to the Cytanium control panel and select **Web** and then select **Web Sites.**



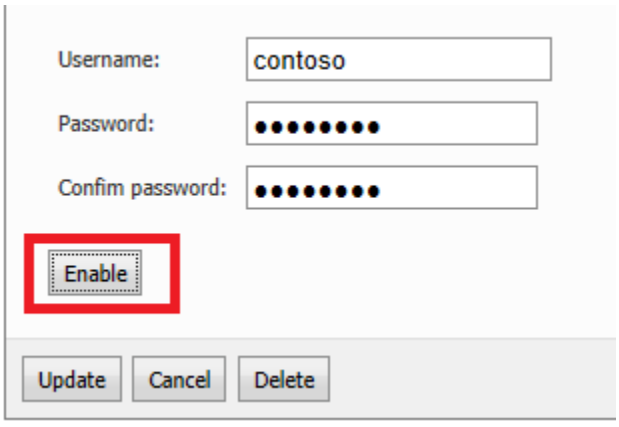
Select the **contosouniversity.com** web site.



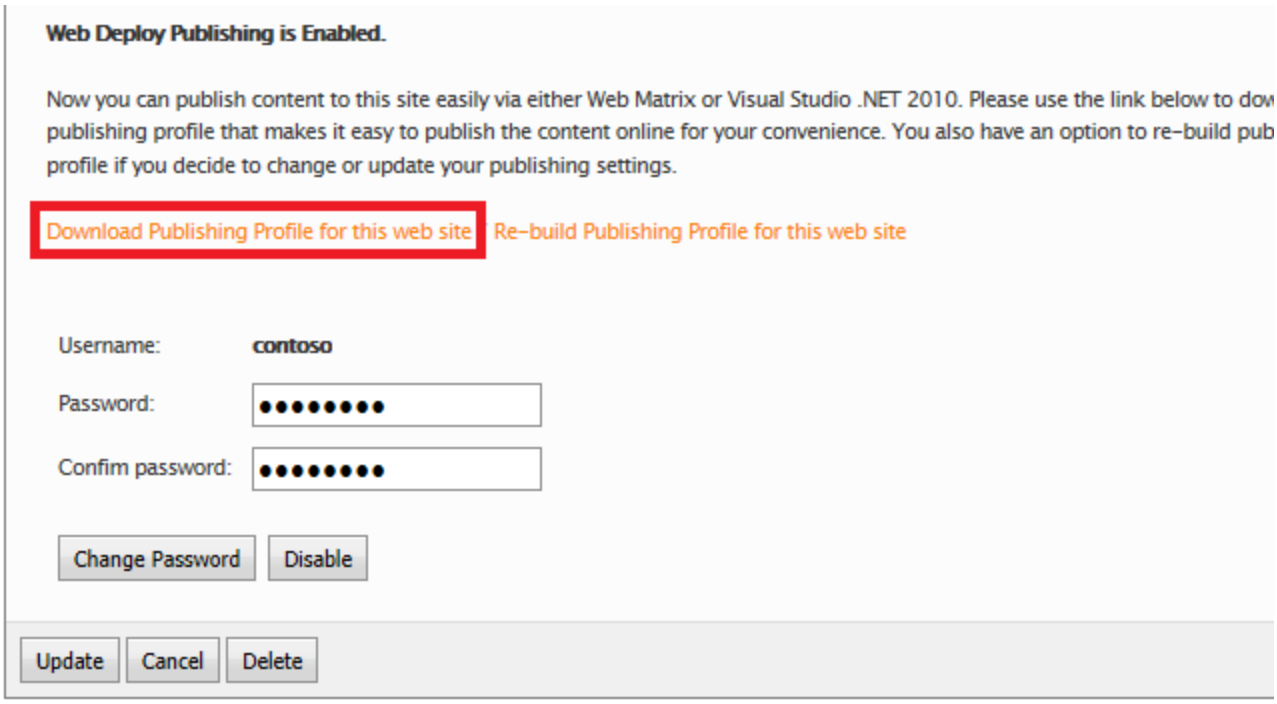
Select the **Web Publishing** tab.



Create credentials to use for web publishing by entering a user name and password. You can enter the same credentials that you use to log on to the control panel. Then click **Enable**.



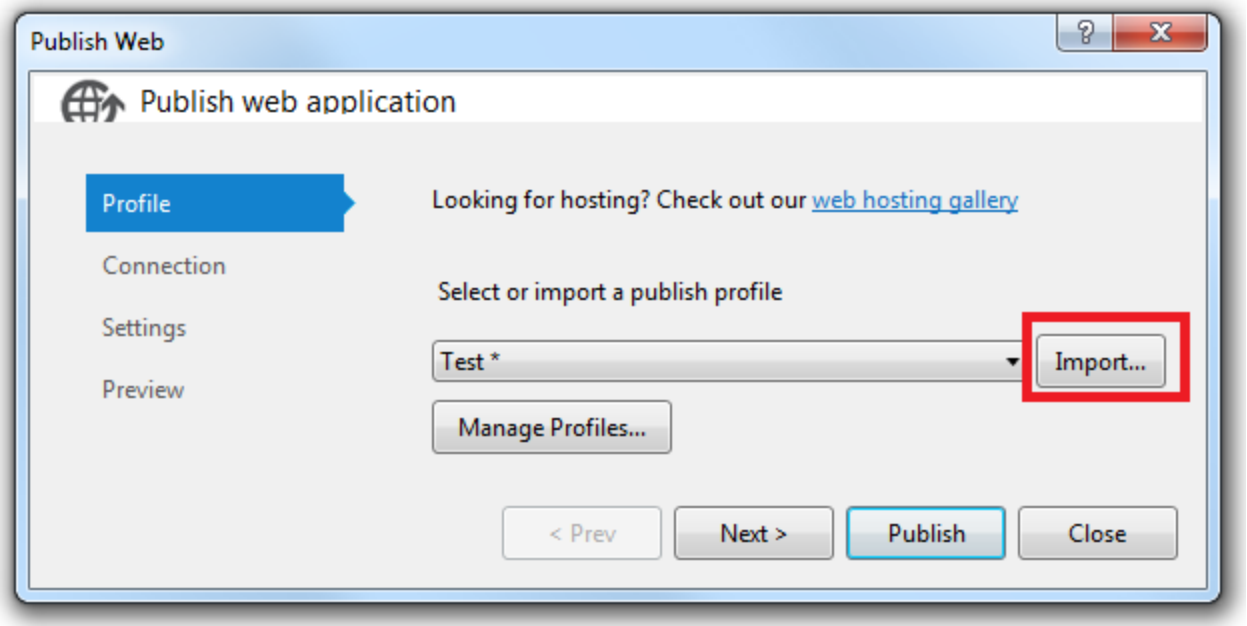
Click **Download Publishing Profile for this web site**.



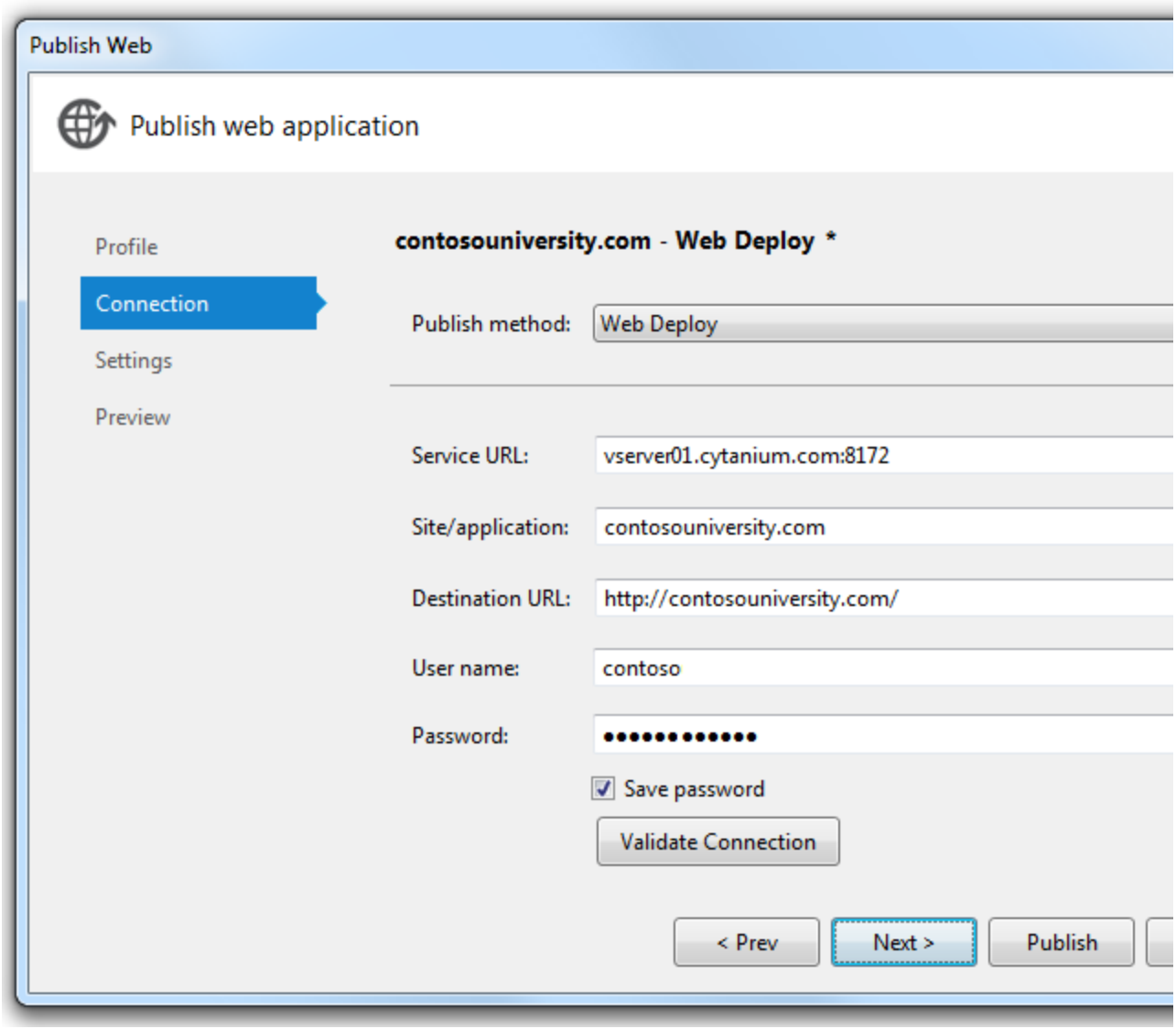
When you are prompted to open or save the file, save it.

In **Solution Explorer** in Visual Studio, right-click the ContosoUniversity project and select **Publish**. The **Publish Web** dialog box opens on the **Preview** tab with the **Test** profile selected because that is the last profile you used.

Select the **Profile** tab and then Click **Import**.



In the **Import Publish Settings** dialog box, select the *.publishsettings* file that you downloaded, and click **Open**. The wizard advances to the Connection tab with all of the fields filled in.

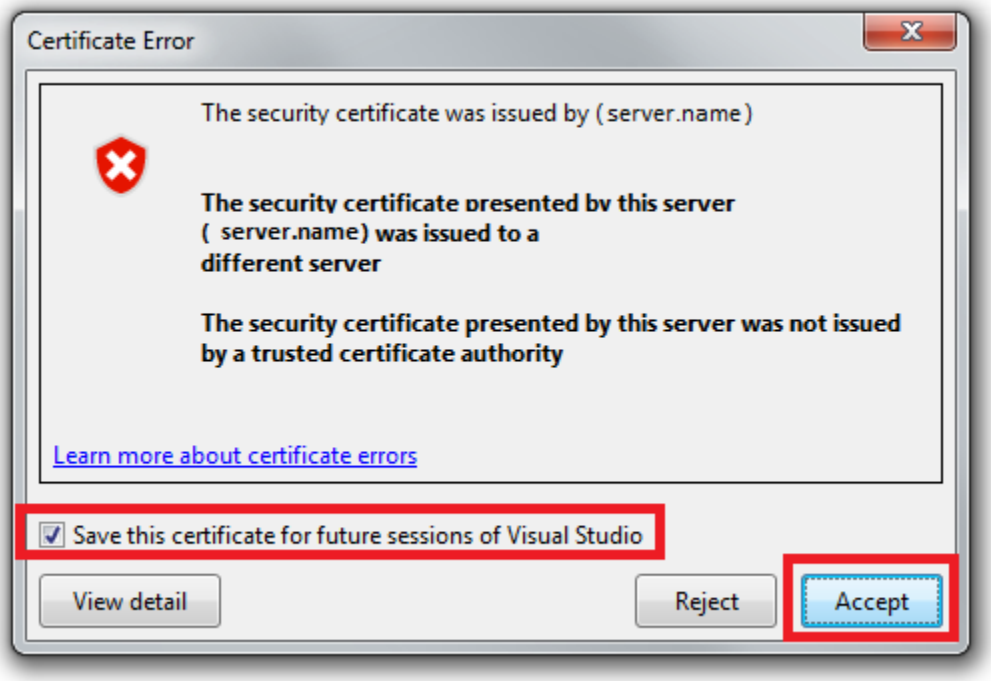


The .publishsettings file puts the planned permanent URL for the site in the Destination URL box, but if you haven't purchased that domain yet, replace the value with the temporary URL. For this example, the URL is *http://contosouniversity.com.vserver01.cytanium.com*. The only purpose of this box is to specify what URL the browser will open to automatically after successfully after deployment. If you leave it blank, the only consequence is that the browser won't start automatically after deployment.

Click **Validate Connection** to verify that the settings are correct and you can connect to the server. As you saw earlier, a green check mark verifies that the connection is successful.

When you click Validate Connection, you might see a **Certificate Error** dialog box. If you do, verify that the server name is what you expect. If it is, select **Save this certificate for future sessions of Visual Studio** and click **Accept**. (This error means that the hosting provider has chosen to avoid the expense of purchasing an SSL

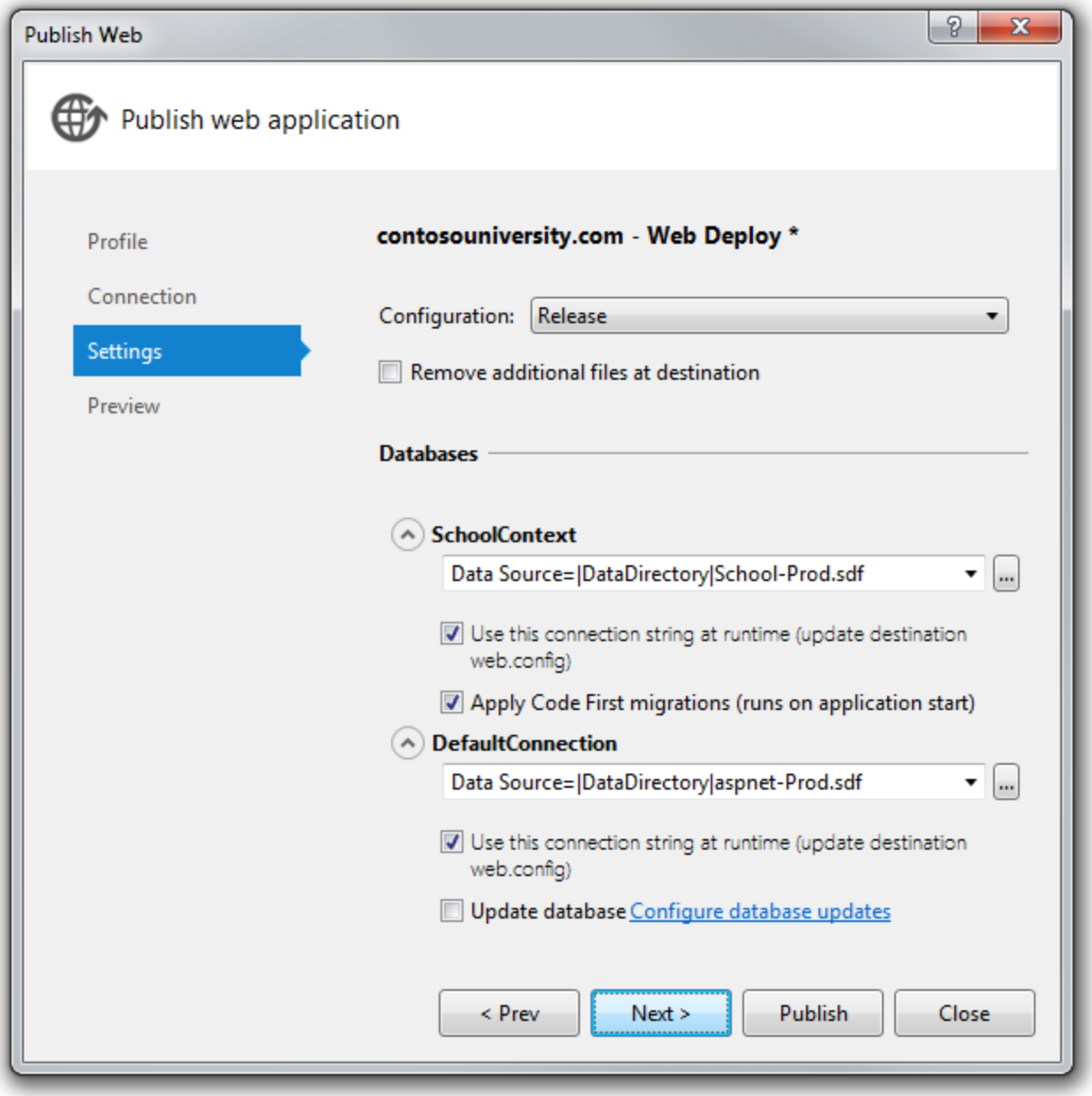
certificate for the URL that you are deploying to. If you prefer to establish a secure connection by using a valid certificate, contact your hosting provider.)



Click **Next**.

In the **Databases** section of the **Settings** tab, enter the same values that you entered for the Test publish profile. You'll find the connection strings you need in the drop-down lists.

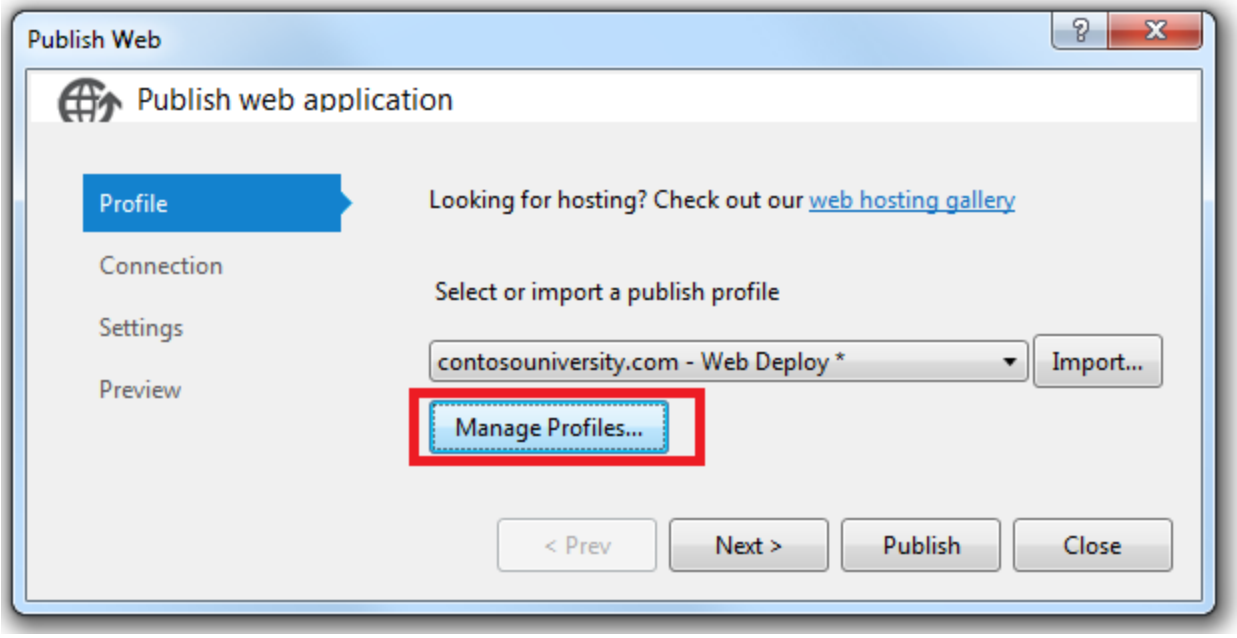
- In the connection string box for **SchoolContext,** select `Data Source=|DataDirectory|School-Prod.sdf`
- Under **SchoolContext**, select **Apply Code First Migrations**.
- In the connection string box for **DefaultConnection**, select `Data Source=|DataDirectory|aspnet-Prod.sdf`
- Under **DefaultConnection**, leave **Update database** cleared.



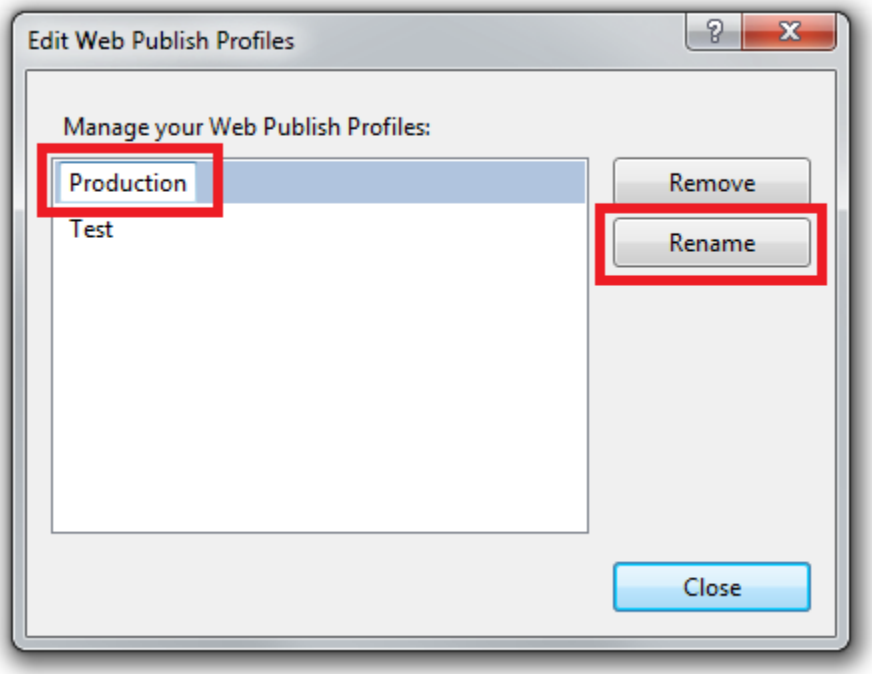
Click **Next**.

In the **Preview** tab, click **Start Preview** to see a list of the files that will be copied. You see the same list that you saw earlier when you deployed to IIS on the local computer.

Before you publish, change the name of the profile so that your Web.Production.config transformation file will be applied. Select the **Profile** tab and click **Manage Profiles**.

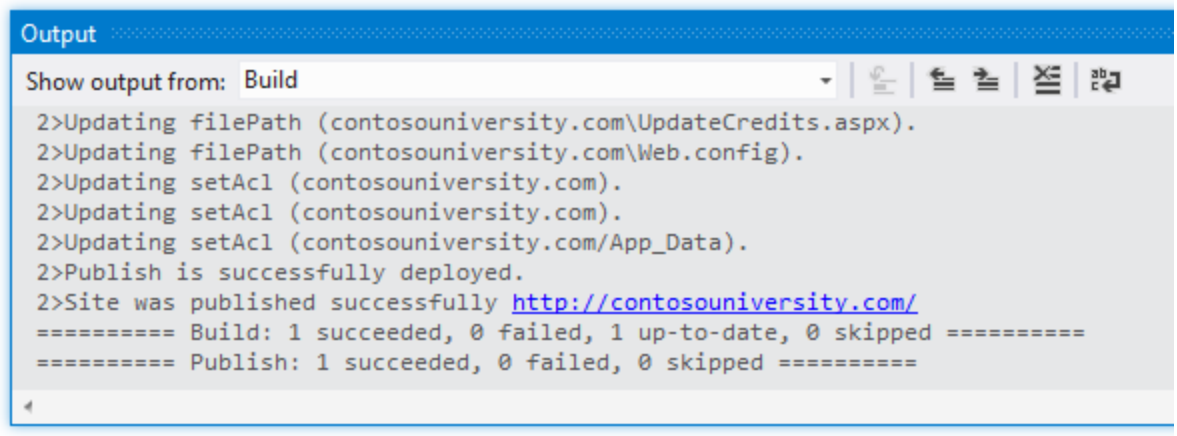


In the **Edit Web Publish Profiles** dialog box, select the production profile, click **Rename**, and change the profile name to Production. Then click **Close**.



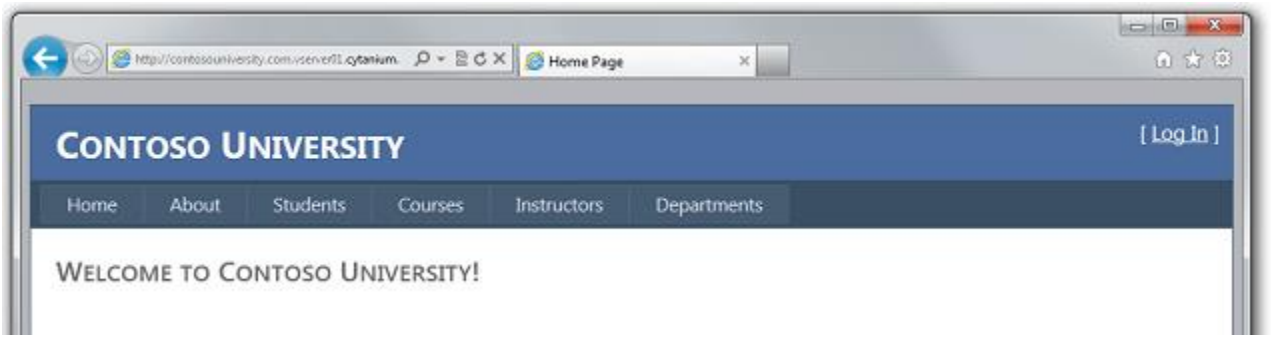
Click **Publish**.

The application is published to the hosting provider. The result shows in the **Output** window.



The browser automatically opens to the URL that you entered in **Destination URL** box on the **Connection** tab of the **Publish Web** wizard. You see the same home page as when you run the site in Visual Studio, except now there is no "(Test)" or "(Dev)" environment indicator in the title bar. This indicates that the environment indicator *Web.config* transformation worked correctly.

Note If you still see "(Test)" in the heading, delete the *obj* folder from the ContosoUniversity project and redeploy. In pre-release versions of the software, the previously applied transformation file (Web.Test.config) might get applied again although you are using the Production profile.

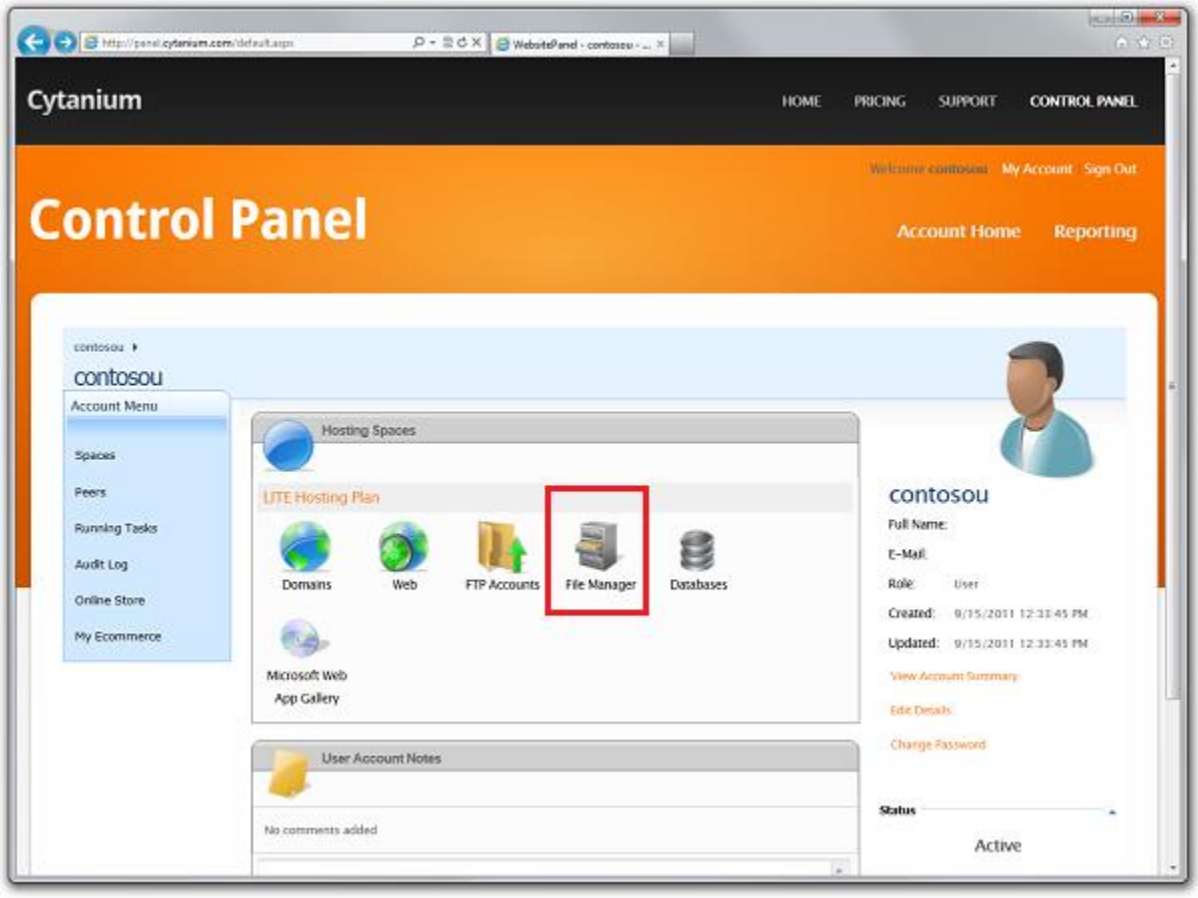


Before you run a page that causes database access, make sure that Elmah will be able to log any errors that occur.

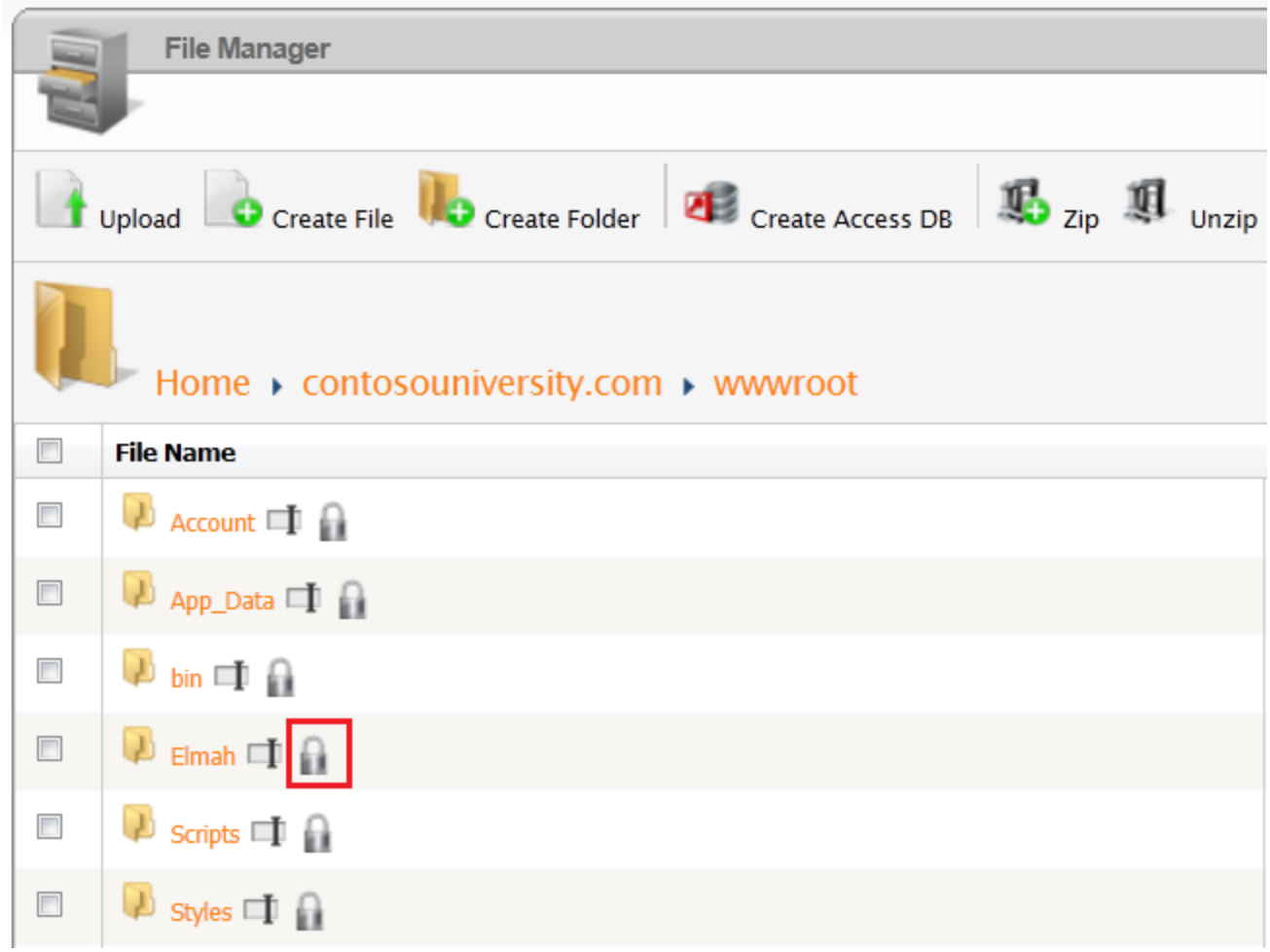
Setting Folder Permissions for Elmah

As you remember from the previous tutorial in this series, you must make sure that the application has Write permissions for the folder in your application where Elmah stores error log files. When you deployed to IIS locally on your computer, you set those permissions manually. In this section, you'll see how to set permissions at Cytanium. (Some hosting providers may not enable you to do this; they may offer one or more predefined folders with Write permissions. In that case you would have to modify your application to use the specified folders.)

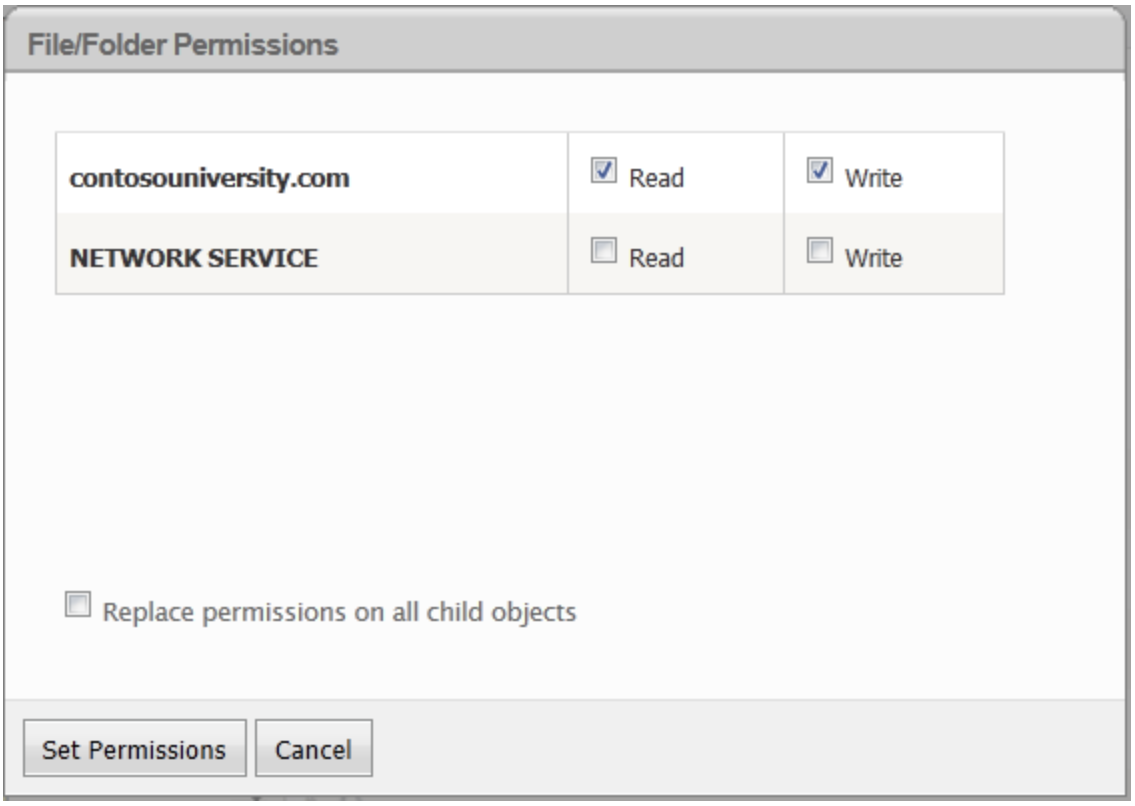
You can set folder permissions in the Cytanium control panel. Go to the control panel URL and select **File Manager**.



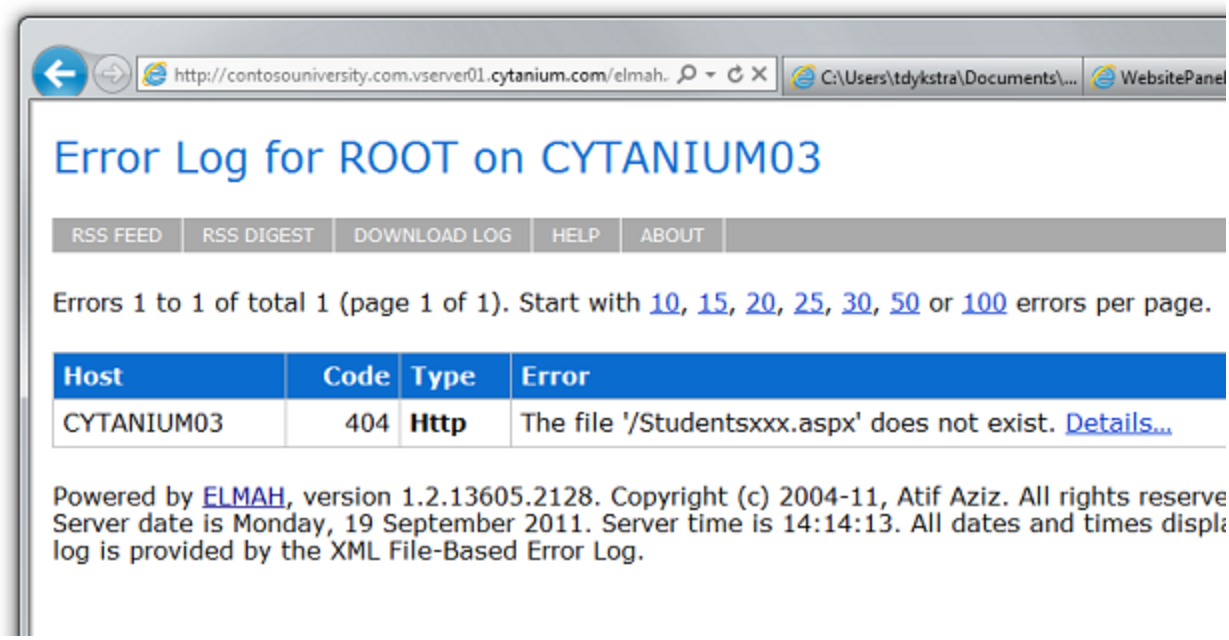
In the **File Manager** box, select **contosouniversity.com** and then **wwwrooot** to see the application's root folder. Click the padlock icon next to **Elmah**.



In the **File/Folder Permissions** window, select the **Read** and **Write** checkboxes for **contosouniversity.com** and click **Set Permissions**.

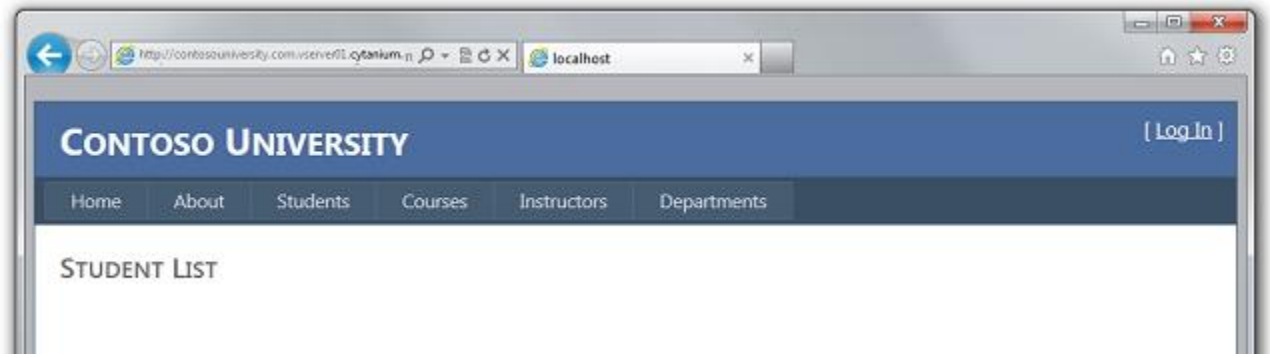


Make sure that Elmah has write access to the *Elmah* folder by causing an error and then displaying the Elmah error report. Request an invalid URL like *Studentsxxx.aspx*. As before, you see the *GenericErrorPage.aspx* page. Click the **Log Out** link, and then run *Elmah.axd*. You get the **Log In** page first, which validates that the *Web.config* transform successfully added Elmah authorization. After you log in, you see the report that shows the error you just caused.

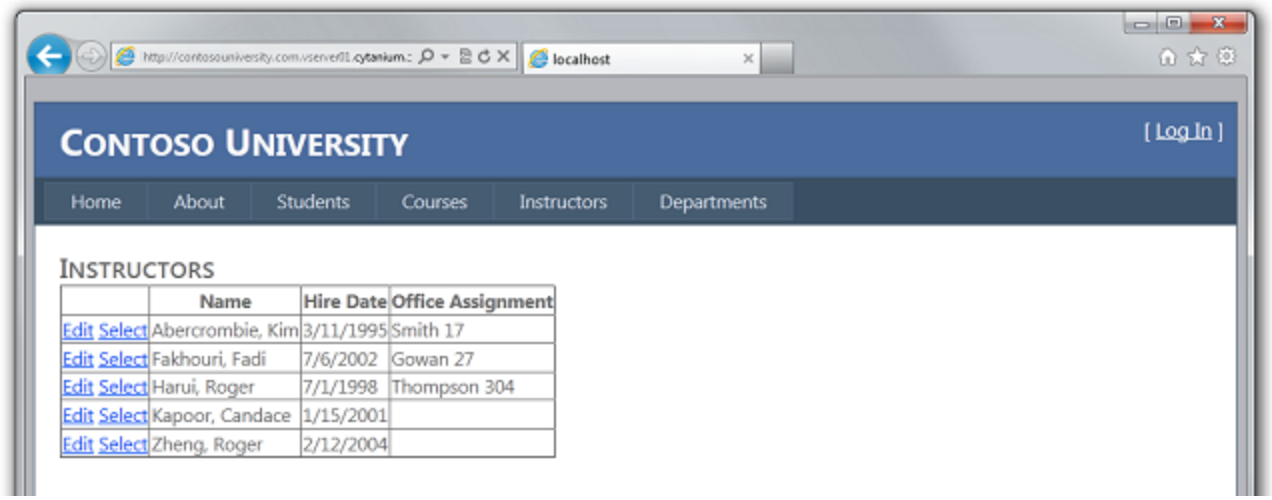


Testing in the Production Environment

Run the **Students** page. The application tries to access the School database for the first time, which triggers Code First Migrations to create the database. When the page displays after a moment's delay, it shows that there are no students.

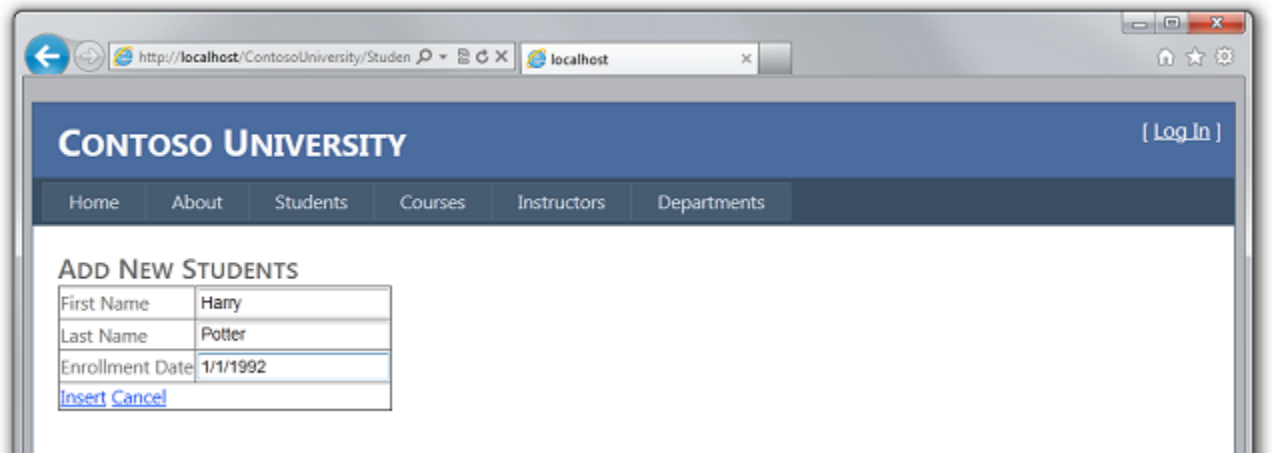


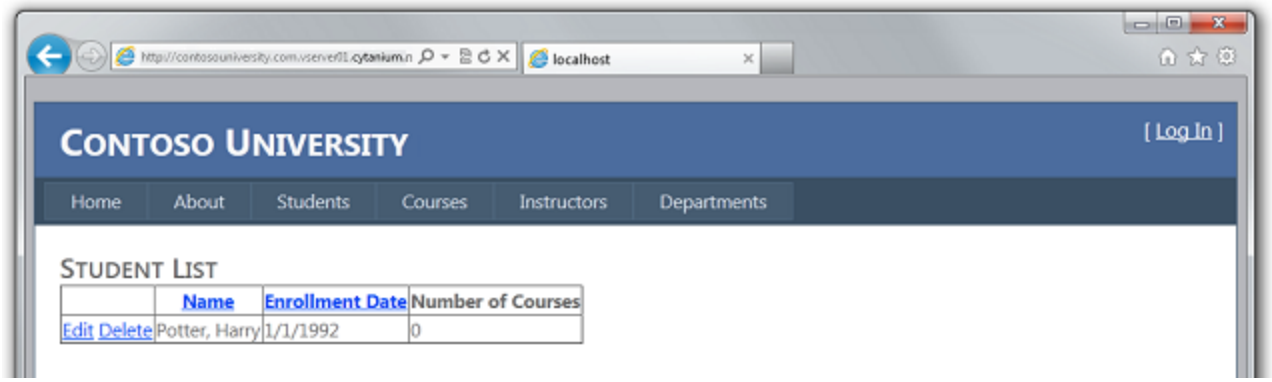
Run the **Instructors** page to verify that the Seed data successfully inserted instructor data in the database.



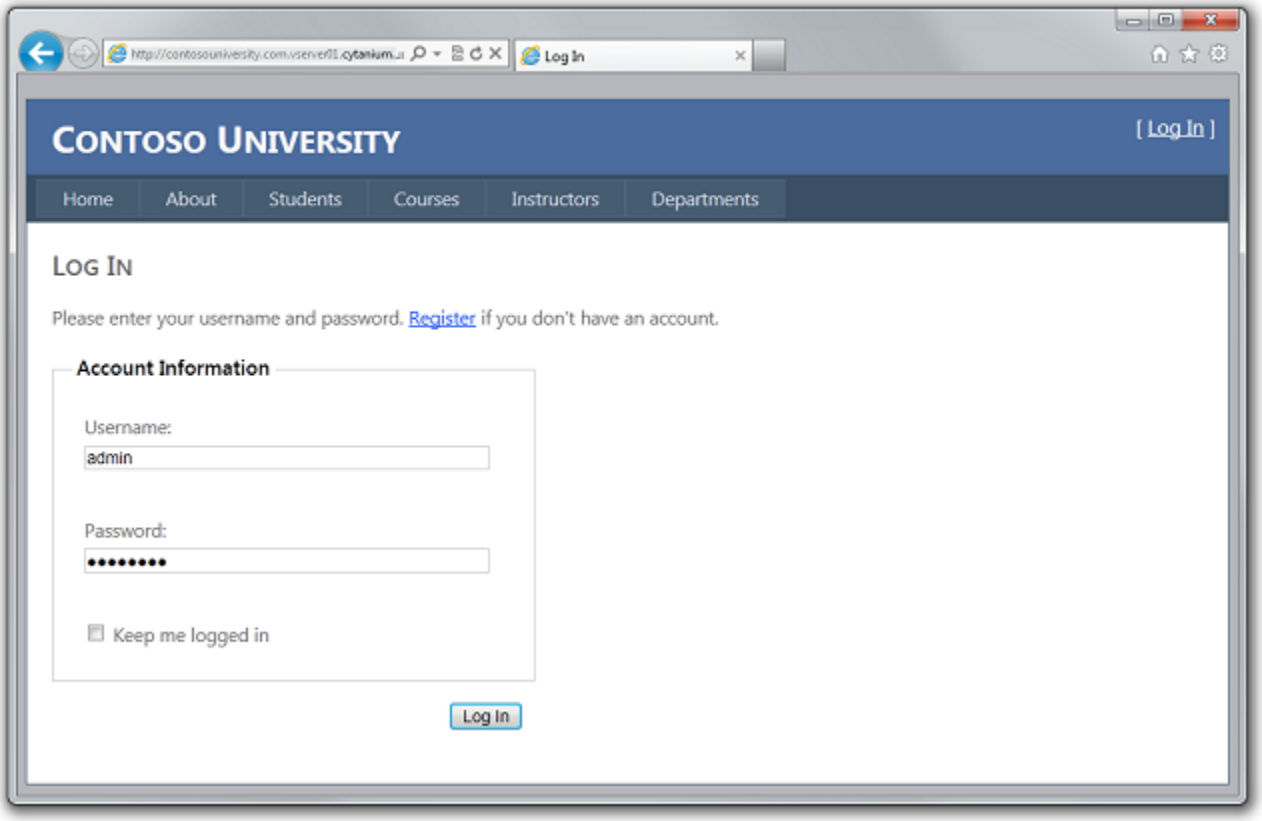
As you did in the test environment, you want to verify that database updates work in the production environment, but you typically do not want to enter test data into your production database. For this tutorial, you'll use the same method you did in test. But in a real application you might want to find a method that validates that database updates are successful without introducing test data into the production database. In some applications, it might be practical to add something and then delete it.

Add a student and then view the data you entered in the **Students** page to verify that you can update data in the database.

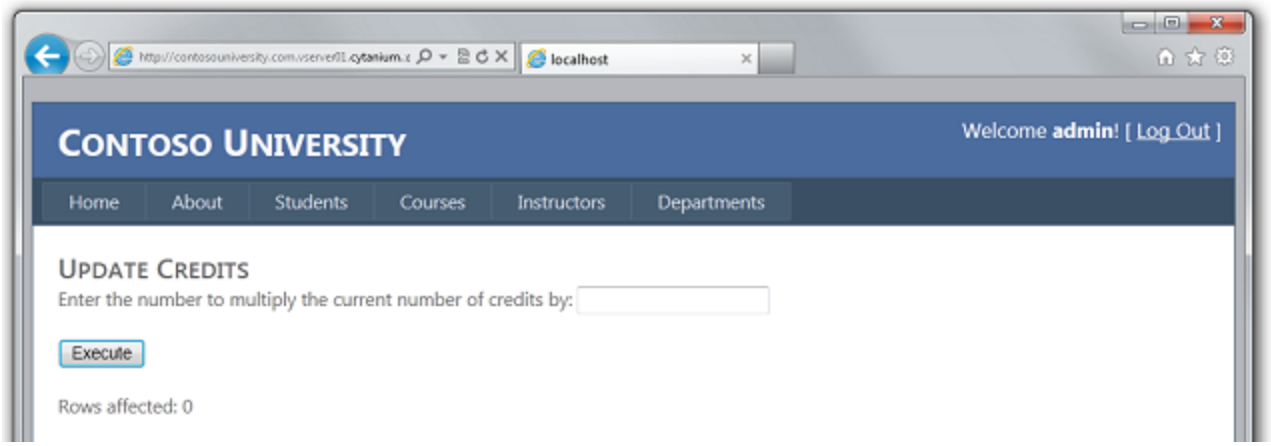




Validate that authorization rules are working correctly by selecting **Update Credits** from the **Courses** menu. The **Log In** page is displayed. Enter your administrator account credentials, click **Log In**, and the **Update Credits** page is displayed.



If login is successful, the **Update Credits** page is displayed. This indicates that the ASP.NET membership database (with the single administrator account) was successfully deployed.



You have now successfully deployed and tested your site and it is available publicly over the Internet.

Creating a More Reliable Test Environment

As explained in the Deploying to the Test Environment tutorial, the most reliable test environment would be a second account at the hosting provider that's just like the production account. This would be more expensive than using local IIS as your test environment, since you would have to sign up for a second hosting account. But if it prevents production site errors or outages, you might decide that it's worth the cost.

Most of the process for creating and deploying to a test account is similar to what you've already done to deploy to production:

- Create a *Web.config* transformation file.
- Create an account at the hosting provider.
- Create a new publish profile and deploy to the test account.

Preventing Public Access to the Test Site

An important consideration for the test account is that it will be live on the Internet, but you don't want the public to use it. To keep the site private you can use one or more of the following methods:

- Contact the hosting provider to set firewall rules that allow access to the testing site only from IP addresses that you use for testing.
- Disguise the URL so that it is not similar to the public site's URL.
- Use a *robots.txt* file to ensure that search engines will not crawl the test site and report links to it in search results.

The first of these methods is obviously the most secure, but the procedure for that is specific to each hosting provider and will not be covered in this tutorial. If you do arrange with your hosting provider to allow only your IP address to browse to the test account URL, you theoretically don't have to worry about search engines crawling it. But even in that case, deploying a *robots.txt* file is a good idea as a backup in case that firewall rule is ever accidentally turned off.

The *robots.txt* file goes in your project folder and should have the following text in it:

```
User-agent: *
Disallow: /
```

The `User-agent` line tells search engines that the rules in the file apply to all search engine web crawlers (robots), and the `Disallow` line specifies that no pages on the site should be crawled.

You probably do want search engines to catalog your production site, so you need to exclude this file from production deployment. To do that, see **Can I exclude specific files or folders from deployment?** in ASP.NET Web Application Project Deployment FAQ. Make sure that you specify the exclusion only for the Production publish profile.

Creating a second hosting account is an approach to working with a test environment that is not required but might be worth the added expense. In the following tutorials, you'll continue to use IIS as your test environment.

In the next tutorial, you'll update application code and deploy your change to the test and production environments.

Deploying a Code-Only Update - 8 of 12

Overview

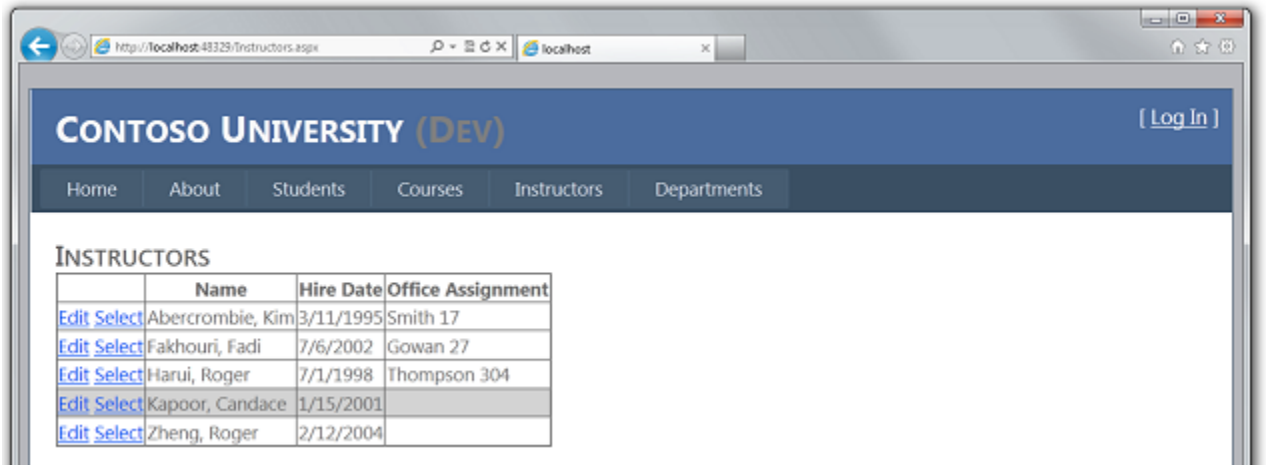
After the initial deployment, your work of maintaining and developing your web site continues, and before long you will want to deploy an update. This tutorial takes you through the process of deploying an update to your application code. This update does not involve a database change; you'll see what's different about deploying a database change in the next tutorial.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Making a Code Change

As a simple example of an update to your application, you'll add to the **Instructors** page a list of courses taught by the selected instructor.

If you run the **Instructors** page, you'll notice that there are **Select** links in the grid, but they don't do anything other than make the row background turn gray.

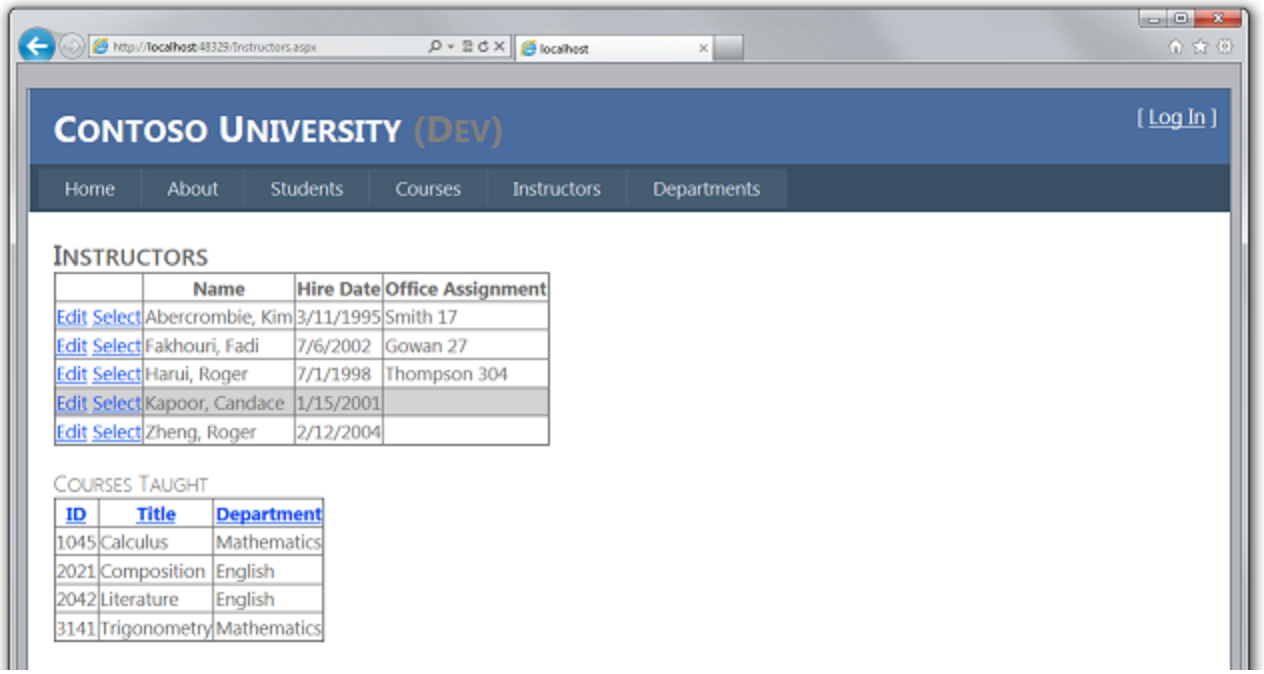


Now you'll add code that runs when the **Select** link is clicked and displays a list of courses taught by the selected instructor .

In *Instructors.aspx*, add the following markup immediately after the **ErrorMessageLabel** `Label` control:

```
<h3>Courses Taught</h3>
<asp:ObjectDataSource ID="CoursesObjectDataSource" runat="server"
TypeName="ContosoUniversity.BLL.SchoolBL"
    DataObjectTypeName="ContosoUniversity.DAL.Course"
SelectMethod="GetCoursesByInstructor">
    <SelectParameters>
        <asp:ControlParameter ControlID="InstructorsGridView" Name="PersonID"
PropertyName="SelectedDataKey.Value"
            Type="Int32" />
    </SelectParameters>
</asp:ObjectDataSource>
<asp:GridView ID="CoursesGridView" runat="server"
DataSourceID="CoursesObjectDataSource"
    AllowSorting="True" AutoGenerateColumns="False" SelectedRowStyle-
BackColor="LightGray"
    DataKeyNames="CourseID">
    <EmptyDataTemplate>
        <p>No courses found.</p>
    </EmptyDataTemplate>
    <Columns>
        <asp:BoundField DataField="CourseID" HeaderText="ID" ReadOnly="True"
SortExpression="CourseID" />
        <asp:BoundField DataField="Title" HeaderText="Title" SortExpression="Title"
/>
        <asp:TemplateField HeaderText="Department" SortExpression="DepartmentID">
            <ItemTemplate>
                <asp:Label ID="GridViewDepartmentLabel" runat="server" Text='<%#
Eval("Department.Name") %>'></asp:Label>
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
```

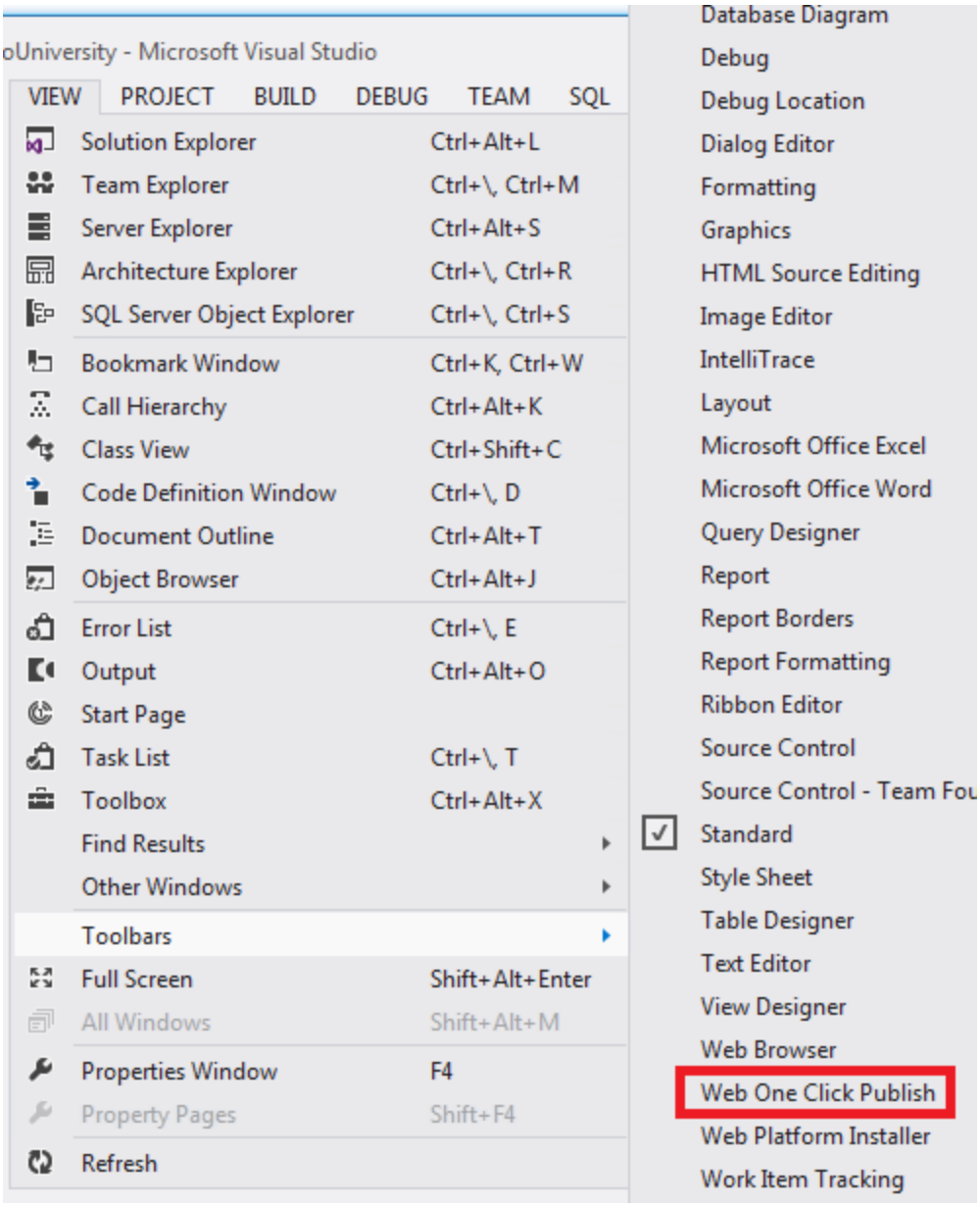
Run the page and select an instructor. You see a list of courses taught by that instructor.



Deploying the Code Update to the Test Environment

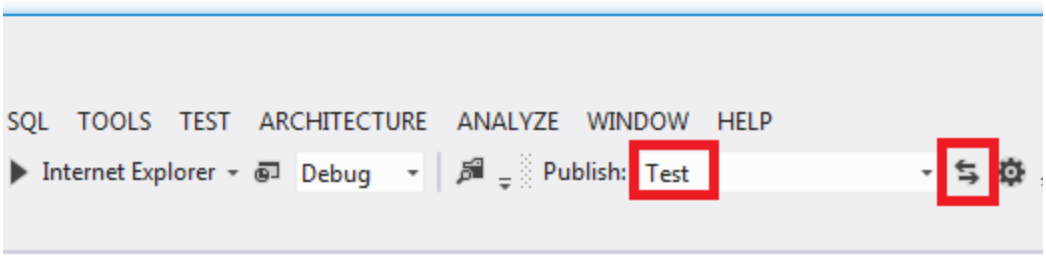
Deploying to the test environment is a simple matter of running one-click publish again. To make this process quicker, you can use the **Web One Click Publish** toolbar.

In the **View** menu, choose **Toolbars** and then select **Web One Click Publish**.

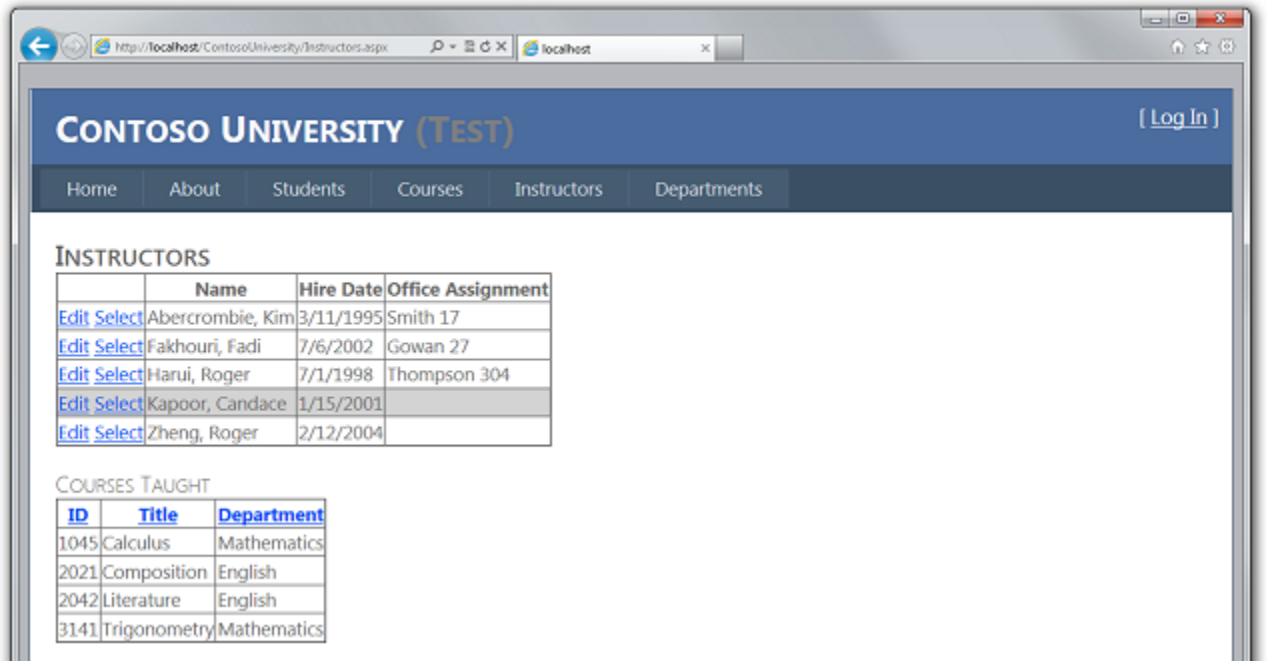


In **Solution Explorer**, select the ContosoUniversity project.

the **Web One Click Publish** toolbar, choose the **Test** publish profile and then click **Publish Web** (the icon with arrows pointing left and right).



Visual Studio deploys the updated application, and the browser automatically opens to the home page. Run the Instructors page and select an instructor to verify that the update was successfully deployed.

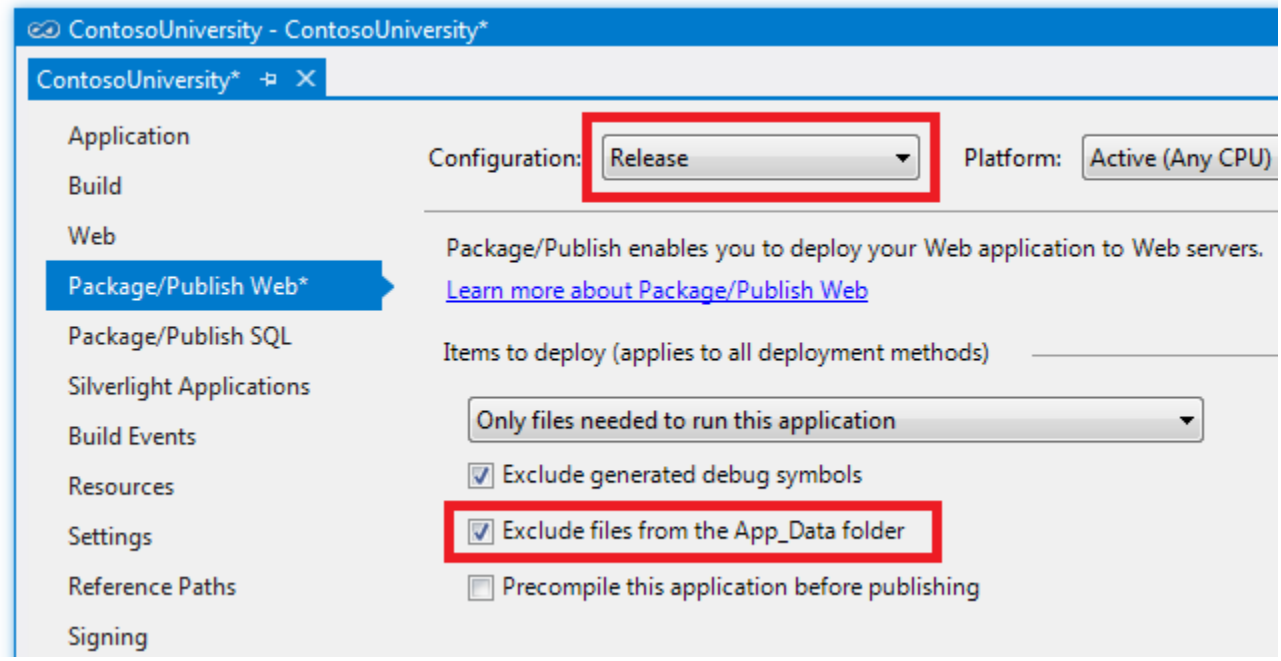


You would normally also do regression testing (that is, test the rest of the site to make sure that the new change didn't break any existing functionality). But for this tutorial you'll skip that step and proceed to deploy the update to production.

Preventing Redeployment of the Initial Database State to Production

In a real application, users interact with your production site after your initial deployment, and the databases are populated with live data. Therefore, you don't want to redeploy the membership database in its initial state, which would wipe out all of the live data. Since SQL Server Compact databases are files in the *App_Data* folder, you have to prevent this by changing deployment settings so that files in the *App_Data* folder aren't deployed.

Open the **Project Properties** window for the ContosoUniversity project, and select the **Package/Publish Web** tab. Make sure that the **Configuration** drop-down box has either **Active (Release)** or **Release** selected, select **Exclude files from the App_Data folder**.



In case you decide to deploy a debug build in the future, it's a good idea to make the same change for the Debug build configuration: change **Configuration** to **Debug** and then select **Exclude files from the App_Data folder**.

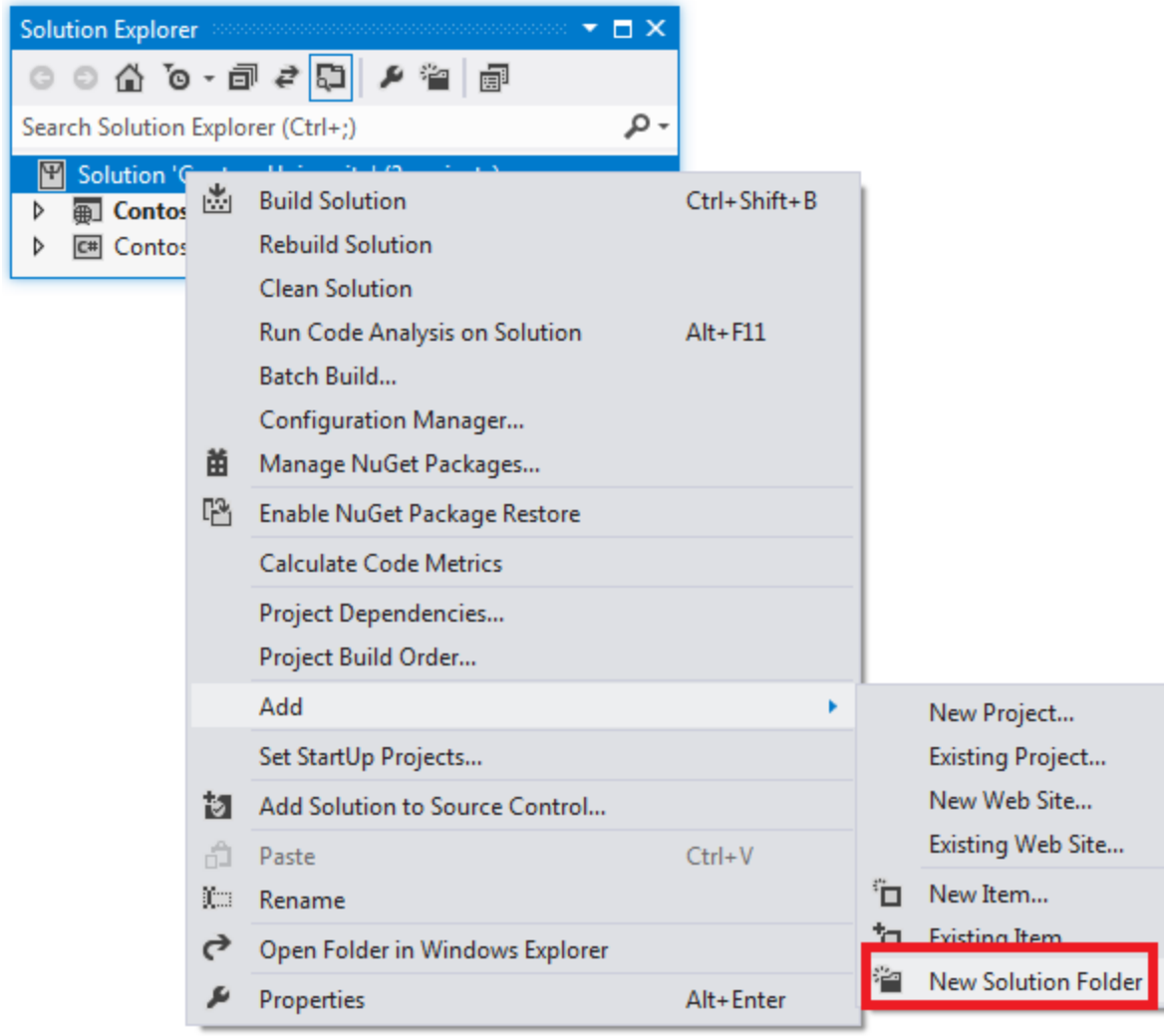
Save and close the **Package/Publish Web** tab.

Important: Make sure that you don't have **Remove additional files at destination** selected in your publish profiles. If you select that option, the deployment process will delete the databases that you have in App_Data in the deployed site, and it will delete the App_Data folder itself.

Preventing User Access to the Production Site During Update

The change you're deploying now is a simple change to a single page. But sometimes you deploy larger changes, and in that case the site can behave strangely if a user requests a page before deployment is finished. To prevent this, you can use an *app_offline.htm* file. When you put a file named *app_offline.htm* in the root folder of your application, IIS automatically displays that file instead of running your application. So to prevent access during deployment, you put *app_offline.htm* in the root folder, run the deployment process, and then remove *app_offline.htm*.

In **Solution Explorer**, right-click the solution (not one of the projects) and select **New Solution Folder**.



Name the folder *SolutionFiles*.

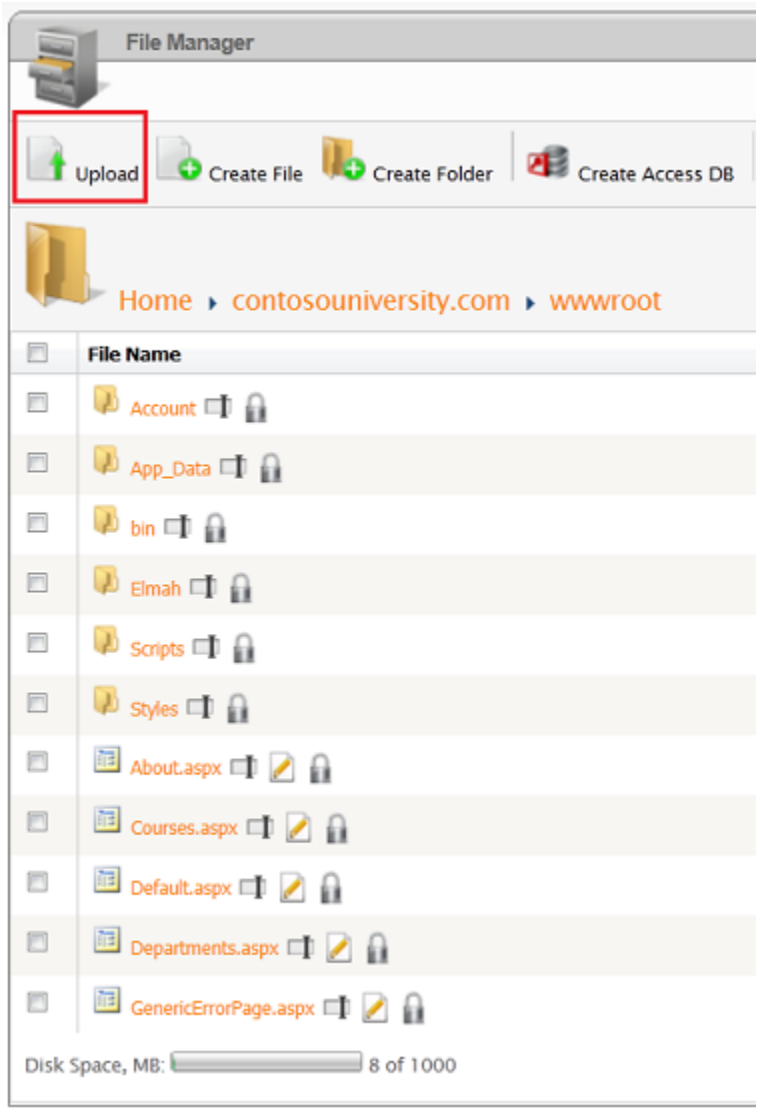
In the new folder create an HTML page named *app_offline.htm*. Replace the existing contents with the following markup:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Contoso University - Under Construction</title>
</head>
<body>
  <h1>Contoso University</h1>
```

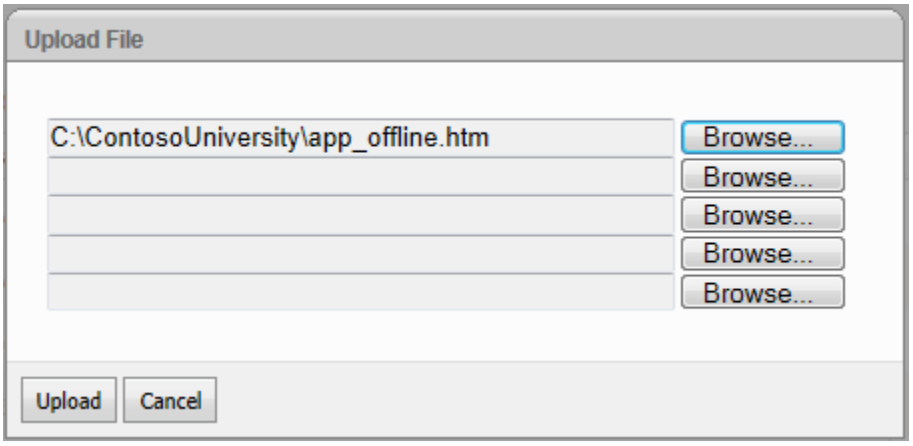
```
  <h2>Under Construction</h2>
  <p>The Contoso University site is temporarily unavailable while we upgrade it.
Please try again later.</p>
</body>
</html>
```

You can copy the *app_offline.htm* file to the site by using an FTP connection or the **File Manager** utility in the hosting provider's control panel. For this tutorial, you'll use the **File Manager**.

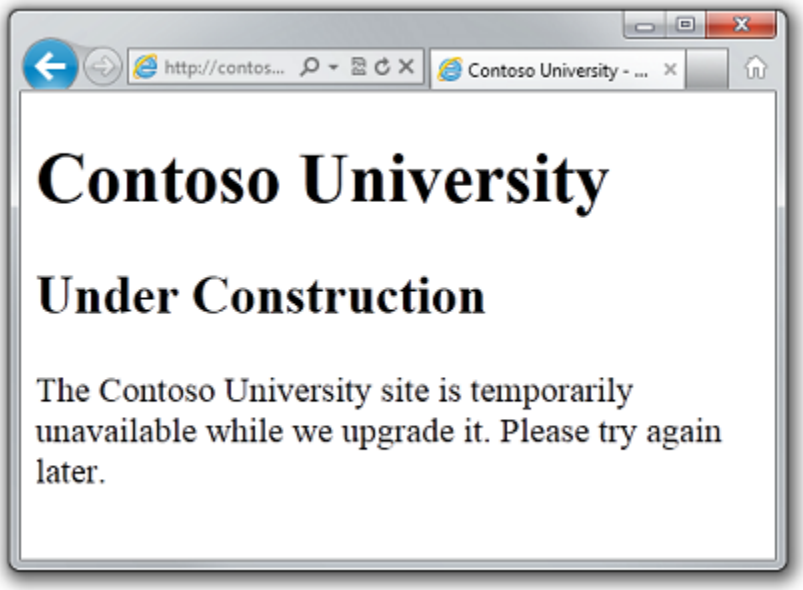
Open the control panel and select **File Manager** as you did in the Deploying to the Production Environment tutorial. Select **contosouniversity.com** and then **wwwroot** to get to your application's root folder, and then click **Upload**.



In the **Upload File** dialog box, select the *app_offline.htm* file and then click **Upload**.



Browse to your site's URL. You see that the *app_offline.htm* page is now displayed instead of your home page.



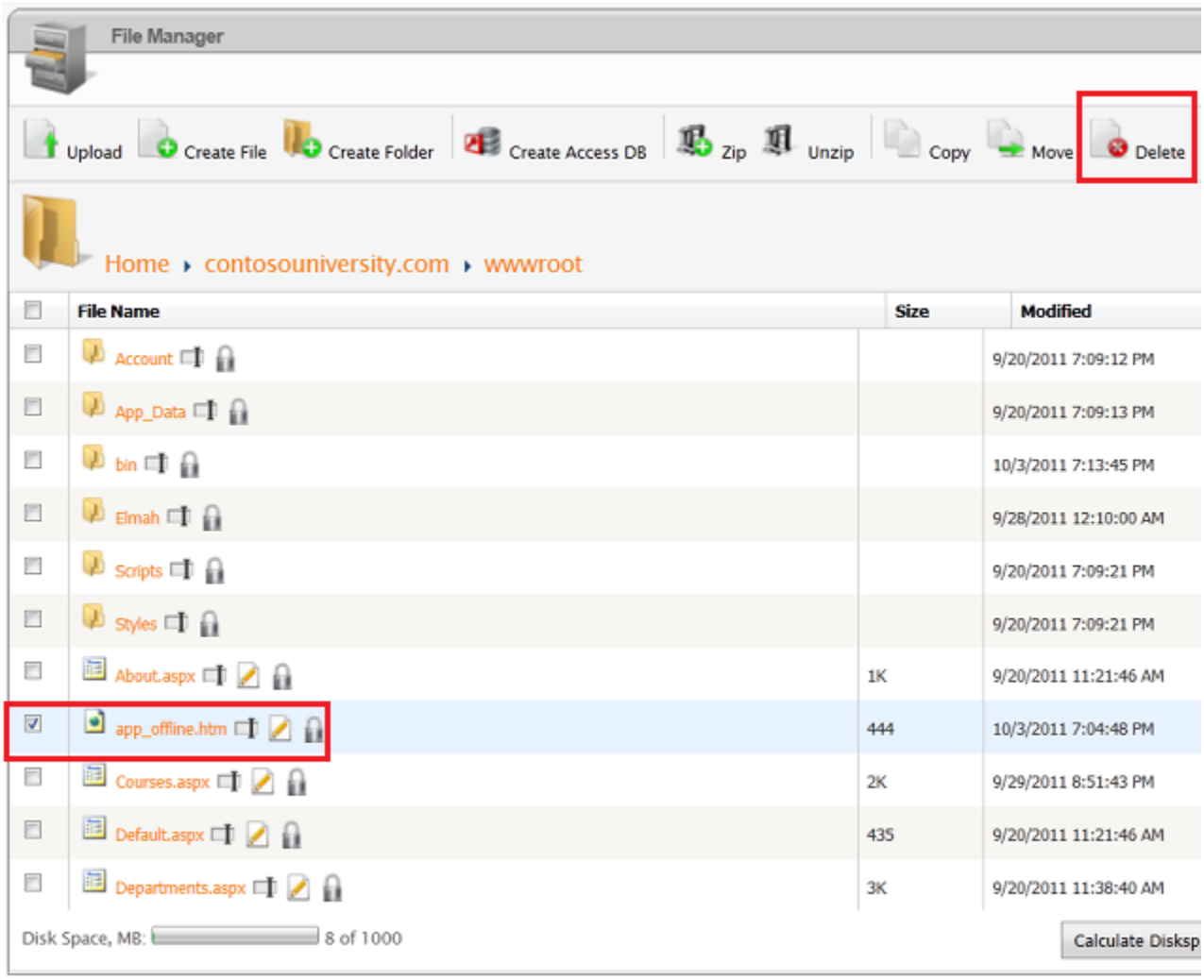
You are now ready to deploy to production.

Deploying the Code Update to the Production Environment

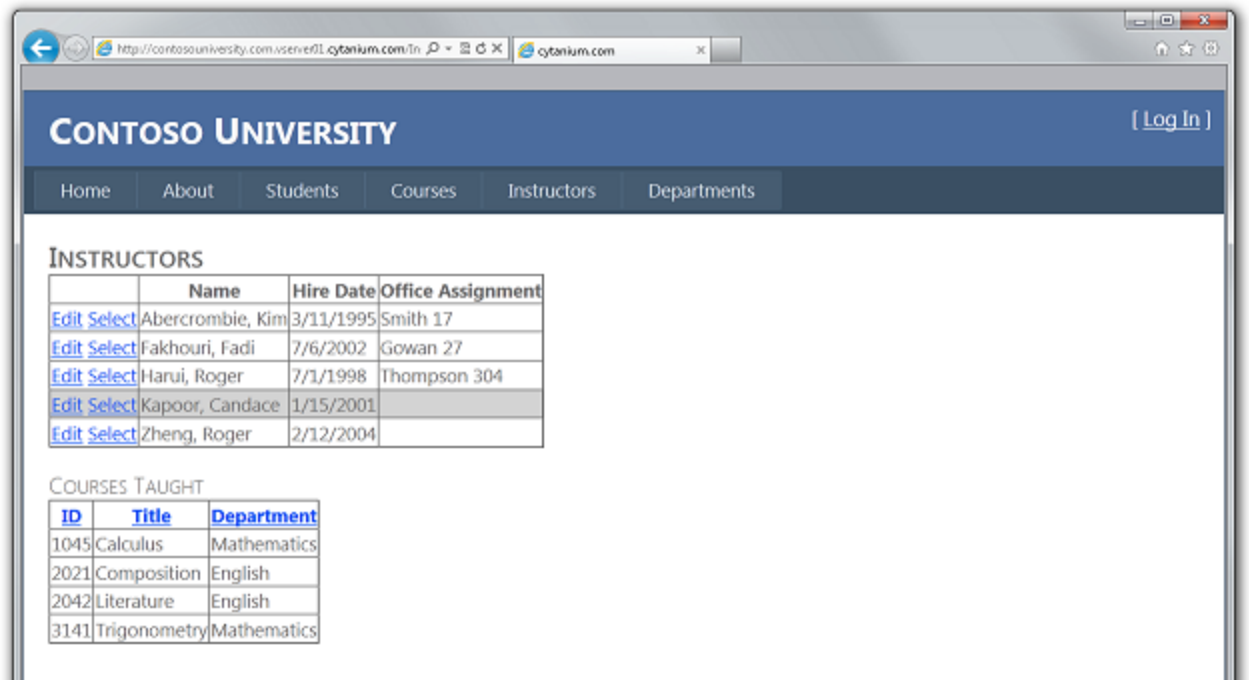
In the **Web One Click Publish** toolbar, choose the **Production** publish profile and then click **Publish Web**.

Visual Studio deploys the updated application and opens the browser to the site's home page. The *app_offline.htm* file is displayed. Before you can test to verify successful deployment, you must remove the *app_offline.htm* file.

Return to the **File Manager** application in the control panel. Select **contosouniversity.com** and **wwwroot**, select **app_offline.htm**, and then click **Delete**.



In the browser, open the Instructors page in the public site, and select an instructor to verify that the update was successfully deployed.



You've now deployed an application update that did not involve a database change. The next tutorial shows you how to deploy a database change.

Deploying a Database Update - 9 of 12

Overview

In this tutorial, you make a database change and related code changes, test the changes in Visual Studio, then deploy the update to both the test and production environments.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Adding a New Column to a Table

In this section, you add a birth date column to the `Person` base class for the `Student` and `Instructor` entities. Then you update the page that displays instructor data so that it displays the new column.

In the *ContosoUniversity.DAL* project, open *Person.cs* and add the following property at the end of the `Person` class (there should be two closing curly braces following it):

```
[DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode = true)]
[Display(Name = "Birth Date")]
public DateTime? BirthDate { get; set; }
```

Next, update the Seed method so that it provides a value for the new column. Open *Migrations\Configuration.cs* and replace the code block that begins `var instructors = new List<Instructor>` with the following code block which includes birth date information:

```
var instructors = new List<Instructor>
{
    new Instructor { FirstMidName = "Kim",     LastName = "Abercrombie", HireDate =
DateTime.Parse("1995-03-11"), BirthDate = DateTime.Parse("1918-08-12"),
OfficeAssignment = new OfficeAssignment { Location = "Smith 17" } },
    new Instructor { FirstMidName = "Fadi",    LastName = "Fakhouri",    HireDate =
DateTime.Parse("2002-07-06"), BirthDate = DateTime.Parse("1960-03-15"),
OfficeAssignment = new OfficeAssignment { Location = "Gowan 27" } },
    new Instructor { FirstMidName = "Roger",   LastName = "Harui",       HireDate =
DateTime.Parse("1998-07-01"), BirthDate = DateTime.Parse("1970-01-11"),
```

```
OfficeAssignment = new OfficeAssignment { Location = "Thompson 304" } },
    new Instructor { FirstMidName = "Candace", LastName = "Kapoor",      HireDate =
DateTime.Parse("2001-01-15"), BirthDate = DateTime.Parse("1975-04-11") },
    new Instructor { FirstMidName = "Roger",   LastName = "Zheng",       HireDate =
DateTime.Parse("2004-02-12"), BirthDate = DateTime.Parse("1957-10-12") }
};
```

In the ContosoUniversity project, open *Instructors.aspx* and add a new template field to display the birth date. Add it between the ones for hire date and office assignment:

```
<asp:TemplateField HeaderText="Birth Date" SortExpression="BirthDate">
    <ItemTemplate>
        <asp:Label ID="InstructorBirthDateLabel" runat="server" Text='<%#
Eval("BirthDate", "{0:d}") %>'></asp:Label>
    </ItemTemplate>
    <EditItemTemplate>
        <asp:TextBox ID="InstructorBirthDateTextBox" runat="server" Text='<%#
Bind("BirthDate", "{0:d}") %>'
            Width="7em"></asp:TextBox>
    </EditItemTemplate>
</asp:TemplateField>
```

(If code indentation gets out of sync, you can press CTRL-K and then CTRL-D to automatically reformat the file.)

Build the solution, and then open the **Package Manager Console** window. Make sure that ContosoUniversity.DAL is still selected as the **Default project**.

In the **Package Manager Console** window, select **ContosoUniversity.DAL** as the **Default project**, and then enter the following command:

```
add-migration AddBirthDate
```

When this command finishes, Visual Studio opens the class file that defines the new `DbMIgration` class, and in the `Up` method you can see the code that creates the new column.

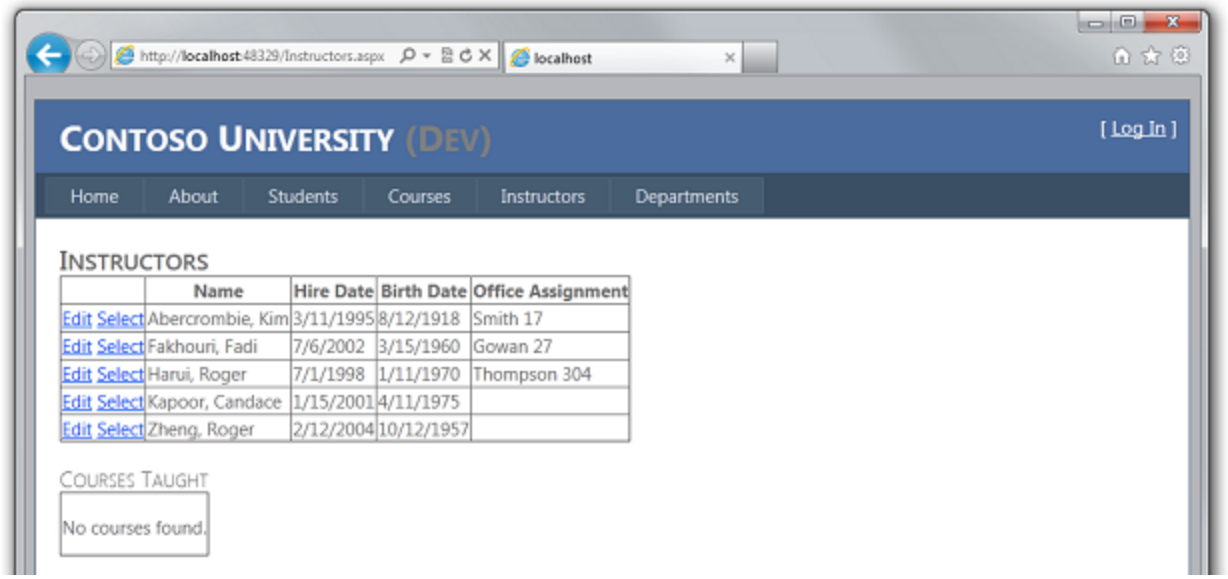
```
public partial class AddBirthDate : DbMigration
{
    public override void Up()
    {
        AddColumn("dbo.Person", "BirthDate", c => c.DateTime());
    }

    public override void Down()
    {
        DropColumn("dbo.Person", "BirthDate");
    }
}
```

Build the solution, and then enter the following command in the **Package Manager Console** window (make sure the ContosoUniversity.DAL project is still selected):

```
update-database
```

When the command finishes, run the application and select the Instructors page. When the page loads, you see that it has the new birth date field.

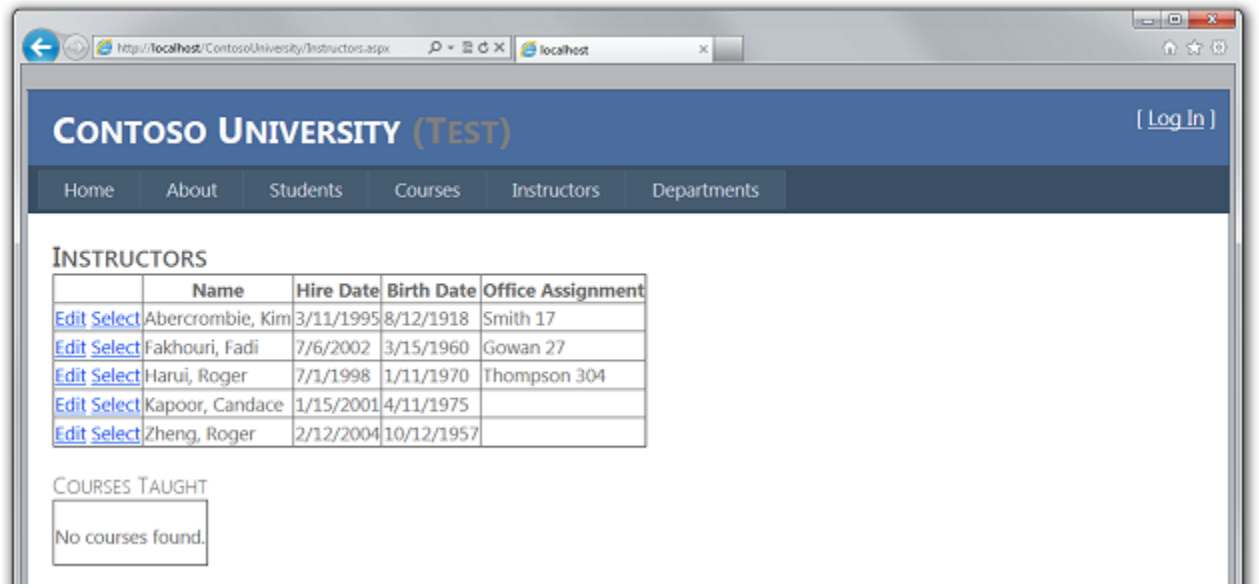


Deploying the Database Update to the Test Environment

In **Solution Explorer** select the ContosoUniversity project.

In the **Web One Click Publish** toolbar, select the **Test** publish profile, and then click **Publish Web**. (If the toolbar is disabled, select the ContosoUniversity project in **Solution Explorer**.)

Visual Studio deploys the updated application, and the browser opens to the home page. Run the Instructors page to verify that the update was successfully deployed. When the application tries to access the database for this page, Code First updates the database schema and runs the `Seed` method. When the page displays, you see the expected **Birth Date** column with dates in it.



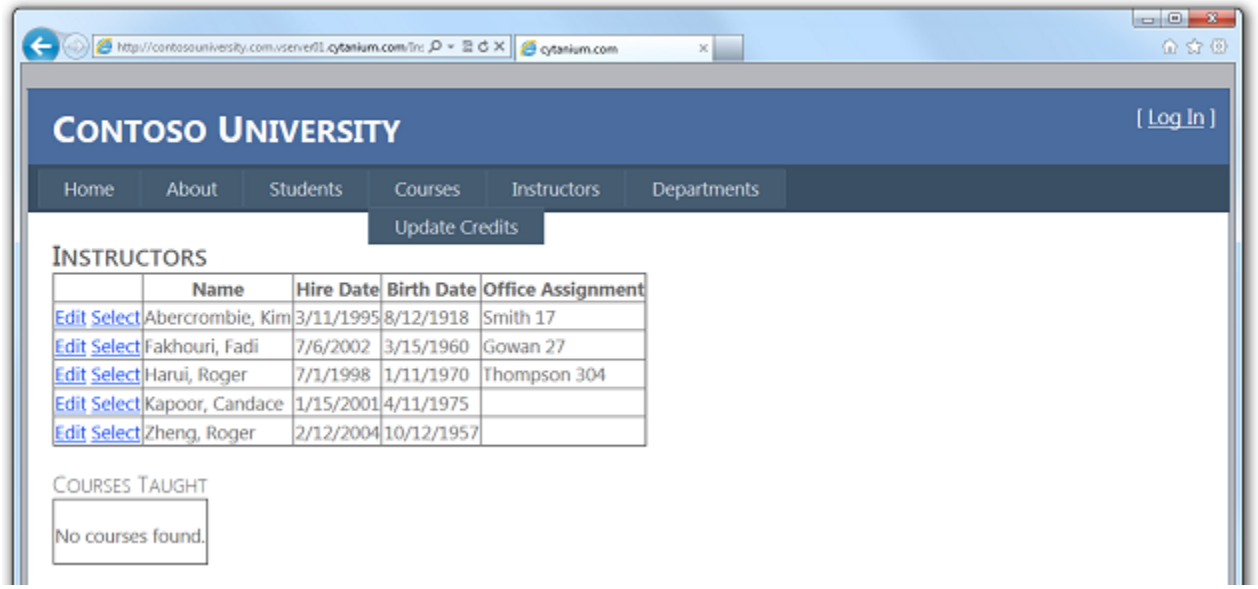
Deploying the Database Update to the Production Environment

You can now deploy to production. The only difference is that you'll use *app_offline.htm* to prevent users from accessing the site and thus updating the database while you're deploying changes. For production deployment perform the following steps:

- Upload the *app_offline.htm* file to the production site.
- In Visual Studio, choose the Production profile in the **Web One Click Publish** toolbar and click **Publish Web**.
- Delete the *app_offline.htm* file from the production site.

Note While your application is in use in the production environment you should be implementing a backup plan. That is, you should be periodically copying the *School-Prod.sdf* and *aspnet-Prod.sdf* files from the production site to a secure storage location, and you should be keeping several generations of such backups. When you update the database, you should make a backup copy from immediately before the change. Then, if you make a mistake and don't discover it until after you have deployed it to production, you will still be able to recover the database to the state it was in before it became corrupted.

When Visual Studio opens the home page URL in the browser, the *app_offline.htm* page is displayed. After you delete the *app_offline.htm* file, you can browse to your home page again to verify that the update was successfully deployed.



You've now deployed an application update that included a database change to both test and production. The next tutorial shows you how to migrate your database from SQL Server Compact to SQL Server Express and SQL Server.

Migrating to SQL Server - 10 of 12

Overview

This tutorial shows you how to migrate from SQL Server Compact to SQL Server. One reason you might want to do that is to take advantage of SQL Server features that SQL Server Compact does not support, such as stored procedures, triggers, views, or replication. For more information about the differences between SQL Server Compact and SQL Server, see the Deploying SQL Server Compact tutorial.

SQL Server Express versus full SQL Server for Development

Once you've decided to upgrade to SQL Server, you might want to use SQL Server or SQL Server Express in your development and test environments. In addition to the differences in tool support and in database engine features, there are differences in provider implementations between SQL Server Compact and other versions of SQL Server. These differences can cause the same code to generate different results. Therefore, if you decide to keep SQL Server Compact as your development database, you should thoroughly test your site in SQL Server or SQL Server Express in a test environment before each deployment to production.

Unlike SQL Server Compact, SQL Server Express is essentially the same database engine and uses the same .NET provider as full SQL Server. When you test with SQL Server Express, you can be confident of getting the same results as you will with SQL Server. You can use most of the same database tools with SQL Server Express that you can use with SQL Server (a notable exception being SQL Server Profiler), and it supports other features of SQL Server like stored procedures, views, triggers, and replication. (You typically have to use full SQL Server in a production website, however. SQL Server Express can run in a shared hosting environment, but it was not designed for that, and many hosting providers do not support it.)

If you are using Visual Studio 2012, you typically choose SQL Server Express LocalDB for your development environment because that is what is installed by default with Visual Studio. However, LocalDB does not work in IIS, so for your test environment you have to use either SQL Server or SQL Server Express.

Combining Databases versus Keeping Them Separate

The Contoso University application has two SQL Server Compact databases: the membership database (*aspnet.sdf*) and the application database (*School.sdf*). When you migrate, you can migrate these databases to two separate databases or to a single database. You might want to combine them in order to facilitate database joins between your application database and your membership database. Your hosting plan might also provide a reason to combine them. For example, the hosting provider might charge more for multiple databases or

might not even allow more than one database. That's the case with the Cytanium Lite hosting account that's used for this tutorial, which allows only a single SQL Server database.

In this tutorial, you'll migrate your two databases this way:

- Migrate to two LocalDB databases in the development environment.
- Migrate to two SQL Server Express databases in the test environment.
- Migrate to one combined full SQL Server database in the production environment.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Installing SQL Server Express

SQL Server Express is automatically installed by default with Visual Studio 2010, but by default it is not installed with Visual Studio 2012. To install SQL Server 2012 Express, click the following link

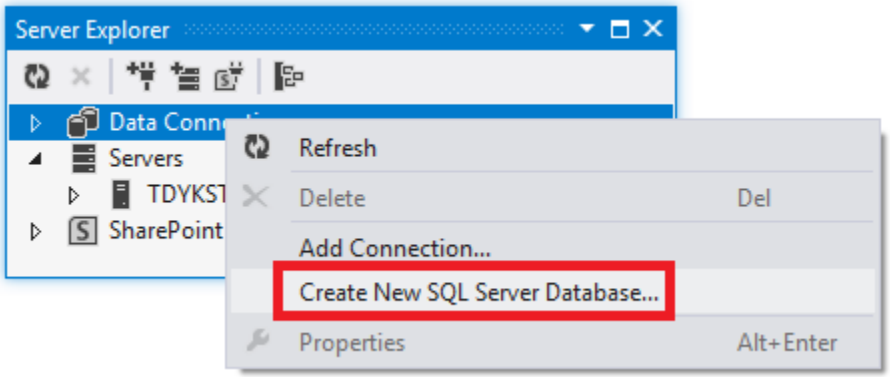
- SQL Server Express 2012

Choose *ENU/x64/SQLEXPR_x64_ENU.exe* or *ENU/x86/SQLEXPR_x86_ENU.exe*, and in the installation wizard accept the default settings. For more information about installation options, see Install SQL Server 2012 from the Installation Wizard (Setup).

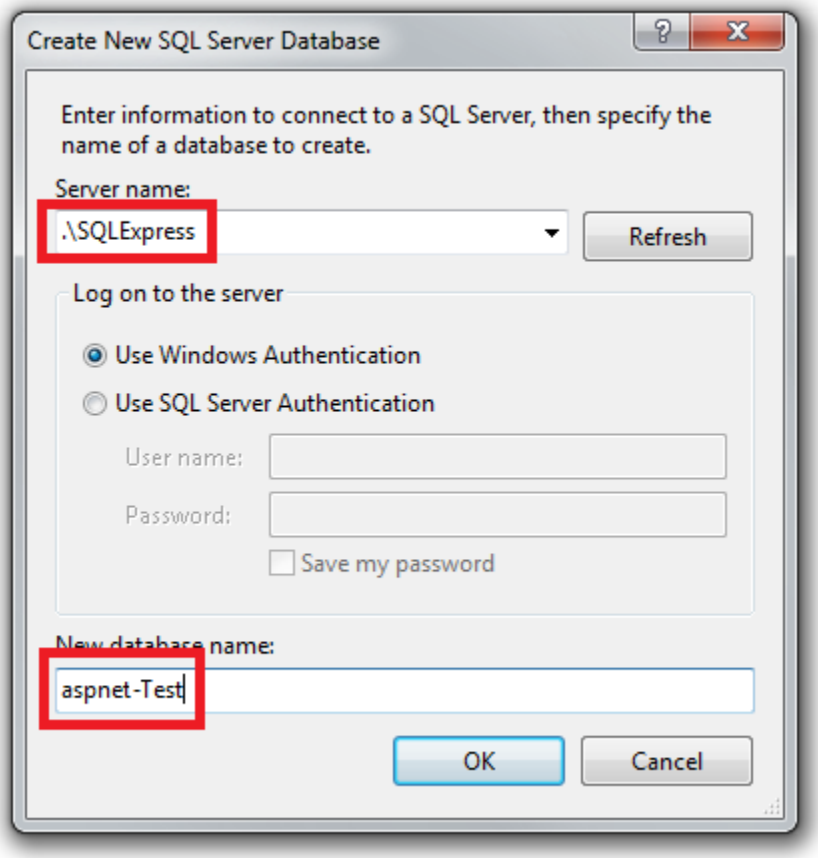
Creating SQL Server Express Databases for the Test Environment

The next step is to create the ASP.NET membership and School databases.

From the **View** menu select **Server Explorer** (**Database Explorer** in Visual Web Developer), and then right-click **Data Connections** and select **Create New SQL Server Database**.



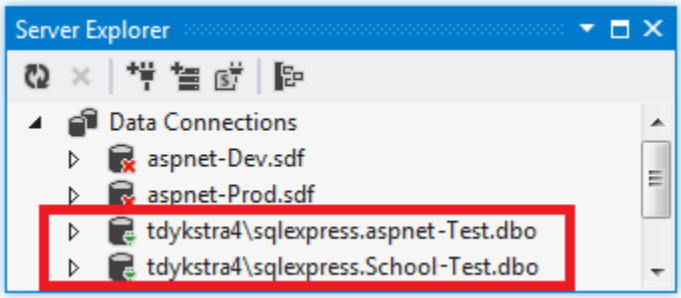
In the **Create New SQL Server Database** dialog box, enter ".\SQLExpress" in the **Server name** box and "aspnet-Test" in the **New database name** box, then click **OK**.



Follow the same procedure to create a new SQL Server Express School database named "School-Test".

(You're appending "Test" to these database names because later you'll create an additional instance of each database for the development environment, and you need to be able to differentiate the two sets of databases.)

Server Explorer now shows the two new databases.



Creating a Grant Script for the New Databases

When the application runs in IIS on your development computer, the application accesses the database by using the default application pool's credentials. However, by default, the application pool identity does not have permission to open the databases. So you have to run a script to grant that permission. In this section you create the script that you'll run later to make sure that the application can open the databases when it runs in IIS.

In the solution's *SolutionFiles* folder that you created in the Deploying to the Production Environment tutorial, create a new SQL file named *Grant.sql*. Copy the following SQL commands into the file, and then save and close the file:

```
IF NOT EXISTS (SELECT name FROM sys.server_principals WHERE name = 'IIS
APPPOOL\DefaultAppPool')
BEGIN
    CREATE LOGIN [IIS APPPOOL\DefaultAppPool]
      FROM WINDOWS WITH DEFAULT_DATABASE=[master],
      DEFAULT_LANGUAGE=[us_english]
END
GO
CREATE USER [ContosoUniversityUser]
  FOR LOGIN [IIS APPPOOL\DefaultAppPool]
GO
EXEC sp_addrolemember 'db_owner', 'ContosoUniversityUser'
GO
```

Note This script is designed to work with SQL Server 2008 and with the IIS settings in Windows 7 as they are specified in this tutorial. If you're using a different version of SQL Server or of Windows, or if you set up IIS on your computer differently, changes to this script might be required. For more information about SQL Server scripts, see SQL Server Books Online.

Security Note This script gives db_owner permissions to the user that accesses the database at run time, which is what you'll have in the production environment. In some scenarios you might want to specify a user that has full database schema update permissions only for deployment, and specify for run time a different user that has permissions only to read and write data. For more information, see **Reviewing the Automatic Web.config Changes for Code First Migrations** in Deploying to IIS as a Test Environment.

Configuring Database Deployment for the Test Environment

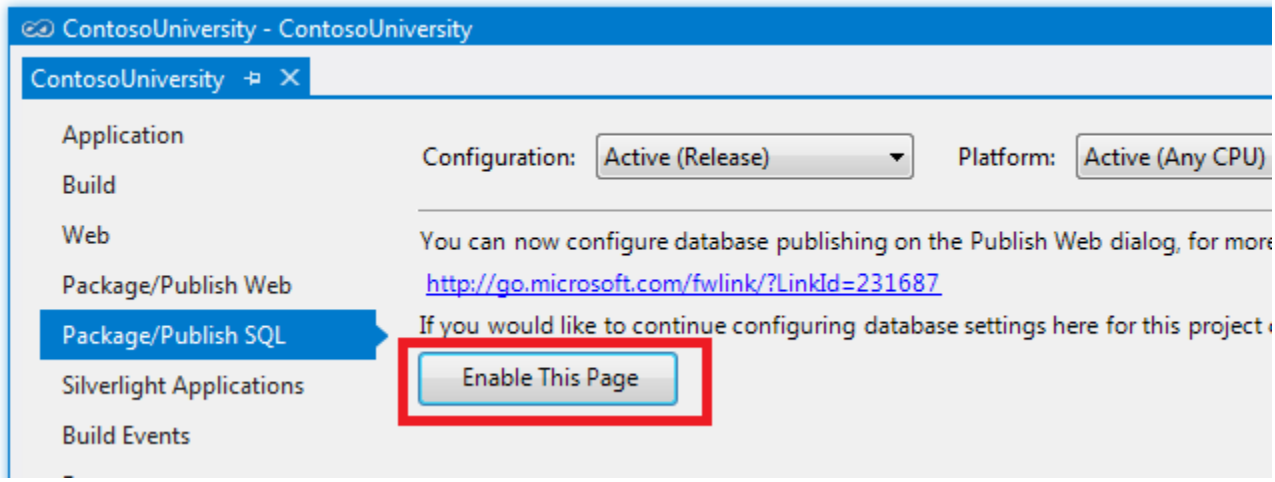
Next, you'll configure Visual Studio so that it will do the following tasks for each database:

- Generate a SQL script that creates the source database's structure (tables, columns, constraints, etc.) in the destination database.
- Generate a SQL script that inserts the source database's data into the tables in the destination database.
- Run the generated scripts, and the Grant script that you created, in the destination database.

Open the **Project Properties** window and select the **Package/Publish SQL** tab.

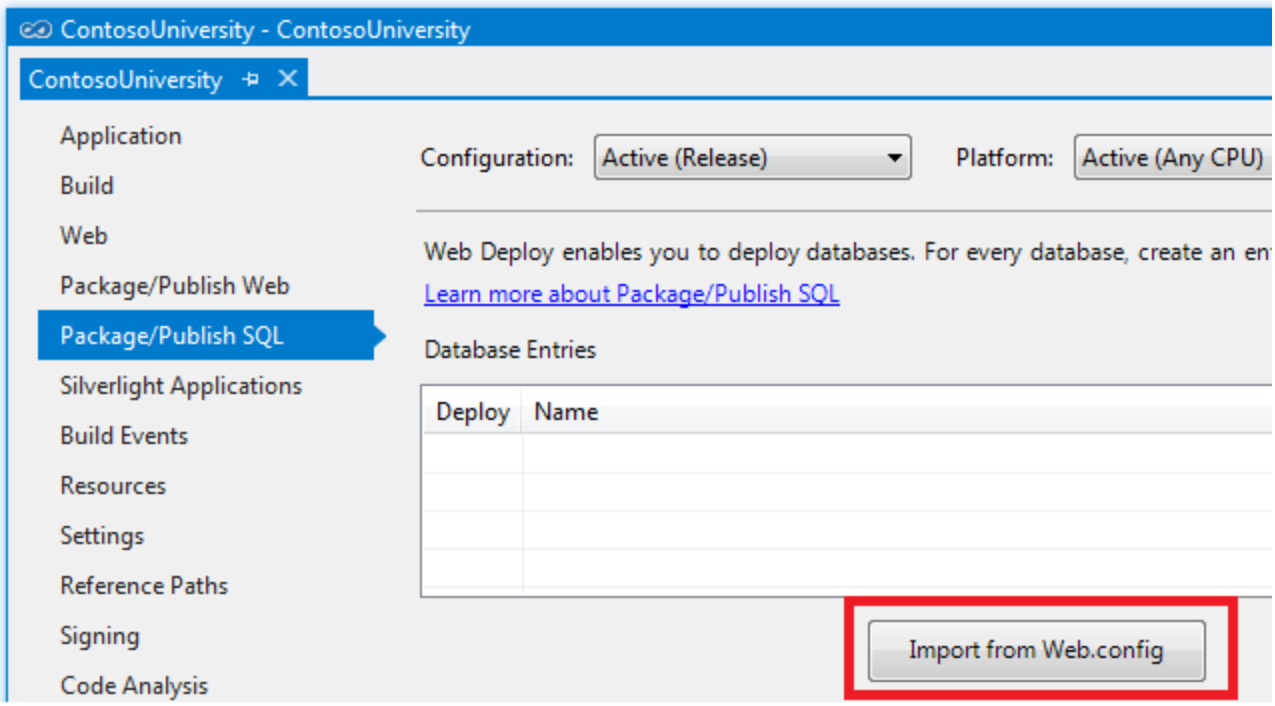
Make sure that **Active (Release)** or **Release** is selected in the **Configuration** drop-down list.

Click **Enable this Page**.



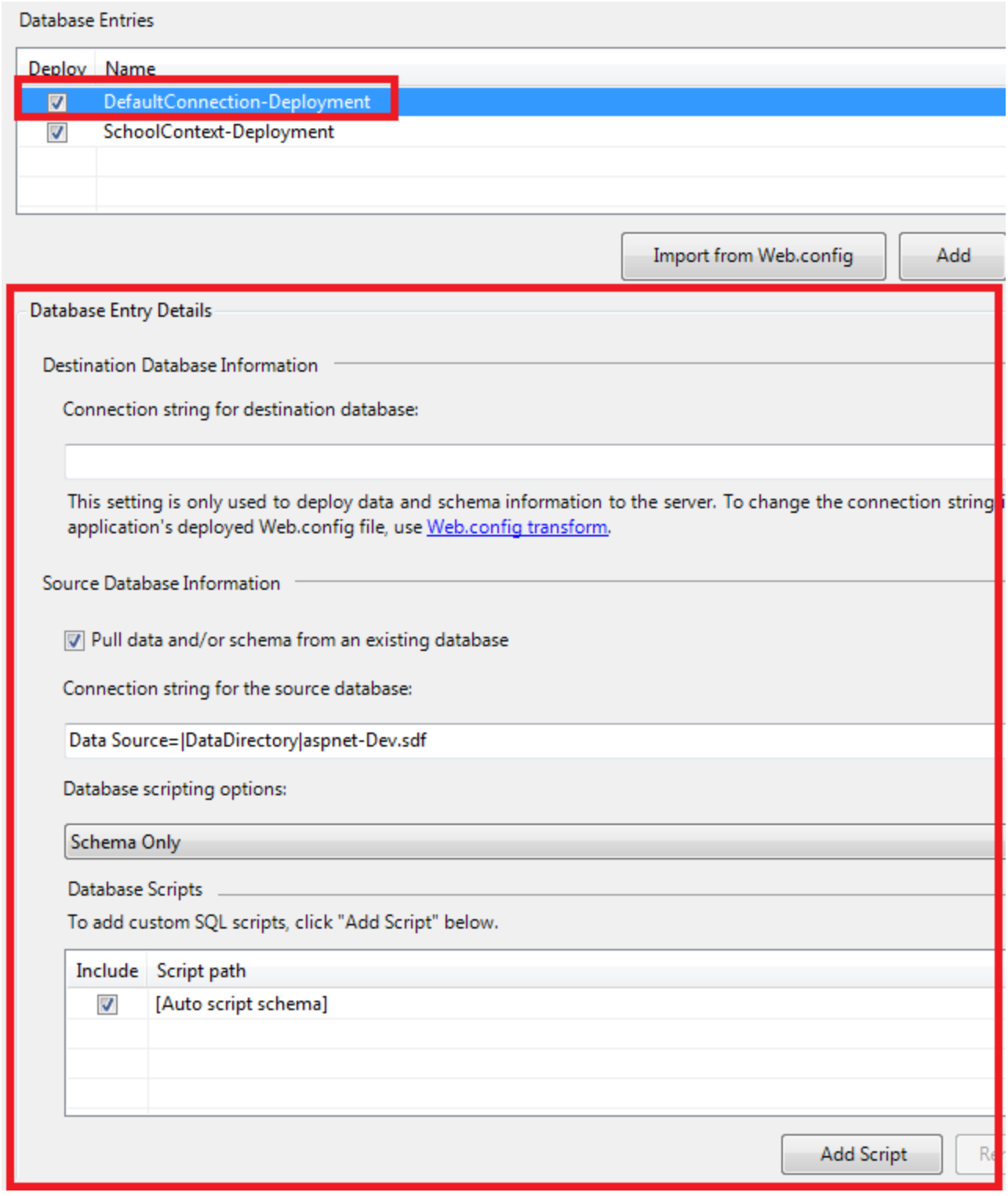
The **Package/Publish SQL** tab is normally disabled because it specifies a legacy deployment method. For most scenarios, you should configure database deployment in the **Publish Web** wizard. Migrating from SQL Server Compact to SQL Server or SQL Server Express is a special case for which this method is a good choice.

Click **Import from Web.config**.



Visual Studio looks for connection strings in the *Web.config* file, finds one for the membership database and one for the School database, and adds a row corresponding to each connection string in the **Database Entries** table. The connection strings it finds are for the existing SQL Server Compact databases, and your next step will be to configure how and where to deploy these databases.

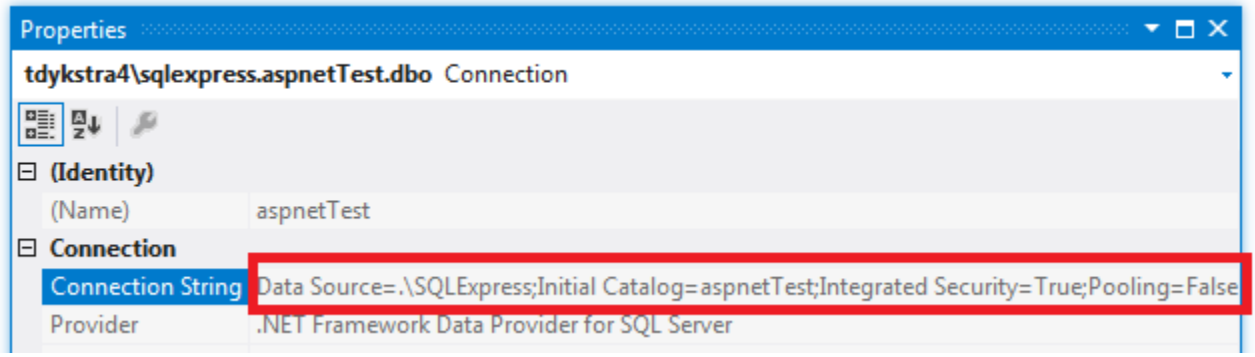
You enter database deployment settings in the **Database Entry Details** section below the **Database Entries** table. The settings shown in the **Database Entry Details** section pertain to whichever row in the **Database Entries** table is selected, as shown in the following illustration.



Configuring Deployment Settings for the Membership Database

Select the **DefaultConnection-Deployment** row in the **Database Entries** table in order to configure settings that apply to the membership database.

In **Connection string for destination database**, enter a connection string that points to the new SQL Server Express membership database. You can get the connection string you need from **Server Explorer**. In **Server Explorer**, expand **Data Connections** and select the **aspnetTest** database, then from the **Properties** window copy the **Connection String** value.



The same connection string is reproduced here:

```
Data Source=.\SQLExpress;Initial Catalog=aspnet-Test;Integrated
Security=True;Pooling=False
```

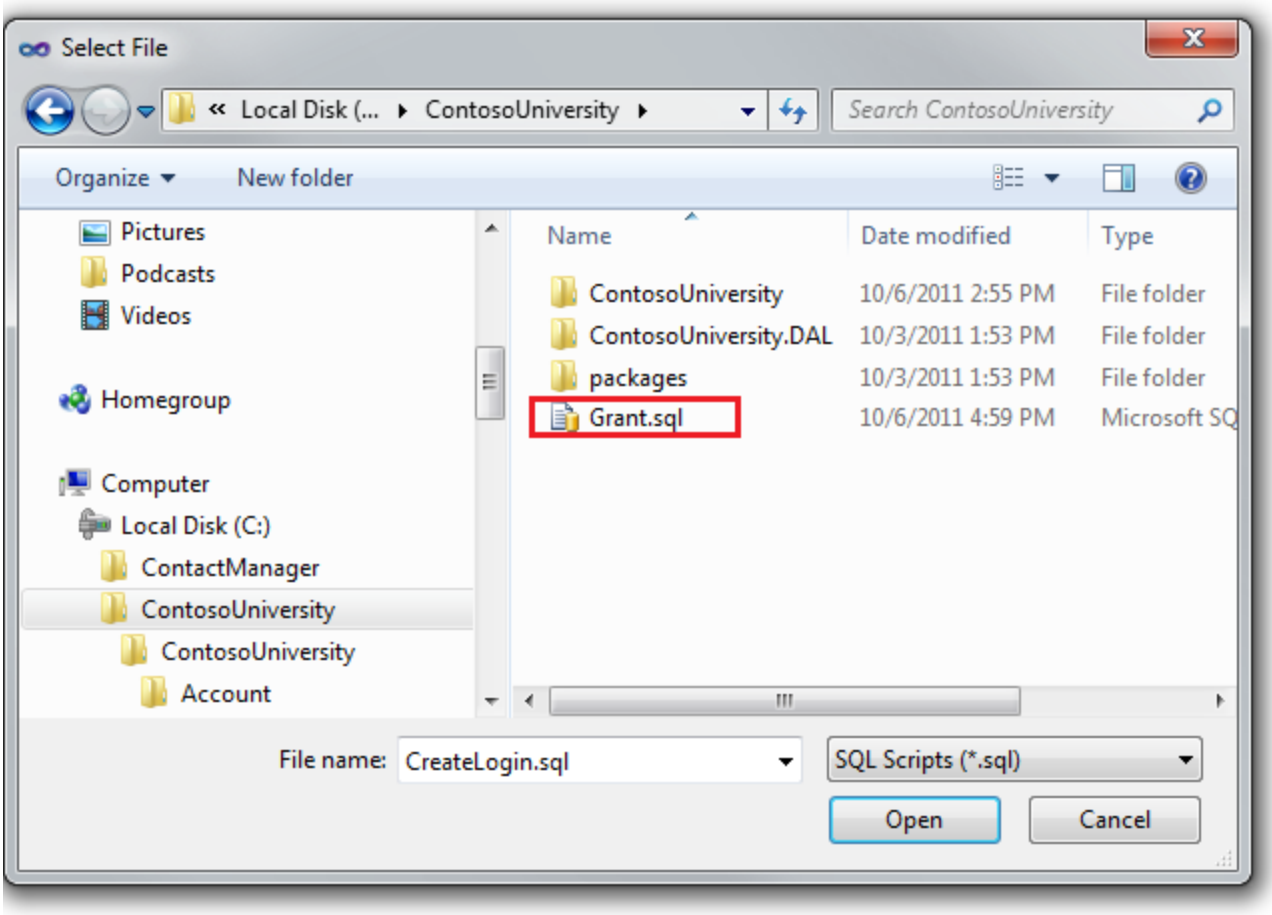
Copy and paste this connection string into **Connection string for destination database** in the **Package/Publish SQL** tab.

Make sure that **Pull data and/or schema from an existing database** is selected. This is what causes SQL scripts to be automatically generated and run in the destination database.

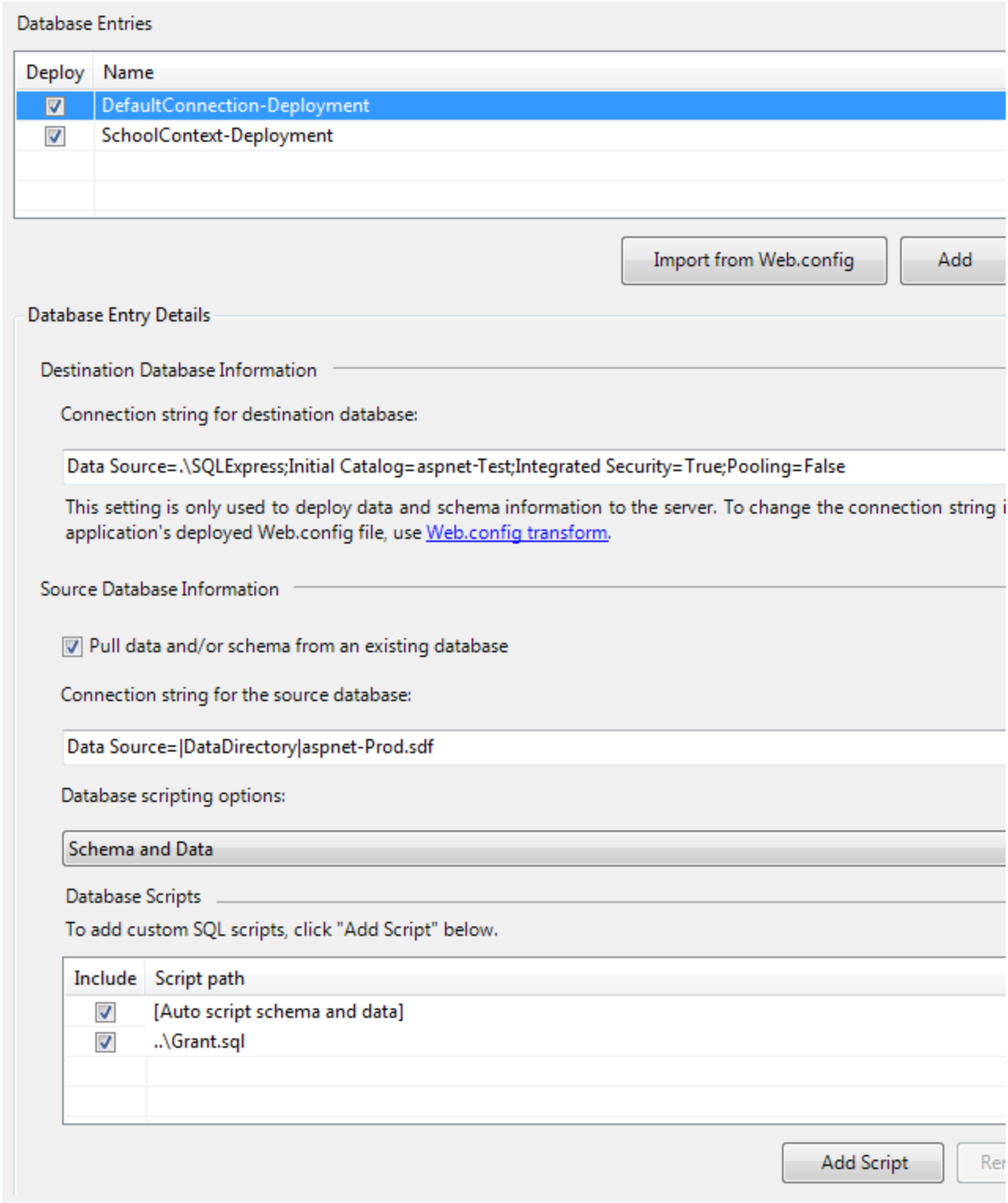
The **Connection string for the source database** value is extracted from the *Web.config* file and points to the development SQL Server Compact database. This is the source database that will be used to generate the scripts that will run later in the destination database. Since you want to deploy the production version of the database, change "aspnet-Dev.sdf" to "aspnet-Prod.sdf".

Change **Database scripting options** from **Schema Only** to **Schema and data**, since you want to copy your data (user accounts and roles) as well as the database structure.

To configure deployment to run the grant scripts that you created earlier, you have to add them to the **Database Scripts** section. Click **Add Script**, and in the **Add SQL Scripts** dialog box, navigate to the folder where you stored the grant script (this is the folder that contains your solution file). Select the file named *Grant.sql*, and click **Open**.



The settings for the **DefaultConnection-Deployment** row in **Database Entries** now look like the following illustration:



Configuring Deployment Settings for the School Database

Next, select the **SchoolContext-Deployment** row in the **Database Entries** table in order to configure deployment settings for the School database.

You can use the same method you used earlier to get the connection string for the new SQL Server Express database. Copy this connection string into **Connection string for destination database** in the **Package/Publish SQL** tab.

```
Data Source=.\SQLExpress;Initial Catalog=School-Test;Integrated
Security=True;Pooling=False
```

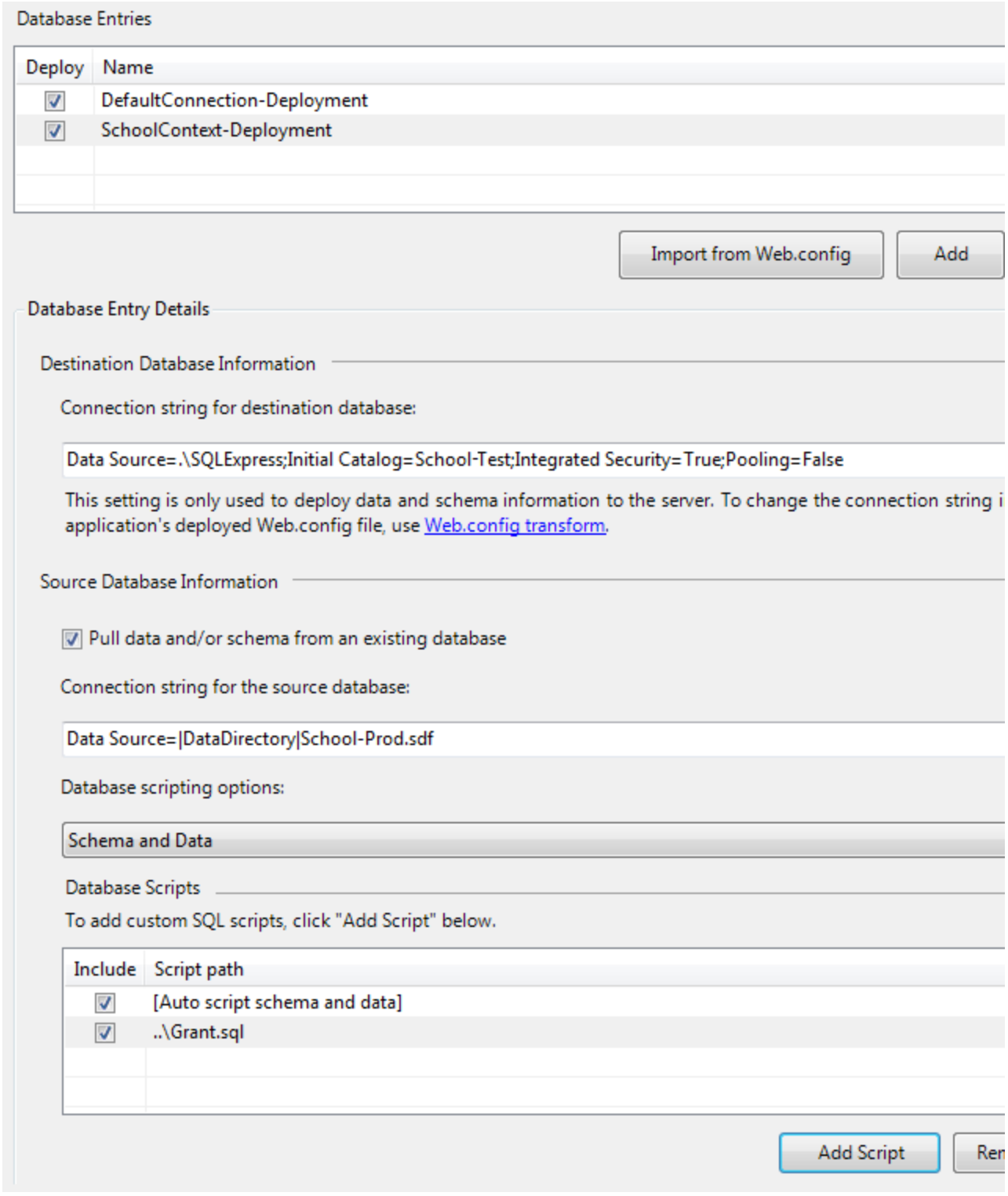
Make sure that **Pull data and/or schema from an existing database** is selected.

The **Connection string for the source database** value is extracted from the *Web.config* file and points to the development SQL Server Compact database. Change "School-Dev.sdf" to "School-Prod.sdf" to deploy the production version of the database. (You never created a School-Prod.sdf file in the App_Data folder, so you'll copy that file from the test environment to the App_Data folder in the ContosoUniversity project folder later.)

Change **Database scripting options** to **Schema and data**.

You also want to run the script to grant read and write permission for this database to the application pool identity, so add the *Grant.sql* script file as you did for the membership database.

When you're done, the settings for the **SchoolContext-Deployment** row in **Database Entries** look like the following illustration:



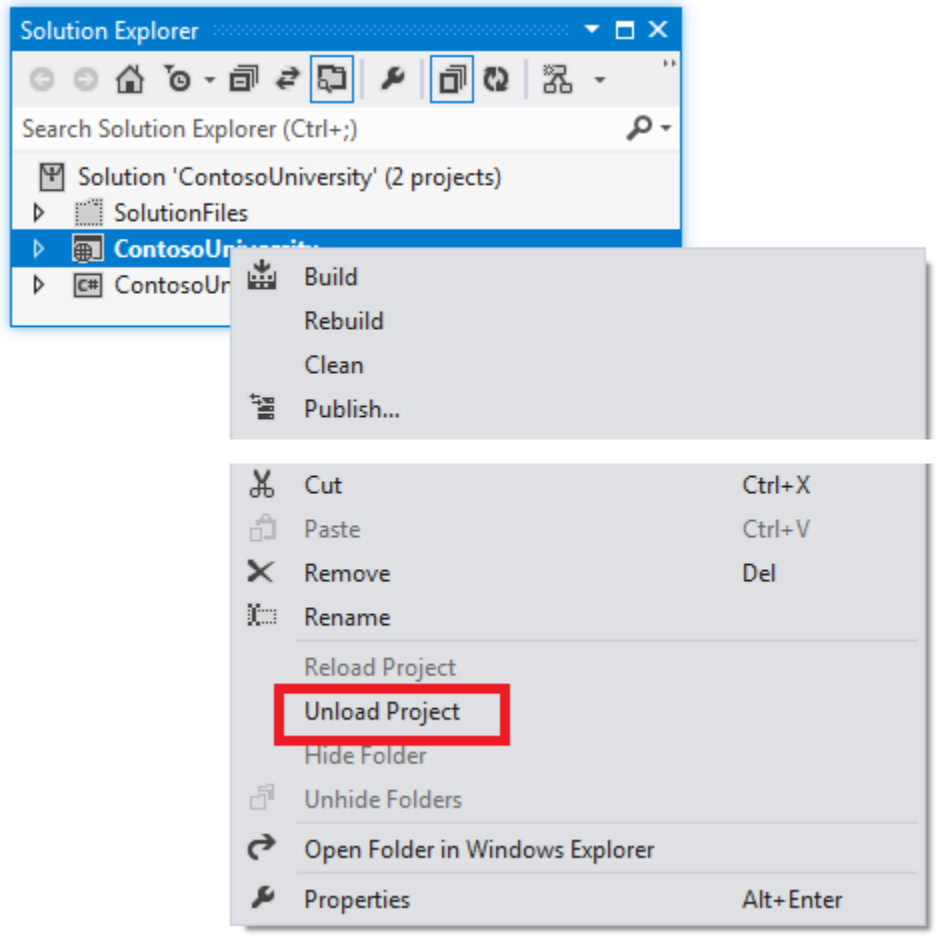
Save the changes to the **Package/Publish SQL** tab.

Copy the *School-Prod.sdf* file from the *c:\inetpub\wwwroot\ContosoUniversity\App_Data* folder to the *App_Data* folder in the ContosoUniversity project.

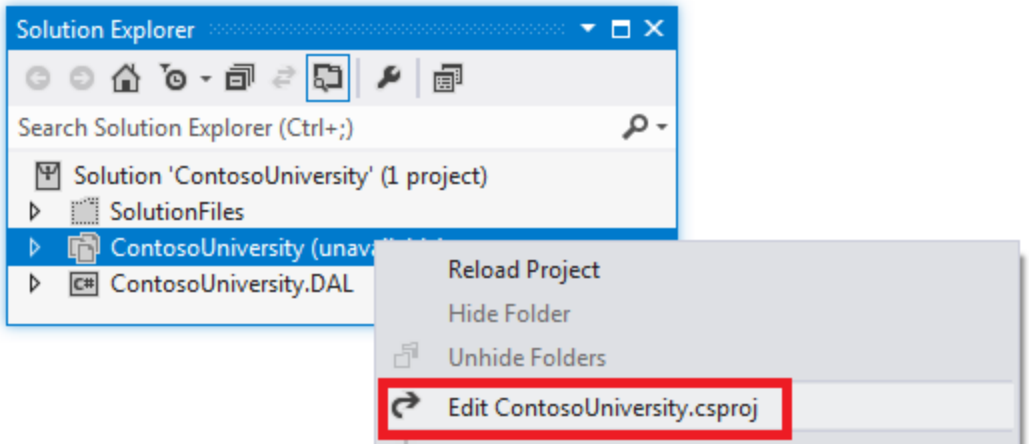
Specifying Transacted Mode for the Grant Script

The deployment process generates scripts that deploy the database schema and data. By default, these scripts run in a transaction. However, custom scripts (like the grant scripts) by default do not run in a transaction. If the deployment process mixes transaction modes, you might get a timeout error when the scripts run during deployment. In this section, you edit the project file in order to configure the custom scripts to run in a transaction.

In **Solution Explorer**, right-click the **ContosoUniversity** project and select **Unload Project**.



Then right-click the project again and select **Edit ContosoUniversity.csproj**.



The Visual Studio editor shows you the XML content of the project file. Notice that there are several `PropertyGroup` elements. (In the image, the contents of the `PropertyGroup` elements have been omitted.)

```
 1   <?xml version="1.0" encoding="utf-8"?>
 2  <Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.microsof
 3    <PropertyGroup>
17    </PropertyGroup>
18    <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
26    </PropertyGroup>
27    <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' '
58    </PropertyGroup>
59    <PropertyGroup Condition="'$(Configuration)|$(Platform)' == 'Test|AnyCPU'">
99    </PropertyGroup>
```

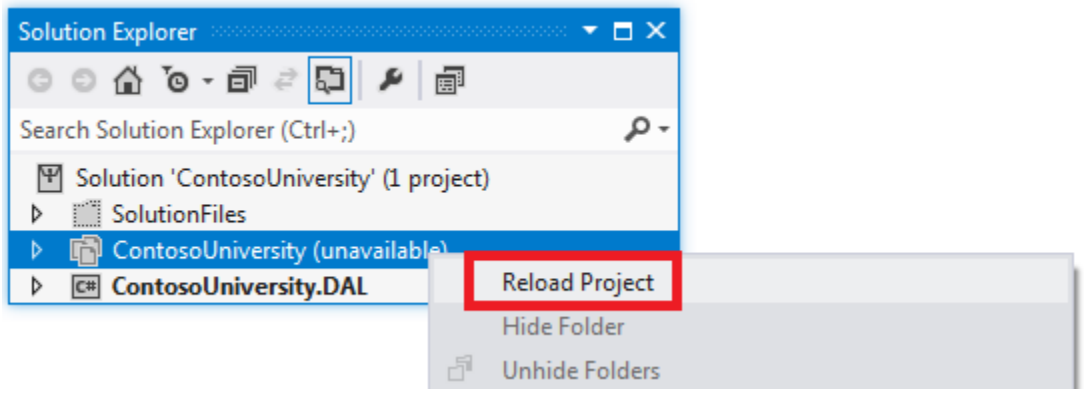
The first one, which has no `Condition` attribute, is for settings that apply regardless of build configuration. One `PropertyGroup` element applies only to the Debug build configuration (note the `Condition` attribute), one applies only to the Release build configuration, and one applies only to the Test build configuration. Within the `PropertyGroup` element for the Release build configuration, you'll see a `PublishDatabaseSettings` element that contains the settings you entered on the **Package/Publish SQL** tab. There is an `Object` element that corresponds to each of the grant scripts you specified (notice the two instances of "Grant.sql"). By default, the `Transacted` attribute of the `Source` element for each grant script is `False`.

```
<PublishDatabaseSettings>
  <Objects>
    <ObjectGroup Name="DefaultConnection-Deployment" Order="1" Enabled="True">
      <Destination Path="Data Source=localhost\SQLExpress%3bInitial Catalog=asp
      <Object Type="dbFullSql" Enabled="True">
        <PreSource Path="data source=.\SQLEXPRESS%3bIntegrated Security=SSPI%3b
        <Source Path="obj\Test\AutoScripts\DefaultConnection-Deployment_SchemaA
      </Object>
      <Object Type="dbFullSql" xmlns="">
        <Source Path="..\Grant.sql" Transacted="False" />
      </Object>
    </ObjectGroup>
    <ObjectGroup Name="SchoolContext-Deployment" Order="2" Enabled="True">
      <Destination Path="Data Source=localhost\SQLExpress%3bInitial Catalog=Sch
      <Object Type="dbFullSql" Enabled="True">
        <PreSource Path="data source=.\SQLEXPRESS%3bIntegrated Security=SSPI%3b
        <Source Path="obj\Test\AutoScripts\SchoolContext-Deployment_SchemaAndDa
      </Object>
      <Object Type="dbFullSql" xmlns="">
        <Source Path="..\Grant.sql" Transacted="False" />
      </Object>
    </ObjectGroup>
  </Objects>
</PublishDatabaseSettings>
```

Change the value of the `Transacted` attribute of the `Source` element to `True`.

```
<PublishDatabaseSettings>
  <Objects>
    <ObjectGroup Name="DefaultConnection-Deployment" Order="1" Enabled="True">
      <Destination Path="Data Source=localhost\SQLExpress%3bInitial Catalog=as
      <Object Type="dbFullSql" Enabled="True">
        <PreSource Path="data source=.\SQLEXPRESS%3bIntegrated Security=SSPI%3
        <Source Path="obj\Test\AutoScripts\DefaultConnection-Deployment_Schema
      </Object>
      <Object Type="dbFullSql" xmlns="">
        <Source Path="..\Grant.sql" Transacted="True" />
      </Object>
    </ObjectGroup>
    <ObjectGroup Name="SchoolContext-Deployment" Order="2" Enabled="True">
      <Destination Path="Data Source=localhost\SQLExpress%3bInitial Catalog=Sc
      <Object Type="dbFullSql" Enabled="True">
        <PreSource Path="data source=.\SQLEXPRESS%3bIntegrated Security=SSPI%3
        <Source Path="obj\Test\AutoScripts\SchoolContext-Deployment_SchemaAndD
      </Object>
      <Object Type="dbFullSql" xmlns="">
        <Source Path="..\Grant.sql" Transacted="True" />
      </Object>
    </ObjectGroup>
  </Objects>
</PublishDatabaseSettings>
```

Save and close the project file, and then right-click the project in **Solution Explorer** and select **Reload Project**.



Setting up Web.Config Transformations for the Connection Strings

The connection strings for the new SQL Express databases that you entered on the **Package/Publish SQL** tab are used by Web Deploy only for updating the destination database during deployment. You still have to set up *Web.config* transformations so that the connection strings in the deployed *Web.config* file point to the new SQL Server Express databases. (When you use the **Package/Publish SQL** tab, you can't configure connection strings in the publish profile.)

Open *Web.Test.config* and replace the `connectionStrings` element with the `connectionStrings` element in the following example. (Make sure you only copy the connectionStrings element, not the surrounding code that is shown here to provide context.)

```
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
  <connectionStrings>
    <add name="DefaultConnection"
         connectionString="Data Source=.\SQLExpress;Initial Catalog=aspnet-
Test;Integrated Security=True;Pooling=False;MultipleActiveResultSets=True"
         providerName="System.Data.SqlClient"
         xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/>
    <add name="SchoolContext"
         connectionString="Data Source=.\SQLExpress;Initial Catalog=School-
Test;Integrated Security=True;Pooling=False;MultipleActiveResultSets=True"
         providerName="System.Data.SqlClient"
         xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/>
  </connectionStrings>
```

```
  <!-- appSettings element, comments, and system.web element -->
</configuration>
```

This code causes the `connectionString` and `providerName` attributes of each `add` element to be replaced in the deployed *Web.config* file. These connection strings are not identical to the ones you entered in the **Package/Publish SQL** tab. The setting "MultipleActiveResultSets=True" has been added to them because it's required for the Entity Framework and the Universal Providers.

Installing SQL Server Compact

The SqlServerCompact NuGet package provides the SQL Server Compact database engine assemblies for the Contoso University application. But now it is not the application but Web Deploy that must be able to read the SQL Server Compact databases, in order to create scripts to run in the SQL Server databases. To enable Web Deploy to read SQL Server Compact databases, install SQL Server Compact on the development computer by using the following link: Microsoft SQL Server Compact 4.0.

Deploying to the Test Environment

In order to publish to the Test environment, you have to create a publish profile that is configured to use the **Package/Publish SQL** tab for database publishing instead of the publish profile database settings.

First, delete the existing Test profile.

In **Solution Explorer**, right-click the ContosoUniversity project, and click **Publish**.

Select the **Profile** tab.

Click **Manage Profiles**.

Select **Test**, click **Remove**, and then click **Close**.

Close the **Publish Web** wizard to save this change.

Next, create a new Test profile and use it to publish the project.

In **Solution Explorer**, right-click the ContosoUniversity project, and click **Publish**.

Select the **Profile** tab.

Select **<New...>** from the drop-down list, and enter "Test" as the profile name.

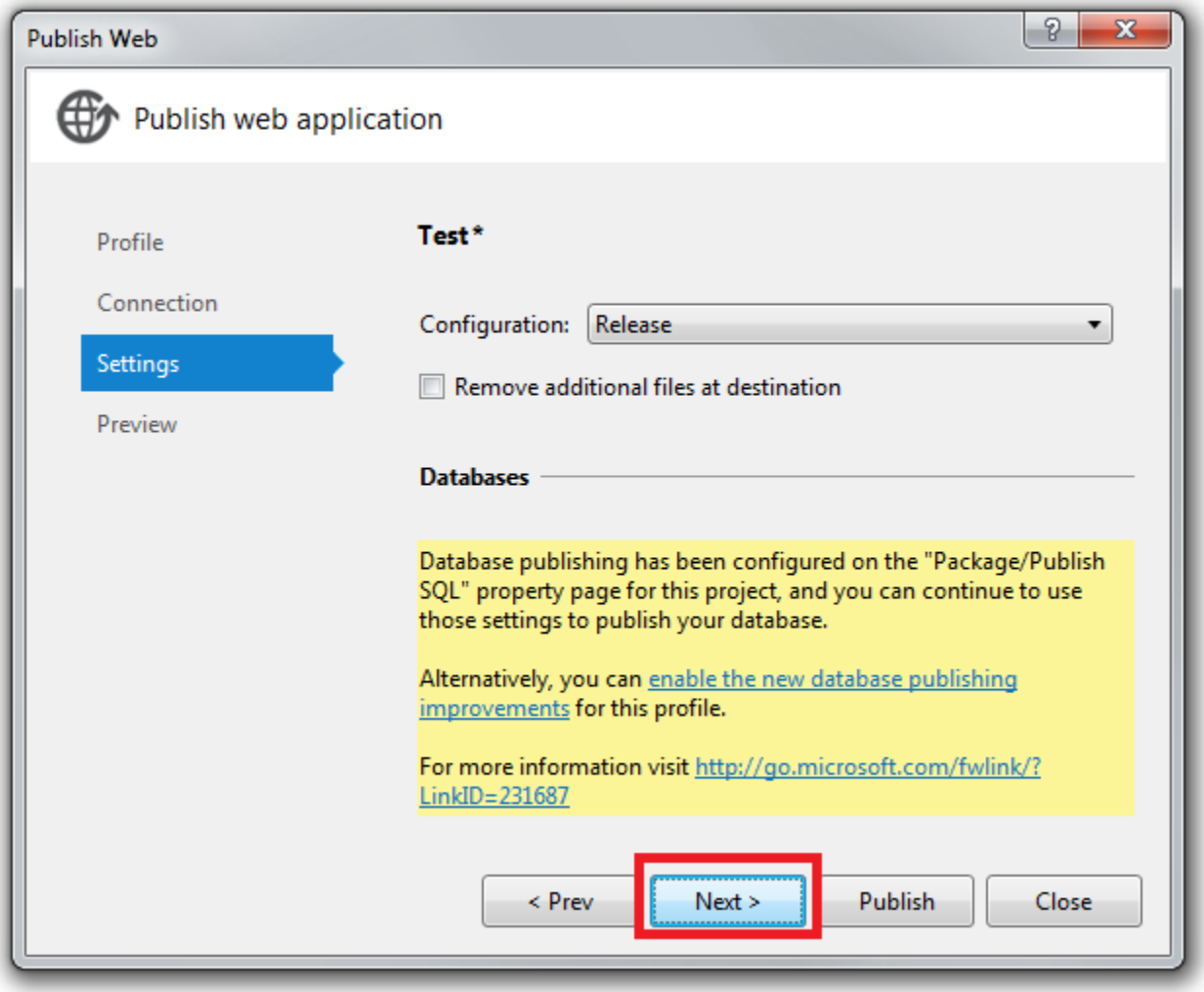
In the **Service URL** box, enter *localhost*.

In the **Site/application** box, enter *Default Web Site/ContosoUniversity*.

In the **Destination URL** box, enter *http://localhost/ContosoUniversity/*.

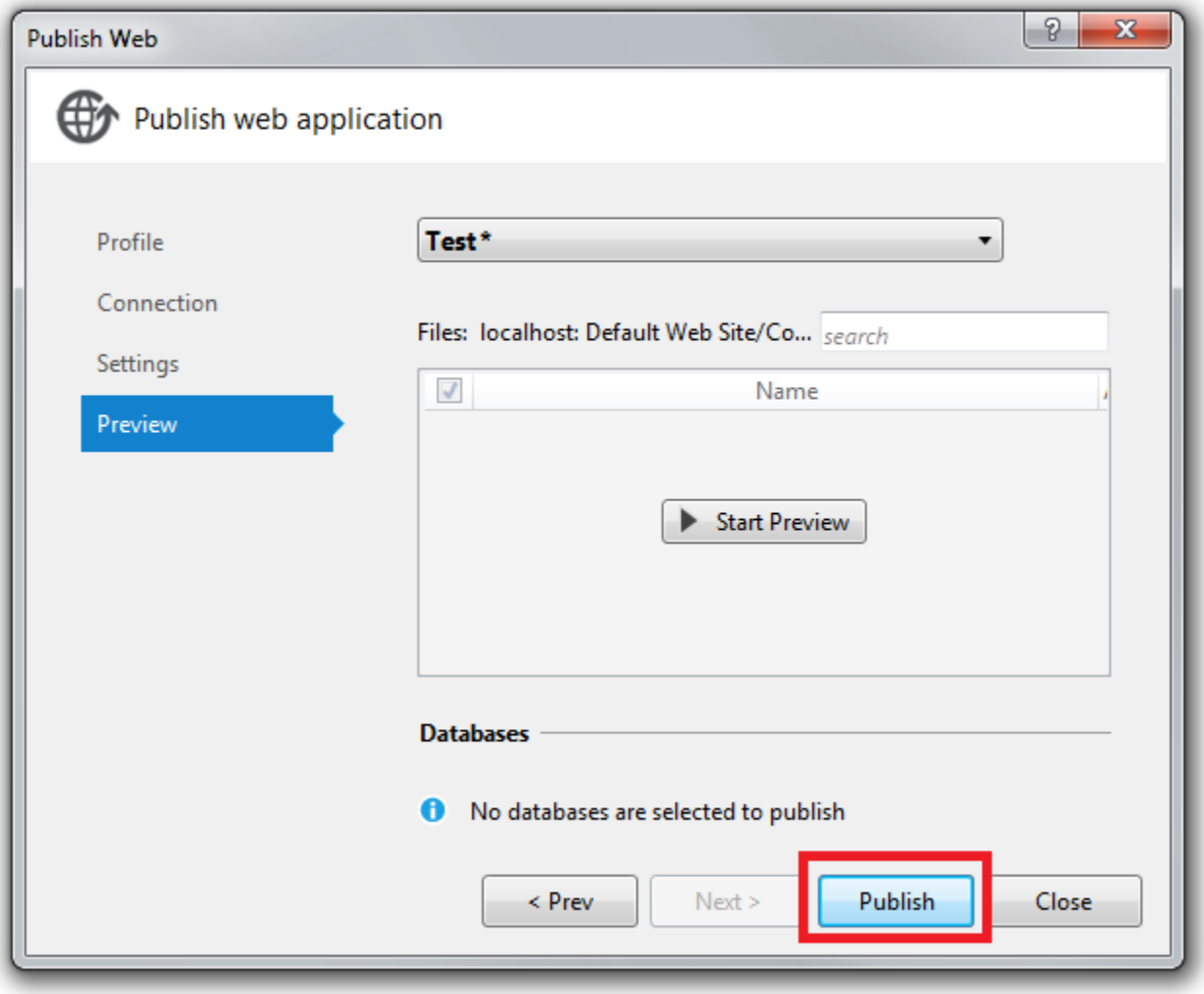
Click **Next**.

The **Settings** tab warns you that the **Package/Publish SQL** tab has been configured, and it gives you an opportunity to override them by clicking enable the new database publishing improvements. For this deployment you don't want to override the **Package/Publish SQL** tab settings, so just click **Next**.



A message on the **Preview** tab indicates that **No databases are selected to publish**, but this only means that database publishing is not configured in the publish profile.

Click **Publish**.



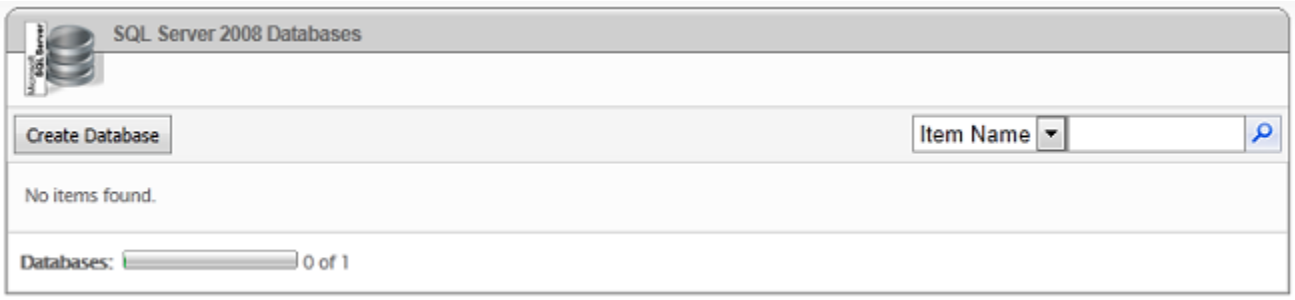
Visual Studio deploys the application and opens the browser to the home page of the site in the test environment. Run the Instructors page to see that it displays the same data that you saw earlier. Run the **Add Students** page, add a new student, and then view the new student in the **Students** page. This verifies that the you can update the database. Select the **Update Credits** page (you'll need to log in) to verify that the membership database was deployed and you have access to it.

Creating a SQL Server Database for the Production Environment

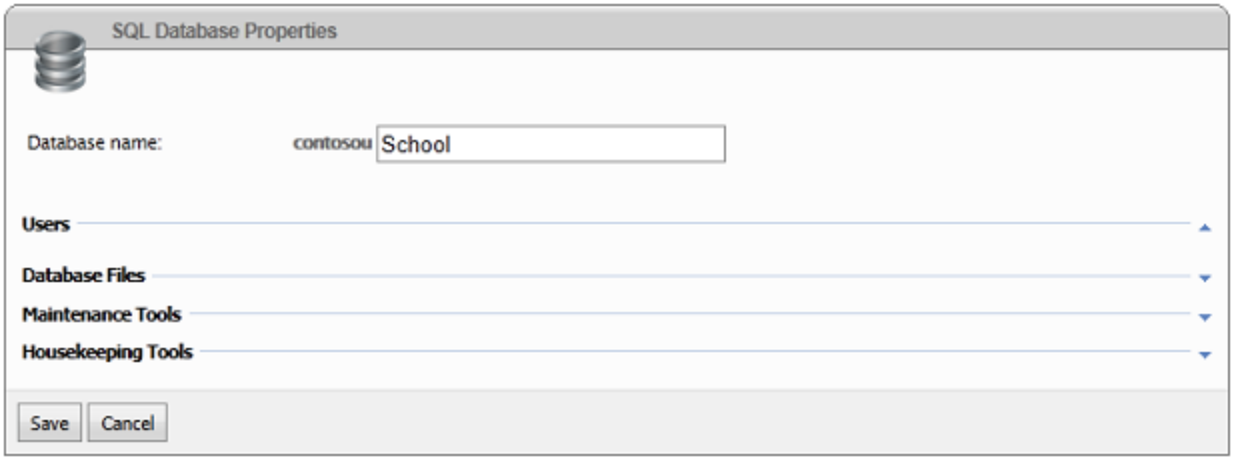
Now that you've deployed to the test environment, you're ready to set up deployment to production. You begin as you did for the test environment, by creating a database to deploy to. As you recall from the Overview, the Cytanium Lite hosting plan only allows a single SQL Server database, so you will set up only one database, not two. All of the tables and data from the membership and School SQL Server Compact databases will be deployed into one SQL Server database in production.

Go to the Cytanium control panel at http://panel.cytanium.com. Hold the mouse over **Databases** and then click **SQL Server 2008**.

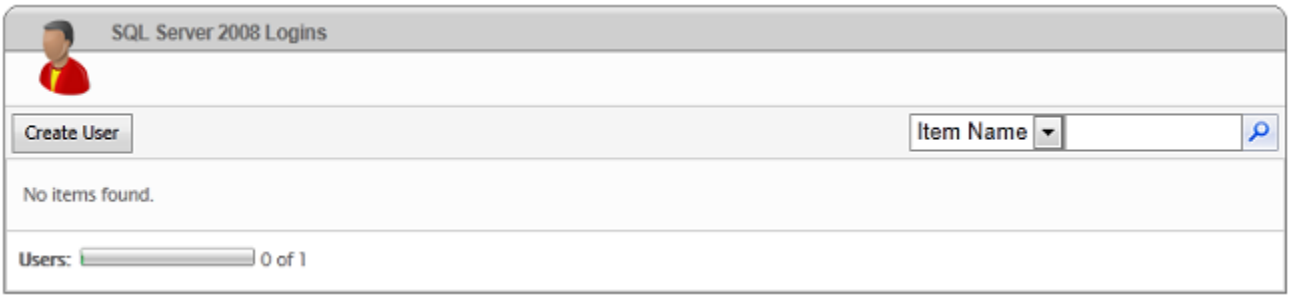
In the **SQL Server 2008** page, click **Create Database**.



Name the database "School" and click **Save**. (The page automatically adds the prefix "contosou", so the effective name will be "contosouSchool".)

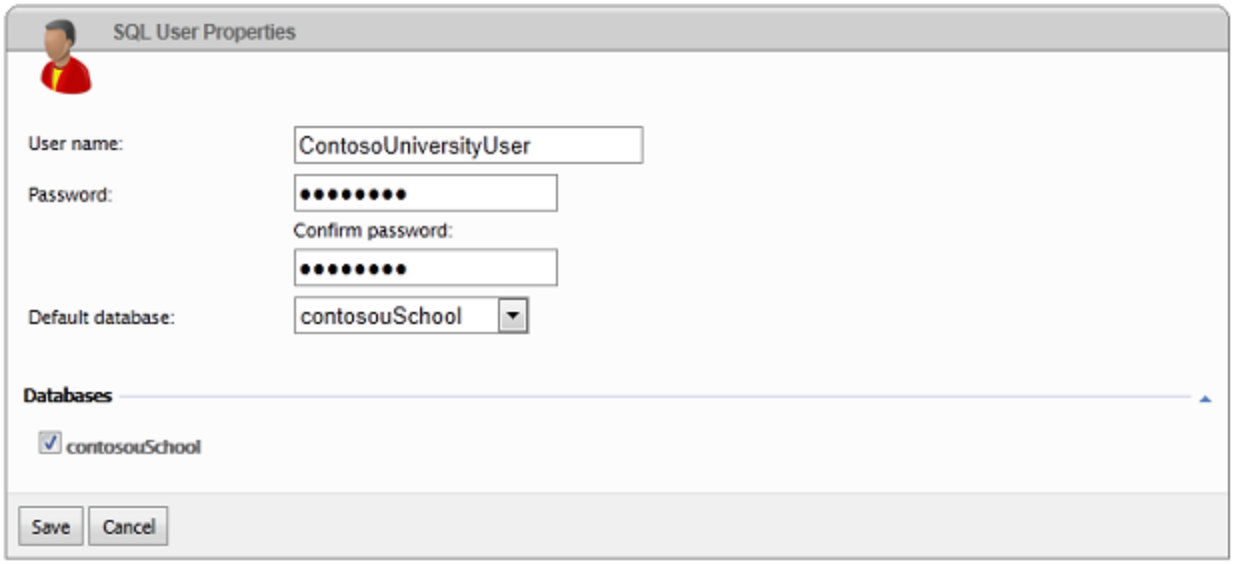


On the same page, click **Create User**. On Cytanium's servers, rather than using integrated Windows security and letting the application pool identity open your database, you'll create a user that has authority to open your database. You'll add the user's credentials to the connection strings that go in the production *Web.config* file. In this step you create those credentials.



Fill in the required fields in the **SQL User Properties** page:

- Enter "ContosoUniversityUser" as the name.
- Enter a password.
- Select **contosouSchool** as the default database.
- Select the **contosouSchool** check box.



Configuring Database Deployment for the Production Environment

Now you're ready to set up database deployment settings in the **Package/Publish SQL** tab, as you did earlier for the test environment.

Open the **Project Properties** window, select the **Package/Publish SQL** tab, and make sure that **Active (Release)** or **Release** is selected in the **Configuration** drop-down list.

When you configure deployment settings for each database, the key difference between what you do for production and test environments is in how you configure connection strings. For the test environment you entered different destination database connection strings, but for the production environment the destination connection string will be the same for both databases. This is because you are deploying both databases to one database in production.

Configuring Deployment Settings for the Membership Database

To configure settings that apply to the membership database, select the **DefaultConnection-Deployment** row in the **Database Entries** table.

In **Connection string for destination database**, enter a connection string that points to the new production SQL Server database that you just created. You can get the connection string from your welcome email. The relevant part of the email contains the following sample connection string:

```
Data Source=vserver01.cytanium.com;Initial Catalog={myDataBase};User
Id={myUsername};Password={myPassword};
```

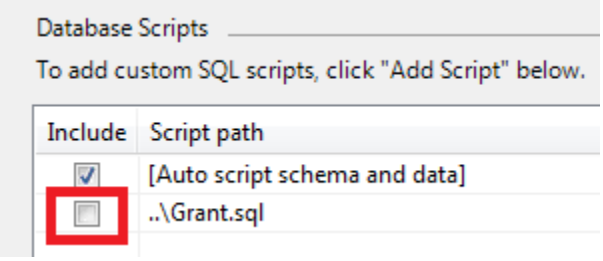
After you replace the three variables, the connection string you need looks like this example:

```
Data Source=vserver01.cytanium.com;Initial Catalog=contosouSchool;User
Id=ContosoUniversityUser;Password=Password;
```

Copy and paste this connection string into **Connection string for destination database** in the **Package/Publish SQL** tab.

Make sure that **Pull data and/or schema from an existing database** is still selected, and that **Database scripting options** is still **Schema and Data**.

In the **Database Scripts** box, clear the check box next to the Grant.sql script.



Configuring Deployment Settings for the School Database

Next, select the **SchoolContext-Deployment** row in the **Database Entries** table in order to configure the School database settings.

Copy the same connection string into **Connection string for destination database** that you copied into that field for the membership database.

Make sure that **Pull data and/or schema from an existing database** is still selected, and that **Database scripting options** is still **Schema and Data**.

In the **Database Scripts** box, clear the check box next to the Grant.sql script.

Save the changes to the **Package/Publish SQL** tab.

Setting Up Web.Config Transforms for the Connection Strings to Production Databases

Next, you'll set up *Web.config* transformations so that the connection strings in the deployed *Web.config* file to point to the new production database. The connection string that you entered on the **Package/Publish SQL** tab for Web Deploy to use is the same as the one the application needs to use, except for the addition of the `MultipleResultSets` option.

Open *Web.Production.config* and replace the `connectionStrings` element with a `connectionStrings` element that looks like the following example. (Only copy the `connectionStrings` element, not the surrounding tags that are provided to show the context.)

```
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
  <connectionStrings>
    <add name="DefaultConnection"
         connectionString="Data Source=vserver01.cytanium.com;Initial
Catalog=contosouSchool;User
Id=ContosoUniversityUser;Password=Password;MultipleActiveResultSets=True"
         providerName="System.Data.SqlClient"
         xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/>
    <add name="SchoolContext"
         connectionString="Data Source=vserver01.cytanium.com;Initial
Catalog=contosouSchool;User
Id=ContosoUniversityUser;Password=Password;MultipleActiveResultSets=True"
         providerName="System.Data.SqlClient"
         xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/>
  </connectionStrings>
  <!-- appSettings element, comments, and system.web element -->
</configuration>
```

You sometimes see advice that tells you to always encrypt connection strings in the *Web.config* file. This might be appropriate if you were deploying to servers on your own company's network. When you are deploying to a shared hosting environment, though, you're trusting the security practices of the hosting provider, and it's not necessary or practical to encrypt the connection strings.

Deploying to the Production Environment

Now you're ready to deploy to production. Web Deploy will read the SQL Server Compact databases in your project's *App_Data* folder and re-create all of their tables and data in the production SQL Server database. In order to publish by using the **Package/Publish Web** tab settings, you have to create a new publish profile for production.

First, delete the existing Production profile as you did the Test profile earlier.

In **Solution Explorer**, right-click the ContosoUniversity project, and click **Publish**.

Select the **Profile** tab.

Click **Manage Profiles**.

Select **Production**, click **Remove**, and then click **Close**.

Close the **Publish Web** wizard to save this change.

Next, create a new Production profile and use it to publish the project.

In **Solution Explorer**, right-click the ContosoUniversity project, and click **Publish**.

Select the **Profile** tab.

Click **Import**, and select the .publishsettings file that you downloaded earlier.

On the **Connection** tab, change the **Destination URL** to the correct temporary URL, which in this example is http://contosouniversity.com.vserver01.cytanium.com.

Rename the profile to Production. (Select the **Profile** tab and click **Manage Profiles** to do that).

Close the **Publish Web** wizard to save your changes.

In a real application in which the database was being updated in production, you would do two additional steps now before you publish:

1. Upload *app_offline.htm*, as shown in the Deploying to the Production Environment tutorial.
2. Use the **File Manager** feature of the Cytanium control panel to copy the *aspnet-Prod.sdf* and *School-Prod.sdf* files from the production site to the *App_Data* folder of the ContosoUniversity project. This ensures that the data you're deploying to the new SQL Server database includes the latest updates made by your production website.

In the **Web One Click Publish** toolbar, make sure that the **Production** profile is selected, and then click **Publish**.

If you uploaded *app_offline.htm* before publishing, you have to use the **File Manager** utility in the Cytanium control panel to delete *app_offline.*htm before you test. You can also at the same time delete the *.sdf* files from the *App_Data* folder.

You can now open a browser and go to the URL of your public site to test the application the same way you did after deploying to the test environment.

Switching to SQL Server Express LocalDB in Development

As was explained in the Overview, it's generally best to use the same database engine in development that you use in test and production. (Remember that the advantage to using SQL Server Express in development is that the database will work the same in your development, test, and production environments.) In this section you'll set up the ContosoUniversity project to use SQL Server Express LocalDB when you run the application from Visual Studio.

The simplest way to perform this migration is to let Code First and the membership system create both new development databases for you. Using this method to migrate requires three steps:

1. Change the connection strings to specify new SQL Express LocalDB databases.
2. Run the Web Site Administration Tool to create an administrator user. This creates the membership database.
3. Use the Code First Migrations update-database command to create and seed the application database.

Updating Connection Strings in the Web.config file

Open the *Web.config* file and replace the `connectionStrings` element with the following code:

```
<connectionStrings>
  <add name="DefaultConnection" connectionString="Data
Source=(LocalDb)\v11.0;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|\aspnet-
Dev.mdf;MultipleActiveResultSets=True;" providerName="System.Data.SqlClient" />
  <add name="SchoolContext" connectionString="Data Source=(LocalDb)\v11.0;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|\School-
Dev.mdf;MultipleActiveResultSets=True;" providerName="System.Data.SqlClient" />
</connectionStrings>
```

Creating the Membership Database

In **Solution Explorer**, select the ContosoUniversity project, and then click **ASP.NET Configuration** in the **Project** menu.

Select the Security tab.

Click **Create or Manage Roles**, and then create an **Administrator** role.

Return to the Security tab.

Click **Create user**, and then select the **Administrator** check box and create a user named admin.

Close the **Web Site Administration Tool**.

Creating the School Database

Open the Package Manager Console window.

In the **Default project** drop-down list, select the ContosoUniversity.DAL project.

Enter the following command:

```
update-database
```

Code First Migrations applies the Initial migration that creates the database and then applies the AddBirthDate migration, then it runs the Seed method.

Run the site by pressing Control-F5. As you did for the test and production environments, run the **Add Students** page, add a new student, and then view the new student in the **Students** page. This verifies that the School database was created and initialized and that you have read and write access to it.

Select the **Update Credits** page and log in to verify that the membership database was deployed and that you have access to it. If you did not migrate your user accounts, create an administrator account and then select the **Update Credits** page to verify that it works.

Cleaning Up SQL Server Compact Files

You no longer need files and NuGet packages that were included to support SQL Server Compact. If you want (this step is not required), you can clean up unneeded files and references.

In **Solution Explorer**, delete the *.sdf* files from the *App_Data* folder and the *amd64* and *x86* folders from the *bin* folder.

In **Solution Explorer**, right-click the solution (not one of the projects), and then click **Manage NuGet Packages for Solution**.

In the left pane of the **Manage NuGet Packages** dialog box, select **Installed packages**.

Select the **EntityFramework.SqlServerCompact** package and click **Manage**.

In the **Select Projects** dialog box, both projects are selected. To uninstall the package in both projects, clear both check boxes, then click **OK**.

In the dialog box that asks if you want to uninstall the dependent packages also, click No. One of these is the Entity Framework package that you have to keep.

Follow the same procedure to uninstall the **SqlServerCompact** package. (The packages must be uninstalled in this order because the **EntityFramework.SqlServerCompact** package depends on the **SqlServerCompact** package.)

You have now successfully migrated to SQL Server Express and full SQL Server. In the next tutorial you'll make another database change, and you'll see how to deploy database changes when your test and production databases use SQL Server Express and full SQL Server.

Deploying a SQL Server Database Update - 11 of 12

Overview

This tutorial shows you how to deploy a database update to a full SQL Server database. Because Code First Migrations does all the work of updating the database, the process is almost identical to what you did for SQL Server Compact in the Deploying a Database Update tutorial.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Adding a New Column to a Table

In this section of the tutorial you'll make a database change and corresponding code changes, then test them in Visual Studio in preparation for deploying them to the test and production environments. The change involves adding an `OfficeHours` column to the `Instructor` entity and displaying the new information in the **Instructors** web page.

In the ContosoUniversity.DAL project, open *Instructor.cs* and add the following property between the `HireDate` and `Courses` properties:

```
 [MaxLength(50)]
public string OfficeHours { get; set; }
```

Update the initializer class so that it seeds the new column with test data. Open *Migrations\Configuration.cs* and replace the code block that begins `var instructors = new List<Instructor>` with the following code block which includes the new column:

```
var instructors = new List<Instructor>
{
    new Instructor { FirstMidName = "Kim",     LastName = "Abercrombie", HireDate =
DateTime.Parse("1995-03-11"), BirthDate = DateTime.Parse("1918-08-12"), OfficeHours =
"8-9AM, 4-5PM", OfficeAssignment = new OfficeAssignment { Location = "Smith 17" } },
    new Instructor { FirstMidName = "Fadi",    LastName = "Fakhouri",    HireDate =
DateTime.Parse("2002-07-06"), BirthDate = DateTime.Parse("1960-03-15"),
OfficeAssignment = new OfficeAssignment { Location = "Gowan 27" } },
```

```
    new Instructor { FirstMidName = "Roger",   LastName = "Harui",       HireDate =
DateTime.Parse("1998-07-01"), BirthDate = DateTime.Parse("1970-01-11"), OfficeHours =
"6AM-6PM", OfficeAssignment = new OfficeAssignment { Location = "Thompson 304" } },
    new Instructor { FirstMidName = "Candace", LastName = "Kapoor",      HireDate =
DateTime.Parse("2001-01-15"), BirthDate = DateTime.Parse("1975-04-11") },
    new Instructor { FirstMidName = "Roger",   LastName = "Zheng",       HireDate =
DateTime.Parse("2004-02-12"), BirthDate = DateTime.Parse("1957-10-12"), OfficeHours =
"By appointment only" }
};
```

In the ContosoUniversity project, open *Instructors.aspx* and add a new template field for office hours just before the closing `</Columns>` tag in the first `GridView` control:

```
<asp:TemplateField HeaderText="Office Hours">
    <ItemTemplate>
        <asp:Label ID="InstructorOfficeHoursLabel" runat="server" Text='<%#
Eval("OfficeHours") %>'></asp:Label>
    </ItemTemplate>
    <EditItemTemplate>
        <asp:TextBox ID="InstructorOfficeHoursTextBox" runat="server" Text='<%#
Bind("OfficeHours") %>'
            Width="14em"></asp:TextBox>
    </EditItemTemplate>
</asp:TemplateField>
```

Build the solution.

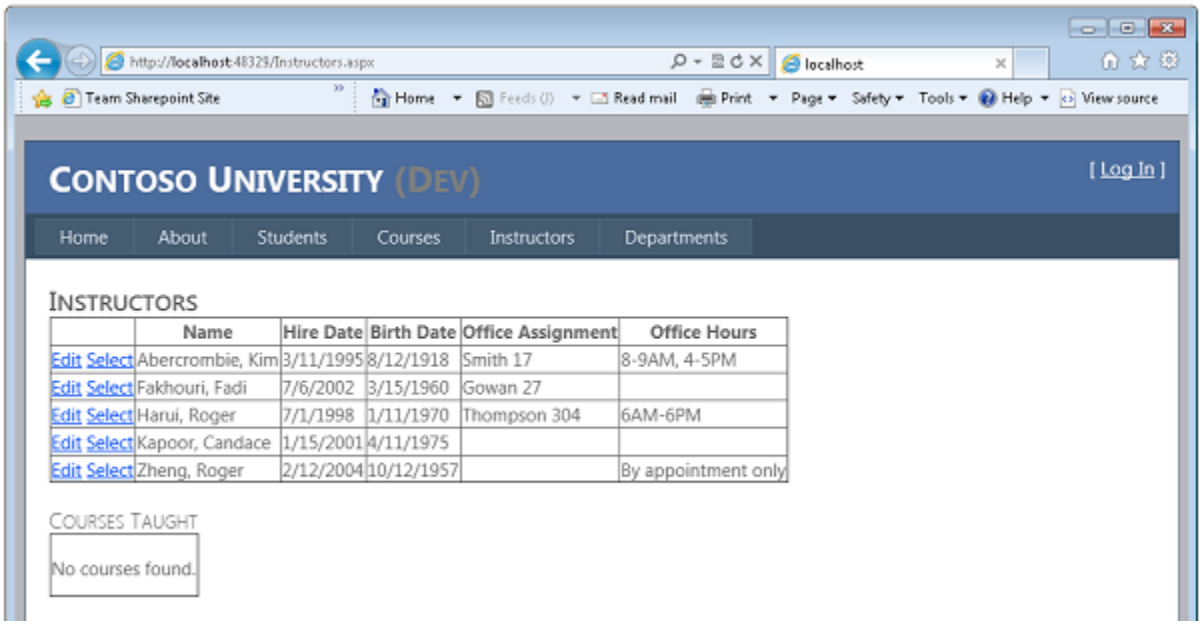
Open the **Package Manager Console** window, and select ContosoUniversity.DAL as the **Default project**.

Enter the following commands:

```
add-migration AddOfficeHoursColumn

update-database
```

Run the application and select the **Instructors** page. The page takes a little longer than usual to load, because the Entity Framework re-creates the database and seeds it with test data.



Deploying the Database Update to the Test Environment

When you use Code First Migrations, the method for deploying a database change to SQL Server is the same as for SQL Server Compact. However, you have to change the Test publish profile because it is still set up to migrate from SQL Server Compact to SQL Server.

The first step is to remove the connection string transformations that you created in the previous tutorial. These are no longer needed because you'll specify connection string transformations in the publish profile, as you did before you configured the **Package/Publish SQL** tab for migration to SQL Server.

Open the *Web.Test.config* file and remove the `connectionStrings` element. The only remaining transformation in the *Web.Test.config* file is for the `Environment` value in the `appSettings` element.

Now you can update the publish profile and publish to the test environment.

Open the **Publish Web** wizard, and then switch to the **Profile** tab.

Select the **Test** publish profile.

Select the **Settings** tab.

Click **enable the new database publishing improvements**.

In the connection string box for **SchoolContext**, enter the same value that you used in the *Web.Test.config* transformation file in the previous tutorial:

```
Data Source=.\SQLExpress;Initial Catalog=School-Test;Integrated
Security=True;Pooling=False;MultipleActiveResultSets=True
```

Select **Execute Code First Migrations (runs on application start)**. (In your version of Visual Studio, the check box might be labeled **Apply Code First Migrations**.)

In the connection string box for **DefaultConnection**, enter the same value that you used in the *Web.Test.config* transformation file in the previous tutorial:

```
Data Source=.\SQLExpress;Initial Catalog=aspnet-Test;Integrated
Security=True;Pooling=False;MultipleActiveResultSets=True
```

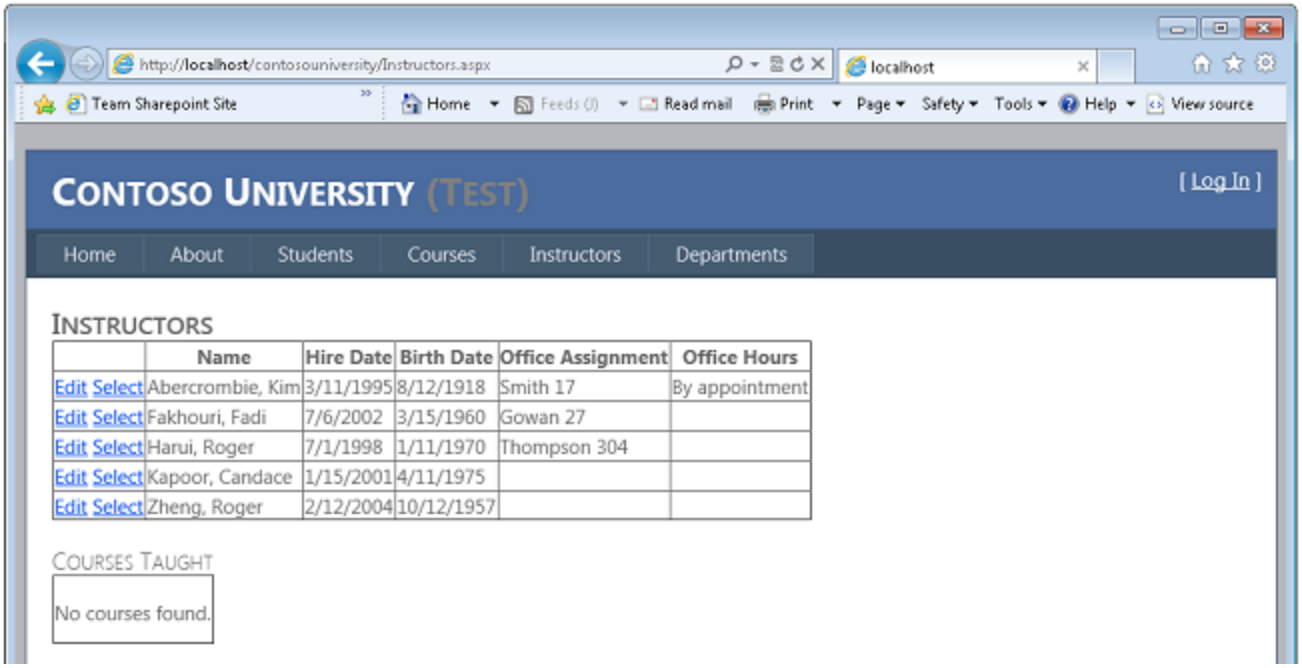
Leave **Update database** cleared.

Click **Publish**.

Visual Studio deploys the code changes to the test environment and opens the browser to the Contoso University home page.

Select the Instructors page.

When the application runs this page, it tries to access the database. Code First Migrations checks if the database is current, and finds that the latest migration has not been applied yet. Code First Migrations applies the latest migration, runs the `Seed` method, and then the page runs normally. You see the new Office Hours column with the seeded data.



Deploying the Database Update to the Production Environment

You have to change the publish profile for the production environment also. In this case you'll remove the existing profile and create a new one by importing an updated .publishsettings file. The updated file will include the connection string for the SQL Server database at Cytanium.

As you saw when you deployed to the test environment, you no longer need connection string transforms in the *Web.Production.config* transformation file. Open that file and remove the `connectionStrings` element. The remaining transformations are for the `Environment` value in the `appSettings` element and the `location` element that restricts access to Elmah error reports.

Before you create a new publish profile for production, download an updated .publishsettings file the same way you did earlier in the Deploying to the Production Environment tutorial. (In the Cytanium control panel, click **Web Sites**, and then click the **contosouniversity.com** website. Select the **Web Publishing** tab, and then click **Download Publishing Profile for this web site**.) The reason you are doing this is to pick up the database connection string in the .publishsettings file. The connection string wasn't available the first time you downloaded the file, because you were still using SQL Server Compact and hadn't created the SQL Server database at Cytanium yet.

Now you can update the publish profile and publish to the production environment.

Open the **Publish Web** wizard, and then switch to the **Profile** tab.

Click **Manage Profiles**, and then delete the Production profile.

Close the **Publish Web** wizard to save this change.

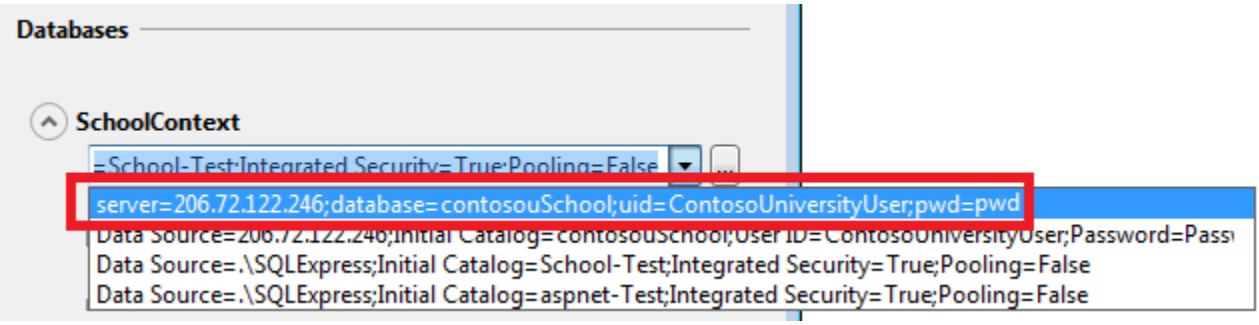
Open the **Publish Web** wizard again, and then click **Import**.

On the **Connection** tab, change **Destination URL** to the appropriate value if you are using a temporary URL.

Click **Next**.

On the **Settings** tab, click **enable the new database publishing improvements**.

In the connection string drop-down list for **SchoolContext**, select the Cytanium connection string.



Select **Execute Code First migrations (runs on application start)**.

In the connection string drop-down list for **DefaultConnection**, select the Cytanium connection string.

Select the **Profile** tab, click **Manage Profiles**, and rename the profile from "contosouniversity.com - Web Deploy" to "Production".

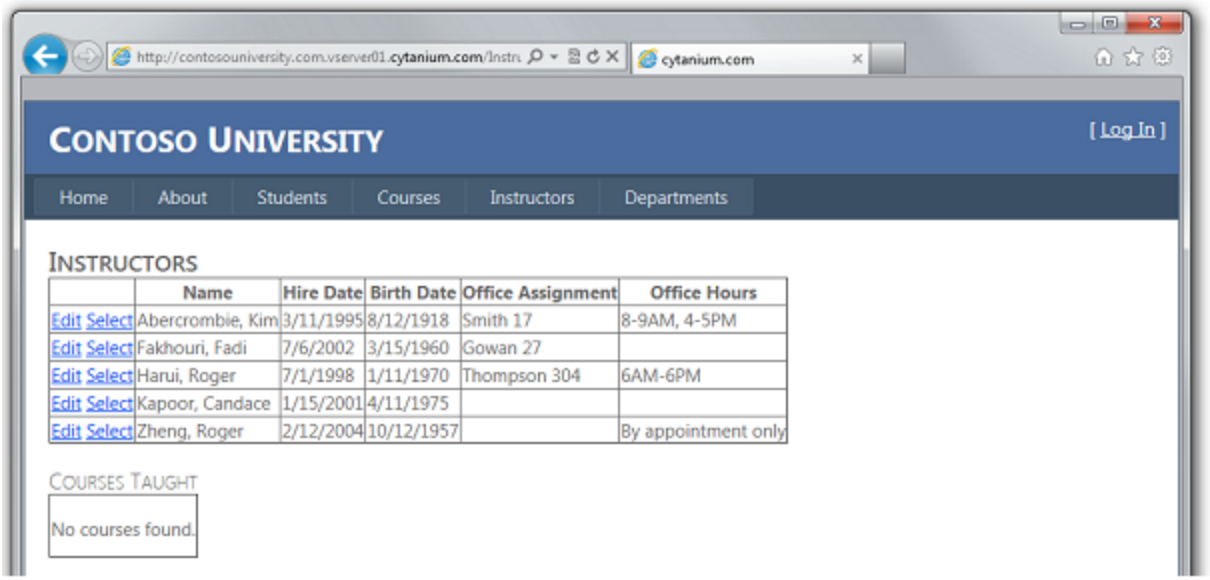
Close the publish profile to save the change, then open it again.

Click **Publish**. (For a real production website, you would copy *app_offline.htm* to production and put it in your project folder before publishing, then remove it when deployment is complete.)

Visual Studio deploys the code changes to the test environment and opens the browser to the Contoso University home page.

Select the Instructors page.

Code First Migrations updates the database the same way it did in the Test environment. You see the new Office Hours column with the seeded data.



You have now successfully deployed an application update that included a database change, using a SQL Server database.

More Information

This completes this series of tutorials on deploying an ASP.NET web application to a third-party hosting provider. For more information about any of the topics covered in these tutorials, see the ASP.NET Deployment Content Map on the MSDN web site.

Acknowledgements

I would like to thank the following people who made significant contributions to the content of this tutorial series:

- Alberto Poblacion, MVP & MCT, Spain
- Jarod Ferguson, Data Platform Development MVP, United States
- Harsh Mittal, Microsoft
- Kristina Olson, Microsoft
- Mike Pope, Microsoft
- Mohit Srivastava, Microsoft
- Raffaele Rialdi, Italy
- Rick Anderson, Microsoft
- Sayed Hashimi, Microsoft (twitter: @sayedihashimi)

- Scott Hanselman (twitter: @shanselman)
- Scott Hunter, Microsoft (twitter: @coolcsh)
- Srđan Božović, Serbia
- Vishal Joshi, Microsoft (twitter: @vishalrjoshi)

Troubleshooting (12 of 12)

This page describes some common problems that may arise when you deploy an ASP.NET web application by using Visual Studio. For each one, one or more possible causes and corresponding solutions are provided.

Server Error in '/' Application - Current Custom Error Settings Prevent Details of the Error from Being Viewed Remotely

Scenario

After deploying a site to a remote host, you get an error message that mentions the customErrors setting in the Web.config file but doesn't indicate what the actual cause of the error was:

```
Server Error in '/' Application.
Runtime Error

Description: An application error occurred on the server. The current custom error
settings
for this application prevent the details of the application error from being viewed
remotely
(for security reasons). It could, however, be viewed by browsers running on the local
server
machine.

Details: To enable the details of this specific error message to be viewable on
remote machines,
please create a <customErrors> tag within a "web.config" configuration file located
in the
root directory of the current web application. This <customErrors> tag should then
have its
"mode" attribute set to "Off".
```

Possible Cause and Solution

By default, ASP.NET shows detailed error information only when your web application is running on the local computer. Generally you don't want to display detailed error information when your web application is publicly available over the Internet, because hackers may be able to use this information to find vulnerabilities in the

application. However, when you are deploying a site or updates to a site, sometimes something will go wrong and you need to get the actual error message.

To enable the application to display detailed error messages when it runs on the remote host, edit the Web.config file to set `customErrors` mode off, redeploy the application, and run the application again:

1. If the application Web.config file has a `customErrors` element in the `system.web` element, change the `mode` attribute to "off". Otherwise add a `customErrors` element in the `system.web` element with the `mode` attribute set to "off", as shown in the following example:

```
<configuration>
  <system.web>
    <customErrors mode="off"/>
  </system.web>
</configuration>
```

2. Deploy the application.
3. Run the application and repeat whatever you did earlier that caused the error to occur. Now you can see what the actual error message is.
4. When you have resolved the error, restore the original `customErrors` setting and redeploy the application.

Access is Denied in a Web Page that Uses SQL Server Compact

Scenario

When you deploy a site that uses SQL Server Compact and you run a page in the deployed site that accesses the database, you see the following error message:

```
Access is denied. (Exception from HRESULT: 0x80070005 (E_ACCESSDENIED))
```

Possible Cause and Solution

The NETWORK SERVICE account on the server needs to be able to read SQL Service Compact native binaries that are in the *bin\amd64* or *bin\x86* folder, but it does not have read permissions for those folders. Set read permission for NETWORK SERVICE on the *bin* folder, making sure to extend the permissions to subfolders.

Cannot Read Configuration File Due to Insufficient Permissions

Scenario

When you click the Visual Studio publish button to deploy an application to IIS on your local machine, publishing fails and the **Output** window shows an error message similar to this:

```
An error occurred when reading the IIS Configuration File 'MACHINE/REDIRECTION'. The identity performing this operation was ... Error: Cannot read configuration file due to insufficient permissions.
```

Possible Cause and Solution

To use one-click publish to IIS on your local machine, you must be running Visual Studio with administrator permissions. Close Visual Studio and restart it with administrator permissions.

Could Not Connect to the Destination Computer ... Using the Specified Process

Scenario

When you click the Visual Studio publish button to deploy an application, publishing fails and the **Output** window shows an error message similar to this:

```
Web deployment task failed.(Could not connect to the destination computer ("<server URL>") using the specified process
("The Web Management Service"). This can happen if a proxy server is interrupting communication with the destination server.
Disable the proxy server and try again.) ... The remote server returned an error: (502) Bad Gateway.
```

Possible Cause and Solution

A proxy server is interrupting communication with the destination server. From the Windows Control Panel or in Internet Explorer, select **Internet Options** and select the **Connections** tab. In the **Internet Properties** dialog box, click **LAN Settings**. In the **Local Area Network (LAN) Settings** dialog box, clear the **Automatically detect settings** checkbox. Then click the publish button again.

If the problem persists, contact your system administrator to determine what can be done with proxy or firewall settings. The problem happens because Web Deploy uses a non-standard port for Web Management Service

deployment (8172); for other connections, Web Deploy uses port 80. When you are deploying to a third-party hosting provider, you are typically using the Web Management Service.

Default .NET 4.0 Application Pool Does Not Exist

Scenario

When you deploy an application that requires the .NET Framework 4, you see the following error message:

```
The default .NET 4.0 application pool does not exist or the application could not be added. Please verify that ASP.NET 4.0 is installed on this machine.
```

Possible Cause and Solution

ASP.NET 4 is not installed in IIS. If the server you are deploying to is your development computer and has Visual Studio 2010 installed on it, ASP.NET 4 is installed on the computer but might not be installed in IIS. On the server that you are deploying to, open an elevated command prompt and install ASP.NET 4 in IIS by running the following commands:

```
cd %windir%\Microsoft.NET\Framework\v4.0.30319
aspnet_regiis.exe -iru
```

You might also need to manually set the .NET Framework version of the default application pool. For more information, see the Deploying to IIS as a Test Environment tutorial.

Format of the initialization string does not conform to specification starting at index 0.

Scenario

After you deploy an application using one-click publish, when you run a page that accesses the database you get the following error message:

```
Format of the initialization string does not conform to specification starting at index 0.
```

Possible Cause and Solution

Open the *Web.config* file in the deployed site and check to see whether the connection string values begin with `$(ReplacableToken_` , as in the following example:

```
 <connectionStrings>
  <add name="DefaultConnection"
connectionString="$(ReplacableToken_DefaultConnection-Web.config Connection
String_0)" providerName="System.Data.SqlServerCe.4.0" />
  <add name="SchoolContext" connectionString="$(ReplacableToken_SchoolContext-
Web.config Connection String_0)" providerName="System.Data.SqlServerCe.4.0" />
</connectionStrings>
```

If the connection strings look like this example, edit the project file and add the following property to the `PropertyGroup` element that is for all build configurations:

```
<AutoParameterizationWebConfigConnectionStrings>False</AutoParameterizationWebConfigC
onnectionStrings>
```

Then redeploy the application.

HTTP 500 Internal Server Error

Scenario

When you run the deployed site, you see the following error message without specific information indicating the cause of the error:

```
HTTP Error 500 - Internal Server Error.
```

Possible Cause and Solution

There are many causes of 500 errors, but one possible cause if you are following these tutorials is that you put an XML element in the wrong place in one of the XML transformation files. For example, you would get this error if you put the transformation that inserts a `<location>` element under `<system.web>` instead of directly under `<configuration>`. The solution in that case is to correct the XML transformation file and redeploy.

HTTP 500.21 Internal Server Error

Scenario

When you run the deployed site, you see the following error message:

```
HTTP Error 500.21 - Internal Server Error. Handler "PageHandlerFactory-Integrated"
has a bad module "ManagedPipelineHandler" in its module list.
```

Possible Cause and Solution

The site you have deployed targets ASP.NET 4, but ASP.NET 4 is not registered in IIS on the server. On the server open an elevated command prompt and register ASP.NET 4 by running the following commands:

```
cd %windir%\Microsoft.NET\Framework\v4.0.30319
aspnet_regiis.exe –iru
```

You might also need to manually set the .NET Framework version of the default application pool. For more information, see the Deploying to IIS as a Test Environment tutorial.

Login Failed Opening SQL Server Express Database in App_Data

Scenario

You updated the *Web.config* file connection string to point to a SQL Server Express database as an *.mdf* file in your *App_Data* folder, and the first time you run the application you see the following error message:

```
System.Data.SqlClient.SqlException: Cannot open database "DatabaseName" requested by
the login. The login failed.
```

Possible Cause and Solution

The name of the *.mdf* file cannot match the name of any SQL Server Express database that has ever existed on your computer, even if you deleted the *.mdf* file of the previously existing database. Change the name of the *.mdf* file to a name that has never been used as a database name and change the *Web.config* file to use the new name. As an alternative, you can use SQL Server Management Studio Express to delete previously existing SQL Server Express databases.

Model Compatibility Cannot be Checked

Scenario

You updated the *Web.config* file connection string to point to a new SQL Server Express database, and the first time you run the application you see the following error message:

```
Model compatibility cannot be checked because the database does not contain model metadata. Ensure that IncludeMetadataConvention has been added to the DbModelBuilder conventions.
```

Possible Cause and Solution

If the database name you put in the Web.config file was ever used before on your computer, a database might already exist with some tables in it. Select a new name that has not been used on your computer before and change the *Web.config* file to point to use this new database name. As an alternative, you can use SQL Server Express Utility or SQL Server Management Studio Express to delete the existing database.

SQL Error When a Script Attempts to Create Users or Roles

Scenario

You are using database deployment configured on the **Package/Publish SQL** tab, SQL scripts that run during deployment include Create User or Create Role commands, and script execution fails when those commands are executed. You might see more detailed messages, such as the following:

```
The approximate location of the error was between lines '1' and '3' of the script.
The verbose log may have more information about the error. The command started with:
CREATE USER [user2] FOR LOGIN [user2] WITH DEFAULT
Error: User does not have permission to perform this action.
```

If this error occurs when you have configured database deployment in the **Publish Web** wizard rather than the **Package/Publish SQL** tab, create a thread in the Configuration and Deployment forum, and the solution will be added to this troubleshooting page.

Possible Cause and Solution

The user account you are using to perform deployment does not have permission to create users or roles. For example, the hosting company might assign the `db_datareader`, `db_datawriter`, and `db_ddladmin` roles to the user account that it sets up for you. These are sufficient for creating most database objects, but not for creating users or roles. One way to avoid the error is by excluding users and roles from database deployment.

You can do this by editing the `PreSource` element for the database's automatically generated script so that it includes the following attributes:

```
CopyAllUsers=false, CopyAllRoles=false
```

For information about how to edit the `PreSource` element in the project file, see How to: Edit Deployment Settings in the Project File. If the users or roles in your development database need to be in the destination database, contact your hosting provider for assistance.

SQL Server Timeout Error When Running Custom Scripts During Deployment

Scenario

You have specified custom SQL scripts to run during deployment, and when Web Deploy runs them, they time out.

Possible Cause and Solution

Running multiple scripts that have different transaction modes can cause time-out errors. By default, automatically generated scripts run in a transaction, but custom scripts do not. If you select the **Pull data and/or schema from an existing database** option on the **Package/Publish SQL** tab, and if you add a custom SQL script, you must change transaction settings on some scripts so that all scripts use the same transaction settings. For more information, see How to: Deploy a Database With a Web Application Project.

If you have configured transaction settings so that all are the same but still get this error, a possible workaround is to run the scripts separately. In the **Database Scripts** grid in the **Package/Publish** SQL tab, clear the **Include** check box for the script that causes the timeout error, then publish the project. Then go back into the **Database Scripts** grid, select that script's **Include** check box, and clear the **Include** check boxes for the other scripts. Then publish the project again. This time when you publish, only the selected custom script runs.

Stream Data of Site Manifest Is Not Yet Available

Scenario

When you are installing a package using the *deploy.cmd* file with the `t` (test) option, you see the following error message:

```
Error: The stream data of
'sitemanifest/dbFullSql[@path='C:\TEMP\AdventureWorksGrant.sql']/sqlScript' is not
yet available.
```

Possible Cause and Solution

The error message means that the command cannot produce a test report. However, the command might run if you use the `y` (actual installation) option. The message indicates only that there is a problem with running the command in test mode.

This Application Requires ManagedRuntimeVersion v4.0

Scenario

When you attempt to deploy, you see the following error message:

```
The application pool that you are trying to use has the 'managedRuntimeVersion'
property set to 'v2.0'. This application requires 'v4.0'.
```

Possible Cause and Solution

ASP.NET 4 is not installed in IIS. If the server you are deploying to is your development computer and has Visual Studio 2010 installed on it, ASP.NET 4 is installed on the computer but might not be installed in IIS. On the server that you are deploying to, open an elevated command prompt and install ASP.NET 4 in IIS by running the following commands:

```
cd %windir%\Microsoft.NET\Framework\v4.0.30319
aspnet_regiis.exe –i
```

Unable to cast Microsoft.Web.Deployment.DeploymentProviderOptions

Scenario

When you are deploying a package, you see the following error message:

```
Unable to cast object of type 'Microsoft.Web.Deployment.DeploymentProviderOptions' to
'Microsoft.Web.Deployment.DeploymentProviderOptions'.
```

Possible Cause and Solution

You are trying to deploy from IIS Manager using the Web Deploy 1.1 UI to a server that has Web Deploy 2.0 installed. If you are using the IIS Remote Administration Tool to deploy by importing a package, check the **New Features Available** dialog box when you establish the connection. (This dialog box might only be shown once when the connection is first established. To clear the connection and start over, close IIS Manager and start it up again by entering `inetmgr /reset` at the command prompt.) If one of the features listed is **Web Deploy UI**, and it has a version number lower than 8, the server you are deploying to might have both 1.1 and 2.0 versions of Web Deploy installed. To deploy from a client that has 2.0 installed, the server must have only Web Deploy 2.0 installed. You will have to contact your hosting provider to resolve this problem.

Unable to load the native components of SQL Server Compact

Scenario

When you run the deployed site, you see the following error message:

```
Unable to load the native components of SQL Server Compact corresponding to the
ADO.NET provider of version 8482. Install the correct version of SQL Server Compact.
Refer to KB article 974247 for more details.
```

Possible Cause and Solution

The deployed site does not have *amd64* and *x86* subfolders with the native assemblies in them under the application's *bin* folder. On a computer that has SQL Server Compact installed, the native assemblies are located in *C:\Program Files\Microsoft SQL Server Compact Edition\v4.0\Private*. The best way to get the correct files into the correct folders in a Visual Studio project is to install the NuGet SqlServerCompact package. Package installation adds a post-build script to copy the native assemblies into *amd64* and *x86*. In order for these to be deployed, however, you have to manually include them in the project. For more information, see the Deploying SQL Server Compact tutorial.

"Path is not valid" error after deploying an Entity Framework Code First application

Scenario

You deploy an application that uses Entity Framework Code First Migrations and a DBMS such as SQL Server Compact which stores its database in a file in the App_Data folder. You have Code First Migrations configured to create the database after your first deployment. When you run the application you get an error message like the following example:

```
The path is not valid. Check the directory for the database. [Path =
c:\inetpub\wwwroot\App_Data\DatabaseName.sdf ]
```

Possible Cause and Solution

Code First is attempting to create the database but the App_Data folder does not exist. Either you didn't have any files in the *App_Data* folder when you deployed, or you selected **Exclude App_Data** on the **Package/Publish Web** tab of the **Project Properties** window. The deployment process won't create a folder on the server if there are no files in the folder to be copied to the server. If you already had the database set up in the site, the deployment process will delete the files and the *App_Data* folder itself if you selected **Remove additional files at destination** in the publish profile. To solve the problem, put a placeholder file such as a .txt file in the *App_Data* folder, make sure you do not have **Exclude App_Data** selected, and redeploy.

"COM object that has been separated from its underlying RCW cannot be used."

Scenario

You have been successfully using one-click publish to deploy your application and then you start getting this error:

```
Web deployment task failed. (Could not complete the request to remote agent URL
'https://serverurl.com/msdeploy.axd?site=sitename'.)
Could not complete the request to remote agent URL
'https://url/msdeploy.axd?site=sitename'.
The request was aborted: The request was canceled.
COM object that has been separated from its underlying RCW cannot be used.
```

Possible Cause and Solution

Closing and restarting Visual Studio is usually all that is required to resolve this error.

Deployment Fails Because User Credentials Used for Publishing Don't Have setACL Authority

Scenario

Publishing fails with an error that indicates you don't have authority to set folder permissions (the user account you are using doesn't have setACL authority).

Possible Cause and Solution

By default, Visual Studio sets read permissions on the root folder of the site and write permissions on the App_Data folder. If you know that the default permissions on site folders are correct and do not need to be set, you disable this behavior by adding **<IncludeSetACLProviderOnDestination>False</IncludeSetACLProviderOnDestination>** to the publish profile file (to affect a single profile) or to the wpp.targets file (to affect all profiles). For information about how to edit these files, see How to: Edit Deployment Settings in Profile (.pubxml) Files.

Access Denied Errors when the Application Tries to Write to an Application Folder

Scenario

Your application errors when it tries to create or edit a file in one of the application folders, because it does not have write authority for that folder.

Possible Cause and Solution

By default, Visual Studio sets read permissions on the root folder of the site and write permissions on the App_Data folder. If your application needs write access to a sub-folder, you can set permissions for that folder as shown in the Setting Folder Permissions and Deploying to the Production Environment tutorials. If your application needs write access to the root folder of the site, you have to prevent it from setting read-only access on the root folder by adding **<IncludeSetACLProviderOnDestination>False</IncludeSetACLProviderOnDestination>** to the publish profile file (to affect a single profile) or to the wpp.targets file (to affect all profiles). For information about how to edit these files, see How to: Edit Deployment Settings in Profile (.pubxml) Files.

Configuration Error - targetFramework attribute references a version that is later than the installed version of the .NET Framework

Scenario

You successfully published a web project that targets ASP.NET 4.5, but when you run the application (with the `customErrors` mode set to "off" in the Web.config file) you get the following error:

```
The 'targetFramework' attribute in the <compilation> element of the Web.config file
is used only to target version 4.0 and later of the .NET Framework (for example,
'<compilation targetFramework="4.0">'). The 'targetFramework' attribute currently
```

```
references a version that is later than the installed version of the .NET Framework.
Specify a valid target version of the .NET Framework, or install the required version
of the .NET Framework.
```

The Source Error box of the error page highlights the following line from Web.config as the cause of the error:

```
<compilation targetFramework="4.5" />
```

Possible Cause and Solution

The server does not support ASP.NET 4.5. Contact the hosting provider to determine when and if support for ASP.NET 4.5 can be added. If upgrading the server is not an option, you have to deploy a web project that targets ASP.NET 4 or earlier instead.

If you deploy an ASP.NET 4 or earlier web project to the same destination, select the **Remove additional files at destination** check box on the **Settings** tab of the **Publish Web** wizard. If you don't select **Remove additional files at destination**, you will continue to get the Configuration Error page.

The project **Properties** windows includes a Target framework drop-down list, but you can't resolve this problem by just changing that from **.NET Framework 4.5** to **.NET Framework 4**. If you change the target framework to an earlier framework version, the project will still have references to the later framework version's assemblies and will not run. You have to manually change those references or create a new project that targets .NET Framework 4 or earlier. For more information, see .NET Framework Targeting for Web Sites.