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

Tom Dykstra

Step-by-Step

Microsoft

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

Tom Dykstra

Summary: This series of tutorials shows you how to make an ASP.NET web application available over the internet by deploying it to a third-party hosting provider. The deployment method used is Visual Studio one-click publish. The tutorials illustrate deployment first to IIS on the development computer for testing. They then show you how to deploy to the hosting provider as the production environment.

Category: Step-By-Step
Applies to: ASP.NET 4.0 and earlier, Visual Studio 2010
Source: ASP.NET site (link to source content)
E-book publication date: May 2012
162 pages

Deployment to a Hosting Provider

Contents

Deployment to a Hosting Provider: Introduction - 1 of 12

Overview

This series of tutorials shows you how to make an ASP.NET web application available over the internet by deploying it to a third-party hosting provider. The deployment method used is Visual Studio one-click publish. The tutorials illustrate deployment first to IIS on the development computer for testing. They then show you how to deploy to the hosting provider as the production environment.

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 to the production environment 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. Therefore, we've included many of these additional techniques in the tutorials, with the hope that the tutorials don't leave out information that prevents 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. (This might require you to adjust the procedures in later tutorials, of course.)

Intended Audience

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

- A continuous integration process (automated builds and deployment) is not used.
- The production environment is a third-party shared 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. A different series of tutorials for the enterprise scenario is under development, and a link to it will be provided here when it is available. In the meantime, for more information about deployment for the enterprise scenario, see ASP.NET Deployment Content Map.

Organizations of all sizes can also deploy to Windows Azure. For Windows Azure guidance, see Cloud Development in the MSDN Library web site.

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.

Web Application Projects versus 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.

ASP.NET Web Forms versus ASP.NET MVC

Contoso University is an ASP.NET Web Forms project, but everything you learn in these tutorials is applicable to ASP.NET MVC as well, because 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 do some extra work to deploy MVC assemblies in the application's *bin* folder.

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 sometimes do not give very good clues to the source of the problem or might not explain how to fix it. 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 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 either of these links, the following item will be installed automatically.)
- Microsoft Visual Studio 2010 SP1 Tools for SQL Server Compact 4.0

Some other software is required in order to complete the application, 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.

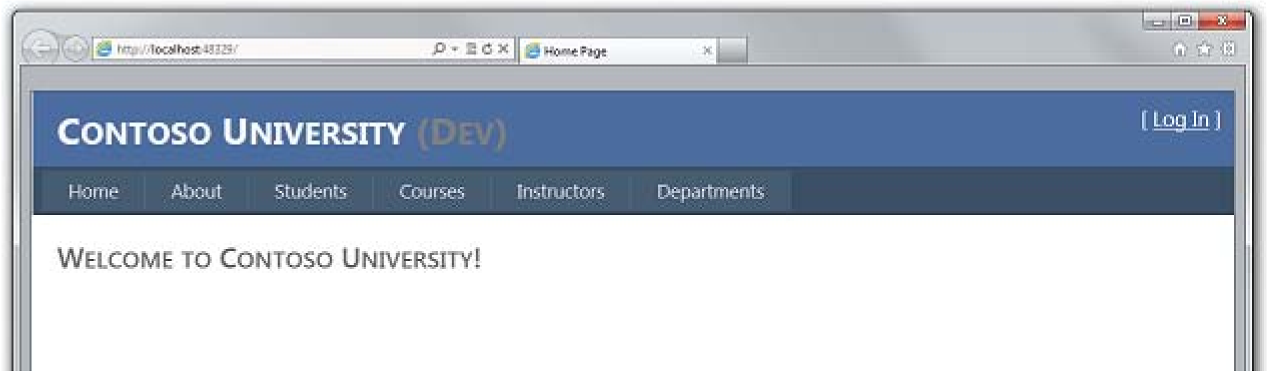
When you have the prerequisites installed, download the Contoso University web application. The *.zip* file contains multiple versions of the project. To work through the steps of the tutorial, start with ContosoUniversity-Begin. To see what the project looks like at the end of the tutorials, open ContosoUniversity-End. 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 2010\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.

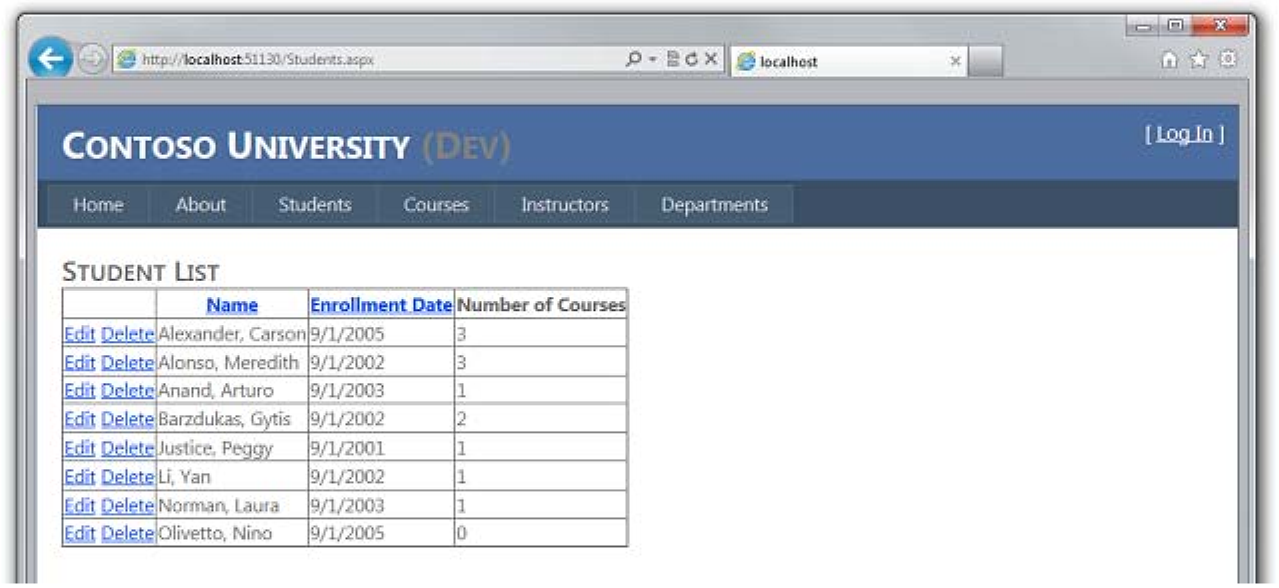


The website pages are accessible from the menu bar and let you perform the following functions:

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

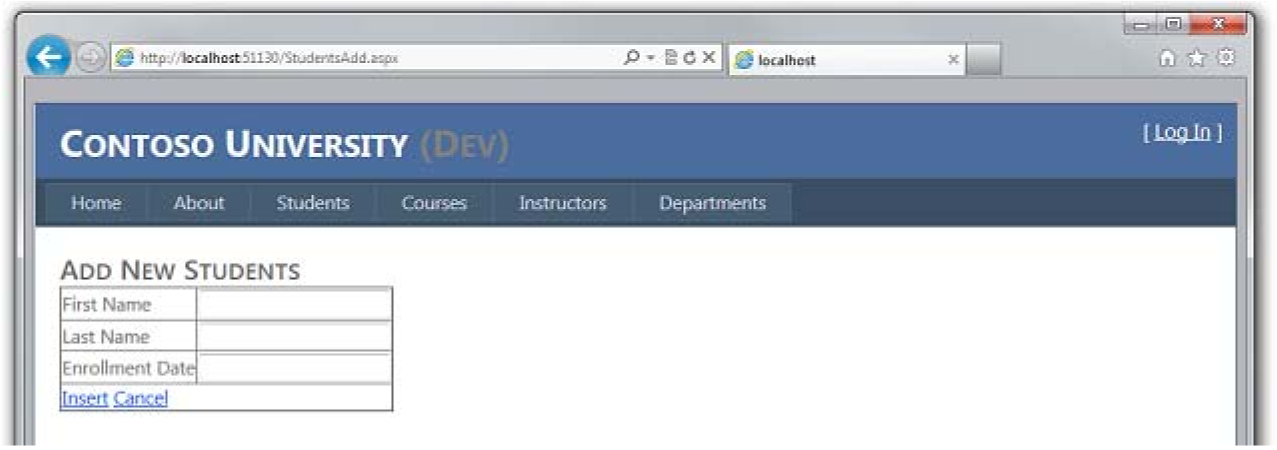
- Display and edit departments.

Below are screen shots of a few representative pages.



STUDENT LIST

	Name	Enrollment Date	Number of Courses
Edit Delete	Alexander, Carson	9/1/2005	3
Edit Delete	Alonso, Meredith	9/1/2002	3
Edit Delete	Anand, Arturo	9/1/2003	1
Edit Delete	Barzdukas, Gytis	9/1/2002	2
Edit Delete	Justice, Peggy	9/1/2001	1
Edit Delete	Li, Yan	9/1/2002	1
Edit Delete	Norman, Laura	9/1/2003	1
Edit Delete	Olivetto, Nino	9/1/2005	0



ADD NEW STUDENTS

First Name	
Last Name	
Enrollment Date	
Insert Cancel	

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 will be 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.
- 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 need 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 4.1 (Code First) to access data in the application database. The assembly that contains Entity Framework 4.1 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 utility writes error information in XML files to a file folder. You need to make sure that the account that ASP.NET runs under in the deployed site has write permission to this folder, and you need 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 in the *Web.config* file that must be changed 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'll prepare for deployment by setting up some of these things to be handled automatically. Others you'll take care of manually.

Deploying SQL Server Compact Databases - 2 of 12

Overview

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 4.1 (Code First).

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 need 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. Another advantage is that it's relatively simple to back up and restore your production data because the data is all in an *.sdf* file in the *App_Data* folder of the deployed site. Backing up and restoring is a simple matter of copying files.

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 2005 and SQL Server 2008 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 use to manipulate schemas and data in SQL Server Express and full SQL Server databases are not available for SQL Compact. For example, you cannot use SQL Server Management Studio or Visual Studio database projects with SQL Server Compact databases. When you use SQL Server Compact, Entity Framework Code First automatically creates your database based on your data model while you are developing an application. But after you deploy, keeping the development and production databases in sync can be more difficult than it would be if you could use a tool such as SQL Server Management Studio. More information about deploying database changes to SQL Server Compact databases is provided in the Deploying a Database Change tutorial.

SQL Server Compact is a good choice for databases that are easy to manage using simple database tools like Server Explorer in Visual Studio. (As you'll do in these tutorials.) But if you think your database might change repeatedly after its initial deployment, the cost savings of using SQL Server Compact might not offset the additional time you would have to invest in database maintenance.

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

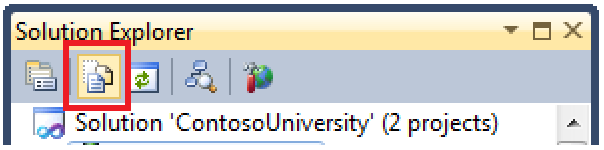
- SqlServerCompact
- System.Web.Providers (ASP.NET universal providers)
- EntityFramework
- EntityFramework.SqlServerCompact

The links point to the current version of the packages, which might not be the same version used in the starter project that you downloaded.

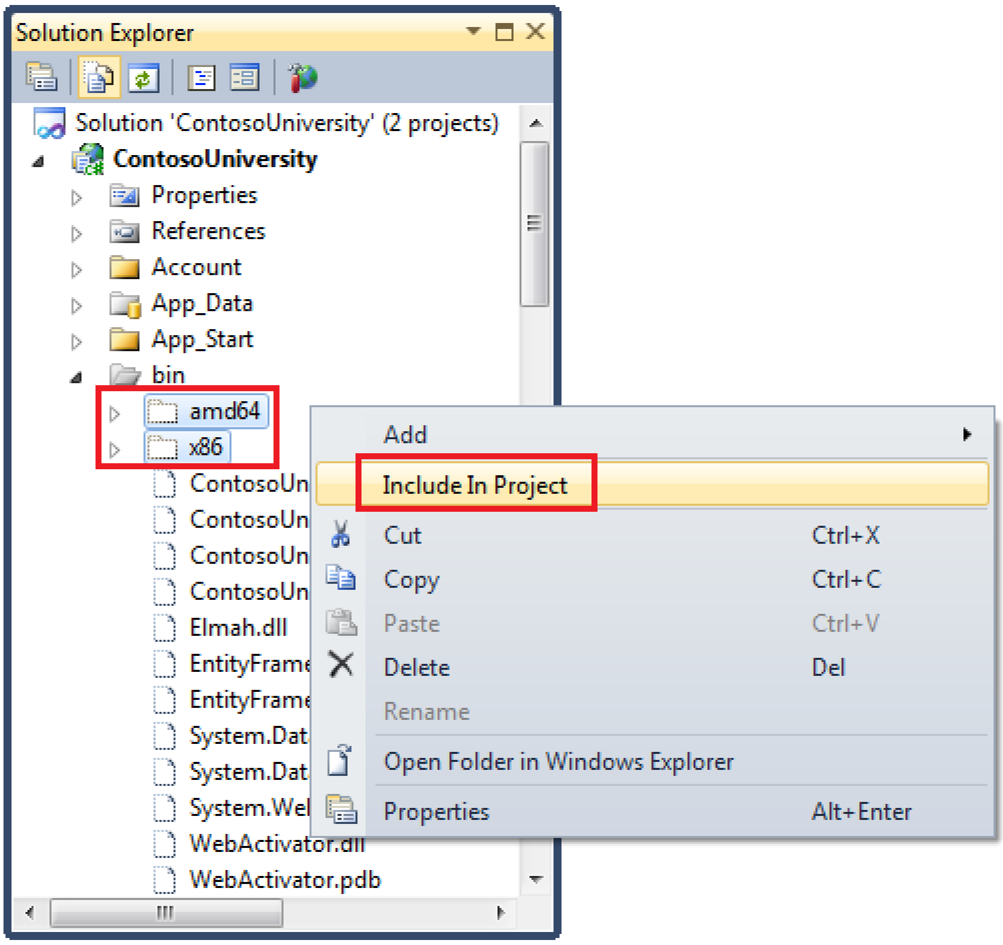
NuGet package installation generally takes care of everything you need to deploy this software with the application. In some cases, this involves tasks such as changing the Web.config file and adding PowerShell scripts that run whenever you build the solution. **If you want to add support for any of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

There is one exception where you have to do something 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 it 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 is true of 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 explained in the Configuring Project Properties tutorial. It is not generally recommended because it can result in the deployment of many more files to the production environment than are needed there.)

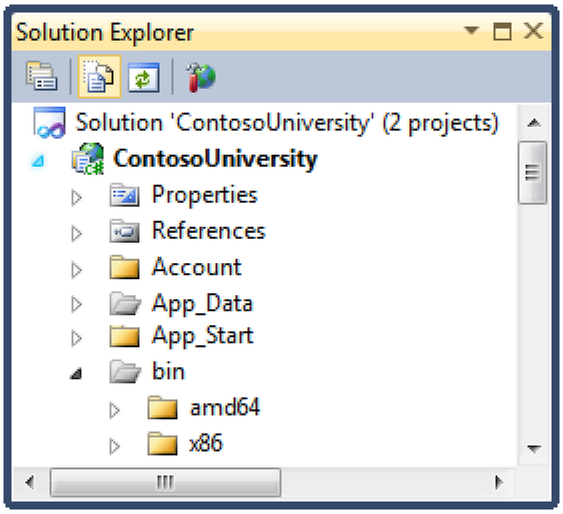
Build the project, and then click **Show all Files**in **Solution Explorer**if you have not already done so. You might also have to click **Refresh**.



Expand the **bin** folder to see the **amd64** and **x86** folders, 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.

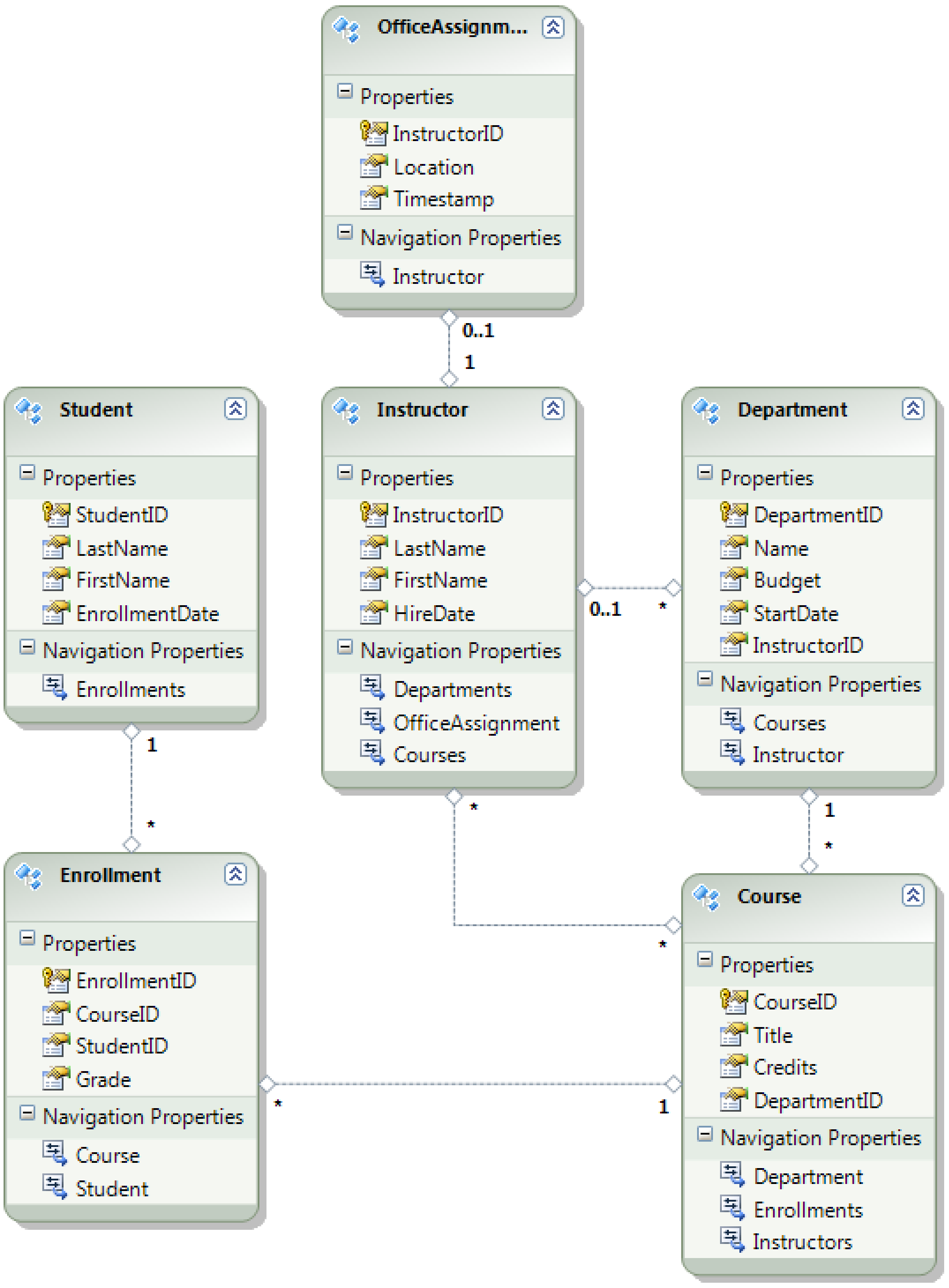


Creating an Application Database for Deployment

When you deploy an application database, typically you cannot 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.

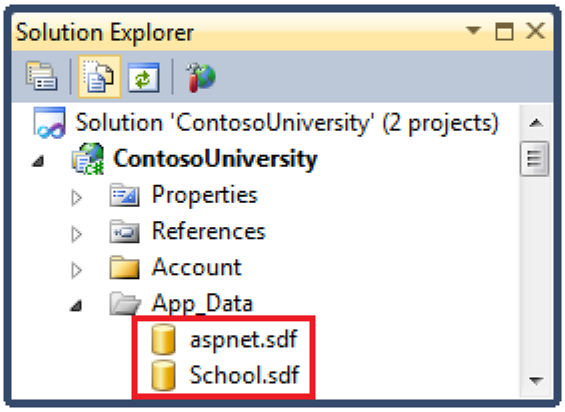
To simulate the common scenario of deploying a database that is not identical to the one you use in development, you'll create a version of the database that has some tables that contain data and some that are empty.

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 already contain data that has to be preloaded when the application goes live. The following steps show you one way to remove test data using the SQL Server tools built into Visual Studio. (You do not need to save your test data because it is generated automatically by an Entity Framework Code First initializer class.)

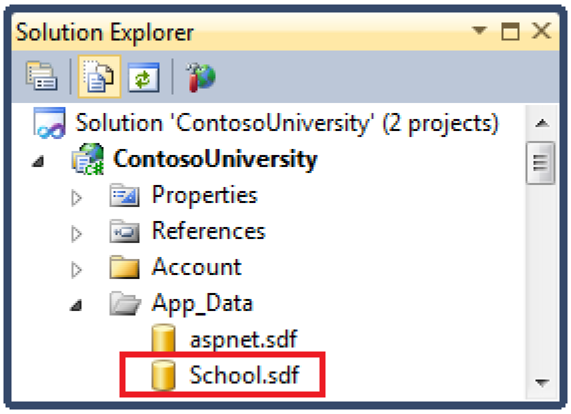
SQL Server Compact databases are contained in *.sdf* files in the *App_Data* folder. In **Solution Explorer**, expand *App_Data* to see the two SQL Server Compact databases, which are represented by *.sdf* files.



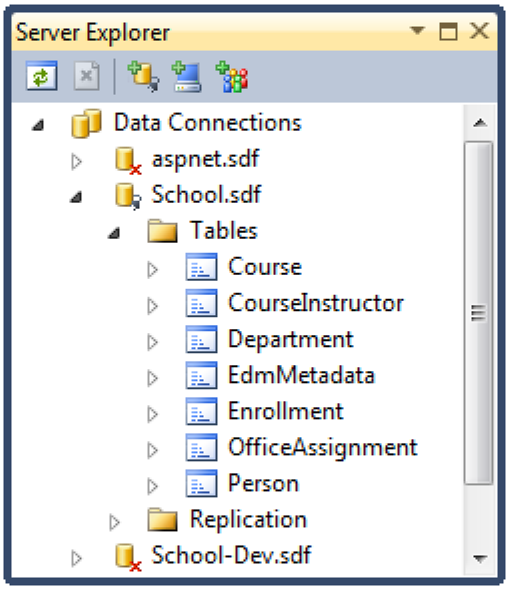
These are your development databases. When you're done with this tutorial, you will have four databases, a development version of each and a production version of each. To make clear what each one is, you'll give them names that make clear whether they contain development or production data: aspnet-Dev.sdf and School-Dev.sdf for development, and aspnet-Prod.sdf and School-Prod.sdf for production.

You'll start by modifying the *School.sdf* database file so that it has only the data that you want to deploy to production. (You'll rename it to *School-Prod.sdf* later in the tutorial.)

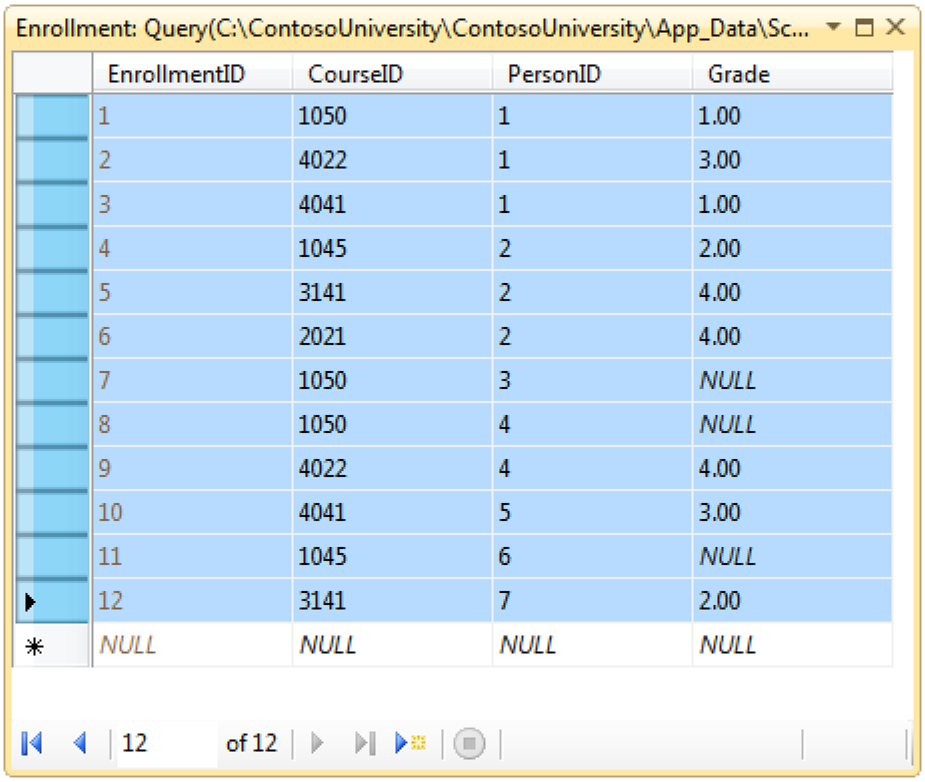
In **Solution Explorer**, double-click **School.sdf** to open **Server Explorer.**



In **Server Explorer**, expand **School.sdf** and then expand **Tables**.

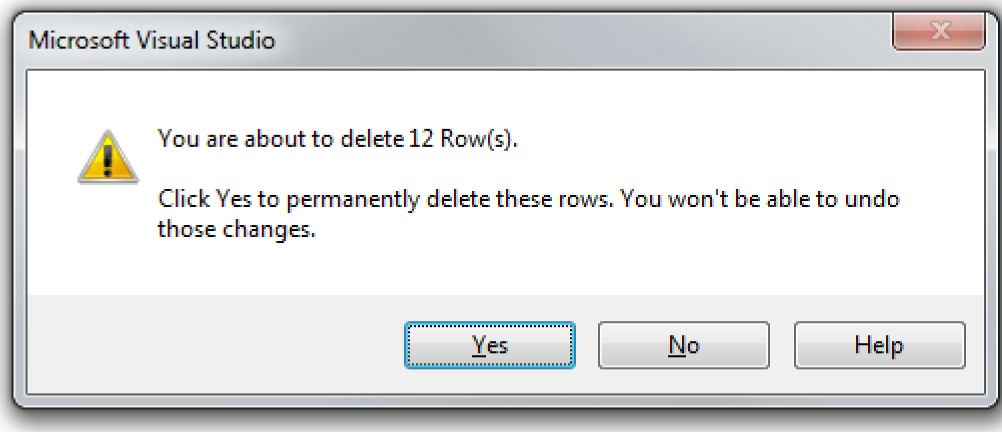


Right-click **Enrollment** and choose **Show Table Data**. Select all rows and press **Delete**.


Enrollment: Query(C:\ContosoUniversity\ContosoUniversity\App_Data\Sc...
12
of 12

EnrollmentID	CourseID	PersonID	Grade
1	1050	1	1.00
2	4022	1	3.00
3	4041	1	1.00
4	1045	2	2.00
5	3141	2	4.00
6	2021	2	4.00
7	1050	3	NULL
8	1050	4	NULL
9	4022	4	4.00
10	4041	5	3.00
11	1045	6	NULL
12	3141	7	2.00
NULL	NULL	NULL	NULL

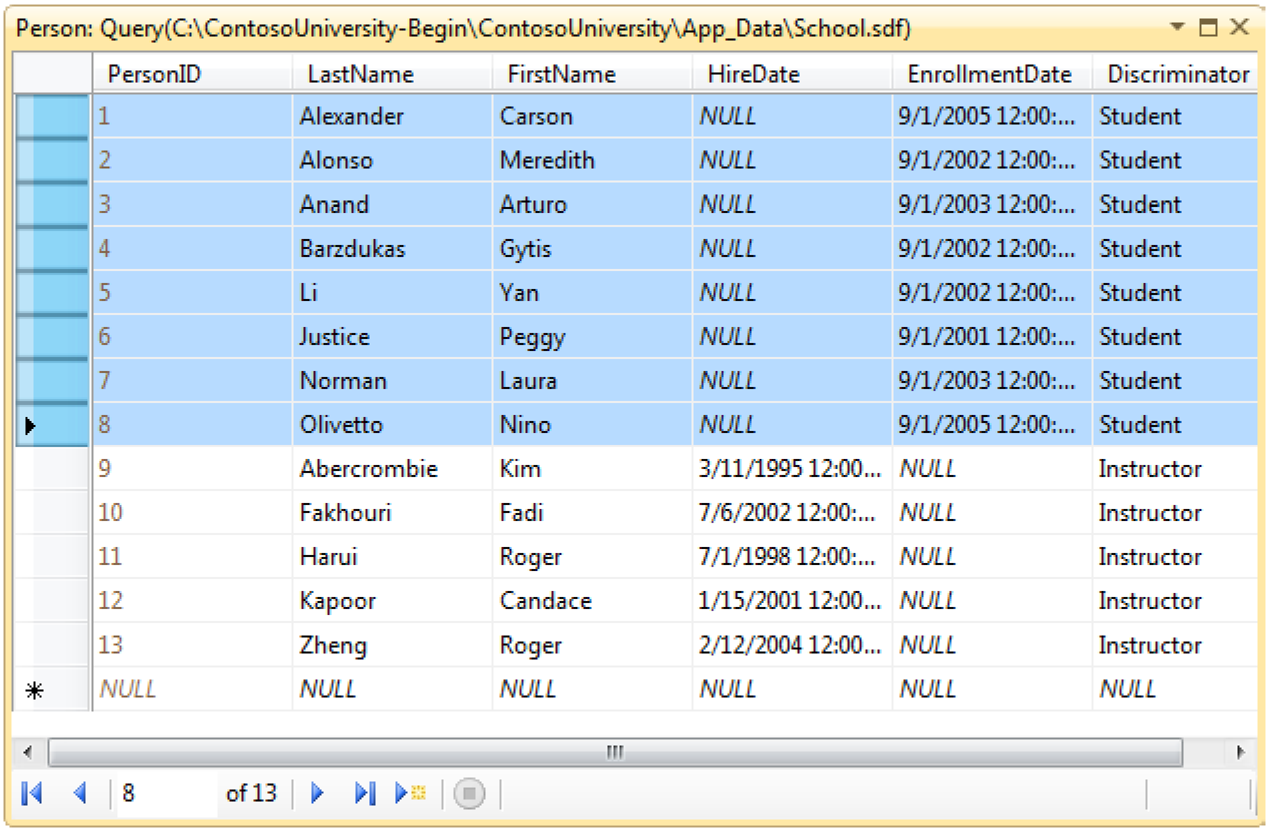
When you are prompted to confirm deletion, click **Yes**.


Microsoft Visual Studio

You are about to delete 12 Row(s).

Click Yes to permanently delete these rows. You won't be able to undo those changes.

Follow the same procedure to delete all student rows in the `Person` table (all rows that have a non-null value in the `EnrollmentDate` column and "Student" in the `Discriminator` column).


Person: Query(C:\ContosoUniversity-Begin\ContosoUniversity\App_Data\School.sdf)

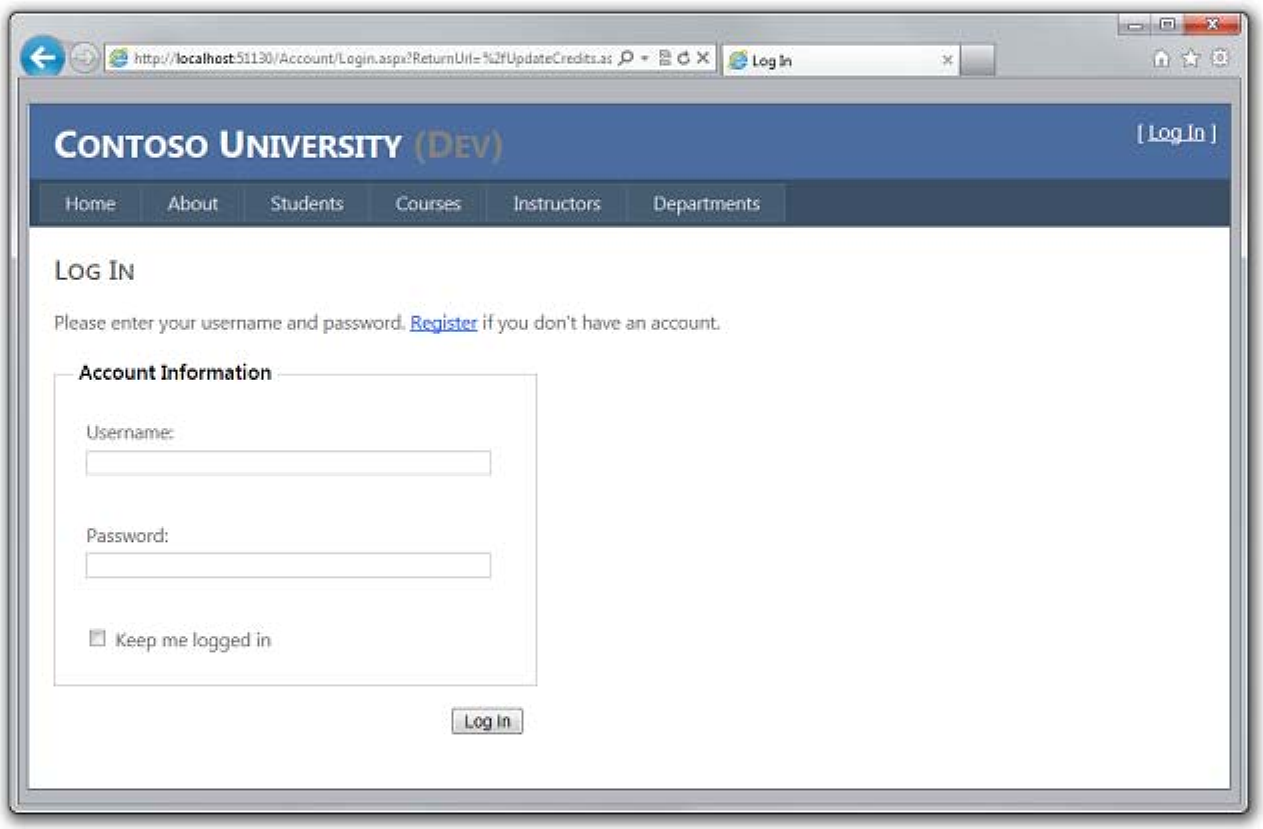
PersonID	LastName	FirstName	HireDate	EnrollmentDate	Discriminator
1	Alexander	Carson	NULL	9/1/2005 12:00:...	Student
2	Alonso	Meredith	NULL	9/1/2002 12:00:...	Student
3	Anand	Arturo	NULL	9/1/2003 12:00:...	Student
4	Barzdukas	Gytis	NULL	9/1/2002 12:00:...	Student
5	Li	Yan	NULL	9/1/2002 12:00:...	Student
6	Justice	Peggy	NULL	9/1/2001 12:00:...	Student
7	Norman	Laura	NULL	9/1/2003 12:00:...	Student
8	Olivetto	Nino	NULL	9/1/2005 12:00:...	Student
9	Abercrombie	Kim	3/11/1995 12:00...	NULL	Instructor
10	Fakhouri	Fadi	7/6/2002 12:00:...	NULL	Instructor
11	Harui	Roger	7/1/1998 12:00:...	NULL	Instructor
12	Kapoor	Candace	1/15/2001 12:00...	NULL	Instructor
13	Zheng	Roger	2/12/2004 12:00...	NULL	Instructor
NULL	NULL	NULL	NULL	NULL	NULL

In **Server Explorer**, right-click *School.sdf* and select **Close Connection**.

The *School.sdf* database file is now ready to be deployed.

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 menu under **Courses**. The application displays the **Log In** page, because only administrators are authorized to use the **Update Credits** page.



LOG IN

Please enter your username and password. Register if you don't have an account.

Account Information

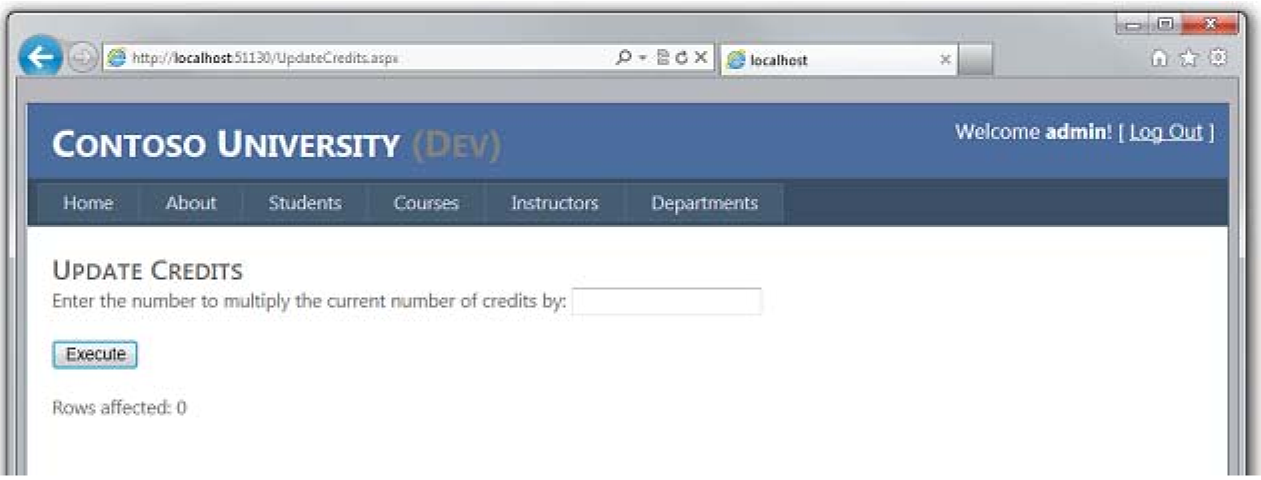
Username:

Password:

Keep me logged in

Log In

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.



UPDATE CREDITS

Enter the number to multiply the current number of credits by:

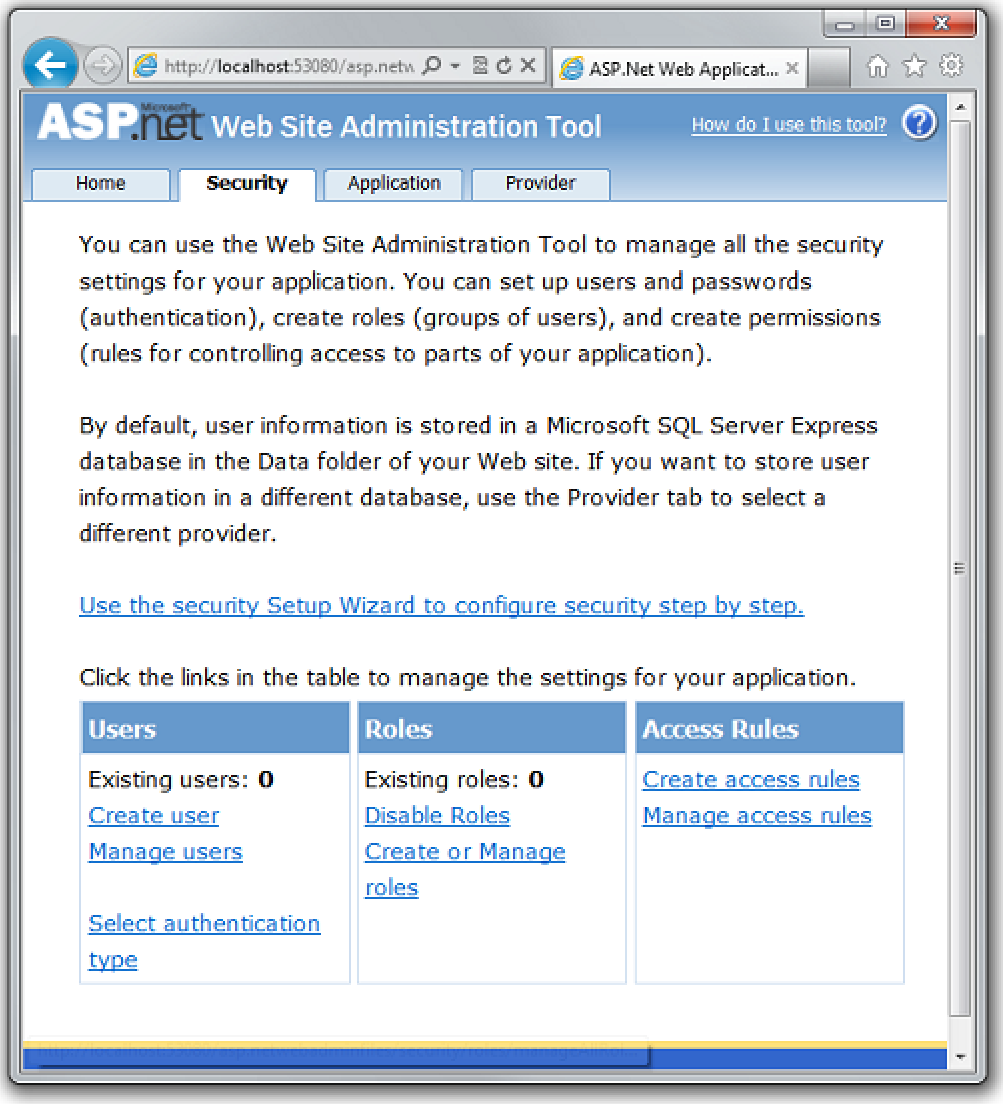
Execute

Rows affected: 0

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 will deploy with 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. In this case there is no automatic initializer that seeds the database with test accounts (as there is for the School database), and you want to keep the test data available so that you can restore it and continue to develop the web site after you deploy it; therefore you need to make a copy of the test database.

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.)

From the **Project** menu, select **ASP.NET Configuration** to run the **Web Site Administration Tool** (WAT), and then select the **Security** tab.



You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).

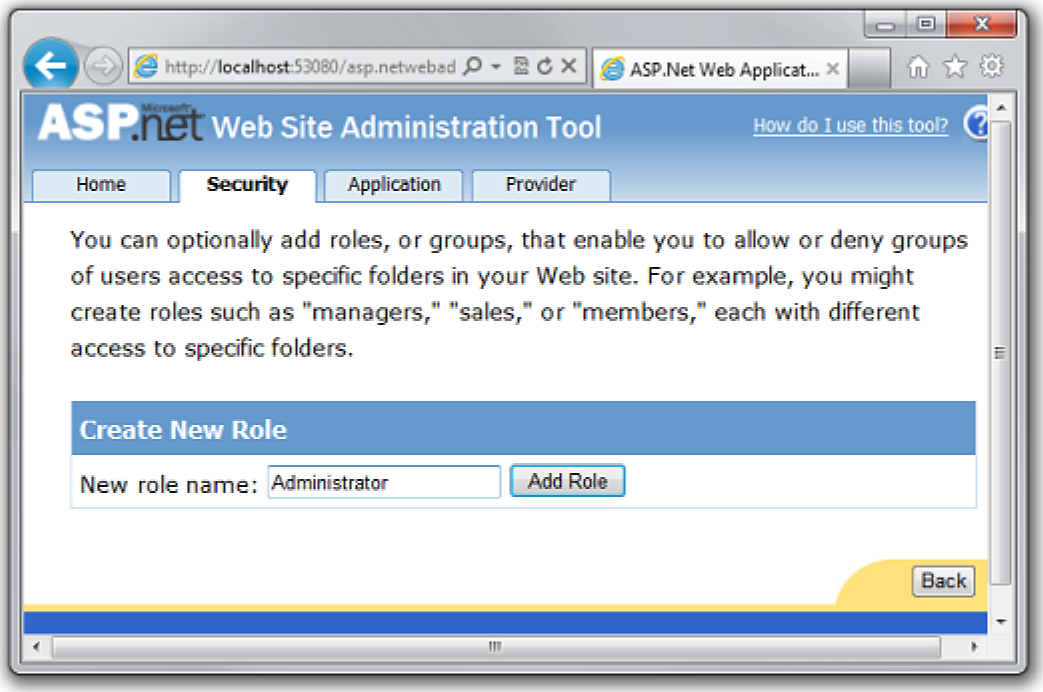
By default, user information is stored in a Microsoft SQL Server Express database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.

Use the security Setup Wizard to configure security step by step.

Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
Existing users: **0** Create user Manage users Select authentication type	Existing roles: **0** Disable Roles Create or Manage roles	Create access rules Manage access rules

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

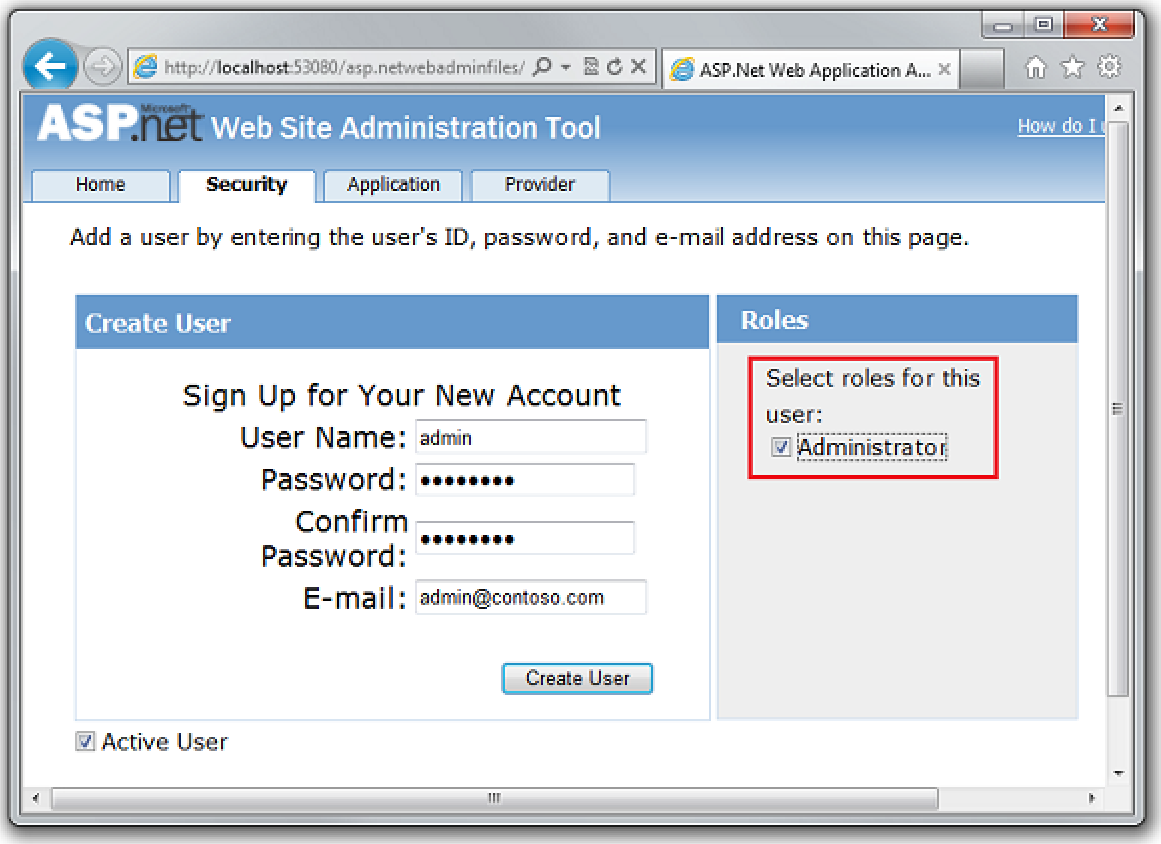


You can optionally add roles, or groups, that enable you to allow or deny groups of users access to specific folders in your Web site. For example, you might create roles such as "managers," "sales," or "members," each with different access to specific folders.

Create New Role

New role name: Administrator

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 to include the user in the Administrator role. The password used in this tutorial is "Pas$w0rd", and you can enter any email address.


http://localhost:53080/asp.netwebadminfiles/
ASP.Net Web Application A...
ASP.net Web Site Administration Tool
How do I
Home
Security
Application
Provider

Add a user by entering the user's ID, password, and e-mail address on this page.

Create User

Sign Up for Your New Account

User Name: admin

Password: ••••••••

Confirm Password: ••••••••

E-mail: admin@contoso.com

Create User

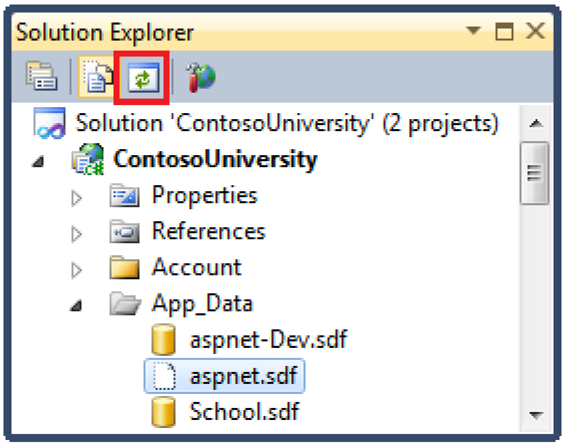
☑ Active User

Roles

Select roles for this user:

☑ Administrator

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

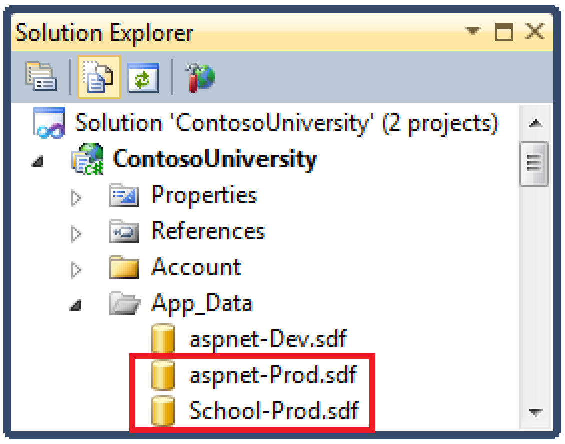


Right-click **aspnet.sdf** and select **Include in Project**.

Renaming the Production Databases

It is not necessary to rename the production databases, but doing so will help make their purpose clear:

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



The two databases are now ready to be deployed.

Setting Connection Strings so that Only Development Databases are Used in Development

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 need 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-dev.sdf file, but that's OK because Code First creates your database for you when it finds that you don't have one -- all you have to do is specify what you want to name it.)

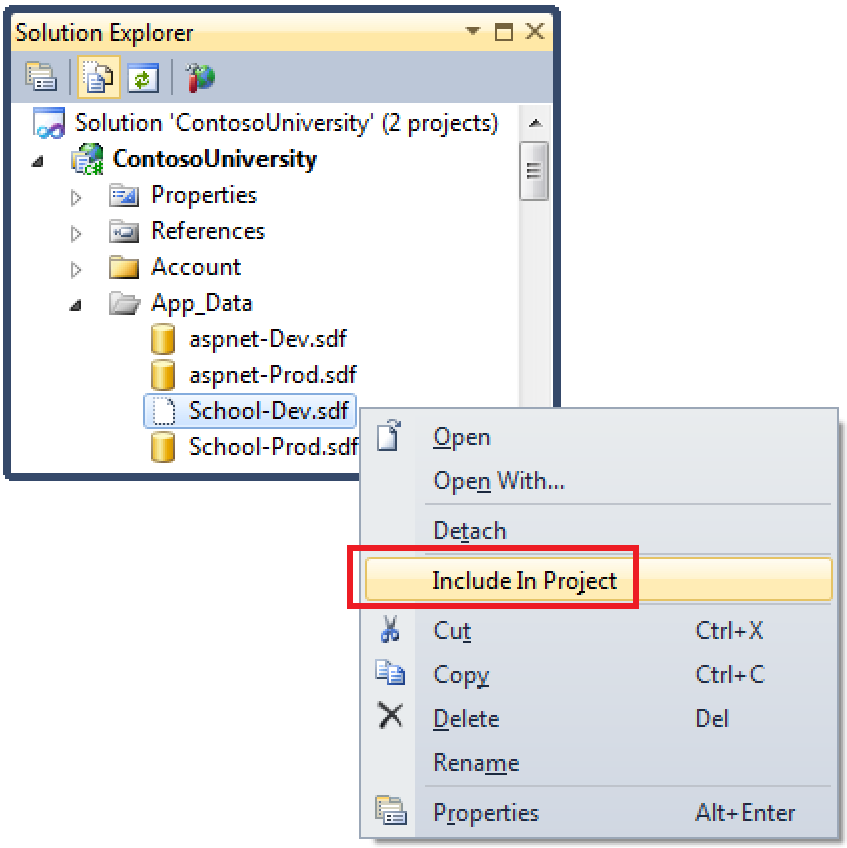
Open the application Web.config file, and locate the connection strings:

```
<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>
```

Run the application and select a page that accesses the database, such as the Students page. This causes Code First to create a new School-Dev.sdf database. When you click Refresh in Solution Explorer, you'll see the database; right-click it and select Include in project.



The SQL Server Compact database engine and production versions of both databases are now ready to be deployed. In the following tutorial you will set up automatic *Web.config* file transformations for settings that need to be different in the development, test, and production environments. (Among the settings that need to be changed are the connection strings: you want the application to use the production version of the database in the test and production environments.)

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 will need to learn how to analyze a NuGet package to determine what it does when it is installed (*Web.config* transformations, PowerShell scripts, etc.); for that, see especially Creating and Publishing a Package and Configuration File and Source Code Transformations.

Web.Config File Transformations - 3 of 12

Overview

Most applications have settings in the *Web.config* file that need to be different when the application is deployed. Making these changes manually every time you deploy would be tedious and error prone. This tutorial shows you how to avoid those problems by automating the process of changing the *Web.config* file.

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 changes that need to be applied to the *Web.config* file when it is deployed. You can specify different changes for different build configurations. The default build configurations are Debug and Release, and you can create custom build configurations.

Web Deploy parameters can be used to specify many different kinds of settings that need to be configured during deployment, including those 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 needs to be done to the *Web.config* file, so you do not need to use Web Deploy parameters.

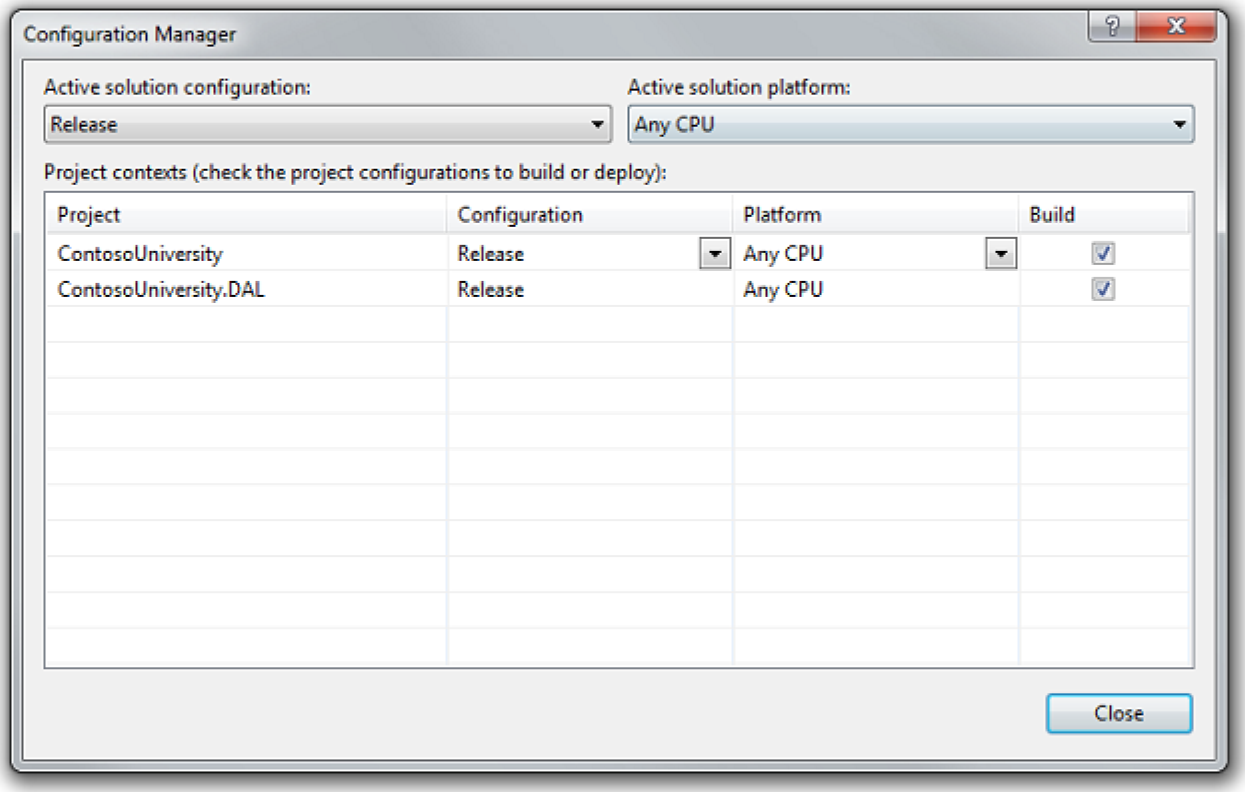
Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Creating a New Build Configuration

You will have two deployment destinations: a test environment and the production environment. When you deploy to the test environment you typically want to deploy a Release build, not a Debug build, but some of the changes to the *Web.config* file need to be different for the test environment than they are for production. Since *Web.config* file transformations are specified by build configuration, you need to create a new build configuration that you can use for the test environment.

You can use the default Release build configuration for transformations intended to be deployed to production. You can use the default Debug build configuration for transformations intended to be deployed to the test environment when you want to be able to debug in that environment. And you'll create a Test build configuration that you can use to specify transformations intended for when you want to deploy a Release build (with debugging disabled) to the test environment.

From the Visual Studio **Build** menu, select **Configuration Manager** to display the **Configuration Manager** dialog box.


Configuration Manager

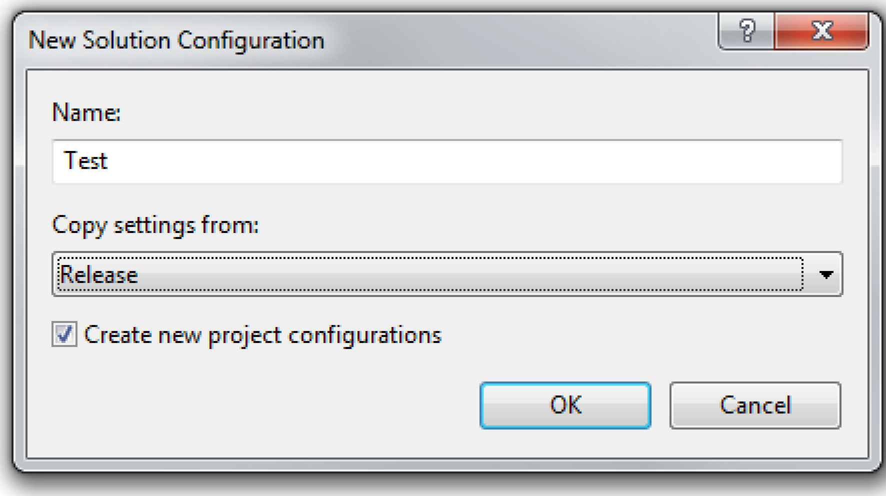
Active solution configuration: Release

Active solution platform: Any CPU

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build
ContosoUniversity	Release	Any CPU	✓
ContosoUniversity.DAL	Release	Any CPU	✓

In the **Active solution configuration** box, select **New**. When the **New Solution Configuration** dialog box appears, enter "Test" as the name of the new build configuration, and copy settings from **Release**. Leave **Create new project configurations** selected, and click **OK**.


New Solution Configuration

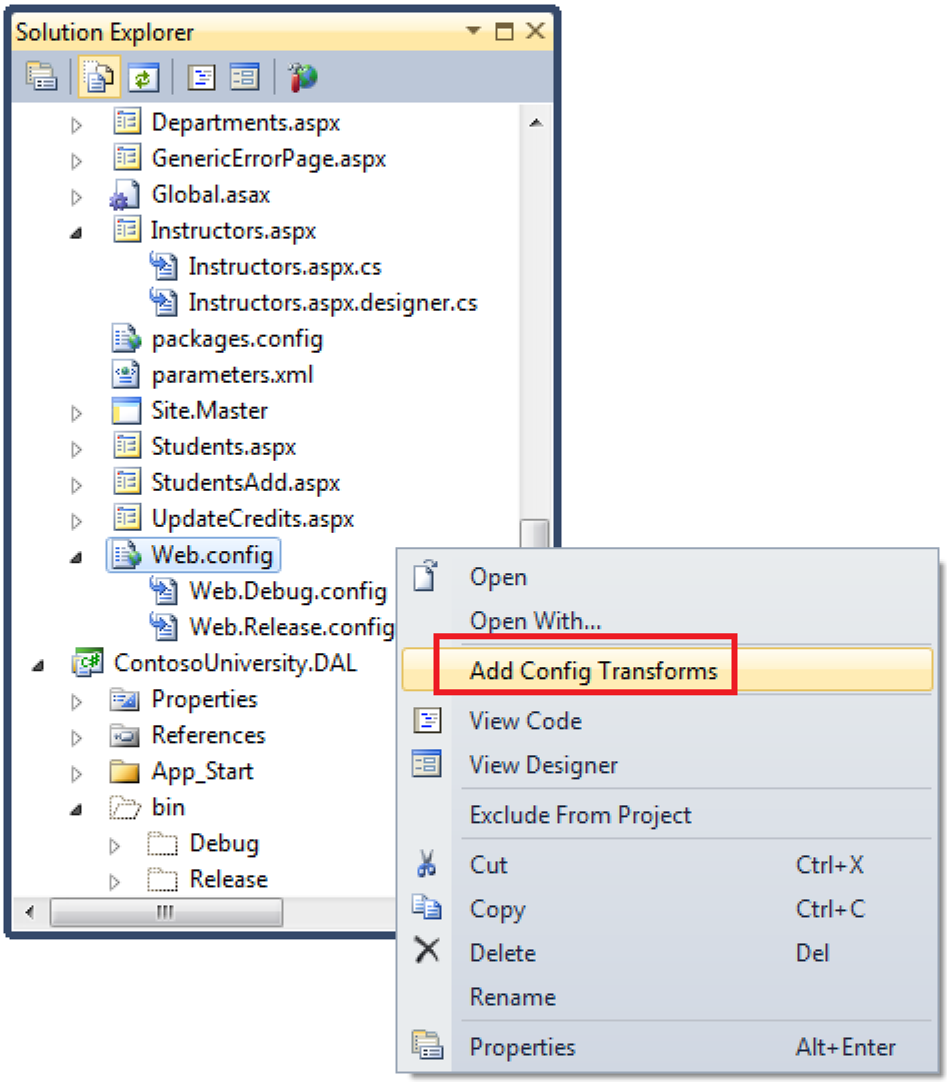
Name: Test

Copy settings from: Release

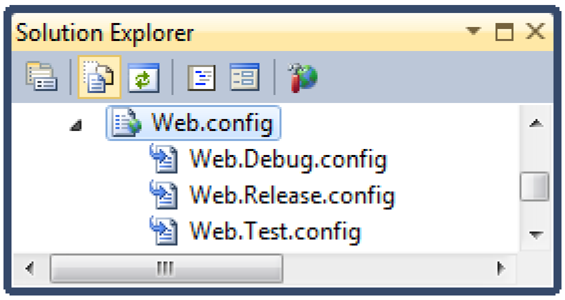
✓ Create new project configurations

Close the **Configuration Manager** dialog box.

You'll also need a *Web.config* transform file for the Test build configuration. In **Solution Explorer**, expand *Web.config* to see the *Web.Debug.config* and *Web.Release.config* files that are created by default. Right-click *Web.config* and select **Add Config Transforms**.



The *Web.Test.config* file is added.



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

Preventing Entity Framework Code First from Dropping the Production Database

On your development computer, Entity Framework Code First is typically configured to automatically drop and re-create the database whenever you change the data model. This behavior is very convenient while you're developing the site and making frequent changes to the data model, but you don't want this to happen in your production site. The recommended way to control the Entity Framework's automatic database initialization function is by using an `appSettings` value in the *Web.config* file.

(Some early Code First tutorials advised you to set the initializer by putting code in the `Application_Start` handler in the *Global.asax* file. If you have an application that does that, remove that code before deploying. It is also possible to handle Code First database initialization in a way that would require no change at all in the project that you deploy; for example, you could configure a test project to do the database initialization.)

In the *Web.config* file is a `DatabaseInitializerForType` setting in the `appSettings` element:

```
<appSettings>
<add key="DatabaseInitializerForType ContosoUniversity.DAL.SchoolContext,
ContosoUniversity.DAL"
    value="ContosoUniversity.DAL.SchoolInitializer ContosoUniversity.DAL" />
<!-- Other settings -->
</appSettings>
```

You need to set up deployment so that it changes the `value` attribute in the deployed site to "Disabled", as shown below:

```
<appSettings>
<add key="DatabaseInitializerForType ContosoUniversity.DAL.SchoolContext,
ContosoUniversity.DAL"
    value="Disabled" />
<!-- Other settings -->
</appSettings>
```

Open *Web.Release.config* and add a new `appSettings` element immediately after the opening `configuration` tag, as shown here. (Make sure you add only the `appSettings` 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">
<appSettings>
<add key="DatabaseInitializerForType ContosoUniversity.DAL.SchoolContext,
ContosoUniversity.DAL"
            value="Disabled" xdt:Transform="SetAttributes"
xdt:Locator="Match(key)"/>
</appSettings>
```

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

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 will cause the `value` attribute of the `appSettings` element for the Entity Framework Code First initializer to be set to "Disabled" in the deployed *Web.config* file.

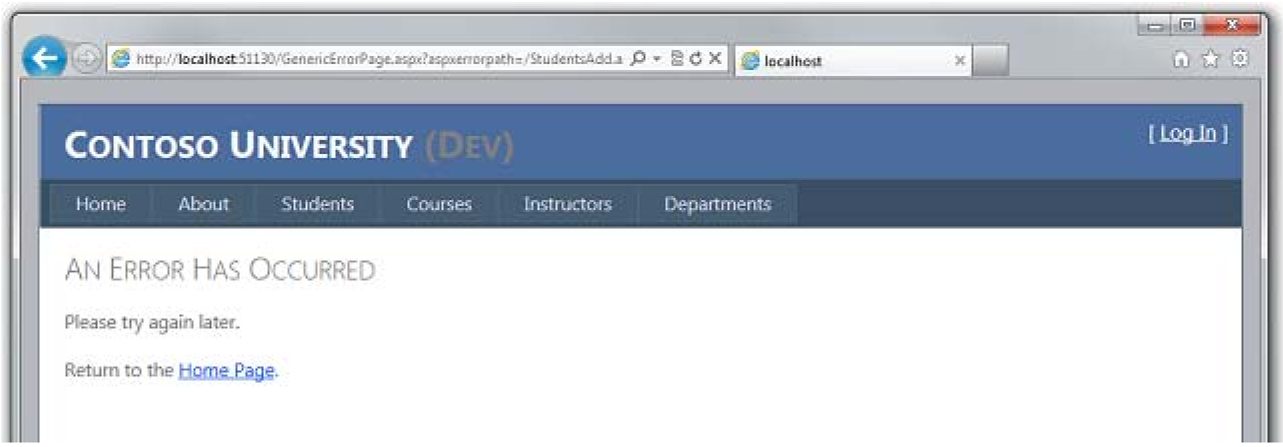
You want the same change to be made whenever you deploy to either test or production; automatic database re-creation should only happen in development. Therefore, add this same code to the *Web.Test.config* file also. (You don't need to update the *Web.Debug.config* file because you won't be creating any Debug builds for this tutorial.)

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 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.

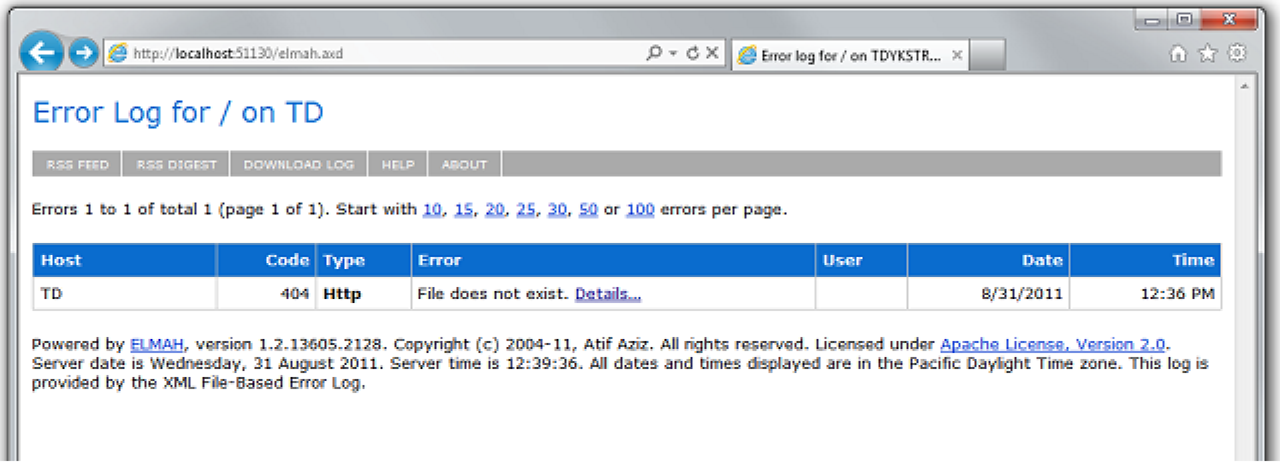

http://localhost:51130/GenericErrorPage.aspx?aspxerrorpath=/StudentsAdd.a

AN ERROR HAS OCCURRED

Please try again later.

Return to the Home 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:


http://localhost:51130/elmah.axd
Error log for / on TDYKSTR...

Error Log for / on TD

RSS FEED | RSS DIGEST | DOWNLOAD LOG | HELP | ABOUT

Errors 1 to 1 of total 1 (page 1 of 1). Start with 10, 15, 20, 25, 30, 50 or 100 errors per page.

Host	Code	Type	Error	User	Date	Time
TD	404	Http	File does not exist. Details...		8/31/2011	12:36 PM

Powered by ELMAH, version 1.2.13605.2128. Copyright (c) 2004-11, Atif Aziz. All rights reserved. Licensed under Apache License, Version 2.0. Server date is Wednesday, 31 August 2011. Server time is 12:39:36. All dates and times displayed are in the Pacific Daylight Time zone. This log is provided by the XML File-Based Error Log.

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 using a transform in the *Web.Release.config* file.

Open *Web.Release.config* and add the following code immediately following the closing tag of the `appSettings` element that you added earlier.

```
<location path="elmah.axd" xdt:Transform="Insert">
<system.web>
<authorization>
<allow roles="Administrator" />
<deny users="*" />
</authorization>
</system.web>
</location>
```

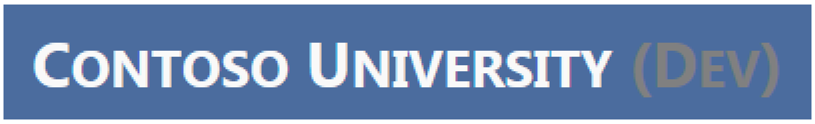
The `Transform` attribute value of "Insert" specifies that this `location` element will be added as a sibling to any existing `location` elements in the *Web.config* file. (There is already one `location` element that specifies authorization rules for the **Update Credits** page.) When you test the production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

You do not need to restrict error log access in the test environment, so you do not need to add this code to the *Web.Test.config* file.

Security Note You should never display error details to the public in a production application, or store that information in a public location. Attackers can sometimes 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 needs to 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>
<!-- Entity Framework initializer setting -->
<add key="Environment" value="Dev" />
</appSettings>
```

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

Open *Web.Release.config* and add an additional `add` element to the `appSettings` element, as shown below:

```
<appSettings>
<!-- Entity Framework initializer transform that you entered earlier -->
<add key="Environment" value="Prod" xdt:Transform="SetAttributes"
xdt:Locator="Match(key)"/>
</appSettings>
```

The `Transform` and `Locator` attributes are similar to the Entity Framework initializer transform that you created earlier.

When you are done, the `appSettings` section in *Web.Release.config* will look like this:

```
<appSettings>
<add key="DatabaseInitializerForType ContosoUniversity.DAL.SchoolContext,
ContosoUniversity.DAL"
        value="Disabled" xdt:Transform="SetAttributes"
xdt:Locator="Match(key)"/>
<add key="Environment" value="Prod" xdt:Transform="SetAttributes"
xdt:Locator="Match(key)"/>
</appSettings>
```

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 this:

```
<appSettings>
<add key="DatabaseInitializerForType ContosoUniversity.DAL.SchoolContext, ContosoUniversity.DAL"
        value="Disabled" xdt:Transform="SetAttributes"
xdt:Locator="Match(key)"/>
<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. By default the *Web.release.config* transform file is automatically created with code that removes the `debug` attribute from the `compilation` element, and since you created the Test build configuration based on Release, the *Web.Test.config* file has this code as well:

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

The `Transform` attribute specifies that the `debug` attribute will be omitted from the deployed *Web.config* file.

Setting Connection Strings

Because you have two versions of your databases, one for development and one for the test and production environments, you need to specify a different connection string in the test and production environments. To do that, add a `<connectionStrings>` element immediately after the opening `<configuration>` tag in both the *Web.Test.config* and the *Web.Release.config* files:

```
<connectionStrings>
<add name="DefaultConnection"
       connectionString="Data Source=|DataDirectory|aspnet-Prod.sdf"
       providerName="System.Data.SqlServerCe.4.0"
       xdt:Transform="SetAttributes" xdt:Locator="Match(name)" />
<add name="SchoolContext"
       connectionString="Data Source=|DataDirectory|School-Prod.sdf"
       providerName="System.Data.SqlServerCe.4.0"
       xdt:Transform="SetAttributes" xdt:Locator="Match(name)" />
</connectionStrings>
```

This code uses the `Match` locator and `SetAttributes` transform attributes the same way you used them for the environment indicator.

You have now specified all of the *Web.config* transformations you need for deployment 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 and the question **Can I exclude specific files or folders from deployment** in ASP.NET Web Application Project Deployment FAQ.

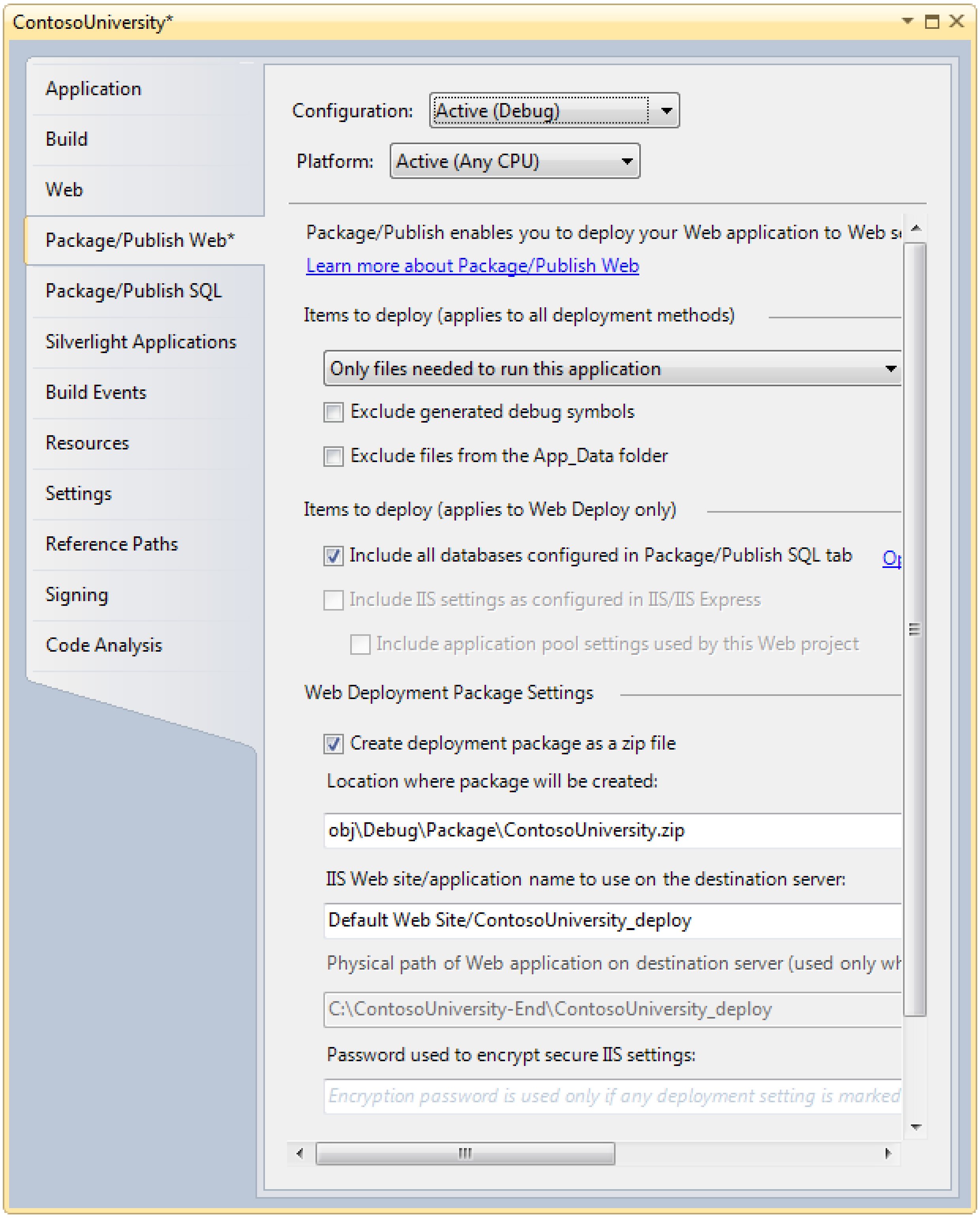
Overview

Some deployment options are configured in project properties that are stored in the project file (the *.csproj* or *.vbproj* file). In mose cases, you can use the **Project Properties** UI built into Visual Studio to work with these settings. In this tutorial you will review and update the deployment settings in **Project Properties**.

Configuring Deployment Settings in the Project Properties Window

Project settings that affect what happens during deployment can be set 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 Test or Debug build. In this section of the tutorial you will review the default settings and make one change to them.

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


ContosoUniversity*

- Application
- Build
- Web
- Package/Publish Web*
- Package/Publish SQL
- Silverlight Applications
- Build Events
- Resources
- Settings
- Reference Paths
- Signing
- Code Analysis

Configuration: Active (Debug)

Platform: Active (Any CPU)

Package/Publish enables you to deploy your Web application to Web s

Learn more about Package/Publish Web

Items to deploy (applies to all deployment methods)

Only files needed to run this application

[ ] Exclude generated debug symbols

[ ] Exclude files from the App_Data folder

Items to deploy (applies to Web Deploy only)

[✓] Include all databases configured in Package/Publish SQL tab Op

[ ] Include IIS settings as configured in IIS/IIS Express

[ ] Include application pool settings used by this Web project

Web Deployment Package Settings

[✓] Create deployment package as a zip file

Location where package will be created:

obj\Debug\Package\ContosoUniversity.zip

IIS Web site/application name to use on the destination server:

Default Web Site/ContosoUniversity_deploy

Physical path of Web application on destination server (used only wh

C:\ContosoUniversity-End\ContosoUniversity_deploy

Password used to encrypt secure IIS settings:

Encryption password is used only if any deployment setting is marked

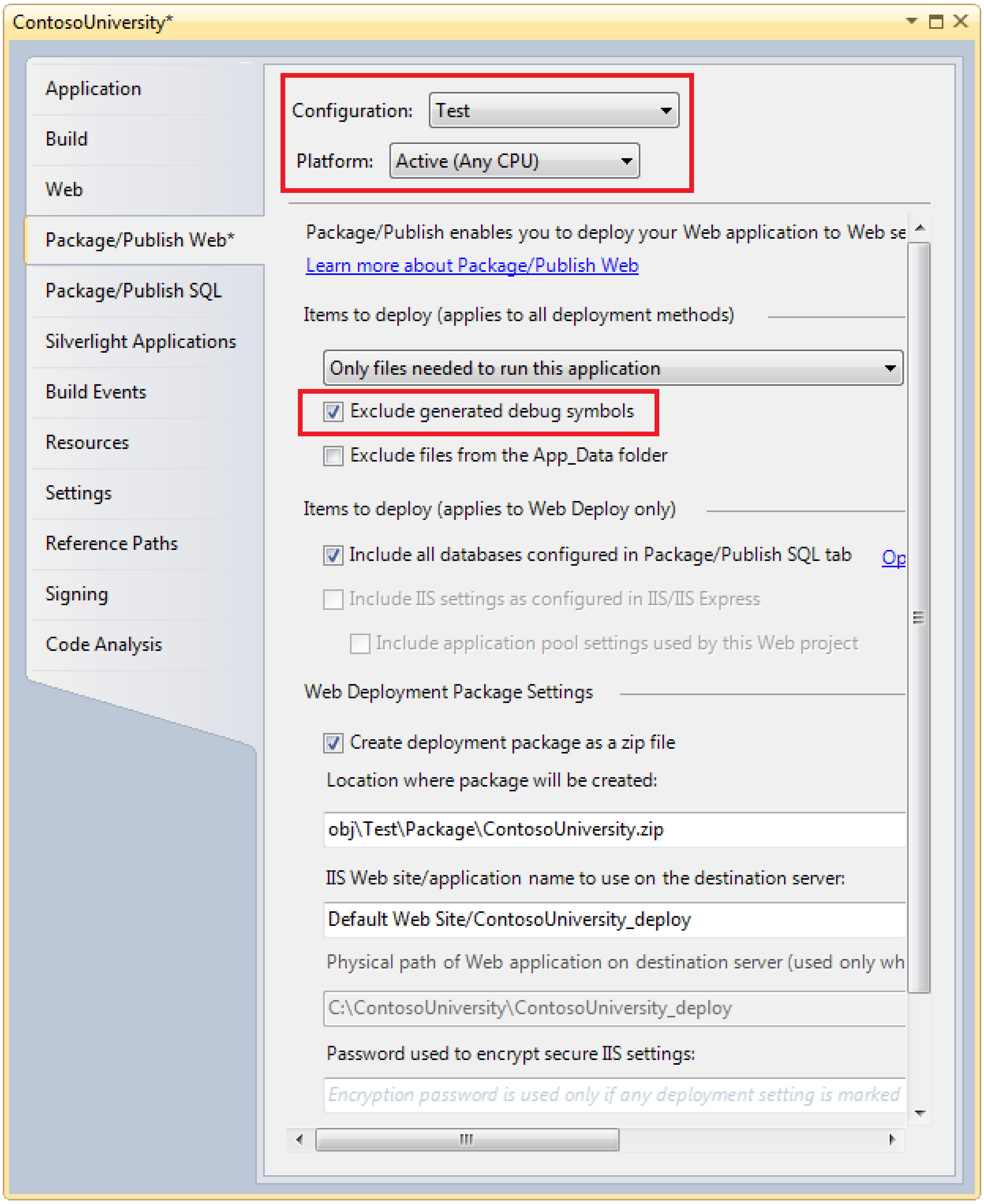
When the dialog box is displayed, it defaults to showing settings for whichever build configuration is currently active for the solution. If the **Configuration** box does not indicate

Active (Test), select **Test** in order to display settings for the Test build configuration, which you will use for deploying to your test environment.

With **Active (Test)** or **Test** selected you see the default values that will be effective when you deploy using the Test build configuration, and you can change one of them:

- Only files needed to run the application will be deployed. Other options are **All files in this project** or **All files in this project folder**. By leaving the default selection unchanged you avoid unnecessarily 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.
- Select **Exclude generated debug symbols** because you are deploying a Release build to Test and will not be debugging when you use this build configuration.
- You do not want to select **Exclude files from the App_Data folder** because your SQL Server Compact database files are in that folder and you need to deploy them. When you deploy updates that do not include database changes you will select this checkbox.
- You do not have databases configured in the **Package/Publish SQL** tab, so the **Include all databases configured in Package/Publish SQL tab** checkbox has no effect. You will use the **Package/Publish SQL** tab when you deploy to full SQL Server databases; you do not need it for SQL Server Compact databases, which are deployed as files.
- Since you will be using one-click publish in these tutorials, the **Web Deployment Package Settings** section which affects deployment packages has no effect.

The **Package/Publish Web** tab for the Test build configuration now looks like this:


ContosoUniversity*

Configuration: Test

Platform: Active (Any CPU)

Package/Publish enables you to deploy your Web application to Web se

Learn more about Package/Publish Web

Items to deploy (applies to all deployment methods)

Only files needed to run this application

[✓] Exclude generated debug symbols

[ ] Exclude files from the App_Data folder

Items to deploy (applies to Web Deploy only)

[✓] Include all databases configured in Package/Publish SQL tab Op

[ ] Include IIS settings as configured in IIS/IIS Express

[ ] Include application pool settings used by this Web project

Web Deployment Package Settings

[✓] Create deployment package as a zip file

Location where package will be created:

obj\Test\Package\ContosoUniversity.zip

IIS Web site/application name to use on the destination server:

Default Web Site/ContosoUniversity_deploy

Physical path of Web application on destination server (used only wh

C:\ContosoUniversity\ContosoUniversity_deploy

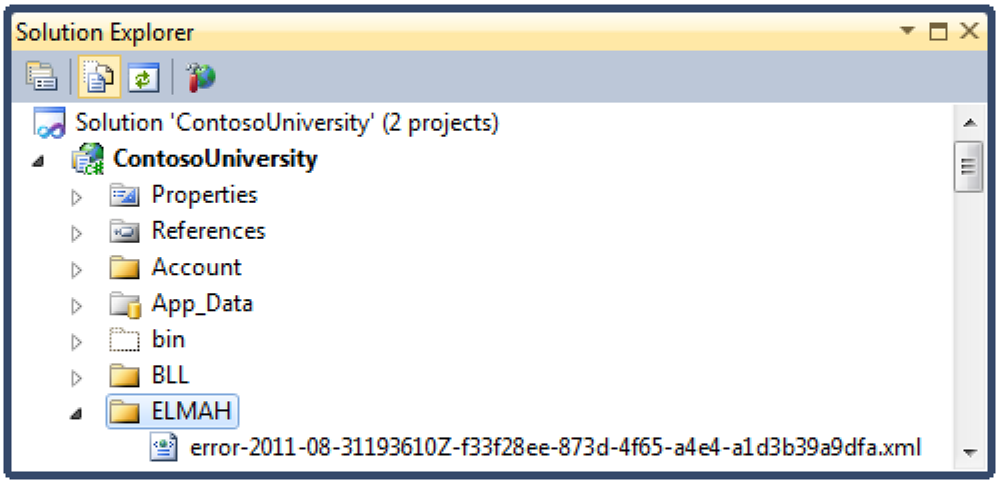
Password used to encrypt secure IIS settings:

Encryption password is used only if any deployment setting is marked

Change the **Configuration** drop-down box to Release. The default values that you see when you switch to the Release build configuration are the same, and you can select **Exclude generated debug symbols** for Release also.

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 details about each error that occurs in a folder named *Elmah*:


Solution Explorer
Solution 'ContosoUniversity' (2 projects)
ContosoUniversity
Properties
References
Account
App_Data
bin
BLL
ELMAH
error-2011-08-31193610Z-f33f28ee-873d-4f65-a4e4-a1d3b39a9dfa.xml

Excluding specific files or folders from deployment is a common requirement; another example would be a folder that users can upload files to. You do not 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 do not 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.)

You set the **Items to deploy** option in the **Package/Publish Web** tab 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 about this, 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 need to do for this file and the folder it's in to be deployed, 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 will deploy the Contoso University site to the test environment and test it there.

Deploying to IIS as a Test Environment - 5 of 12

Overview

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 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 using the same process that you will use later to deploy it to your production environment. This 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 will be a third-party hosting provider, the ideal test environment would be a second account with the hosting provider that you use only for testing, but that is 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 will change a Web.config file setting in order to make the site run more like it will in a 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 will 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 below.

```
<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, which will enable you to catch as early as possible any attempts by it to do something that requires high trust and would fail in production.

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 already have installed IIS, Web Deploy or any of their required components, the Web Platform Installer will install only what is missing.

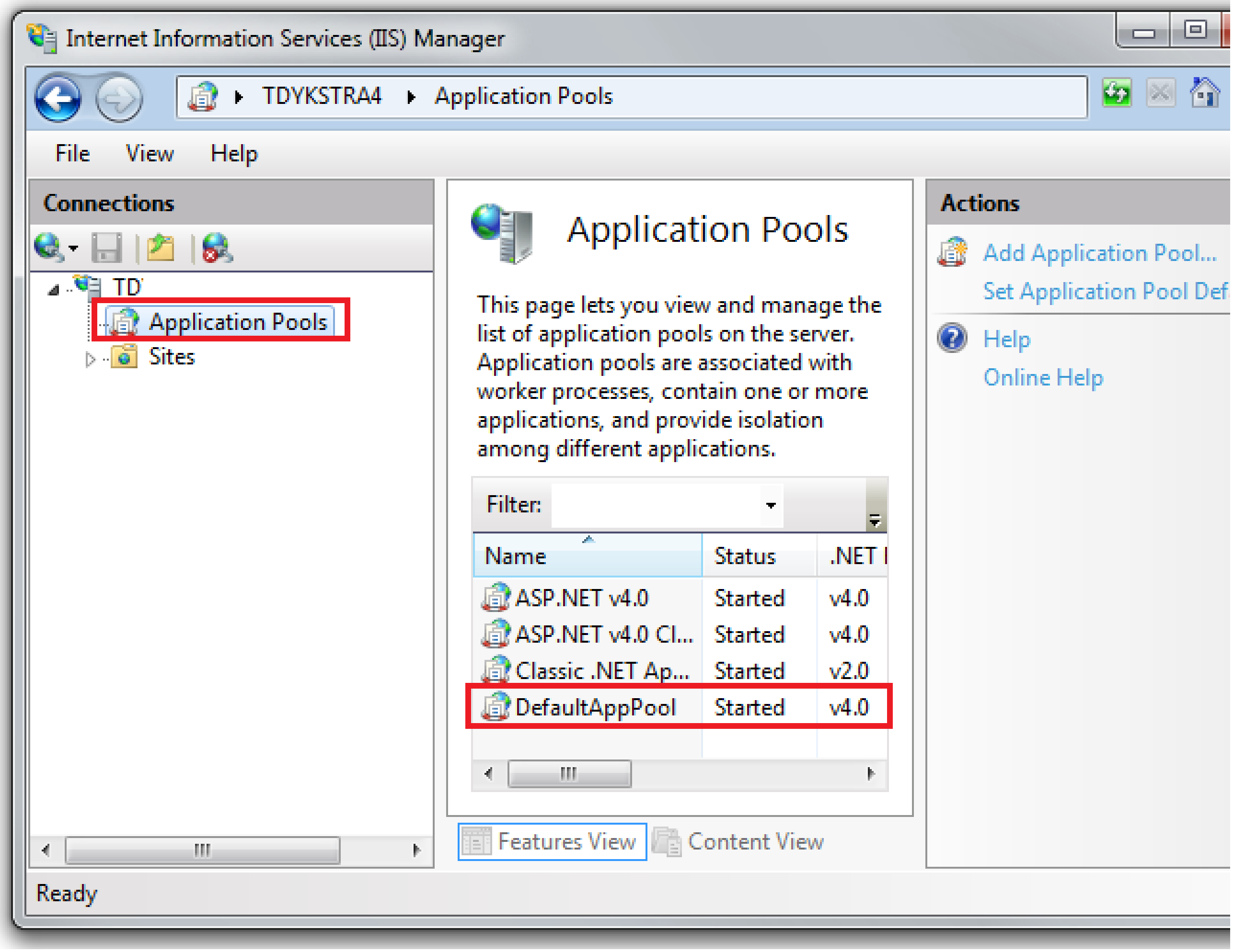
- Install IIS and Web Deploy using WebPI

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**.

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.


Internet Information Services (IIS) Manager
TDYKSTRA4 ▸ Application Pools
File
View
Help
Connections
TD
Application Pools
Sites
Application Pools

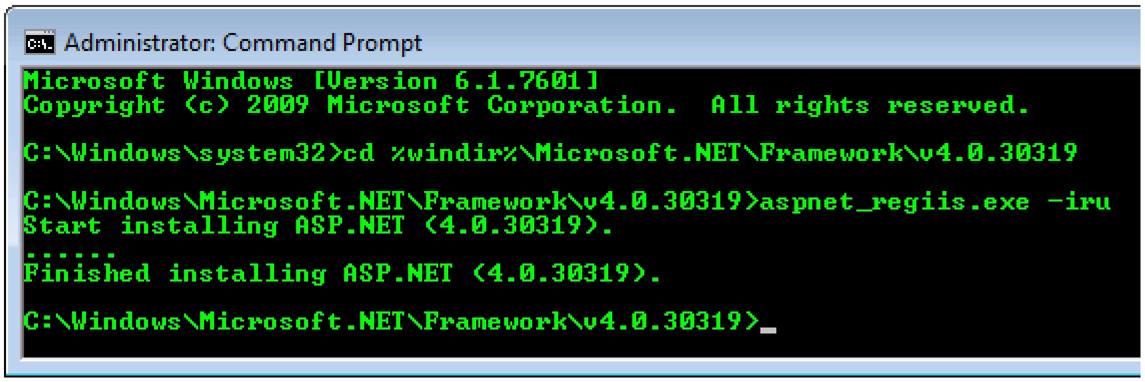
This page lets you view and manage the list of application pools on the server. Application pools are associated with worker processes, contain one or more applications, and provide isolation among different applications.

Name	Status	.NET
ASP.NET v4.0	Started	v4.0
ASP.NET v4.0 Cl...	Started	v4.0
Classic .NET Ap...	Started	v2.0
DefaultAppPool	Started	v4.0

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

Open a command prompt window by right-clicking **Cmd** 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:

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

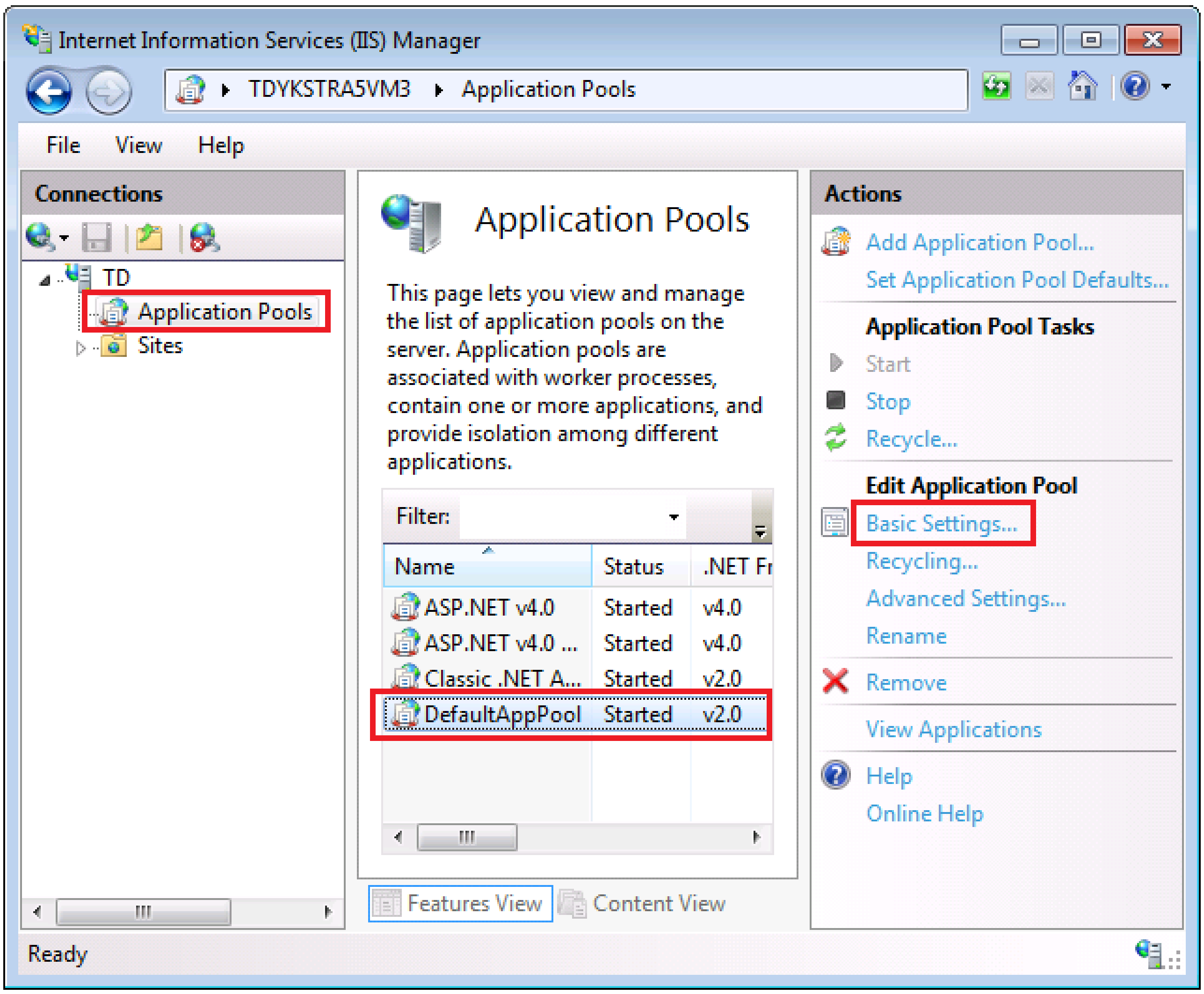

Administrator: Command Prompt

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>cd %windir%\Microsoft.NET\Framework\v4.0.30319

C:\Windows\Microsoft.NET\Framework\v4.0.30319>aspnet_regiis.exe -iru
Start installing ASP.NET (4.0.30319).
......
Finished installing ASP.NET (4.0.30319).

C:\Windows\Microsoft.NET\Framework\v4.0.30319>_
```

This 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 need to change the application pool to .NET 4.

Run **IIS Manager** 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**.



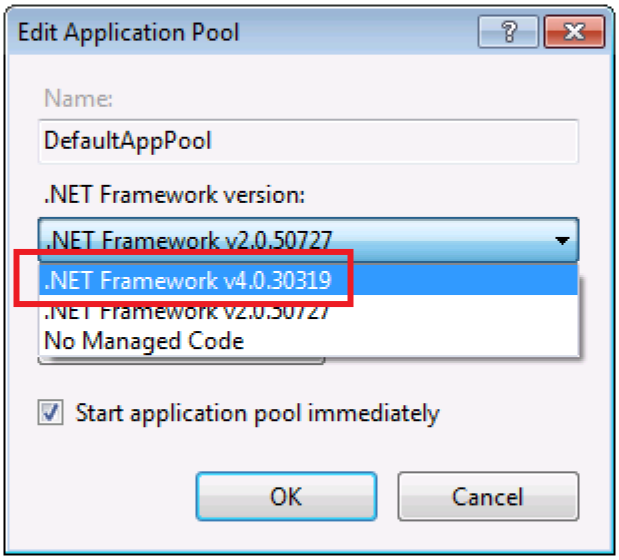
Application Pools

This page lets you view and manage the list of application pools on the server. Application pools are associated with worker processes, contain one or more applications, and provide isolation among different applications.

Filter:

Name	Status	.NET Fr
ASP.NET v4.0	Started	v4.0
ASP.NET v4.0 ...	Started	v4.0
Classic .NET A...	Started	v2.0
DefaultAppPool	Started	v2.0

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


Edit Application Pool
Name:
DefaultAppPool
.NET Framework version:
.NET Framework v2.0.50727
.NET Framework v4.0.30319
.NET Framework v2.0.50727
No Managed Code
Start application pool immediately
OK
Cancel

You are now ready to publish to IIS.

Publishing to IIS

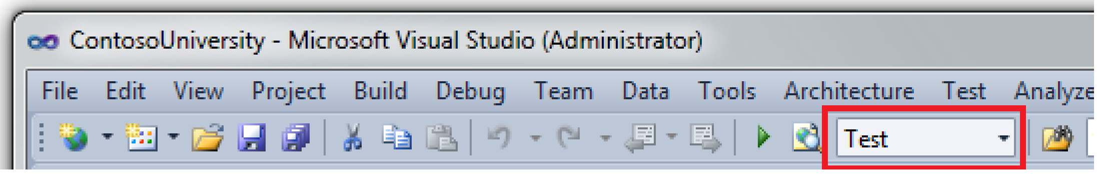
There are several ways you can deploy using Visual Studio 2010 and Web Deploy:

- 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.

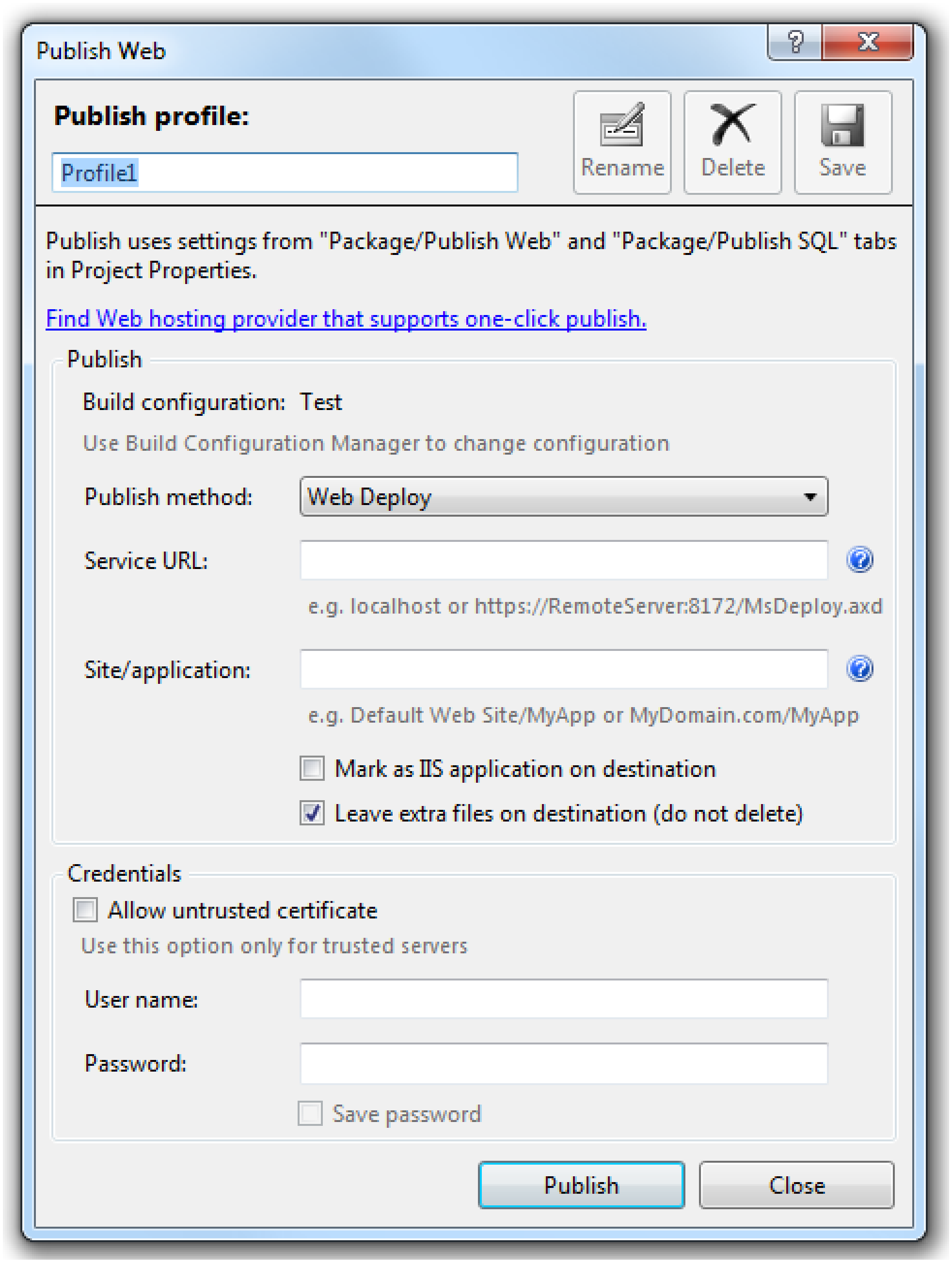
The process you went through in the previoius tutorials to set up Visual Studio to automate deployment tasks applies to all of these three methods. In these tutorials you will use the first of these methods. For information about using deployment packages, see ASP.NET Deployment Content Map.

Before publishing, make sure you are running Visual Studio in administrator mode (right-click **Visual Studio 2010** in the Windows 7 **Start** menu, and select **Run as Administrator**).

Change the active build configuration to Test. You can use the toolbar, as shown below, or select **Configuration Manager** from the **Build** menu.


ContosoUniversity - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze
Test

In **Solution Explorer**, right-click the ContosoUniversity project and select **Publish**. The **Publish Web** dialog box appears.


Publish Web

Publish profile: Profile1

Publish uses settings from "Package/Publish Web" and "Package/Publish SQL" tabs in Project Properties.

Find Web hosting provider that supports one-click publish.

Publish

Build configuration: Test

Use Build Configuration Manager to change configuration

Publish method: Web Deploy

Service URL:

e.g. localhost or https://RemoteServer:8172/MsDeploy.axd

Site/application:

e.g. Default Web Site/MyApp or MyDomain.com/MyApp

☐ Mark as IIS application on destination

☑ Leave extra files on destination (do not delete)

Credentials

☐ Allow untrusted certificate

Use this option only for trusted servers

User name:

Password:

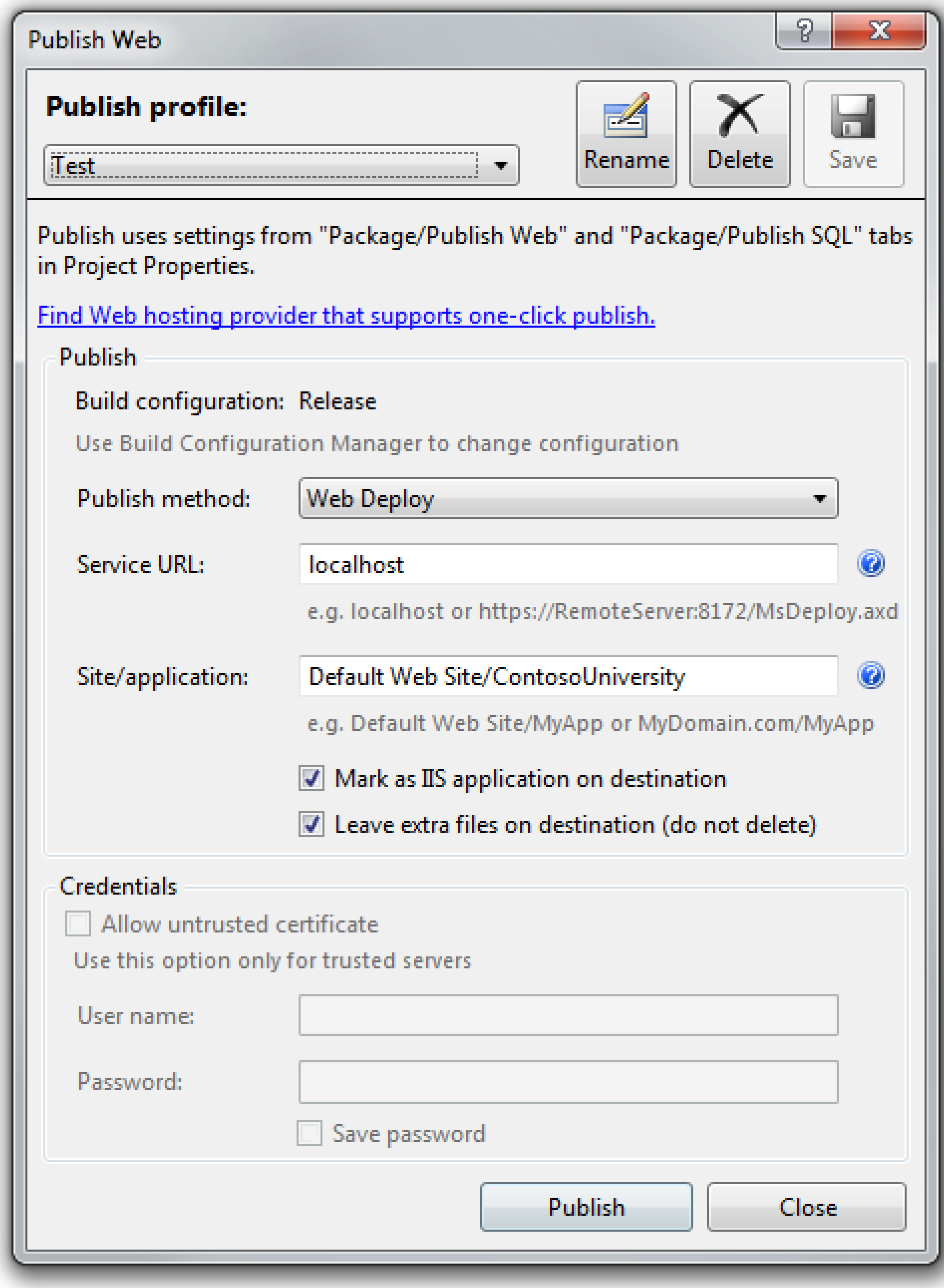
☐ Save password

Change the profile name to "Test". (If you see a drop-down box instead of a text box, select **new** from the drop-down in order to be able to enter a profile name.)

Enter "localhost" for the **Service URL**.

Enter "Default Web Site/ContosoUniversity" for the **Site/application**.

Select **Mark as IIS application on destination**. (There are very few scenarios in which you would not want to deploy a web project as an IIS application. One example is if the project that you're deploying is not actually a web application but only contains virtual directory content, such as images, XML files, and so forth. In that case, you might want to clear this check box so that the deployed project is not set up as an IIS application.)


Publish Web

Publish profile: Test

Publish uses settings from "Package/Publish Web" and "Package/Publish SQL" tabs in Project Properties.

Find Web hosting provider that supports one-click publish.

Publish

Build configuration: Release

Use Build Configuration Manager to change configuration

Publish method: Web Deploy

Service URL: localhost

e.g. localhost or https://RemoteServer:8172/MsDeploy.axd

Site/application: Default Web Site/ContosoUniversity

e.g. Default Web Site/MyApp or MyDomain.com/MyApp

[✓] Mark as IIS application on destination

[✓] Leave extra files on destination (do not delete)

Credentials

[ ] Allow untrusted certificate

Use this option only for trusted servers

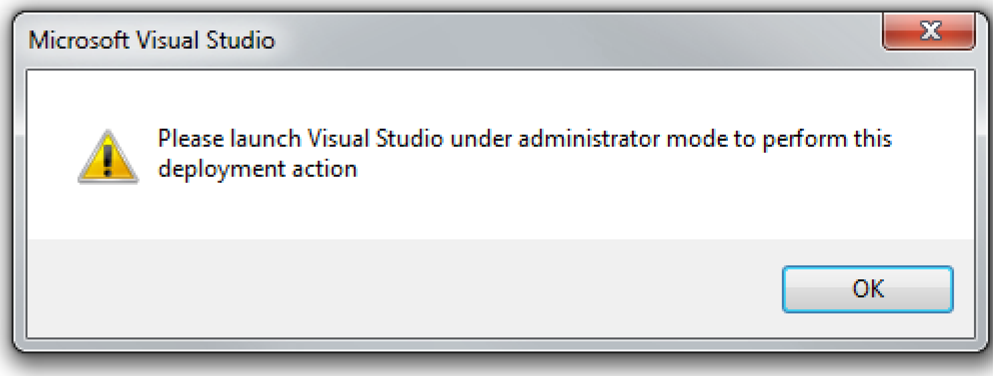
User name:

Password:

[ ] Save password

Click **Publish**.

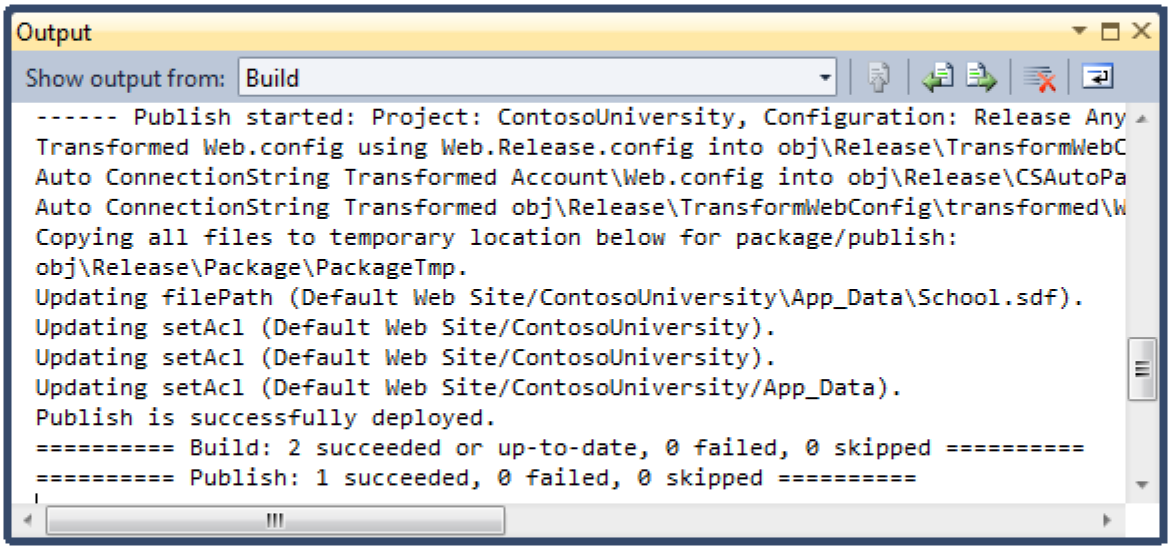
If you get the dialog box shown below, you need to close Visual Studio and re-open it in administrator mode, as directed above.


Microsoft Visual Studio

Please launch Visual Studio under administrator mode to perform this deployment action

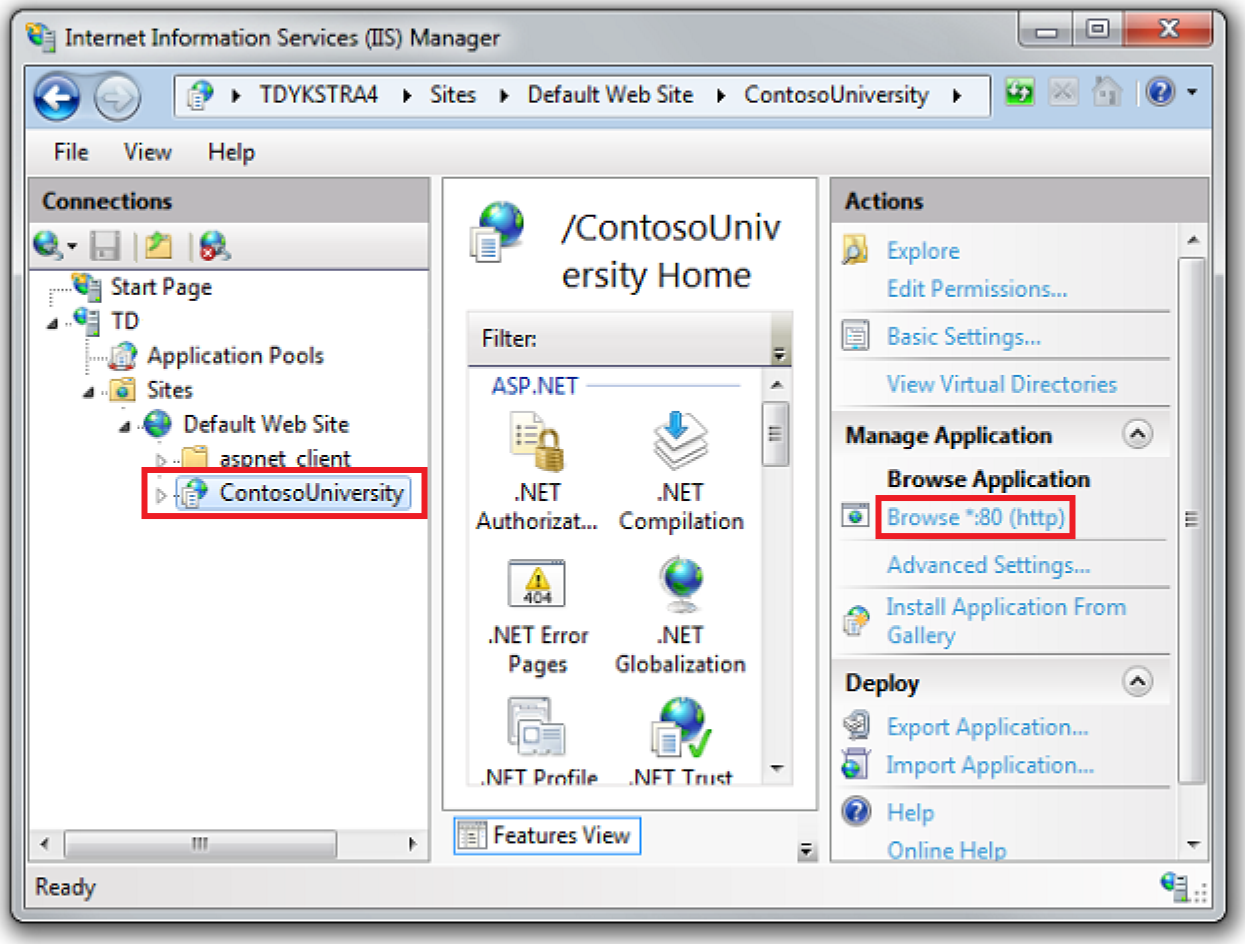
If you close Visual Studio and reopen it while following the directions in these tutorials, you will need to remember each time to run Visual Studio in administrator mode. Note that if you fail to do that, you will not always get this helpful dialog box when you click **Publish**; sometimes Visual Studio will attempt to publish but will fail with a message that indicates a permissions failure.

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

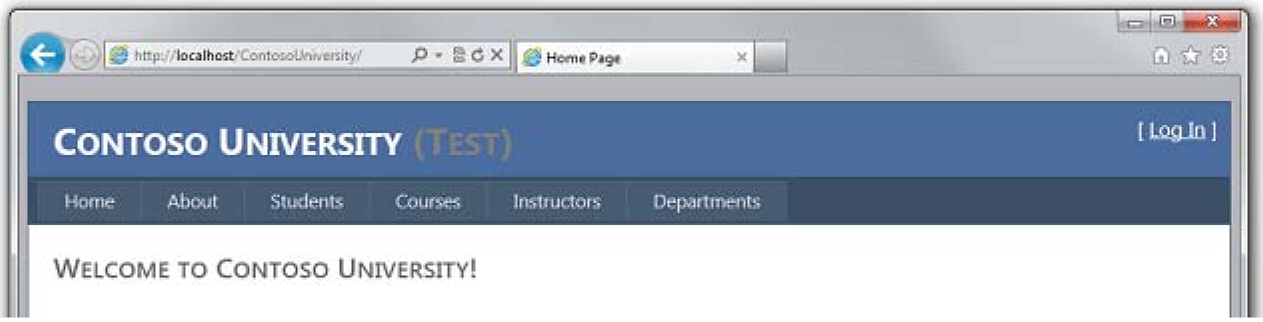


```
------ Publish started: Project: ContosoUniversity, Configuration: Release Any
Transformed Web.config using Web.Release.config into obj\Release\TransformWebC
Auto ConnectionString Transformed Account\Web.config into obj\Release\CSAutoPa
Auto ConnectionString Transformed obj\Release\TransformWebConfig\transformed\W
Copying all files to temporary location below for package/publish:
obj\Release\Package\PackageTmp.
Updating filePath (Default Web Site/ContosoUniversity\App_Data\School.sdf).
Updating setAcl (Default Web Site/ContosoUniversity).
Updating setAcl (Default Web Site/ContosoUniversity).
Updating setAcl (Default Web Site/ContosoUniversity/App_Data).
Publish is successfully deployed.
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

Run IIS Manager, and you now see the new **ContosoUniversity** application under **Default Web Site** in the **Connections** pane. (If IIS Manager was already running, you will have to click **Refresh**.) In the **Actions** pane click **Browse *:80 (http)** to verify that you can browse to your home page.

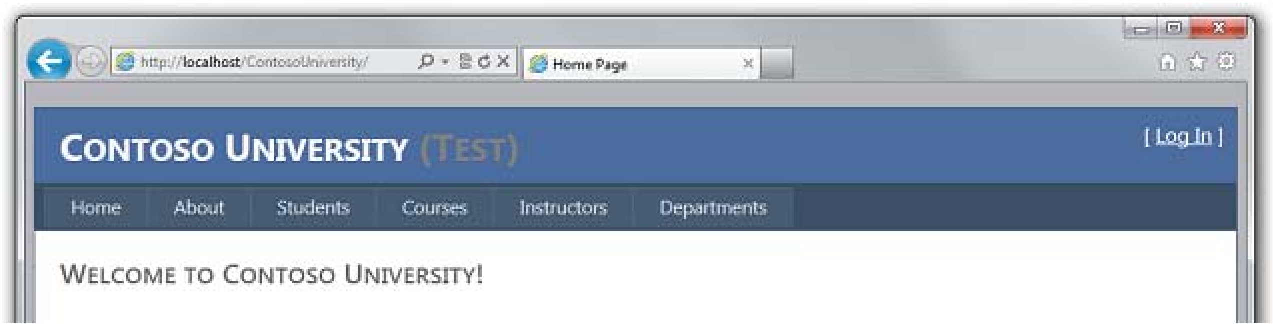


The Contoso University Home page appears in the browser.


http://localhost/ContosoUniversity/
Home Page
Contoso University (Test)
[Log In]
Home
About
Students
Courses
Instructors
Departments
Welcome to Contoso University!

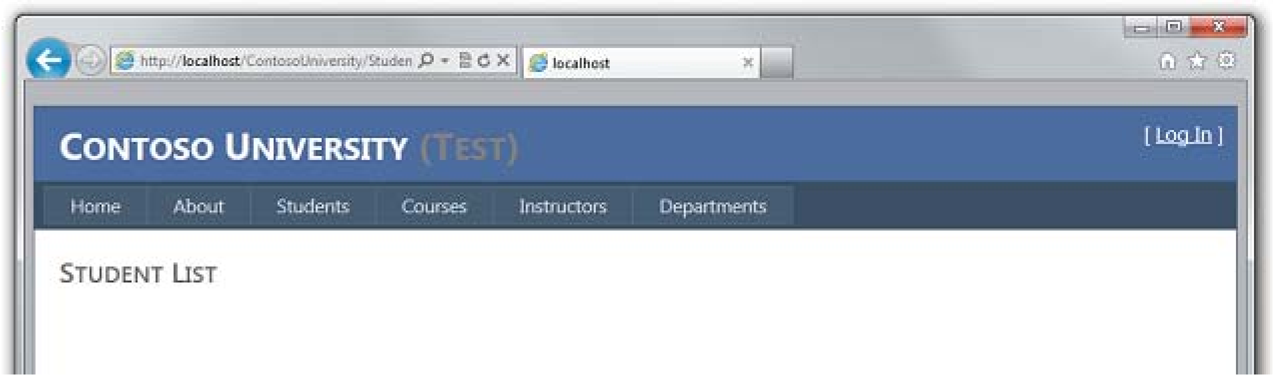
Testing in the Test Environment

Open a browser and go to the URL http://localhost/ContosoUniversity to run the Home page again. Notice that the environment indicator shows "(Test)" instead of "(Dev)", which shows that the *Web.config* transformation for the environment indicator was successful.



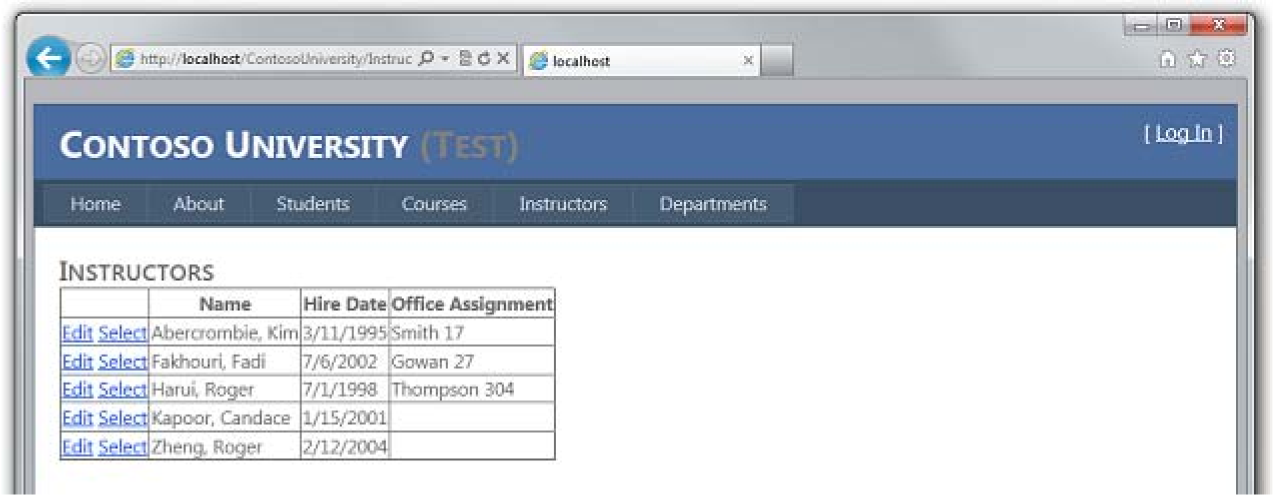
WELCOME TO CONTOSO UNIVERSITY!

Run the **Students** page to verify the deployed database has no students:


http://localhost/ContosoUniversity/Studen
localhost
CONTOSO UNIVERSITY (TEST)
[Log In]
Home
About
Students
Courses
Instructors
Departments

STUDENT LIST

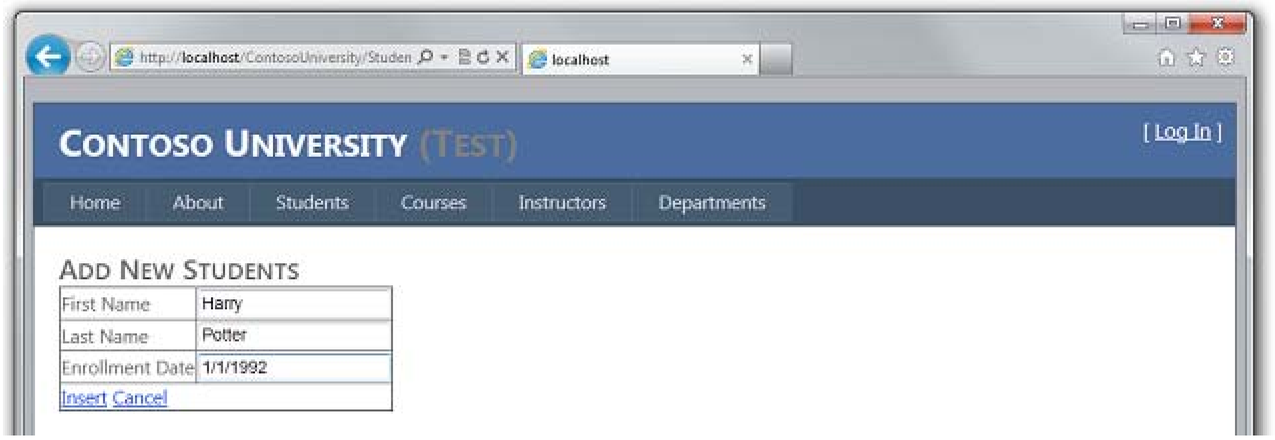
Run the **Instructors** page to verify that other data is still in the database:


http://localhost/ContosoUniversity/Instruc
localhost
CONTOSO UNIVERSITY (TEST)
[Log In]
Home
About
Students
Courses
Instructors
Departments

INSTRUCTORS

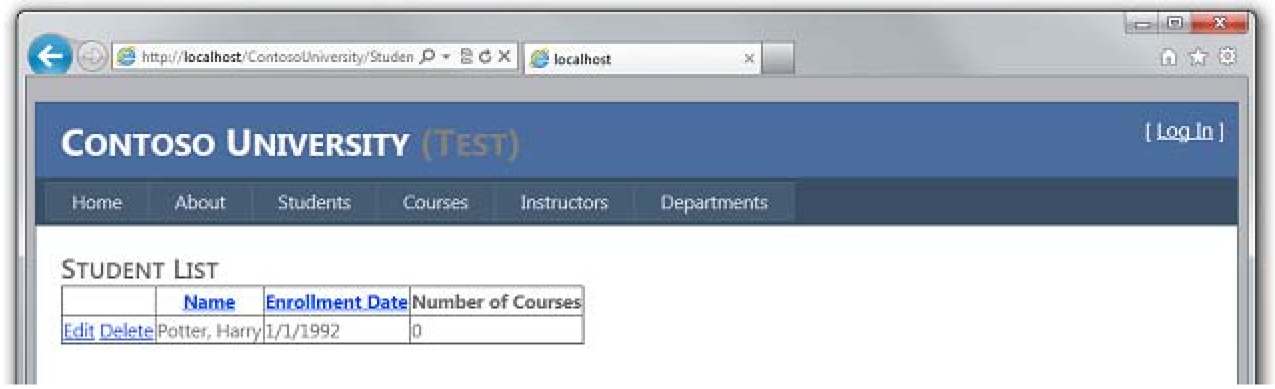
	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	Smith 17
Edit Select	Fakhouri, Fadi	7/6/2002	Gowan 27
Edit Select	Harui, Roger	7/1/1998	Thompson 304
Edit Select	Kapoor, Candace	1/15/2001	
Edit Select	Zheng, Roger	2/12/2004	

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:


http://localhost/ContosoUniversity/Studen
localhost
CONTOSO UNIVERSITY (TEST)
[Log In]
Home
About
Students
Courses
Instructors
Departments

ADD NEW STUDENTS

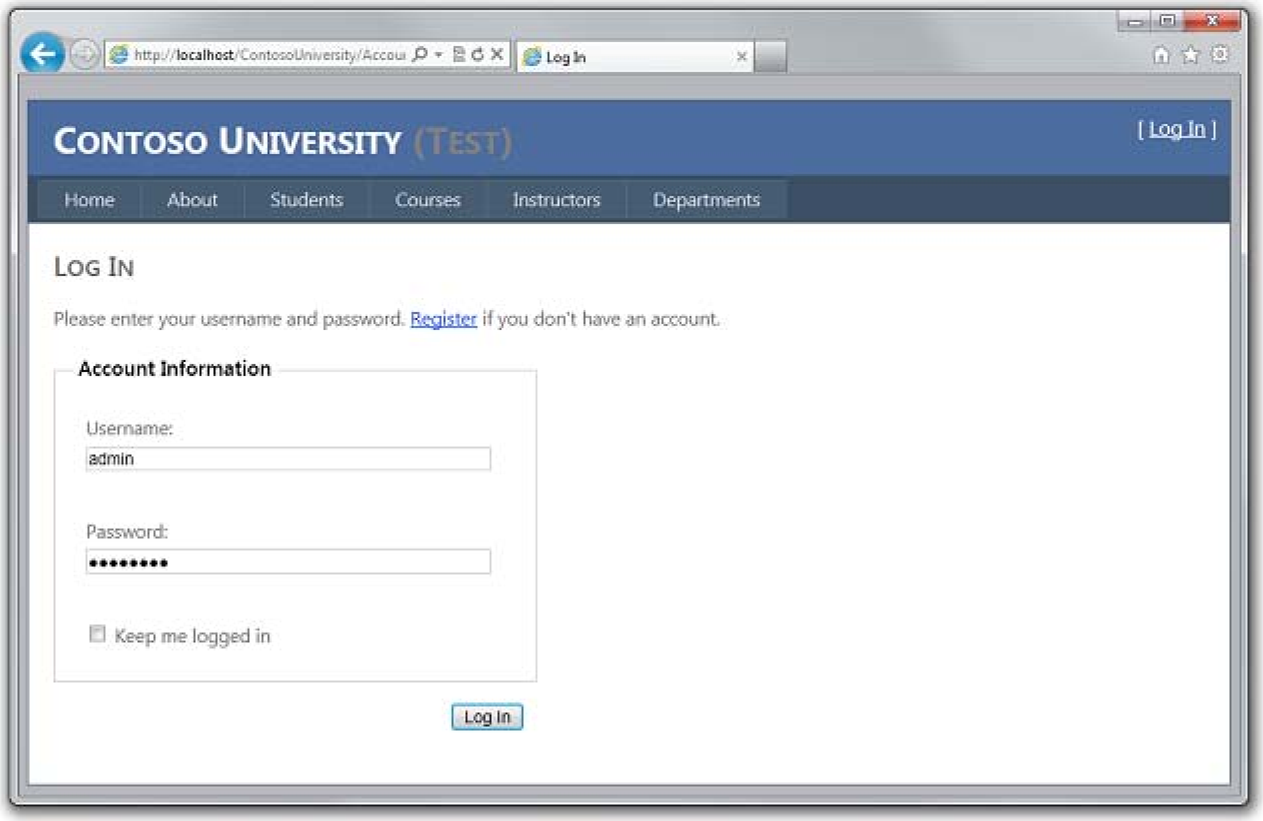
First Name	Harry
Last Name	Potter
Enrollment Date	1/1/1992
Insert Cancel	


http://localhost/ContosoUniversity/Studen
localhost
CONTOSO UNIVERSITY (TEST)
[Log In]
Home
About
Students
Courses
Instructors
Departments

STUDENT LIST

	Name	Enrollment Date	Number of Courses
Edit Delete	Potter, Harry	1/1/1992	0

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.



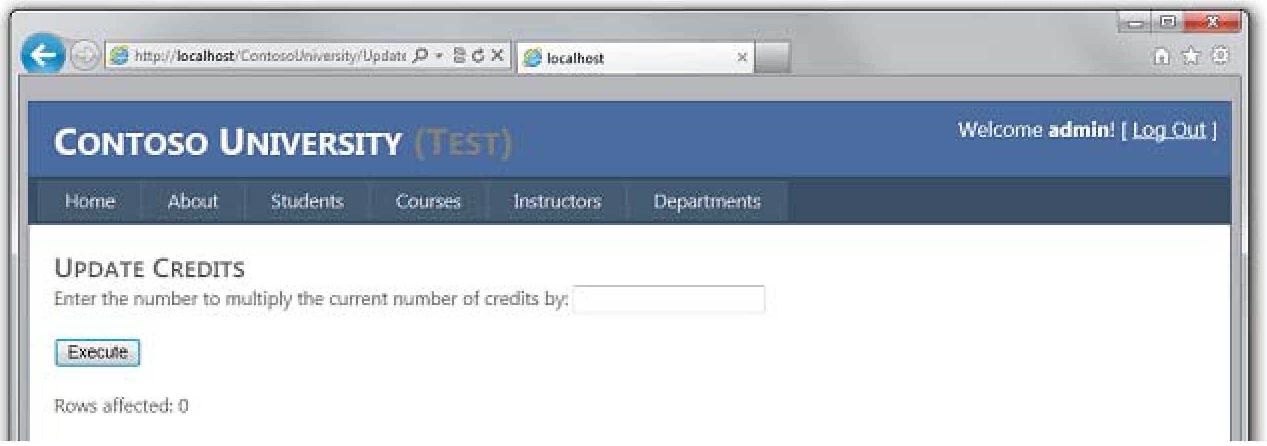
Please enter your username and password. Register if you don't have an account.

Account Information

Username: admin

Password: ●●●●●●●●●

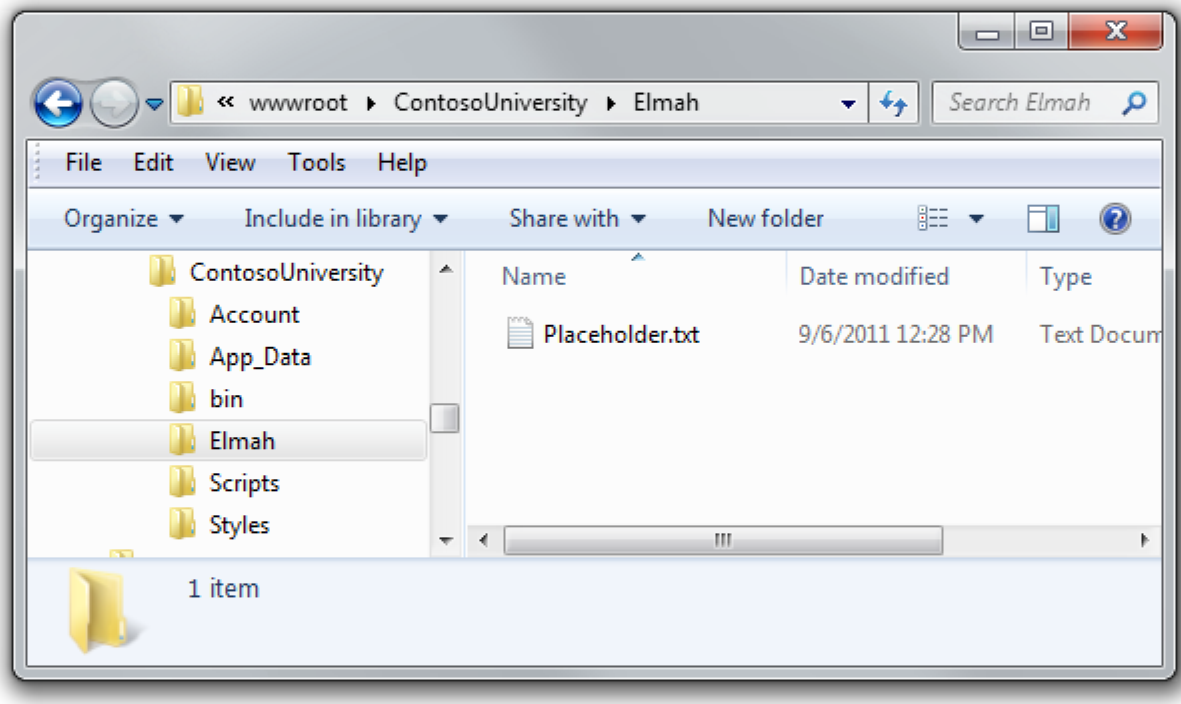
Keep me logged in


http://localhost/ContosoUniversity/Update
localhost
CONTOSO UNIVERSITY (TEST)
Welcome admin! [Log Out]
Home
About
Students
Courses
Instructors
Departments
Update Credits

Enter the number to multiply the current number of credits by:

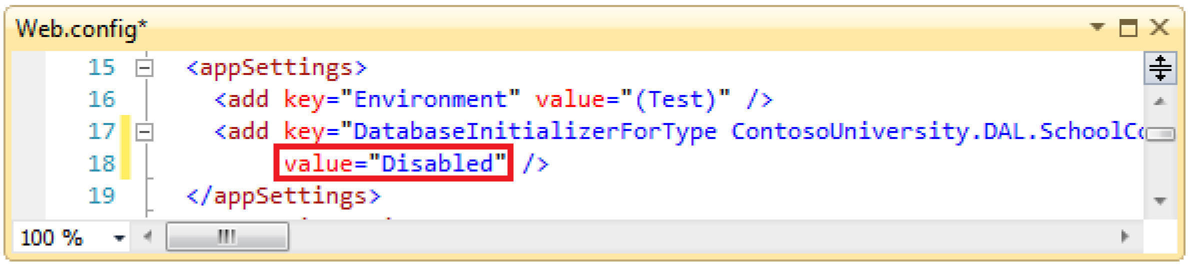
Rows affected: 0

Verify that an *Elmah* folder exists with only the placeholder file in it. This verifies that the code you set up in the project file to exclude XML files in this folder from deployment worked correctly.



Name	Date modified	Type
Placeholder.txt	9/6/2011 12:28 PM	Text Docum

Open the *Web.config* file in the deployed application at *C:\inetpub\wwwroot\ContosoUniversity* and verify that the Code First database initializer is disabled:



```
<appSettings>
  <add key="Environment" value="(Test)" />
  <add key="DatabaseInitializerForType ContosoUniversity.DAL.SchoolCo
      value="Disabled" />
</appSettings>
```

You have now deployed your application to IIS on your development computer and tested it there. This verifies that the deployment process not only 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 WebForms 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 an introduction to the deployment automation tools introduced in Visual Studio 2010, with details about one-click publish and deployment packages, see ASP.NET Web Application Project Deployment Overview.

For more information about what sort of issues might arise when you deploy an application to a hosting environment where it 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

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 will have read and execute permissions on your web application's files and folders, but it will not have write access.

This becomes an issue if your application creates or updates files, which is a common need in web applications. In Contoso University, 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 result in data being written to a folder in your site.

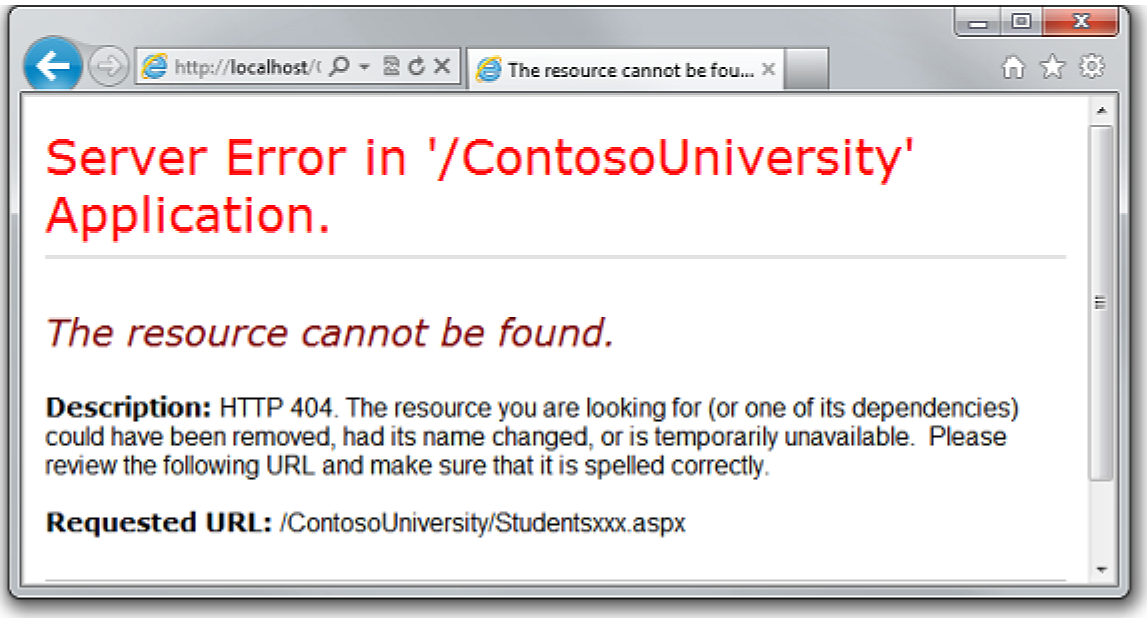
In this tutorial you'll run a test that shows that Elmah is not able to write log files, grant write permission on the Elmah folder, and rerun the test to verify that the error is corrected.

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'll 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:



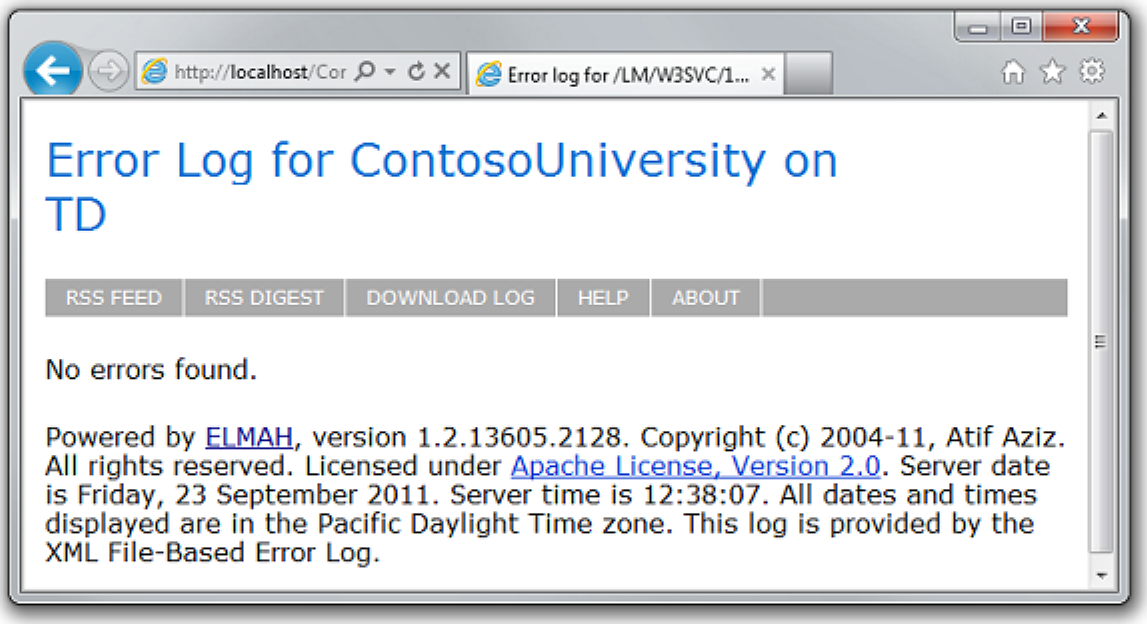
Server Error in '/ContosoUniversity' Application.

The resource cannot be found.

Description: HTTP 404. The resource you are looking for (or one of its dependencies) could have been removed, had its name changed, or is temporarily unavailable. Please review the following URL and make sure that it is spelled correctly.

Requested URL: /ContosoUniversity/Studentsxxx.aspx

Now run *Elmah.axd* to see the error report. After you log in with your administrator credentials ("admin" and "Pas$w0rd"), you see an empty error log page:



Error Log for ContosoUniversity on TD

No errors found.

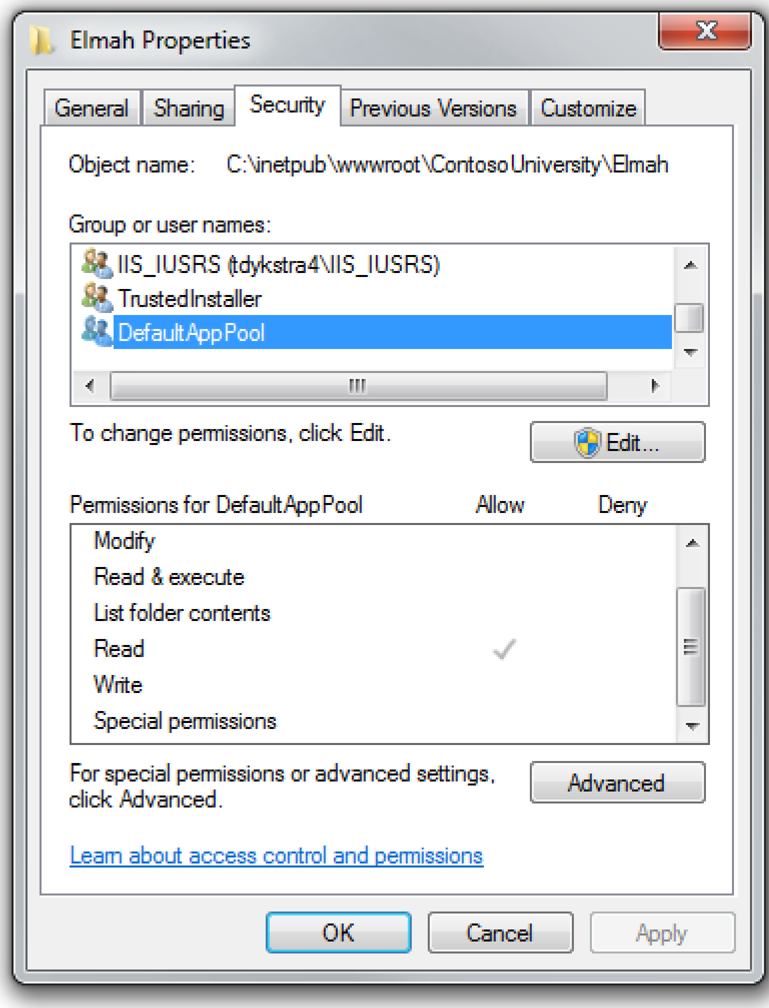
Powered by ELMAH, version 1.2.13605.2128. Copyright (c) 2004-11, Atif Aziz. All rights reserved. Licensed under Apache License, Version 2.0. Server date is Friday, 23 September 2011. Server time is 12:38:07. All dates and times displayed are in the Pacific Daylight Time zone. This log is provided by the XML File-Based Error Log.

No error is reported 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 need 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, and select **Properties**, then select the **Security** tab.



Object name: C:\inetpub\wwwroot\ContosoUniversity\Elmah

Group or user names:
IIS_IUSRS (tdykstra4\IIS_IUSRS)
TrustedInstaller
DefaultAppPool

To change permissions, click Edit.

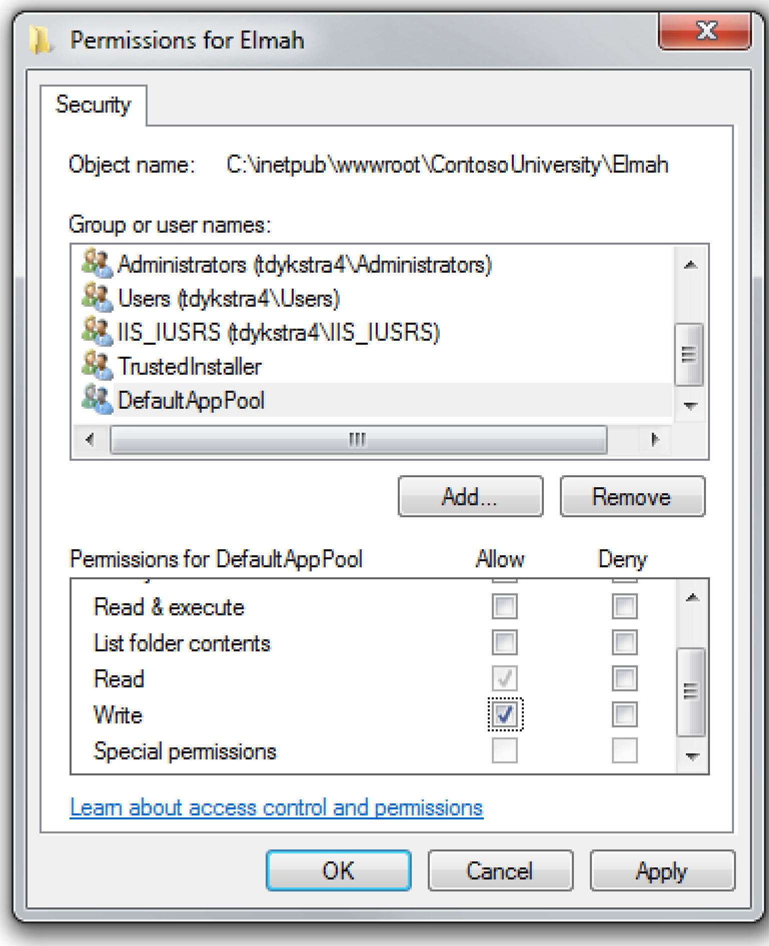
Permissions for DefaultAppPool	Allow	Deny
Modify		
Read & execute		
List folder contents		
Read	✓	
Write		
Special permissions		

For special permissions or advanced settings, click Advanced.

Learn about access control and permissions

(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, you need to 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.


Permissions for Elmah

Security

Object name: C:\inetpub\wwwroot\ContosoUniversity\Elmah

Group or user names:

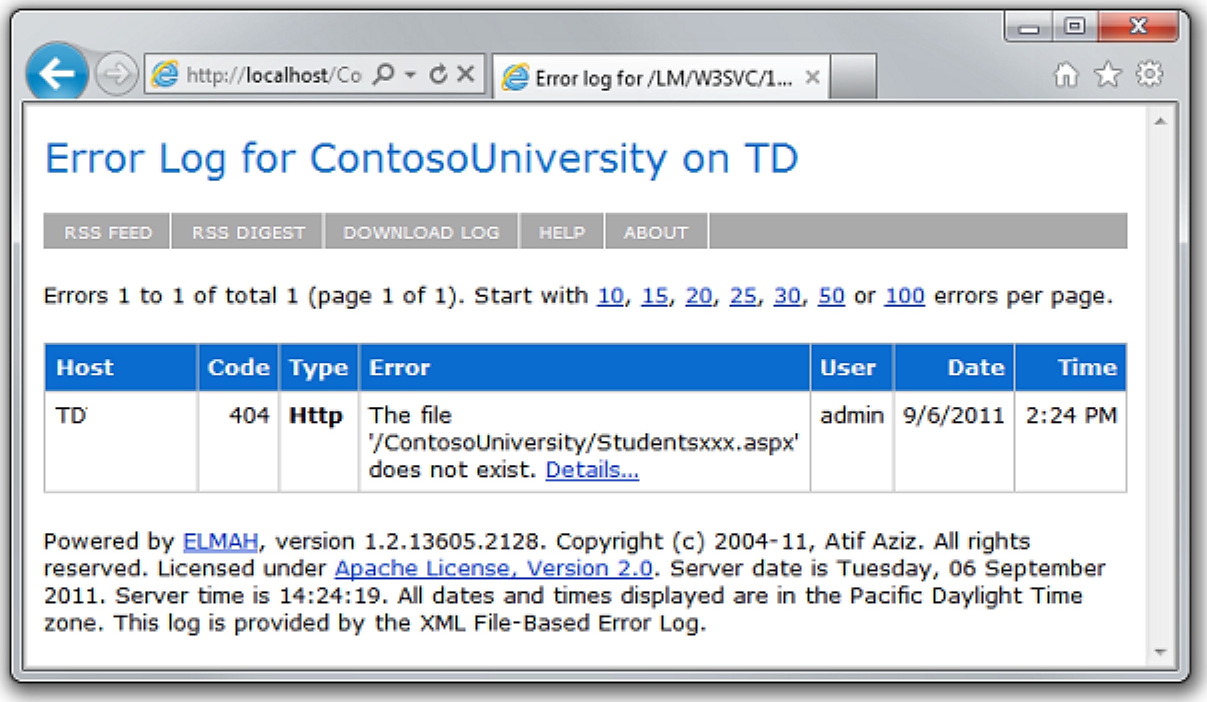
- Administrators (tdykstra4\Administrators)
- Users (tdykstra4\Users)
- IIS_IUSRS (tdykstra4\IIS_IUSRS)
- TrustedInstaller
- DefaultAppPool

Permissions for DefaultAppPool	Allow	Deny
Read & execute	☐	☐
List folder contents	☐	☐
Read	✓	☐
Write	✓	☐
Special permissions	☐	☐

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.



Error Log for ContosoUniversity on TD

RSS FEED | RSS DIGEST | DOWNLOAD LOG | HELP | ABOUT

Errors 1 to 1 of total 1 (page 1 of 1). Start with 10, 15, 20, 25, 30, 50 or 100 errors per page.

Host	Code	Type	Error	User	Date	Time
TD	404	**Http**	The file '/ContosoUniversity/Studentsxxx.aspx' does not exist. Details...	admin	9/6/2011	2:24 PM

Powered by ELMAH, version 1.2.13605.2128. Copyright (c) 2004-11, Atif Aziz. All rights reserved. Licensed under Apache License, Version 2.0. Server date is Tuesday, 06 September 2011. Server time is 14:24:19. All dates and times displayed are in the Pacific Daylight Time zone. This log is provided by the XML File-Based Error Log.

Updating a database also requires write permissions for the *App_Data* folder if the database is in a file in that folder, as the SQL Compact databases are. In that case, however, you did not 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

Now that you have deployed to IIS and tested there, you are ready to make the application available to the public. In this tutorial you will set up an account with a hosting provider and deploy your application to 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.

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 using the gallery of providers on the Microsoft.com/web site.

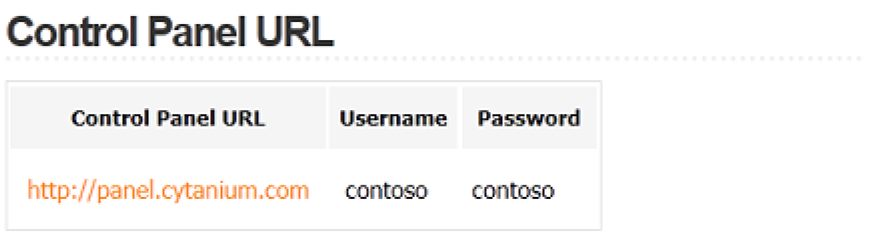
Creating an Account

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 will need it for later tutorials in the series that show how to migrate to full SQL Server.

For these tutorials you don't need to register a new domain name. You can select the option to use an existing domain name, enter whatever name you want, and test to verify successful deployment using the temporary URL assigned to the site by the provider.

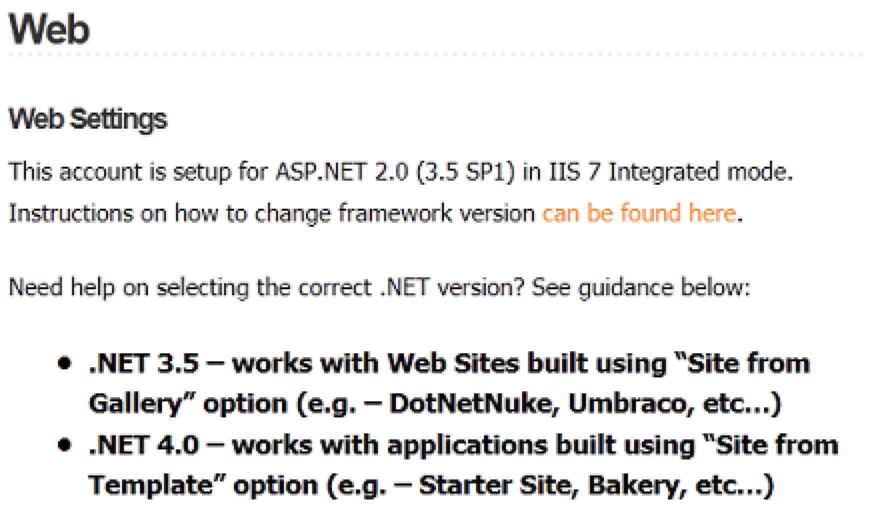
After the account has been created, you typically receive a welcome email that contains all the information you need to deploy and manage your site. Again, the specifics of the information that your hosting provider sends you might be slightly different, but will be similar to those shown here. The Cytanium 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 will need to change this setting later.


Web

Web Settings

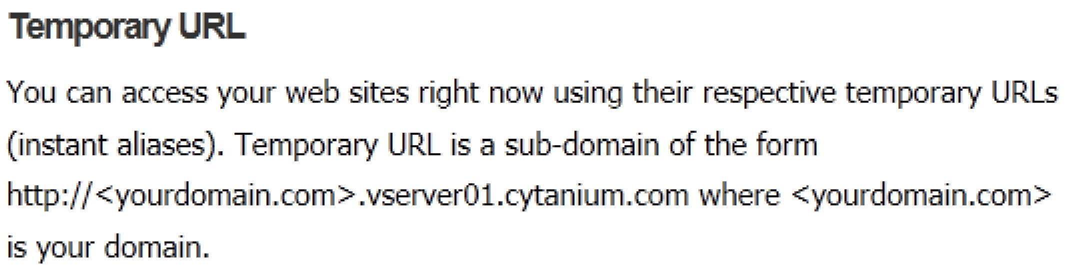
This account is setup for ASP.NET 2.0 (3.5 SP1) in IIS 7 Integrated mode.

Instructions on how to change framework version can be found here.

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

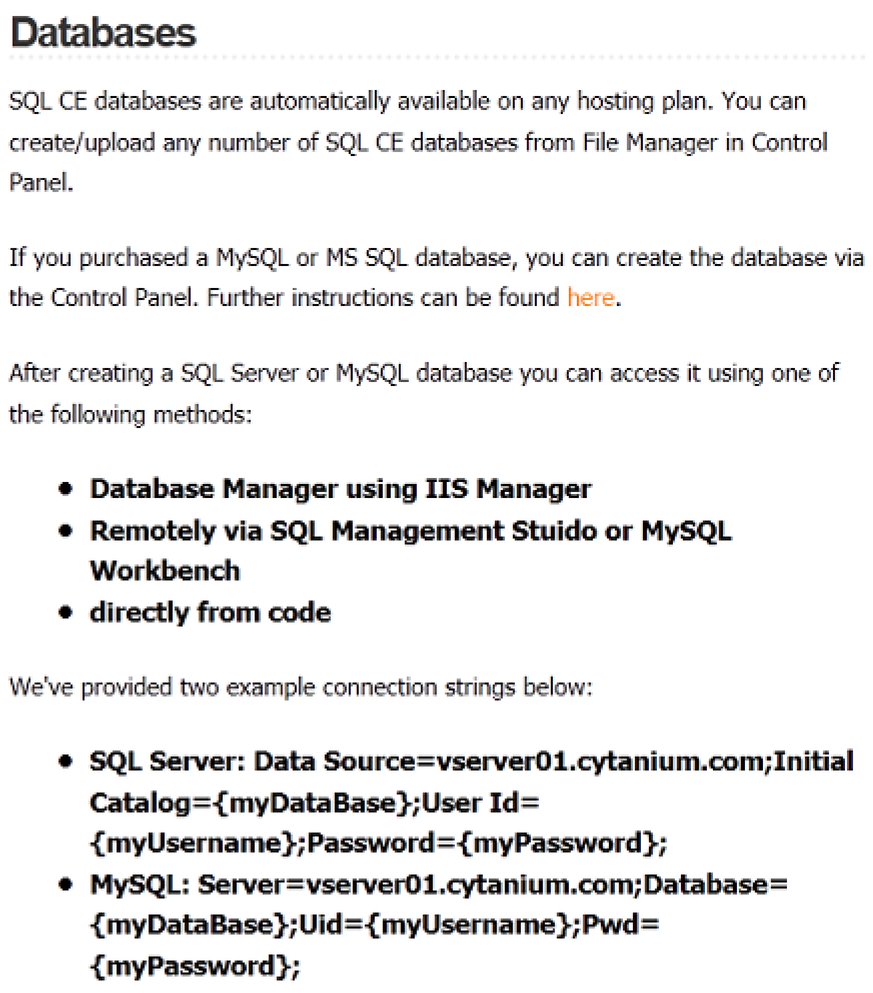
- **.NET 3.5 – works with Web Sites built using "Site from Gallery" option (e.g. – DotNetNuke, Umbraco, etc...)**
- **.NET 4.0 – works with applications built using "Site from Template" option (e.g. – Starter Site, Bakery, etc...)**

- 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 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.

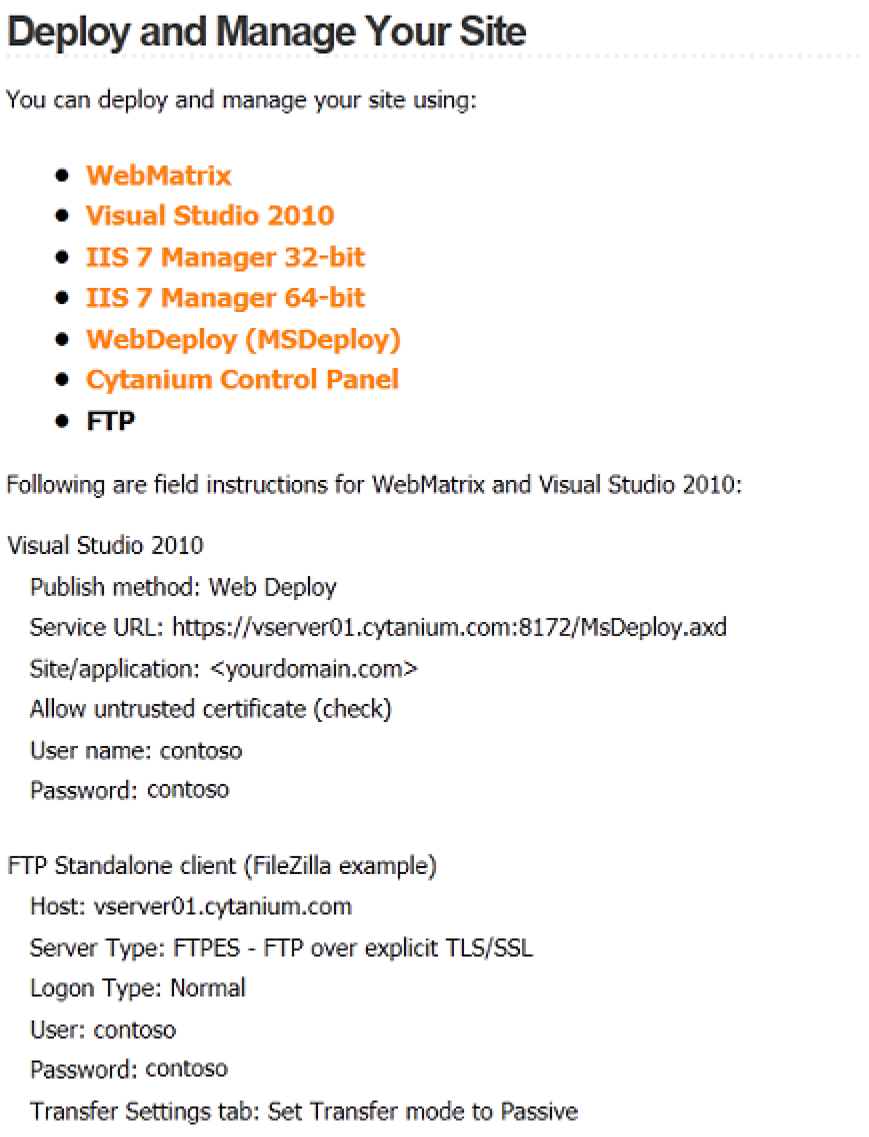
After creating a SQL Server or MySQL database you can access it using one of the following methods:

- **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:

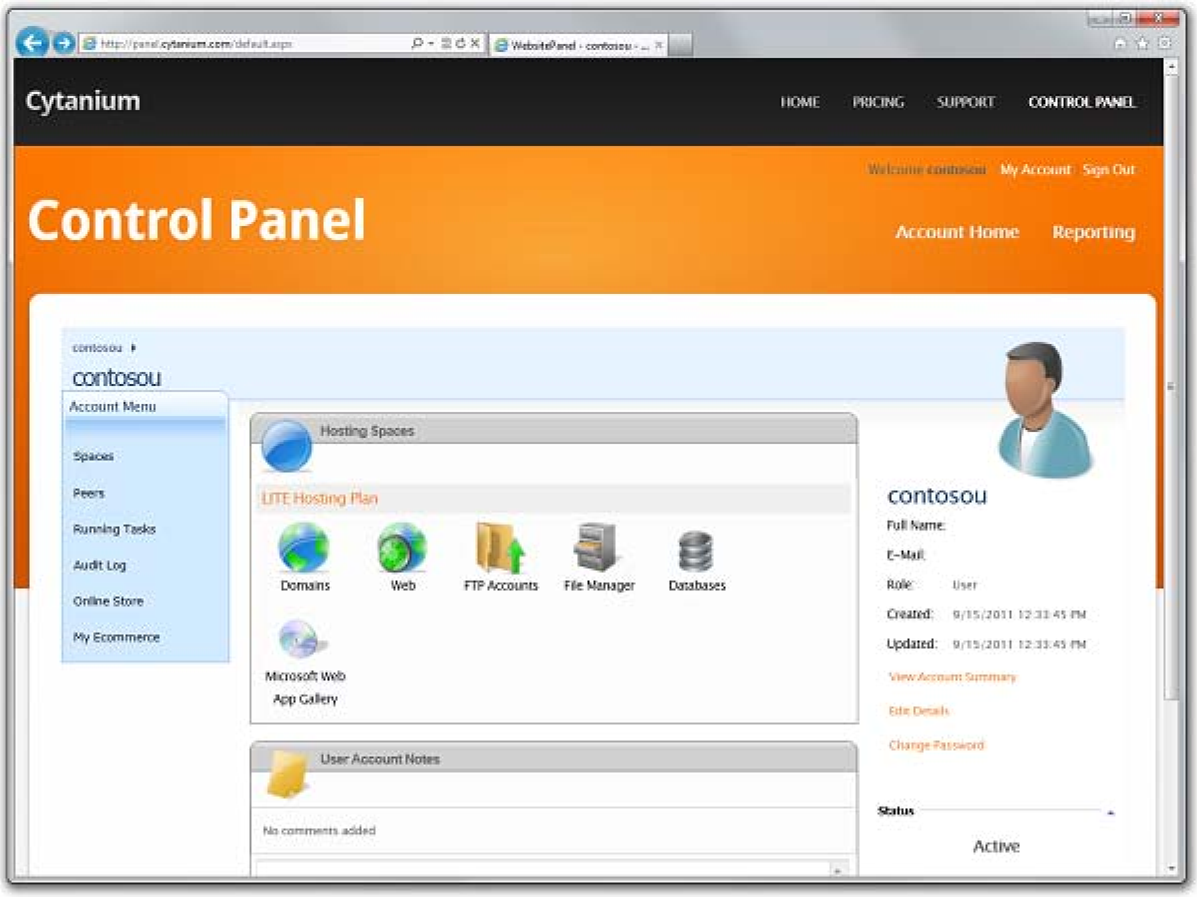
Visual Studio 2010
Publish method: Web Deploy
Service URL: https://vserver01.cytanium.com:8172/MsDeploy.axd
Site/application: <yourdomain.com>
Allow untrusted certificate (check)
User name: contoso
Password: contoso

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.

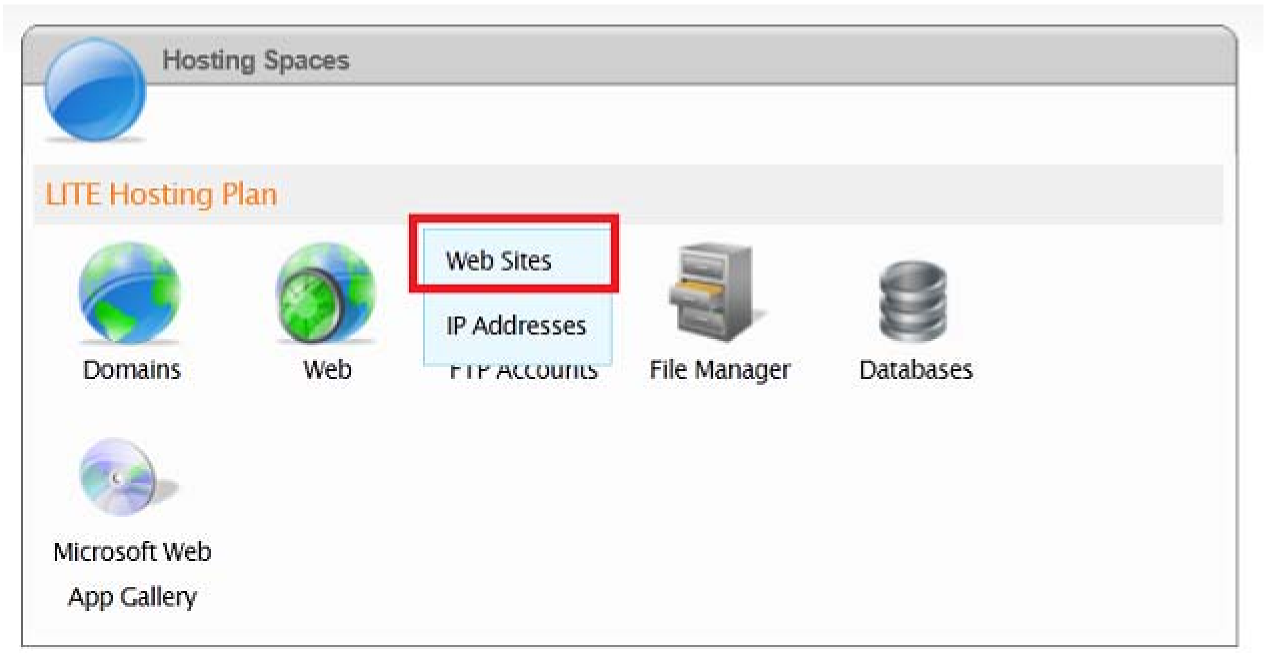
Go to the control panel URL. After logging in with your user name and password, you see the control panel.



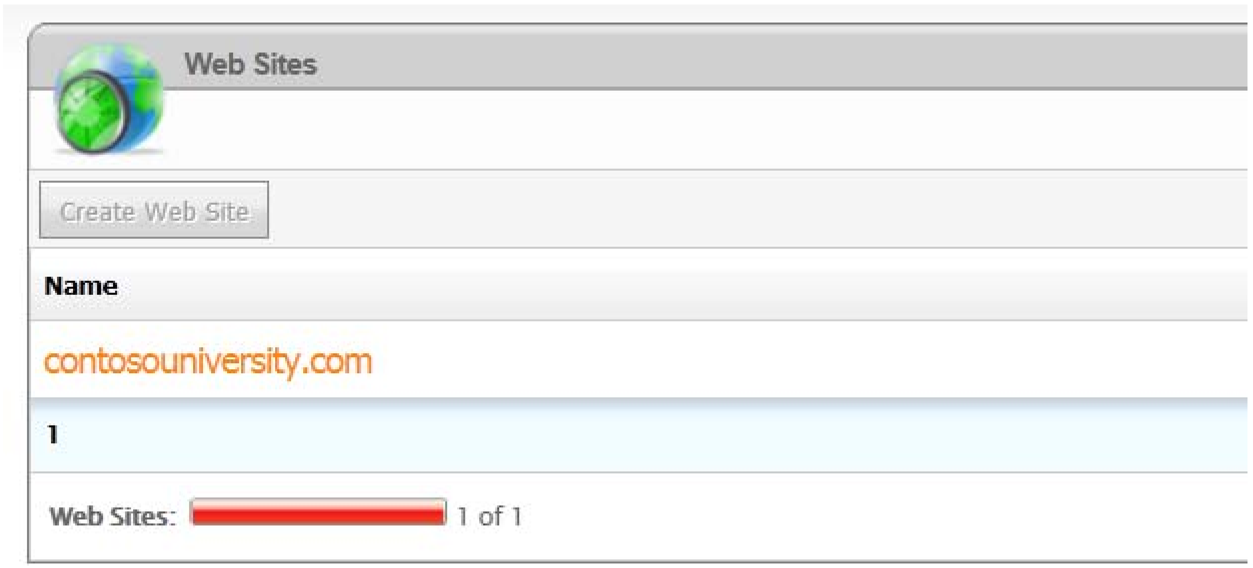
No comments added

contosou
Full Name:
E-Mail:
Role: User
Created: 9/15/2011 12:33:45 PM
Updated: 9/15/2011 12:33:45 PM

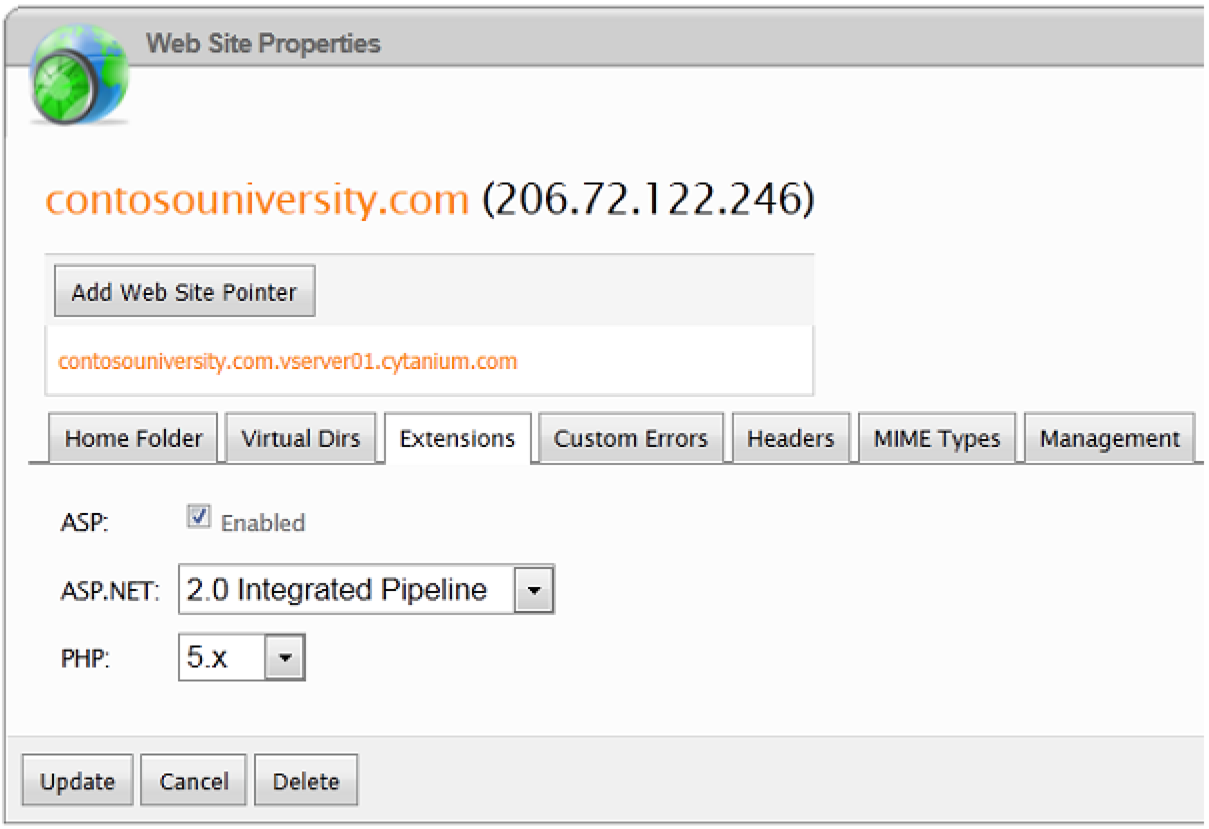
In the **Hosting Spaces** box, hold the pointer over the Web icon and select **Web Sites** from the menu.



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.

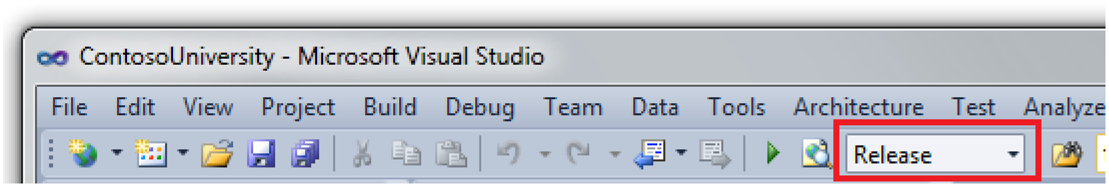


ASP: ✓ Enabled
ASP.NET: 2.0 Integrated Pipeline
PHP: 5.x

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

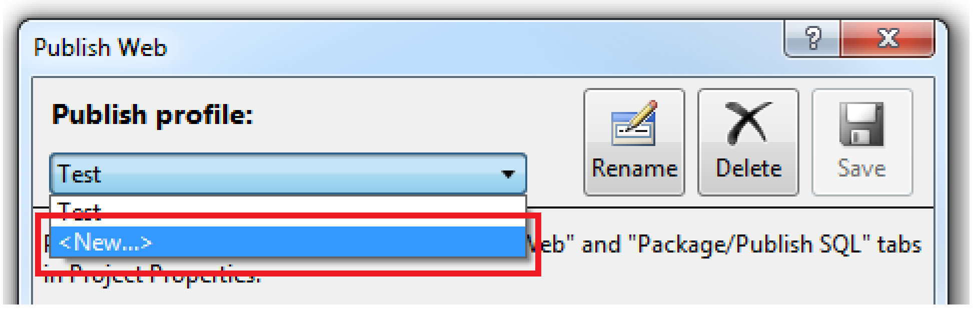
Publishing to the Hosting Provider

Change the active build configuration to Release. You can use the toolbar, as shown below, or select **Configuration Manager** from the **Build** menu.



In **Solution Explorer**, right-click the ContosoUniversity project and select **Publish**. The **Publish Web** dialog box appears with the **Test** profile selected because that is all you have created so far.

In the **Publish profile** box, select **new**.



Enter "Production" for the new profile name.

Enter the **Service URL** value that the hosting provider sent you in the welcome email.

Enter the **Site/application** value that the hosting provider sent you in the welcome email.

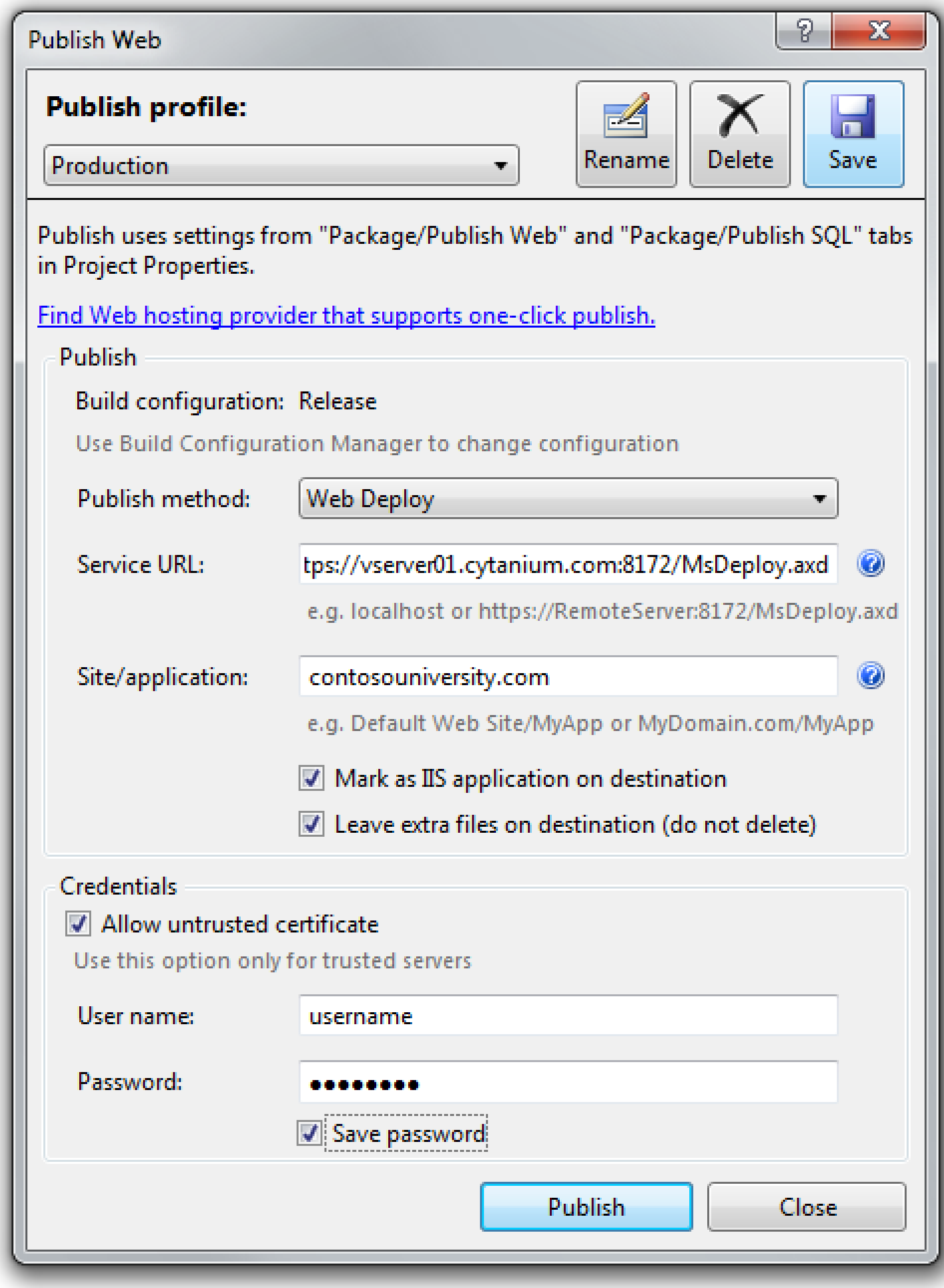
Select **Mark as IIS application on destination**.

Make sure that **Leave extra files on destination (do not delete)** is selected. If you clear this checkbox, Web Deploy will delete any files from the destination site that it does not find in the source site. For the initial deployment this will have no effect because there are not yet any files in the destination site to be deleted, but in subsequent update deployments, selecting this option could have unintended results. For example, any log files in the Elmah folder in production would be deleted because identical files would not exist in your Visual Studio project.

Select **Allow untrusted certificate**.

Enter the credentials provided to you by the hosting provider.

Select **Save password** so that you don't have to enter the password every time you publish.



Publish Web

Publish profile: Production

Publish uses settings from "Package/Publish Web" and "Package/Publish SQL" tabs in Project Properties.

Find Web hosting provider that supports one-click publish.

Publish

Build configuration: Release

Use Build Configuration Manager to change configuration

Publish method: Web Deploy

Service URL: tps://vserver01.cytanium.com:8172/MsDeploy.axd

e.g. localhost or https://RemoteServer:8172/MsDeploy.axd

Site/application: contosouniversity.com

e.g. Default Web Site/MyApp or MyDomain.com/MyApp

[✓] Mark as IIS application on destination

[✓] Leave extra files on destination (do not delete)

Credentials

[✓] Allow untrusted certificate

Use this option only for trusted servers

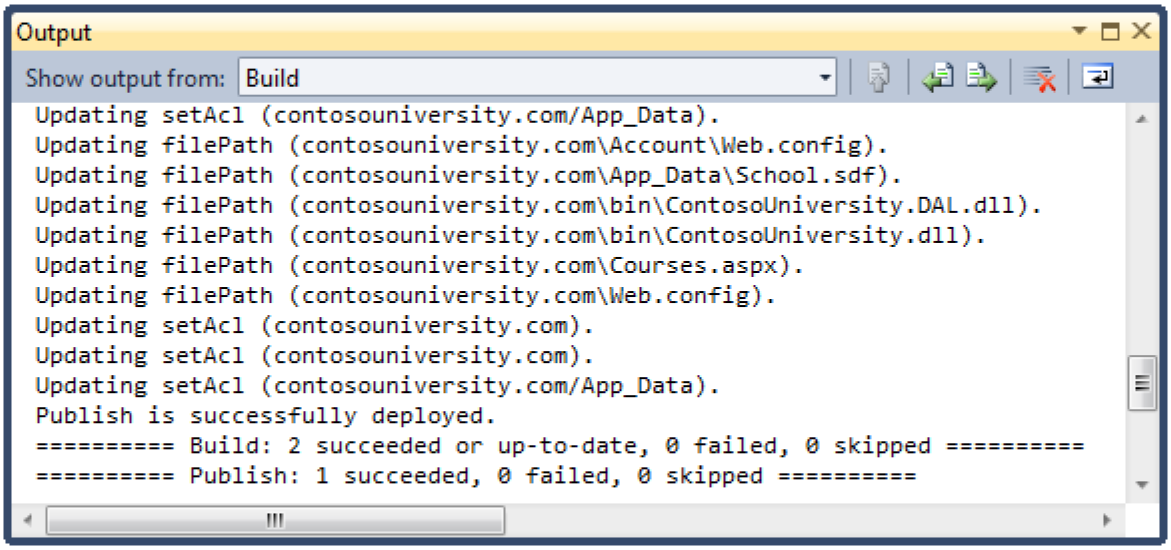
User name: username

Password: ●●●●●●●●●

[✓] Save password

Click **Publish**.

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

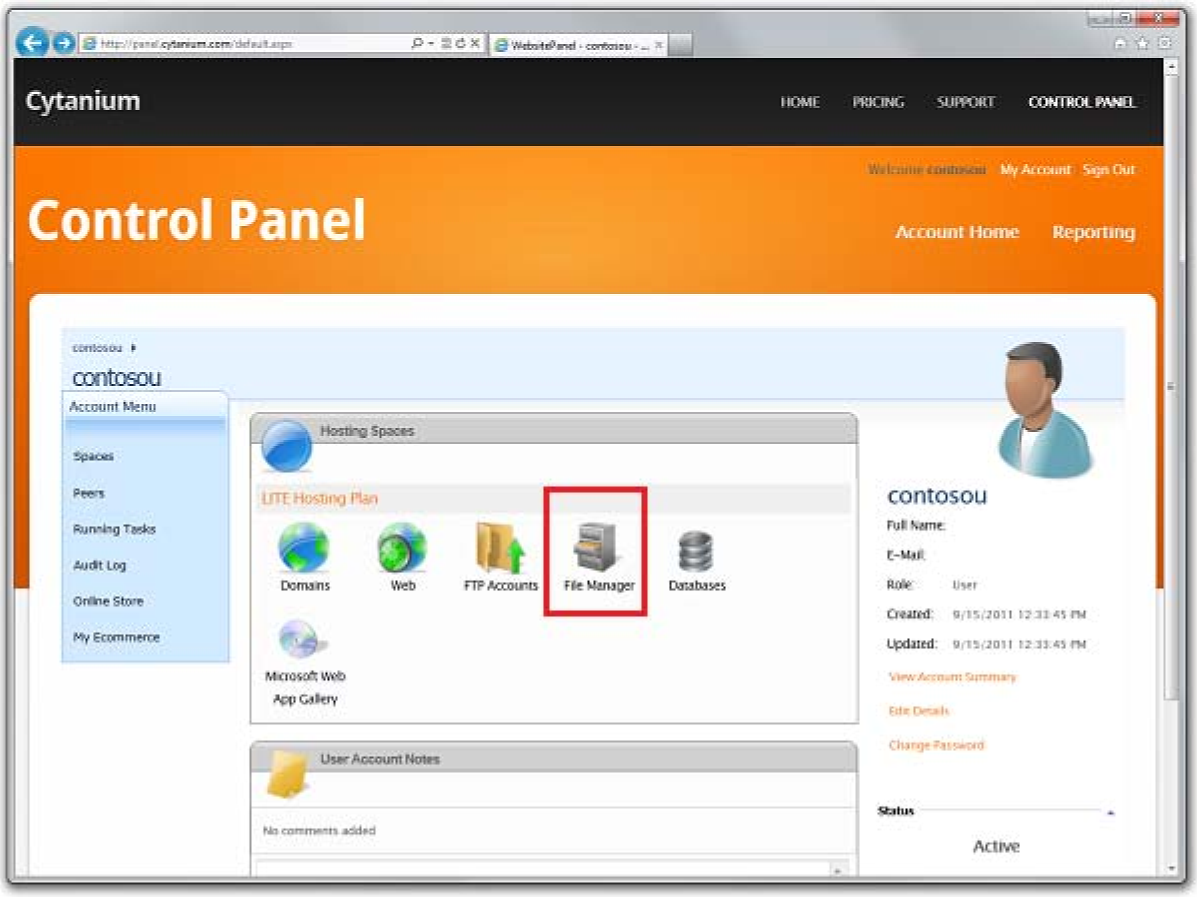


```
Updating setAcl (contosouniversity.com/App_Data).
Updating filePath (contosouniversity.com\Account\Web.config).
Updating filePath (contosouniversity.com\App_Data\School.sdf).
Updating filePath (contosouniversity.com\bin\ContosoUniversity.DAL.dll).
Updating filePath (contosouniversity.com\bin\ContosoUniversity.dll).
Updating filePath (contosouniversity.com\Courses.aspx).
Updating filePath (contosouniversity.com\Web.config).
Updating setAcl (contosouniversity.com).
Updating setAcl (contosouniversity.com).
Updating setAcl (contosouniversity.com/App_Data).
Publish is successfully deployed.
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

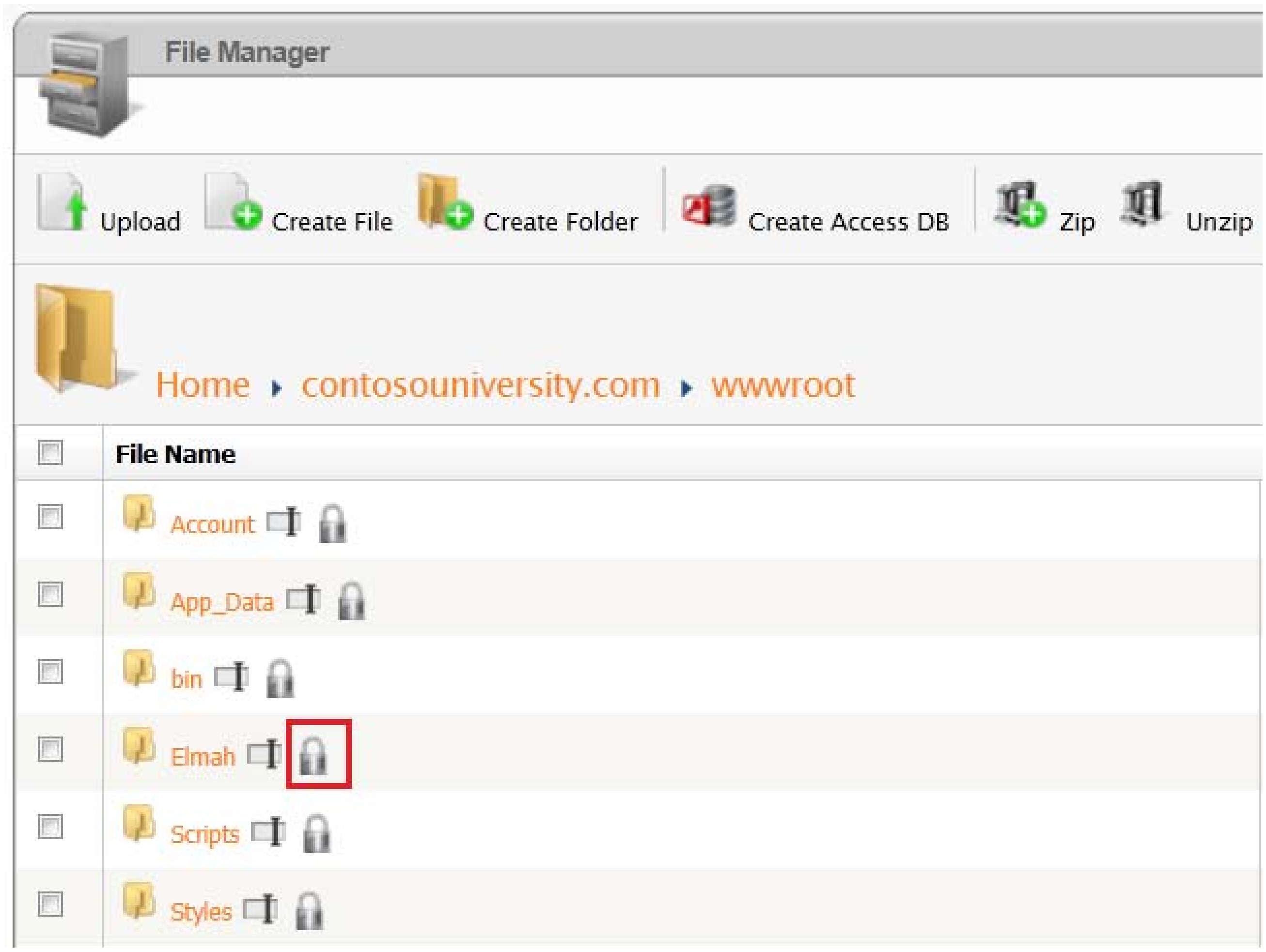
Setting Folder Permissions for Elmah

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 error log files are stored as XML 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 which case you would have to modify your application to use the specified folders.)

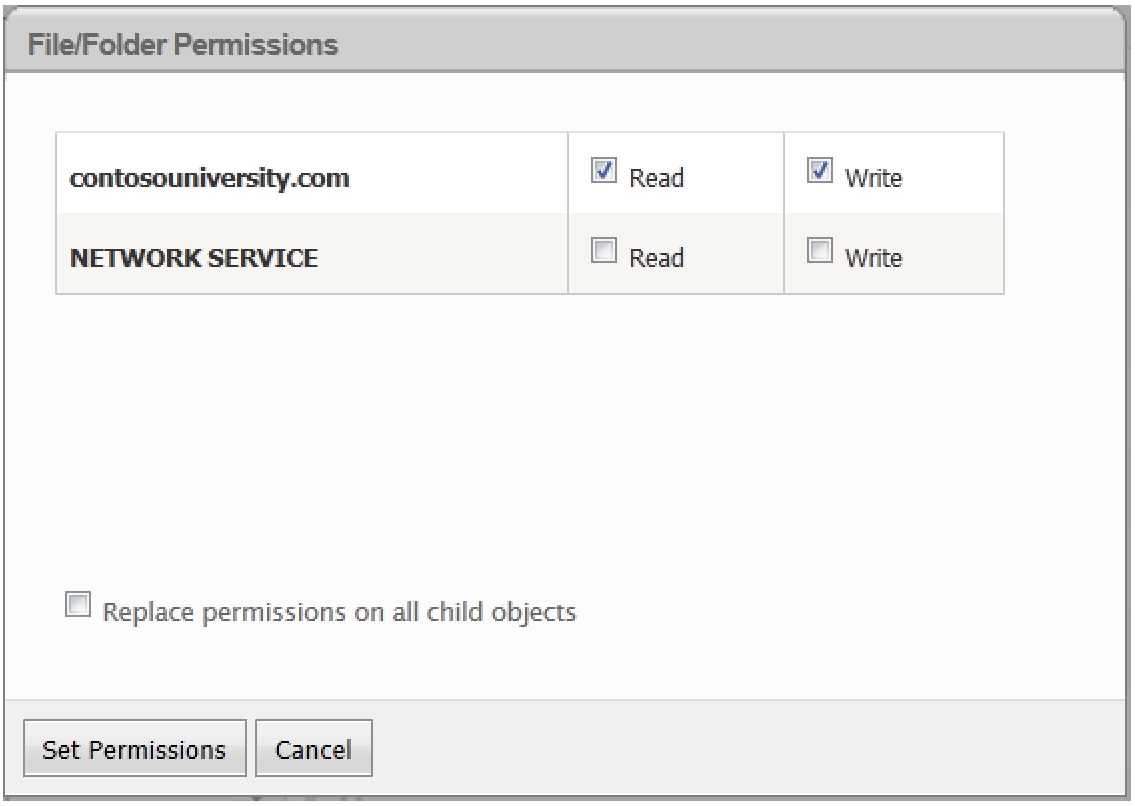
You can set folder permissions in the Cytanium control panel. Go to the control panel URL and select **File Manager**.



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**.

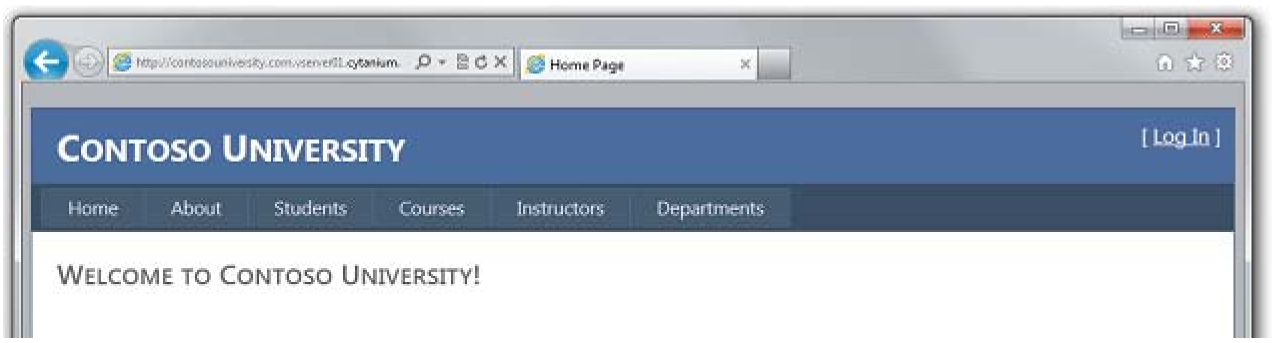

File/Folder Permissions

contosouniversity.com	✓ Read	✓ Write
NETWORK SERVICE	☐ Read	☐ Write

☐ Replace permissions on all child objects

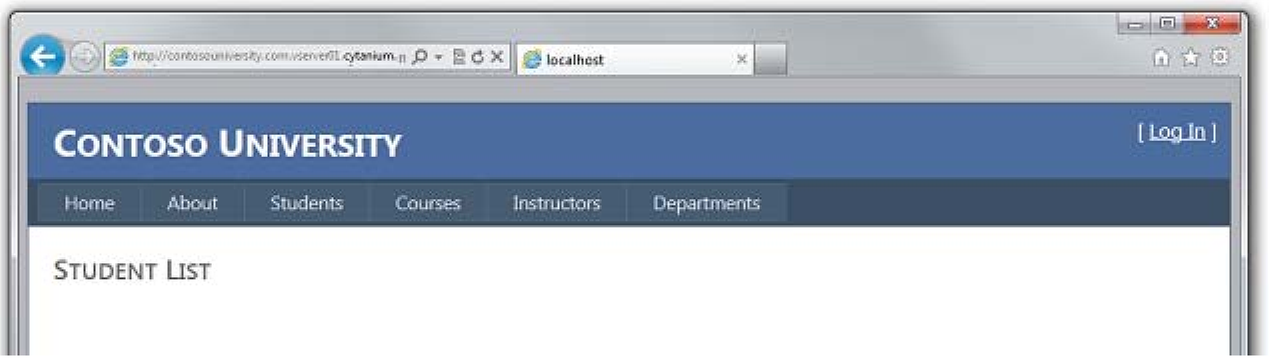
Testing in the Production Environment

Open a browser and go to the temporary URL that the provider specified when you created the account. (For this example, the URL is *http://contosouniversity.com.vserver01.cytanium.com*) 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, which indicates that the environment indicator *Web.config* transformation worked correctly.



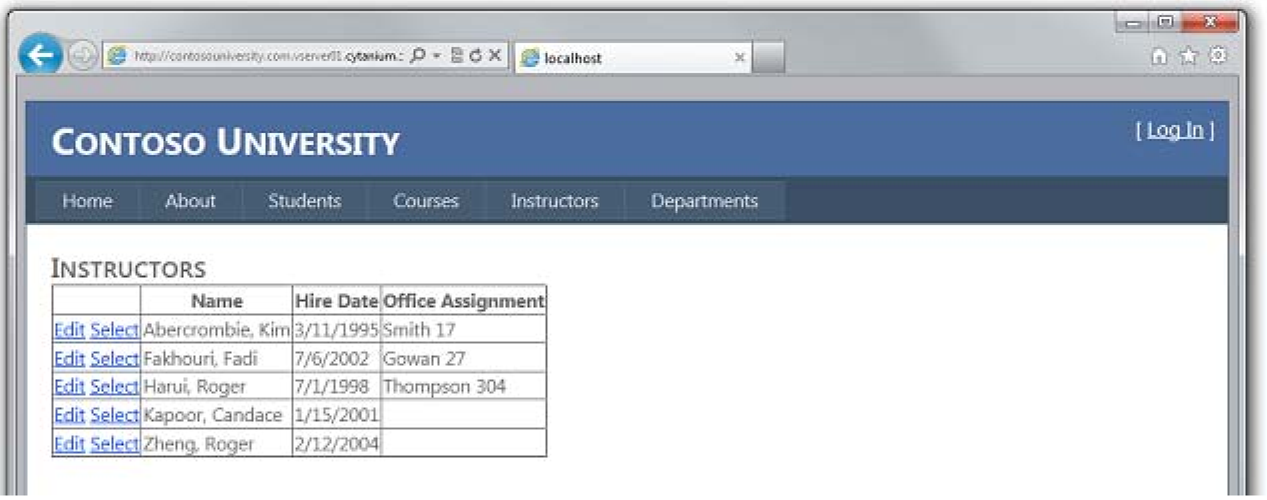
WELCOME TO CONTOSO UNIVERSITY!

Run the **Students** page to verify that the deployed database has no students:



STUDENT LIST

Run the **Instructors** page to verify that other data is still in the database:

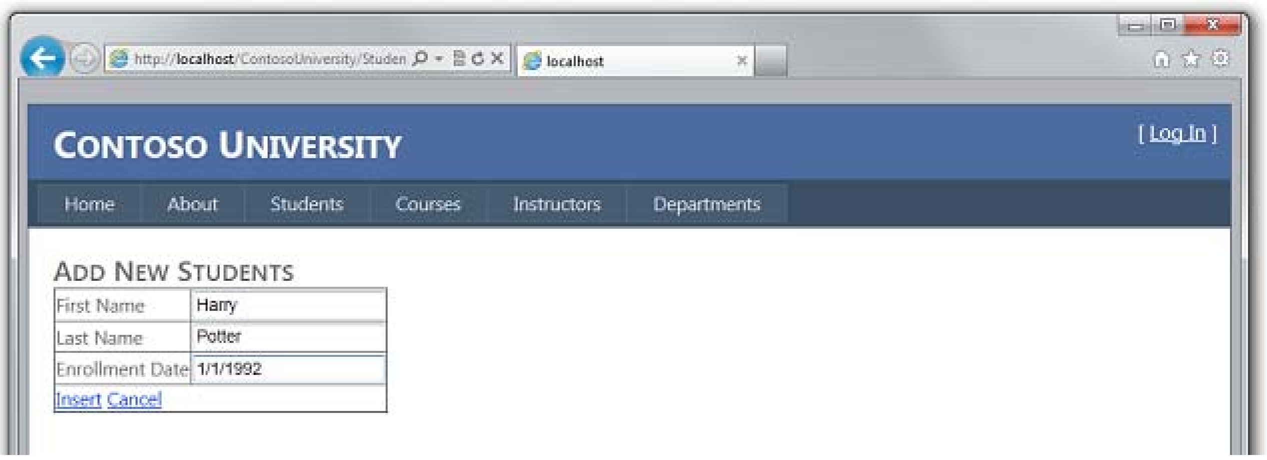


INSTRUCTORS

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	Smith 17
Edit Select	Fakhouri, Fadi	7/6/2002	Gowan 27
Edit Select	Harui, Roger	7/1/1998	Thompson 304
Edit Select	Kapoor, Candace	1/15/2001	
Edit Select	Zheng, Roger	2/12/2004	

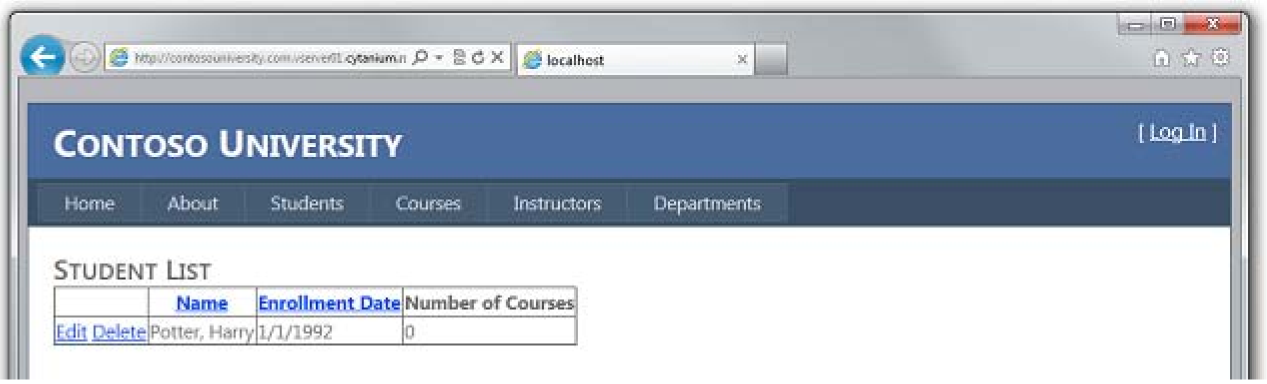
As you did in the test environment, you want to verify that database updates work in the production environment, but you typically do not want to enter test data into your production database. For this tutorial you'll use the same method you did in test. But in a real application you might want to find a different method, something you can do to validate that database updates are successful but 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 database update functionality works:



ADD NEW STUDENTS

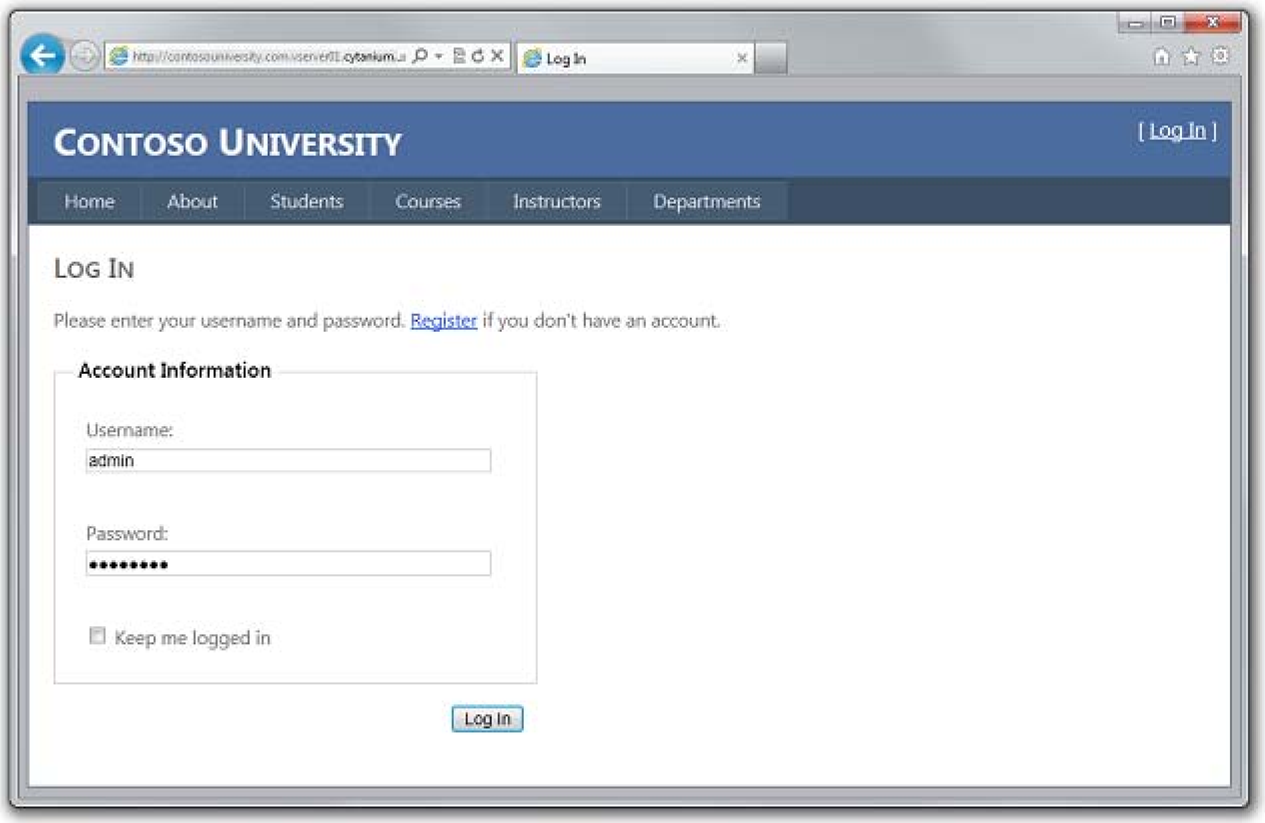
First Name	Harry
Last Name	Potter
Enrollment Date	1/1/1992
Insert Cancel	



STUDENT LIST

	Name	Enrollment Date	Number of Courses
Edit Delete	Potter, Harry	1/1/1992	0

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.



LOG IN

Please enter your username and password. Register if you don't have an account.

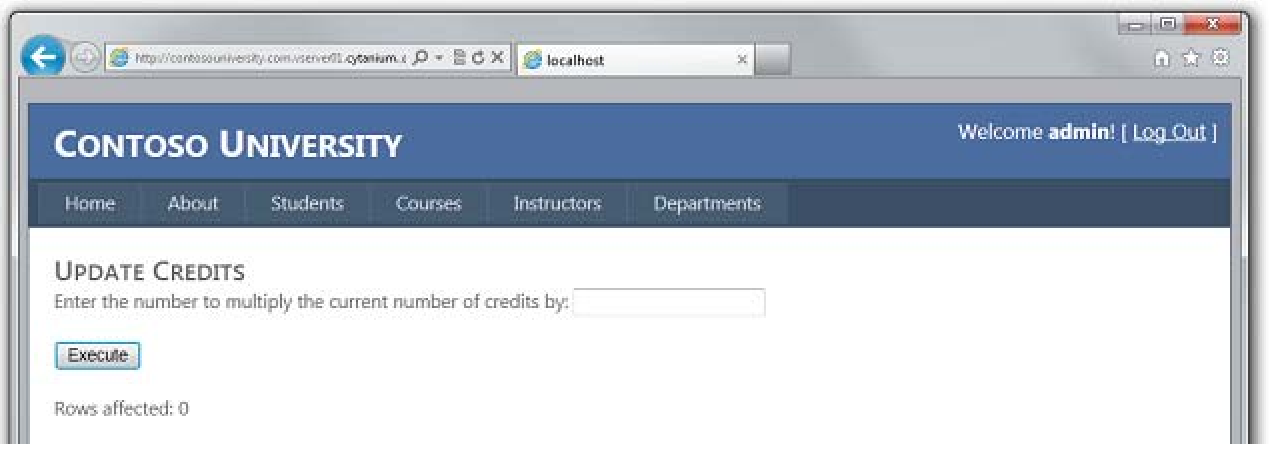
Account Information

Username: admin

Password: ••••••••

☐ Keep me logged in

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.

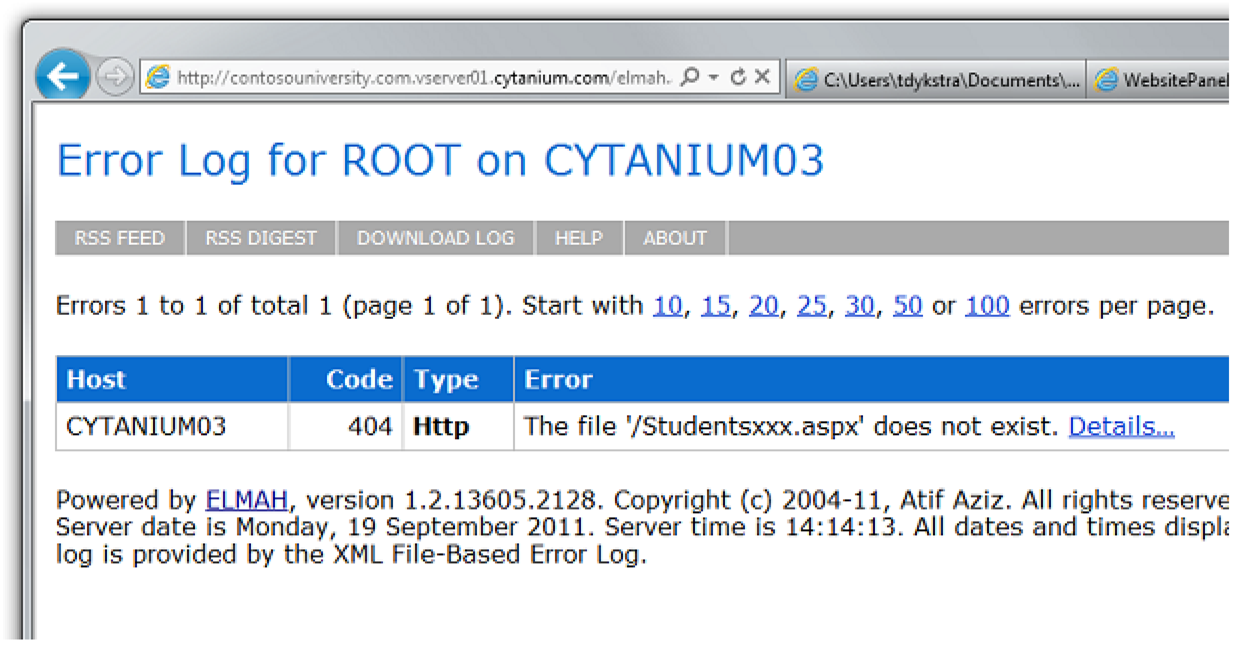


UPDATE CREDITS

Enter the number to multiply the current number of credits by:

Rows affected: 0

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 showing the error you just caused.


http://contosouniversity.com.vserver01.cytanium.com/elmah.
C:\Users\tdykstra\Documents\...
WebsitePanel

Error Log for ROOT on CYTANIUM03

RSS FEED
RSS DIGEST
DOWNLOAD LOG
HELP
ABOUT

Errors 1 to 1 of total 1 (page 1 of 1). Start with 10, 15, 20, 25, 30, 50 or 100 errors per page.

Host	Code	Type	Error
CYTANIUM03	404	Http	The file '/Studentsxxx.aspx' does not exist. Details...

Powered by ELMAH, version 1.2.13605.2128. Copyright (c) 2004-11, Atif Aziz. All rights reserve
Server date is Monday, 19 September 2011. Server time is 14:14:13. All dates and times displ
log is provided by the XML File-Based Error Log.

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 new build configuration and *Web.config* transformation file as you did in the Web.config File Transformations and Configuring Project Properties tutorials.
- Create an account at the hosting provider and set up for deployment using the new build configuration, as you did using the production build configuration in this tutorial.
- 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 do this:

- 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 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 need 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 you specify the exclusion only for the **Release** build configuration.

Creating a second hosting account is an approach to working with a test environment that is not absolutely 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 make a change to the application and deploy your change to the test and production environments.

Deploying a Code-Only Update - 8 of 12

Overview

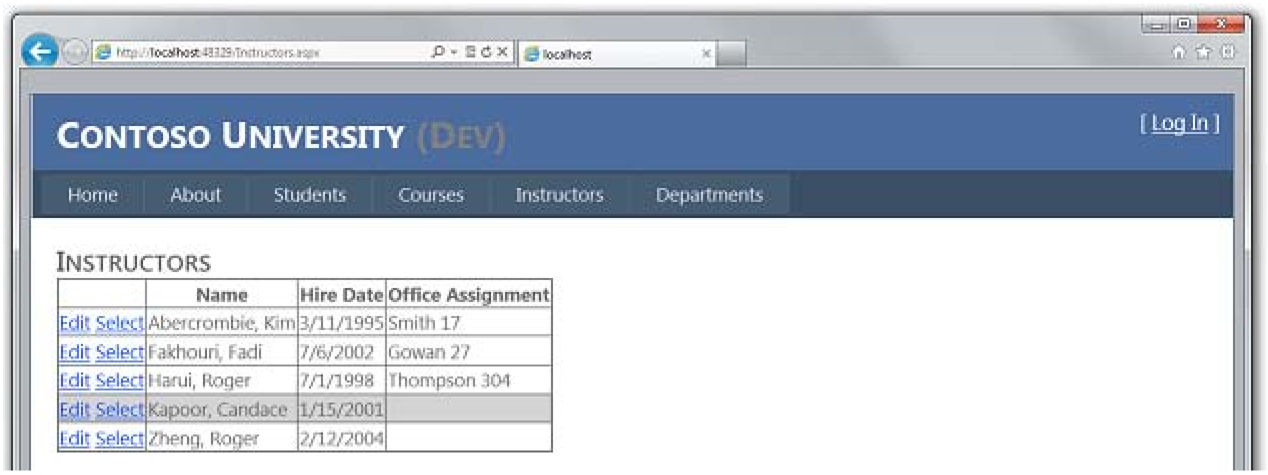
After the initial deployment, your work of maintaining and developing your web site continues, and before long you will want to deploy an update. This tutorial takes you through the process of deploying an update to your application code that does not involve a database change. (You'll see what's different 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 will notice that there are **Select** links in the grid, but they don't do anything other than make the row background turn gray.



INSTRUCTORS

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	Smith 17
Edit Select	Fakhouri, Fadi	7/6/2002	Gowan 27
Edit Select	Harui, Roger	7/1/1998	Thompson 304
Edit Select	Kapoor, Candace	1/15/2001	
Edit Select	Zheng, Roger	2/12/2004	

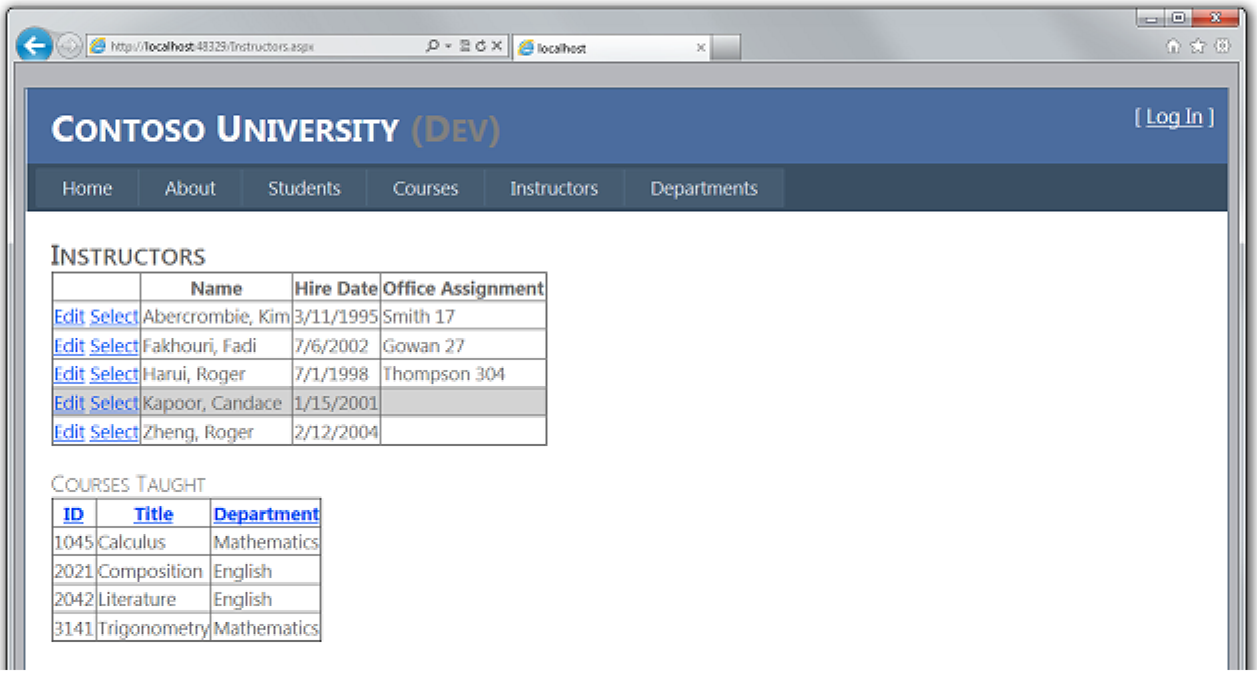
Now you'll add code that runs when the **Select** link is clicked and displays a list of courses taught by the selected instructor .

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.



Instructors

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	Smith 17
Edit Select	Fakhouri, Fadi	7/6/2002	Gowan 27
Edit Select	Harui, Roger	7/1/1998	Thompson 304
Edit Select	Kapoor, Candace	1/15/2001	
Edit Select	Zheng, Roger	2/12/2004	

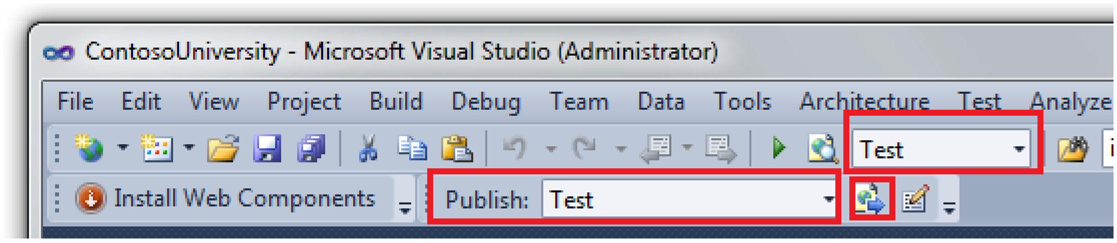
Courses Taught

ID	Title	Department
1045	Calculus	Mathematics
2021	Composition	English
2042	Literature	English
3141	Trigonometry	Mathematics

Deploying the Code Update to the Test Environment

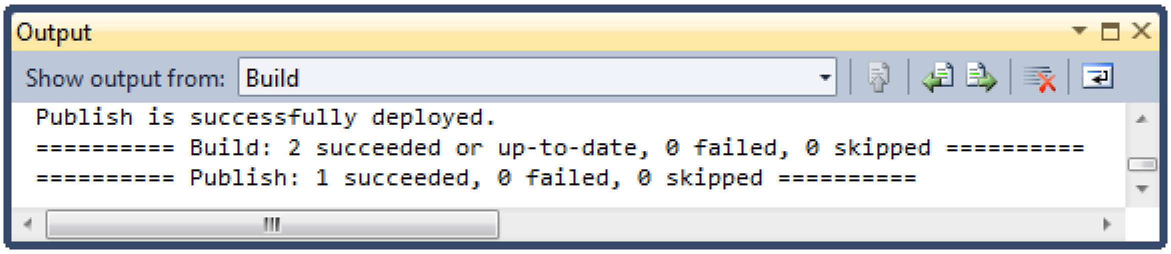
Deploying to the test environment is a simple matter of running one-click publish again.

In the **Solution Configurations** drop-down box, select the **Test** build configuration. In the **Publish** profile drop-down box select **Test**, and then click **Publish Web**.



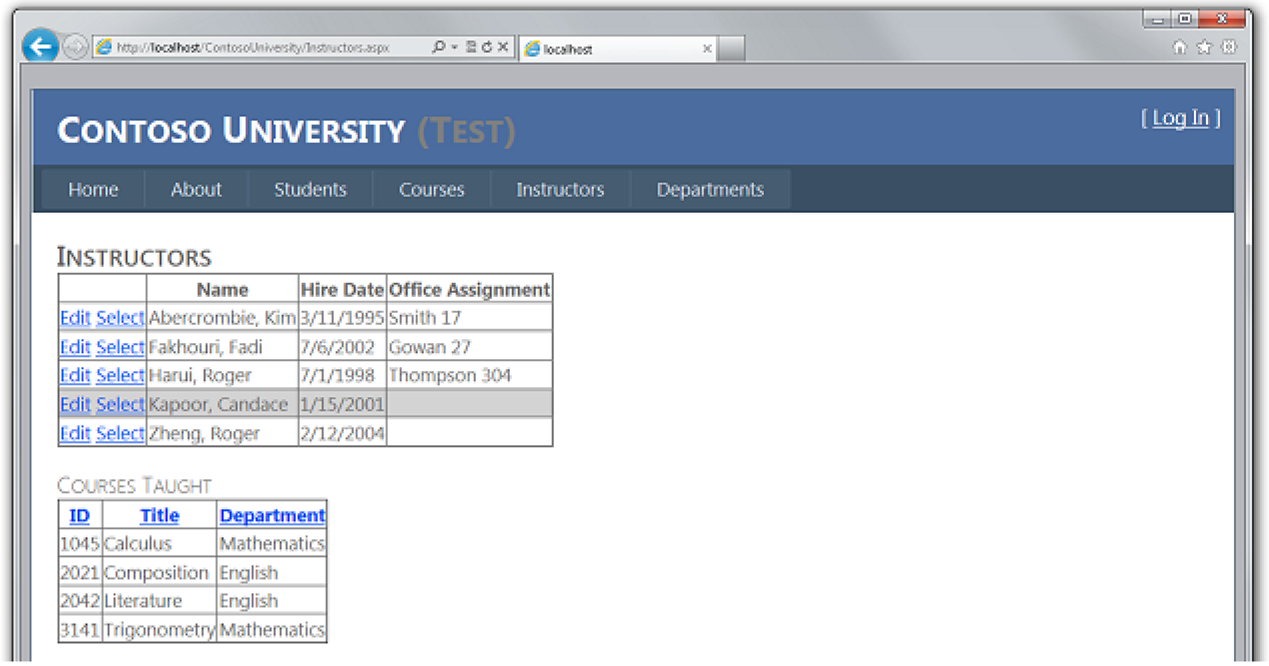
If you have customized Visual Studio and these settings aren't available in your toolbars, select the **Active solution configuration** in the **Configuration Manager** dialog box (select **Configuration Manager** from the **Build** menu), select the Test profile in the **Publish Web** dialog box (right-click the ContosoUniversity project in **Solution Explorer** and select **Publish**), and click **Publish** in the **Publish Web** dialog box.

When you click **Publish**, Visual Studio deploys the updated application and reports success in the **Output** window.



```
Publish is successfully deployed.
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

You can now open a browser and run http://localhost/ContosoUniversity/Instructors.aspx. Click a **Select** link to verify that the update was successfully deployed.



CONTOSO UNIVERSITY (TEST)

INSTRUCTORS

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	Smith 17
Edit Select	Fakhouri, Fadi	7/6/2002	Gowan 27
Edit Select	Harui, Roger	7/1/1998	Thompson 304
Edit Select	Kapoor, Candace	1/15/2001	
Edit Select	Zheng, Roger	2/12/2004	

COURSES TAUGHT

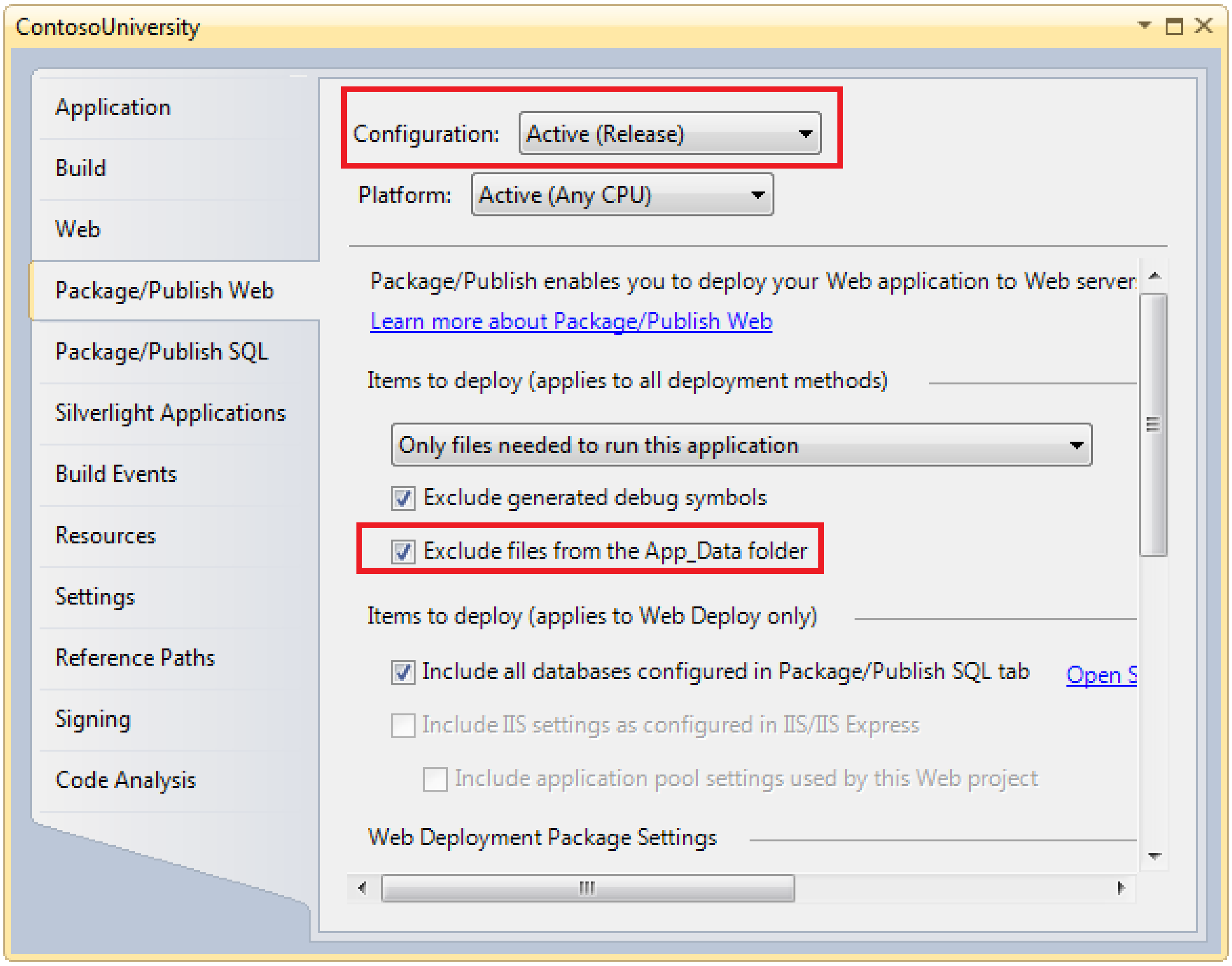
ID	Title	Department
1045	Calculus	Mathematics
2021	Composition	English
2042	Literature	English
3141	Trigonometry	Mathematics

You would normally also do regression testing at this point (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. If you don't change deployment settings, when you deploy the update, you'll also redeploy the databases in their initial state, which would wipe out all of the live data. Since your SQL Server Compact databases are files in the *App_Data* folder, you need to prevent this by changing deployment settings so that files in the *App_Data* folder aren't deployed.

Open the **Project Properties** window and click 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**, and then save and close the page.


ContosoUniversity

Configuration: Active (Release)

Platform: Active (Any CPU)

Package/Publish enables you to deploy your Web application to Web server

Learn more about Package/Publish Web

Items to deploy (applies to all deployment methods)

Only files needed to run this application

☑ Exclude generated debug symbols

☑ Exclude files from the App_Data folder

Items to deploy (applies to Web Deploy only)

☑ Include all databases configured in Package/Publish SQL tab Open S

☐ Include IIS settings as configured in IIS/IIS Express

☐ Include application pool settings used by this Web project

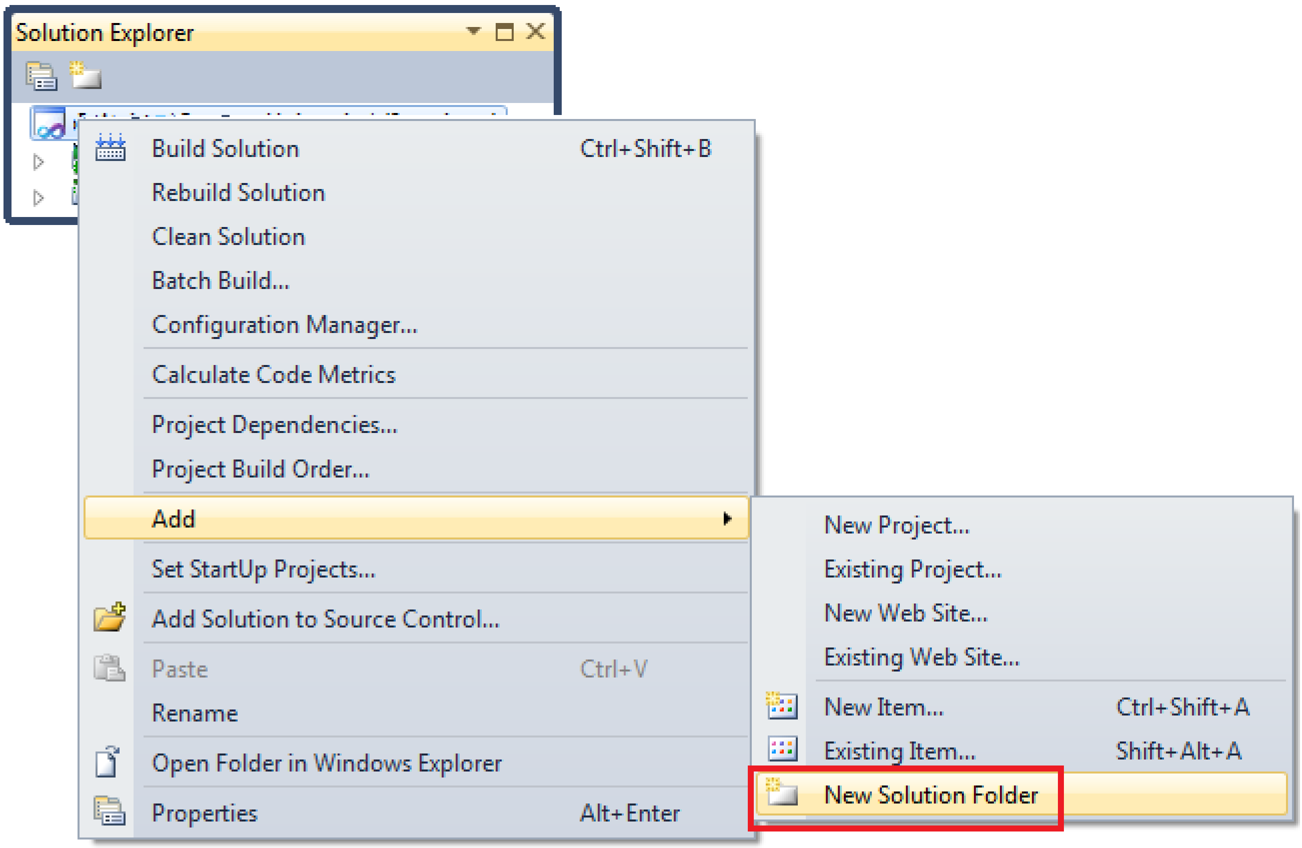
Web Deployment Package Settings

Make the same change for deployment to the Test environment: change **Configuration** to **Test** and then select **Exclude files from the App_Data folder**.

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'll 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 the project) and select **New Solution Folder**.

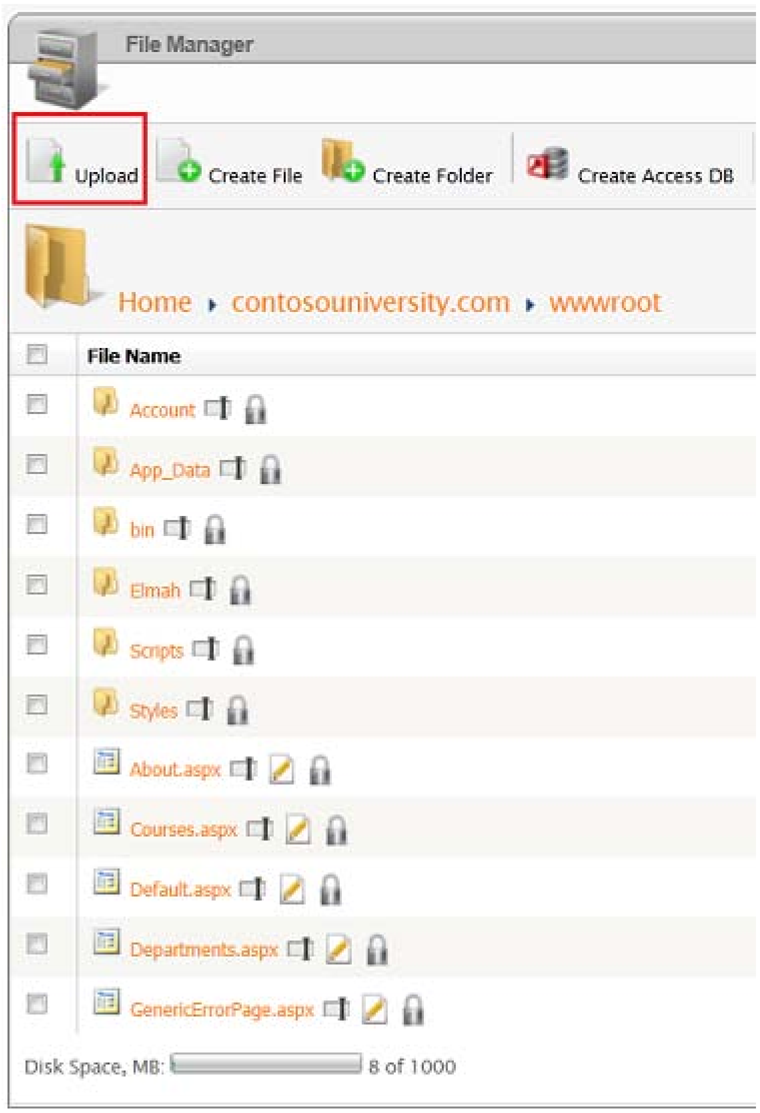


Name the folder *SolutionFiles*. In the new folder create a new HTML page named *app_offline.htm*. Replace the existing contents with the following markup:

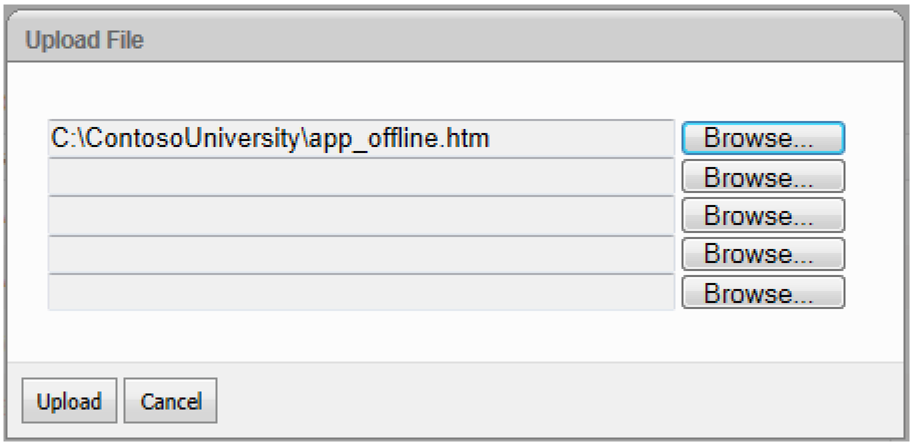
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Contoso University - Under Construction</title>
</head>
<body>
<h1>Contoso University</h1>
<h2>Under Construction</h2>
<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 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**.

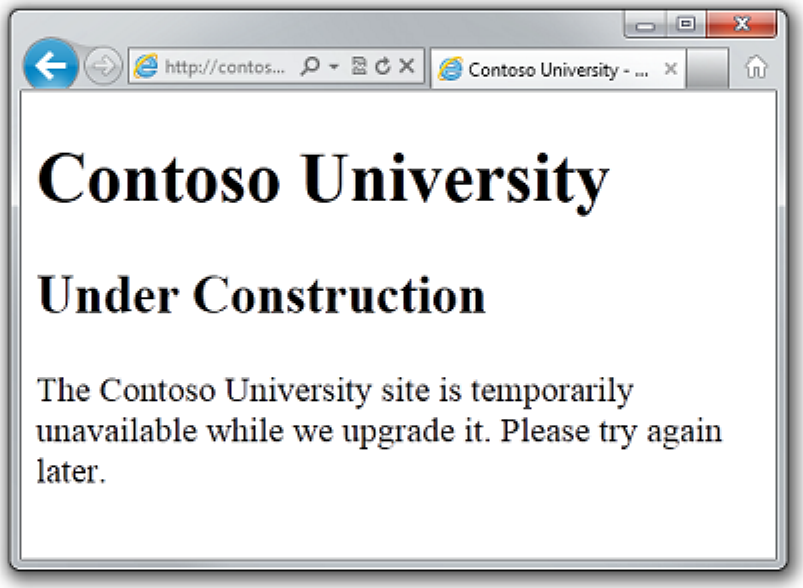


In the **Upload File** dialog box, select the *app_offline.htm* file and then click **Upload**.



C:\ContosoUniversity\app_offline.htm

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



Contoso University

Under Construction

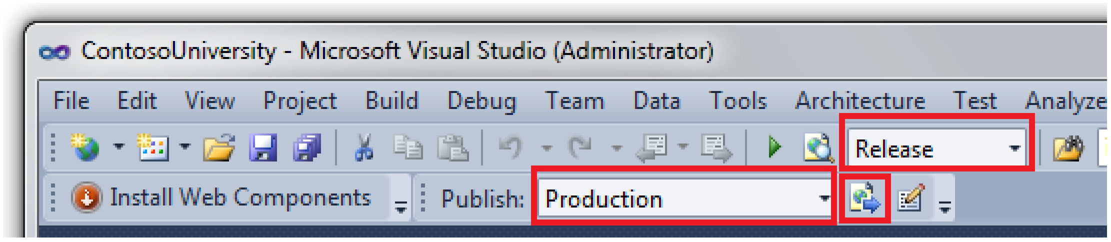
The Contoso University site is temporarily unavailable while we upgrade it. Please try again later.

You are now ready to deploy to production.

(These tutorials indicate that you should select the **Leave Extra files on destination (do not delete)** option in the **Publish** profile. If you decide to clear that checkbox so that an update deployment will delete unneeded files in production, make sure that you put a copy of *app_offline.htm* in your project folder before you deploy. Otherwise Web Deploy will delete *app_offline.htm* during deployment, possibly before you are ready for it to be removed.)

Deploying the Code Update to the Test Environment

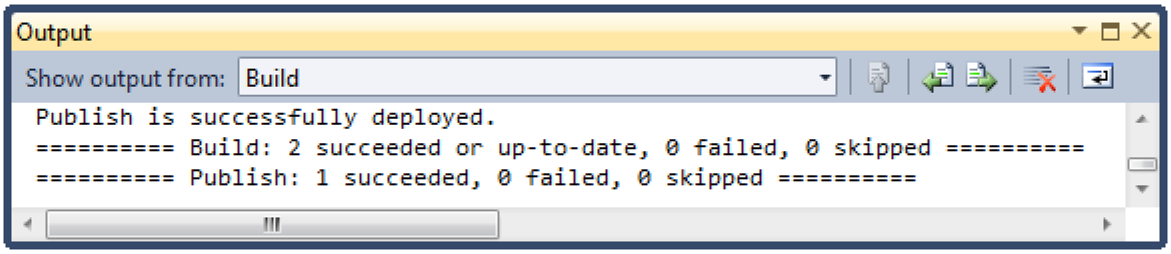
In the **Solution Configurations** drop-down box, select **Release** build configuration and in the **Publish** profile drop-down box select **Production**. Then click **Publish Web**.



Release

Publish: Production

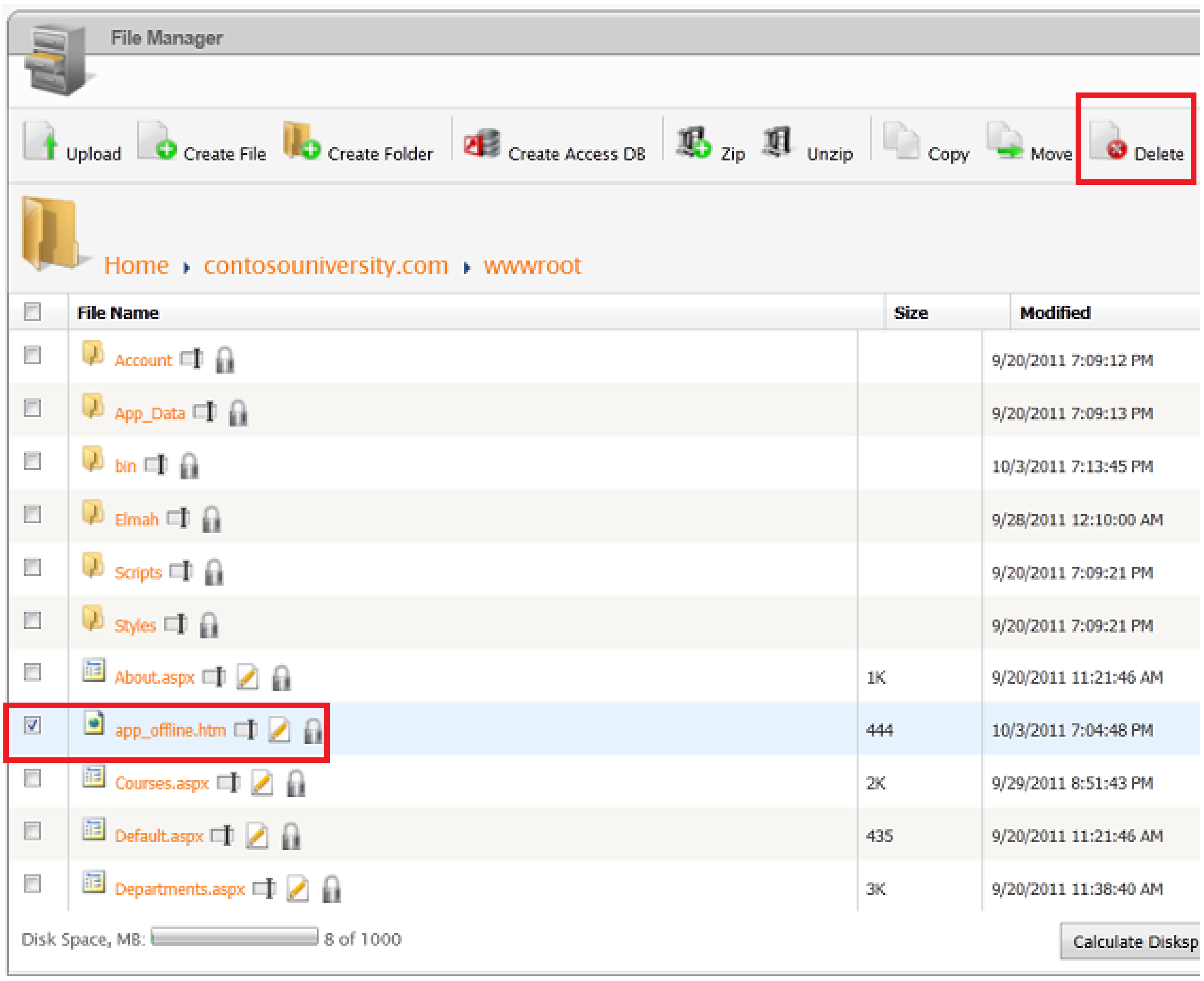
Visual Studio deploys the updated application and reports success in the **Output** window.



```
Publish is successfully deployed.
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

Before you can test to verify successful deployment, you must remove the *app_offline.htm* file.

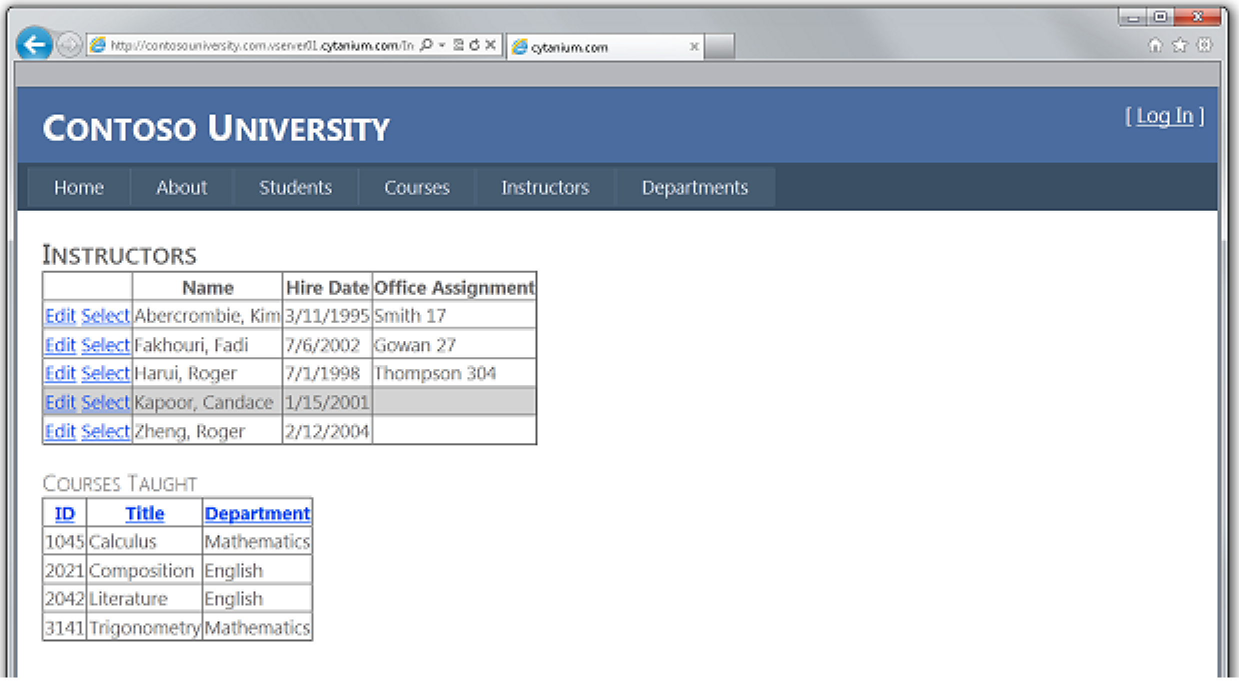
Return to the **File Manager** application in the control panel. Select **contosouniversity.com** and **wwwroot**, select **app_offline.htm**, and then click **Delete**.



	File Name	Size	Modified
☐	Account		9/20/2011 7:09:12 PM
☐	App_Data		9/20/2011 7:09:13 PM
☐	bin		10/3/2011 7:13:45 PM
☐	Elmah		9/28/2011 12:10:00 AM
☐	Scripts		9/20/2011 7:09:21 PM
☐	Styles		9/20/2011 7:09:21 PM
☐	About.aspx	1K	9/20/2011 11:21:46 AM
✓	app_offline.htm	444	10/3/2011 7:04:48 PM
☐	Courses.aspx	2K	9/29/2011 8:51:43 PM
☐	Default.aspx	435	9/20/2011 11:21:46 AM
☐	Departments.aspx	3K	9/20/2011 11:38:40 AM

Disk Space, MB: 8 of 1000

Now open a browser and browse to *Instructors.aspx* in the public site. Click a **Select** link to verify that the update was successfully deployed.



CONTOSO UNIVERSITY

INSTRUCTORS

	Name	Hire Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	Smith 17
Edit Select	Fakhouri, Fadi	7/6/2002	Gowan 27
Edit Select	Harui, Roger	7/1/1998	Thompson 304
Edit Select	Kapoor, Candace	1/15/2001	
Edit Select	Zheng, Roger	2/12/2004	

COURSES TAUGHT

ID	Title	Department
1045	Calculus	Mathematics
2021	Composition	English
2042	Literature	English
3141	Trigonometry	Mathematics

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.

Overview

Sooner or later an application update will involve changes to a database (either schema or data or both) as well as to code. In this tutorial you'll 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'll add a birth date column to the `Person` base class for the `Student` and `Instructor` entities. You'll also update the pages that display student and instructor data so that they display 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)]
[Required(ErrorMessage = "Birth date is required.")]
[Display(Name = "Birth Date")]
public DateTime? BirthDate { get; set; }
```

Next you need to update the initializer class so that it provides a value for the new column. Open *SchoolInitializer.cs* and find the code block that begins `var students = new List<Student>`. Replace that block with the following code, which includes birth date information for the students:

```
var students = new List<Student>
{
    new Student { FirstMidName = "Carson",   LastName = "Alexander",
EnrollmentDate = DateTime.Parse("2005-09-01"), BirthDate =
DateTime.Parse("1990-01-01") },
    new Student { FirstMidName = "Meredith", LastName = "Alonso",
EnrollmentDate = DateTime.Parse("2002-09-01"), BirthDate =
DateTime.Parse("1989-01-15") },
    new Student { FirstMidName = "Arturo",   LastName = "Anand",
EnrollmentDate = DateTime.Parse("2003-09-01"), BirthDate =
DateTime.Parse("1988-02-01") },
    new Student { FirstMidName = "Gytis",    LastName = "Barzdukas",
EnrollmentDate = DateTime.Parse("2002-09-01"), BirthDate =
DateTime.Parse("1987-03-15") },
    new Student { FirstMidName = "Yan",      LastName = "Li",
EnrollmentDate = DateTime.Parse("2002-09-01"), BirthDate =
DateTime.Parse("1985-11-11") },
    new Student { FirstMidName = "Peggy",    LastName = "Justice",
EnrollmentDate = DateTime.Parse("2001-09-01"), BirthDate =
DateTime.Parse("1970-11-21") },
    new Student { FirstMidName = "Laura",    LastName = "Norman",
EnrollmentDate = DateTime.Parse("2003-09-01"), BirthDate =
DateTime.Parse("1992-10-11") },
    new Student { FirstMidName = "Nino",     LastName = "Olivetto",
EnrollmentDate = DateTime.Parse("2005-09-01"), BirthDate =
DateTime.Parse("1986-06-06") }
};
```

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") },
    new Instructor { FirstMidName = "Fadi",    LastName = "Fakhouri",
HireDate = DateTime.Parse("2002-07-06"), BirthDate = DateTime.Parse("1960-03-
15") },
    new Instructor { FirstMidName = "Roger",   LastName = "Harui",
HireDate = DateTime.Parse("1998-07-01"), BirthDate = DateTime.Parse("1970-01-
11") },
    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>
```

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

In *Students.aspx*, add a new dynamic field for birth date immediately before the one for enrollment date:

```
<asp:DynamicField DataField="BirthDate" HeaderText="Birth Date"
SortExpression="BirthDate" />
```

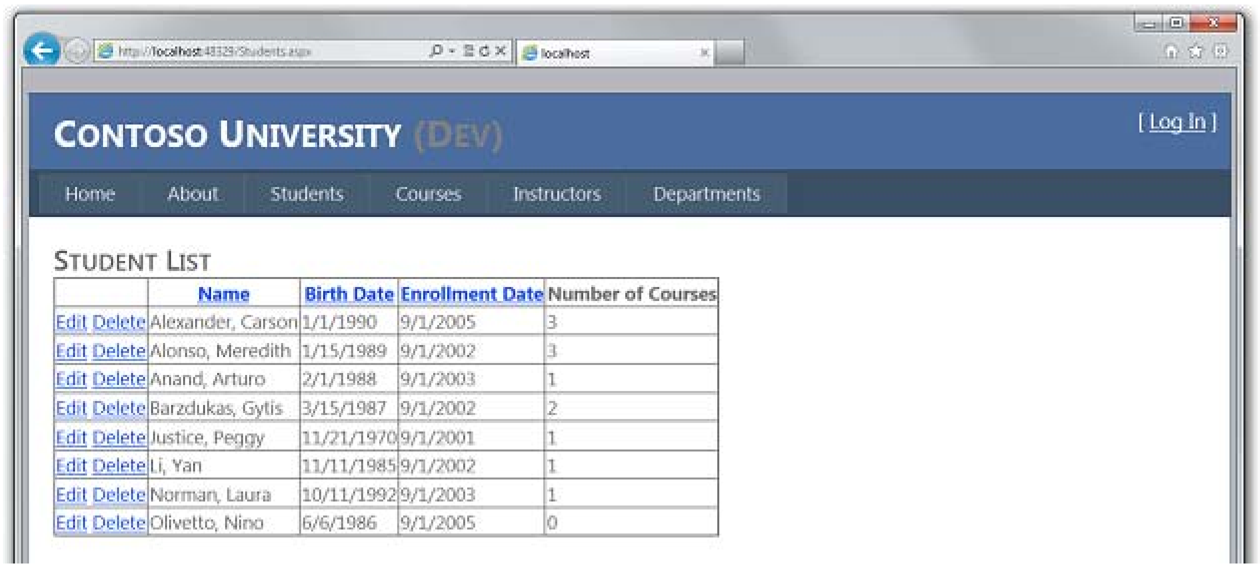
In *StudentsAdd.aspx*, add a new bound field for birth date immediately before the one for enrollment date:

```
<asp:BoundField DataField="BirthDate" HeaderText="Birth Date"
    SortExpression="BirthDate" />
```

In *BLL\SchoolBL.cs*, the `switch` statement in the `GetStudents` method supports the column sorting functionality for the **Students** page. Add two new `case` statements immediately before the `default` statement to support sorting the new birth date column:

```
case "BirthDate":
    students = students.OrderBy(s => s.BirthDate);
    break;
case "BirthDate DESC":
    students = students.OrderByDescending(s => s.BirthDate);
    break;
```

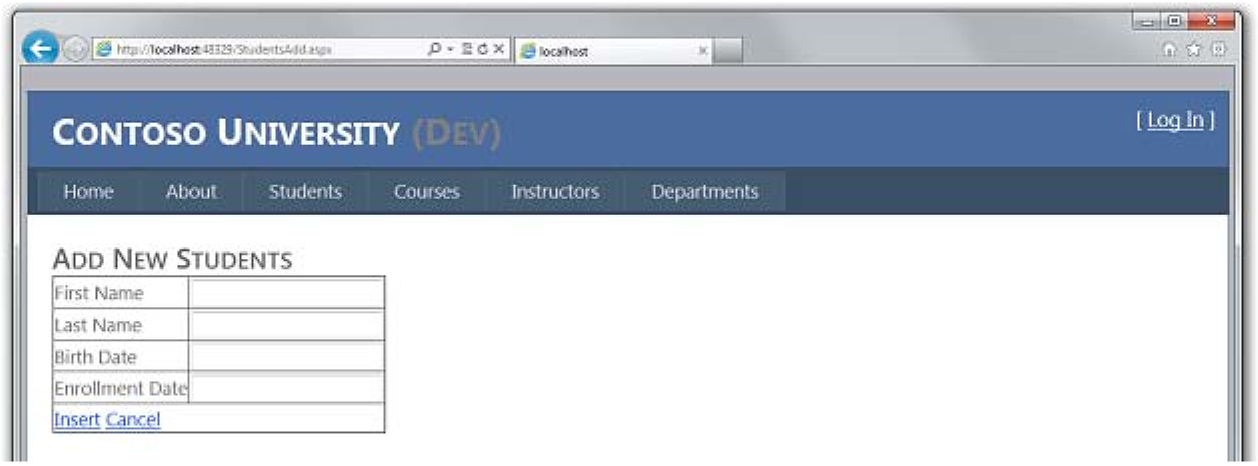
Run the application and select the **Students** page. You might notice that the page takes a little longer than usual to load. This is because the Entity Framework recognizes that the model has changed, so it drops and re-creates the database, then seeds it with the test data specified in your initializer class. When the page loads, you see that it has the new birth date field.



STUDENT LIST

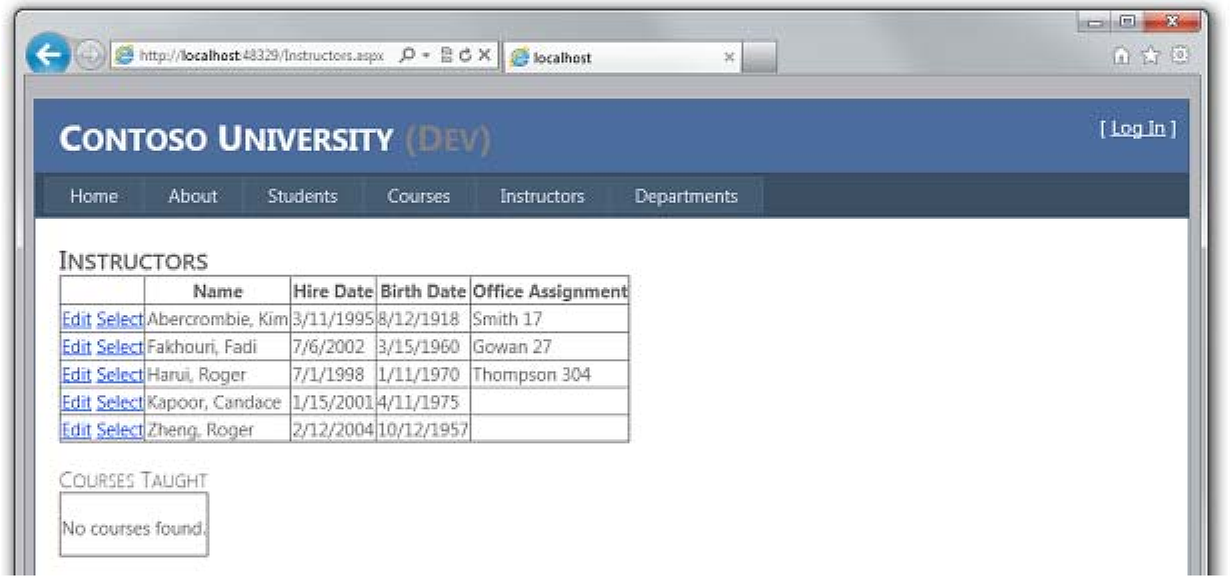
	Name	Birth Date	Enrollment Date	Number of Courses
Edit Delete	Alexander, Carson	1/1/1990	9/1/2005	3
Edit Delete	Alonso, Meredith	1/15/1989	9/1/2002	3
Edit Delete	Anand, Arturo	2/1/1988	9/1/2003	1
Edit Delete	Barzdukas, Gytis	3/15/1987	9/1/2002	2
Edit Delete	Justice, Peggy	11/21/1970	9/1/2001	1
Edit Delete	Li, Yan	11/11/1985	9/1/2002	1
Edit Delete	Norman, Laura	10/11/1992	9/1/2003	1
Edit Delete	Olivetto, Nino	6/6/1986	9/1/2005	0

Select the **Add Students** and **Instructors** pages to verify that you see the new field.



ADD NEW STUDENTS

First Name	
Last Name	
Birth Date	
Enrollment Date	
Insert Cancel	



INSTRUCTORS

	Name	Hire Date	Birth Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	8/12/1918	Smith 17
Edit Select	Fakhouri, Fadi	7/6/2002	3/15/1960	Gowan 27
Edit Select	Harui, Roger	7/1/1998	1/11/1970	Thompson 304
Edit Select	Kapoor, Candace	1/15/2001	4/11/1975	
Edit Select	Zheng, Roger	2/12/2004	10/12/1957	

COURSES TAUGHT

No courses found.

Deploying the Database Update to the Test Environment

In your production site, your users have been entering data, and you want to preserve that data when you deploy this change to production. You don't actually need to preserve the data in the test environment. However, you want to use the same method for deploying to the test environment that you'll use later to deploy to production, to make sure that the method works correctly. Therefore, you'll deploy to the test environment using a method that closely follows what you'll do for production database. For production, here's an outline of the procedure you'll follow (omitting the *app_offline.htm* steps):

- Download *School-Prod.sdf* from the production site to the *App_Data* folder in your Visual Studio project.
- Manually apply the schema change to it, and manually update the data.
- Upload *School-Prod.sdf* back to the production site.
- Deploy the project.

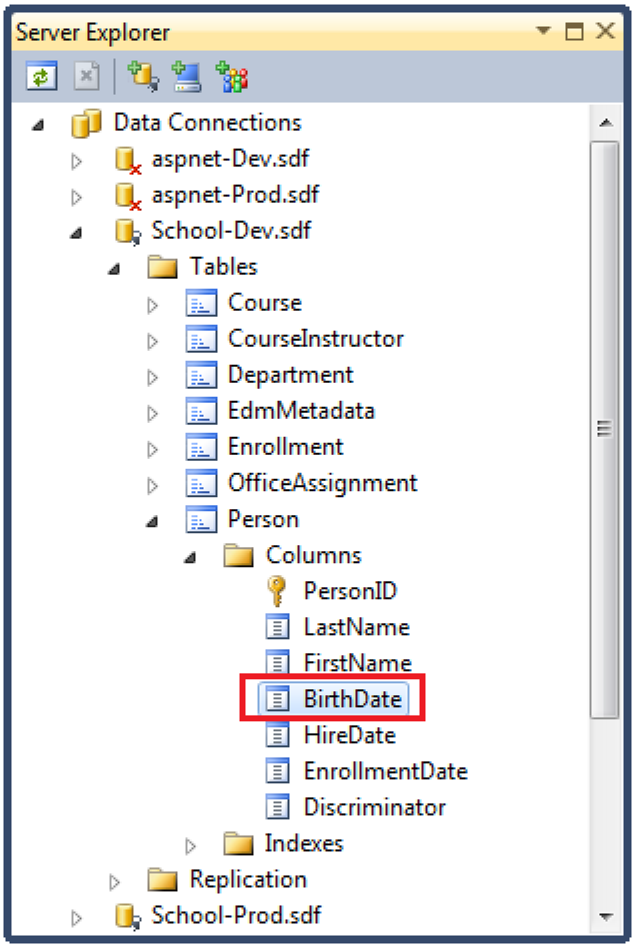
(As an alternative approach, instead of copying the file back to the wwwroot folder manually, you could let Visual Studio do it as part of deployment. But then you would have to change deployment settings in project properties in order to specify that the School database should be copied but the membership databse should be excluded. With this method you do not need to change any deployment settings, and you can continue to deploy just by using one-click publish with existing settings for all site updates.)

Applying the Schema Change

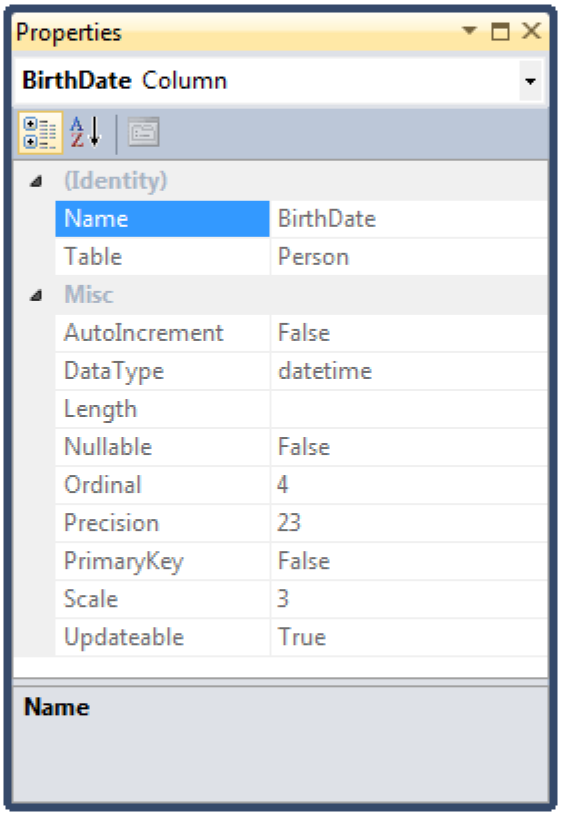
For deployment to the Test environment, you don't need to worry about getting an up-to-the-minute copy of the database, so you can update the copy of *School-Prod.sdf* that you already have in the project without copying down the current copy of the database.

To begin, look at the changes that the Entity Framework made to your School database, so that you can manually make the same change to your test and production databases. In **Solution Explorer**, double-click *App_Data\School.sdf* to open a connection to the School database in **Server Explorer**.

In **Server Explorer**, expand **School-Dev.sdf**, expand **Tables**, expand **Person**, and expand **Columns**. You see the new **BirthDate** column that Code First created after you added a `BirthDate` property to the Person entity:



In the **Properties** window, you can see that the data type of this column is `datetime`. In this case, the name and type is all that you will need in order to replicate the change manually.



Properties

BirthDate Column

(Identity)	
Name	BirthDate
Table	Person
Misc	
AutoIncrement	False
DataType	datetime
Length	
Nullable	False
Ordinal	4
Precision	23
PrimaryKey	False
Scale	3
Updateable	True

Name

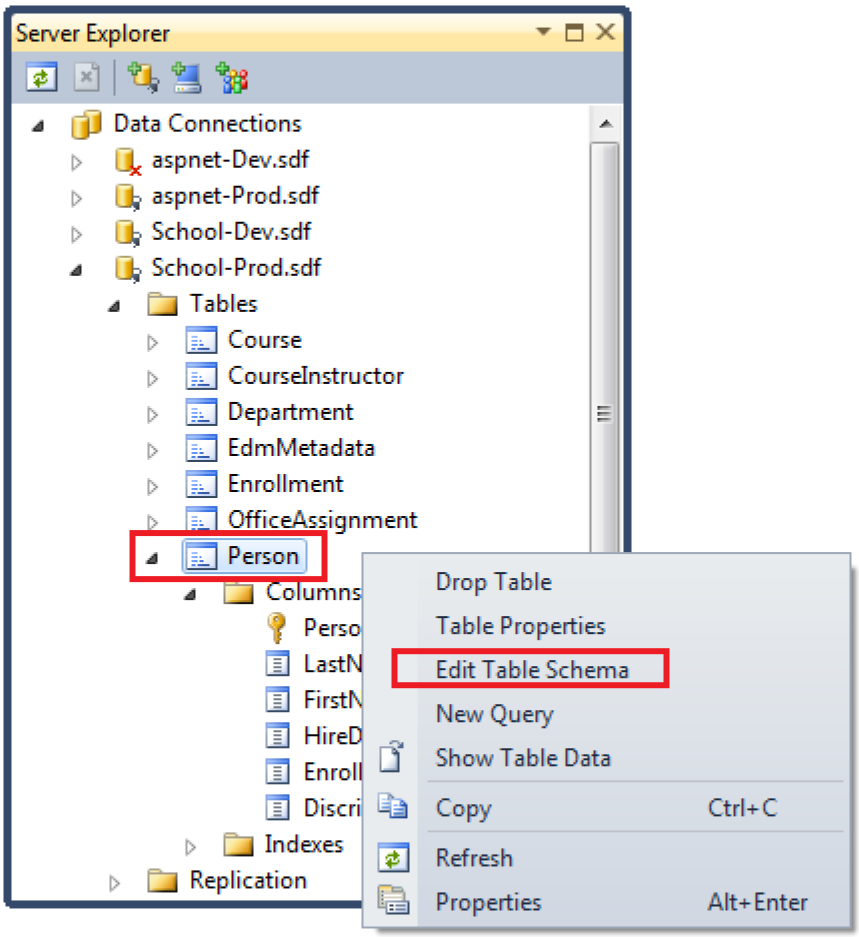
In **Server Explorer**, right-click **School-Dev.sdf** and select **Close Connection**.

(This is just one example of a simple database change. In a real application, database changes are often more complex, of course. This tutorial isn't focused on how to make and track database changes, but on how database changes affect deployment procedures. The general approach is this: figure out what database changes you expect from changes to the data model, look at what Code First has done to the database, and then replicate those changes manually.)

The next step is to manually make the same change in the *School-Prod.sdf* file.

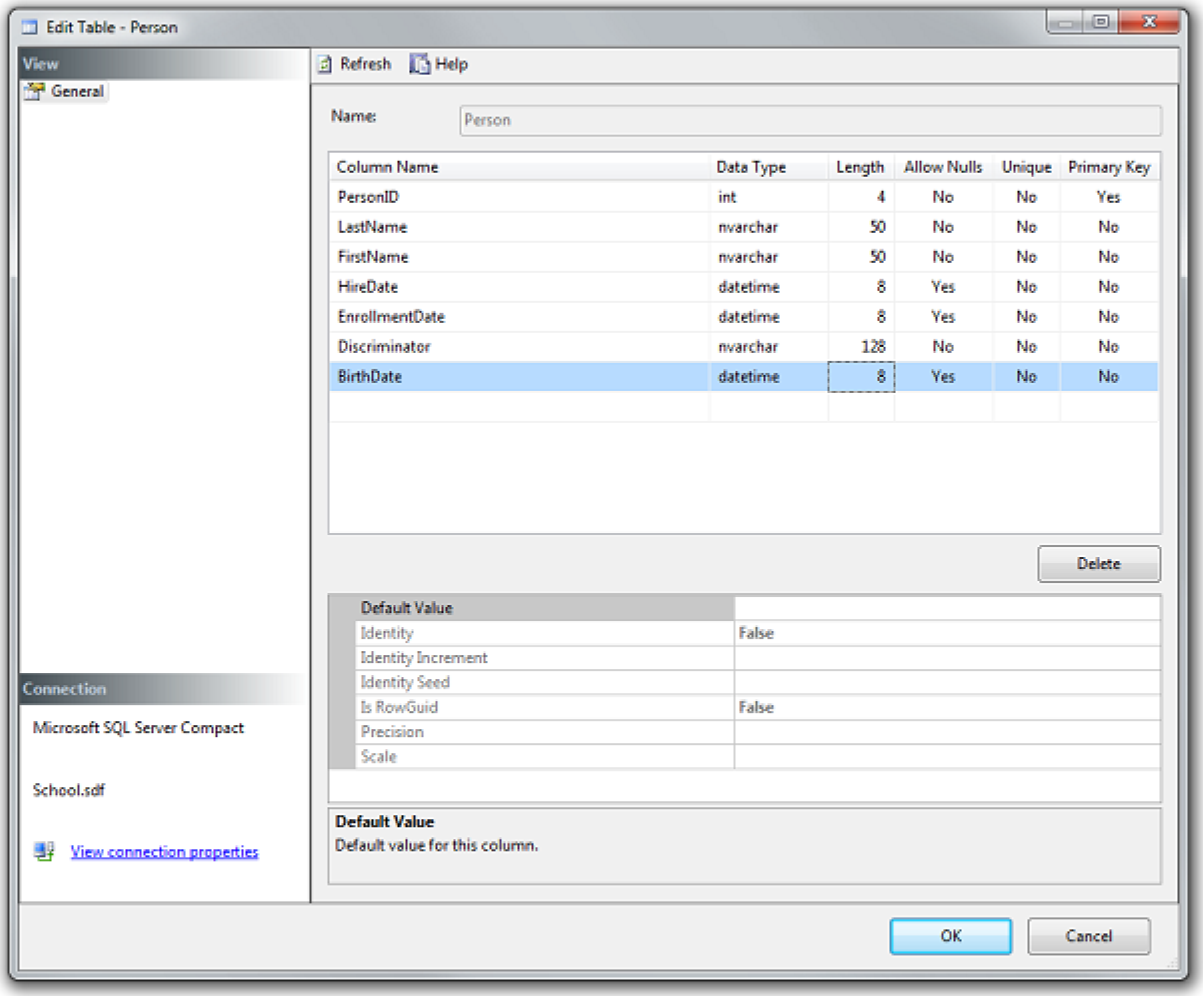
In **Server Explorer**, expand **School-Prod.sdf**, expand **Tables**, expand **Person**, and expand **Columns** and you'll see the old table structure, this time without the new **BirthDate** column.

Right-click **Person** and select **Edit Table Schema** to display the **Edit Table** dialog box.



In the **Edit Table** dialog box, click the line below **Discriminator** to enter information for a new column, and then enter the following:

- **Column Name**: BirthDate
- **Data Type**: datetime


Edit Table - Person

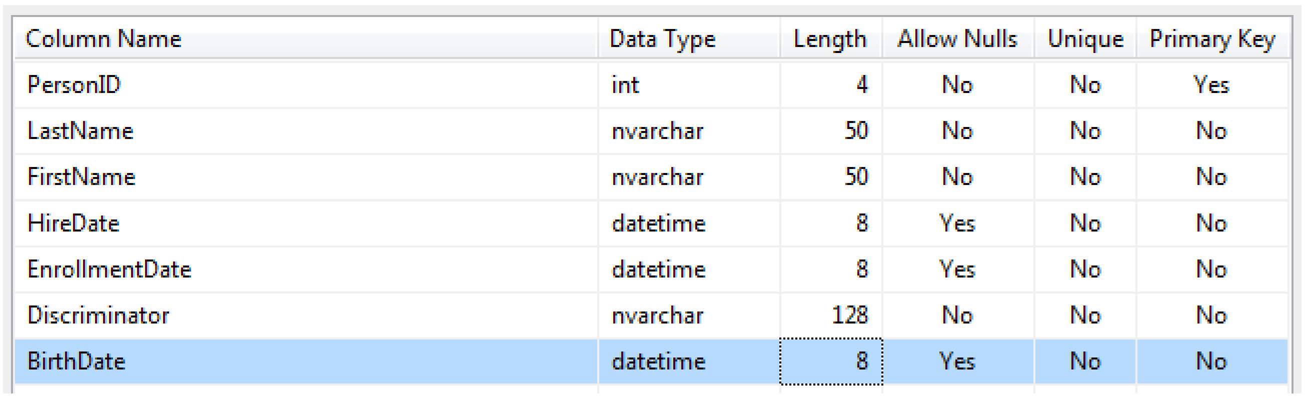
Name: Person

Column Name	Data Type	Length	Allow Nulls	Unique	Primary Key
PersonID	int	4	No	No	Yes
LastName	nvarchar	50	No	No	No
FirstName	nvarchar	50	No	No	No
HireDate	datetime	8	Yes	No	No
EnrollmentDate	datetime	8	Yes	No	No
Discriminator	nvarchar	128	No	No	No
BirthDate	datetime	8	Yes	No	No

Default Value	
Identity	False
Identity Increment	
Identity Seed	
Is RowGuid	False
Precision	
Scale	

Default Value
Default value for this column.

Connection: Microsoft SQL Server Compact, School.sdf


Column Name	Data Type	Length	Allow Nulls	Unique	Primary Key
PersonID	int	4	No	No	Yes
LastName	nvarchar	50	No	No	No
FirstName	nvarchar	50	No	No	No
HireDate	datetime	8	Yes	No	No
EnrollmentDate	datetime	8	Yes	No	No
Discriminator	nvarchar	128	No	No	No
BirthDate	datetime	8	Yes	No	No

The default values for **Length**, **Allow Nulls**, **Unique**, and **Primary Key** are fine as they are.

Click **OK**.

The database will now work correctly with the new code. But the position of the `BirthDate` column in the test database (and later in the production database as well) does not exactly match what Entity Framework Code First has created in the development database. The Entity Framework drops and re-creates the table, and orders the columns in the database to match the code. However, the Visual Studio tool for SQL Server Compact you're using to update the existing table only allows you to put new columns at the end of the column list. As you can imagine, if you make additional database updates this way, it won't be long before the test and production databases look very different from the development database, even if they work the same way with the code.

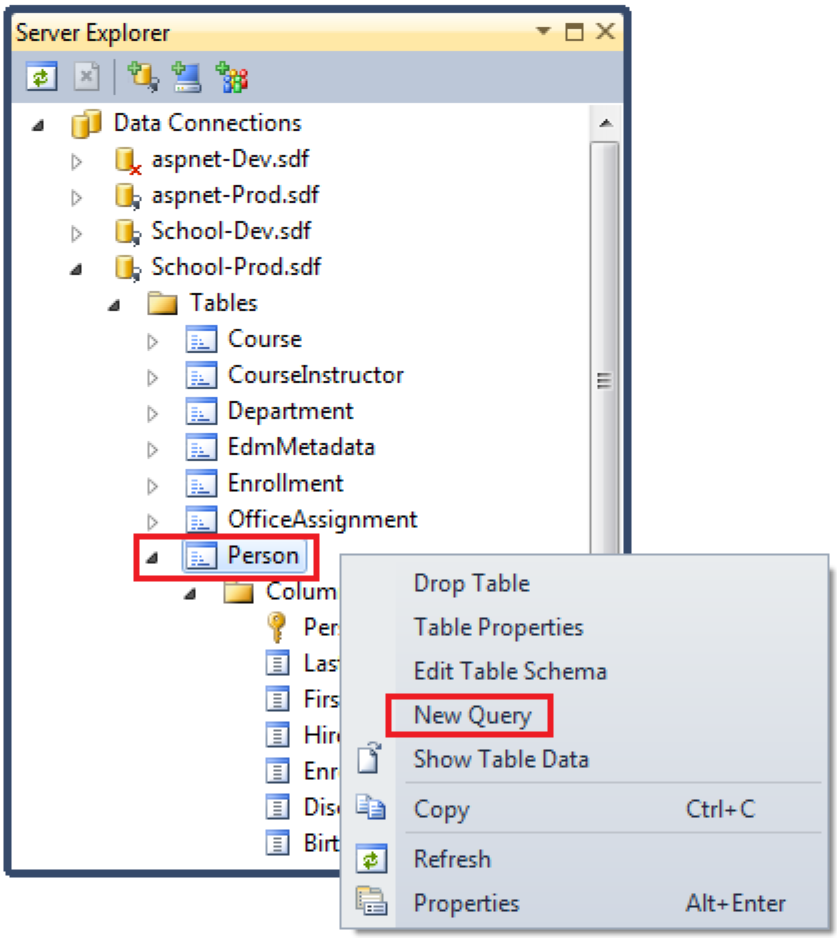
As you work with your databases, you might also run into other limitations of the Visual Studio tool for SQL Server Compact, or you might not be able to replicate a particular change that was done automatically by Code First. If this becomes an issue for you, you have several options, such as the following:

- You can use the database manipulation feature of WebMatrix, which has more features than the SQL Server Compact tooling in Visual Studio.
- You can use more full-featured third-party or open source tools, such as the SQL Server Compact Toolbox and SQL Compact data and schema script utility that are available on the CodePlex site.
- You can write and run your own DDL (data definition language) scripts to manipulate the database schema.
- You can migrate to SQL Server Express and full SQL Server. With these you can use SQL Server Management Studio, which offers a full range of database manipulation features, and you can use Visual Studio 2010 SQL Server database projects or the Database Publishing Wizard to automatically generate scripts. (You might have other reasons for migrating to SQL Server anyway, as explained in the Deploying SQL Server Compact Databases tutorial.)
- You can use Entity Framework Code First Migrations, which automates the process of migrating existing databases to match a new data model. As this tutorial is being written, this software has not yet been released. However, a prerelease version is available for installation in a NuGet package. For the latest information on its availability, see the Entity Framework team blog.

Updating the Data

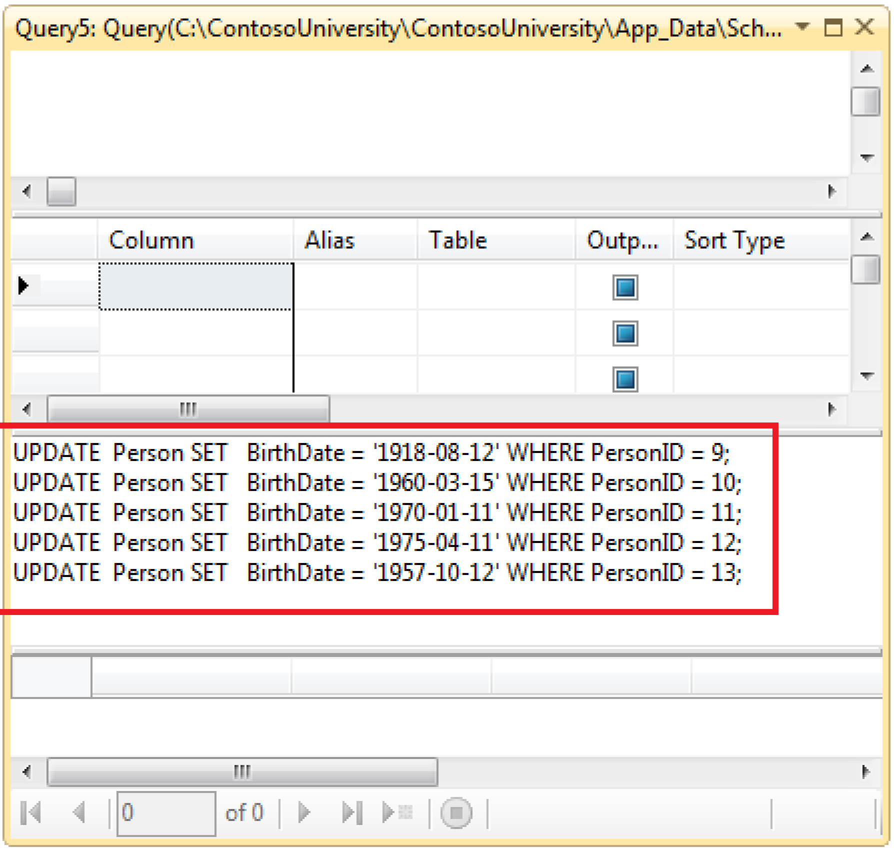
The test database now has a structure that will work with your code, but you need to add the instructor birth dates to it. You could create and run a SQL script or enter the data manually. For this tutorial a script has been created for you, and you'll run that.

In **Server Explorer**, right-click the **Person** table of the *School-Prod.sdf* database and select **New Query**.



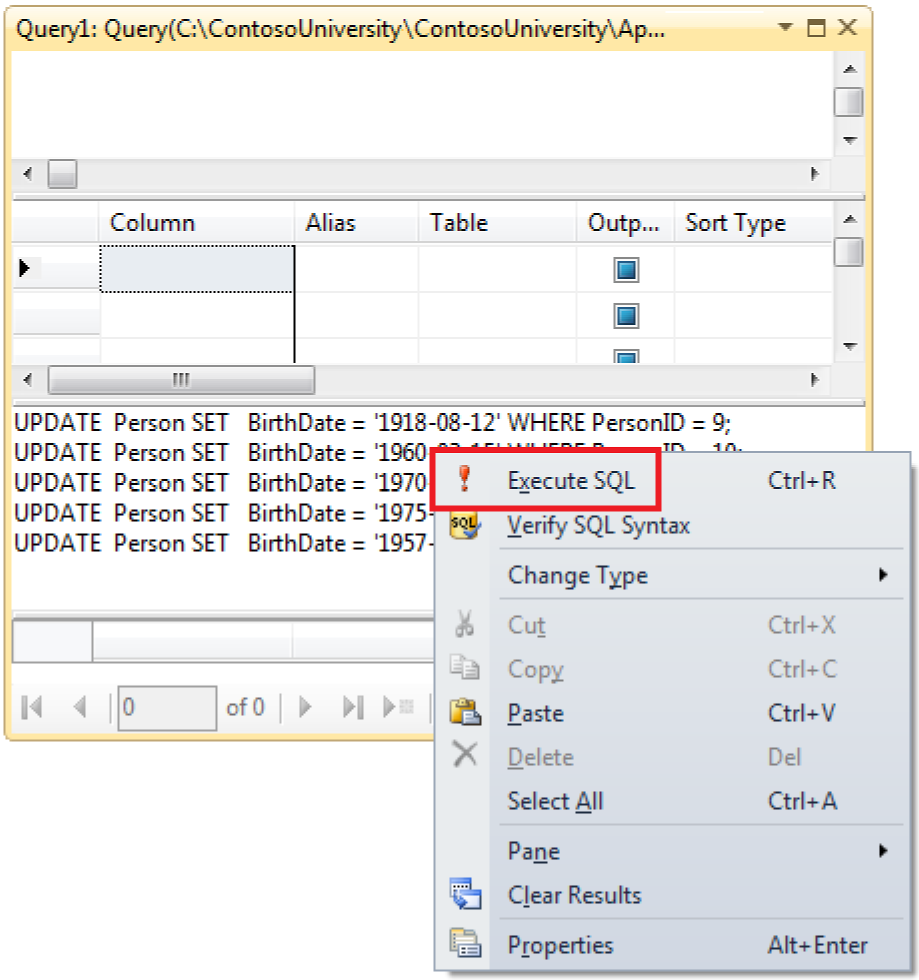
Close the **Add Table** dialog box when it appears, then copy the following SQL statements and paste them into the **SQL** pane of the query window:

```
UPDATE  Person SET  BirthDate = '1918-08-12' WHERE PersonID = 9;
UPDATE  Person SET  BirthDate = '1960-03-15' WHERE PersonID = 10;
UPDATE  Person SET  BirthDate = '1970-01-11' WHERE PersonID = 11;
UPDATE  Person SET  BirthDate = '1975-04-11' WHERE PersonID = 12;
UPDATE  Person SET  BirthDate = '1957-10-12' WHERE PersonID = 13;
```


Query5: Query(C:\ContosoUniversity\ContosoUniversity\App_Data\Sch...
Column
Alias
Table
Outp...
Sort Type
of 0

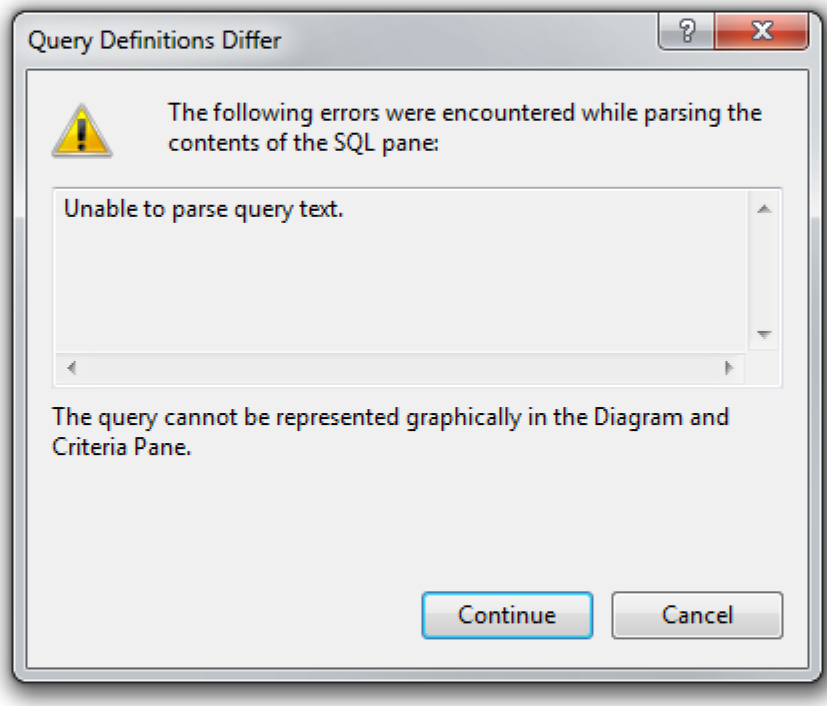
```
UPDATE  Person SET  BirthDate = '1918-08-12' WHERE PersonID = 9;
UPDATE  Person SET  BirthDate = '1960-03-15' WHERE PersonID = 10;
UPDATE  Person SET  BirthDate = '1970-01-11' WHERE PersonID = 11;
UPDATE  Person SET  BirthDate = '1975-04-11' WHERE PersonID = 12;
UPDATE  Person SET  BirthDate = '1957-10-12' WHERE PersonID = 13;
```

Right-click the **SQL** pane and select **Execute SQL**.


Query1: Query(C:\ContosoUniversity\ContosoUniversity\Ap...

```
UPDATE  Person SET  BirthDate = '1918-08-12' WHERE PersonID = 9;
UPDATE  Person SET  BirthDate = '1960
UPDATE  Person SET  BirthDate = '1970
UPDATE  Person SET  BirthDate = '1975-
UPDATE  Person SET  BirthDate = '1957-
```

When you get the **Query Definitions Differ** dialog box, click **Continue**. This message just tells you that the query can't be represented graphically in the designer pane.

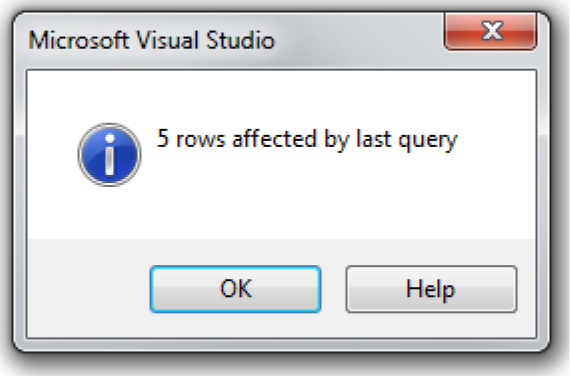

Query Definitions Differ

The following errors were encountered while parsing the contents of the SQL pane:

Unable to parse query text.

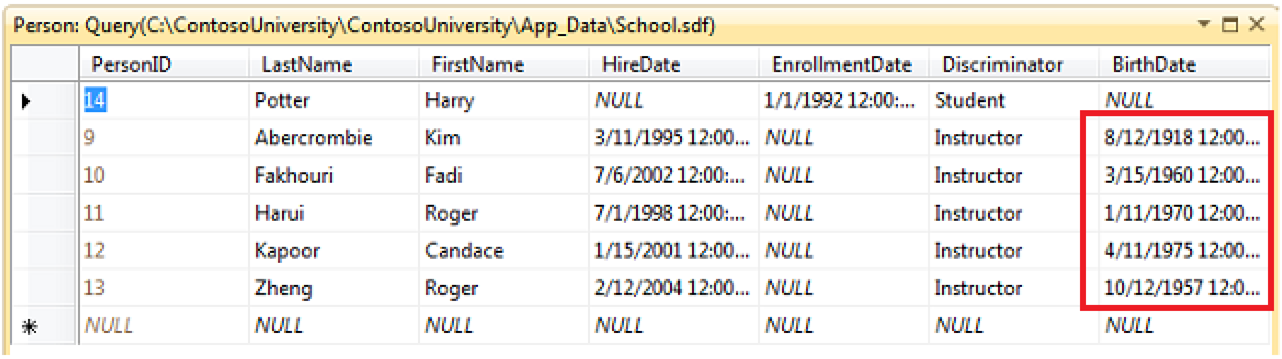
The query cannot be represented graphically in the Diagram and Criteria Pane.

The query runs, and a dialog box confirms that 5 rows were updated.


Microsoft Visual Studio

5 rows affected by last query

In **Server Explorer**, right-click the **Person** table again and select **Show Table Data** to see that the instructors now all have birth dates.



Person: Query(C:\ContosoUniversity\ContosoUniversity\App_Data\School.sdf)

	PersonID	LastName	FirstName	HireDate	EnrollmentDate	Discriminator	BirthDate
▶	14	Potter	Harry	NULL	1/1/1992 12:00:...	Student	NULL
	9	Abercrombie	Kim	3/11/1995 12:00...	NULL	Instructor	8/12/1918 12:00...
	10	Fakhouri	Fadi	7/6/2002 12:00:...	NULL	Instructor	3/15/1960 12:00...
	11	Harui	Roger	7/1/1998 12:00:...	NULL	Instructor	1/11/1970 12:00...
	12	Kapoor	Candace	1/15/2001 12:00...	NULL	Instructor	4/11/1975 12:00...
	13	Zheng	Roger	2/12/2004 12:00...	NULL	Instructor	10/12/1957 12:0...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

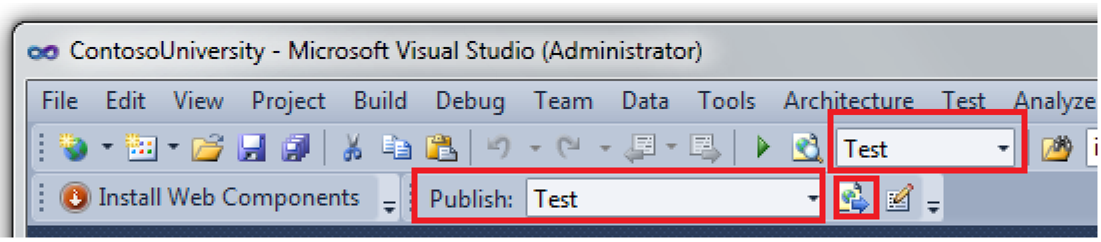
The test database is now ready to deploy.

Copying the Database to the Test Site

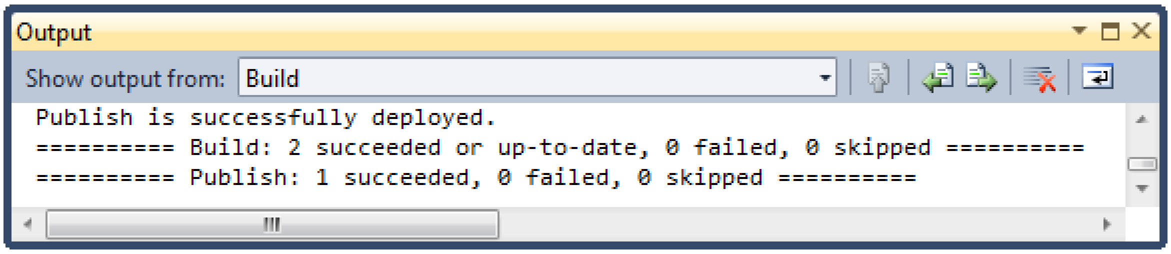
Use Windows Explorer to copy *School-Prod.sdf* from the *App_Data* folder in the ContosoUniversity project to the *App_Data* folder in the test site (*C:\inetpub\wwwroot\contosouniversity\App_Data*), overwriting the copy of *School-Prod.sdf* that is already in the destination folder.

Deploying the Code Changes that Go With the Database Change

You are now ready to deploy the code update. In the **Solution Configurations** drop-down box, select **Test** for build configuration and in the **Publish** profile drop-down box select **Test** . Click **Publish Web**. (If you can't do this in the toolbars because you have have customized toolbars, see the Deploying a Code-Only Update tutorial for an alternative method.)

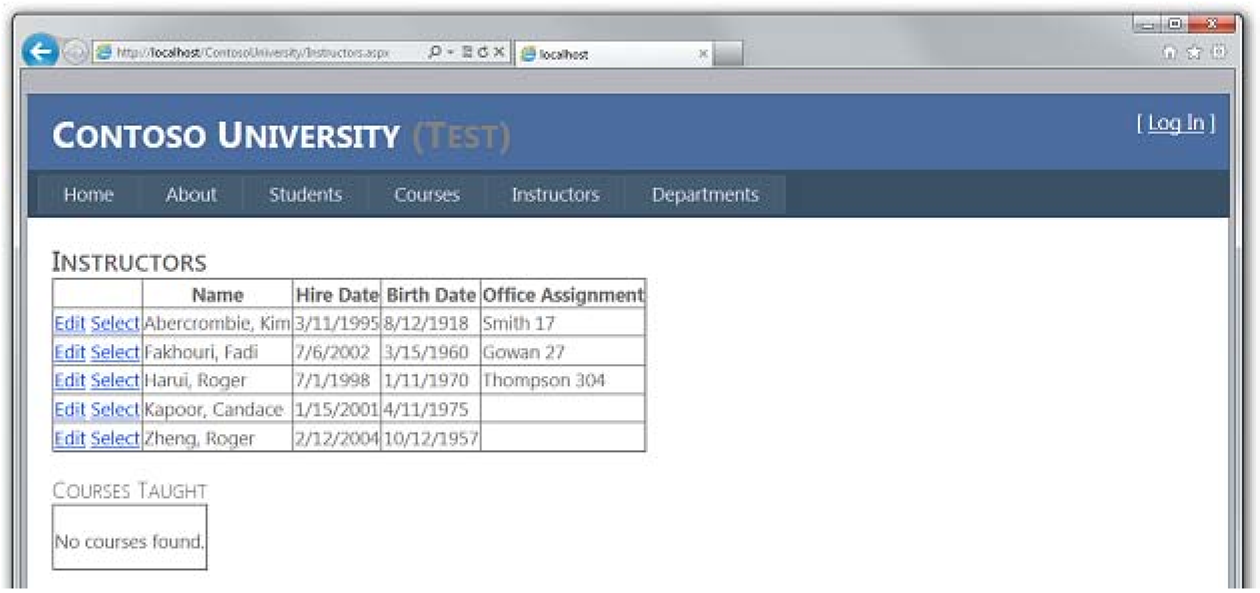


Visual Studio deploys the updated application and reports success in the **Output** window.



```
Publish is successfully deployed.
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

You can now and run the application at http://localhost/contosouniversity to verify that the update was successfully deployed. You see the birth date with actual dates in the **Instructors** page.


http://localhost/ContosoUniversity/Instructors.aspx
localhost
CONTOSO UNIVERSITY (TEST)
[Log In]
Home
About
Students
Courses
Instructors
Departments

INSTRUCTORS

	Name	Hire Date	Birth Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	8/12/1918	Smith 17
Edit Select	Fakhouri, Fadi	7/6/2002	3/15/1960	Gowan 27
Edit Select	Harui, Roger	7/1/1998	1/11/1970	Thompson 304
Edit Select	Kapoor, Candace	1/15/2001	4/11/1975	
Edit Select	Zheng, Roger	2/12/2004	10/12/1957	

COURSES TAUGHT

No courses found.

You can also run the **Students** and **Add Students** pages to verify that the update was successful.

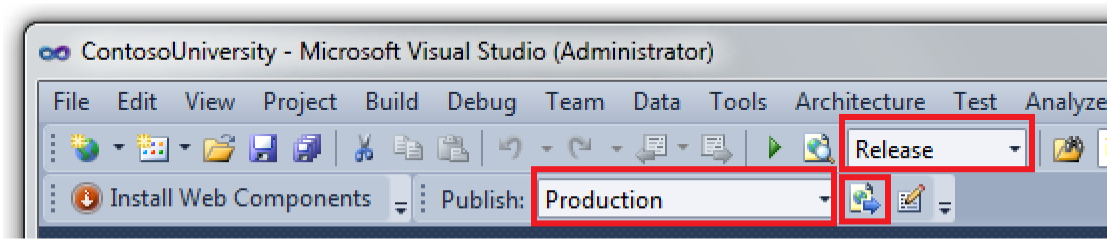
Deploying the Database Update to the Production Environment

You can now deploy to production. You'll duplicate the process you used in test, except for downloading the production database first and using *app_offline.htm* to prevent users from using the site and thus updating the dataabase while you're deploying changes. For production deployment perform the following steps:

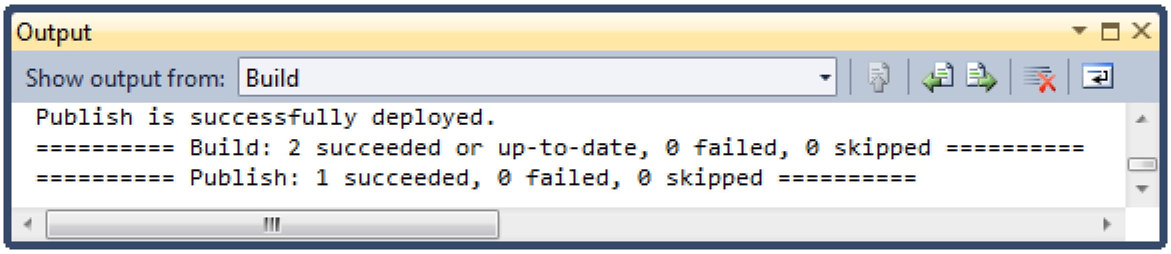
- Upload the *app_offline.htm* file to the production site.
- Download *School-Prod.sdf* from the production site to your local *App_Data* folder, and copy it to a backup location (see the note below).
- Manually add the new column.
- Run the update script. (In a real site you might have to change the script before running it, since the data in the production site might have changed since the script was originally created.)
- Upload the updated *School-Prod.sdf* file to the production site.
- Publish from Visual Studio.

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 make a change like this one, 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.

The last step is the same as what you did in the previous tutorial for a code-only update. In the **Solution Configurations** drop-down box, select **Release** and in the **Publish** profile drop-down box select **Production**. Then click **Publish Web**.


ContosoUniversity - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze
Release
Install Web Components
Publish:
Production

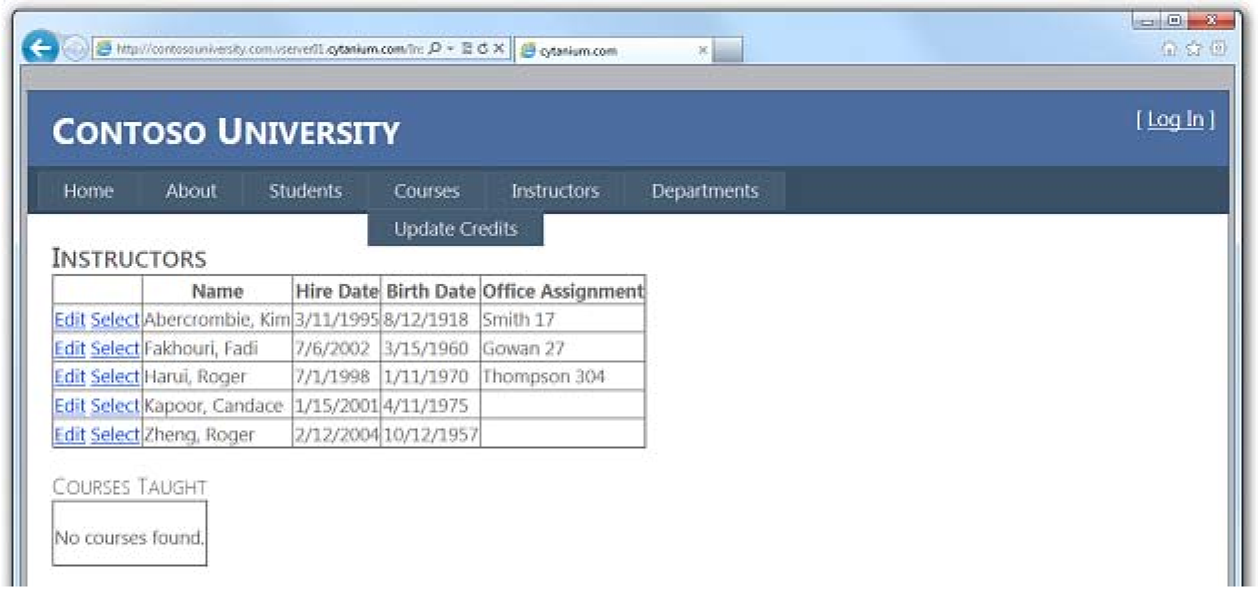
Visual Studio deploys the updated application and reports success in the **Output** window.



```
Publish is successfully deployed.
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

Before you can test to verify successful deployment, you have to remove the *app_offline.htm* file.

You can now open a browser and browse your public site to verify that the update was successfully deployed.



INSTRUCTORS

	Name	Hire Date	Birth Date	Office Assignment
Edit Select	Abercrombie, Kim	3/11/1995	8/12/1918	Smith 17
Edit Select	Fakhouri, Fadi	7/6/2002	3/15/1960	Gowan 27
Edit Select	Harui, Roger	7/1/1998	1/11/1970	Thompson 304
Edit Select	Kapoor, Candace	1/15/2001	4/11/1975	
Edit Select	Zheng, Roger	2/12/2004	10/12/1957	

COURSES TAUGHT

No courses found.

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 full SQL Server.

Migrating to SQL Server - 10 of 12

Overview

The Deploying SQL Server Compact and Deploying a Database Update tutorials explained some of the reasons why you might eventually want to upgrade from SQL Server Compact to full SQL Server. This tutorial shows you how to do that.

SQL Server Express versus full SQL Server for Development

Once you've decided to upgrade to full SQL Server, you'll want to use SQL Server Express or full SQL Server 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.

Typically, you choose SQL Server Express for your development and test environment (when the test environment is on your local computer) because SQL Server Express is free and is installed with Visual Studio by default. 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 full SQL Server in production. You can use most of the same database tools with SQL Server Express that you can use with full SQL Server (a notable exception being SQL Server Profiler), and it supports other features of full SQL Server like stored procedures, views, triggers, and replication. (You typically need to use full SQL Server in production, 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.)

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 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:

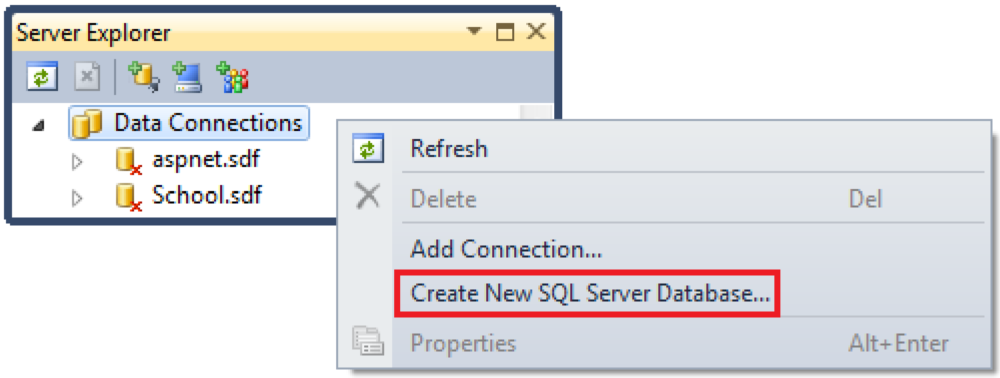
- Migrate to two SQL Server Express databases in the development and test environments.
- Migrate to one combined full SQL Server database in the production environment.

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.

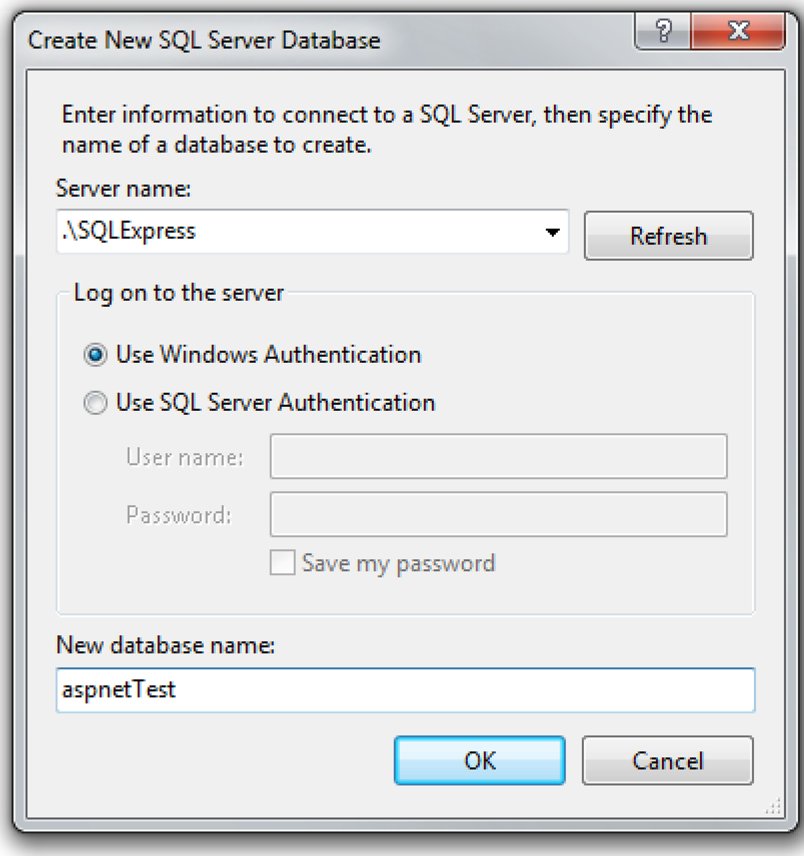
Creating SQL Server Express Databases for the Test Environment

Before you can deploy to SQL Server Express databases in the test environment, you have to manually create the databases. You won't have to manually create tables or insert data into them; Web Deploy will do that for you automatically.

From the **View** menu select **Server Explorer**, 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 "aspnetTest" in the **New database name** box, then click **OK**.


Create New SQL Server Database

Enter information to connect to a SQL Server, then specify the name of a database to create.

Server name: .\SQLExpress

Log on to the server

[✓ Use Windows Authentication]

Use SQL Server Authentication

User name:

Password:

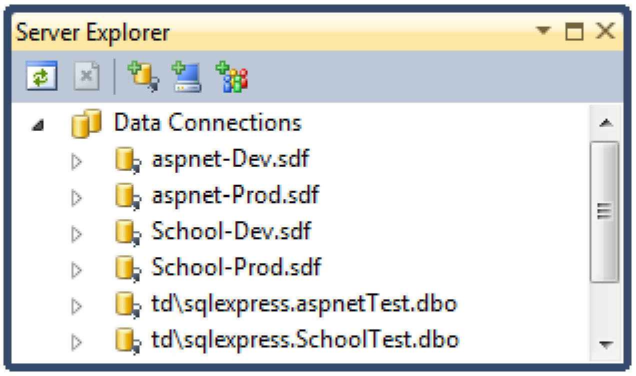
Save my password

New database name: aspnetTest

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

(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.


Server Explorer

- Data Connections
  - aspnet-Dev.sdf
  - aspnet-Prod.sdf
  - School-Dev.sdf
  - School-Prod.sdf
  - td\sqlexpress.aspnetTest.dbo
  - td\sqlexpress.SchoolTest.dbo

Creating Grant Scripts for the New Databases

When the application runs in IIS on your development computer, the database will be accessed using the default application pool's credentials. However, by default, the application pool identity does not have permission to open the databases. So you need 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_datareader', 'ContosoUniversityUser'
GO
EXEC sp_addrolemember 'db_datawriter', '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.

Configuring Database Deployment for the Test Environment

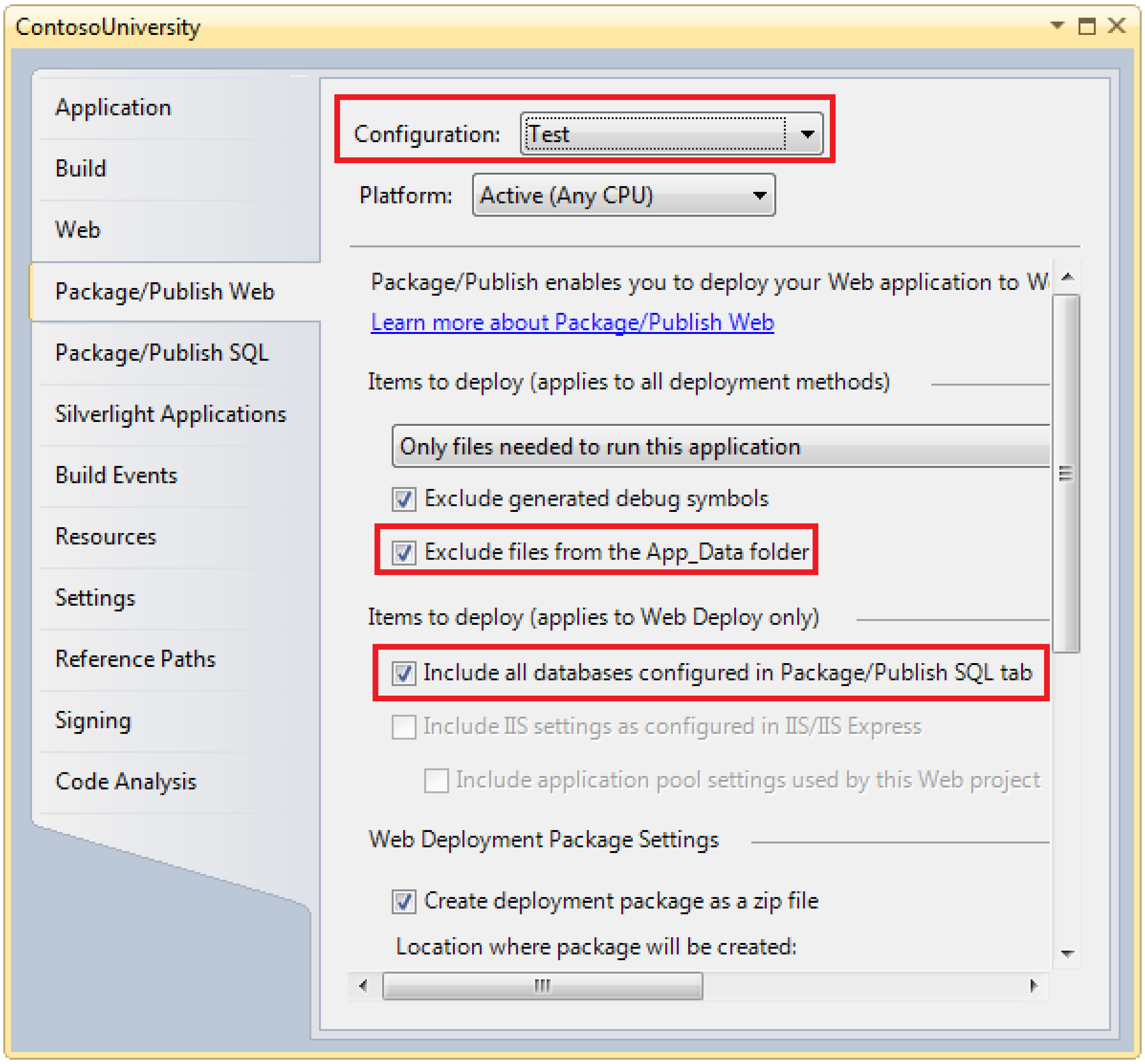
Currently your deployment settings are designed to handle database deployment that involves just copying files in the *App_Data* folder. Now you need to configure deployment to create SQL scripts and run them in the destination database. Essentially, what Web Deploy will do for you is the following for each database that you are deploying:

- 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 scripts in the destination database.

In order to begin configuring settings for deployment to the test environment, open the **Project Properties** window and select the **Package/Publish Web** tab. Then select **Active (Test)** or **Test** in the **Configuration** drop-down list.

Make sure that **Exclude files from the App_Data folder** is selected. Web Deploy will read the data in the *.sdf* files in *App_Data* in order to create tables and data in the new SQL Server Express databases, but it should not copy the files to the destination.

Make sure that **Include all databases configured in Package/Publish SQL tab** is selected. The **Package/Publish SQL** tab is where you'll configure deployment to the new SQL Server Express databases. Until now this check box has been selected, but it has had no effect because you didn't specify any databases on the **Package/Publish SQL** tab.



Configuration: Test

Platform: Active (Any CPU)

Package/Publish enables you to deploy your Web application to W

Learn more about Package/Publish Web

Items to deploy (applies to all deployment methods)

Only files needed to run this application

☑ Exclude generated debug symbols

☑ Exclude files from the App_Data folder

Items to deploy (applies to Web Deploy only)

☑ Include all databases configured in Package/Publish SQL tab

☐ Include IIS settings as configured in IIS/IIS Express

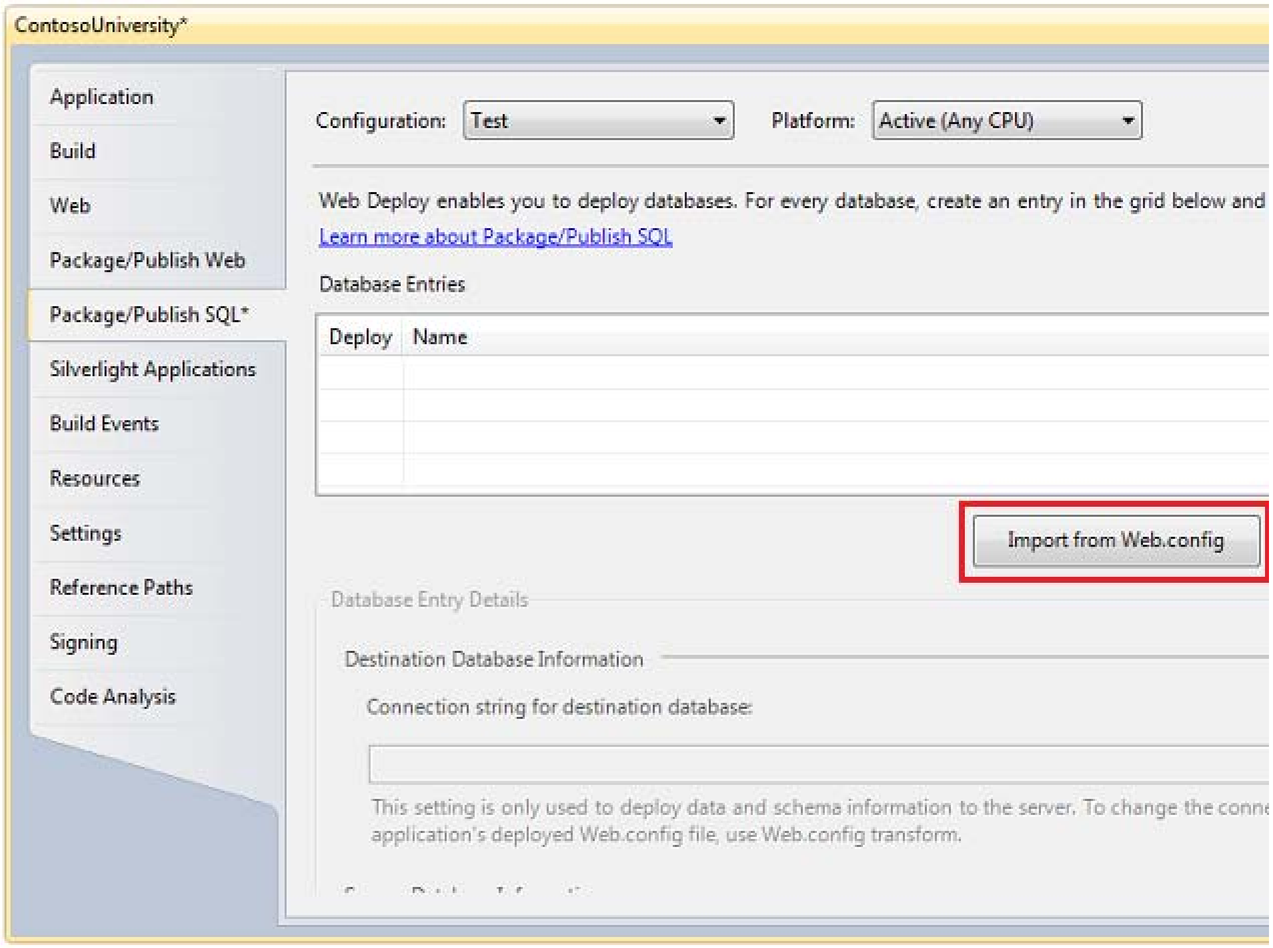
☐ Include application pool settings used by this Web project

Web Deployment Package Settings

☑ Create deployment package as a zip file

Location where package will be created:

Select the **Package/Publish SQL** tab, and with the build configuration still set to **Test**, click **Import from Web.config**.



Configuration: Test Platform: Active (Any CPU)

Web Deploy enables you to deploy databases. For every database, create an entry in the grid below and

Learn more about Package/Publish SQL

Database Entries

Deploy	Name

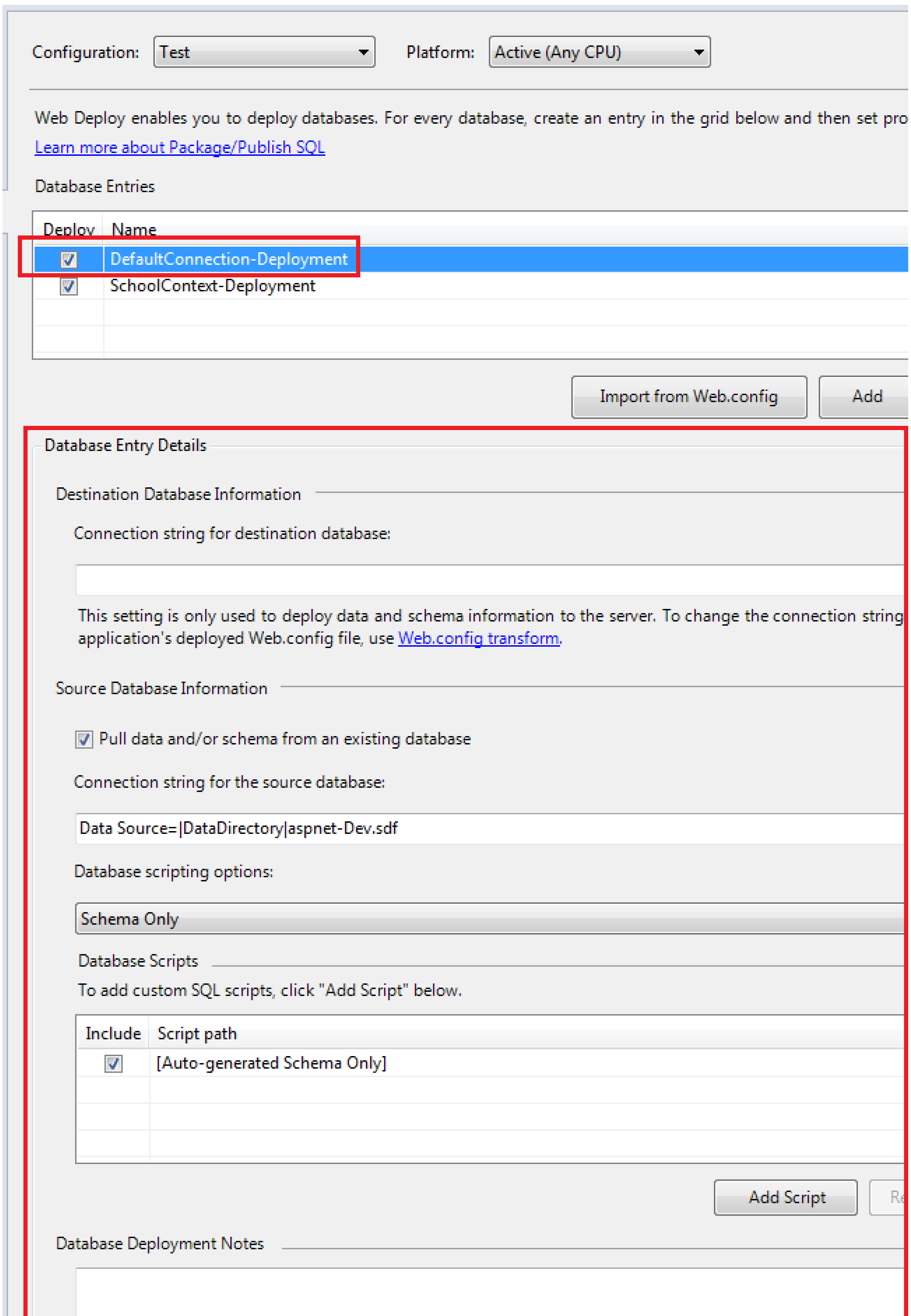
Database Entry Details

Destination Database Information

Connection string for destination database:

This setting is only used to deploy data and schema information to the server. To change the conne
application's deployed Web.config file, use Web.config transform.

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. (The changes you need to make will be detailed below. At this point it's just important to understand the master-detail relationship between the **Database Entries** and **Database Entry Details** sections.)


Configuration: Test   Platform: Active (Any CPU)

Web Deploy enables you to deploy databases. For every database, create an entry in the grid below and then set pro

Learn more about Package/Publish SQL

Database Entries

Deploy	Name
☑	DefaultConnection-Deployment
☑	SchoolContext-Deployment

Import from Web.config   Add

Database Entry Details

Destination Database Information

Connection string for destination database:

This setting is only used to deploy data and schema information to the server. To change the connection string application's deployed Web.config file, use Web.config transform.

Source Database Information

☑ Pull data and/or schema from an existing database

Connection string for the source database:

Data Source=|DataDirectory|aspnet-Dev.sdf

Database scripting options:

Schema Only

Database Scripts

To add custom SQL scripts, click "Add Script" below.

Include	Script path
☑	[Auto-generated Schema Only]

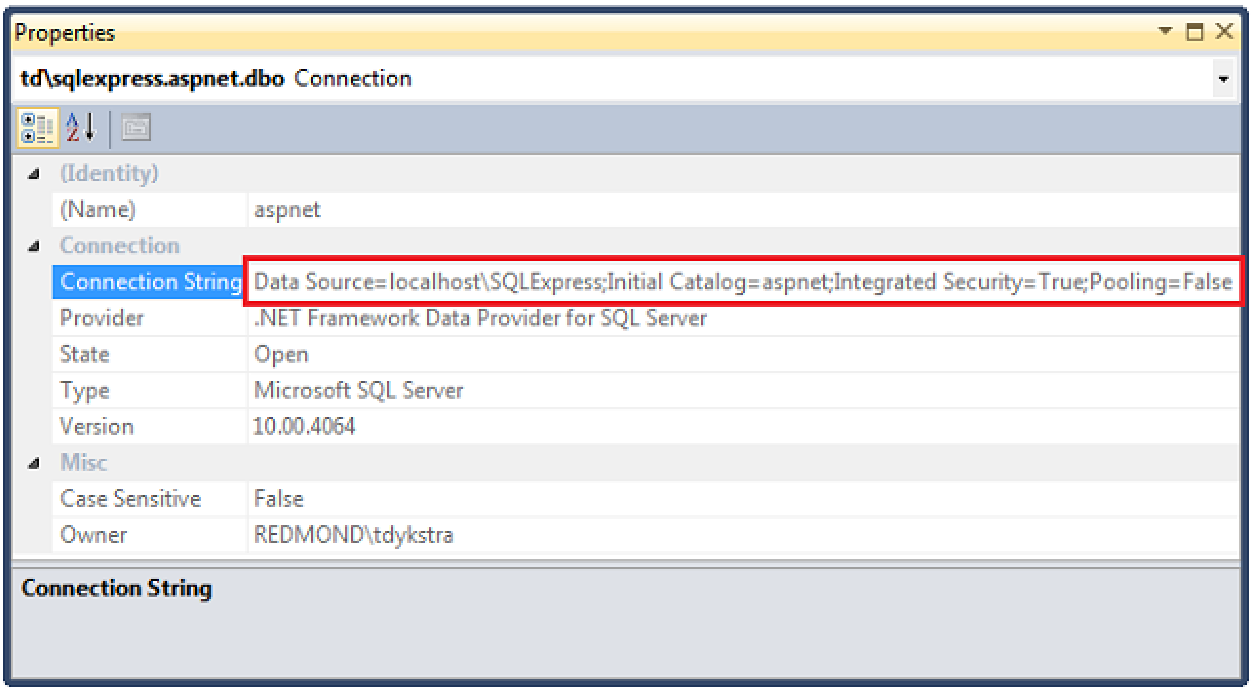
Add Script

Database Deployment Notes

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**, you need to 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.



(Identity)	
(Name)	aspnet
Connection	
Connection String	Data Source=localhost\SQLExpress;Initial Catalog=aspnet;Integrated Security=True;Pooling=False
Provider	.NET Framework Data Provider for SQL Server
State	Open
Type	Microsoft SQL Server
Version	10.00.4064
Misc	
Case Sensitive	False
Owner	REDMOND\tdykstra

The connection string is also shown below. Copy this connection string into **Connection string for destination database** in the **Package/Publish SQL** tab.

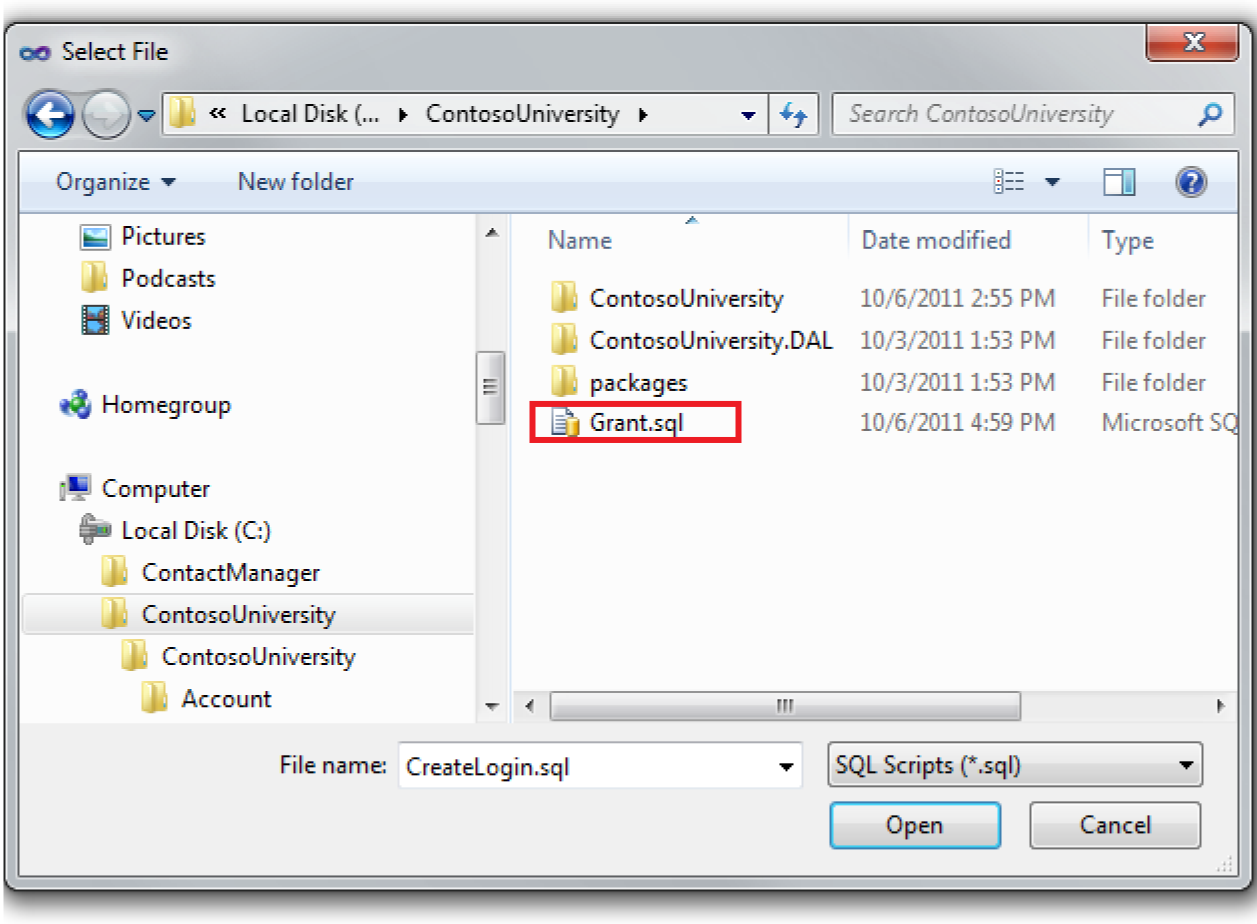
```
Data Source=.\SQLExpress;Initial Catalog=aspnetTest;Integrated
Security=True;Pooling=False
```

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 (test accounts) as well as the database structure.

To configure deployment to run the grant scripts that you created, you need to add them to the **Database Scripts** section. Click **Add Script**, and in the **Select File** 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**.

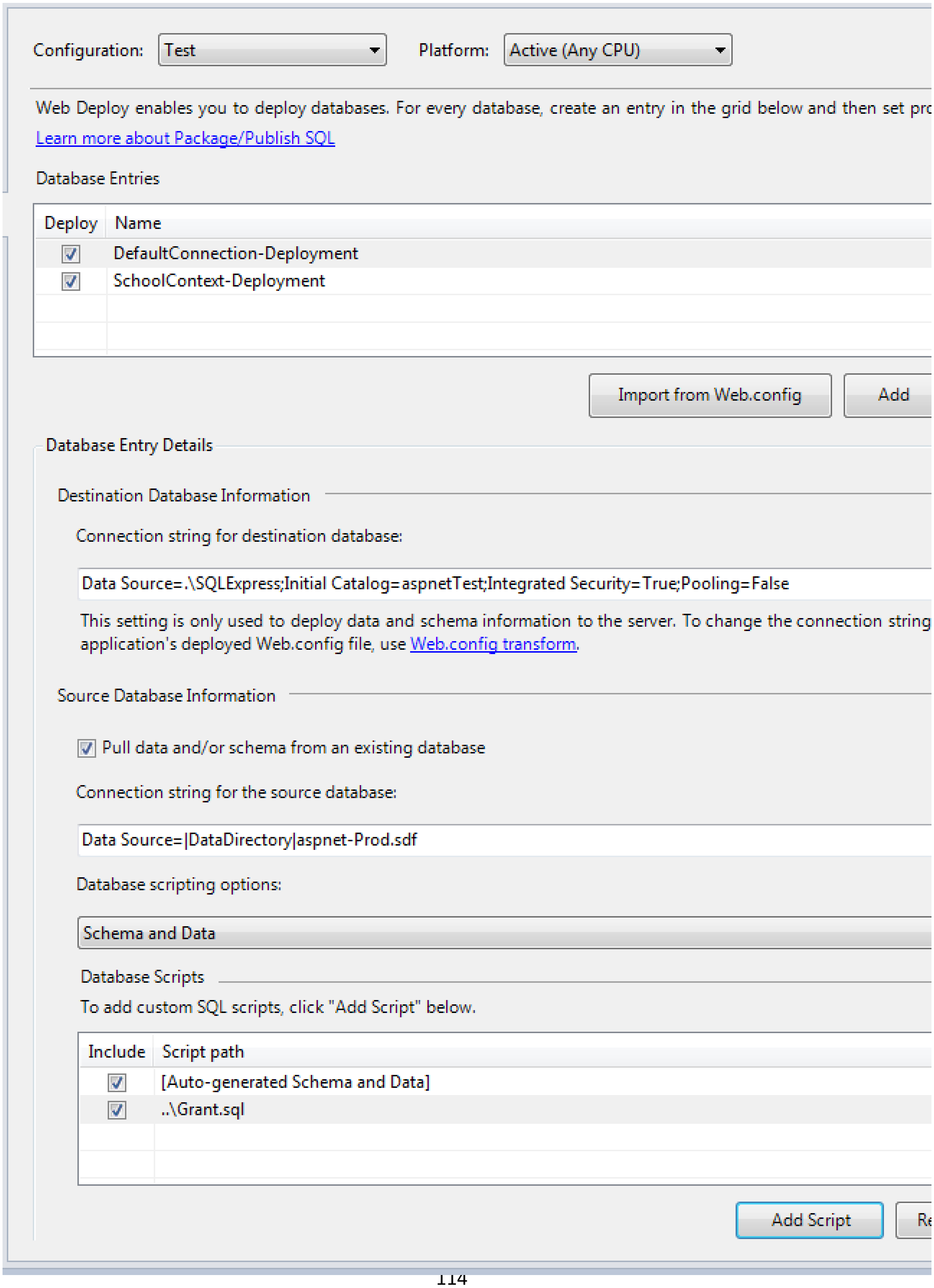


Name	Date modified	Type
ContosoUniversity	10/6/2011 2:55 PM	File folder
ContosoUniversity.DAL	10/3/2011 1:53 PM	File folder
packages	10/3/2011 1:53 PM	File folder
Grant.sql	10/6/2011 4:59 PM	Microsoft SQ

File name: CreateLogin.sql

SQL Scripts (*.sql)

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


Configuration: Test

Platform: Active (Any CPU)

Web Deploy enables you to deploy databases. For every database, create an entry in the grid below and then set pro

Learn more about Package/Publish SQL

Database Entries

Deploy	Name
✓	DefaultConnection-Deployment
✓	SchoolContext-Deployment

Import from Web.config | Add

Database Entry Details

Destination Database Information

Connection string for destination database:

Data Source=.\SQLExpress;Initial Catalog=aspnetTest;Integrated Security=True;Pooling=False

This setting is only used to deploy data and schema information to the server. To change the connection string application's deployed Web.config file, use Web.config transform.

Source Database Information

[✓] Pull data and/or schema from an existing database

Connection string for the source database:

Data Source=|DataDirectory|aspnet-Prod.sdf

Database scripting options:

Schema and Data

Database Scripts

To add custom SQL scripts, click "Add Script" below.

Include	Script path
✓	[Auto-generated Schema and Data]
✓	..\Grant.sql

Add Script | Re

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 deployment settings.

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

```
Data Source=.\SQLExpress;Initial Catalog=SchoolTest;Integrated
Security=True;Pooling=False
```

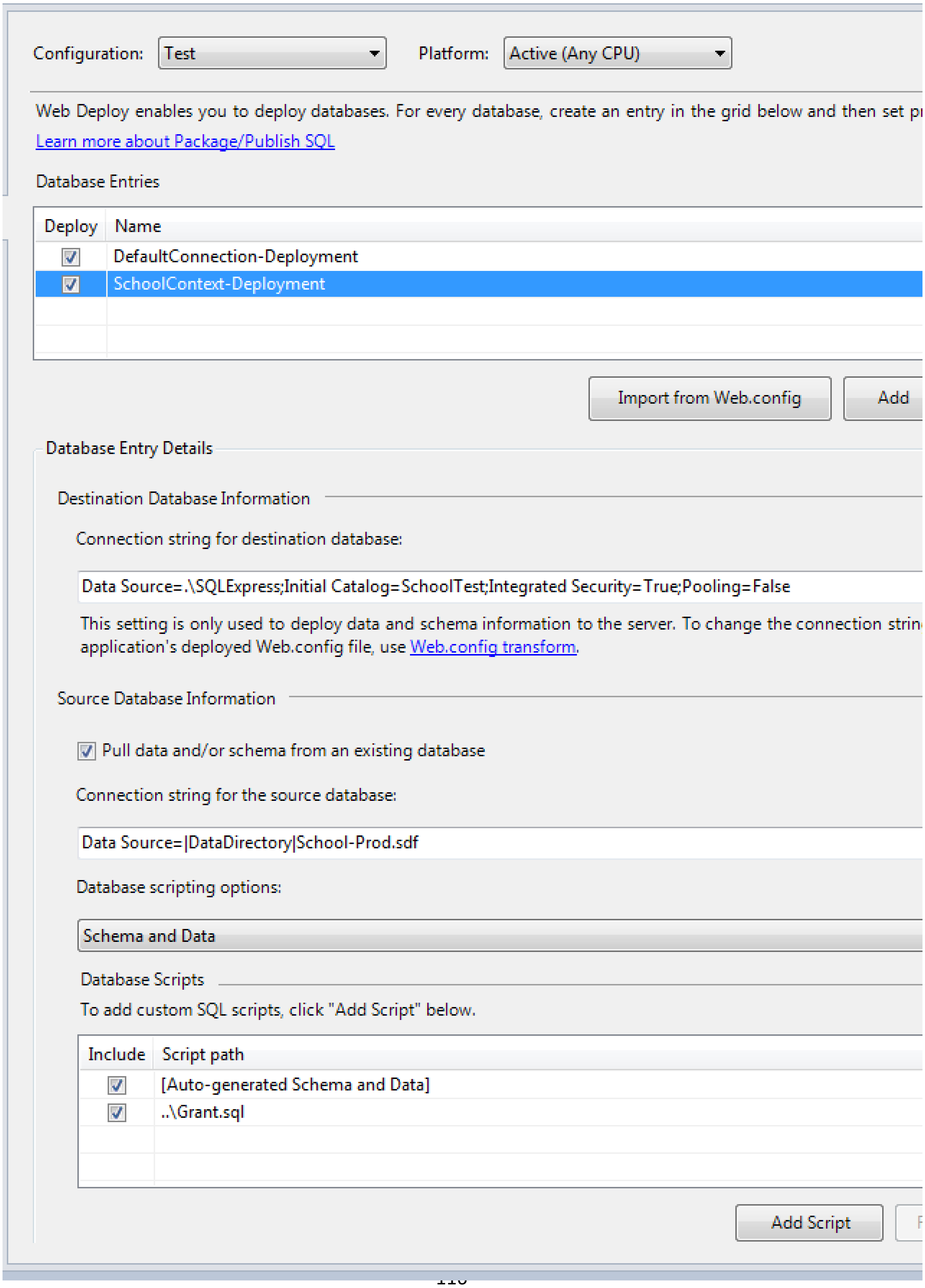
Make sure that **Pull data and/or schema from an existing database** is selected.

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.

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 this:



Configuration: Test

Platform: Active (Any CPU)

Web Deploy enables you to deploy databases. For every database, create an entry in the grid below and then set p

Learn more about Package/Publish SQL

Database Entries

Deploy	Name
✓	DefaultConnection-Deployment
✓	SchoolContext-Deployment

Database Entry Details

Destination Database Information

Connection string for destination database:

Data Source=.\SQLExpress;Initial Catalog=SchoolTest;Integrated Security=True;Pooling=False

This setting is only used to deploy data and schema information to the server. To change the connection strin
application's deployed Web.config file, use Web.config transform.

Source Database Information

✓ Pull data and/or schema from an existing database

Connection string for the source database:

Data Source=|DataDirectory|School-Prod.sdf

Database scripting options:

Schema and Data

Database Scripts

To add custom SQL scripts, click "Add Script" below.

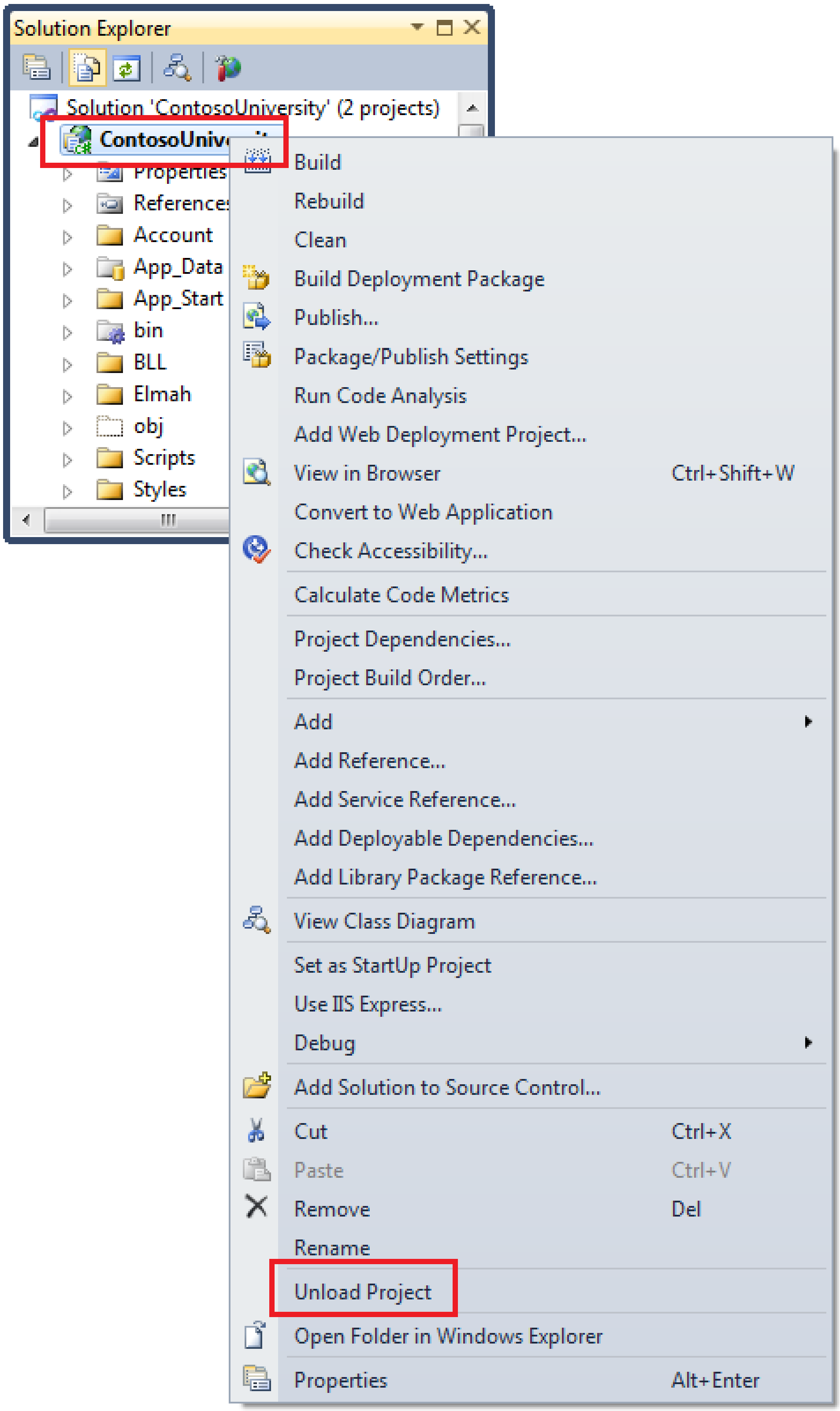
Include	Script path
✓	[Auto-generated Schema and Data]
✓	..\Grant.sql

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

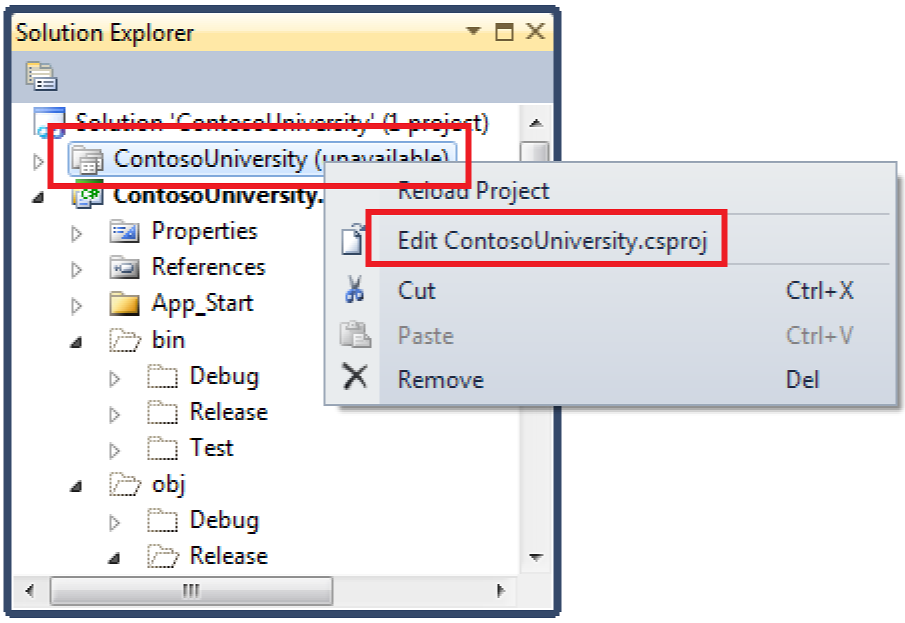
Specifying Transacted Mode for the Grant Script

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'll edit the project file in order to configure the custom scripts to run in a transaction.

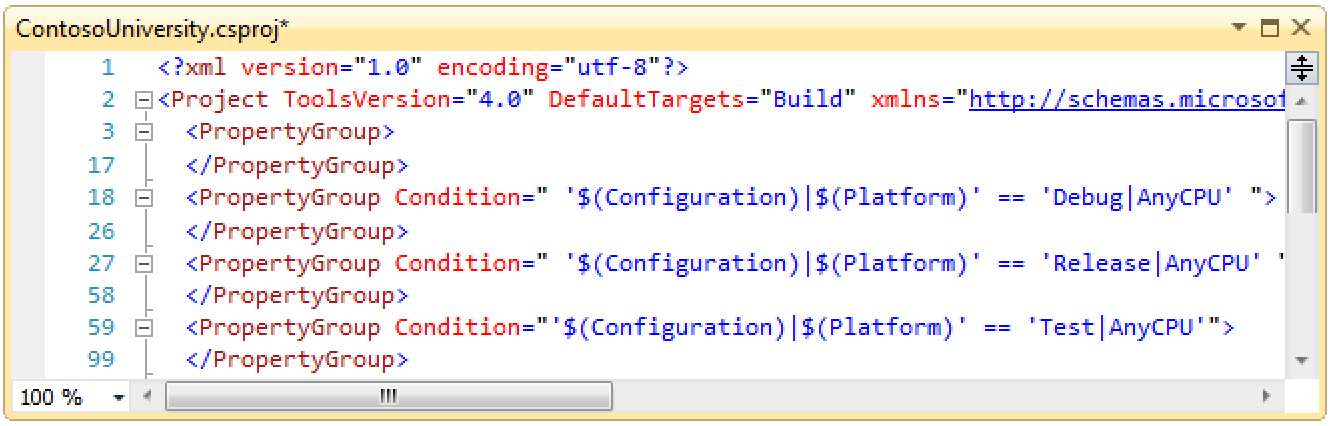
In **Solution Explorer**, right-click the **ContosoUniversity** project and select **Unload Project**.



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

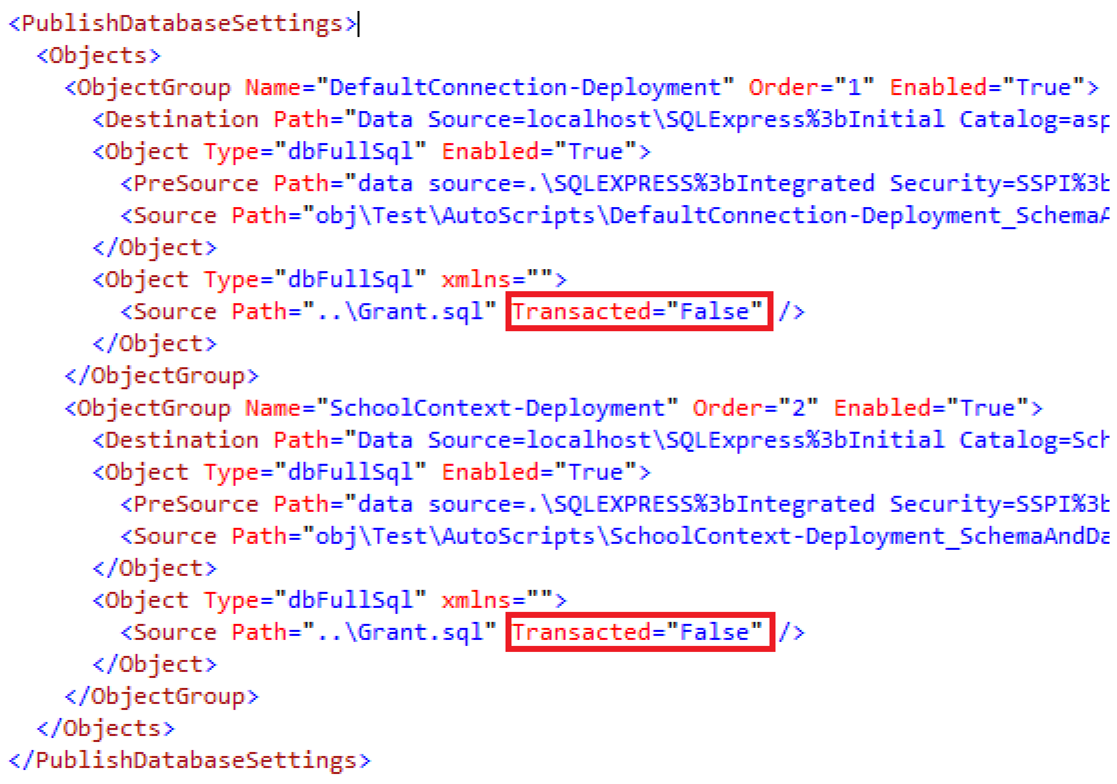


The Visual Studio editor shows you the XML content of the project file. Notice that there are several `PropertyGroup` elements. (In the image, the contents of the `PropertyGroup` elements have been omitted.)



```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.microsof
  <PropertyGroup>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' "
  </PropertyGroup>
  <PropertyGroup Condition="'$(Configuration)|$(Platform)' == 'Test|AnyCPU'">
  </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 Test build configuration, you'll see a `PublishDatabaseSettings` element that contains the settings you entered on the **Package/Publish SQL** tab:

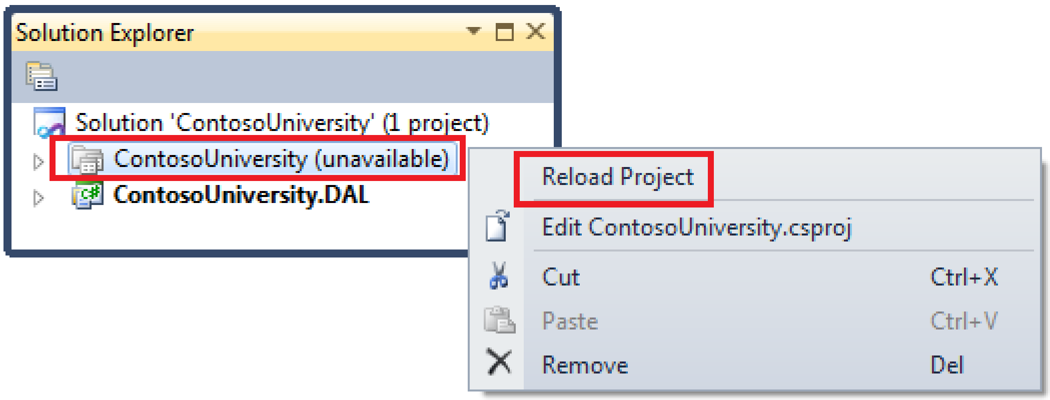


```
<PublishDatabaseSettings>
  <Objects>
    <ObjectGroup Name="DefaultConnection-Deployment" Order="1" Enabled="True">
      <Destination Path="Data Source=localhost\SQLExpress%3bInitial Catalog=asp
      <Object Type="dbFullSql" Enabled="True">
        <PreSource Path="data source=.\SQLEXPRESS%3bIntegrated 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>
```

You'll notice there is an `Object` element that corresponds to each of the grant scripts you specified (notice the two instances of "Grant.sql"). In the `Object` elements for the grant scripts, change the value of the `Transacted` attribute of the `Source` element to `True`. The `PublishDatabaseSettings` element now looks like this:

```
<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**.


Solution Explorer
Solution 'ContosoUniversity' (1 project)
ContosoUniversity (unavailable)
ContosoUniversity.DAL
Reload Project
Edit ContosoUniversity.csproj
Cut Ctrl+X
Paste Ctrl+V
Remove Del

Setting up Web.Config Transforms for the Connection Strings to Test Databases

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.

Open *Web.Test.config* and replace the `connectionStrings` element with the `connectionStrings` element shown below:

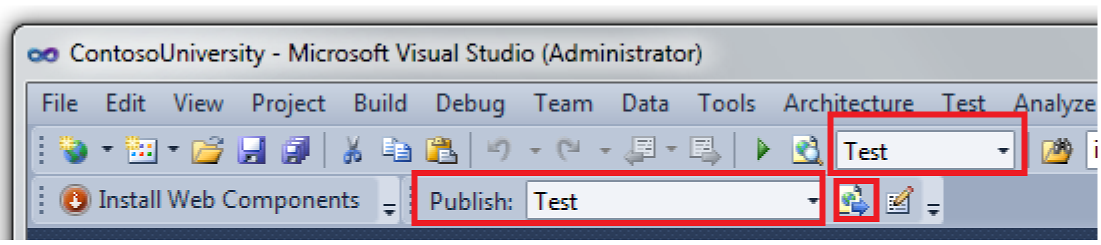
```
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-
Transform">
<connectionStrings>
<add name="DefaultConnection"
         connectionString="Data Source=.\SQLExpress;Initial
Catalog=aspnetTest;Integrated
Security=True;Pooling=False;MultipleActiveResultSets=True"
         providerName="System.Data.SqlClient"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

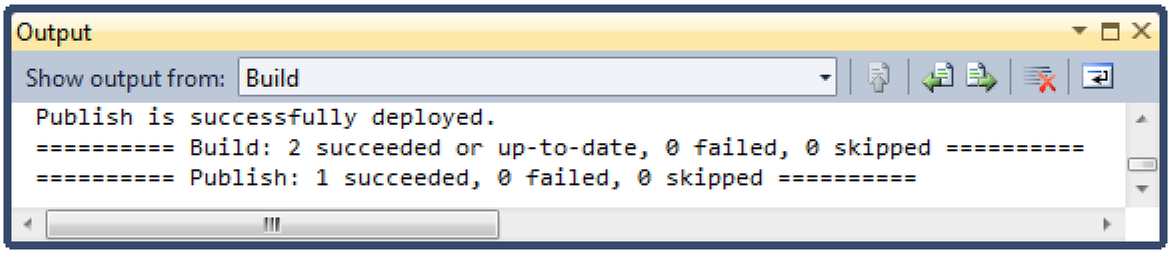
Deploying to the Test Environment

You are now ready to deploy to the test environment.

In the **Solution Configurations** drop-down box, select the **Test** build configuration, and in the **Publish** profile drop-down box, select **Test**. Click **Publish Web**. (If you can't do this in the toolbars because you have have customized toolbars, see the Deploying a Code-Only Update tutorial for an alternative method.)


ContosoUniversity - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze
Test
Install Web Components
Publish: Test

Visual Studio deploys the updated application and reports success in the **Output** window.



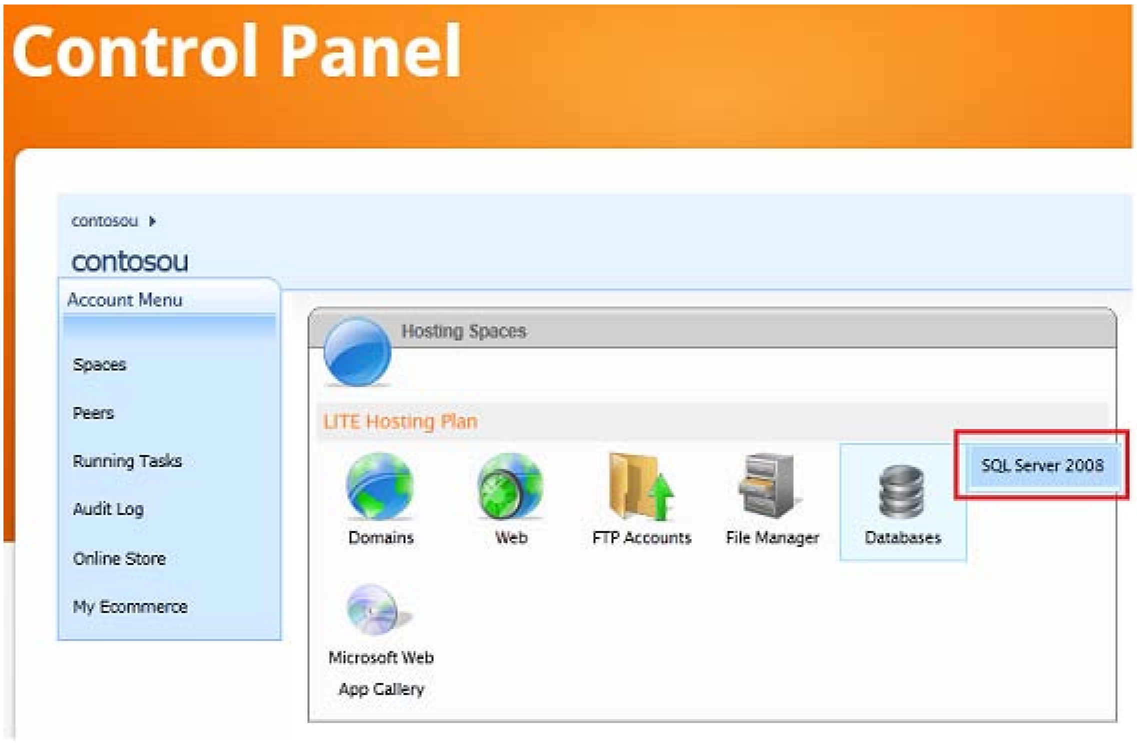
```
Publish is successfully deployed.
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

You can now open a browser and run http://localhost/ContosoUniversity to test the application. Run the **Add Students** page, add a new student, then view the new student in the **Students** page. This verifies that the School database was deployed and that that you have read and write access to it. 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.

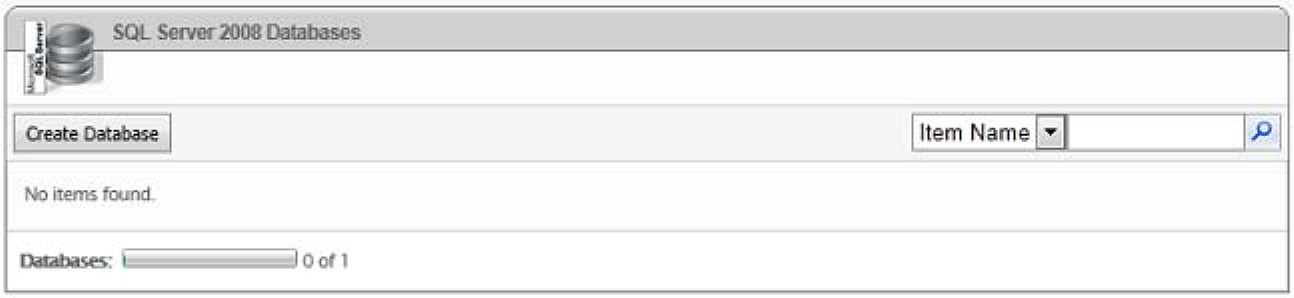
Creating a SQL Server Database for the Production Environment

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**.



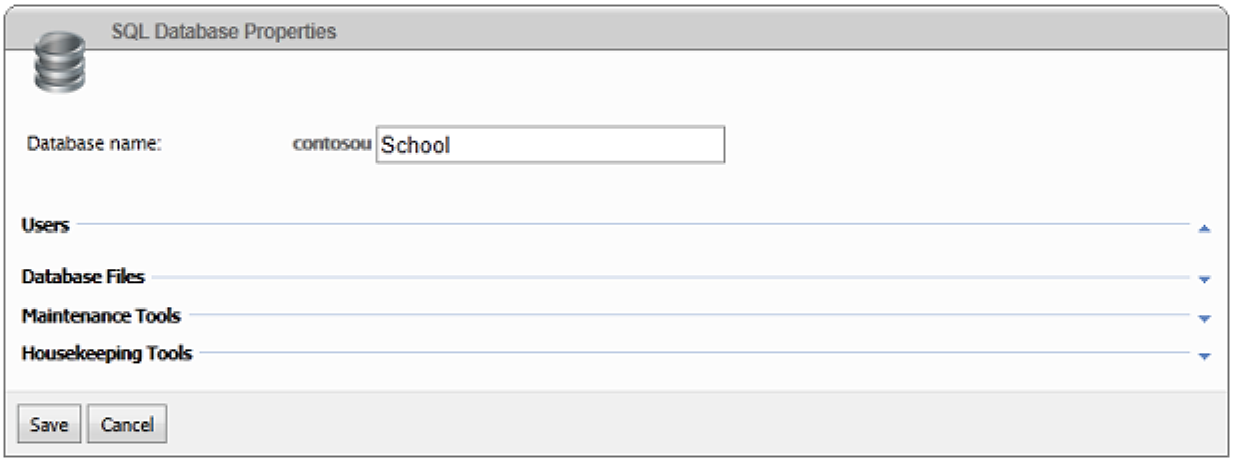
In the **SQL Server 2008** page, click **Create Database**.



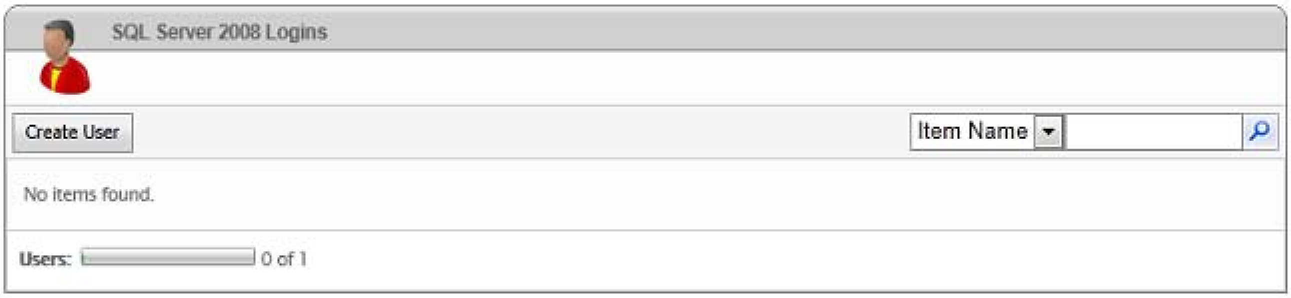
No items found.

Databases: 0 of 1

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

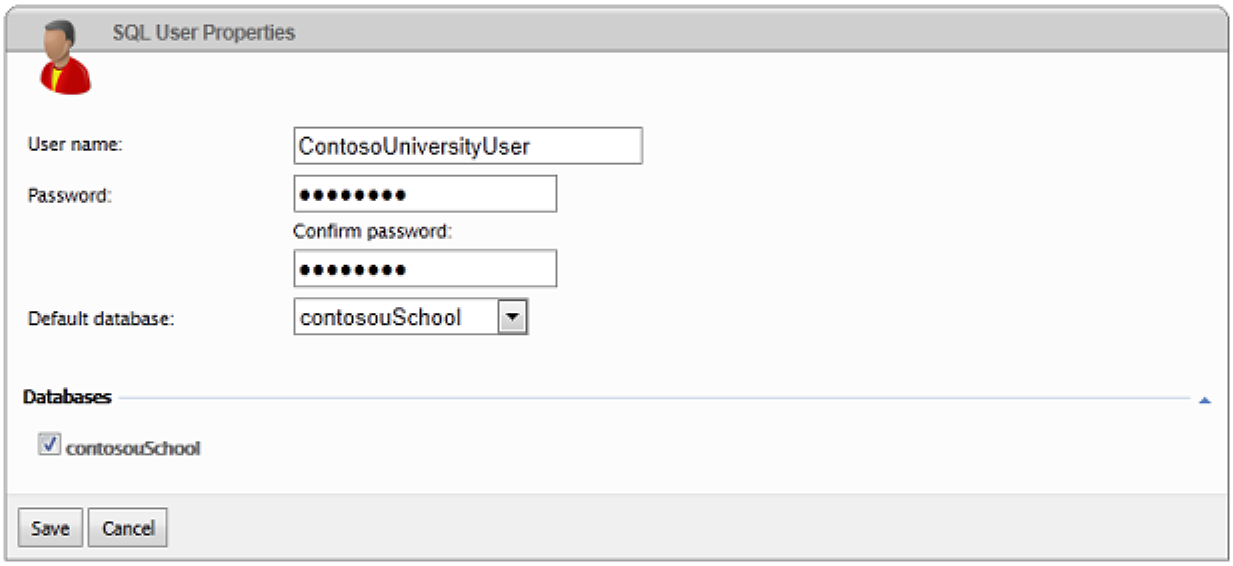


On the same page, click **Create User**. On Cytanium's servers, rather than using integrated Windows security and letting the application pool identity open your database, you'll create a user that has authority to open your database. You'll add the user's credentials to the connection strings that go 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. This tutorial will assume you have entered Pas$w0rd.
- Select **contosouSchool** as the default database.
- Select the **contosouSchool** check box.


SQL User Properties

User name: ContosoUniversityUser

Password: ••••••••

Confirm password: ••••••••

Default database: contosouSchool

Databases

✓ contosouSchool

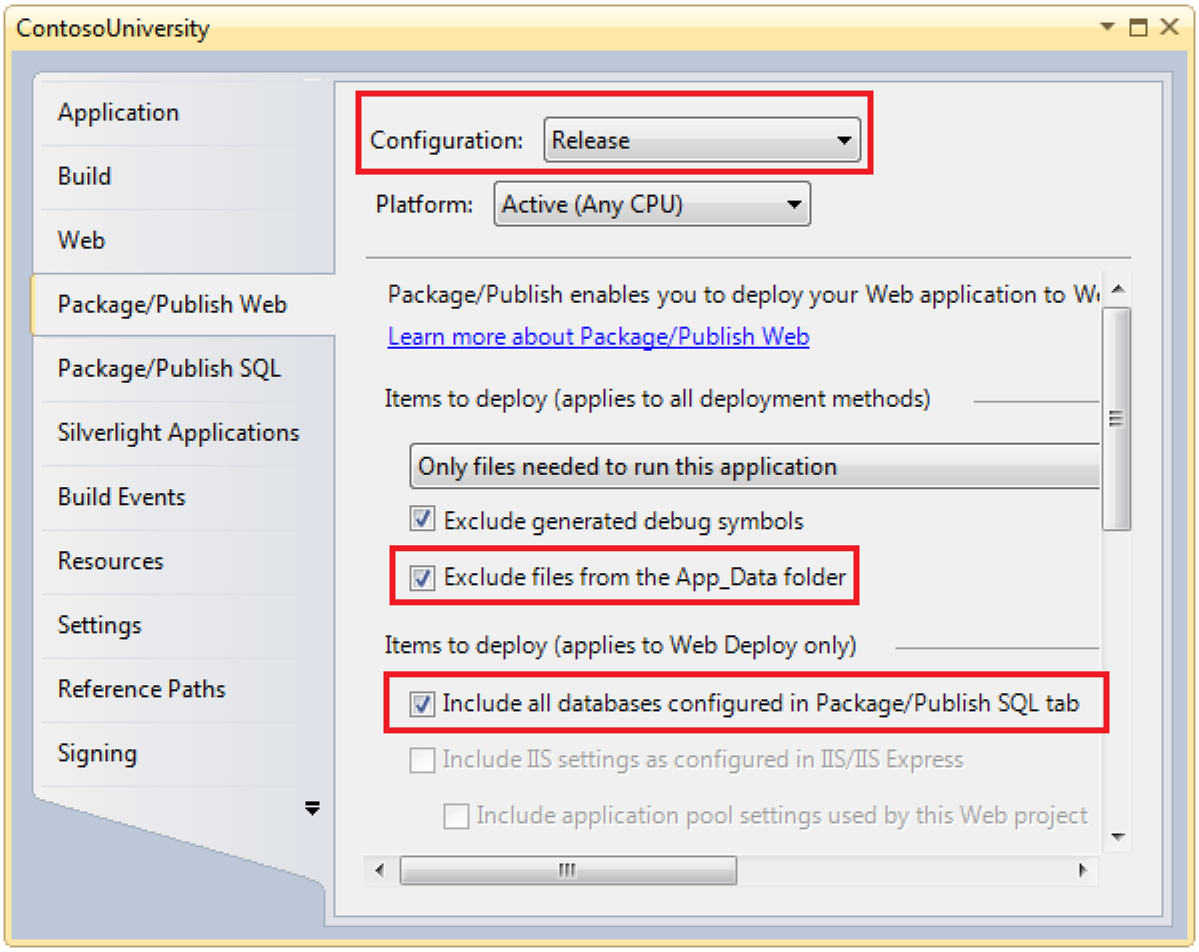
Configuring Database Deployment for the Production Environment

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

Open the **Project Properties** window, select the **Package/Publish Web** tab, and select **Active (Release)** or **Release** in the **Configuration** drop-down list.

Select **Exclude files from the App_Data folder**. Database deployment to SQL Server does not involve copying files in the *App_Data* folder, it involves generating and running scripts in the destination database.

Make sure that **Include all databases configured in Package/Publish SQL tab** is selected.



ContosoUniversity

Application
Build
Web
Package/Publish Web
Package/Publish SQL
Silverlight Applications
Build Events
Resources
Settings
Reference Paths
Signing

Configuration: Release

Platform: Active (Any CPU)

Package/Publish enables you to deploy your Web application to W

Learn more about Package/Publish Web

Items to deploy (applies to all deployment methods)

Only files needed to run this application

☑ Exclude generated debug symbols

☑ Exclude files from the App_Data folder

Items to deploy (applies to Web Deploy only)

☑ Include all databases configured in Package/Publish SQL tab

☐ Include IIS settings as configured in IIS/IIS Express

☐ Include application pool settings used by this Web project

Select the **Package/Publish SQL** tab. With the build configuration still set to **Release**, click **Import from Web.config** as you did earlier for the Test build configuration. The same two rows are added to the **Database Entries** table, one for each of the existing SQL Server Compact databases.

When you configure deployment settings for each database, the key difference between what you are doing for production and test environments is that 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 source 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 is this:

```
Data Source=vserver01.cytanium.com;Initial Catalog=contosouSchool;User
Id=ContosoUniversityUser;Password=Pas$w0rd;
```

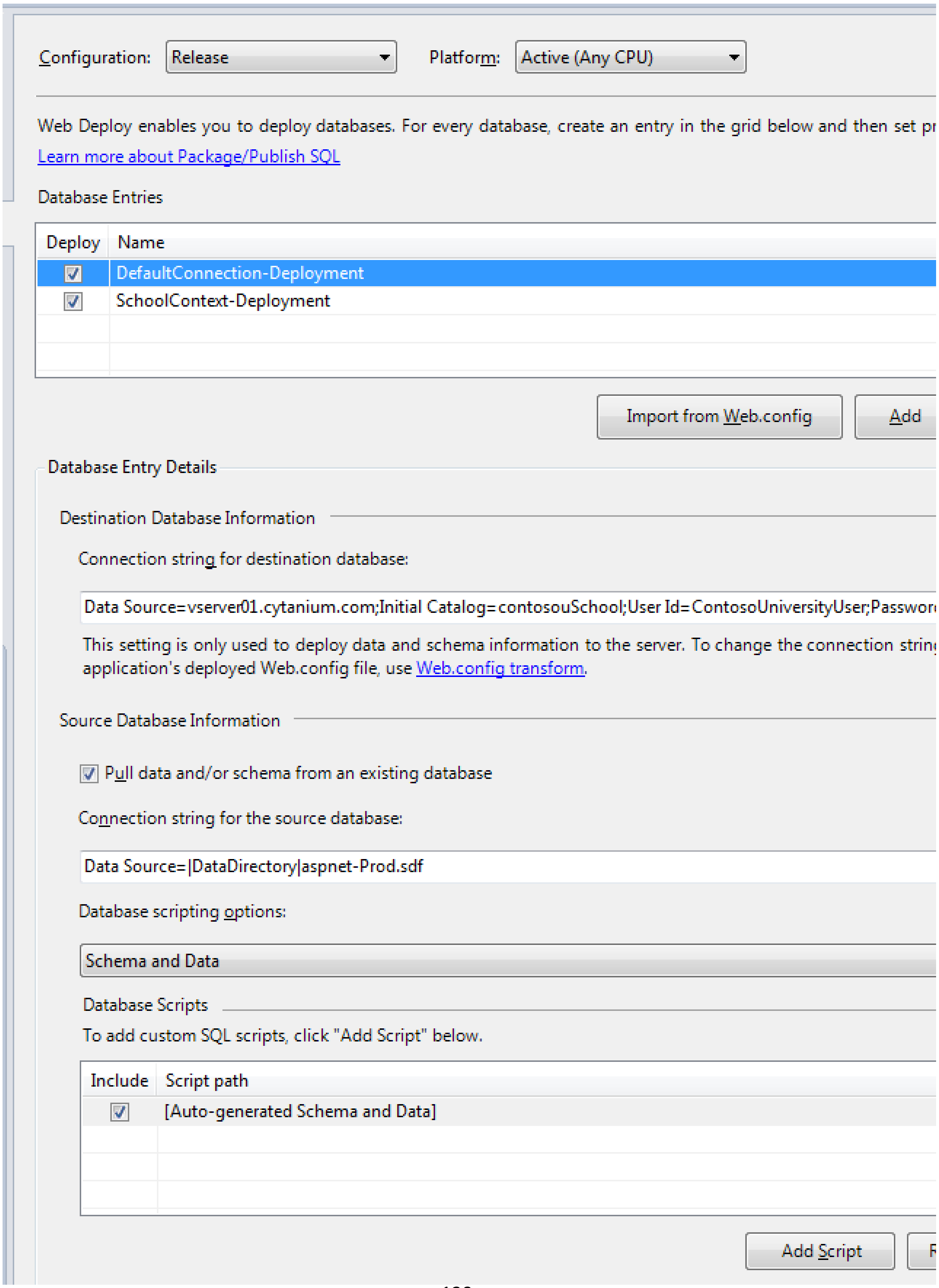
Copy this connection string into **Connection string for destination database** in the **Package/Publish SQL** tab.

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

The **Connection string for the source database** value is extracted from the *Web.config* file and points to the development SQL Server Compact database in the App_Data folder. Change "aspnet-Dev.sdf" to "aspnet-Prod.sdf".

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

When you're done, the settings for the **DefaultConnection-Deployment** row in **Database Entries** look like this:



Configuration: Release

Platform: Active (Any CPU)

Web Deploy enables you to deploy databases. For every database, create an entry in the grid below and then set pr

Learn more about Package/Publish SQL

Database Entries

Deploy	Name
✓	DefaultConnection-Deployment
✓	SchoolContext-Deployment

Database Entry Details

Destination Database Information

Connection string for destination database:

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

This setting is only used to deploy data and schema information to the server. To change the connection strin application's deployed Web.config file, use Web.config transform.

Source Database Information

✓ Pull data and/or schema from an existing database

Connection string for the source database:

Data Source=|DataDirectory|aspnet-Prod.sdf

Database scripting options:

Schema and Data

Database Scripts

To add custom SQL scripts, click "Add Script" below.

Include	Script path
✓	[Auto-generated Schema and Data]

Configuring Deployment Settings for the School Database

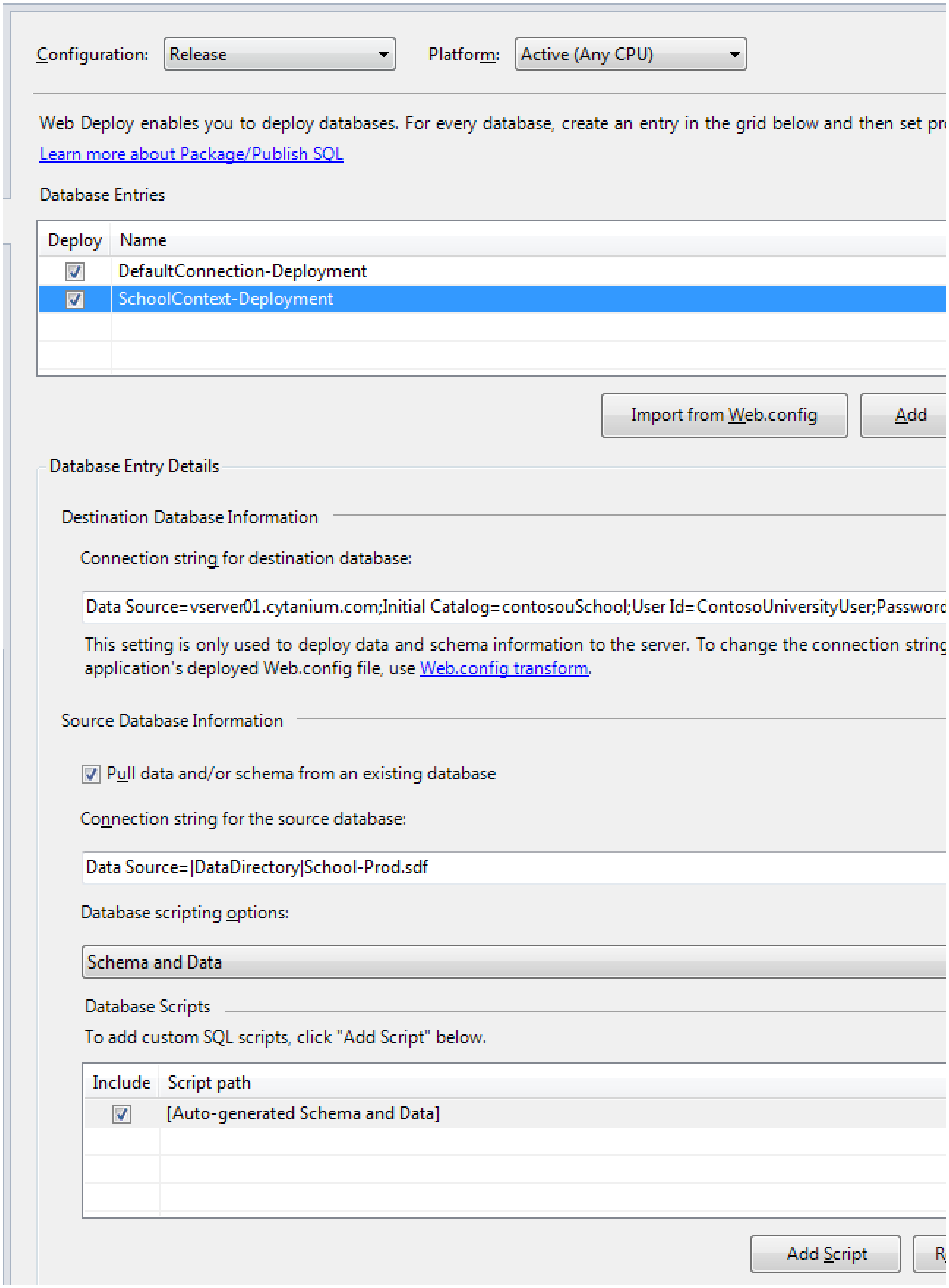
Next, select the **SchoolContext-Deployment** row in the **Database Entries** table in order to configure the School database settings.

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 selected. The **Connection string for the source database** value is extracted from the *Web.config* file and points to the development database. Change "School-Dev.sdf" to "School-Prod.sdf".

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

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


Configuration: Release

Platform: Active (Any CPU)

Web Deploy enables you to deploy databases. For every database, create an entry in the grid below and then set pr

Learn more about Package/Publish SQL

Database Entries

Deploy	Name
✓	DefaultConnection-Deployment
✓	SchoolContext-Deployment

Database Entry Details

Destination Database Information

Connection string for destination database:

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

This setting is only used to deploy data and schema information to the server. To change the connection strinc application's deployed Web.config file, use Web.config transform.

Source Database Information

✓ Pull data and/or schema from an existing database

Connection string for the source database:

Data Source=|DataDirectory|School-Prod.sdf

Database scripting options:

Schema and Data

Database Scripts

To add custom SQL scripts, click "Add Script" below.

Include	Script path
✓	[Auto-generated Schema and Data]

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.onfig* file 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.Release.config* and replace the `connectionStrings` element with the `connectionStrings` element shown below:

```
<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=Pas$w0rd;MultipleActiveResultSets=True"
         providerName="System.Data.SqlClient"
         xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/>
<add name="SchoolContext"
         connectionString="Data Source=vserver01.cytanium.com;Initial
Catalog=contosouSchool;User
Id=ContosoUniversityUser;Password=Pas$w0rd;MultipleActiveResultSets=True"
         providerName="System.Data.SqlClient"
         xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/>
</connectionStrings>
<!-- appSettings element, comments, and system.web element -->
</configuration>
```

You will 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 neither necessary nor practical to encrypt the connection strings.

Deploying to the Production Environment

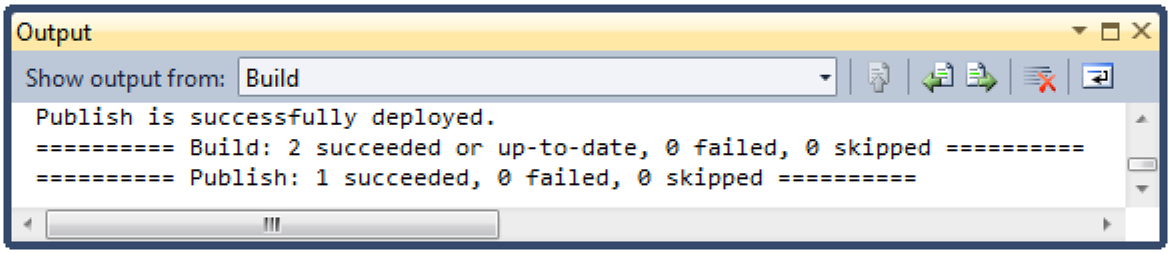
Now you're ready to deploy to production, which will read the SQL Server Compact databases in your project's *App_Data* folder and re-create all of their tables and data in the production SQL Server database.

Before you deploy, upload *app_offline.htm*, as shown in the Deploying to the Production Environment tutorial.

In a real application in which the database was being updated in production, you would then 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 would ensure that the data you're deploying to the new SQL Server database includes the latest updates made by your production website.

In the **Solution Configurations** drop-down box, select **Release**, and in the **Publish** profile drop-down box select **Production**. Click **Publish Web**. (If you can't do this in the toolbars because you have have customized toolbars, see the [Deploying a Code-Only Update]() tutorial for an alternative method.)

Visual Studio deploys the updated application and reports success in the **Output** window.



```
Publish is successfully deployed.
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

Before you test the site, use the **File Manager** in the Cytanium control panel to delete *app_offline.htm*. 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. Run the **Add Students** page, add a new student, then view the new student in the **Students** page. This verifies that the School database was deployed and that you have read and write access to it. Select the **Update Credits** page (which requires you to log in) to verify that the membership database was deployed and you have access to it.

Switching to SQL Server Express 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.

The simplest way to do that is to let Code First and the membership system create both new development databases for you:

- Change the connection strings to specify new SQL Express databases.
- Run the application, and Code First automatically creates and seeds the application database.

- Click Logon and then Register to register whatever user accounts you need for testing, and the ASP.NET membership system automatically creates and iniitializes the membership database.

However, if you have a large number of test user accounts and you don't want to manually re-enter them, you can use the deployment process to migrate that data from SQL Server Compact to SQL Server Express. If you prefer to do that, follow the instructions that start in the next section. If you do not need to see how to do that, skip the following sections until you get to the **Updating Connection Strings in the Web.config file** section.

Creating a Development Membership Database

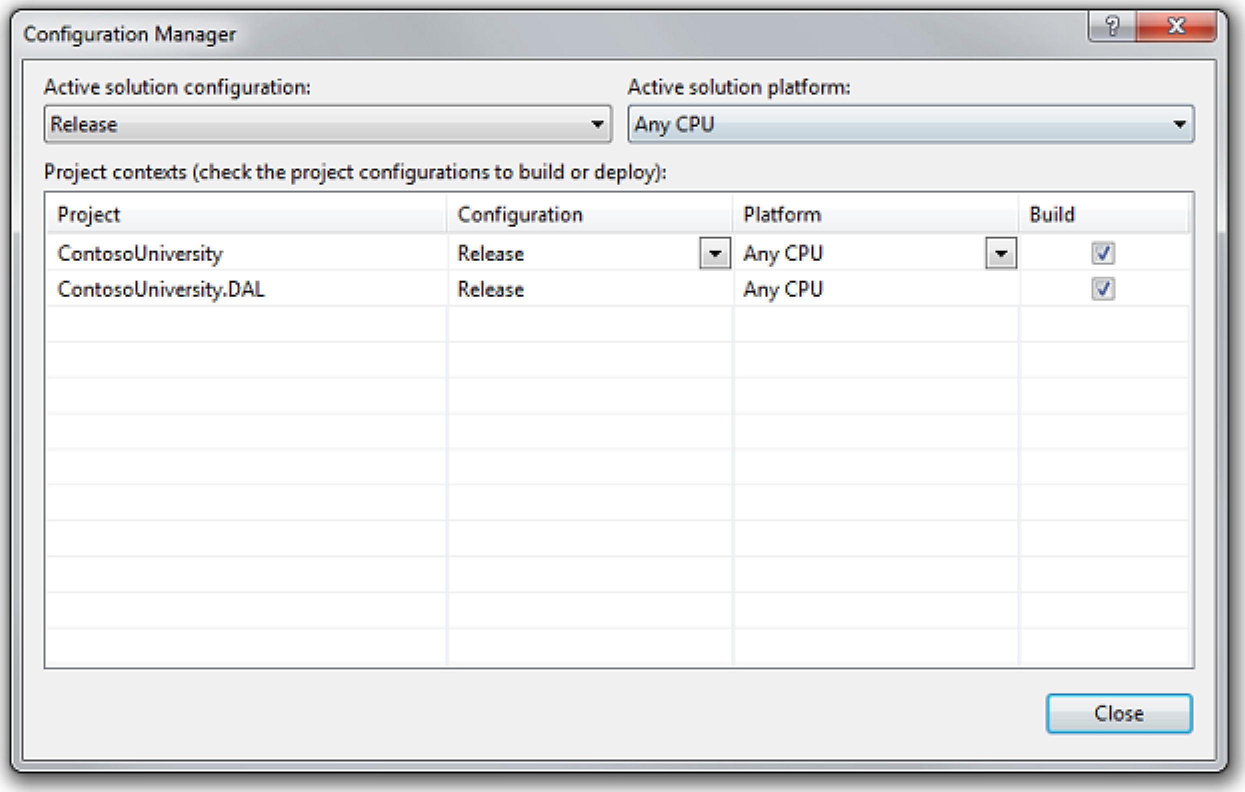
From the **View** menu, select **Server Explorer**, then right-click **Data Connections** and select **Create New SQL Server Database**.

In the **Create New SQL Server Database** dialog box, enter ".\SQLExpress" in the **Server name** box and "aspnetDev" in the **New database name** box. Click **OK**.

Configuring Database Deployment

You'll need to configure deployment settings in order to set up deployment to the new development membership database, but you don't want to change your existing settings for the test environment. To create new settings that you can use without affecting the existing ones, create a new build configuration named "MigrateToSQLExpress" based on the existing Test build configuration.

From the Visual Studio **Build** menu, select **Configuration Manager** to display the **Configuration Manager** dialog box.


Configuration Manager

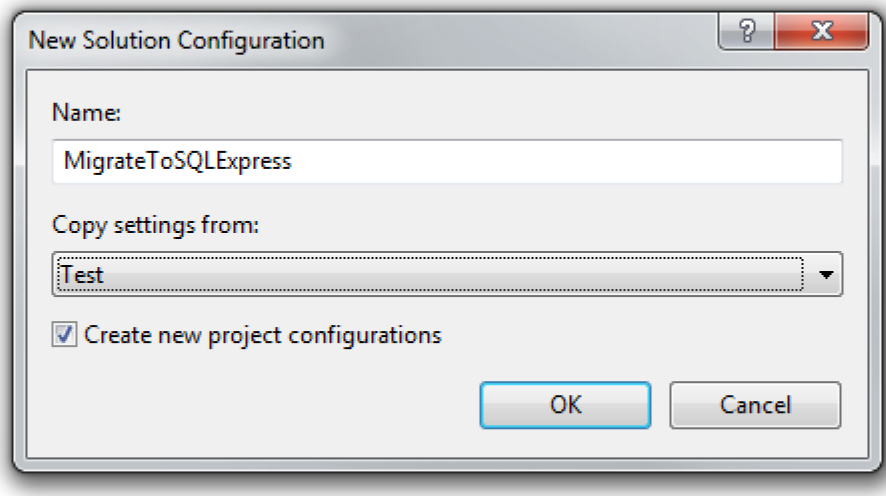
Active solution configuration: Release

Active solution platform: Any CPU

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build
ContosoUniversity	Release	Any CPU	✓
ContosoUniversity.DAL	Release	Any CPU	✓

In the **Active solution configuration** box, select **New**. When the **New Solution Configuration** dialog box appears, enter "MigrateToSQLExpress" as the name of the new build configuration, and copy settings from **Test**. Leave **Create new project configurations** selected, and click **OK**.


New Solution Configuration

Name: MigrateToSQLExpress

Copy settings from: Test

✓ Create new project configurations

Open the **Project Properties** window and select the **Package/Publish Web** tab. With **Active (MigrateToSQLExpress)** in the **Configuration** drop-down list, select **Exclude files from the App_Data folder** and make sure **Include all databases configured in Package/Publish SQL tab** is selected.

Select the **Package/Publish SQL** tab. With the build configuration set to **Active(MigrateToSQLExpress)**, click **Import from Web.config**.

In the **Database Entries** table, clear the check box next to **SchoolContext-Deployment** (since you don't need to deploy to the School database), then select **DefaultConnection-Deployment** to configure membership database settings.

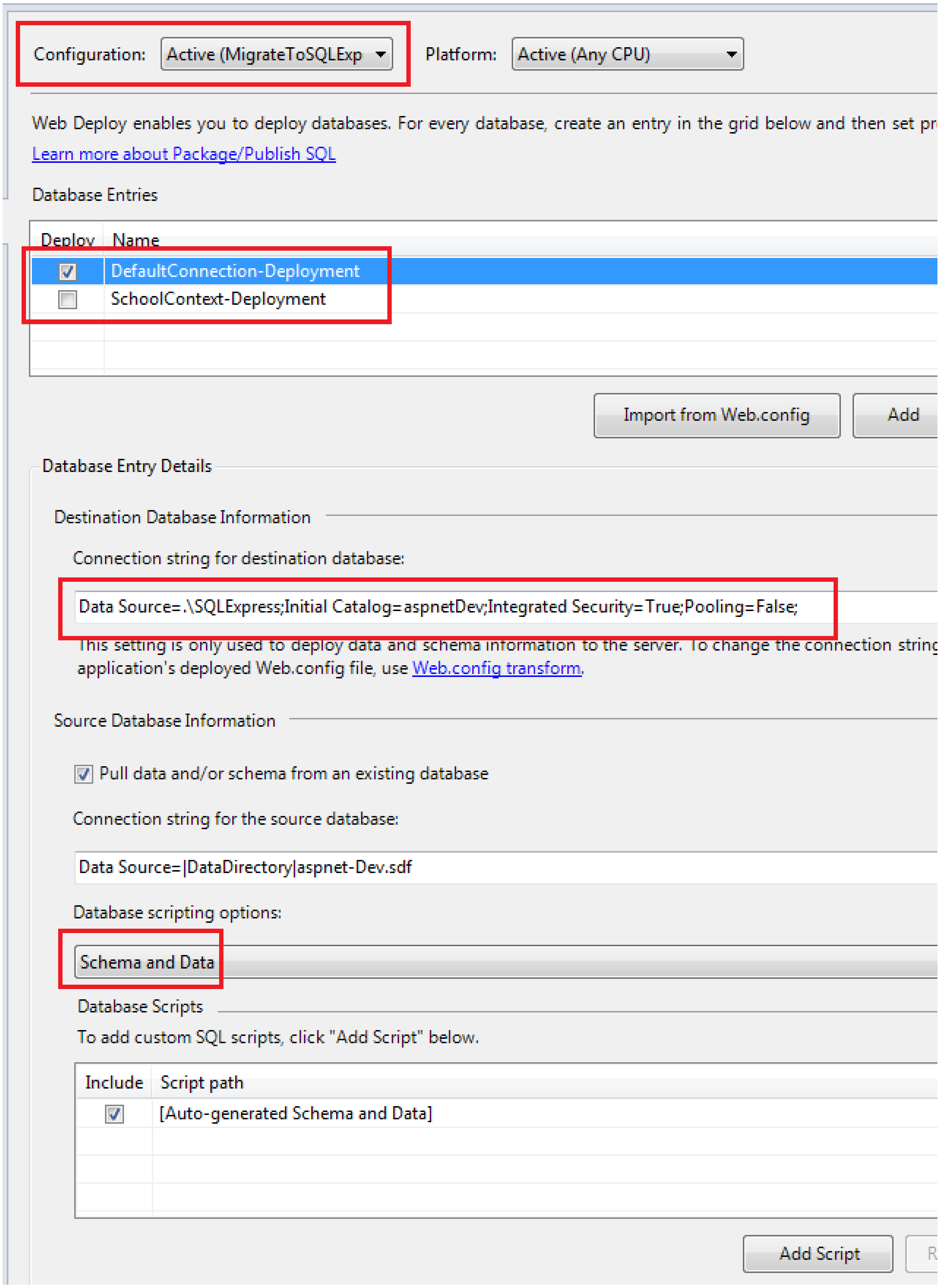
In **Connection string for destination database**, enter the connection string shown below. This points to the new SQL Server Express membership database that you just created for your development environment.

```
Data Source=.\SQLExpress;Initial Catalog=aspnetDev;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 your development SQL Server Compact membership database.

Change **Database scripting options** from **Schema Only** to **Schema and Data**.


Configuration: Active (MigrateToSQLExp

Platform: Active (Any CPU)

Web Deploy enables you to deploy databases. For every database, create an entry in the grid below and then set pr

Learn more about Package/Publish SQL

Database Entries

Deploy	Name
✓	DefaultConnection-Deployment
	SchoolContext-Deployment

Database Entry Details

Destination Database Information

Connection string for destination database:

```
Data Source=.\SQLExpress;Initial Catalog=aspnetDev;Integrated Security=True;Pooling=False;
```

This setting is only used to deploy data and schema information to the server. To change the connection strin
application's deployed Web.config file, use Web.config transform.

Source Database Information

[✓] Pull data and/or schema from an existing database

Connection string for the source database:

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

Database scripting options:

Schema and Data

Database Scripts

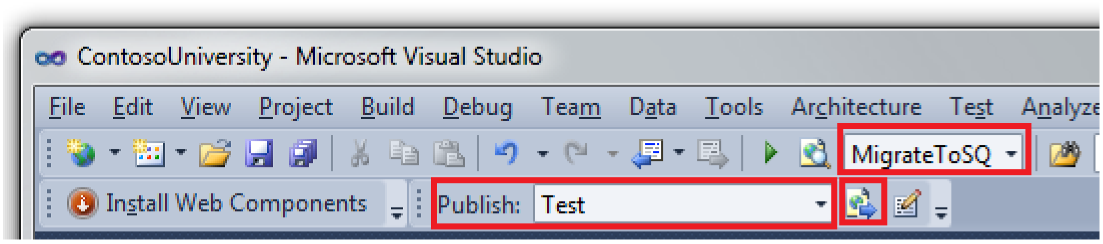
To add custom SQL scripts, click "Add Script" below.

Include	Script path
✓	[Auto-generated Schema and Data]

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

Deploying to the Test Environment

In the **Solution Configurations** drop-down box, select **MigrateToSQLExpress** build configuration. In the **Publish** profile drop-down box, select **Test**. Click **Publish Web**. (If you can't do this in the toolbars because you have have customized toolbars, see the [Deploying a Code-Only Update]() tutorial for an alternative method.)


ContosoUniversity - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze
MigrateToSQ
Install Web Components
Publish: Test

Visual Studio deploys the updated application and reports success in the **Output** window.

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=localhost\SQLExpress;Initial Catalog=aspnetDev;Integrated
Security=True;Pooling=False;MultipleActiveResultSets=True;"
providerName="System.Data.SqlClient" />
<add name="SchoolContext" connectionString="Data
Source=localhost\SQLExpress;Initial Catalog=SchoolDev;Integrated
Security=True;Pooling=False;MultipleActiveResultSets=True;"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

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, 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.

If you followed the instructions above to use the deployment process to migrate your user accounts, 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.

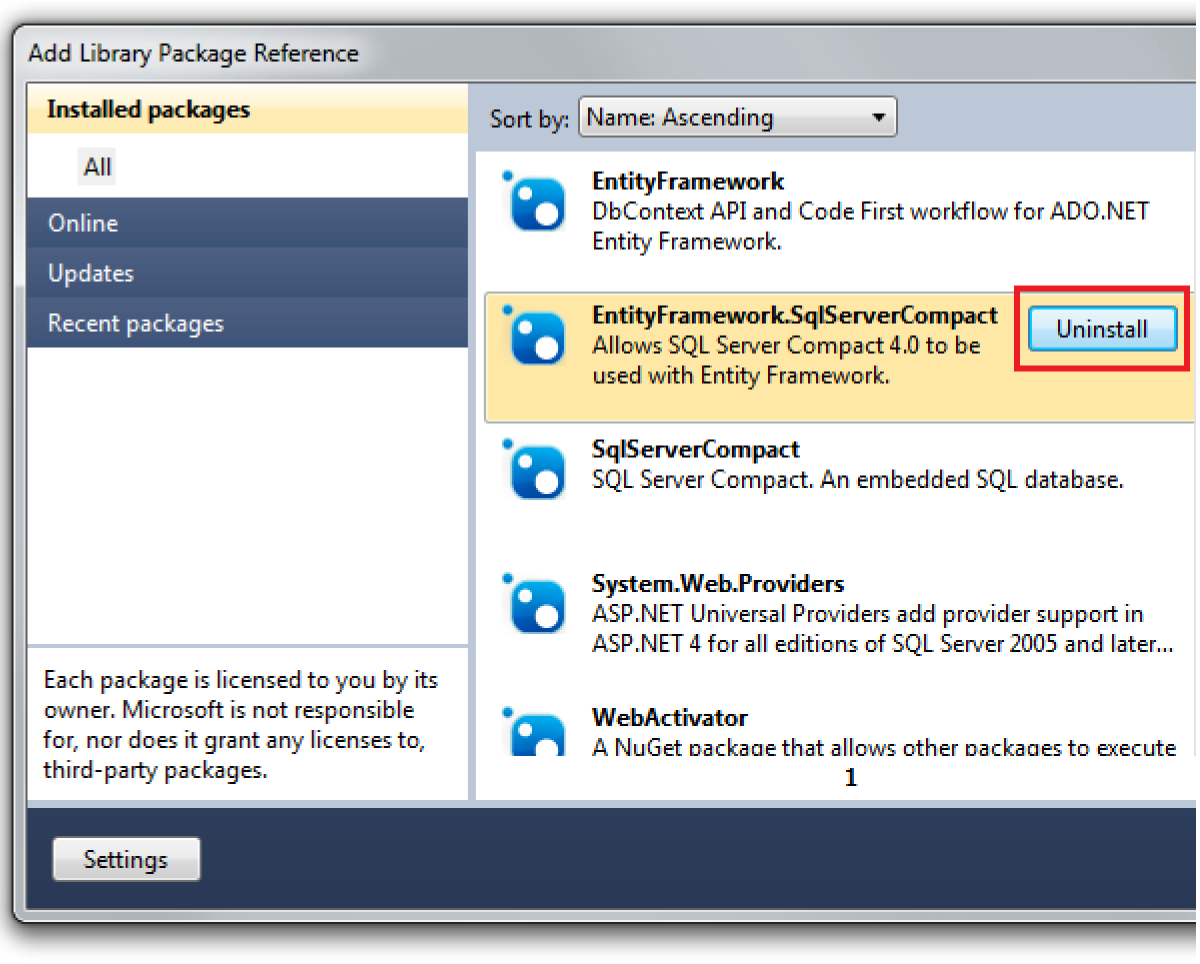
Cleaning Up SQL Server Compact Files

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 ContosoUniversity project and select **Add Library Package Reference**.

In the **Add Library Package Reference** dialog box, select the **EntityFramework.SqlServerCompact** package and click **Uninstall**.



EntityFramework
DbContext API and Code First workflow for ADO.NET Entity Framework.

EntityFramework.SqlServerCompact
Allows SQL Server Compact 4.0 to be used with Entity Framework.

SqlServerCompact
SQL Server Compact. An embedded SQL database.

System.Web.Providers
ASP.NET Universal Providers add provider support in ASP.NET 4 for all editions of SQL Server 2005 and later...

WebActivator
A NuGet package that allows other packages to execute

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

When uninstallation is complete, do the same for the and **SqlServerCompact** package. (The packages must be uninstalled in this order because the **EntityFramework.SqlServerCompact** package depends on the **SqlServerCompact** package.)

Follow the same procedure to remove the same two packages from the ContosoUniversity.DAL project.

Finally, you can delete the MigrateToSqlExpress Build configuration, since you created it only for the migration. Select **Configuration Manager** from the **Build** menu, and then in the **Configuration Manager** dialog box select **Edit** in the **Active solution configuration** drop-down box. In the **Edit Solution Configurations** dialog box, select **MigrateToSQLExpress** and click **Remove**.

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. The process differs from what you saw in the Deploying a Database Update tutorial, which showed how to deploy a database change to a SQL Server Compact database.

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; }
```

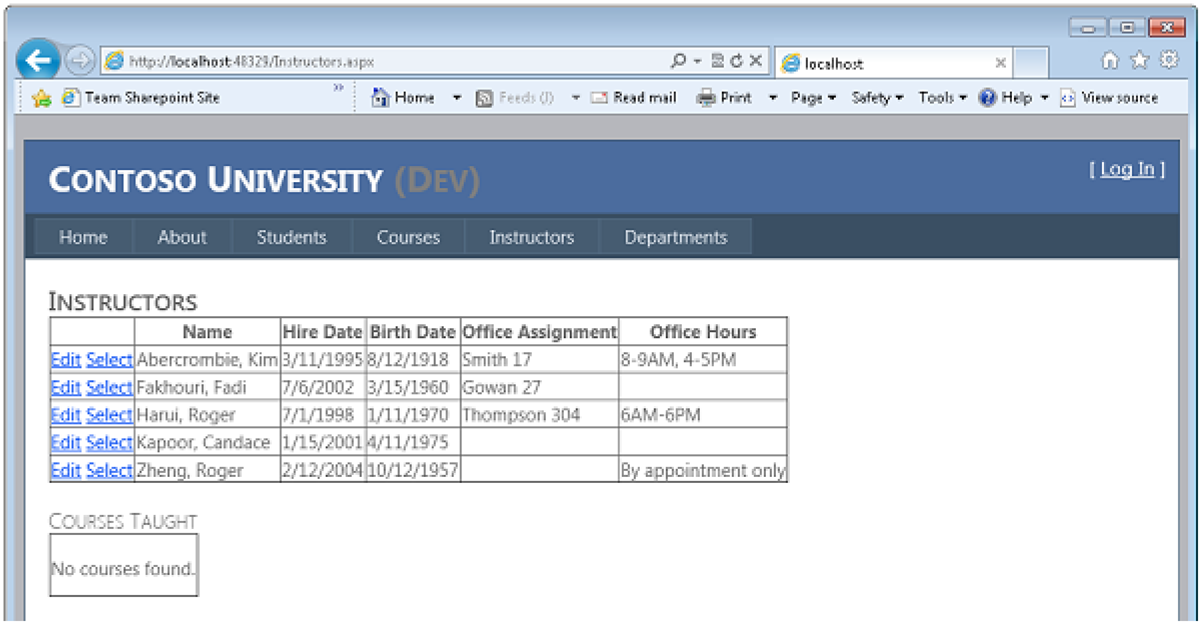
Update the initializer class so that it seeds the new column with test data. Open *SchoolInitializer.cs* and replace the code block that begins `var instructors = new List<Instructor>` with the following code block which includes the new column:

```
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" },
    new Instructor { FirstMidName = "Fadi",    LastName = "Fakhouri",
HireDate = DateTime.Parse("2002-07-06"), BirthDate = DateTime.Parse("1960-03-
15") },
    new Instructor { FirstMidName = "Roger",   LastName = "Harui",
HireDate = DateTime.Parse("1998-07-01"), BirthDate = DateTime.Parse("1970-01-
11"), OfficeHours = "6AM-6PM" },
    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:

```
<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>
```

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.



INSTRUCTORS

	Name	Hire Date	Birth Date	Office Assignment	Office Hours
Edit Select	Abercrombie, Kim	3/11/1995	8/12/1918	Smith 17	8-9AM, 4-5PM
Edit Select	Fakhouri, Fadi	7/6/2002	3/15/1960	Gowan 27	
Edit Select	Harui, Roger	7/1/1998	1/11/1970	Thompson 304	6AM-6PM
Edit Select	Kapoor, Candace	1/15/2001	4/11/1975		
Edit Select	Zheng, Roger	2/12/2004	10/12/1957		By appointment only

COURSES TAUGHT

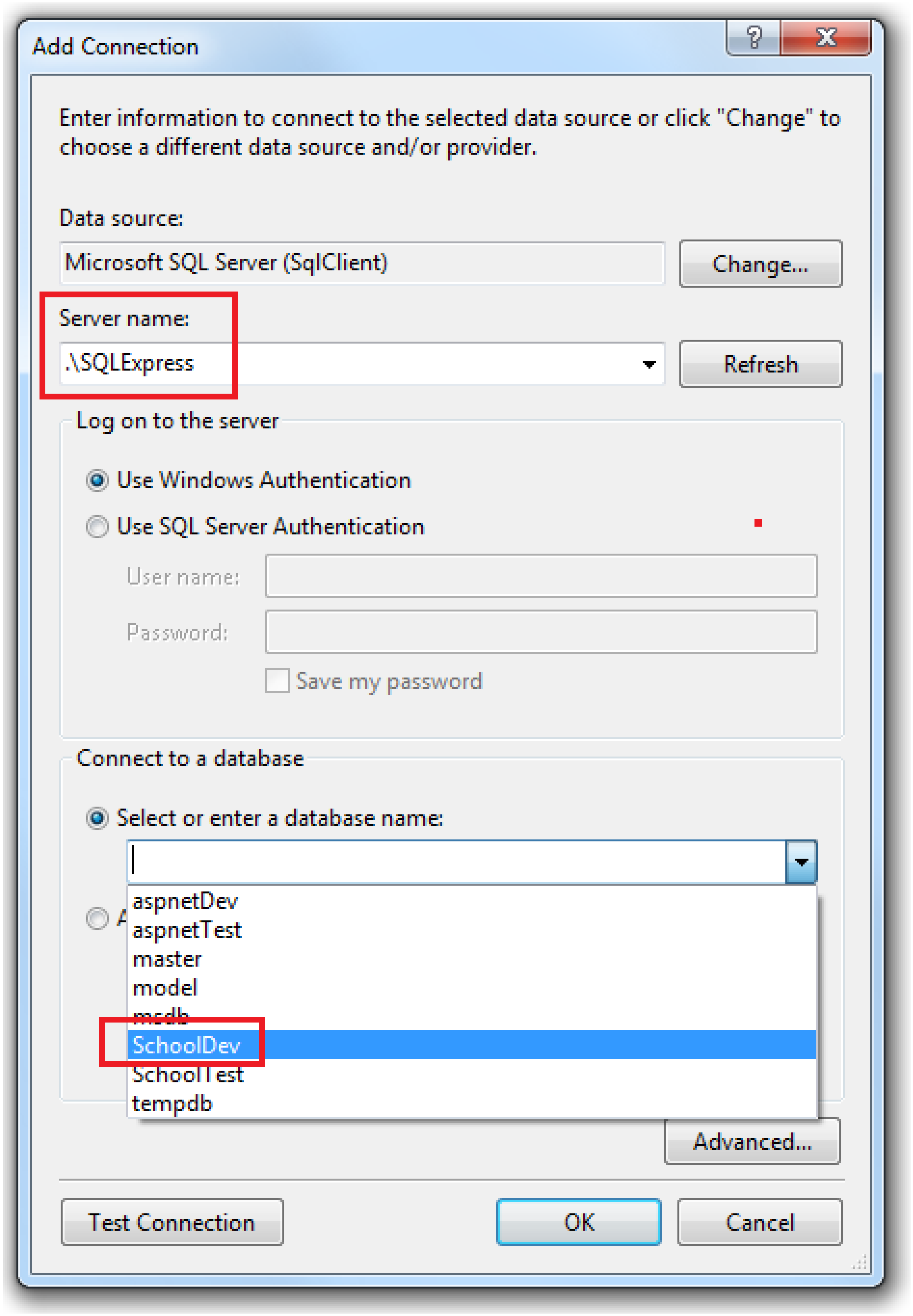
No courses found.

Preparing a SQL Script for the Database Update

As with the SQL Server Compact database change that you made and deployed in an earlier tutorial, you want to deploy to the test environment using the same method that you will use later to deploy to production. In this case, that means you need to create a SQL script to make the database change, because you can't update full SQL Server databases by copying files the way you can with SQL Server Compact databases. As was pointed out in the Deploying a Database Update tutorial, tools are available for SQL Server that will compare databases and automatically generate database change scripts. For this tutorial, the script you need will be provided for you.

To see what Code First has done to the database, you can use **Server Explorer** again. You will need to start by adding a connection to the new SchoolDev database.

In **Server Explorer**, right-click **Data Connections** and select **Add Connection**. In the Add Connection dialog box, enter ".\SQLExpress" as the **Server Name**. You can then expand **Select or enter a database name** and select the SchoolDev database.



Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

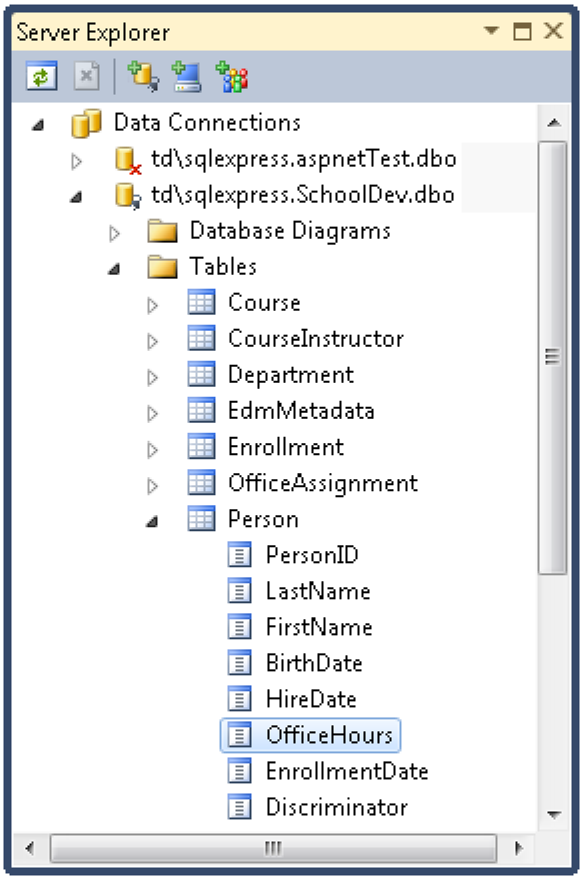
Data source:
Microsoft SQL Server (SqlClient)

Server name:
.\SQLExpress

Log on to the server
(●) Use Windows Authentication
( ) Use SQL Server Authentication
User name:
Password:
[ ] Save my password

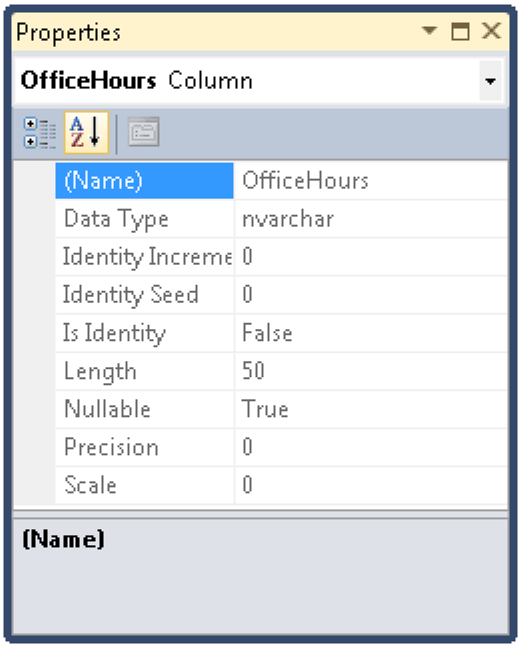
Connect to a database
(●) Select or enter a database name:
aspnetDev
aspnetTest
master
model
msdb
SchoolDev
SchoolTest
tempdb

expand **SchoolDev**, expand **Tables**, and expand **Person**. Expand **Columns** to see the new **OfficeHours** column.


Server Explorer

- Data Connections
  - td\sqlexpress.aspnetTest.dbo
  - td\sqlexpress.SchoolDev.dbo
    - Database Diagrams
    - Tables
      - Course
      - CourseInstructor
      - Department
      - EdmMetadata
      - Enrollment
      - OfficeAssignment
      - Person
        - PersonID
        - LastName
        - FirstName
        - BirthDate
        - HireDate
        - OfficeHours
        - EnrollmentDate
        - Discriminator

In the **Properties** window, you can see that the data type of this column is `nvarchar` and the length is 50.


Properties

OfficeHours Column

(Name)	OfficeHours
Data Type	nvarchar
Identity Increme	0
Identity Seed	0
Is Identity	False
Length	50
Nullable	True
Precision	0
Scale	0

(Name)

In **Server Explorer**, right-click **SchoolDev** and select **Close Connection**.

The following script adds this new column to the `Person` table. Copy this script into a new *.sql* file named *AddOfficeHoursColumn.sql* in the *SolutionFiles* solution folder that you created in the Deploying a Database Update tutorial.

```
BEGIN TRANSACTION
SET QUOTED_IDENTIFIER ON
SET ARITHABORT ON
SET NUMERIC_ROUNDABORT OFF
SET CONCAT_NULL_YIELDS_NULL ON
SET ANSI_NULLS ON
SET ANSI_PADDING ON
SET ANSI_WARNINGS ON
COMMIT
BEGIN TRANSACTION
GO
CREATE TABLE dbo.Tmp_Person
        (
        PersonID int NOT NULL IDENTITY (15, 1),
        LastName nvarchar(50) NOT NULL,
        FirstName nvarchar(50) NOT NULL,
        BirthDate datetime NULL,
        HireDate datetime NULL,
        OfficeHours nvarchar(50) NULL,
        EnrollmentDate datetime NULL,
        Discriminator nvarchar(128) NOT NULL
        )  ON [PRIMARY]
GO
ALTER TABLE dbo.Tmp_Person SET (LOCK_ESCALATION = TABLE)
GO
SET IDENTITY_INSERT dbo.Tmp_Person ON
GO
IF EXISTS(SELECT * FROM dbo.Person)
         EXEC('INSERT INTO dbo.Tmp_Person (PersonID, LastName, FirstName, 
BirthDate, HireDate, EnrollmentDate, Discriminator)
                SELECT PersonID, LastName, FirstName, BirthDate, HireDate, 
EnrollmentDate, Discriminator FROM dbo.Person WITH (HOLDLOCK TABLOCKX)')
GO
SET IDENTITY_INSERT dbo.Tmp_Person OFF
GO
ALTER TABLE dbo.CourseInstructor
        DROP CONSTRAINT Course_Instructors_Target
GO
ALTER TABLE dbo.Department
        DROP CONSTRAINT Department_Administrator
GO
ALTER TABLE dbo.Enrollment
        DROP CONSTRAINT Student_Enrollments
GO
ALTER TABLE dbo.OfficeAssignment
        DROP CONSTRAINT Instructor_OfficeAssignment
GO
DROP TABLE dbo.Person
GO
```

```
EXECUTE sp_rename N'dbo.Tmp_Person', N'Person', 'OBJECT'
GO
ALTER TABLE dbo.Person ADD CONSTRAINT
        PK__Person__000000000000002C PRIMARY KEY CLUSTERED
        (
        PersonID
        ) WITH( STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

GO
COMMIT
BEGIN TRANSACTION
GO
ALTER TABLE dbo.OfficeAssignment ADD CONSTRAINT
        Instructor_OfficeAssignment FOREIGN KEY
        (
        PersonID
        ) REFERENCES dbo.Person
        (
        PersonID
        ) ON UPDATE  NO ACTION
         ON DELETE  NO ACTION

GO
ALTER TABLE dbo.OfficeAssignment SET (LOCK_ESCALATION = TABLE)
GO
COMMIT
BEGIN TRANSACTION
GO
ALTER TABLE dbo.Enrollment ADD CONSTRAINT
        Student_Enrollments FOREIGN KEY
        (
        PersonID
        ) REFERENCES dbo.Person
        (
        PersonID
        ) ON UPDATE  NO ACTION
         ON DELETE  CASCADE

GO
ALTER TABLE dbo.Enrollment SET (LOCK_ESCALATION = TABLE)
GO
COMMIT
BEGIN TRANSACTION
GO
ALTER TABLE dbo.Department ADD CONSTRAINT
        Department_Administrator FOREIGN KEY
        (
        PersonID
        ) REFERENCES dbo.Person
        (
        PersonID
        ) ON UPDATE  NO ACTION
         ON DELETE  NO ACTION

GO
ALTER TABLE dbo.Department SET (LOCK_ESCALATION = TABLE)
```

```
GO
COMMIT
BEGIN TRANSACTION
GO
ALTER TABLE dbo.CourseInstructor ADD CONSTRAINT
        Course_Instructors_Target FOREIGN KEY
        (
        PersonID
        ) REFERENCES dbo.Person
        (
        PersonID
        ) ON UPDATE  NO ACTION
         ON DELETE  CASCADE

GO
ALTER TABLE dbo.CourseInstructor SET (LOCK_ESCALATION = TABLE)
GO
COMMIT
```

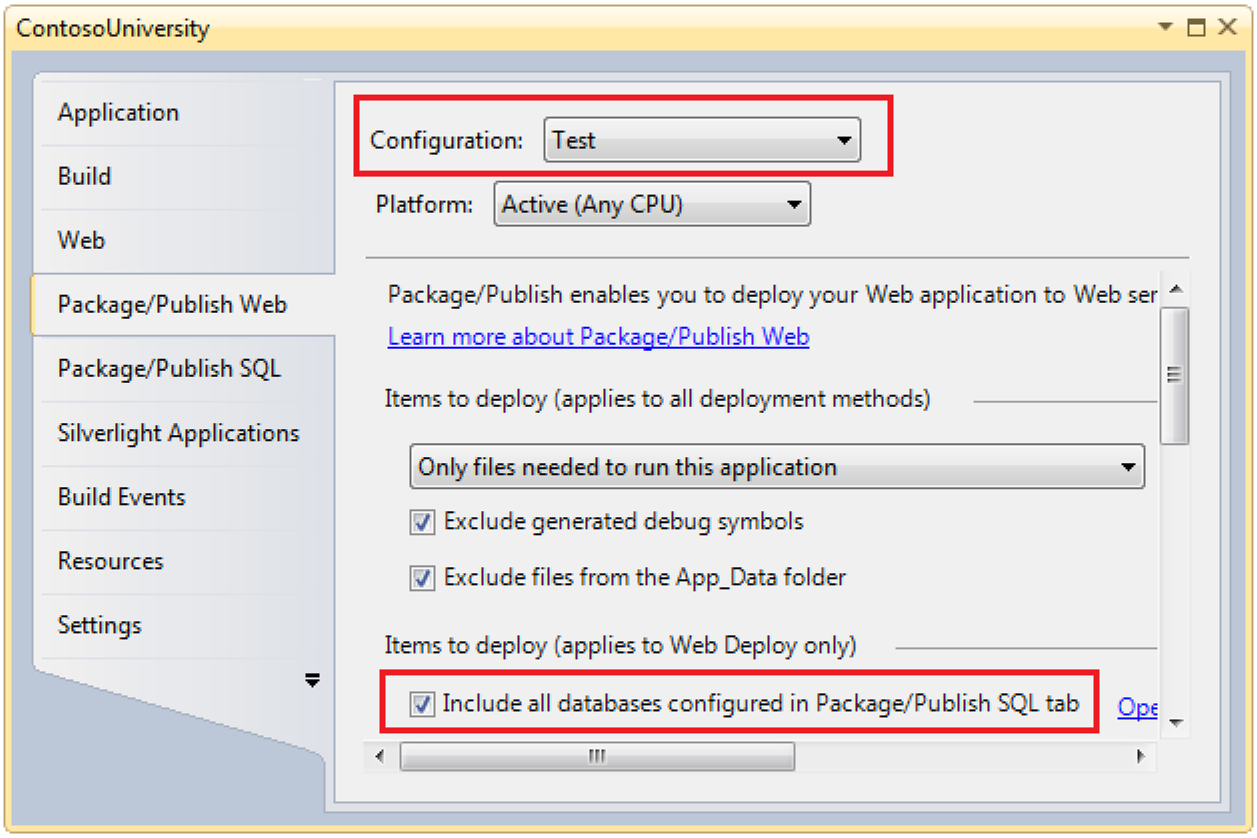
You could also create a script to update data in the `OfficeHours` column. But for this change you'll assume that your users will update the data online after the application change is deployed.

Deploying the Database Update to the Test Environment

For the test environment, you need to make sure that the schema creation scripts, data insertion scripts, and grant scripts that you ran for the first deployment to SQL Server do not run again. In their place you need to specify that this new script should run.

Open the **Project Properties** window and select the **Package/Publish Web** tab. Make sure that you have either **Active (Test)** or **Test** selected in the **Configuration** box.

Verify that **Include all databases configured in Package/Publish SQL tab** is selected.


ContosoUniversity

Configuration: Test

Platform: Active (Any CPU)

Package/Publish enables you to deploy your Web application to Web ser

Learn more about Package/Publish Web

Items to deploy (applies to all deployment methods)

Only files needed to run this application

☑ Exclude generated debug symbols

☑ Exclude files from the App_Data folder

Items to deploy (applies to Web Deploy only)

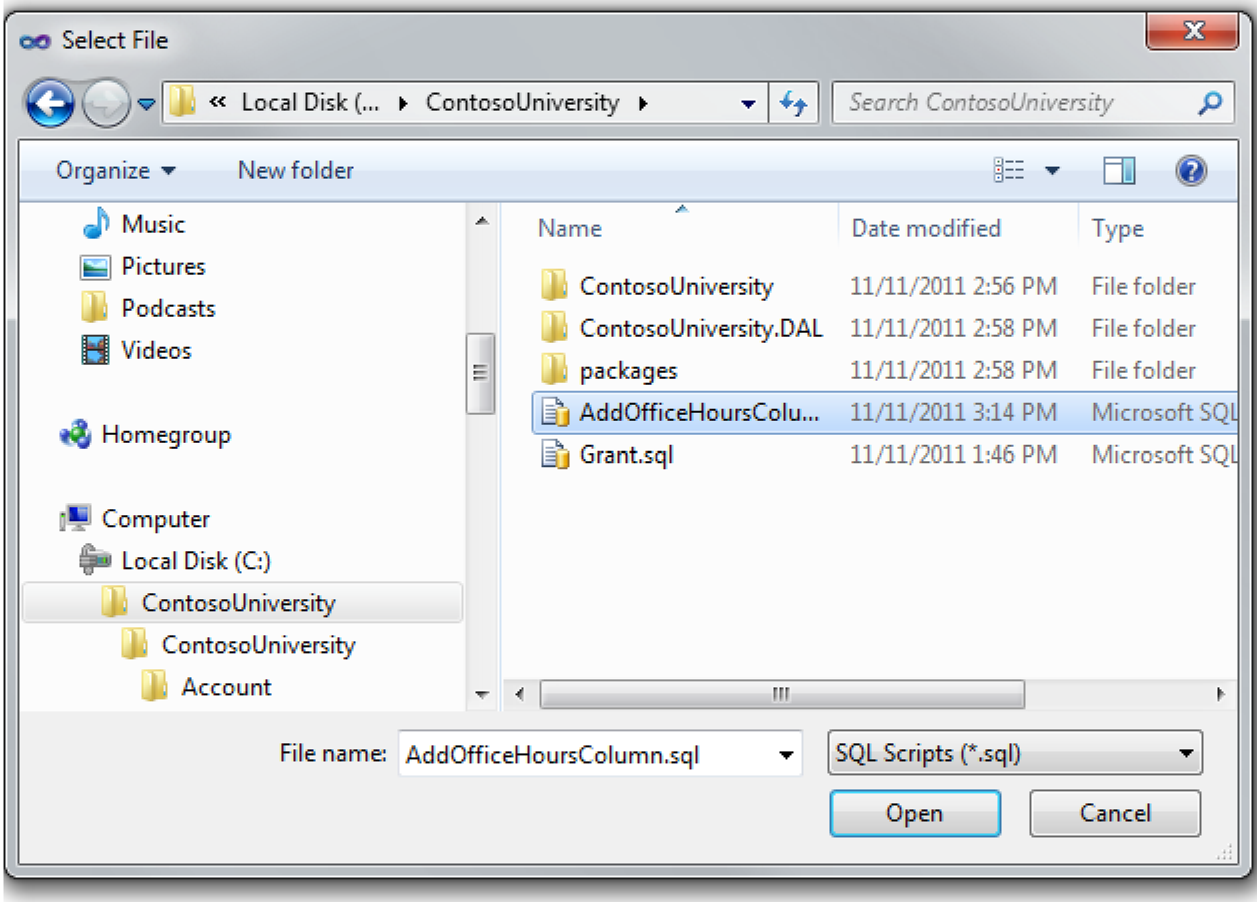
☑ Include all databases configured in Package/Publish SQL tab Ope

Select the **Package/Publish SQL** tab and make sure that **Configuration** is still set to **Test**. In the **Database Entries** table, clear the check box next to **DefaultConnection-Deployment**, since you have no changes to deploy to the membership database.

In the **Database Entries** table, select **SchoolContext-Deployment** in order to configure settings that apply to the School database.

In the **Database Entry Details** section clear the check boxes next to both of the scripts that are currently listed in the **Database Scripts** table. (Clearing the check box for the auto-generated script also clears the **Pull data and/or schema from an existing database** check box.)

Under the **Database Scripts** table, click **Add Script**. In the **Select File** dialog box, select the *AddOfficeHoursColumn.sql* file that you saved in the solution folder. Then click **Open**.

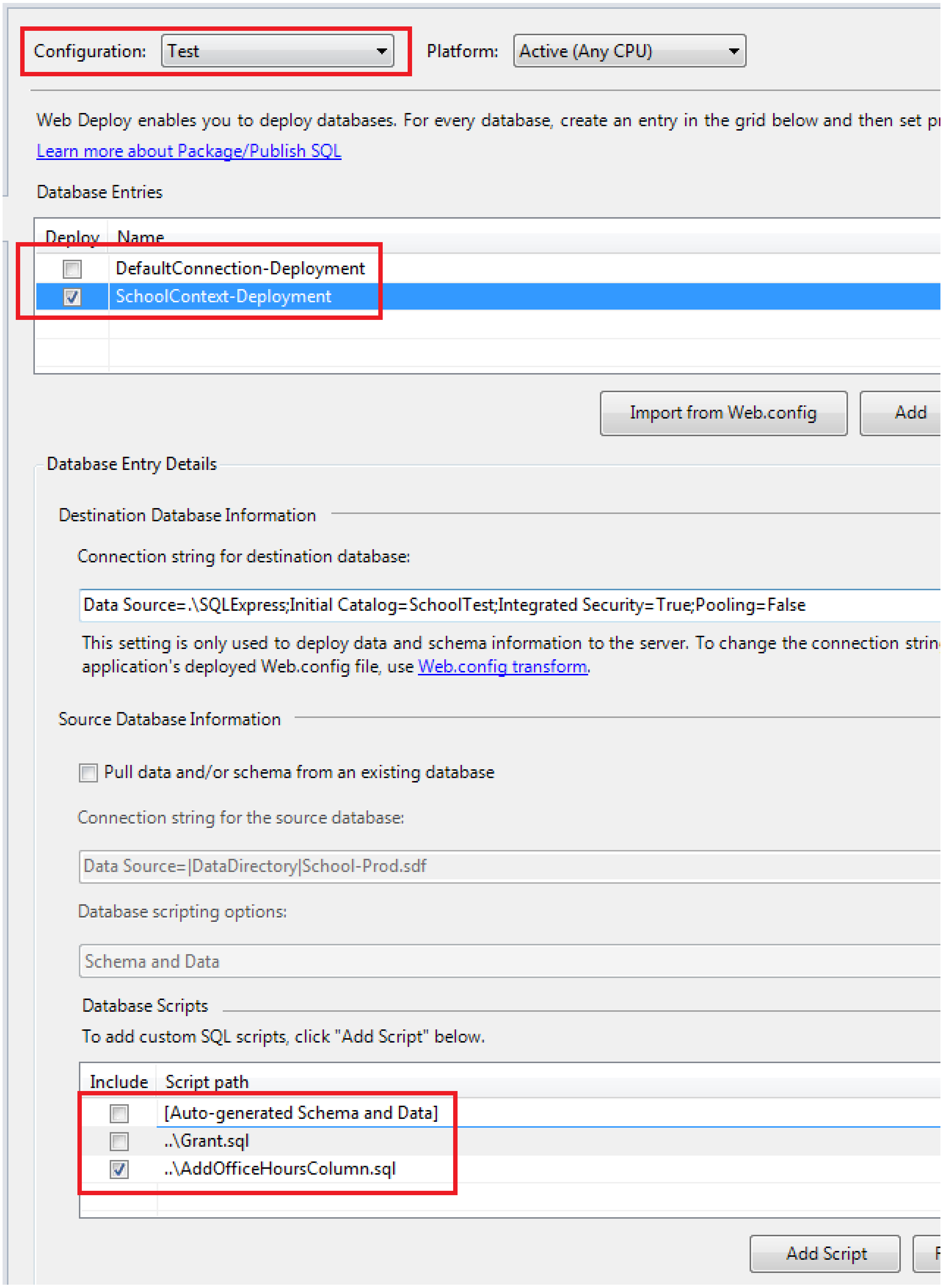


Name	Date modified	Type
ContosoUniversity	11/11/2011 2:56 PM	File folder
ContosoUniversity.DAL	11/11/2011 2:58 PM	File folder
packages	11/11/2011 2:58 PM	File folder
AddOfficeHoursColu...	11/11/2011 3:14 PM	Microsoft SQL
Grant.sql	11/11/2011 1:46 PM	Microsoft SQL

File name: AddOfficeHoursColumn.sql

SQL Scripts (*.sql)

When you are done, the **Package/Publish SQL** tab will look like this:



Configuration: Test

Platform: Active (Any CPU)

Web Deploy enables you to deploy databases. For every database, create an entry in the grid below and then set pr

Learn more about Package/Publish SQL

Database Entries

Deploy	Name
☐	DefaultConnection-Deployment
✓	SchoolContext-Deployment

Import from Web.config | Add

Database Entry Details

Destination Database Information

Connection string for destination database:

Data Source=.\SQLExpress;Initial Catalog=SchoolTest;Integrated Security=True;Pooling=False

This setting is only used to deploy data and schema information to the server. To change the connection strin application's deployed Web.config file, use Web.config transform.

Source Database Information

☐ Pull data and/or schema from an existing database

Connection string for the source database:

Data Source=|DataDirectory|School-Prod.sdf

Database scripting options:

Schema and Data

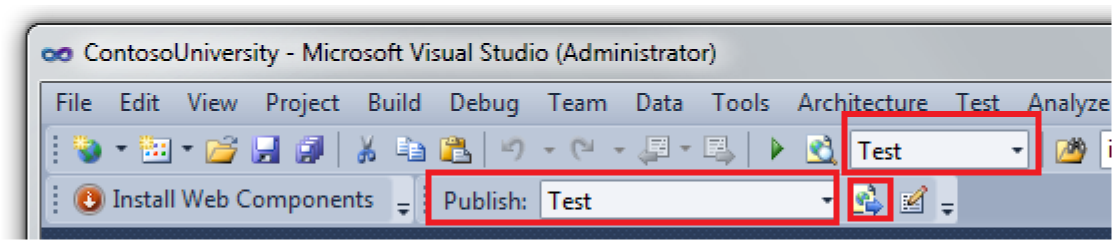
Database Scripts

To add custom SQL scripts, click "Add Script" below.

Include	Script path
☐	[Auto-generated Schema and Data]
☐	..\Grant.sql
✓	..\AddOfficeHoursColumn.sql

Add Script

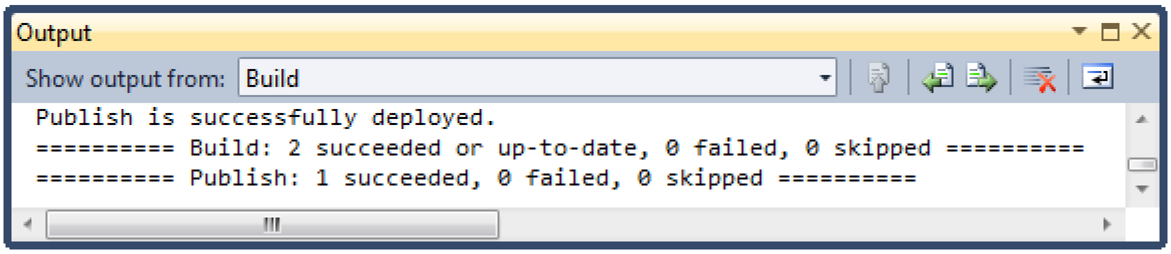
In the **Solution Configurations** drop-down box, select **Test**. In the **Publish** profile drop-down box. select **Test**. Click **Publish Web**. (If you can't do this in the toolbars because you have have customized toolbars, see the Deploying a Code-Only Update tutorial for an alternative method.)


ContosoUniversity - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze
Install Web Components

Test

Publish: Test

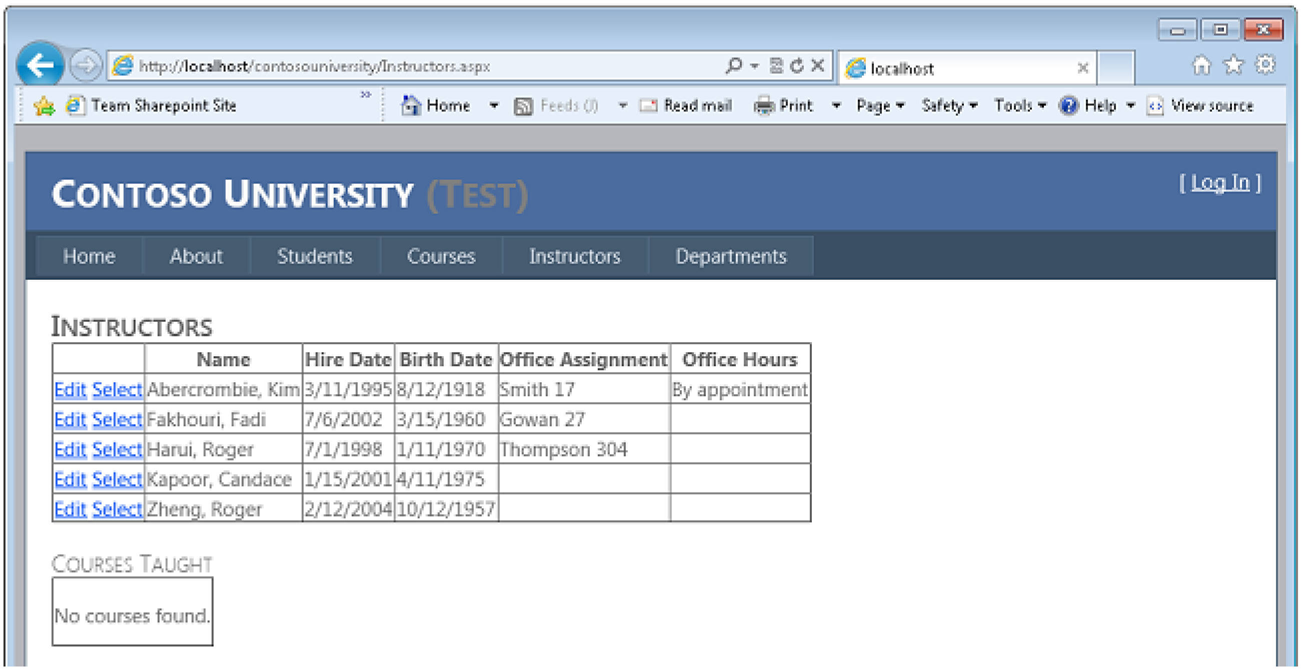
Visual Studio deploys the updated application and reports success in the **Output** window.



Show output from: Build

```
Publish is successfully deployed.
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

You can now open a browser and run the application to verify that the update was successfully deployed. You see the office hours column, and you can edit an instructor to add data in it. In the image below, an Office Hours value was added to the first instructor in the list.



INSTRUCTORS

	Name	Hire Date	Birth Date	Office Assignment	Office Hours
Edit Select	Abercrombie, Kim	3/11/1995	8/12/1918	Smith 17	By appointment
Edit Select	Fakhouri, Fadi	7/6/2002	3/15/1960	Gowan 27	
Edit Select	Harui, Roger	7/1/1998	1/11/1970	Thompson 304	
Edit Select	Kapoor, Candace	1/15/2001	4/11/1975		
Edit Select	Zheng, Roger	2/12/2004	10/12/1957		

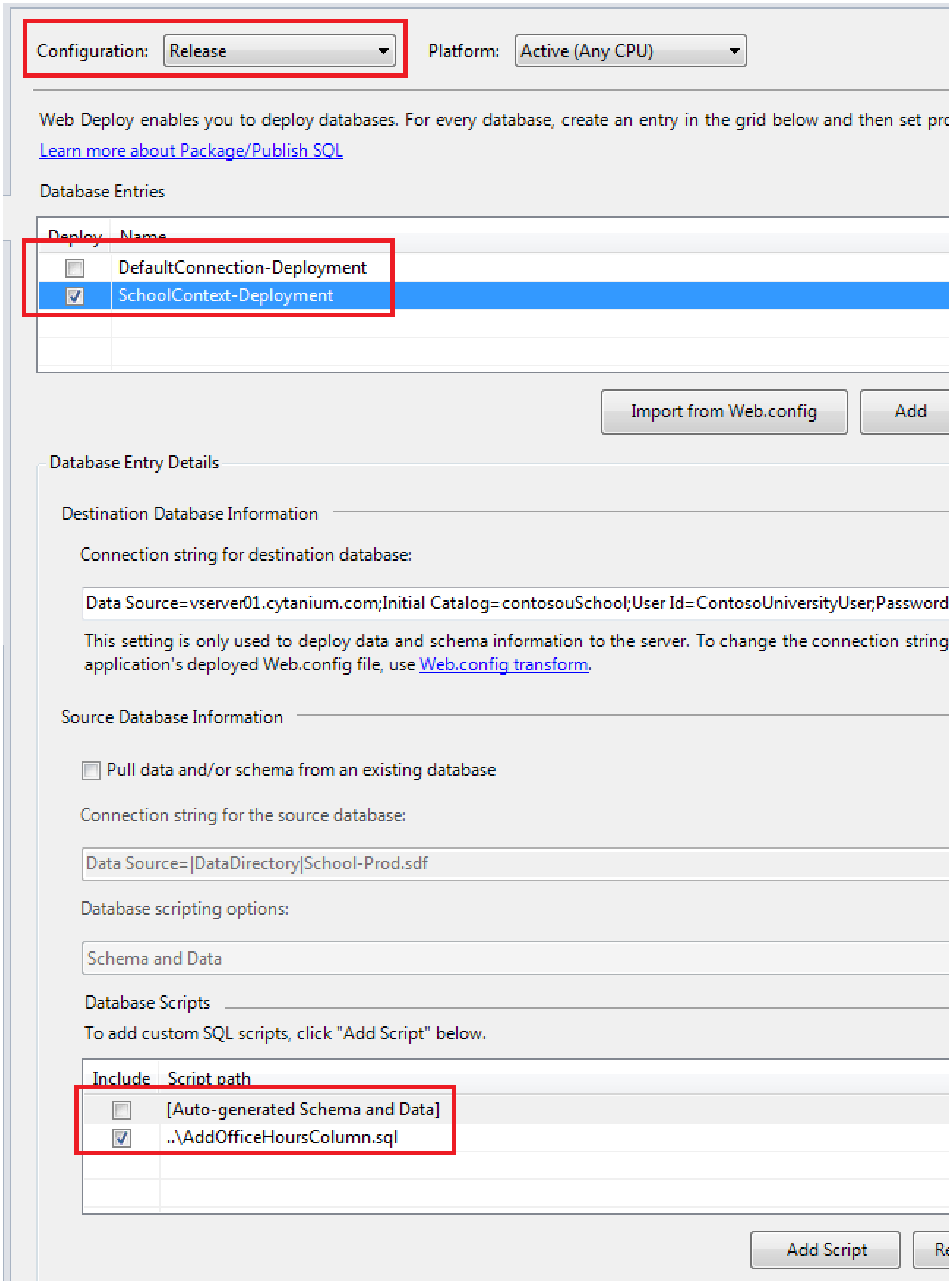
COURSES TAUGHT

No courses found.

Deploying the Database Update to the Production Environment

With the exception of copying *app_offline.htm* before deployment and removing it afterward, the process for deploying to production is the same.

When you are done configuring the **Package/Publish SQL** tab, it will look like this:


Configuration: Release

Platform: Active (Any CPU)

Web Deploy enables you to deploy databases. For every database, create an entry in the grid below and then set pro

Learn more about Package/Publish SQL

Database Entries

Deploy	Name
☐	DefaultConnection-Deployment
☑	SchoolContext-Deployment

Import from Web.config

Add

Database Entry Details

Destination Database Information

Connection string for destination database:

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

This setting is only used to deploy data and schema information to the server. To change the connection string application's deployed Web.config file, use Web.config transform.

Source Database Information

☐ Pull data and/or schema from an existing database

Connection string for the source database:

Data Source=|DataDirectory|School-Prod.sdf

Database scripting options:

Schema and Data

Database Scripts

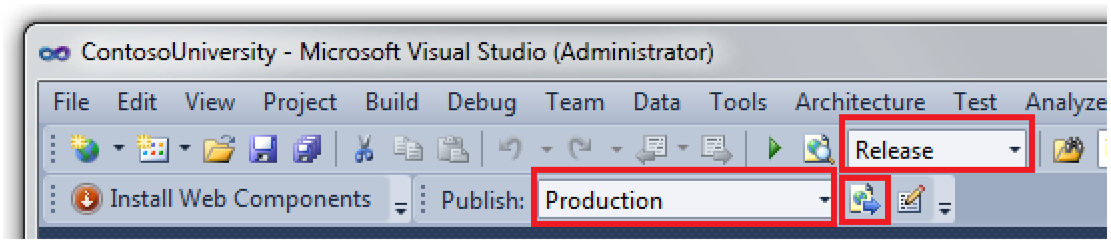
To add custom SQL scripts, click "Add Script" below.

Include	Script path
☐	[Auto-generated Schema and Data]
☑	..\AddOfficeHoursColumn.sql

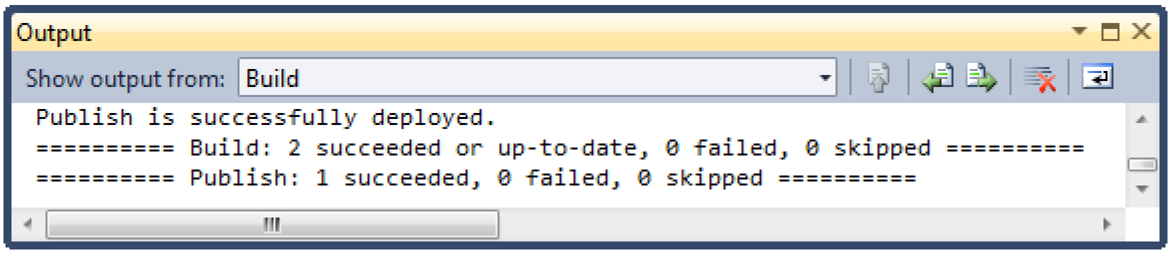
Add Script

Re

When this is done, you can deploy to production. In the **Solution Configurations** drop-down box, select **Release** build configuration, and in the **Publish** profile drop-down box, select **Production**. Click **Publish Web**. (If you can't do this in the toolbars because you have have customized toolbars, see the Deploying a Code-Only Update tutorial for an alternative method.)


ContosoUniversity - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze
Release
Install Web Components
Publish: Production

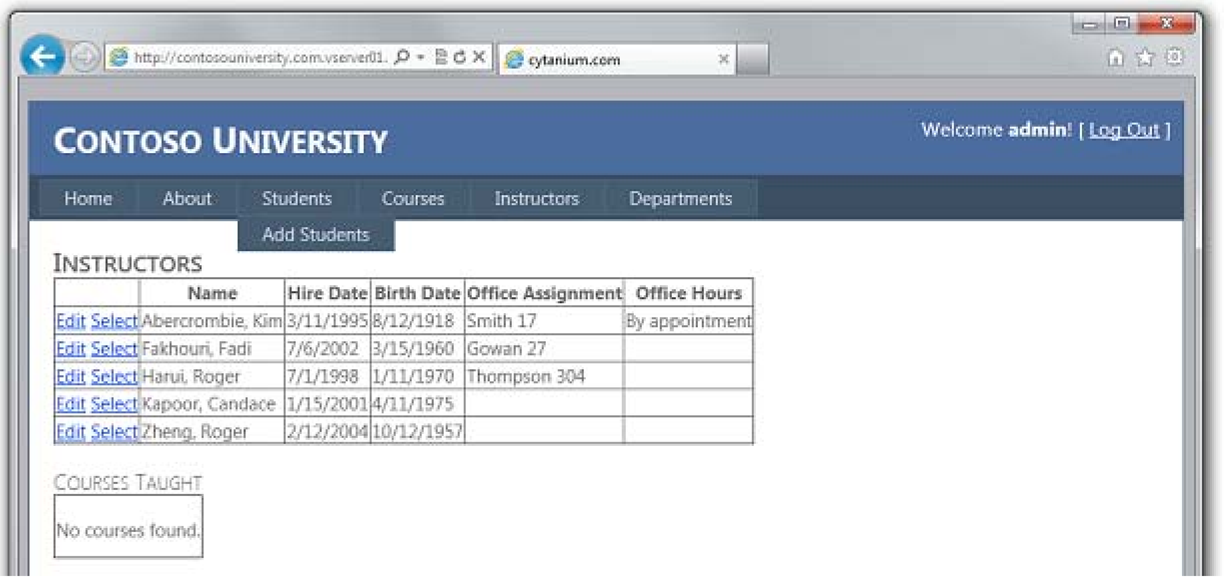
Visual Studio deploys the updated application and reports success in the **Output** window.



```
Publish is successfully deployed.
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

Before you can test to verify successful deployment, you have to remove the *app_offline.htm* file.

You can now open a browser and browse your public site to verify that the update was successfully deployed. In the following image, the first instructor in the list was edited to add a value to the Office Hours field to verify that writing to the database works.



INSTRUCTORS

	Name	Hire Date	Birth Date	Office Assignment	Office Hours
Edit Select	Abercrombie, Kim	3/11/1995	8/12/1918	Smith 17	By appointment
Edit Select	Fakhouri, Fadi	7/6/2002	3/15/1960	Gowan 27	
Edit Select	Harui, Roger	7/1/1998	1/11/1970	Thompson 304	
Edit Select	Kapoor, Candace	1/15/2001	4/11/1975		
Edit Select	Zheng, Roger	2/12/2004	10/12/1957		

COURSES TAUGHT

No courses found.

You have now successfully deployed an application update that included a database change, using the same change script for both SQL Server Express and SQL Server.

More Information

This completes this series of tutorials on deploying an ASP.NET web application to a third-party hosting provider. For an example that shows how to use a deployment package to deploy the ContosoUniversity application that you work with in these tutorials, see Using a Web Deploy package to deploy to IIS on the dev box and to a third party host on Sayed Hashimi's blog. 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 document:

- Alberto Poblacion, MVP & MCT, Spain
- Jarod Ferguson, Data Platform Development MVP, United States
- Kristina Olson, Microsoft
- Mike Pope, Microsoft
- Mohit Srivastava, Microsoft
- Raffaele Rialdi, Italy
- Rick Anderson, Microsoft
- Sayed Hashimi, Microsoft
- Scott Hanselman, Microsoft
- Scott Hunter, Microsoft
- Sr• an Božovi•, Serbia
- Vishal Joshi, Microsoft

Troubleshooting (12 of 12)

You develop a web application in order to make it available to people over the Internet. But web programming tutorials typically stop right after they've shown you how to get something working on your development computer. This series of tutorials begins where the others leave off: you've built a web site, tested it, and it's ready to go. What's next? These tutorials guide you through how to deploy. You first deploy to IIS on your local development computer for testing, and then to a third-party hosting provider. The application that you'll deploy is a Web application project that uses ASP.NET Web Forms, the Entity Framework, SQL Server, and the ASP.NET membership system.

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.

« Previous Tutorial | Next Tutorial »

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

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, 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, 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</AutoParameterizationWe
bConfigConnectionStrings>
```

Then redeploy the application.

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.

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. If you have SQL Server Management Studio (SSMS) installed, you can resolve this problem by deleting the existing database. If you do not have SSMS, you can 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.

SQL Error When a Script Attempts to Create Users or Roles

Scenario

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.
```

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.

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.