Chapter 3

Quick Start - MVC

Table of Contents

Using Haystack Generated Code in MVC

Haystack generates generic .NET code that can be used in any version of .NET from 4.5 and later. The generated code relies on a few things in order to work.

1. Some PDSA .DLLs need to be added to your Visual Studio project
2. A reference to System.Linq.Dynamic.dll (included in Haystack)
3. Configuration entries need to be added to your Web.Config or App.Config file in your Visual Studio project.
4. To distribute your .NET application you will need to create a runtime license from Haystack. A runtime license can only be generated from a purchased copy of Haystack, not the trial version.

Quick Start for MVC Applications

This document describes how to use Haystack to quickly generate a working **MVC Application** in C#**.**

From your start menu locate the Haystack folder and click on the **Haystack Code Generator** icon. This will start the Haystack Code Generator for .NET (Figure 1). After starting Haystack add a new Haystack Project. A project is used to generate all objects from a SQL Server database.

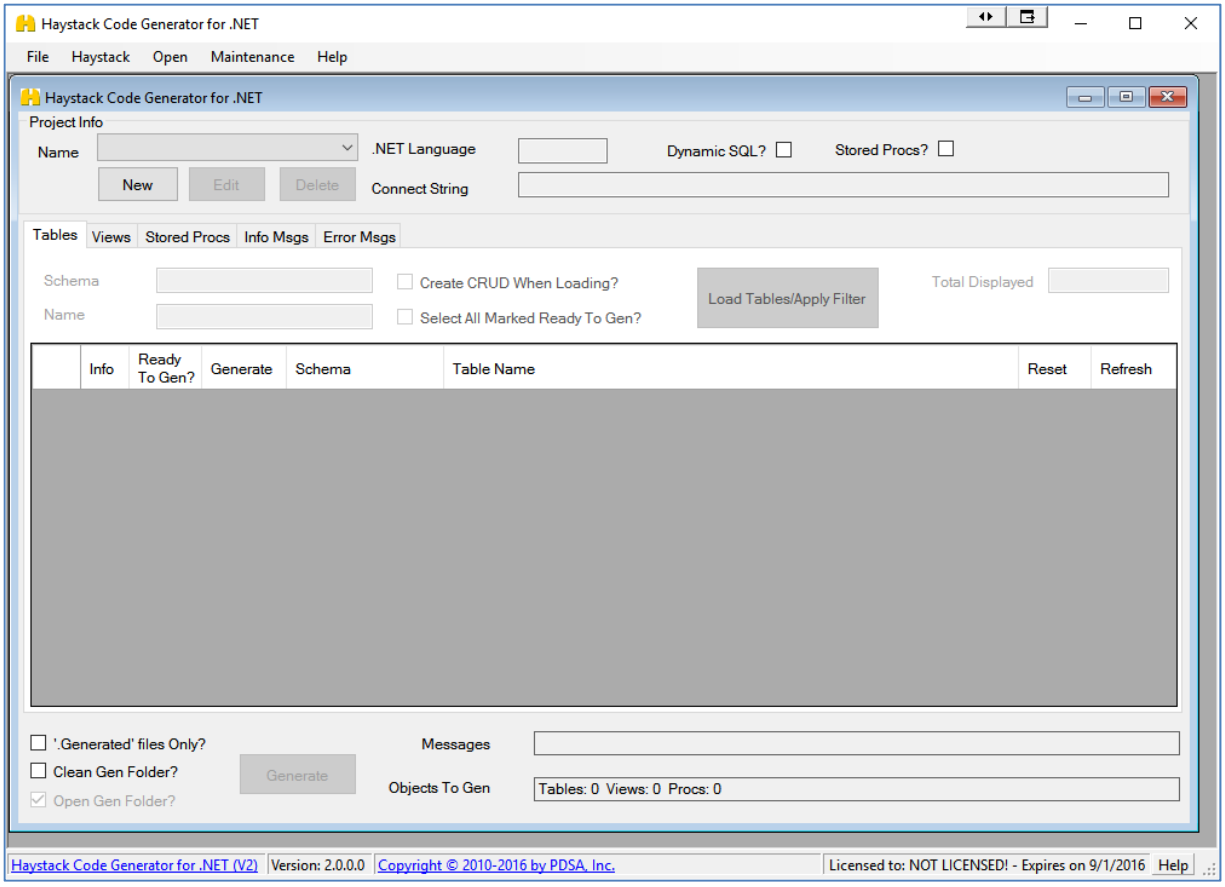

Figure 1: Haystack Main Screen

Add New Haystack Project for MVC

Click on the “New” button beneath the Project Info Name Combo Box (Figure 2) to add a new project to Haystack.

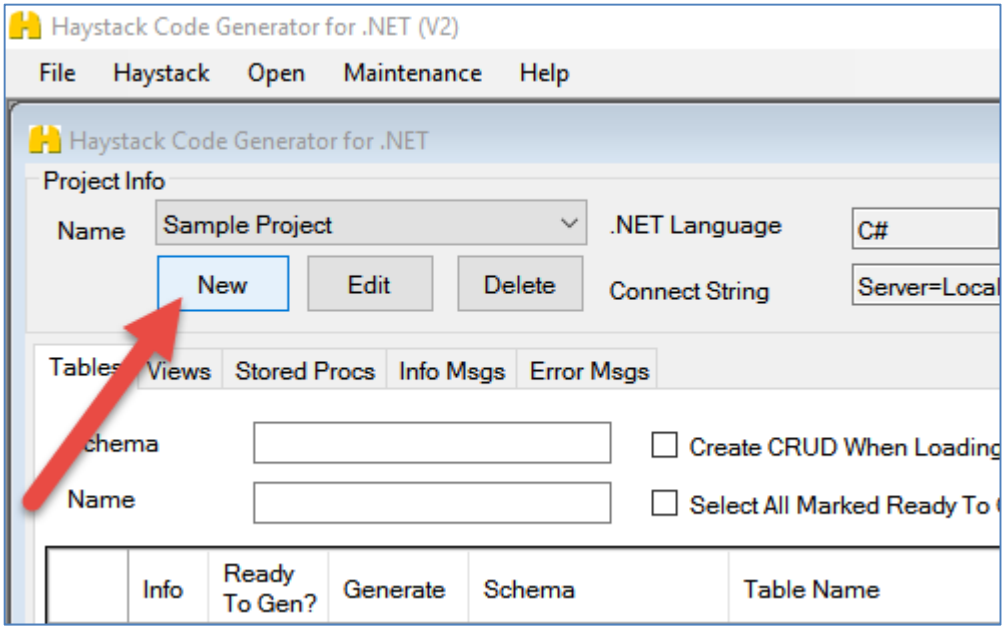

Figure 2: Click "New" to create a new Project

You will now be presented with the **Project Information** screen. On this screen is where you will create the project that points to a database.

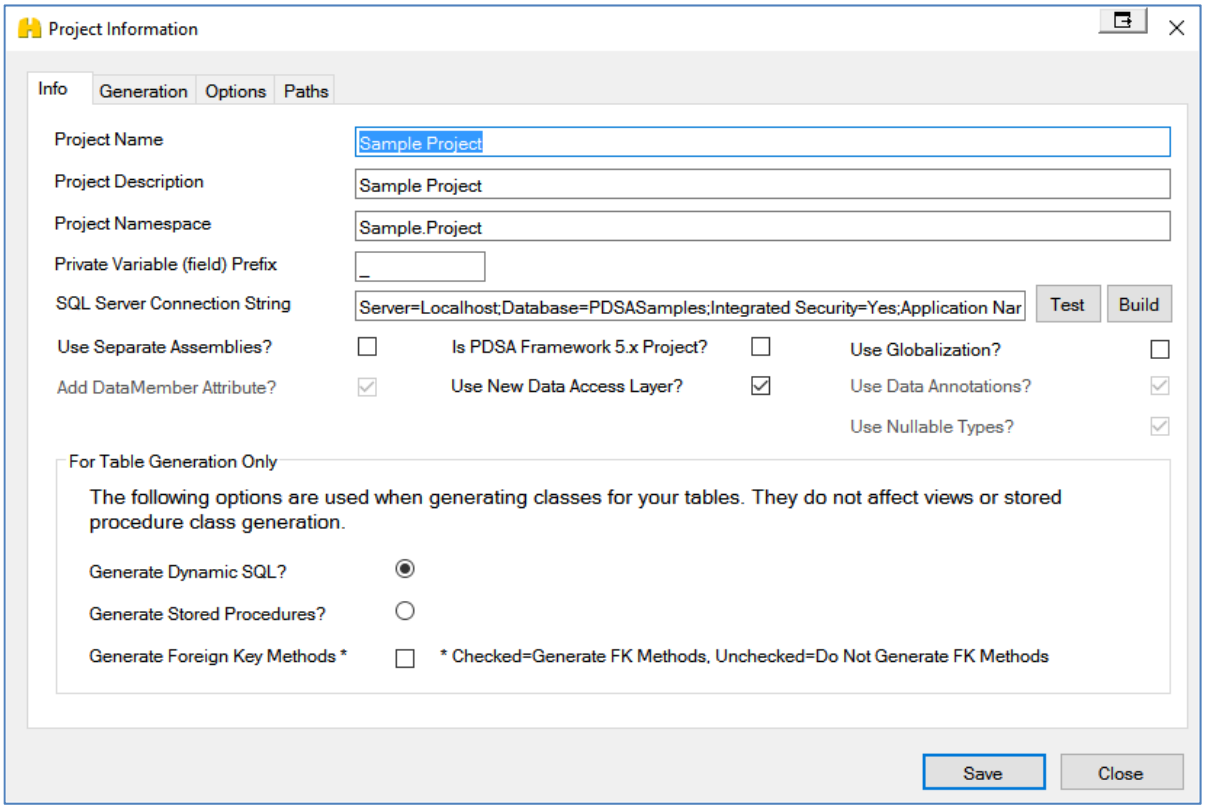

Figure 3: Project Information (Info Tab)

All you need to create a project is on the first and second tabs (Figure 3). For this quick start, fill in a connection string to one of your databases, or use the PDSASamples database you may have installed with Haystack.

Click on the **Generation Tab** (Figure 4) and select the "MVC – Data Access Layer" template as shown in Figure 4.

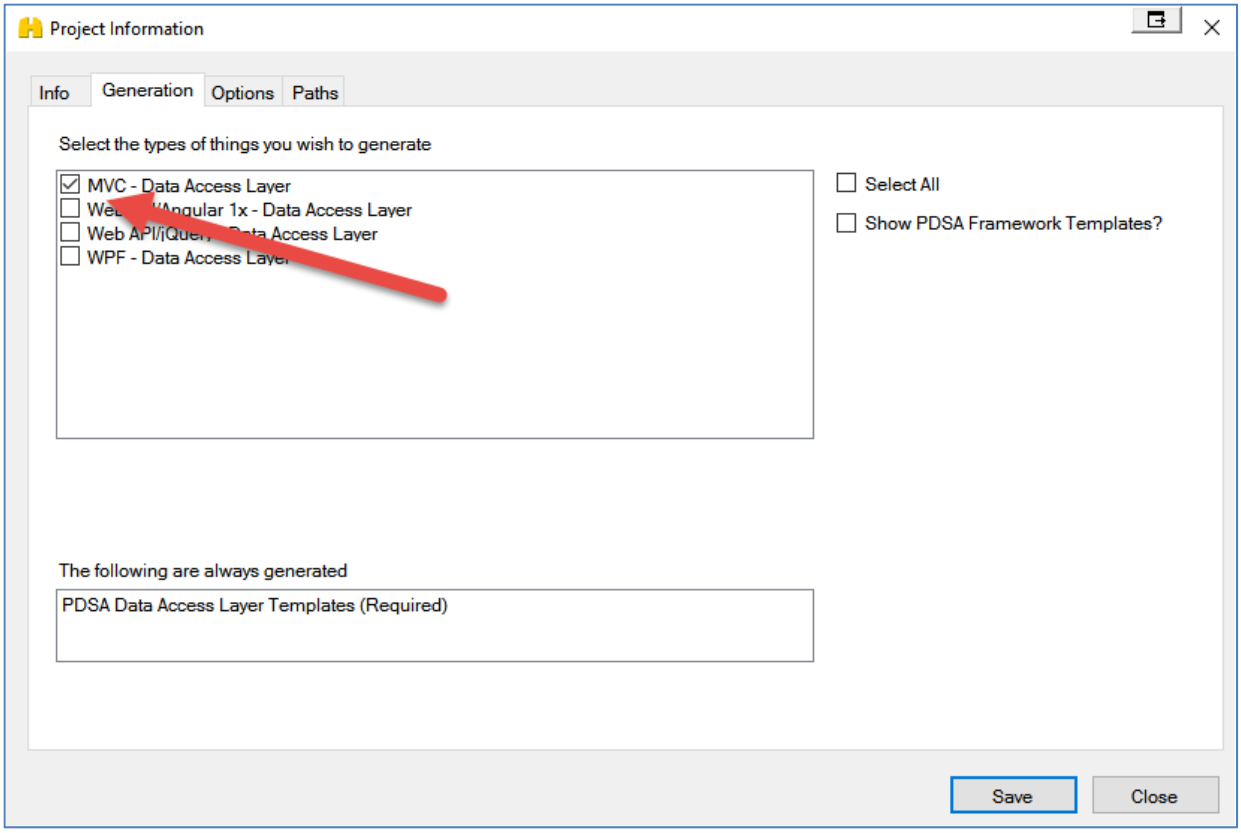

Figure 4: Project Information (Generate Tab)

Click the **Save** button to save this new project into the Haystack database.

Generate CRUD Classes for Tables

After you have created a project and set the connection string to a valid server and database, you are ready to read in the list of tables in the database. Click on the **Load Tables/Apply Filter** button (Figure 5) to load in all the tables.

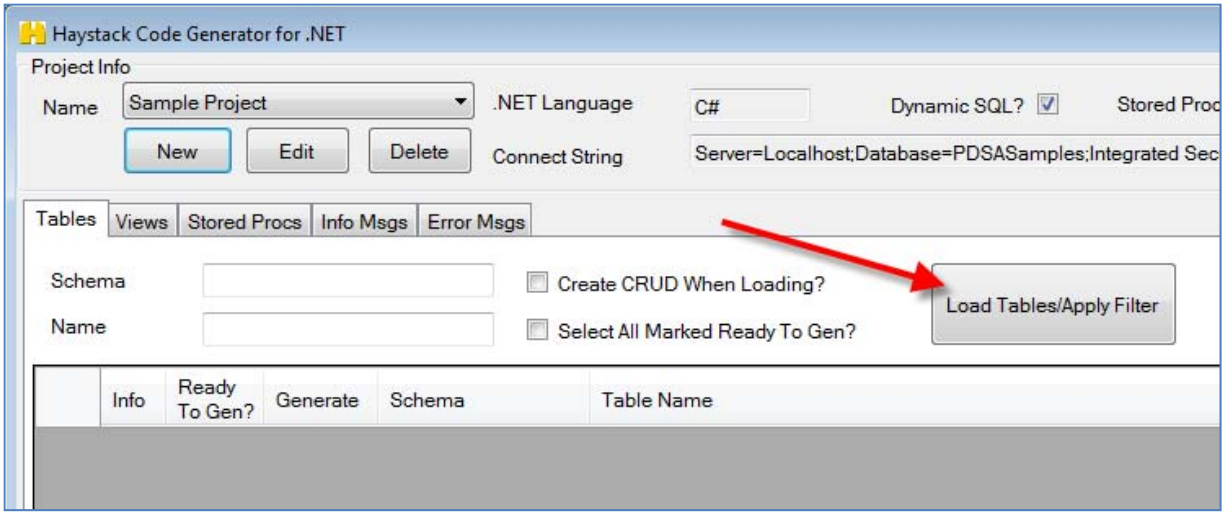

Figure 5: Load Tables

After clicking the **Load Tables/Apply Filter** button the grid on the lower part of the screen will be filled in with the names of the tables that match the filter.

At this point you need to click on the **Info** button to the left of a table to load all of the columns for the table and display the Table Information Screen (Figure 6). You must open the table to load all columns in order to generate classes for a table.

Table Information Screen

On the Table Information Screen (Figure 6) you can add additional business rules for each column. You can read more about this screen in another chapter. But to start all you have to do right now is to click on the **Save Table Info** button. This saves all the columns for the table into the Haystack database so it can be generated.

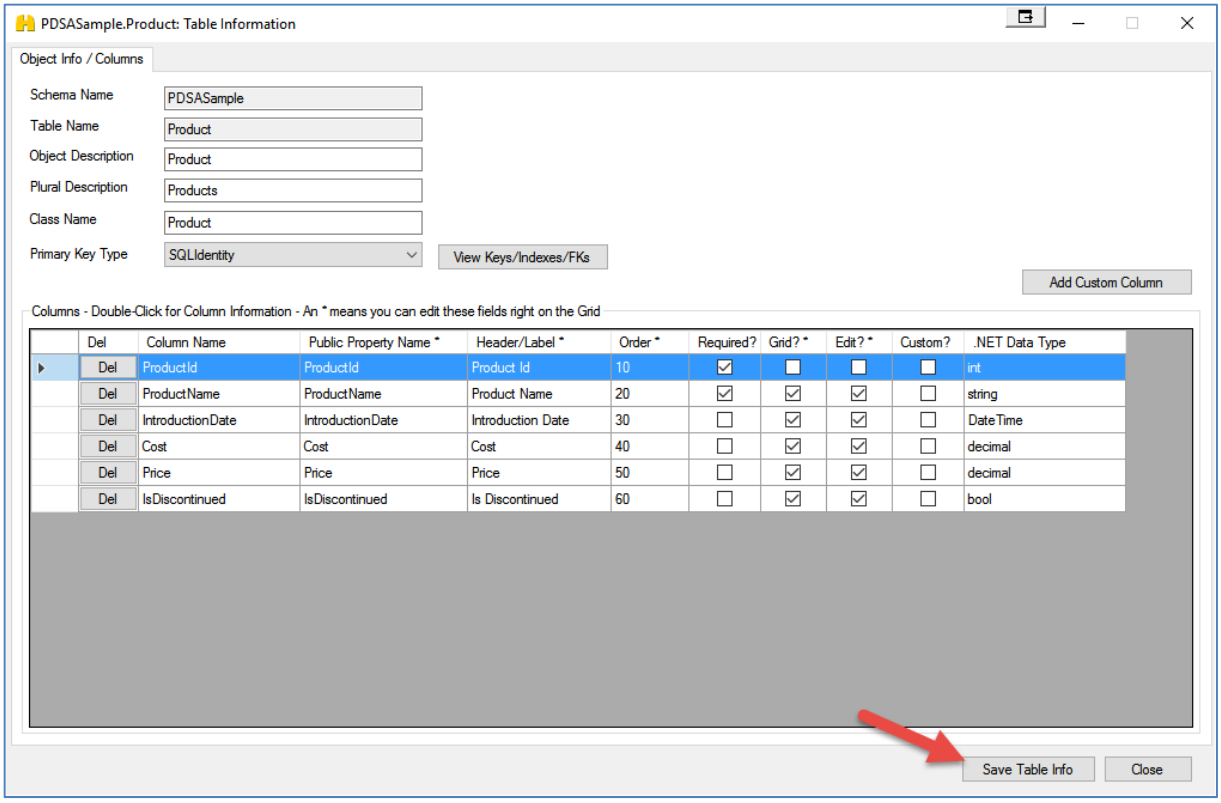

Figure 6: Table Information Screen

Generate Table Data Classes

After saving the table information you will notice that the **Generate** check box is now checked (Figure 7). You may now click on the **Generate** button to generate the CRUD classes for the table selected.

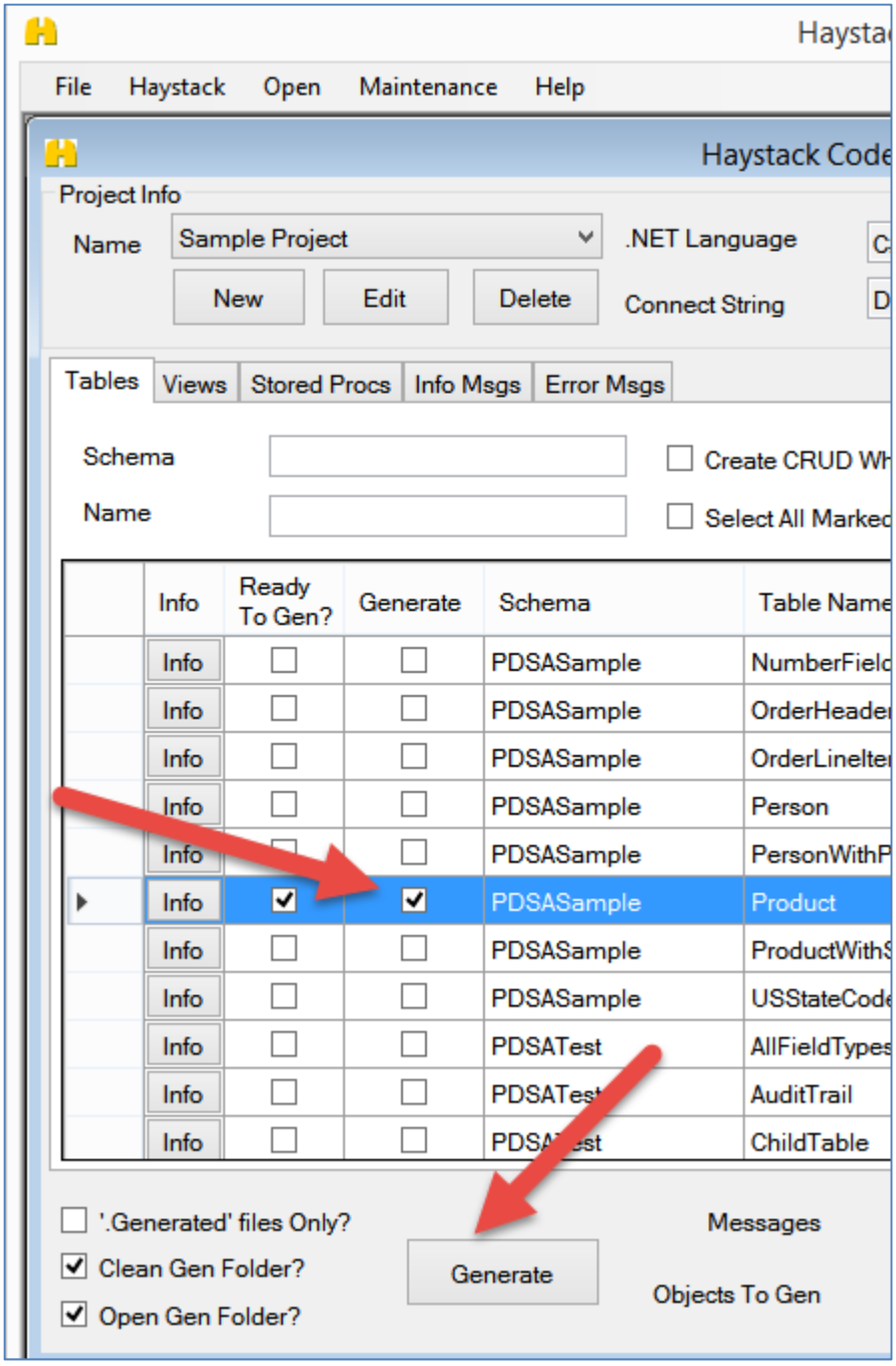

Figure 7: Generate Code

After the generation completes, a Windows Explorer window will open where all of the code was generated.

Create New MVC Project

Once the data access classes and MVC Pages are generated you are ready to put them into a project and try them out.

Open Visual Studio 2015 (or later).

Click on **File** | **New** | **Project** from the menu

1. Choose the **Web** Templates (Figure 8)
2. Click on the **ASP.NET Web Application (.NET Framework)** template.
3. Fill in the **Name:** field with **Sample.Project** (or whatever namespace you used in the Project Information screen in Haystack).
4. Type in the path where you wish to save this project.
5. Click the OK button.

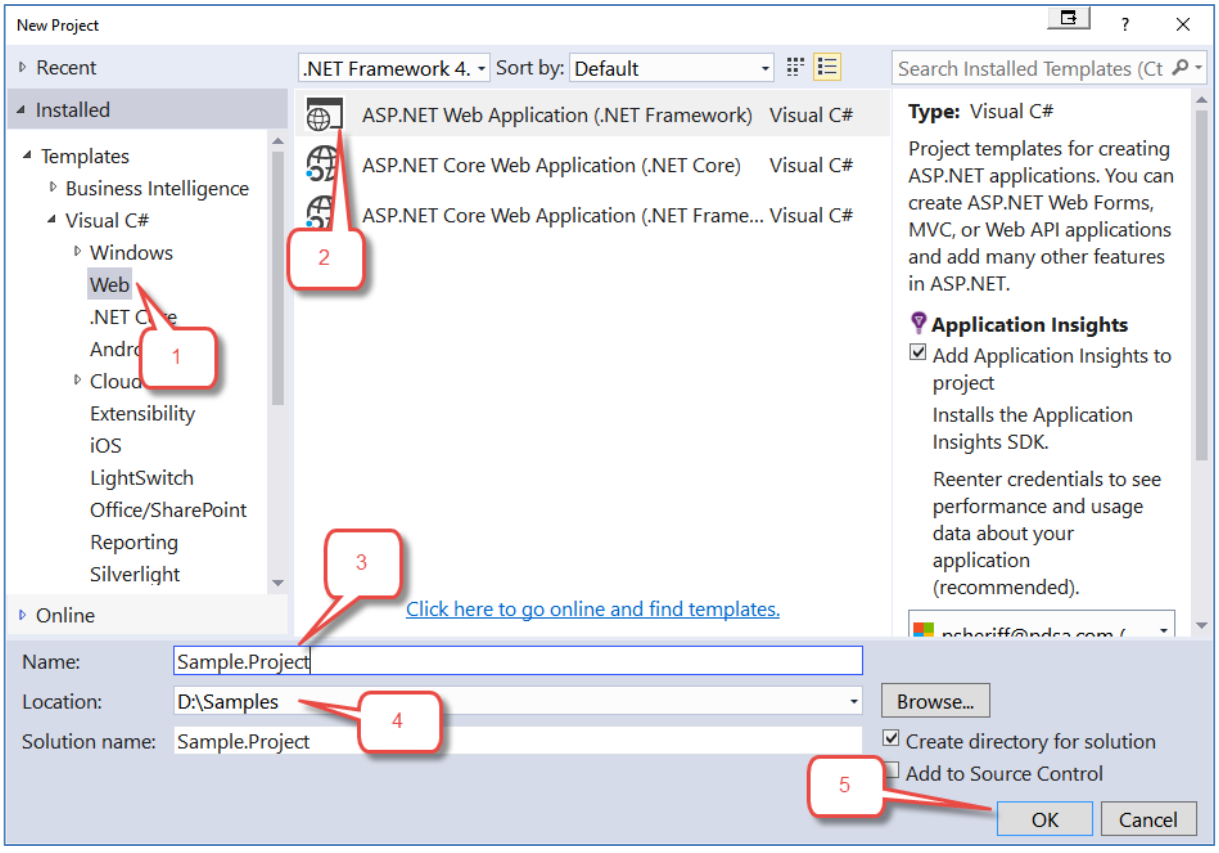

Figure 8: Create a new Project

On the next screen (Figure 9) select the MVC template and check the “Web API” check box. Click the **OK** button to have it create the new project.

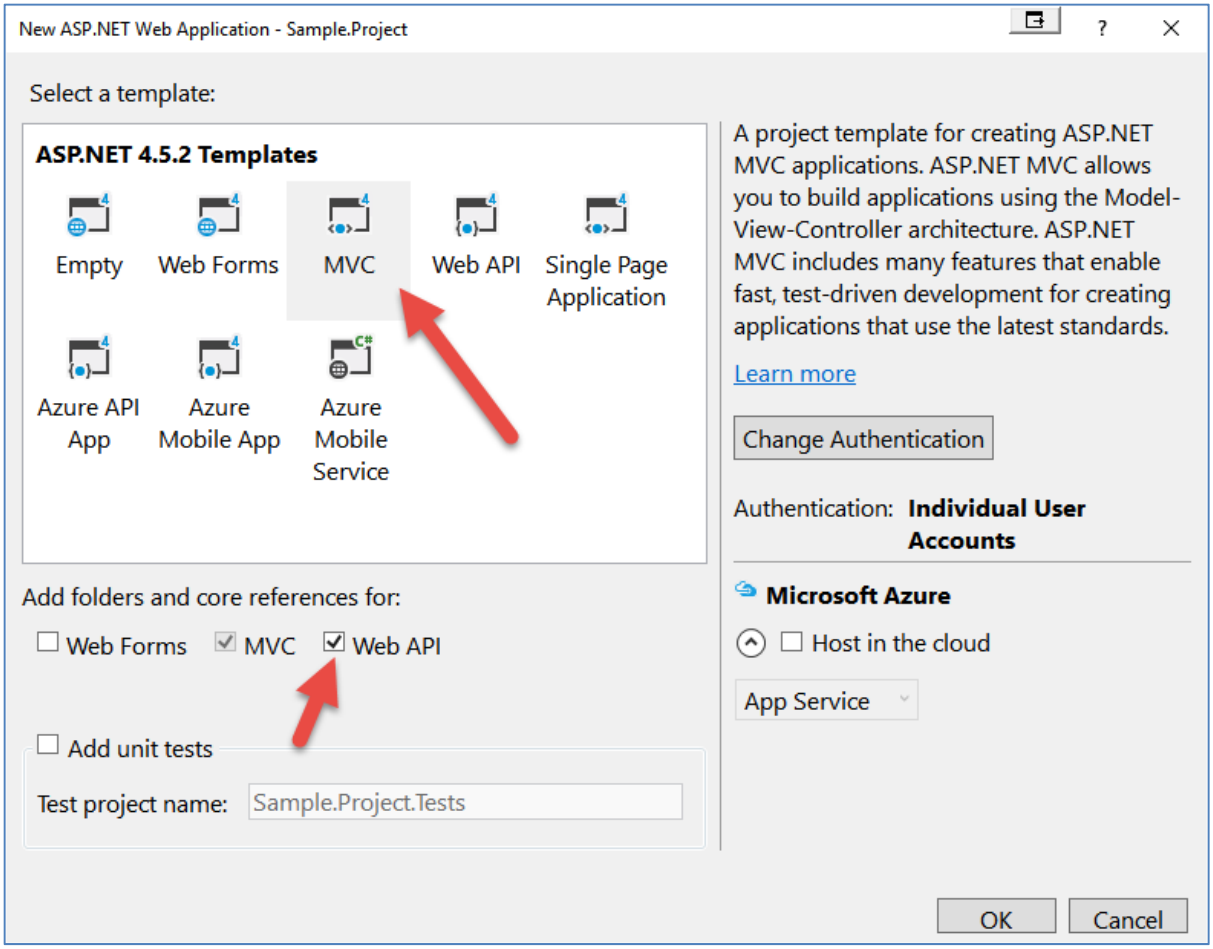

Figure 9: Select MVC and Web API

Copy Generated Files

In the Windows Explorer window (Figure 10) where the code was generated, expand the **MVC** folder and select all the folders and files and copy and paste those into the root of your new web application project in Visual Studio.

NOTE:	If you are prompted that folders already exist with the same name and do you wish to replace them, answer **Yes**.

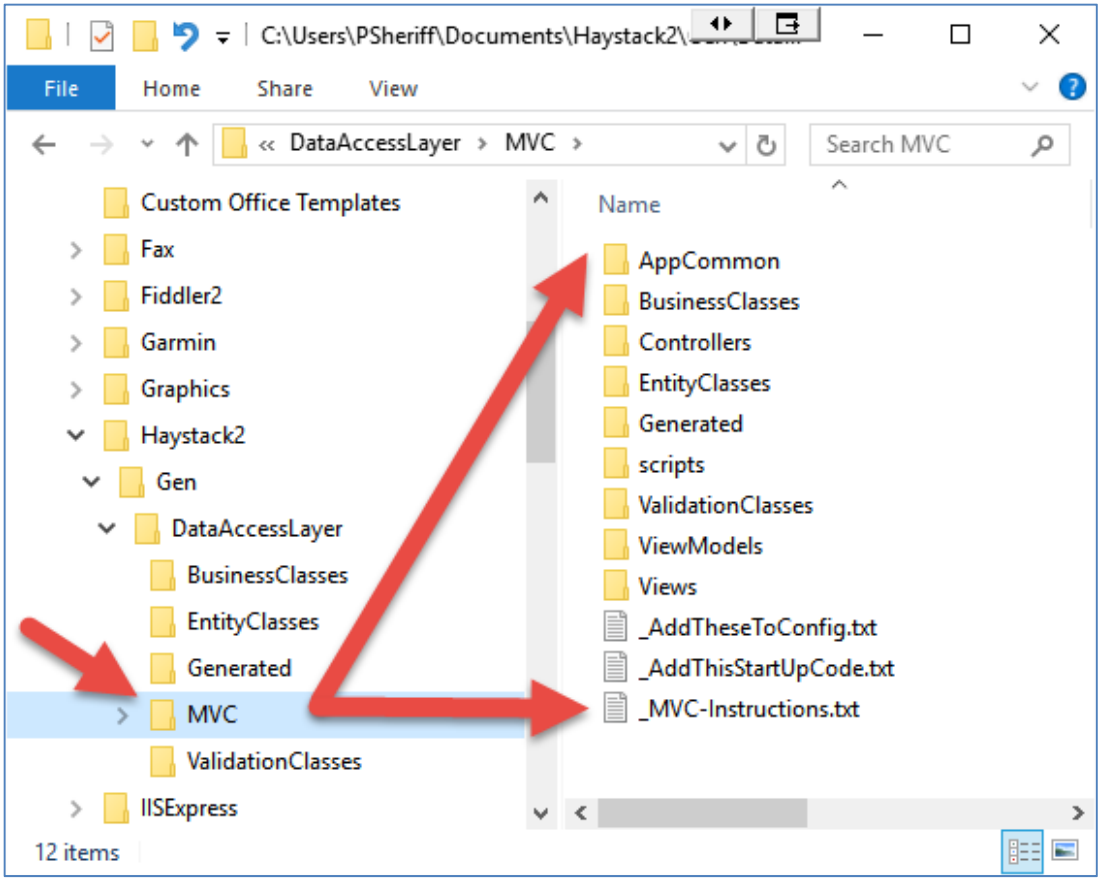

Figure 10: Copy all folders from the MVC folder into your Visual Studio project

Your Solution Explorer window in your MVC application should now look like Figure 11.

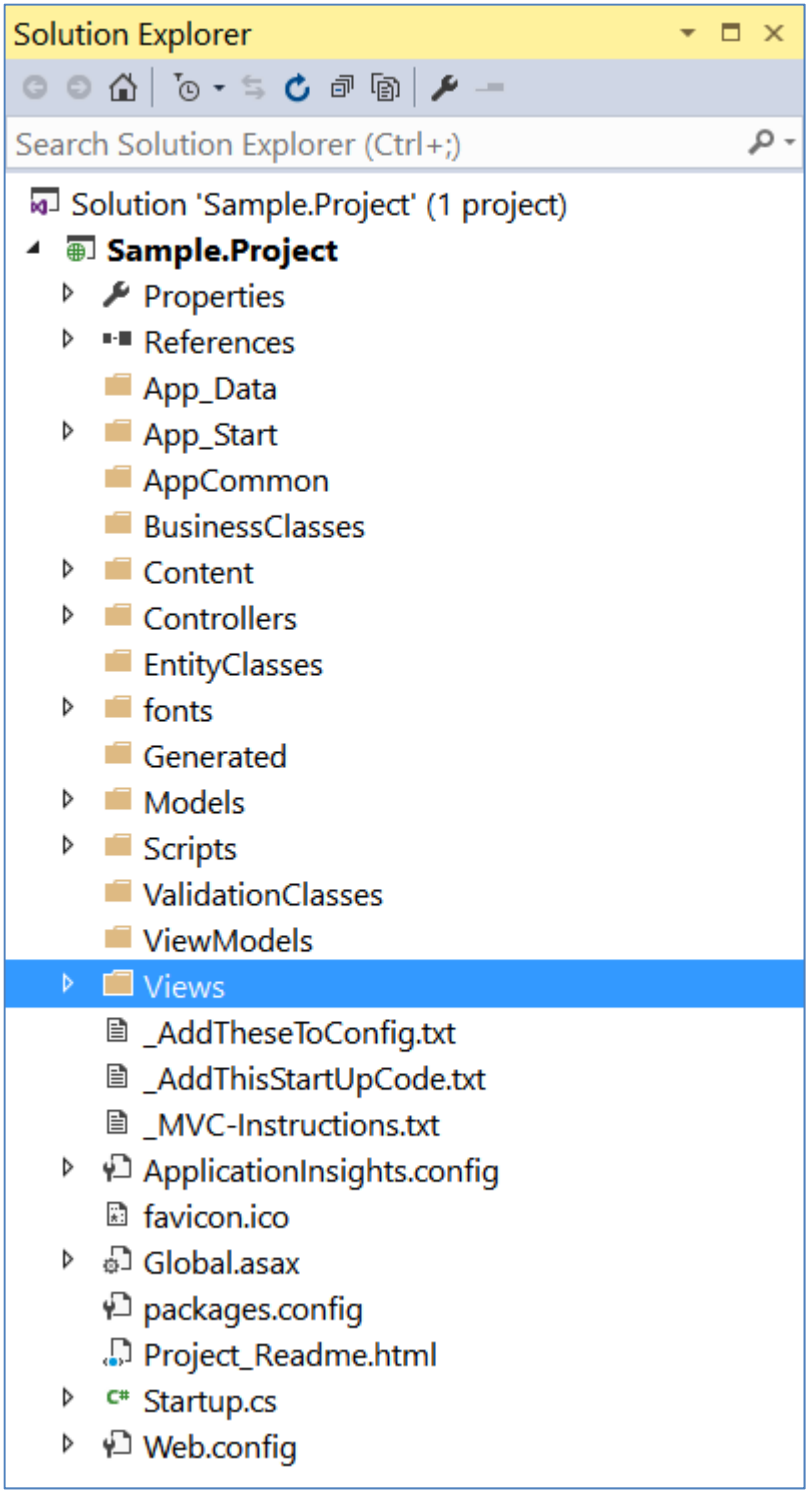

Figure 11: Your Visual Studio project after adding generated code

Include Files in Project (VS Bug) - MVC

Click on the project in the Solution Explorer window, then click on the “Show All Files” icon as shown in Figure 12.

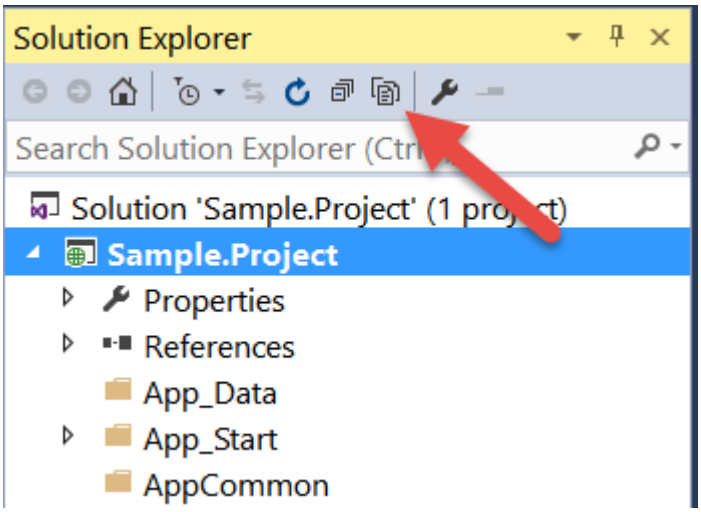

Figure 12: Click the Show All Files button to add the hidden files

Go to each of the following folders, locate the files you copied in, right mouse click and "Include in Project" each file.

- AppCommon
- BusinessClasses
- EntityClasses
- Generated
- ValidationClasses
- ViewModels
- Views\[CLASS NAME]

NOTE:	This is a bug in Visual Studio where the first time you add a new folder to the project any files contained in that new folder do not get automatically added.

Each time you add a new view, by default a new folder is added, so you will need to include the .cshtml file for each new class. Or, you may simply copy the .cshtml file into an existing folder.

Add DLLs for MVC

In order for the generated classes to compile you need to add some DLLs to the project. Click on the **References** folder in your Visual Studio project. Right mouse click and select **Add Reference…** from the context menu.

Click the **Browse** button.

Navigate to the folder where you installed Haystack which is typically **C:\Program Files (x86)\Haystack2** and under the folder named **_Resources-For-MVC** select the following three DLL files shown in Figure 13.

- Desaware.MachineLicense40.dll
- PDSA.Common.dll
- System.Linq.Dynamic.dll

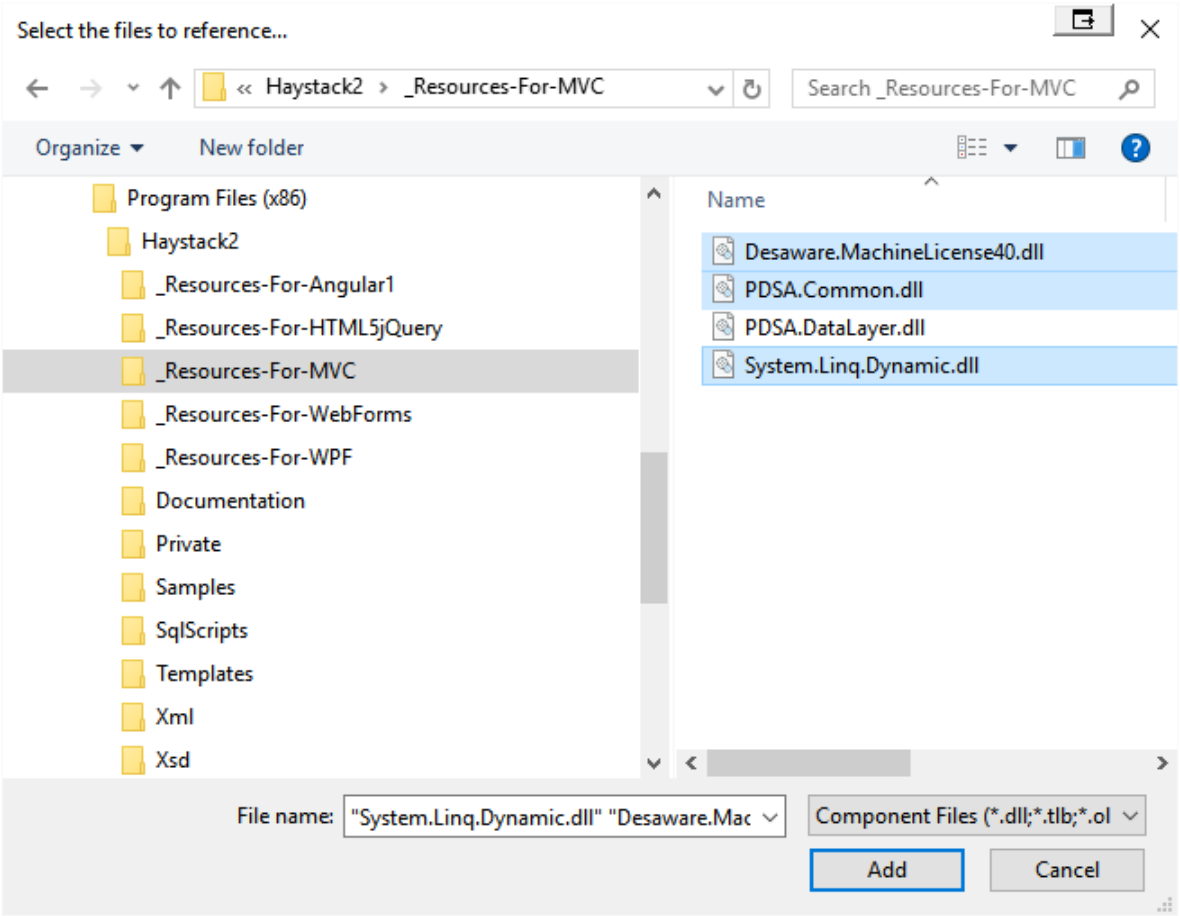

Figure 13: Reference the PDSA DLLs

Add the .NET Framework DLL called System.Runtime.Serialization.dll as shown in Figure 14.

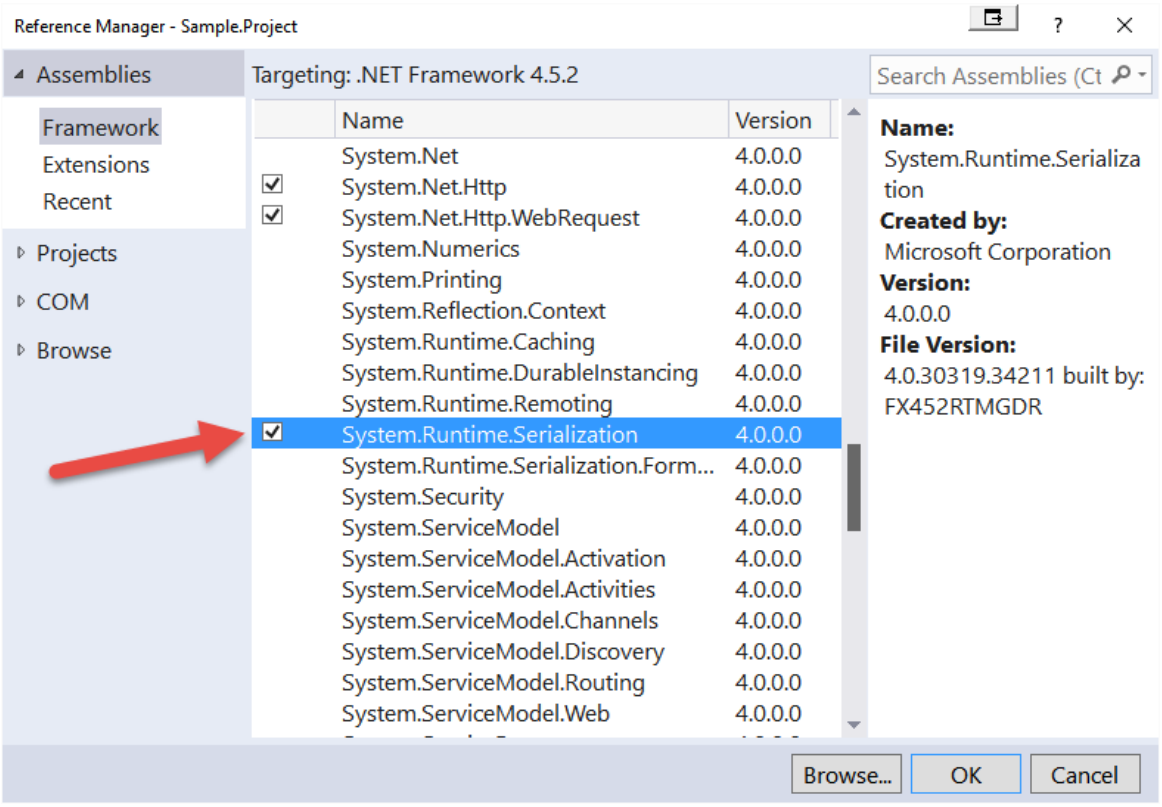

Figure 14: Reference the System.Runtime.Serialization DLL

At this point you should be able to successfully build the project. It is not ready to run yet, but everything should compile.

Fix up the Web.Config File for MVC

You need to add a connection string to your Web.Config file in order for the data classes to connect to the database from which you generated your classes. Open the file _AddTheseToConfig.txt in your project. In there you will find the connection string to add.

Copy and paste the connection string into the <connectionString> element in your Web.Config.

Delete the _AddTheseToConfig.txt file.

Modify the Global.asax in MVC

To initialize the PDSA Data Access layer you need to add some startup code. Open the _AddThisStartUpCode.txt file and copy the contents to the clipboard. Open the Global.asax file and paste this code after the last line in the Application_Start() method. You will need to add two 'using' statements in order for this code to compile as shown in Figure 15.

```
using PDSA.DataAccess;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Http;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;

namespace Sample.Project
{
    0 references
    public class MvcApplication : System.Web.HttpApplication
    {
        0 references
        protected void Application_Start() {
            AreaRegistration.RegisterAllAreas();
            GlobalConfiguration.Configure(WebApiConfig.Register);
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);

            // Initialize Database Manager Object
            PDSADataManager.Instance = new PDSADataManager(
                new PDSADataSqlServer(
                    ConfigurationManager.ConnectionStrings["Sample.Project"].ConnectionString));

            // Set the application name
            PDSADataManager.Instance.ApplicationName = "Sample Project";

        }
```

Figure 15: Add initialization code to the Global.asax Application_Start() method

Delete the _AddThisStartUpCode.txt file.

Delete the _MVC-Instructions.txt file.

Modify the Home\Index Page in MVC

NOTE:	This step is optional. You can just open your generated .cshtml page and run the application.

Open the \Home\Index.cshtml file.

Go to the bottom of the page.

Add code to call your generated view(s). You can add code like the following, replacing the string **"Product"** in the Html.ActionLink() with the name of your generated view.

```
<div class="row">
   <div class="col-md-12">
      Click here for @Html.ActionLink("Products", "Product",
                                      "Product")
   </div>
</div>
```

Run the MVC Application

Press F5 to run the MVC application

Click on your new link and your generated view should appear.

Congratulations! You created a complete List, Add, Edit and Delete page in just a matter of minutes.

Summary

In this chapter you learned how to quickly generate CRUD classes and put those classes into an MVC project and test out the generated classes and MVC pages.

NOTE:	The PDSA DLLs ONLY run in VS.NET when you are using the DEMO version of Haystack. In order to make them run from a website you need to purchase the full version of Haystack.

Chapter Index