Chapter 4

Quick Start - WPF

Table of Contents

Using Haystack Generated Code in WPF

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 WPF Applications

This document describes how to use Haystack to quickly generate a working **WPF 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.

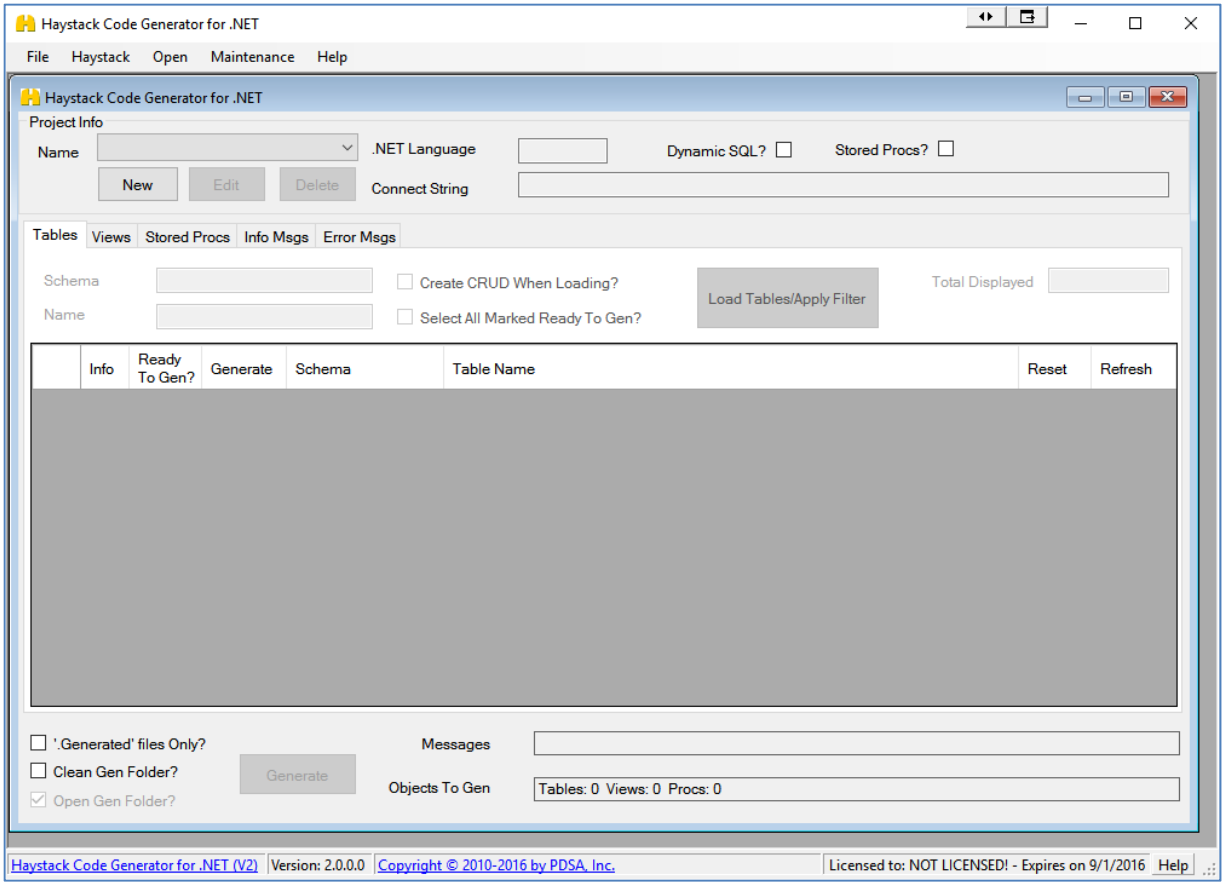

Figure 1: Haystack Main Screen

Add New Haystack Project for WPF

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

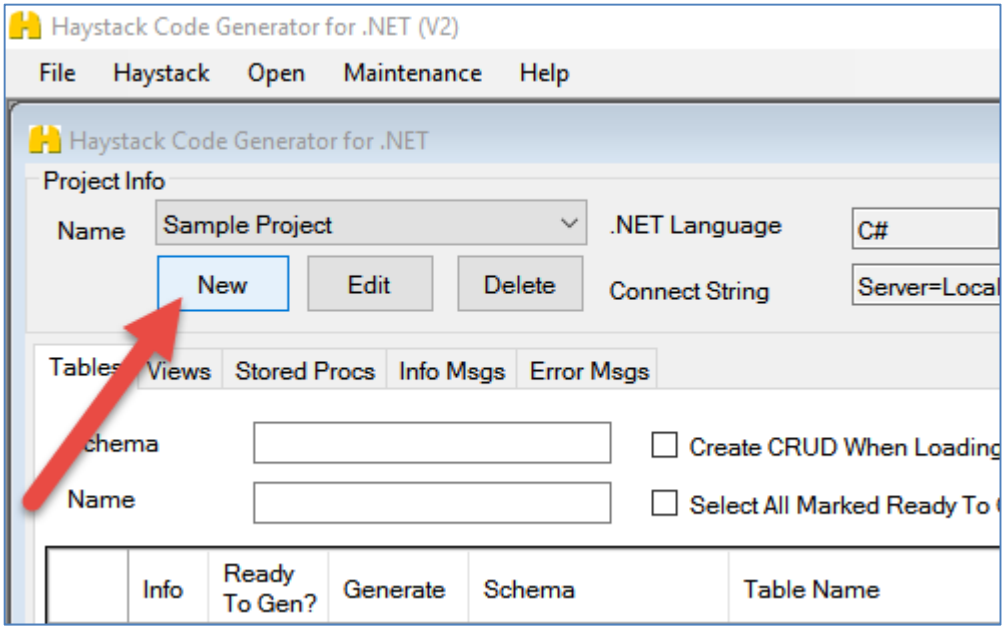

Figure 2: Click "New" to create a new Project

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.

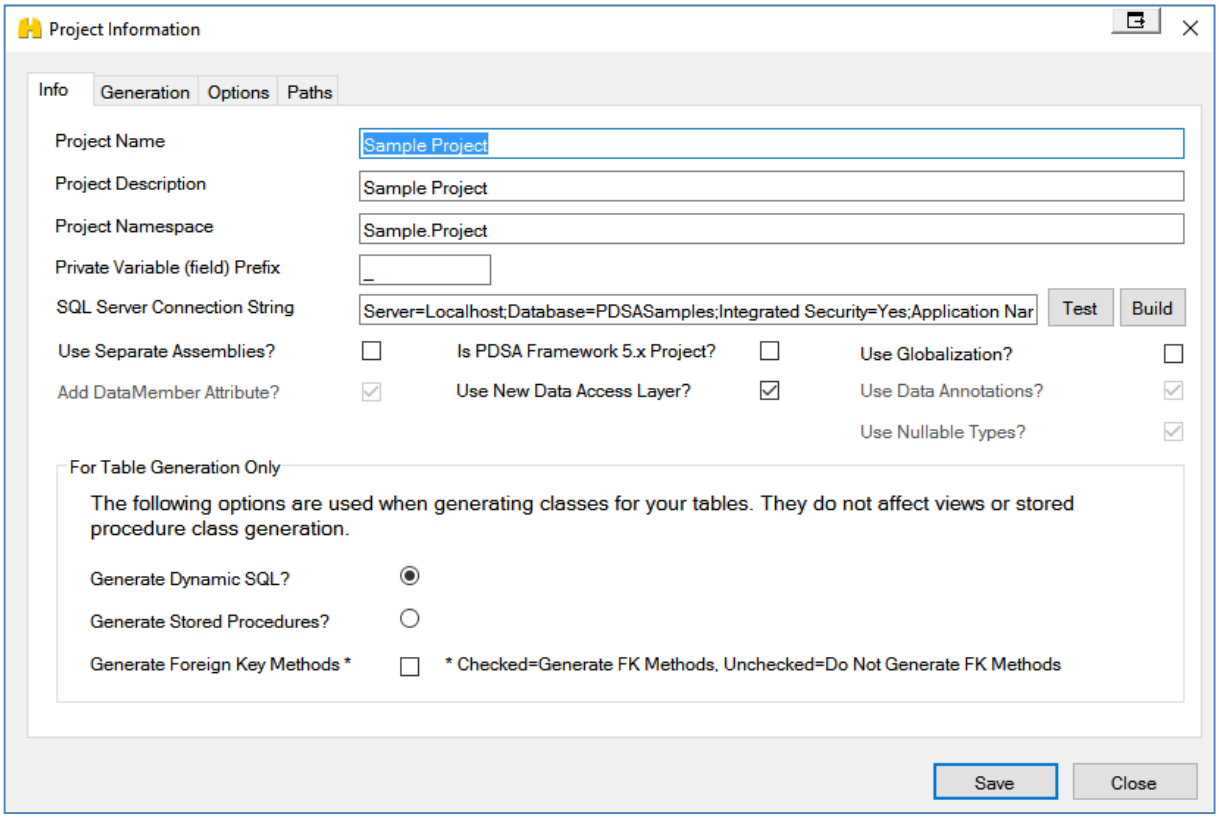

Figure 3: Project Information (Info Tab)

Click on the **Generation Tab** (Figure 4) and select the “WPF – Data Access Layer” template as shown in Figure 4: Project Information (Generation Tab)

.

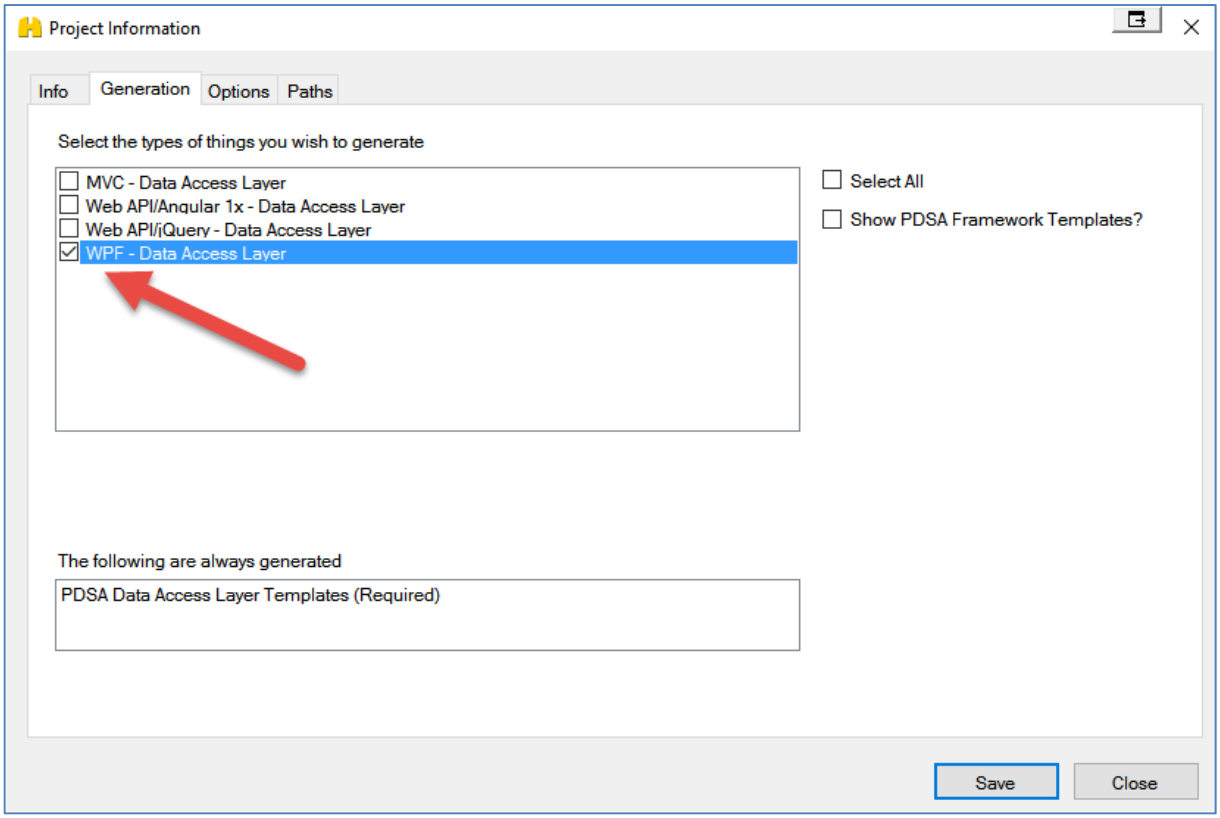

Figure 4: Project Information (Generation Tab)

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

Generate CRUD Classes for Tables - WPF

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

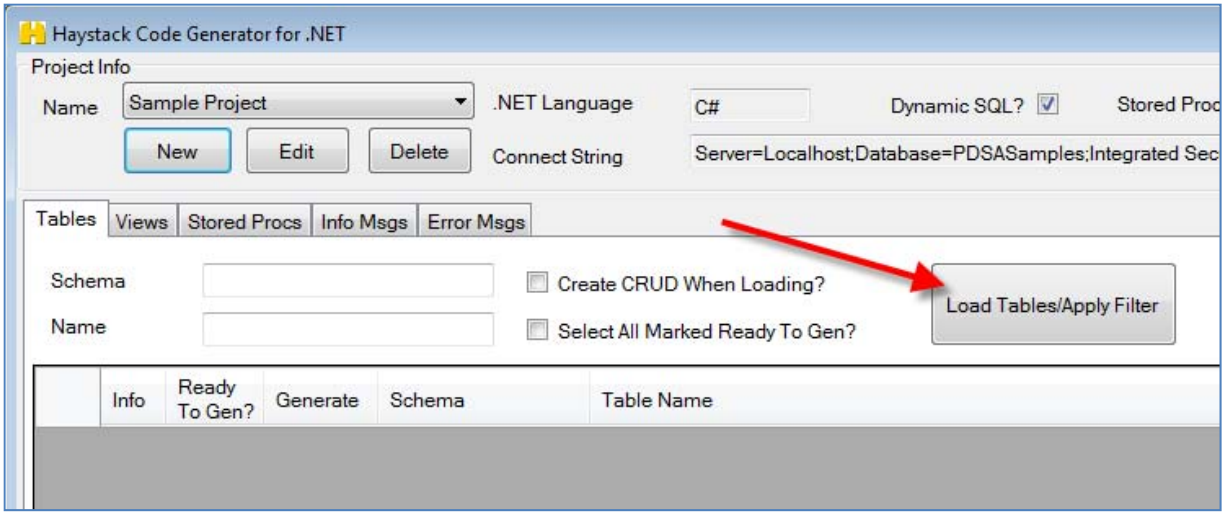

Figure 5: Load Tables

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 for WPF

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.

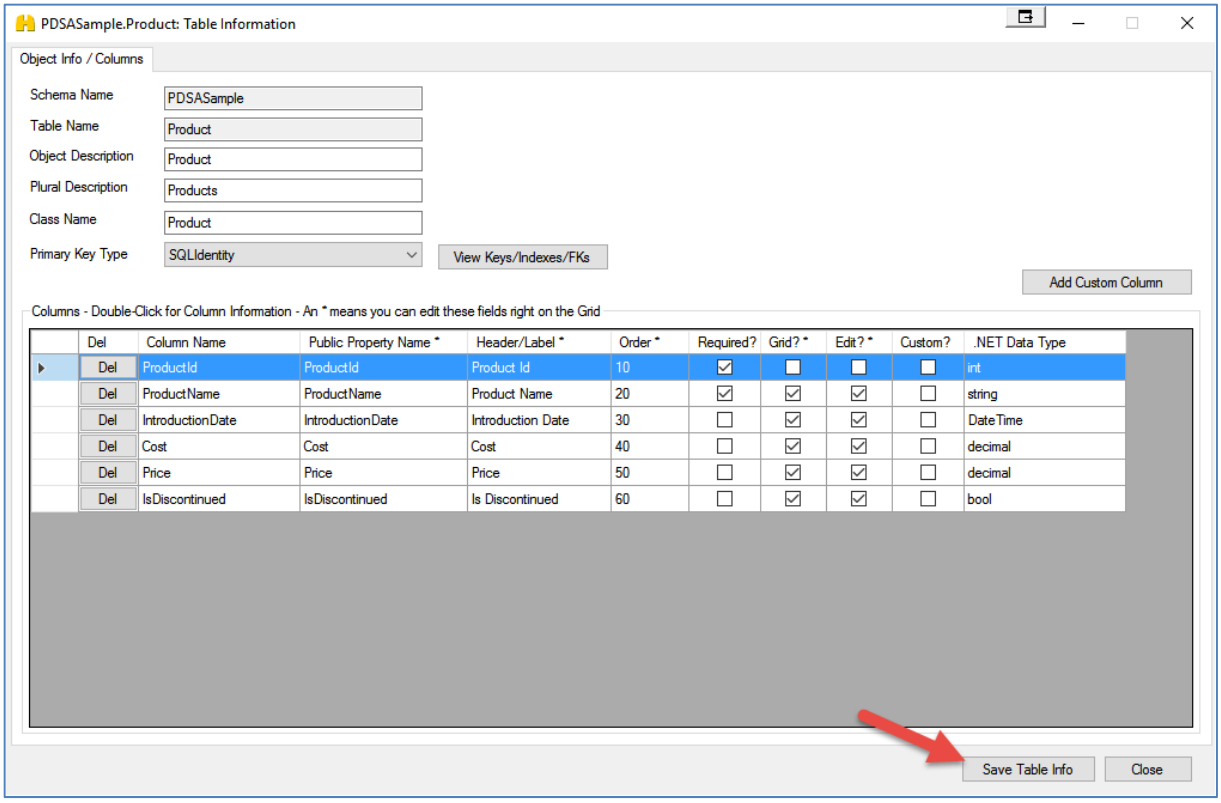

Figure 6: Table Information Screen

Generate Table Data Classes for WPF

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.

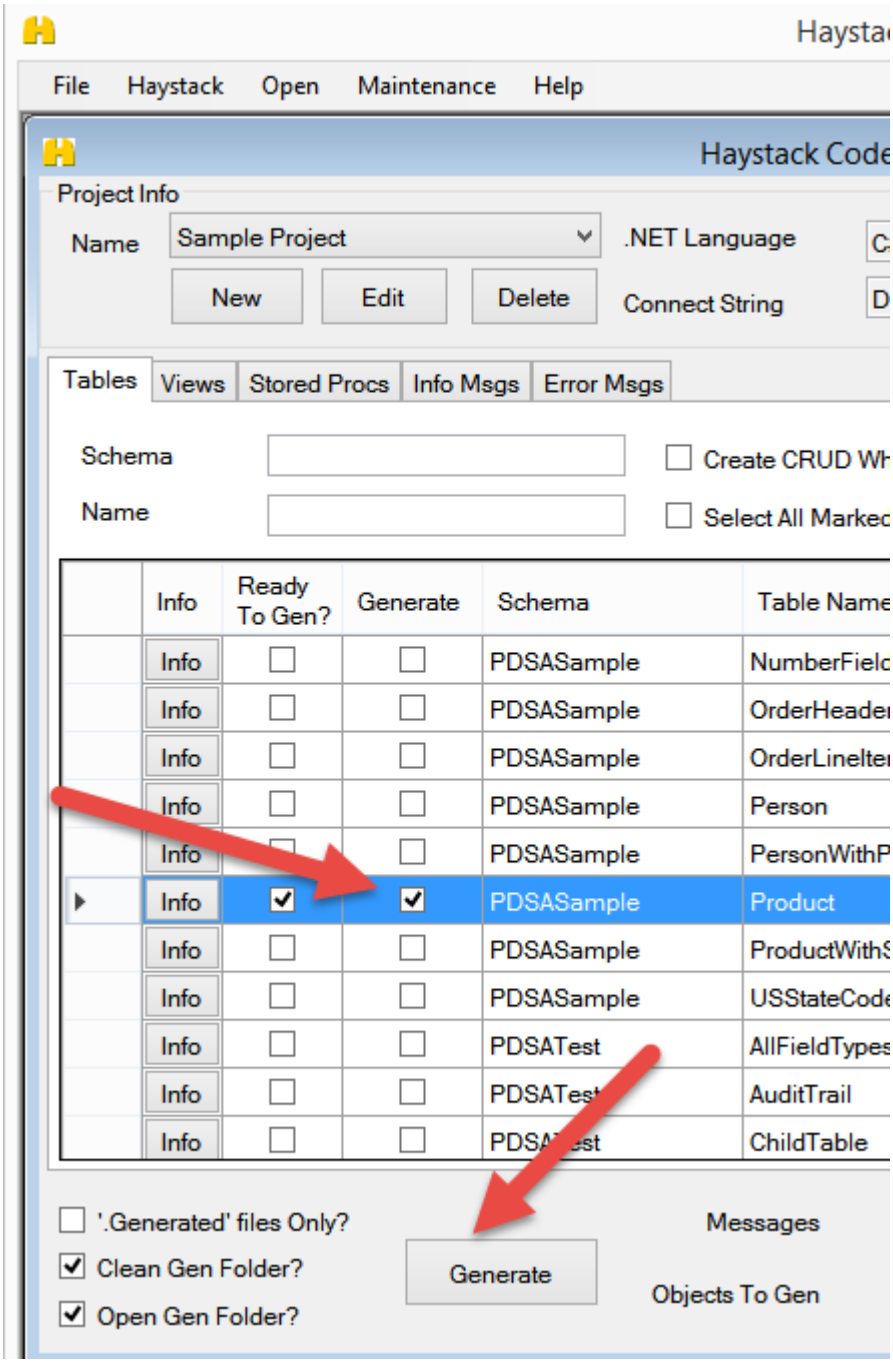

Figure 7: Generate Code

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

Create New WPF Application

Once the classes and user controls 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
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.

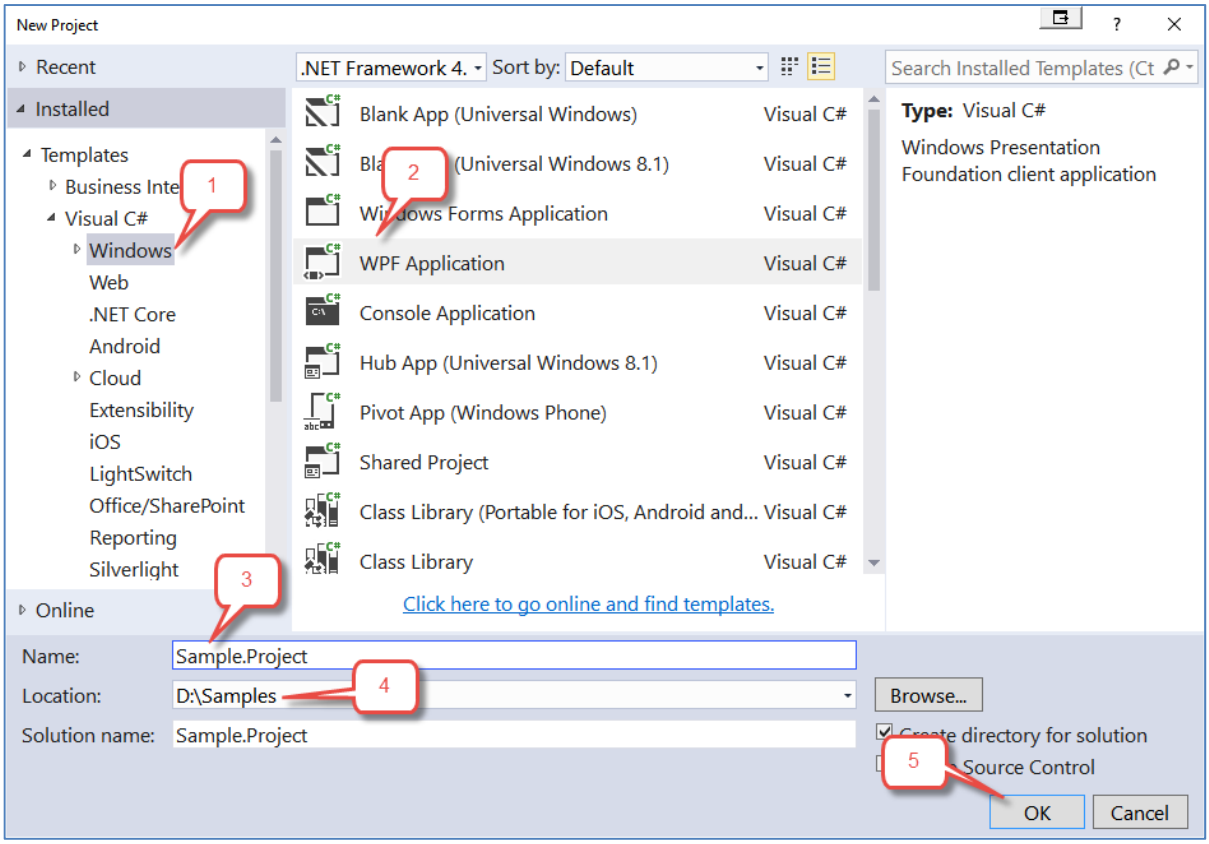

Figure 8: Create a new WPF Application

Copy Generated Files for WPF

In the Windows Explorer window (Figure 9) where the code was generated, expand the **WPF** folder and select all the folders and files and copy into the root of your new WPF project.

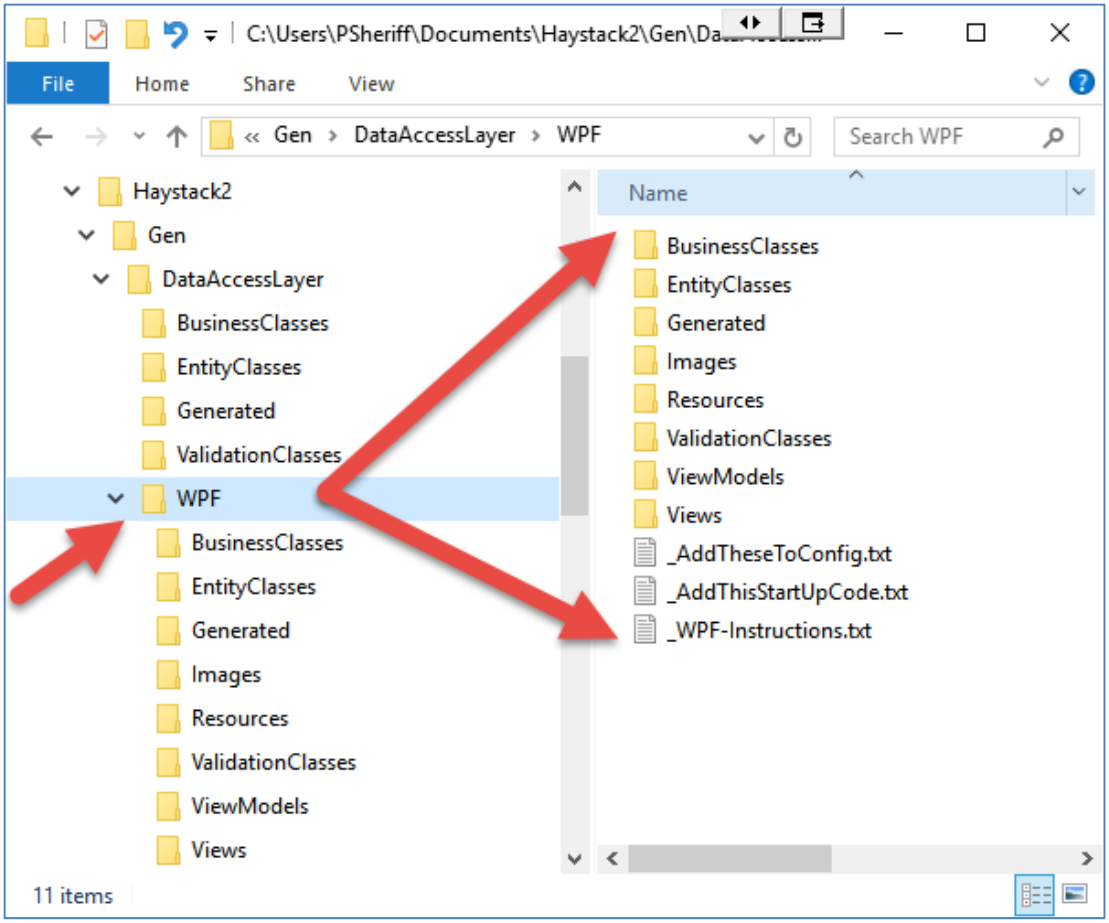

Figure 9: Copy the various views and view models into your WPF project

Your Solution Explorer window in your WPF application should now look like the Figure 10.

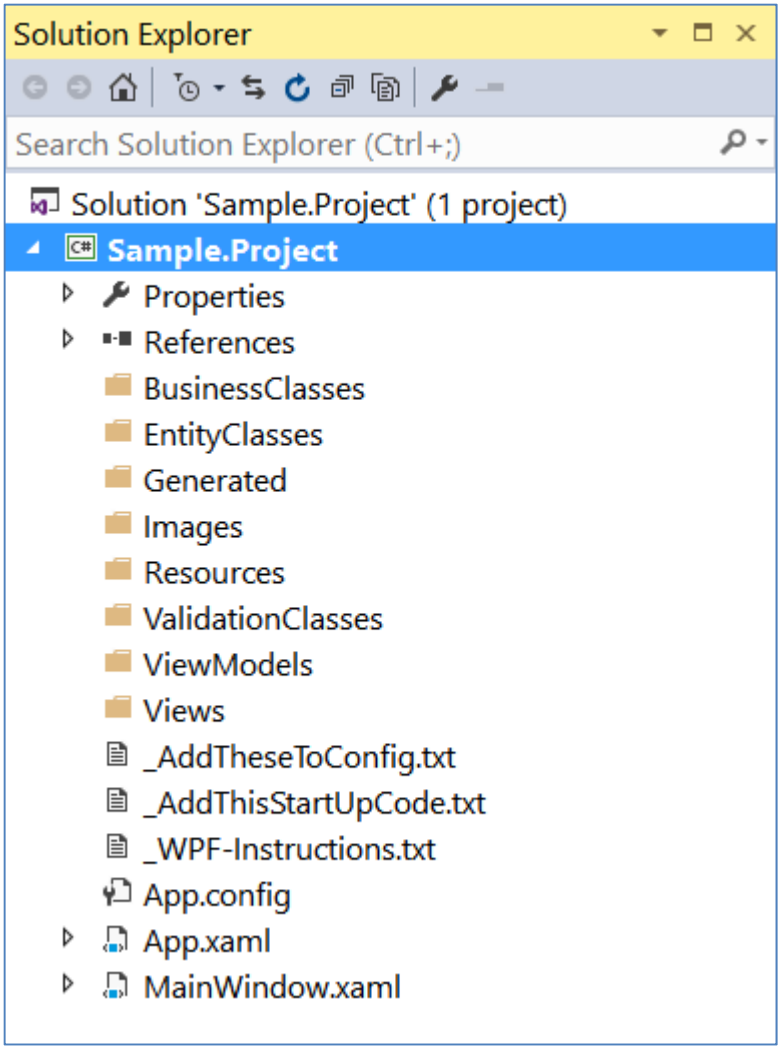

Figure 10: Your Visual Studio project after adding generated code

Include Files in WPF Project (VS Bug)

Click on the project in the Solution Explorer window, then click on the “Show All Files” icon as shown in Figure 11.

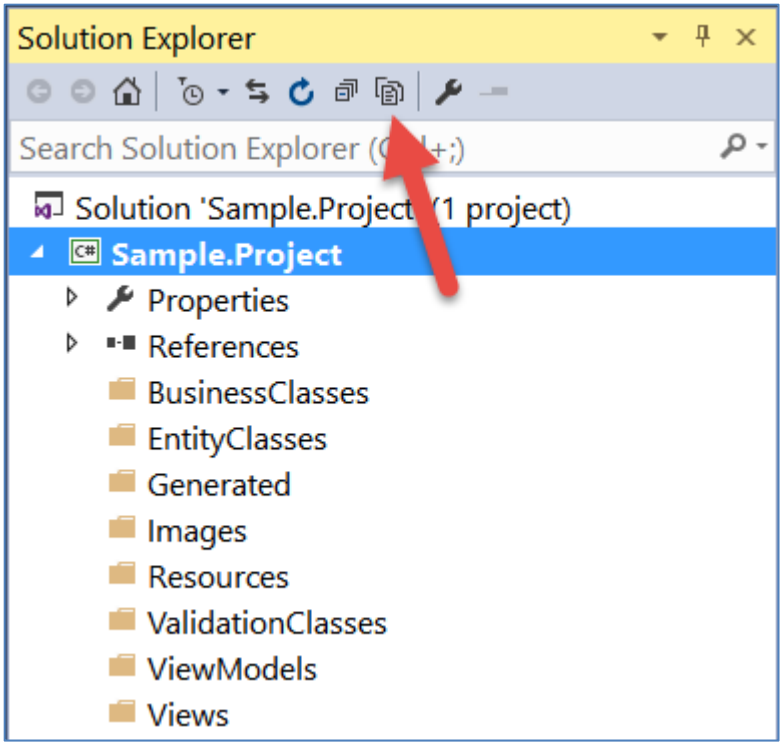

Figure 11: Click the Show All Files button to add the hidden files

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

- BusinessClasses
- EntityClasses
- Generated
- Images
- Resources
- ValidationClasses
- ViewModels
- Views

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.

Add DLLs to WPF Project

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-WPF** select the following four DLL files shown in Figure 12.

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

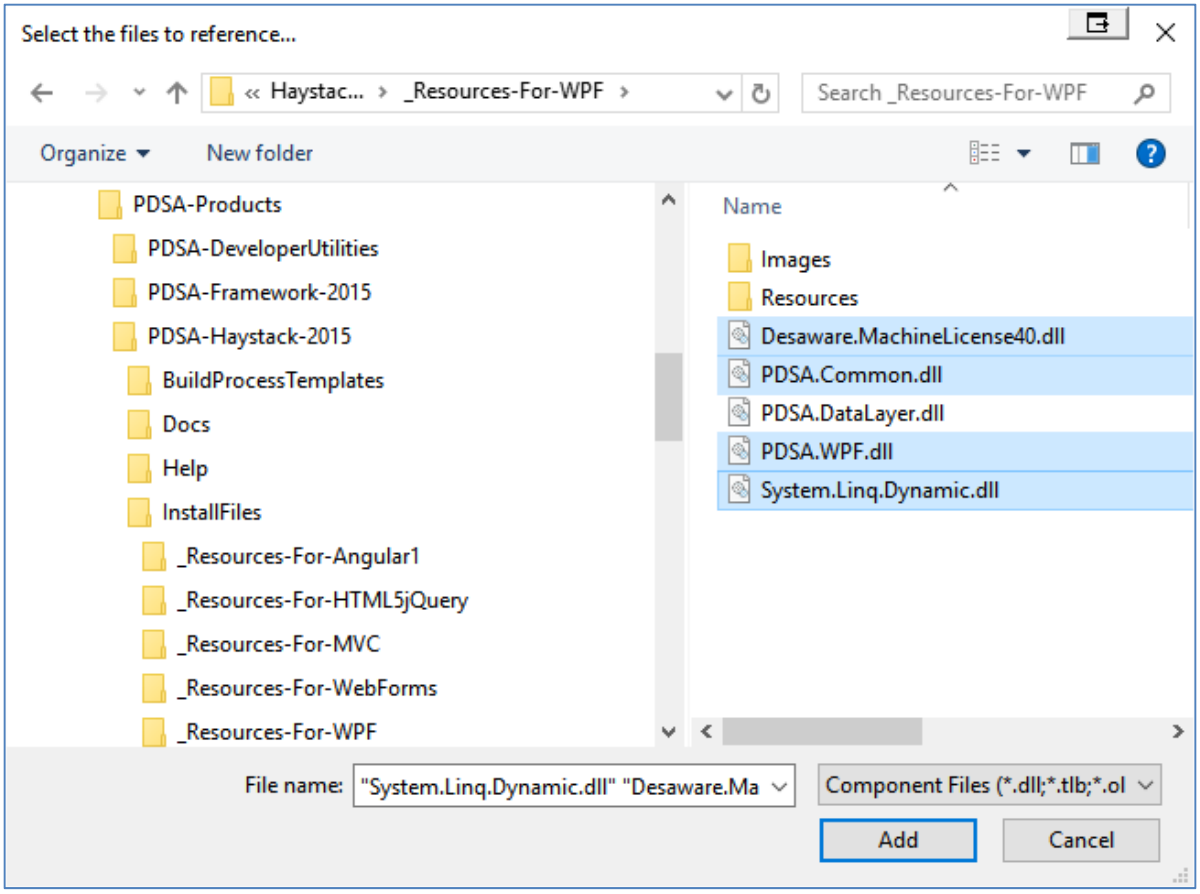

Figure 12: Reference the PDSA DLLs

Add three.NET Framework DLLs System.ComponentModel.DataAnnotations.dll, System.Configuration.dll and System.Runtime.Serialization.dll as shown in Figure 13 and Figure 14.

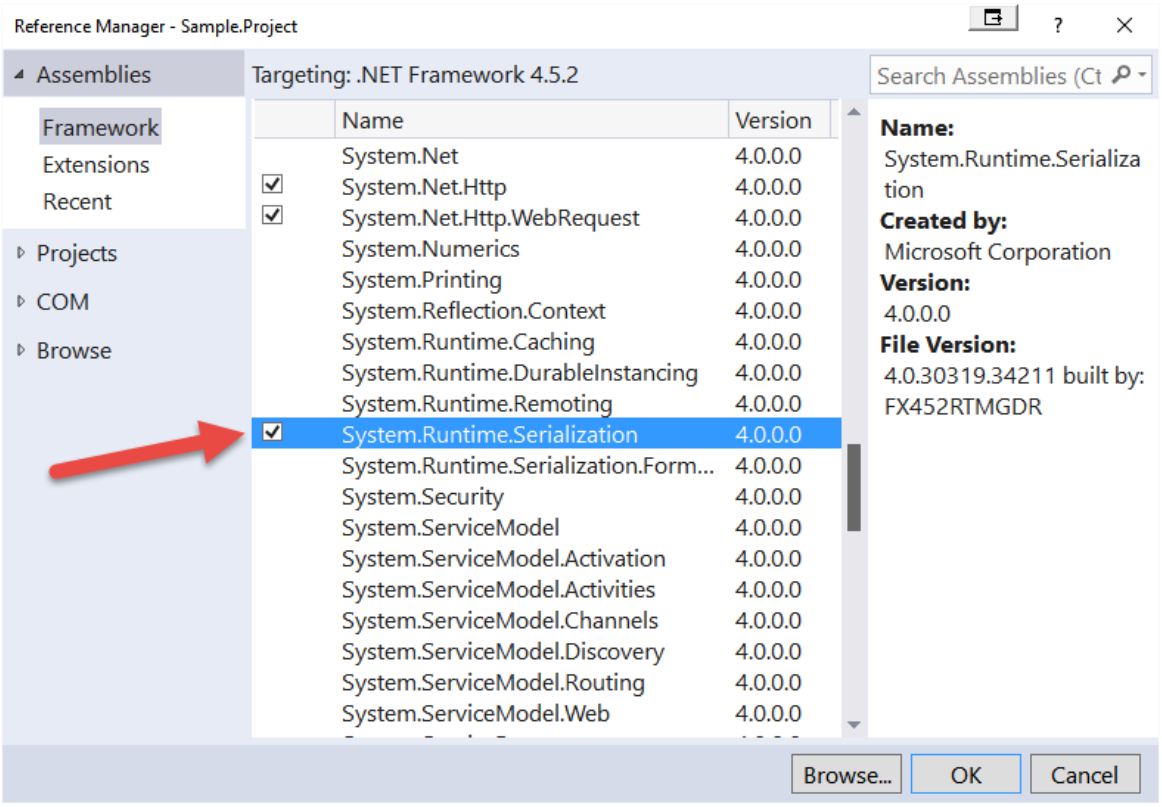

Figure 13: Reference the System.Runtime.Serialization DLL

Add a reference to System.ComponentModel.DataAnnotations.dll and System.Configuration.dll.

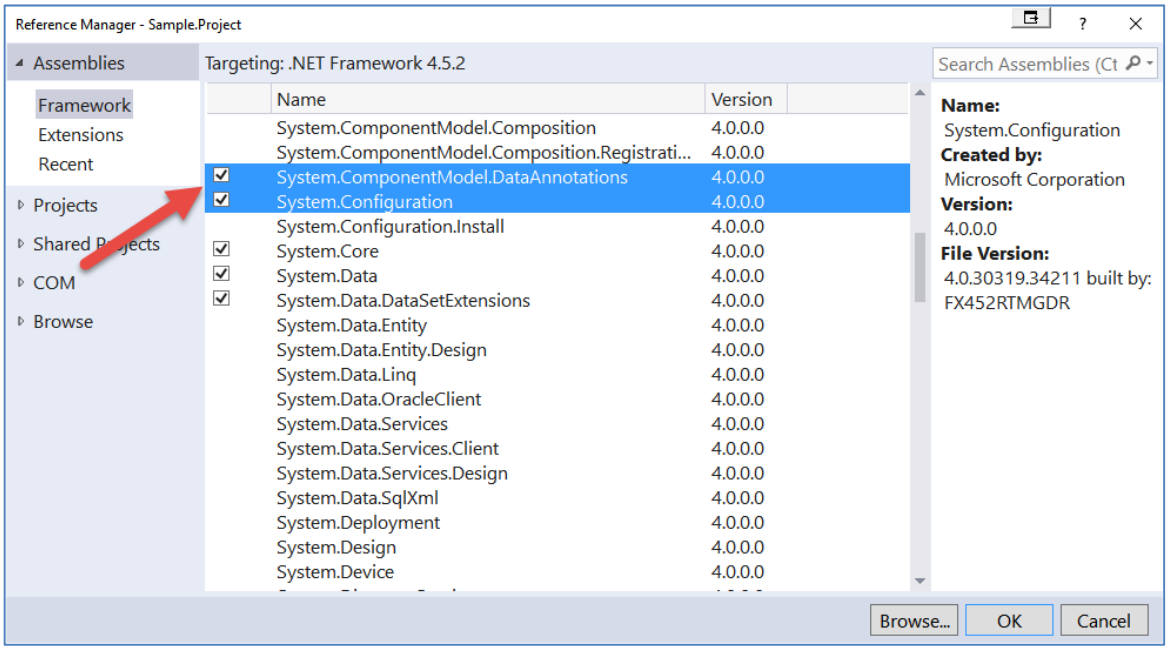

Figure 14: Reference the System.ComponentModel.DataAnnotations and System.Configuration DLLs

Fix up the App.Config File in WPF

You need to add a connection string to your App.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 entire <connectionString> element into your App.Config.

Delete the _AddTheseToConfig.txt file.

Modify the App.xaml.cs in WPF

To initialize the PDSA Data Access layer you need to add some startup code. Open the _AddThisStartUpCode.txt file and copy the entire contents to the clipboard. Open the App.xaml.cs file and paste this code within the App class. You will need to add a new 'using' statement 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.Data;
using System.Linq;
using System.Threading.Tasks;
using System.Windows;

namespace Sample.Project
{
  /// <summary>
  /// Interaction logic for App.xaml
  /// </summary>
  3 references
  public partial class App : Application
  {
    1 reference
    protected override void OnStartup(StartupEventArgs e) {
      base.OnStartup(e);

      // 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.

Add WPF User Control to Main Window

Select **Build** | **Rebuild Solution** from the Visual Studio menu.

Bring up the Toolbox (**View** | **Toolbox** on the Visual Studio menu) and you will see the WPF user control generated by Haystack as shown in Figure 16.

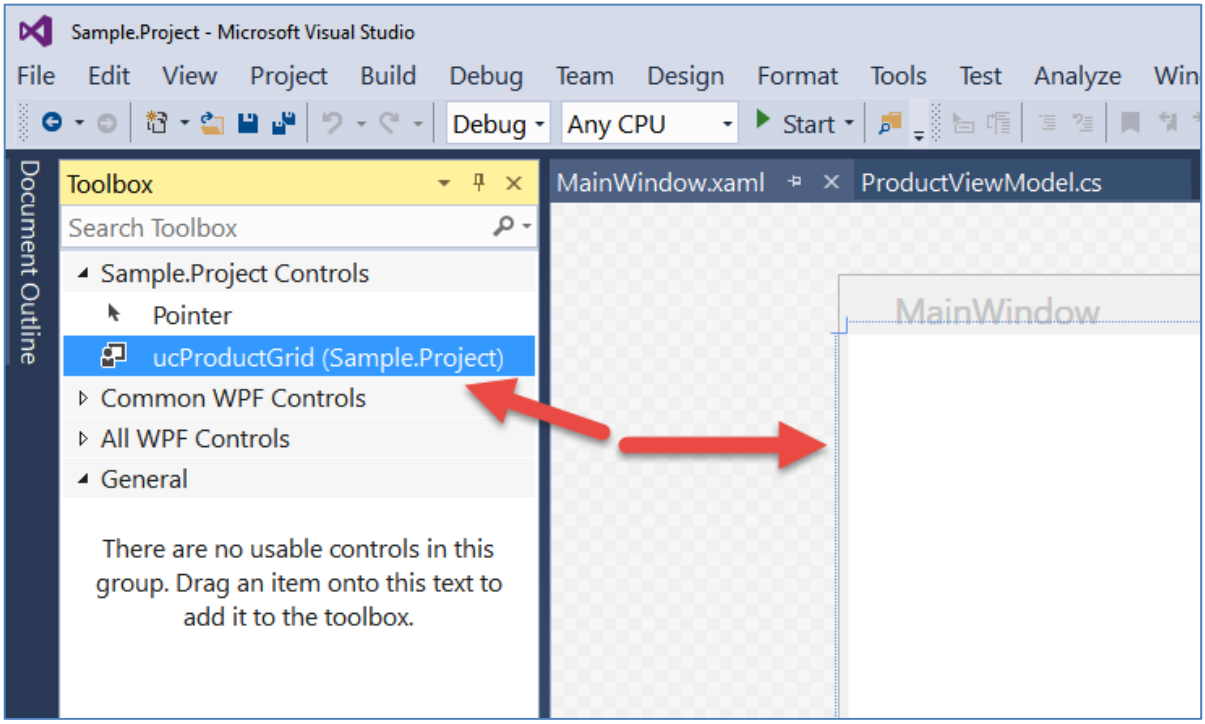

Figure 16: Drag and Drop one of your user controls onto your MainWindow.xaml

Drag and drop a control that control onto the MainWindow.xaml.

Right mouse click on the control and choose **Layout | Reset All** from the context menu.

In the MainWindow XAML view, delete the Height=”350” Width=”525” from the <Window> element as shown in Figure 17.

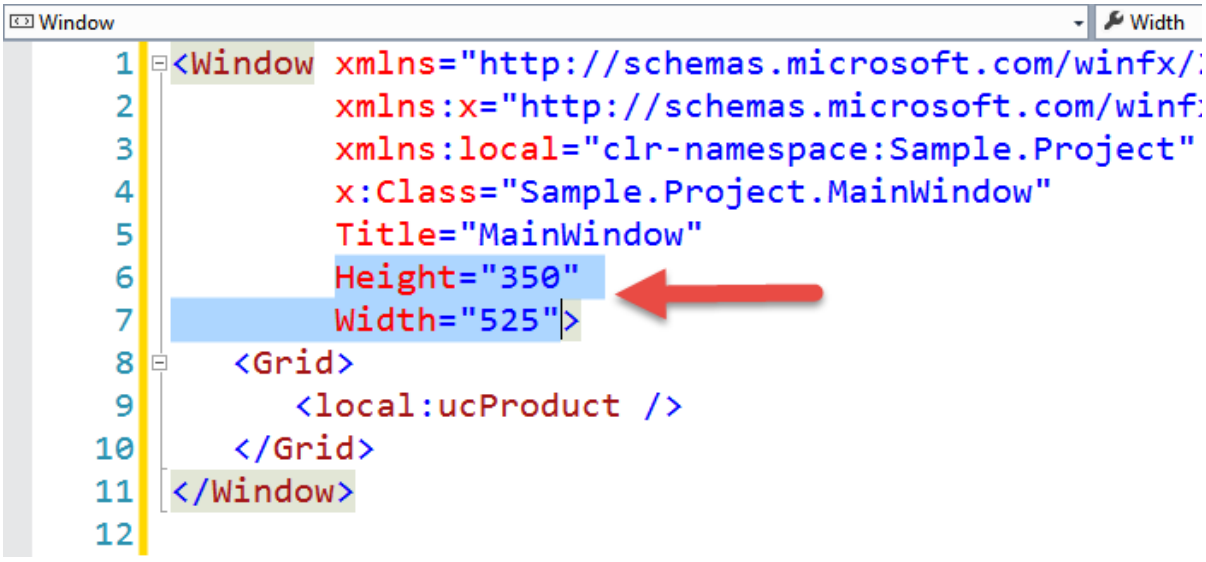

Figure 17: Eliminate Height and Width in the Window XAML.

Run the WPF application to see your generated WPF screen appear.

NOTE:	If you receive an error about a Namespace not being found, this is normally because the Namespace you created in the Project screen in Haystack does not match the name of the application you created. Simply perform a search and replace on this name to fix this.

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

Summary

In this chapter you learned how to quickly generate CRUD classes and a WPF user control, copy those classes into a WPF project and test out the generated classes and controls.

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 an .EXE file you need to purchase the full version of Haystack.

Chapter Index