An Architecture for WPF Applications

In this blog post, you learn to create a standard architecture for your WPF applications. You learn what common classes you need, what kind of library to put those classes into, and how each of the libraries are referenced from your main application.

Architecture Overview

Designing an architecture for a WPF application is like any other kind of application you build. You always strive to make sure classes and libraries of classes are as reusable and as stand-alone as possible. This makes the maintenance of applications easier, and also increases their reusability and testability. Figure 1 shows a breakdown of the different class libraries you would have to support for each WPF application you build.

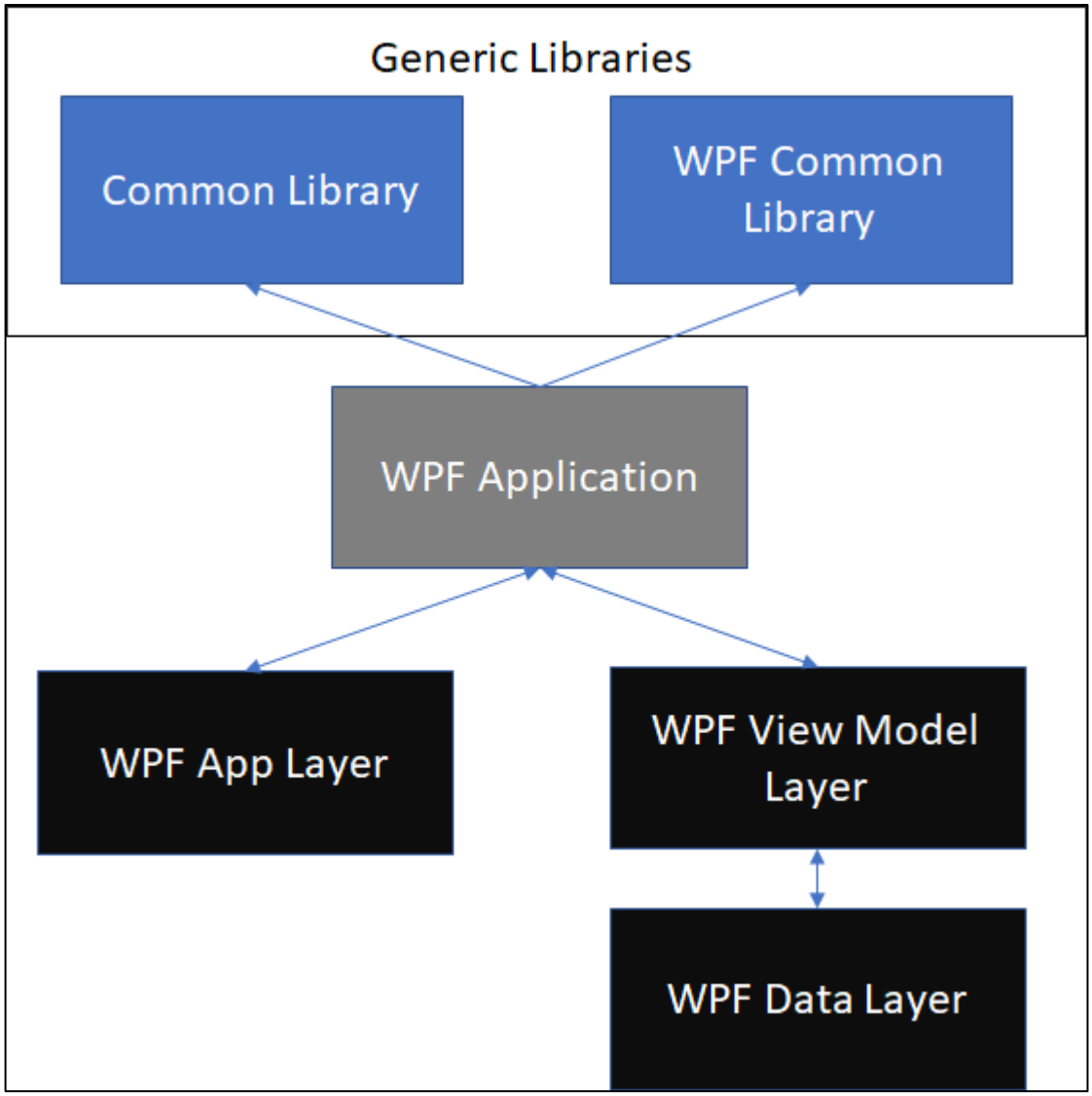

Figure 1: Overview of a WPF Architecture

If you were to implement the architecture shown in Figure 1 in a .NET solution, the results would look like Figure 2.

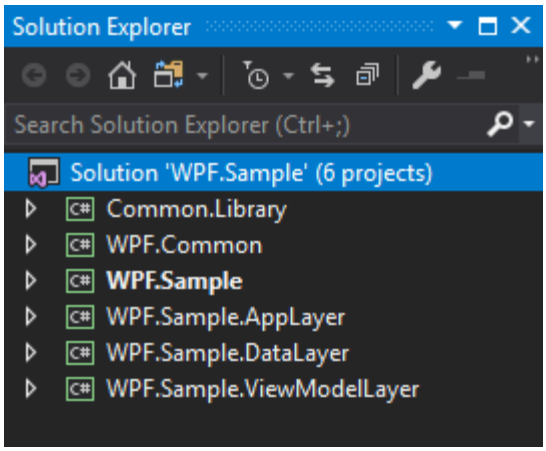

Figure 2: A sample WPF solution

A description of each project and what each one is used for is shown in Table 1.

Project	Description
Common.Library	This is a class library project into which you can add classes and other items that are UI agnostic (i.e. these classes can be used in any type of project such as Windows Forms, Web Forms, MVC, Web API, WPF, etc).
WPF.Common	This is a WPF library project into which you can add classes, user controls, resource dictionaries, converters, and other XAML to be used in any WPF application. Keep the items in this project generic and able to be used in any WPF application.
WPF.Sample	This is a sample WPF application that shows how all of these projects are connected together.
WPF.Sample.AppLayer	This is a WPF library project into which you can add any WPF classes, user controls, resource dictionaries, etc. that are unique for this WPF application.
WPF.Sample.DataLayer	This is a class library project into which you can put any data access classes you need to support this project. The Entity Framework (EF) library is already added into this project to make it easy for you to add your own EF entity and model classes.
WPF.Sample.ViewModelLayer	This is a class library project into which you can put any view model classes you need to support this WPF project. View model classes are used to bind to the UI and insulate the WPF project from the data access layer.

Table 1: Description of each project in your WPF sample solution

Let's look at each project and the types of classes you should be putting into each of these projects.

Common.Library Project

The sample Common.Library project (Figure 3) that comes with this blog post, illustrates a common set of classes that can be used in any .NET application. The types of classes you place into this class library project should NOT have any dependencies on a specific UI such as Windows Forms, WPF, or ASP.NET.

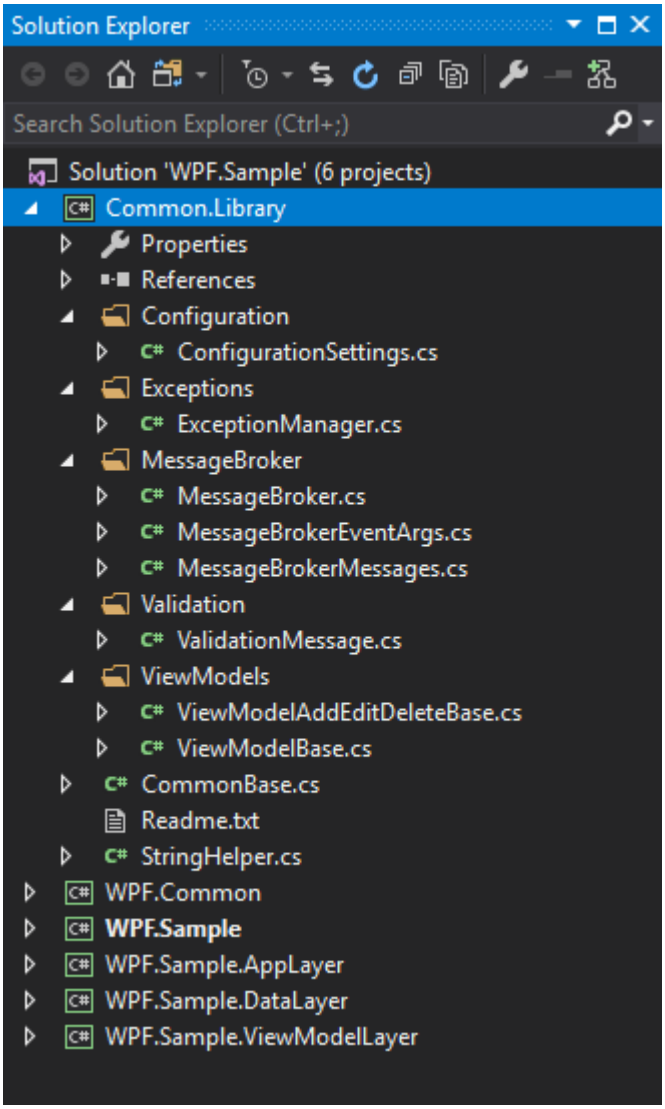

Figure 3: The Common.Library Project

Each class in this Common.Library project is described in Table 2.

Class	Description
ConfigurationSettings	A class for reading settings from a configuration file.
ExceptionManager	A class for publishing exceptions.

MessageBroker	The main class that sends the messages and defines the event signature for receiving messages. Using a message broker helps you avoid strong coupling between components in your applications.
MessageBrokerEventArgs	The MessageBrokerEventArgs class inherits from the System.EventArgs class and adds a couple of additional properties needed for our message broker system. The properties are *MessageName* and *MessagePayload*. The *MessageName* property is a string value. The *MessagePayload* property is an object type so that any kind of data may be passed as a message.
MessageBrokerMessages	This class contains public constants that can be used for sending standard messages such as "CloseControl," "DisplayStatusMessage," etc. Instead of repeating strings throughout your application, it is better to use constants within a class.
ViewModelAddEditDeleteBase	A base view model class for screens in which you list, add, edit, and delete items from a database table. This class inherits from the ViewModelBase class.
ViewModelBase	A base view model class for all your view models. This class inherits from the CommonBase class.
ValidationMessage	This class contains two properties; *PropertyName* and *Message*. This is used to report any validation rules that fail.
CommonBase	This class implements the INotifyPropertyChanged event. This class also implements a Clone() method used to copy all properties from one object to another and forces each property to raise its INotifyPropertyChanged event.
StringHelper	A class for you to put any methods to help you work with strings. There are a couple of methods in here such as CreateRandomString(), IsValidEmail(), IsAllLowerCase() and IsAllUpperCase().

Table 2: Description of each item in the Common.Library project

WPF.Common Library

The WPF.Common class library (Figure 4) is created as the WPF User Control Library project from the Visual Studio list of project templates. Using these templates includes the DLLs necessary to support WPF user controls, resource dictionaries, converters, and other WPF-specific items.

Where the Common.Library is UI-agnostic, the WPF.Common library is for you to add any component necessary to support any WPF application you develop. Keep the components in this library generic so you can use it with any WPF application, and not just the one you are currently developing. Any components that are specific to your current project belong in the "WPF" or the "AppLayer" projects.

I have included a few folders to illustrate what you might add to this project. The **Converters** folder is where you put any data type converter classes. The **Resources** folder is where you put resource dictionaries. The **UserControls** folder is where you put any commonly used user controls.

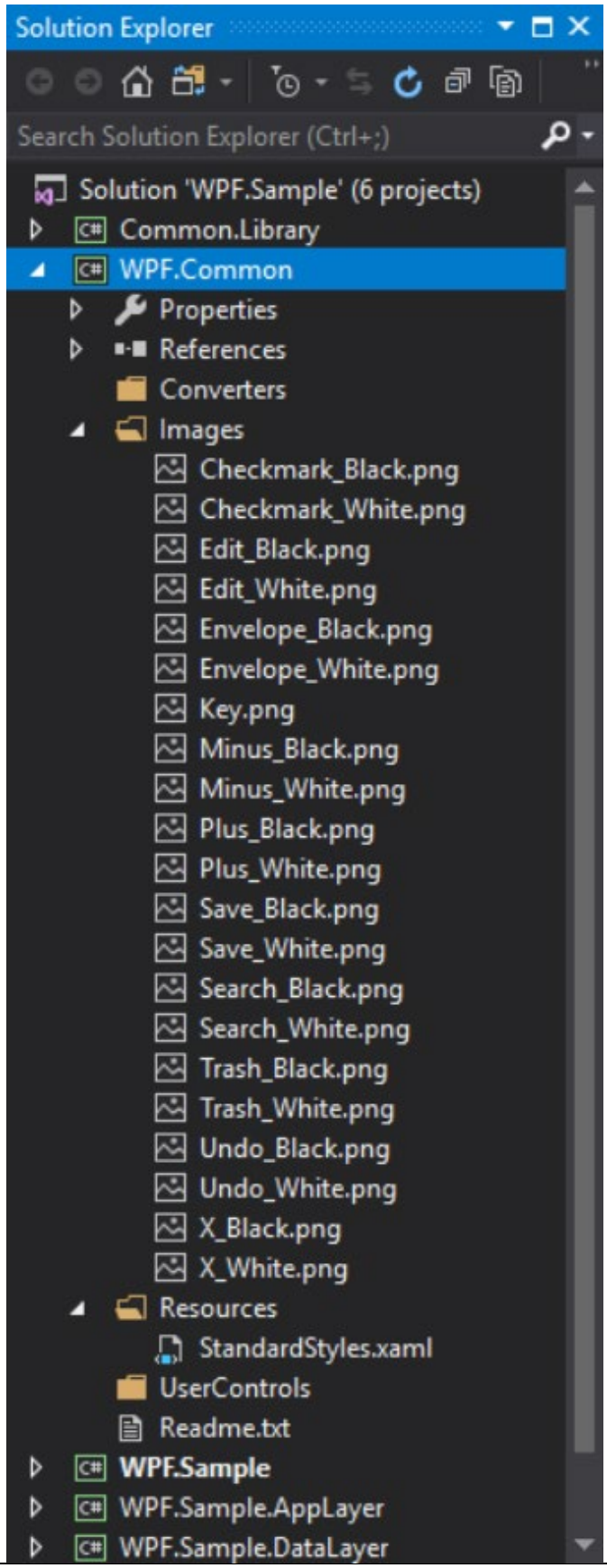

Figure 4: The WPF.Common Project

Each item in this WPF.Common project is described in Table 3.

File	Description
\Converters folder	A folder where you can put any WPF converter classes
\Images folder	A folder of some standard images you might find useful.
StandardStyles.xaml	A resource dictionary of styles.
\UserControls	A folder for any reusable user controls

Table 3: Description of each item in the WPF.Common project

WPF.Sample Project

This project (Figure 5) is a starting point for the WPF application you are creating. I have included a MainWindow with a menu system, some images, and a sample SQL Server express database with a table in it. You will normally take this project and start adding on your own user controls to display within the main window.

In future blog posts, you are going to use this project to build some standard business screens.

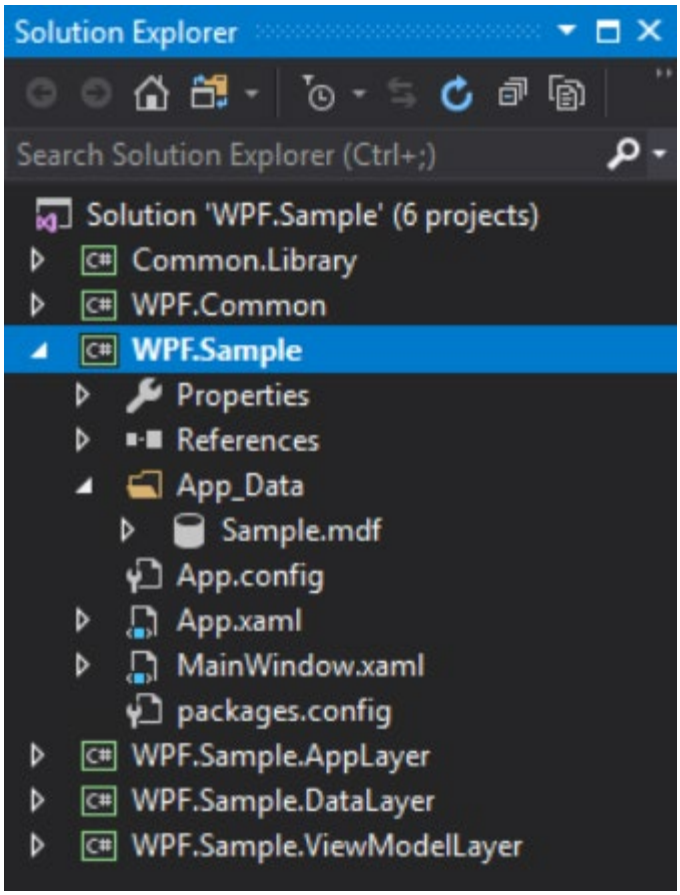

Figure 5: The WPF.Sample Project

Each item in this WPF.Sample project is described in Table 4.

Item Name	Description
Sample.mdf	A SQL Server express database that contains a single User table.
App.config	The configuration settings for this application.
App.xaml	Any styles for this application. The App class contains code to set the Domain property DataDirectory and reads all configuration settings for this application.
MainWindow.xaml	The main window for this application.

Table 4: Description of each item in the WPF.Sample project

WPF.Sample.AppLayer Library

The WPF.Sample.AppLayer library (Figure 6) is where you can add classes that are specific to this application.

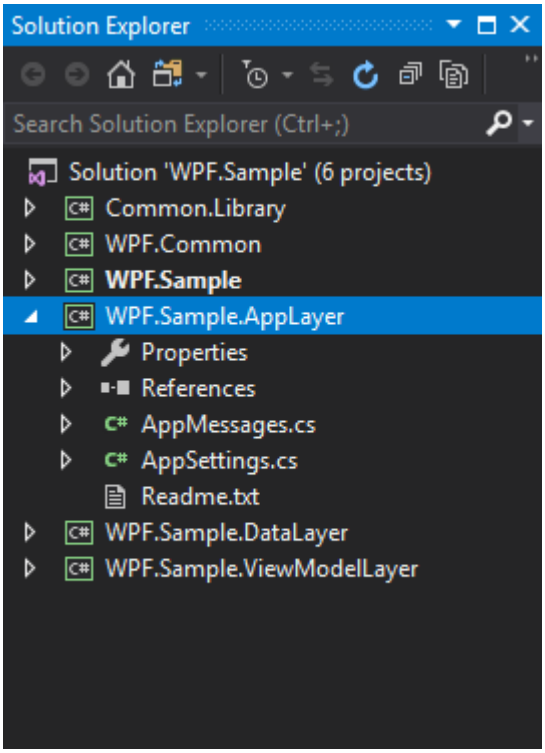

Figure 6: The WPF.Sample.AppLayer Project

Each item in this WPF.Sample.AppLayer project is described in Table 5.

Class Name	Description
AppMessages	This class inherits from the MessageBrokerMessages class. Add any application-specific messages you need for your WPF application.
AppSettings	This class inherits from the ConfigurationSettings class in the Common.Library project. Place any properties for any global settings you need for your WPF application. Also write code to load those settings from the App.config file into the LoadSettings() method.

Table 5: Description of each item in the WPF.Sample.AppLayer project

WPF.Sample.DataLayer Library

You should always keep your data access classes in a separate library. This helps you reuse these classes in any other application should you need to. The WPF.Sample.DataLayer library (Figure 7) uses the Entity Framework to access the Sample.mdf file contained in the WPF.Sample project. You may use any data access technology you want in this project.

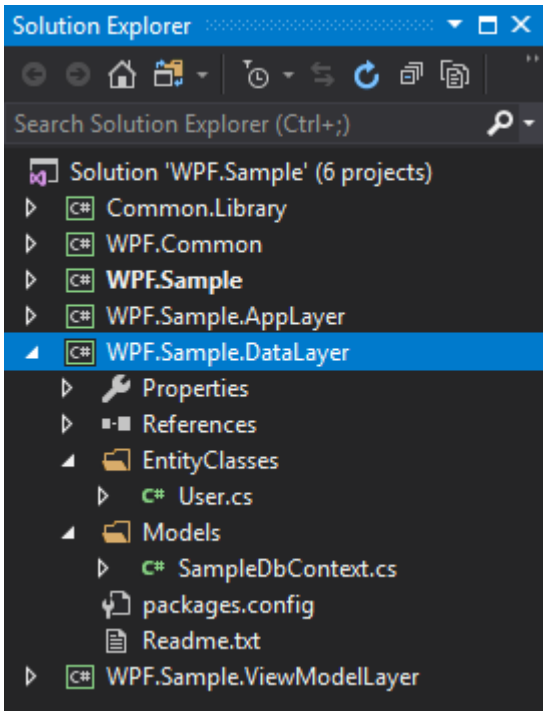

Figure 7: The WPF.Sample.DataLayer Project

Each item in this WPF.Sample.DataLayer project is described in Table 6.

Class Name	Description
User	This is a simple entity class that models each column in the User table contained in the Sample.mdf database.
SampleDbContext	This class inherits from the DbContext class from the Entity Framework. It creates a DbSet<User> property to allow you to access the data in the User table.

Table 6: Description of each item in the WPF.Sample.DataLayer project

NOTE: This project is for illustration purposes only. Feel free to use any data access mechanism you want.

WPF.Sample.ViewModelLayer Library

You should always use the MVVM design pattern when creating WPF applications. All your view models for each of your screens should be put into the WPF.Sample.ViewModelLayer class library (Figure 8). Add additional view model classes to this project as you add new screens to your WPF application.

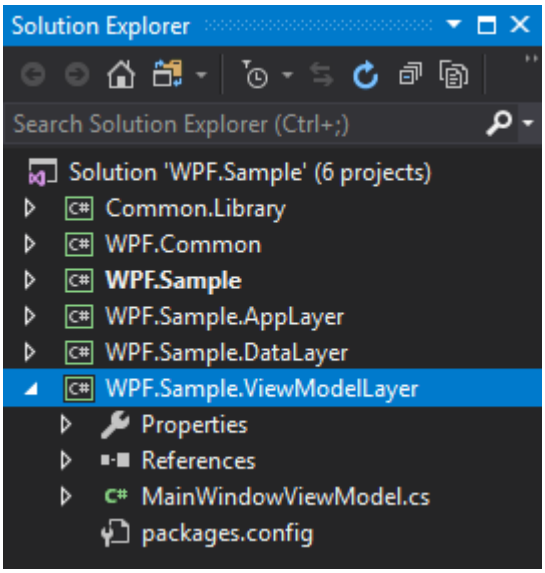

Figure 8: The WPF.Sample.ViewModelLayer Project

Each item in this WPF.Sample project is described in Table 7.

Class Name	Description
MainWindowViewModel	This is the view model for the main window in the WPF application.

Table 7: Description of each item in the WPF.Sample.ViewModelLayer project

Summary

It is important to create a good starting architecture prior to writing any application. Use "Separation of Concerns" as the guiding principle when writing applications and you will never go wrong. Feel free to add more layers to the sample outlined in this blog post. Also feel free to add additional classes to each of the projects as needed. The sample outlined should give you a good head-start on building reusable, maintainable, and testable WPF applications.

NOTE: You can download the sample code for this article by visiting my website at http://www.pdsa.com/downloads. Select “Fairway/PDSA Blog,” then select “An Architecture for WPF Applications” from the dropdown list.