# Introduction (5 minutes or less)

Introduce me and what I am going to talk about in some general terms.

Hi every one, welcome to the Prism presentation. First a little about me: my name is Mike Ponti. I have been working in a number of software and IT roles over the last 20+ years. With respect to software development, I have worked on a wide variety of projects: ERP integrations, WebAPI and activex, CAD addins, enterprise productivity apps, asp.net webform applications, asp.net mvc, some windows phone apps, a lot of win32 WPF applications, lately some windows 10 UWP apps and finally some global dev ops.

One of the open source libraries that I have come to find very useful, is a library called Prism. I use it on pretty much any xaml type app. When Mark asked for volunteers on presenting, I thought this would be a good topic for the group.

A bit about Prism. Prism was originally developed by the Microsoft Patterns and Practices team to provide a framework for developing XAML apps in WPF, Silverlight and Windows Phone. It went through a number of releases finally reaching release 5 and supporting Windows 8, 8.1 for both Silverlight and Windows Runtime.

After version 5, Microsoft Patterns and Practices turned it over to the open source community and is now maintained on GitHub. Along the way it has deprecated support for the various Windows 8, 8.1 platforms. It still actively supports WPF apps and has introduced support for the UWP platform and Xamarin Forms as well.

Regardless of the target platform, Prism encourages a number of design patterns within XAML apps to make your app loosely coupled and easily maintainable. Patterns such as MVVM, dependency injection, event aggregation and others. Depending on your project, it also implements a number of other platform-specific patterns to help with your app. In the case of UWP, it helps with navigation, session state, gestures and others.

# MVVM

Is everyone familiar with MVVM?

What is MVVM? Wikipedia says:

A software architectural pattern that facilitates the separation of the user interface presentation (View) from the business data (Model). This separation is handled by the ViewModel (VM).

That is a pretty high-level view of what is going on. In the case of Xaml apps, you tell your pages to bind to a .NET class, called the viewmodel. Your viewmodel class will implement the INotifyPropertyChanged interface. When it does that, changing the value of a control property, such as the text in a textbox, is done in your class and through the binding, is reflected in the view. Using this type of architecture means you can keep your business logic separated from your view. And it means that your business logic will become a lot easier to unit test.

# Dependency Injection

Dependency injection is a software design pattern where a central object or container, is used to create other objects. What makes it special, is that container uses reflection to load all of the required dependencies into the object that is being created. For example, you could have a view model that requires access to a list of records. When the view model is created, reflection shows that the service to retrieve the records is required and it is automatically generated from the container and injected into the viewmodel. The viewmodel has no knowledge of how that service is created. Furthermore, if the service were to be defined as an interface, you could swap implementations of the service without ever having to change the viewmodel.

# Event Aggregation

Event aggregation is a method in Prism that is used for communication between different objects. You define events (and their data), and other objects can subscribe to these events and publish to them. Using this as a mechanism to communicate between objects means that you never have to call an object jdirectly, and any number of objects can get the message without the originator being concerned.

An easy example of this would be having objects subscribe to settings changed event. The object that is in charge of the settings would publish the new settings and every object that is interested in the settings would subscribe to the event. The beautiful thing is that the object in charge of the settings has no idea of who is interested in the settings and is just happily working away in isolation. If you add another object that needs the settings, it just subscribes and nobody else needs to know.

# So Why Use Prism?

While it is great to do everything from scratch to learn how a platform functions, this can get pretty old after a couple of times. There is a lot of code that is really the same from app to app. I like to use Prism because it has a lot of that common functionality already built up and ready to use in your app by simply following some conventions. It also uses those handy patterns that I talked about previously to make your app easy to test and easy to maintain.

But, there is still some work to be done, especially with setting up the hamburger style navigation. But once you have something setup, you can export your project as a template and reuse it on future projects.

It’s also not a bad idea to try out some of the other frameworks available. Another good one that I have used on the desktop side is Galasoft.MvvmLight. There is also Template10 and Caliburn.

# UWP Application and Frame Objects Background

Has anybody written a UWP app before? A little bit of background on it.

The application object is the main object of a UWP program and also the entry point for a number of program conditions. You should handle launching and suspending.

Upon starting up the application, the OnLaunched method is called. It is the method that performs the app initialization and then shows the first page of the app. To start, the app frame is setup. You can think of the Frame as the object that contains your app’s pages. Not only does it contain all of the pages, but it also performs all of the navigation for the app. When the OnLaunched event is called, the Frame object is either retrieved or created depending on how the application is started up. Your app needs to make use of the Navigate method of the Frame object to move between the different pages.

If the app has to suspend for some reason, the OnSuspending method is called. It is there you can perform any operations needed to save data or preserve state. As this happens at the application level, it can be tricky to pop that up to your current page and handle it appropriately.

Out of the box, the UWP app template doesn’t give you a whole lot. It creates the application object and the Frame, and one page. Upon startup, it navigates to the main page. Other than that, you are on your own. It is up to you to figure out how you want to architect your app so that navigation will be available, handle gestures, session state etc.

# App Introduction

In the presentation, we are going to make a very simple to-do app. We will use the Prism Framework and convert the standard template over into a hamburger style navigation app. We will setup pages for listing, adding, editing and deleting items. We will take a look at the navigation facilities in Prism and how they work with session state and make a custom navigator to better work with hamburgers.

Links:

Source code: <https://github.com/PrismLibrary>

Patterns and Practices: <https://msdn.microsoft.com/library/ff921345.aspx> and <http://compositewpf.codeplex.com/>