MVVM architecture in Energy Filter Monitor

.NET software architecture/design proposal

#### Document history

|  |  |  |
| --- | --- | --- |
| Contributor | Date (YYYYMMDD) | Description |
| Panagiotis Georgiadis | 20190627 | Document creation |
|  |  |  |

#### References

|  |  |  |
| --- | --- | --- |
| 1 | [[blogs.msdn.microsoft.com]](https://blogs.msdn.microsoft.com/ivo_manolov/2012/03/17/model-view-viewmodel-mvvm-applications-general-introduction/) | Model-View-ViewModel (MVVM) Applications: General Introduction |
|  |  |  |

# Introduction

With this model we aim to set guidelines for refactoring the Energy Filter Monitor application into an MVVM-based architecture.

## Initial design

Initially, the Energy Filter component was supported by the **CefidTestApp** application.

Upon further development, the **Energy Filter Monitor** application was created, which provides a more advanced UI for service/factory users to control the filter.

The Energy Filter Monitor application is a WPF-based implementation with a simplistic design that couples the UI elements with energy filter data in code-behind (C#). The basic design principle is: each control that exposes some filter functionality owns a dependency-injected IOM interface. At initialization, the application will connect the IOM and pass the interface(s) around to all instantiated controls.

## MVVM principles and examples

MVVM is more of a discipline than a specific library or framework. It is a set of guidelines that directs the programmer to separate (decouple) the logic of presentation from the data and algorithms of the underlying model

Advantages and disadvantages

- may induce maintenance overhead, significant for small applications

- lack of experience in MVVM can make it difficult to perform rapid prototyping

+ less code duplication

+ loose coupling between UI and model

+ increased testability

An explanation of how and when to use MVVM vs traditional design is given in [1].

Sample frameworks have been created and are available on public repositories. Of the most useful ones for beginners are:

[mvvm-light](https://github.com/lbugnion/mvvmlight)

[prism](https://github.com/PrismLibrary/Prism)

# Application requirements

The Energy Filter Monitor will display and control the various elements of the energy filter (specifically, the Iliad configuration).

The UI layout and look-and-feel is decided by stakeholders. The only precondition is that it is aligned with the Microsoft User Experience guidelines for Windows 10 desktop .NET applications.

|  |  |  |
| --- | --- | --- |
| ID | Description | Details |
| EFM-REQ-001 | The Microscope Launcher has a shortcut to the Energy Filter Monitor application |  |
| EFM-REQ-002 | The Energy Filter Monitor has a registered application icon |  |
| EFM-REQ-003 | A single-window application is preferred over a multi-dialog application | Pop-up (error) message boxes, modeless and modal dialogs should not be the way to communicate information to the user. Instead, all functionality shall reside in a single view (i.e. window)  The application window size shall be such that it fits the standard microscope PC screen [RAPID compliance test] (MaxWidth=1920, MaxHeight=1200 [TBD])  Screen resolution and scaling shall be taken into consideration (e.g. UI shall remain legible when changing DPI) [RAPID compliance test] |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## UI Elements

Looking at the Energy Filter model, we notice that several of the elements share their UI-control characteristics. Thus we can design composite UserControls to be reused for such common elements.

The additional UserControls that can be useful are:

**LabeledCombo**:

A Label next to a ComboBox; for a given element name that can take a value out of a list of available values.

*This control can be used i.e. for Aperture selection and the various filter Settings.*

**LabeledNumericUpDown**:

Content: the name of the filter element  
Value: the value of the element with given name  
Unit: a short string representing the measurement unit for the respective Value  
Delta: the current value step size (out of a predefined list of deltas)  
Up/Down spin buttons: Increase/Decrease the Value by Delta  
Up/Down delta spin buttons: select next/previous step size (from a predefined list of deltas)

*This control can be used i.e. for all available multipoles (values and offsets), as well as prism, energy shifts, and slit properties*

# Design principles

## Application-level design

.NET IOM wrapper

C++ Model

COM interop

ViewModel1

ViewModel2

ViewModelN

View1

View2

ViewM

Model

The application’s business logic resides in the C++ implementation of the energy\_filter model. The model is exposed via Microsoft COM to the IOM interface layer.

The MVVM Model is a .NET wrapper of the provided IOM interfaces.

View

The main application window comprises sections represented by UserControls that typically group together filter properties that relate to each other. Each of these UserControls is an MVVM View (of type Windows.UI.Xaml.FrameworkElement); each view should eventually bind to the model so that the UI elements show correct information and they allow usage of the filter capabilities.

ViewModel

The ViewModel is the intermediary layer in this design which ‘*assists’* with the binding of the view to the model. The ViewModel is a Windows.UI.Xaml.Data.INotifyPropertyChanged interface; it provides properties that will request data from the underlying model.

In order to bind to a ViewModel, a FrameworkElement only needs to set its DataContext property during initialization, as follows:

// FrameworkElement ctor

this.DataContext = viewModel;

Then the data of the View can be bound to the ViewModel properties.

### Implementation details

Model

For simplicity, the model can be a naïve singleton whose instance is shared among multiple ViewModels.

ViewModel

A ViewModel exposes properties in a specific way, which is captured in its base class BindableBase.cs.  
When the ViewModel becomes the DataContext of a view, its properties are available as binding paths in XAML.

View

The application may contain multiple Views (no restrictions on nested Views). Eah View is mapped to exactly one ViewModel; therefore the application may have multiple ViewModels. For a View to obtain the instance of its respective ViewModel we can use

* Dependency injection, or
* A ViewModelLocator (factory object) which contains mappings of Views to ViewModels