Functional Programming in F#

Quantitative Strategies

Zbigniew Fiałkiewicz
Artur Tadrała
Krzysztof Skrzętnicki

CREDIT SUISSE

Model – View – ViewModel architecture (MVVM)

- When you start building your app it is good for it to have a well defined structure – what goes where?
- One of possible ways to do this is MVVM
- Anything you add to your app is either:
  - Model (describing the *domain* your app is working in)
  - View (describing declaratively how to display stuff)
  - ViewModel (a model but not for your *domain*, but for the *UI* of your app)

- Further reading:
  - https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel
  - https://www.wintellect.com/model-view-viewmodel-mvvm-explained/

MVVM: Model

- What is the domain of your app?
  - Traffic modelling app: cars, buses, lanes, lights etc.
  - Financial app: trades, documents, market data etc.
- Should define meaningful actions/transformations of an object
- Should be reusable if needed in different application
- Shouldn’t define how to display them

MVVM: ViewModel (1/2)

- Describes the **model** for your **UI**
- Contains all the logic specific to UI and your model
  - If you need a list of „displayed trades" this is the place you put this in.
  - If you need to keep the state of your application, this is the place.
- There are frameworks that help implementing VMs.
- For smaller apps you hardly need them and can start bare bones.

MVVM: ViewModel (2/2)

- You do need to support certain interfaces:
  - ICommand (for executing actions, buttons etc.)
  - INotifyPropertyChanged (for notyfing the UI some VM property has changed)
  - You will see concrete examples soon in the Pricer app.

MVVM: View (1/2)

- Describes your user interface (UI) (declaratively)
- The display engine is interpreting the description and doing the work for you.
- Commonly used technology in .NET world: XAML / WPF
- XAML: eXtensible Application Markup Language
  - XML based
  - Easy to build simple stuff and it scales fairly well
  - You can define and reuse components (but you won't be need to do that in your use cases)
  - A lot of concepts overall, but easy to get started with by modifying existing examples

MVVM: View (2/2)

```
<Window x:Class="SimpleWpfApp1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:SimpleWpfApp1"
        mc:Ignorable="d"
            Title="The title of main window" Height="350" Width="525">
    <StackPanel>
        <Button>Say hi!</Button>
        <TextBlock>Just a piece of text</TextBlock>
    </StackPanel>
</Window>
```

- Further reading:
  - https://wpf-tutorial.com/

XAML: basics

- XML describing how the UI looks like
  - Control with some text as a content:

```
<Button>Say hi!</Button>
```

  - Often have children:

```
<StackPanel>
    <Button>Say hi!</Button>
</StackPanel>
```

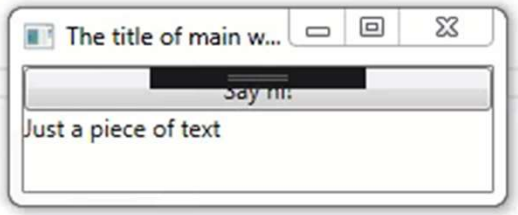
  - Properties are common:

```
<StackPanel Margin="10" Orientation="Horizontal">
```

XAML: live editing

- If you debug the program from VS you can edit the XAML on the fly
- Very useful to rapidly develop the UI – immediate feedback

```
6       xmlns:local="clr-namespace:SimpleWpfApp1"
7       mc:Ignorable="d"
8       Title="The title of main window" Height="100" Width="250"
9       Topmost="True">
0   <StackPanel>
1       <Button>Say hi!</Button>
2       <TextBlock>Just a piece of text</TextBlock>
3   </StackPanel>
4 </Window>
5
```



XAML: binding

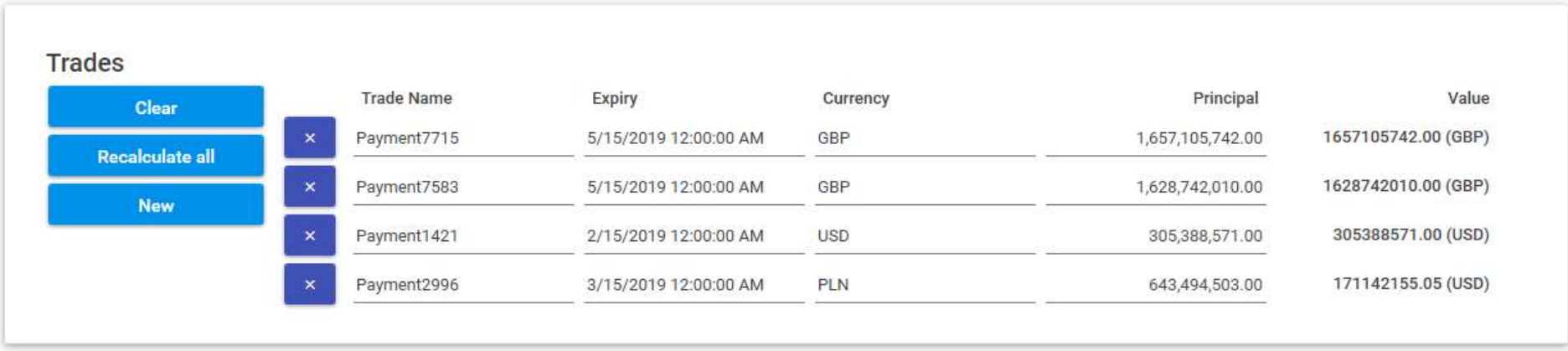
- XAML is static content – doesn't change after compilation
- We need a mechanism for stuff that changes
- Solution: bindings
- Allows to pass (changing) data between View and ViewModel
- Example use cases:
  - Status of buttons (enabled, disabled)
  - Displaying trades in a list
  - Getting search text from user
    - `<TextBox Text="{Binding SearchString}" />`
    - `SearchString` property on respective ViewModel!

Pricer app walkthrough (1/8)

- This is the app that you will be extending
- Simple enough to be fully understood (with time)
- Still you should focus on parts that you really need to modify
- The rest can be safely ignored

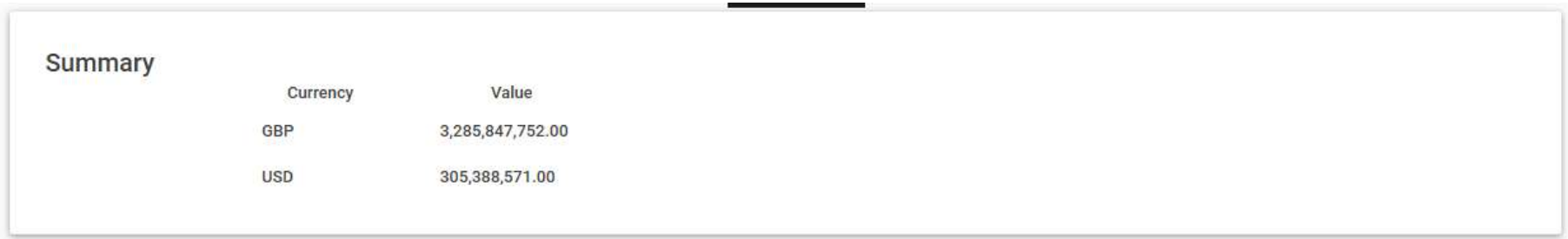
Pricer app: trades (2/8)

- You can add, edit, remove trades
- Value is calculated when you add or remove a trade
- You can force recalculation (useful when untracked dependency changes, e.g. some piece of market data)
- Value can have different currency than trade … more info later



Pricer app: summary (3/8)

- Summarizes total trade value by currency
- Adds/removes new currency rows as available



Summary

Currency	Value
GBP	3,285,847,752.00
USD	305,388,571.00

Pricer app: market data (4/8)

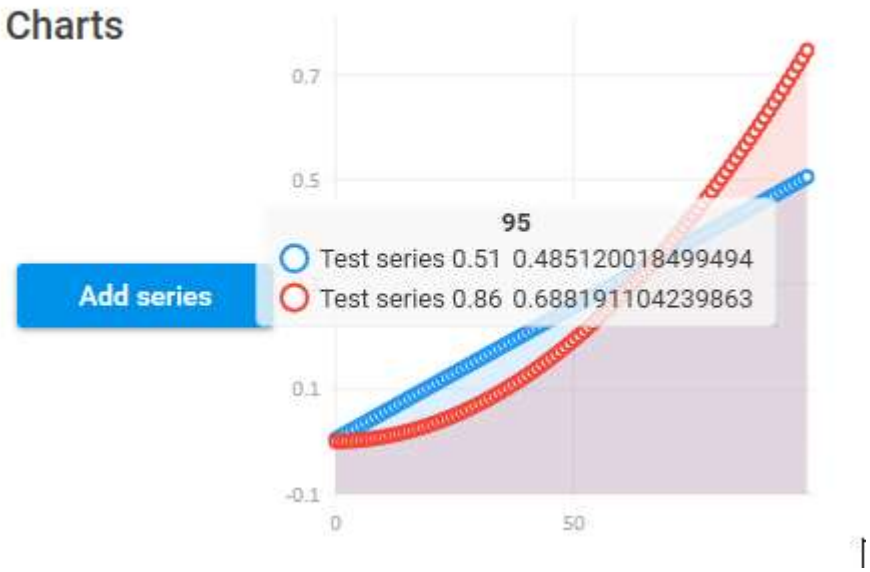
- Allows a list of arbitrary key/values (string -> string)
- Available in trade valuation context
- Dummy FX rates as examples

Pricer app: configuration data (5/8)

- Allows a list of arbitrary key/values (string -> string)
- Available in trade valuation context
- Basically same as market data, but useful to distinguish the concepts

Pricer app: charts (6/8)

- Proof of concept showing how to use
- You will have to develop it further to make some use of it



Pricer app: key parts of code (7/8)

- Solution structure (View.csproj, ViewModel.fsproj)
- View: MainWindow.xaml
- ViewModel
  - MainViewModel.fs
  - ConfigurationViewModel.fs
  - PaymentViewModel.fs
- Models
  - PaymentModel.fs
- Utilities
  - SimpleCommand.fs : compatible Commands
  - ViewModelBase.fs : INotifyPropertyChanged plumbing
  - OptionDoubleToDoubleConverter.fs: convert data types before displaying them

Pricer app: summary (8/8)

- A lot of things happening in this app
- Should be extensible enough for your purposes
- If you get stuck:
  - https://wpf-tutorial.com/
  - https://stackoverflow.com/
  - Mail us.