FREE eBook

LEARNING mvvm-light

Free unaffiliated eBook created from
Stack Overflow contributors.

#mvvm-light

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: mvvm-light

It is an unofficial and free mvvm-light ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official mvvm-light.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with mvvm-light

Remarks

MVVM-light is a toolkit written in C# which helps to accelerate the creation and development of MVVM applications in WPF, Silverlight, Windows Store, Windows Phone and Xamarin.

Webpage: http://www.mvvmlight.net/

There is a cross-platform MVVM sample from the author of the library which can be found at https://github.com/lbugnion/sample-crossplatform-flowers

Examples

RelayCommand

The `RelayCommand` implements the `ICommand` interface and can therefore be used to bind to `Command`s in XAML (as the `Command` property of the `Button` element)

The constructor takes two arguments; the first is an `Action` which will be executed if `ICommand.Execute` is called (e.g. the user clicks on the button), the second one is a `Func<bool>` which determines if the action can be executed (defaults to true, called `canExecute` in the following paragraph).

the basic structure is as follows:

```
public ICommand MyCommand => new RelayCommand(
    () =>
    {
        //execute action
        Message = "clicked Button";
    },
    () =>
    {
        //return true if button should be enabled or not
        return true;
    }
);
```

Some notable effects:

- If `canExecute` returns false, the `Button` will be disabled for the user
- Before the action is really executed, `canExecute` will be checked again
- You can call `MyCommand.RaiseCanExecuteChanged();` to force reevaluation of the `canExecute Func`

RelayCommand

The `RelayCommand<T>` is similar to the `RelayCommand`, but allows to directly pass an object to the

command. It implements the `ICommand` interface and can therefore be used to bind to `Command`s in XAML (e.g. as the `Command` property of the `Button` element). You can then use the `CommandParameter` property to pass the object to the command.

XAML example:

```
<Button Command="{Binding MyCommand}" CommandParameter="{Binding MyModel}" />
```

The constructor takes two arguments; the first is an Action which will be executed if ICommand.Execute is called (e.g. the user clicks on the button), the second one is a Func<string,bool> which determines if the action can be executed (defaults to true, called canExecute in the following paragraph). the basic structure is as follows:

```
public RelayCommand<string> MyCommand => new RelayCommand<string>(
    obj =>
    {
        //execute action
        Message = obj;
    },
    obj =>
    {
        //return true if button should be enabled or not
        return obj != "allowed";
    }
);
```

Some notable effects:

- If `canExecute` returns `false`, the `Button` will be disabled for the user
- Before the action is really executed, `canExecute` will be checked again
- You can call `MyCommand.RaiseCanExecuteChanged();` to force reevaluation of the `canExecute Func`

ObservableObject

The `ObservableObject` class contains some helpful methods to help with the MVVM pattern.

The `RaisePropertyChanged` provides a compile safe method to raise property changed events.
It can be called with

```
RaisePropertyChanged(() => MyProperty);
```

The `Set` method can be used in the property setter to set the new value and raise the property changed event (only if change occurred). It returns `true` if change occurred and `false` otherwise. example usage:

```
private string _myValue;
public string MyValue
{
    get { return _myValue; }
    set { Set(ref _myValue, value); }
}
```

ViewModelBase

`ViewModelBase` extends `ObservableObject` and adds some methods useful for viewmodels.

The property `IsInDesignMode` or `IsInDesignModeStatic` allows to determine if the code is executed in the design mode (in Visual Studio Design View) or not. The two properties are exactly the same.

Read Getting started with mvvm-light online: https://riptutorial.com/mvvm-light/topic/10610/getting-started-with-mvvm-light

Credits

S. No	Chapters	Contributors
1	Getting started with mvvm-light	Community, Florian Moser