



EBook Gratuito

APPENDIMENTO

mvvm-light

Free unaffiliated eBook created from
Stack Overflow contributors.

#mvvm-light

Sommario

Di.....	1
Capitolo 1: Iniziare con mvvm-light.....	2
Osservazioni.....	2
Examples.....	2
RelayCommand.....	2
RelayCommand.....	2
ObservableObject.....	3
ViewModelBase.....	4
Titoli di coda.....	5

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.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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

Capitolo 1: Iniziare con mvvm-light

Osservazioni

MVVM-light è un toolkit scritto in C # che aiuta ad accelerare la creazione e lo sviluppo di applicazioni MVVM in WPF, Silverlight, Windows Store, Windows Phone e Xamarin.

Pagina web: <http://www.mvmlight.net/>

Esiste un campione MVVM multiplatforma dall'autore della libreria che può essere trovato su <https://github.com/lbugnion/sample-crossplatform-flowers>

Examples

RelayCommand

`RelayCommand` implementa l'interfaccia `ICommand` e può quindi essere utilizzato per collegarsi a `Command` s in XAML (come proprietà `Command` dell'elemento `Button`)

Il costruttore prende due argomenti; il primo è `Action` che verrà eseguita se `ICommand.Execute` viene chiamato (ad esempio, l'utente fa clic sul pulsante), il secondo è un `Func<bool>` che determina se l'azione può essere eseguita (il valore predefinito è `true`, chiamato `canExecute` in il seguente paragrafo).

la struttura di base è la seguente:

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

Alcuni effetti notevoli:

- Se `canExecute` restituisce `false`, il `Button` verrà disabilitato per l'utente
- Prima che l'azione sia realmente eseguita, `canExecute` verrà ricontrollato
- Puoi chiamare `MyCommand.RaiseCanExecuteChanged()` ; forzare la rivalutazione di `canExecute Func`

RelayCommand

`RelayCommand<T>` è simile a `RelayCommand` , ma consente di passare direttamente un oggetto al

comando. Implementa l'interfaccia `ICommand` e può quindi essere utilizzato per collegarsi a `Command` s in XAML (ad esempio come proprietà `Command` dell'elemento `Button`). È quindi possibile utilizzare la proprietà `CommandParameter` per passare l'oggetto al comando.

Esempio XAML:

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

Il costruttore prende due argomenti; il primo è un'azione che verrà eseguita se `ICommand.Execute` viene chiamato (ad esempio, l'utente fa clic sul pulsante), il secondo è `Func<string, bool>` che determina se l'azione può essere eseguita (il valore predefinito è `true`, chiamato `canExecute` nel seguente paragrafo). la struttura di base è la seguente:

```
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";
    }
);
```

Alcuni effetti notevoli:

- Se `canExecute` restituisce `false` , il `Button` verrà disabilitato per l'utente
- Prima che l'azione sia realmente eseguita, `canExecute` verrà ricontrollato
- Puoi chiamare `MyCommand.RaiseCanExecuteChanged()` ; forzare la rivalutazione di `canExecute` `Func`

ObservableObject

La classe `ObservableObject` contiene alcuni metodi utili per aiutare con il pattern MVVM.

`RaisePropertyChanged` fornisce un metodo di compilazione sicuro per aumentare gli eventi modificati di proprietà.

Può essere chiamato con

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

Il metodo `Set` può essere utilizzato nel setter della proprietà per impostare il nuovo valore e generare l'evento modificato della proprietà (solo se si sono verificati cambiamenti). Restituisce `true` se il cambiamento si è verificato e `false` altro modo.

esempio di utilizzo:

```
private string _myValue;
public string MyValue
{
```

```
get { return _myValue; }  
set { Set(ref _myValue, value); }  
}
```

ViewModelBase

ViewModelBase **estende** `ObservableObject` e aggiunge alcuni metodi utili per viewmodels.

La proprietà `IsInDesignMode` o `IsInDesignModeStatic` consente di determinare se il codice viene eseguito in modalità progettazione (in Visual Studio Design View) o meno. Le due proprietà sono esattamente le stesse.

Leggi Iniziare con mvvm-light online: <https://riptutorial.com/it/mvvm-light/topic/10610/iniziare-con-mvvm-light>

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con mvvm-light	Community , Florian Moser