



# MVVM

## Паттерн Model-View-ViewModel

Последнее обновление: 21.06.2016



Помогите им найти вашу  
компанию в Google!

[НАЧНИТЕ СЕЙЧАС](#)

Компенсируем  
до \$60

Google AdWo

Паттерн MVVM (Model - View - ViewModel) основывается на разделении функциональной части приложения на три ключевых компонента:

- **View** - представление или пользовательский интерфейс
- **Model** - модель или данные, которые используются в приложении
- **ViewModel** - промежуточный слой между представлением и данными, который обеспечивает их взаимодействие

Преимуществом использования данного паттерна является меньшая связанность между компонентами и разделение ответственности между ними. То есть Model отвечает за данные, View отвечает за графический интерфейс, а ViewModel - за логику приложения.

Легкость реализации паттерна MVVM в Xamarin Forms стала возможной благодаря ранее рассмотренному механизму привязки.

Рассмотрим простейший пример. Определим класс данных или модели:

```
public class Phone
{
    public string Title { get; set; }
    public string Company { get; set; }
    public int Price { get; set; }
}
```

Также добавим в проект класс, который назовем PhoneViewModel со следующим содержимым:

```
using System.ComponentModel;

public class PhoneViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    public Phone Phone { get; set; }

    public PhoneViewModel()
    {
    }
```

```

        Phone = new Phone();
    }

    public string Title
    {
        get { return Phone.Title; }
        set
        {
            if (Phone.Title != value)
            {
                Phone.Title = value;
                OnPropertyChanged("Title");
            }
        }
    }

    public string Company
    {
        get { return Phone.Company; }
        set
        {
            if (Phone.Company != value)
            {
                Phone.Company = value;
                OnPropertyChanged("Company");
            }
        }
    }

    public int Price
    {
        get { return Phone.Price; }
        set
        {
            if (Phone.Price != value)
            {
                Phone.Price = value;
                OnPropertyChanged("Price");
            }
        }
    }

    protected void OnPropertyChanged(string propName)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(propName));
    }
}

```

Это и будет компонент ViewModel, который связывает данные и визуальный интерфейс. По большому счету она представляет обертку над классом Phone, определяя все те же свойства. Для упрощения задачи сам объект Phone создается в конструкторе, хотя в реальности там могла бы быть более сложная логика, например, по получению объекта из базы данных.

Важно, что данный класс реализует интерфейс **INotifyPropertyChanged**, что позволяет уведомлять систему об изменении его свойств с помощью события PropertyChanged.

Теперь создадим визуальную часть. Определим на главной странице MainPage.xaml следующее содержимое:

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="MvvmApplication.MainPage">
    <StackLayout>
        <Entry Text="{Binding Title}" FontSize="Medium" />
    </StackLayout>
</ContentPage>

```

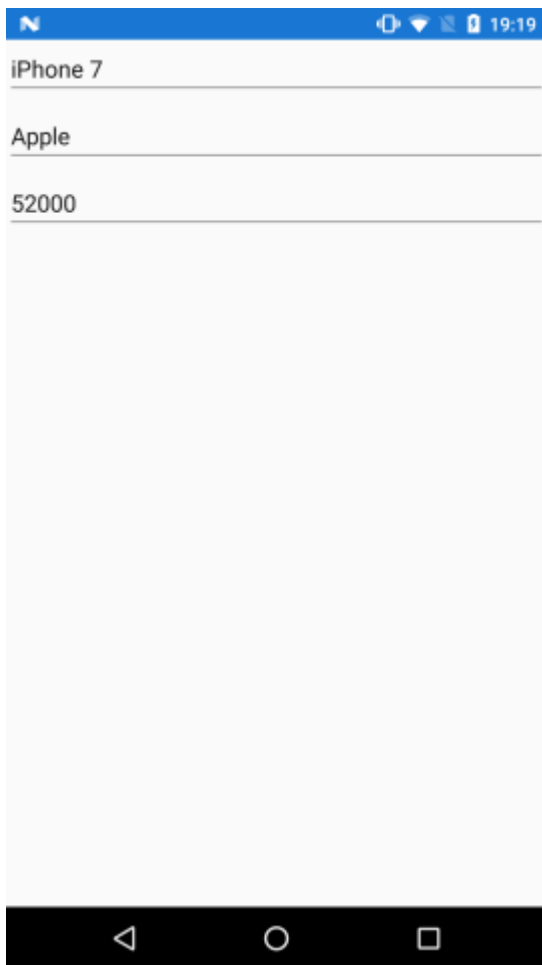
```
<Entry Text="{Binding Company}" FontSize="Medium" />
<Entry Text="{Binding Price}" FontSize="Medium" />
</StackLayout>
</ContentPage>
```



Здесь определена привязка к свойству ViewModel.

А в конструкторе страницы в файле кода с# пропишем в качестве контекста данных для страницы определенную ранее ViewModel:

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        this.BindingContext = new PhoneViewModel
        {
            Title = "iPhone 7",
            Company = "Apple",
            Price=52000
        };
    }
}
```

И при запуске приложение выведет нам все данные о viewmodel, переданной во view:





Всё для чистоты и гигиены в организациях.  
Доступные цены.  
Быстрая доставка.

[Подробнее](#)

[Назад](#) [Содержание](#) [Вперед](#)

Яндекс.Директ

Сушилка для рук BXG

21vek.by

Яндекс.Директ

Металлопрокат в Минске.  
Недорого!

metallopt.by

3 Комментариев metanit.com

1 Войти ▾

[♥ Рекомендовать](#)
[🔗 Поделиться](#)

Лучшее в начале ▾

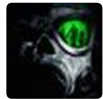


Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ

ИЛИ ЧЕРЕЗ DISQUS (?)

Имя

**Alexey** • 2 месяца назад

+1 к тому, чтобы был рассмотрен также фреймворк MVVM Light или же MVVM Cross. Было бы замечательно просто

1 ^ | ▾ • Ответить • Поделиться ›

**amankkg** • 3 года назад

планируете рассмотреть MVVM Light или MVVMCross?

^ | ▾ • Ответить • Поделиться ›

**Metanit** Модератор ➔ amankkg • 3 года назад

пока не планируется, но в будущем возможно будут соответствующие материалы

^ | ▾ • Ответить • Поделиться ›

ТАКЖЕ НА METANIT.COM

**Vue.js | Именованные слоты**

5 комментариев • 4 месяца назад



**Metanit** — все должно работать, ничего не изменилось, должно быть вы что-то не то делаете Просто уберите из ...

**Angular в ASP.NET Core | CRUD и маршрутизация. Часть 1**

1 комментарий • 3 месяца назад



**Vadim Prokopchuk** — Добрый вечер. Попытался в текущий пример добавить стилизацию, но столкнулся ...

**Go | http.Client**

1 комментарий • месяц назад



**Roman** — Спасибо. Замечательные статьи. Но хотелось бы больше сетевого программирования. Очень ...

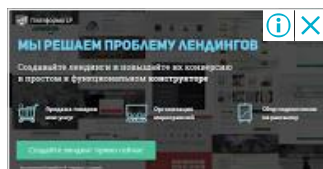
**Go | Синхронизация**

1 комментарий • месяц назад



**Roman** — Получается, что каналы и карты передаются в функцию по ссылке, верно?

[✉ Подписаться](#)
[D Добавь Disqus на свой сайт](#)
[Добавить Disqus](#)
[Добавить](#)
[🔒 Конфиденциальность](#)



## Удобный конструктор сайтов



Запусти свой сайт  
баз навыков  
программирования  
за час. Бесплатный  
период 14 дней.



---

[Вконтакте](#) | [Twitter](#) | [Google+](#) | [Канал сайта на youtube](#) | [Помощь сайту](#)

Copyright © metanit.com, 2012-2017. Все права защищены.