

# Шаблон MVVM

---

# Введение

---

.NET MAUI

# Введение

---

Шаблон **Model-View-ViewModel (MVVM)** обеспечивает разделение между тремя программными уровнями:

- пользовательским интерфейсом XAML, называемым **представлением**,
- базовыми данными, называемыми **моделью**, и
- промежуточным звеном между представлением и моделью, называемым **моделью представления**.

# Введение

---

Представление и модель представления часто связаны через привязки данных (Bindings), определенные в XAML.

**BindingContext** для представления обычно является экземпляром модели представления.

**Можно сказать, что модель представления описывает поведение представления**

# Привязка данных

---

MVVM

# Привязка данных

---

Для двусторонней привязки данных модели представления обычно реализуют интерфейс **INotifyPropertyChanged**, который предоставляет классу возможность вызывать событие **PropertyChanged** всякий раз, когда изменяется одно из его свойств.

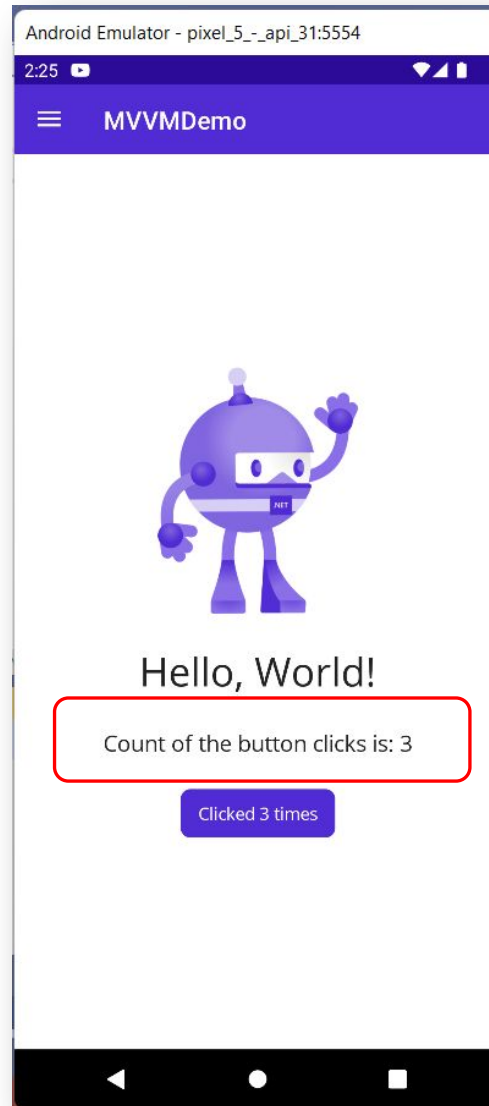
# Привязка данных

---

Механизм привязки данных в .NET MAUI прикрепляет обработчик к этому событию `PropertyChanged`, чтобы его можно было уведомлять об изменении свойства и обновлять цель (элементы XAML) с новым значением.



# Привязка данных (Пример)



# Привязка данных (Пример)

```
public class MVVMDemoViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    int _counter;
    public int Counter {
        get => _counter;
        set
        {
            if (_counter == value) return;
            _counter = value;
            OnPropertyChanged();
        }
    }

    private void OnPropertyChanged([CallerMemberName] string property="")
    {
        if(!String.IsNullOrEmpty(property))
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(property));
    }
}
```

# Привязка данных (Пример)

```
public partial class MVVMDemo : ContentPage
{
    int count = 0;
    MVVMDemoViewModel viewModel;
    public MVVMDemo(MVVMDemoViewModel model)
    {
        InitializeComponent();
        viewModel = model;
        BindingContext = model;
    }

    private void OnCounterClicked(object sender, EventArgs e)
    {
        count++;
        viewModel.Counter = count;

        if (count == 1)
            CounterBtn.Text = $"Clicked {count} time";
        else
            CounterBtn.Text = $"Clicked {count} times";
    }
}
```

# Привязка данных (Пример)

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:models="clr-namespace:MAUI_LK.ViewModels"
  x:DataType="models:MVVMDemoViewModel"
  x:Class="MAUI_LK.Pages.MVVMDemo"
  Title="MVVMDemo">
  <ScrollView>
    <VerticalStackLayout Spacing="25" Padding="30,0" VerticalOptions="Center">
      ...
      <HorizontalStackLayout HorizontalOptions="Center">
        <Label Text="Count of the button clicks is: " FontSize="18" />
        <Label Text="{Binding Counter}" FontSize="18"/>
      </HorizontalStackLayout>
      <Button
        x:Name="CounterBtn" Text="Click me"
        Clicked="OnCounterClicked"
        HorizontalOptions="Center" />
      </VerticalStackLayout>
    </ScrollView>
  </ContentPage>
```

# Привязка данных (Пример)

---

```
public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        ...
        AddServices(builder.Services);
        return builder.Build();
    }

    private static void AddServices(IServiceCollection services)
    {
        // Services
        services.AddSingleton<SQLiteService>();

        // View models
        services.AddTransient<MVVMDemoViewModel>();

        // Pages
        services.AddTransient<MVVMDemo>();
    }
}
```

# Привязка данных (Пример)

---

Выделим реализацию INotifyPropertyChanged в отдельный класс:

```
public class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged([CallerMemberName] string property = "")
    {
        if (!String.IsNullOrEmpty(property))
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(property));
    }
}
```

# Привязка данных (Пример)

---

Изменим класс MVVMDemoViewModel:

```
public class MVVMDemoViewModel : ViewModelBase
{
    int _counter;
    public int Counter {
        get => _counter;
        set
        {
            if (_counter == value) return;
            _counter = value;
            OnPropertyChanged();
        }
    }
}
```

# Команды

---

MVVM



# Команды

---

Часто пользователю нужно инициировать команды, которые влияют на что-то в модели представления. Эти команды обычно сигнализируются нажатием кнопки или нажатием пальца, и традиционно они обрабатываются в файле `code-behind`, например, в обработчике события `Clicked` кнопки или события `Tapped` объекта `TapGestureRecognizer`.

# Команды

---

Интерфейс команд обеспечивает альтернативный подход к реализации команд, который гораздо лучше подходит для архитектуры MVVM.

Модель представления может содержать команды, которые представляют собой методы, выполняемые в ответ на определенное действие в представлении, такое как нажатие кнопки. Привязки определяются между этими командами и кнопкой

# Команды

---

Интерфейс **ICommand** определен в пространстве имен **System.Windows.Input** и состоит из двух методов и одного события:

`void Execute (object arg)`

`bool CanExecute(object arg)`

`event EventHandler CanExecuteChanged`

# Команды

---

Модель представления может определять свойства типа  **ICommand**. Затем вы можете привязать эти свойства к свойству **Command** каждой кнопки или другого элемента или, возможно, к пользовательскому представлению, которое реализует этот интерфейс.

# Команды

---

Дополнительно можно установить свойство **CommandParameter** для идентификации отдельных объектов Button (или других элементов), которые привязаны к этому свойству модели представления. Внутри Button вызывает метод **Execute** каждый раз, когда пользователь нажимает кнопку, передавая методу Execute свой **CommandParameter**.

# Команды

---

Метод **CanExecute** и событие **CanExecuteChanged** используются в случаях, когда нажатие кнопки может быть недействительным в данный момент, и в этом случае кнопка должна отключиться. Кнопка вызывает **CanExecute** при первом задании свойства **Command** и всякий раз, когда вызывается событие **CanExecuteChanged**. Если **CanExecute** возвращает **false**, **Button** отключается и не генерирует вызовы **Execute**.

# Команды (Пример)

```
public class MVVMDemoViewModel : ViewModelBase
```

```
{  
    int _counter;  
    public ICommand IncreaseCounter { get; set; }
```

```
    public int Counter {  
        get => _counter;  
        set  
        {  
            if (_counter == value) return;  
            _counter = value;  
            OnPropertyChanged();  
        }  
    }  
}
```

```
public MVVMDemoViewModel()  
{  
    IncreaseCounter = new Command(() => Counter++ );  
}
```

# Команды

---

<Button

x:Name="CounterBtn"

Text="Click me"

Command="{Binding IncreaseCounter}"

HorizontalOptions="Center" />



# Команды (Пример)

---

```
<Frame Background="WhiteSmoke">
  <Frame.GestureRecognizers>
    <TapGestureRecognizer
      Command="{Binding
        Source={RelativeSource
          AncestorType={x:Type models:DoctorsListViewModel}},
        Path=ShowDetailsCommand }"
      CommandParameter="{Binding Id}"/>
    </Frame.GestureRecognizers>
  <Grid ColumnDefinitions="Auto,*" RowDefinitions="20,*">
    ...
  </Grid>
</Frame>
```

# CommunityToolkit

---

MVVM

# CommunityToolkit

---

Пакет **CommunityToolkit.Mvvm** (также известный как MVVM Toolkit, ранее называвшийся Microsoft.Toolkit.Mvvm) — это современная, быстрая и модульная библиотека MVVM.

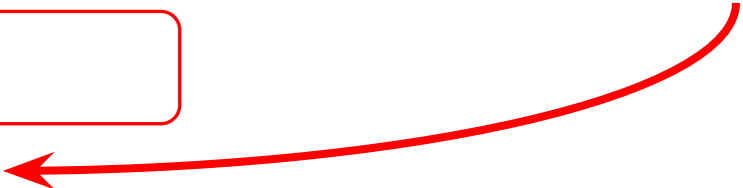
<https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/>

# CommunityToolkit

```
using CommunityToolkit.Mvvm.ComponentModel;  
using CommunityToolkit.Mvvm.Input;
```

```
namespace MAUI_LK.ViewModels
```

```
{  
    public partial class MVVMDemoViewModel : ObservableObject  
    {  
        [RelayCommand]  
        public void IncreaseCounter() => Counter++;  
  
        [ObservableProperty]  
        int _counter  
    }  
}
```



# CommunityToolkit

---

<Button

x:Name="CounterBtn"

Text="Click me"

Command="{Binding IncreaseCounterCommand}"

HorizontalOptions="Center" />

# CommunityToolkit

---

.NET MAUI Community Toolkit — это набор повторно используемых элементов для разработки приложений с помощью .NET MAUI, включая анимацию, поведение, преобразователи, эффекты и помощники. Он упрощает и демонстрирует общие задачи разработчиков при создании приложений для iOS, Android, macOS и WinUI с использованием .NET MAUI.

Набор инструментов сообщества MAUI доступен в виде набора пакетов NuGet для новых или существующих проектов .NET MAUI.

<https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/>

# CommunityToolkit

---

Для использования .NET MAUI Community Toolkit:

1. Загрузите NuGet пакет **CommunityToolkit.Maui**

2. В классе MauiProgram:

builder

.UseMauiApp<App>()

.UseMauiCommunityToolkit()

# CommunityToolkit

---

Иногда требуется привязать событие к команде.

Для этого можно использовать  
EventToCommandBehavior из библиотеки  
**CommunityToolkit.Maui**



# CommunityToolkit (Пример)

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:models="clr-namespace:MAUI_LK.ViewModels"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MAUI_LK.Pages.SQLiteDemo"
    x:DataType="models:SQLiteDemoViewModel"
    Title="SQLiteDemo">
    <ContentPage.Behaviors>
        <toolkit:EventToCommandBehavior
            EventName="Loaded"
            Command="{Binding LoadedCommand}"/>
    </ContentPage.Behaviors>
    <VerticalStackLayout>
        ...
    </VerticalStackLayout>
</ContentPage>
```

# CommunityToolkit (Еще пример)

```
<Frame Margin="5,10" CornerRadius="20" Padding="10">
```

```
    <Picker
```

```
        Title="Выберите специализацию"
```

```
        VerticalOptions="Center"
```

```
        ItemsSource="{Binding Specialities}"
```

```
        SelectedItem="{Binding SelectedSpeciality}"
```

```
        ItemDisplayBinding="{Binding Name}" >
```

```
        <Picker.Behaviors>
```

```
            <toolkit:EventToCommandBehavior
```

```
                EventName="SelectedIndexChanged"
```

```
                Command="{Binding SpecialitySelectedCommand}" />
```

```
        </Picker.Behaviors>
```

```
    </Picker>
```

```
</Frame>
```

# CommunityToolkit

---

## (Пример использования AvatarView)

```
<toolkit:AvatarView  
    WidthRequest="60" HeightRequest="60"  
    ImageSource="{Binding Id,  
        Converter={StaticResource Avatar}}"  
    BorderWidth="2" BorderColor="LightGrey"/>
```

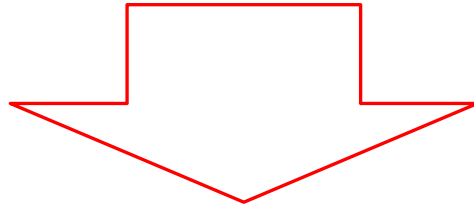
# CommunityToolkit

---

## (Регистрация сервисов)

```
services.AddTransient<MVVMDemoViewModel>();
```

```
services.AddTransient<MVVMDemo>();
```



```
services.AddTransient<MVVMDemo, MVVMDemoViewModel>();
```

# Publisher-Subscriber pattern

---

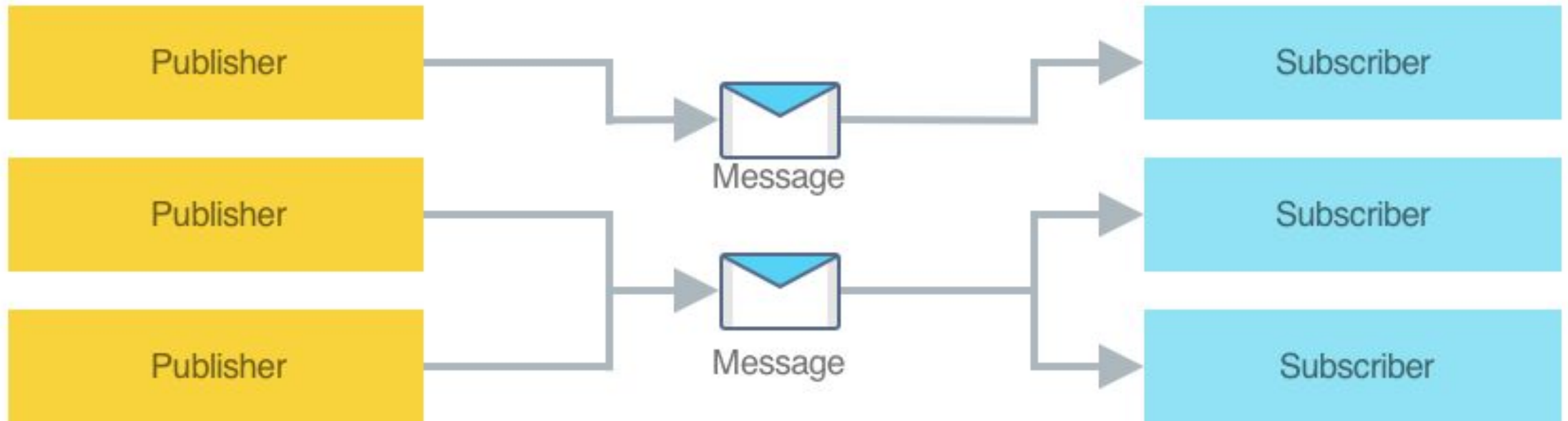
# Publisher-Subscriber pattern

---

Класс `MessagingCenter` .NET MAUI реализует шаблон публикации-подписки, позволяя обмениваться сообщениями между компонентами.

Этот механизм позволяет издателям и подписчикам общаться без ссылок друг на друга, помогая уменьшить зависимости между ними.

# Publisher-Subscriber pattern



<https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/messagingcenter>

# Publisher-Subscriber pattern

---

Издатели отправляют сообщения с помощью метода **MessagingCenter.Send**, а подписчики прослушивают сообщения с помощью метода **MessagingCenter.Subscribe**. Кроме того, подписчики также могут отказаться от подписки на сообщения, если это необходимо, с помощью метода **MessagingCenter.Unsubscribe**.



# Publisher-Subscriber pattern (Пример)

[RelayCommand]

```
async Task CreateOrder()
```

```
{
```

```
    ...
```

```
    var result = await _orderService.CreateOrderAsync(order);
```

```
    if(!result)
```

```
    {
```

```
        await Shell.Current.DisplayAlert("Error", "Fail to add new order. Try later", "Ok");
```

```
    }
```

```
    else
```

```
    {
```

```
        MessagingCenter.Send<CreateOrderViewModel>(this, "update");
```

```
    }
```

```
    await Shell.Current.GoToAsync($"{nameof(OrdersList)}");
```

```
}
```

# Publisher-Subscriber pattern (Пример)

```
public OrdersListViewModel(IOrderService orderService)
{
    _orderService = orderService;
    UpdateOrdersAsync();
    MessagingCenter.Subscribe<CreateOrderVewModel>(this, "update",
        async (sender) => await UpdateOrdersAsync());
}
```