

.Net MAUI

XAML

Grid Layout

.NET MAUI

Grid Layout

Класс **Grid** определяет следующие свойства:

- ❑ **Column** типа **int**, который является прикрепленным свойством, указывающим номер столбца представления в родительской сетке. Значение по умолчанию для этого свойства равно 0.
- ❑ **ColumnDefinitions** типа **ColumnDefinitionCollection** — это список объектов **ColumnDefinition**, определяющих ширину столбцов сетки.
- ❑ **ColumnSpacing** типа **double** указывает расстояние между столбцами сетки. Значение по умолчанию этого свойства равно 0.
- ❑ **ColumnSpan** типа **int**, которое является прикрепленным свойством, указывающим общее количество столбцов, которые охватывает представление в родительской сетке. Значение этого свойства по умолчанию равно 1.

Grid Layout

Класс **Grid** определяет следующие свойства:

- ❑ **Row** типа **int**, которое является прикрепленным свойством, указывающим номер строки представления в родительской сетке. Значение по умолчанию для этого свойства равно 0.
- ❑ **RowDefinitions** типа **RowDefinitionCollection** — это список объектов **RowDefinition**, определяющих высоту строк сетки.
- ❑ **RowSpacing** типа **double** указывает расстояние между строками сетки. Значение по умолчанию этого свойства равно 0.
- ❑ **RowSpan** типа **int**, которое является прикрепленным свойством, указывающим общее количество строк, занимаемых представлением в родительской сетке. Значение этого свойства по умолчанию равно 1.

Grid Layout

Класс **RowDefinition** определяет свойство **Height** типа **GridLength**, а класс **ColumnDefinition** определяет свойство **Width** типа **GridLength**. Структура **GridLength** задает высоту строки или ширину столбца с точки зрения перечисления **GridUnitType**, состоящего из трех элементов:

Абсолютный — высота строки или ширина столбца представляет собой значение в аппаратно-независимых единицах (число в XAML).

Авто — высота строки или ширина столбца автоматически изменяется в зависимости от содержимого ячейки (автоматически в XAML).

Звезда — оставшаяся высота строки или ширина столбца распределяются пропорционально (число, за которым следует * в

Grid Layout

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="2*" />

<RowDefinition Height="*" />

<RowDefinition Height="100" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="*" />

<ColumnDefinition Width="*" />

</Grid.ColumnDefinitions>

...

</Grid>

Grid Layout

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="2*" />

<RowDefinition Height="*" />

<RowDefinition Height="100" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="*" />

<ColumnDefinition Width="*" />

</Grid.ColumnDefinitions>

...

</Grid>

Grid Layout

```
<Grid RowDefinitions="2*,*,100"  
      ColumnDefinitions="*,*">  
  <Label Grid.Row="1" Text="Row1, Col 0"/>  
</Grid>
```


Ресурсы

.NET MAUI

Ресурсы

Ресурс — это любой объект, который можно использовать в вашем пользовательском интерфейсе. Наиболее распространенными примерами являются шрифты, цвета и размеры. Однако в качестве ресурсов можно также хранить сложные объекты, такие как экземпляры **Style** и **OnPlatform**.

Ресурсы

<Label TextColor="Blue" FontSize="14"/>

<Button TextColor="Blue" FontSize="14"/>

Ресурсы

<Color x:Key="DefaultColor">Blue</Color>

<x:Double x:Key="DefaultSize">14</x:Double>

<Label TextColor="{StaticResource DefaultColor}"
FontSize="{StaticResource DefaultSize}"/>

<Button TextColor="{StaticResource DefaultColor}"
FontSize="{StaticResource DefaultSize}"/>

Ресурсы

StaticResource — это расширение разметки для поиска ресурсов в словаре ресурсов.

Ресурсы

ResourceDictionary — это класс библиотеки .NET MAUI, настроенный для использования с ресурсами пользовательского интерфейса. Это словарь, поэтому он хранит пары ключ/значение. Тип ключа ограничен строкой, а значением может быть любой объект.

Каждая страница .NET MAUI XAML имеет свойство **Resources**, которое может содержать объект **ResourceDictionary**. Свойство по умолчанию имеет значение null, поэтому вам необходимо создать экземпляр словаря, прежде чем вы сможете его использовать.

Ресурсы

<ContentPage.Resources>

<ResourceDictionary>

<Color x:Key="Default">Blue</Color>

<x:Double x:Key="DefaultSize">14</x:Double>

</ResourceDictionary>

</ContentPage.Resources>

Ресурсы

Каждый элемент управления на странице также может иметь собственный словарь ресурсов.

Ресурсы

Стандартный способ определить значения для конкретной платформы — использовать объект **OnPlatform** при определении ресурса:

```
<OnPlatform x:Key="textColor" x:TypeArguments="Color">
```

```
    <On Platform="iOS" Value="Silver" />
```

```
    <On Platform="Android" Value="Green" />
```

```
    <On Platform="WinUI" Value="Yellow" />
```

```
    <On Platform="MacCatalyst" Value="Pink" />
```

```
</OnPlatform>
```

Введение в привязку данных

Введение в привязку данных

Привязка данных — это способ связывания свойств двух объектов таким образом, чтобы изменения одного свойства автоматически отражались в другом свойстве.

Один из двух объектов, задействованных в привязке данных, почти всегда является элементом, производным от View и формирующим часть визуального интерфейса страницы.

Другой объект либо:

- Другой производный объект View, обычно на той же странице.
- Объект в файле кода.

Введение в привязку данных

Привязка данных пользовательского интерфейса связывает пару свойств между двумя объектами, по крайней мере один из которых обычно является объектом пользовательского интерфейса. Эти два объекта называются целью (**target**) и источником (**source**):

Target — это объект (и свойство), для которого установлена привязка данных.

Source — это объект (и свойство), на который ссылается привязка данных.

Привязка через BindingContext

```
<Label x:Name="label"  
    Text="TEXT"  
    FontSize="48"  
    HorizontalOptions="Center"  
    VerticalOptions="Center" />
```

```
<Slider x:Name="slider"  
    Maximum="360"  
    VerticalOptions="Center" />
```

```
public MainPage()  
{  
    InitializeComponent();  
    label.BindingContext = slider;  
    label.SetBinding(Label.RotationProperty, "Value");  
}
```

Привязка через BindingContext

Только в XAML:

```
<Label Text="TEXT"  
    FontSize="80"  
    HorizontalOptions="Center"  
    VerticalOptions="Center"  
    BindingContext="{x:Reference Name=slider}"  
    Rotation="{Binding Path=Value}" />
```

```
<Slider x:Name="slider"  
    Maximum="360"  
    VerticalOptions="Center" />
```

Привязка без BindingContext

Только в XAML:

```
<Label Text="TEXT"  
    FontSize="40"  
    HorizontalOptions="Center"  
    VerticalOptions="Center"  
    Scale="{Binding Source={x:Reference slider},  
        Path=Value}" />
```

```
<Slider x:Name="slider"  
    Minimum="-2"  
    Maximum="2"  
    VerticalOptions="Center" />
```

Привязка к свойствам в code-behind

```
public MainPage()
{
    InitializeComponent();
    BindingContext = this;
}
private bool _isOk;
public bool IsOk
{
    get=>_isOk;
    set
    {
        if(_isOk == value) return;
        _isOk = value;
        OnPropertyChanged();
    }
}
```


Привязка к свойствам в code-behind

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:MAUI_LK"
  x:DataType="local:MainPage"
  x:Class="MAUI_LK.MainPage">
  <ContentPage.Resources>
    <local:BoolToColorValueConverter x:Key="BoolToColor"/>
  </ContentPage.Resources>
  <VerticalStackLayout Spacing="25" Padding="30,0" VerticalOptions="Center">
    <Border>
      <Entry FontSize="40" Placeholder="Enter"
        HorizontalOptions="Center" VerticalOptions="Center"
        TextChanged="Entry_TextChanged"
        TextColor="{Binding IsOk, Converter={StaticResource BoolToColor}}" />
    </Border>
    <Button Text="Confirm" IsEnabled="{Binding IsOk}" />
  </VerticalStackLayout>
</ContentPage>
```

Привязка к свойствам в code-behind

```
private void Entry_TextChanged(object sender, TextChangedEventArgs e)
{
    IsOk=e.NewTextValue.Length>3;
}
```

Привязка к свойствам в code-behind

```
internal class BoolToColorValueConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        if((bool)value)
            return Colors.Black;
        return Colors.Red;
    }

    public object ConvertBack(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Относительная привязка

Относительные привязки .NET MAUI предоставляют возможность установить источник привязки относительно положения цели привязки.

Они создаются с помощью расширения разметки **RelativeSource** и задаются как свойство **Source** выражения привязки.

Относительная привязка

Расширение разметки `RelativeSource` поддерживается классом `RelativeSourceExtension`, который определяет следующие свойства:

- ❑ Режим типа **`RelativeBindingSourceMode`** описывает расположение источника привязки относительно положения цели привязки.
- ❑ **`AncestorLevel`**, тип `int`, необязательный уровень предка для поиска, когда свойство `Mode` имеет значение `FindAncestor`. `AncestorLevel`, равный `n`, пропускает `n-1` экземпляров `AncestorType`.
- ❑ **`AncestorType`**, тип `Type`, тип искомого предка, когда свойство `Mode` имеет значение `FindAncestor`.

Относительная привязка

Привязка к самому себе:

```
<Border BackgroundColor = "Blue"  
    WidthRequest="200"  
    HeightRequest="{Binding Source={RelativeSource Self}, Path=WidthRequest}"  
    HorizontalOptions="Center" />
```

Относительная привязка

Привязка к родительскому объекту:

```
<CollectionView ItemsSource="{Binding Doctors}" SelectionMode="Single">
    ...
    <CollectionView.ItemTemplate>
        <DataTemplate x:DataType="data:Doctor">
            <Frame Background="WhiteSmoke">
                <Frame.GestureRecognizers>
                    <TapGestureRecognizer
                        Command="{Binding
                            Source={RelativeSource AncestorType={x:Type models:DoctorsListViewModel} },
                            Path=ShowDetailsCommand }"
                        CommandParameter="{Binding Id}"/>
                </Frame.GestureRecognizers>
            </Frame>
        </DataTemplate>
    </CollectionView.ItemTemplate>
</CollectionView>
```

The diagram illustrates relative binding in XAML. A red box highlights the `Doctors` property in the `ItemsSource` binding of the `CollectionView`. Another red box highlights the `Source` property in the `TapGestureRecognizer` command binding. A red curved line connects these two boxes, indicating that the `Source` is resolved relative to the `CollectionView` ancestor.

CollectionView

CollectionView

CollectionView — это представление для вывода списков данных с использованием различных спецификаций макета.

CollectionView

CollectionView включает следующие свойства, определяющие отображаемые данные и их внешний вид:

□ **ItemsSource** типа **IEnumerable** указывает коллекцию отображаемых элементов и имеет значение по умолчанию, равное null.

□ **ItemTemplate** типа **DataTemplate** указывает шаблон, применяемый к каждому элементу в коллекции отображаемых элементов.

Эти свойства поддерживаются объектами **BindableProperty**, что означает, что свойства могут быть целями привязок данных.

CollectionView

```
<CollectionView ItemsSource="{Binding Monkeys}">
  <CollectionView.ItemTemplate>
    <DataTemplate>
      <Grid Padding="10" RowDefinitions="Auto,Auto" ColumnDefinitions="Auto,Auto">
        <Image Grid.RowSpan="2"
          Source="{Binding ImageUrl}" Aspect="AspectFill"
          HeightRequest="60" WidthRequest="60" />
        <Label Grid.Column="1" Text="{Binding Name}" FontAttributes="Bold" />
        <Label Grid.Row="1" Grid.Column="1"
          Text="{Binding Location}" FontAttributes="Italic"
          VerticalOptions="End" />
      </Grid>
    </DataTemplate>
  </CollectionView.ItemTemplate>
</CollectionView>
```

CollectionView

Если CollectionView требуется обновлять по мере добавления, удаления или изменения элементов в базовой коллекции, базовая коллекция должна быть коллекцией IEnumerable, которая отправляет уведомления об изменении свойства, например **ObservableCollection**.

CollectionView **выдаст исключение**, если его ItemsSource будет обновлен вне потока пользовательского интерфейса.

Стили

Стили

В приложениях .NET MAUI страницы часто содержат несколько элементов управления с одинаковым внешним видом.

Однако настройка внешнего вида каждого отдельного элемента управления – это дублирование кода, с возможным появлением ошибок. Вместо этого можно создать стиль, определяющий внешний вид, а затем применить его к нужным элементам управления.

Стили

Приложению можно придать стиль, используя класс `Style`, чтобы сгруппировать коллекцию значений свойств в один объект, который затем можно применить к нескольким визуальным элементам.

Это помогает уменьшить повторяющуюся разметку и упрощает изменение внешнего вида приложений.

Стили

- Явный (explicit) объект **Style** определяется указанием **TargetType** и значения **x:Key**, а также установкой свойства **Style** целевого элемента на ссылку **x:Key**.
- Неявный (implicit) объект **Style** определяется путем указания только **TargetType**. Затем объект **Style** будет автоматически применен ко всем элементам этого типа. Однако к подклассам **TargetType** стиль не применяется автоматически.

Стили

См. файл

Resources/Styles/Styles.xaml

Стили

Свойство **Style.ApplyToDerivedTypes** позволяет применить стиль к элементам управления, производным от базового типа, на который ссылается свойство **TargetType**.

Установка для этого свойства значения **true** позволяет одному стилю ориентироваться на несколько типов при условии, что типы являются производными от базового типа, указанного в свойстве **TargetType**.

Стили

Стили могут наследоваться от других стилей, чтобы уменьшить дублирование и обеспечить повторное использование.

Это достигается установкой свойства **Style.BasedOn** в существующий стиль.

В XAML этого можно добиться, задав для свойства **BasedOn** расширение разметки **StaticResource**, которое ссылается на ранее созданный стиль.

Стили

```
<Style BasedOn="{StaticResource Key=labelBase}"  
    x:Key="labelBold"  
    TargetType="Label">  
    <Setter Property="FontAttributes" Value="Bold" />  
</Style>
```

Стили

Глобальные ресурсы (app.xaml):

```
<Application.Resources>  
  <ResourceDictionary>  
    <ResourceDictionary.MergedDictionaries>  
      <ResourceDictionary Source="Resources/Styles/Colors.xaml" />  
      <ResourceDictionary Source="Resources/Styles/Styles.xaml" />  
    </ResourceDictionary.MergedDictionaries>  
  </ResourceDictionary>  
</Application.Resources>
```

Специфика платформы

Специфика платформы

<VerticalStackLayout>

...

<VerticalStackLayout.BackgroundColor>

<OnPlatform x:TypeArguments="Color">

<On Platform="iOS" Value="Silver" />

<On Platform="Android" Value="Green" />

<On Platform="WinUI" Value="Yellow" />

</OnPlatform>

</VerticalStackLayout.BackgroundColor>

...

</VerticalStackLayout>

Пользовательские элементы

Пользовательские элементы UI

ContentView — это элемент управления, который позволяет создавать настраиваемые повторно используемые элементы управления.

Класс **ContentView** определяет свойство **Content** типа **View**, которое представляет содержимое **ContentView**. Это свойство поддерживается объектом **BindableProperty**, что означает, что оно может быть целью привязок данных и стилей.

Пользовательские элементы UI

```
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="AsclepiusP.Views.RatingView"
  xmlns:local="clr-namespace:AsclepiusP.Views"
  xmlns:drawables="clr-namespace:AsclepiusP.Views.Drawables"
  x:Name="this"
  Loaded="this_Loaded" >
  <ContentView.Resources>
    <drawables:StarDrawable x:Key="StarDrawable"/>
  </ContentView.Resources>

  <HorizontalStackLayout BindingContext="{x:Reference this}">
    <GraphicsView x:Name="Star" Drawable="{StaticResource StarDrawable}"
      WidthRequest="22"
      HeightRequest="22">
    </GraphicsView>
    <Label Text="{Binding Rating, StringFormat='{0:F1}'}" Margin="5,0,0,0" />
  </HorizontalStackLayout>
</ContentView>
```

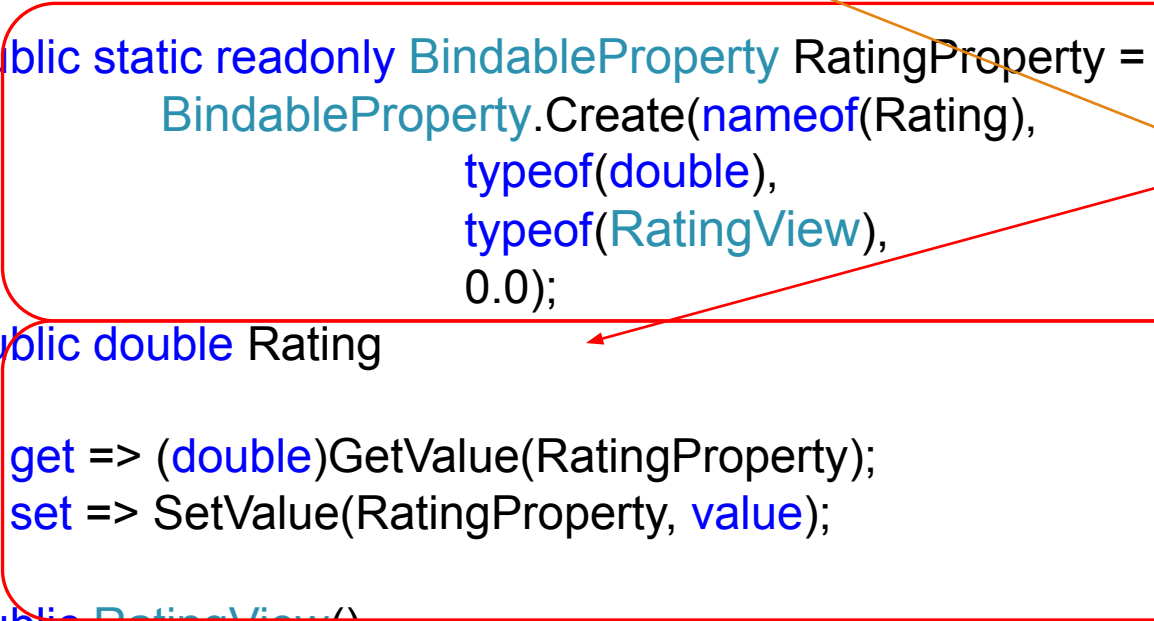
Пользовательские элементы UI

```
public partial class RatingView : ContentView
{
    public static readonly BindableProperty RatingProperty =
        BindableProperty.Create(nameof(Rating),
                                typeof(double),
                                typeof(RatingView),
                                0.0);

    public double Rating
    {
        get => (double)GetValue(RatingProperty);
        set => SetValue(RatingProperty, value);
    }

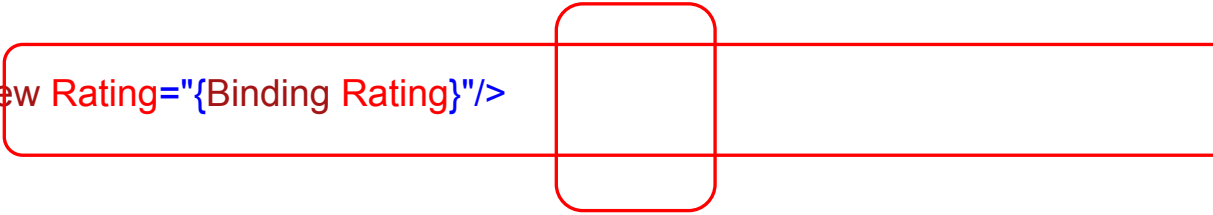
    public RatingView()
    {
        InitializeComponent();
    }

    ...
}
```



Пользовательские элементы UI

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:converters="clr-namespace:AsclepiusP.ValueConverters"
  xmlns:controls="clr-namespace:AsclepiusP.Views">
  <CollectionView ItemsSource="{Binding Doctors}" SelectionMode="Single">
    <CollectionView.ItemTemplate>
      <DataTemplate>
        <Frame Background="WhiteSmoke">
          <Frame.GestureRecognizers>
            <TapGestureRecognizer
              Command="{Binding Source={RelativeSource AncestorType={x:Type models:DoctorsListViewModel} },
                Path=ShowDetailsCommand }"
              CommandParameter="{Binding Id}"/>
          </Frame.GestureRecognizers>
          <controls:RatingView Rating="{Binding Rating}"/>
        </Frame>
      </DataTemplate>
    </CollectionView.ItemTemplate>
  </CollectionView>
</ContentPage>
```



Triggers

Triggers

Триггеры позволяют декларативно выражать действия в XAML, которые изменяют внешний вид элементов управления на основе событий или изменений данных.

Property Triggers

```
<Entry Placeholder="Enter name">  
  <Entry.Triggers>  
    <Trigger TargetType="Entry"  
      Property="IsFocused"  
      Value="True">  
      <Setter Property="BackgroundColor"  
        Value="Yellow" />  
    <!-- Multiple Setter elements are allowed -->  
  </Trigger>  
</Entry.Triggers>  
</Entry>
```

<https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/triggers#property-triggers>

Style Triggers

```
<Style TargetType="Entry">
  <Style.Triggers>
    <Trigger TargetType="Entry"
      Property="IsFocused"
      Value="True">
      <Setter Property="BackgroundColor"
        Value="Yellow" />
    <!-- Multiple Setter elements are allowed -->
  </Trigger>
</Style.Triggers>
</Style>
```

<https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/triggers#apply-a-trigger-using-a-style>

Data Triggers

```
<Button Text="Save">
  <Button.Triggers>
    <DataTrigger TargetType="Button"
      Binding="{Binding Source={x:Reference entry},
        Path=Text.Length}"
      Value="0">
      <Setter Property="IsEnabled"
        Value="False" />
      <!-- Multiple Setter elements are allowed -->
    </DataTrigger>
  </Button.Triggers>
</Button>
```

<https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/triggers#data-triggers>

Многопоточность

Многопоточность

Большинство операционных систем используют **однопоточную** модель для кода, связанного с **пользовательским интерфейсом**. Эта модель необходима для правильной сериализации событий пользовательского интерфейса, включая нажатия клавиш и сенсорный ввод.

Этот поток часто называют **основным потоком**, **поток** пользовательского интерфейса или **поток** пользовательского интерфейса.

Многопоточность

Недостатком этой модели является то, что весь код, обращающийся к элементам пользовательского интерфейса, должен выполняться **в основном потоке приложения.**

Многопоточность

Пример: датчики акселерометра или компаса. Все датчики могут возвращать информацию во вторичном потоке при использовании с более высокими скоростями обнаружения. Если обработчику событий требуется доступ к элементам пользовательского интерфейса, он должен вызывать код в основном потоке.

МНОГОПОТОЧНОСТЬ

```
async Task UpdateOrdersAsync()
```

```
{  
    if (IsBusy) return;  
    IsRefreshing = true;  
    var orders = await _orderService.GetOrdersAsync();  
    if (orders == null && !orders.Any())  
    {  
        IsRefreshing=false;  
        return;  
    };  
    MainThread.BeginInvokeOnMainThread(() =>
```

```
{  
    IsBusy = true;  
    orders = orders.Where(o => o.State != OrderState.Complete);  
    Orders.Clear();  
    foreach (var order in orders)  
        Orders.Add(order);  
});
```

```
IsBusy = false;  
IsRefreshing = false;  
}
```

Orders – список типа
`ObservableCollection`

Многопоточность

С помощью класса `MainThread` можно определить, выполняется ли текущий код в основном потоке. Свойство `MainThread.IsMainThread` возвращает значение **true**, если код, вызывающий это свойство, выполняется в основном потоке, и значение **false**, если это не так.

МНОГОПОТОЧНОСТЬ

```
if (MainThread.IsMainThread)  
    MyMainThreadCode();
```

```
else  
    MainThread.BeginInvokeOnMainThread(MyMainThreadCode);
```