

Элементы в Xamarin и их свойства

Позиционирование элементов на странице

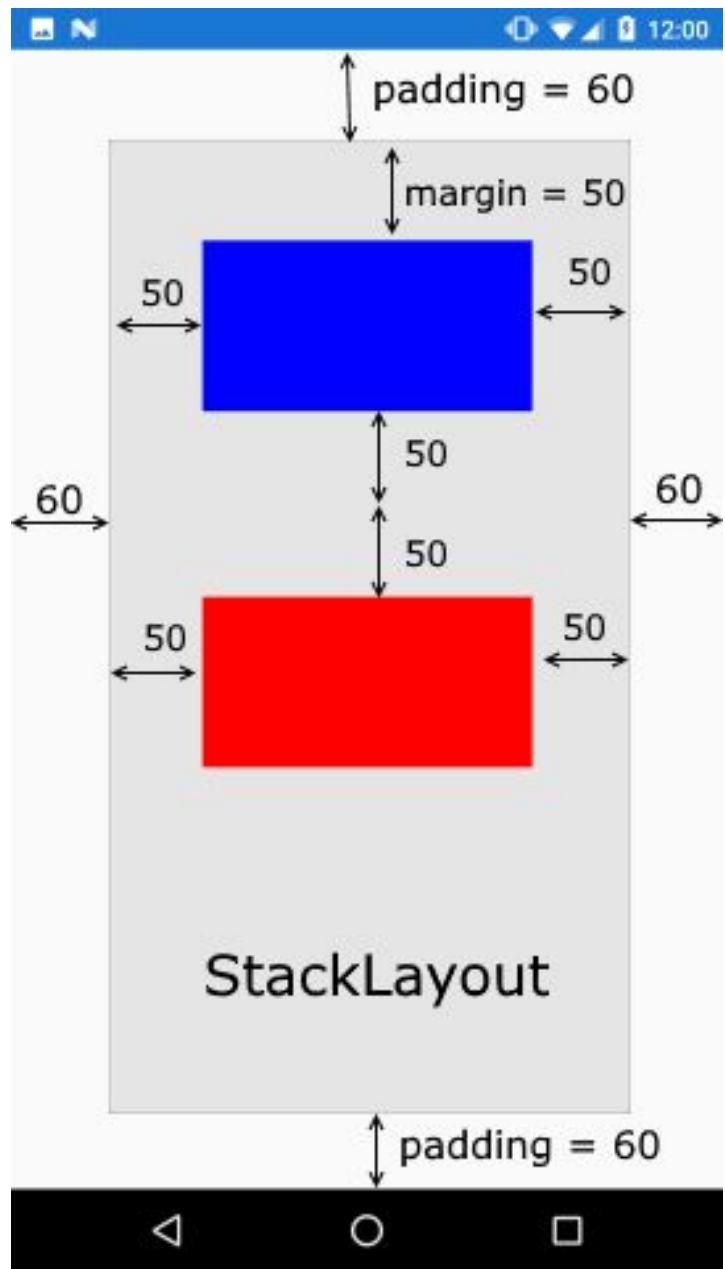
Отступы

Для установки отступов у элементов применяются свойства **Margin** и **Padding**.

Свойство **Margin** определяет внешний отступ элемента от других элементов или контейнера. А свойство **Padding** устанавливает внутренние отступы - от внутреннего содержимого элемента до его границ. Оба этих свойства представляют структуру **Thickness**.

B XAML:

```
<StackLayout Padding="60">  
    <BoxView Color="Blue" Margin="50"  
HeightRequest="100" />  
    <BoxView Color="Red" Margin="50"  
HeightRequest="100" />  
</StackLayout>
```



Установка в коде C#:

```
var stackLayout = new StackLayout
{
    Padding = new Thickness(60),
    Children = {
        new BoxView { Color = Color.Blue, Margin = new Thickness (50) },
        new BoxView { Color = Color.Red, Margin = new Thickness (50) }
    }
};
Content = stackLayout;
```

Структура **Thickness** имеет свойства **Left, Top, Right, Bottom**, поэтому задать ее определение мы можем тремя способами:

- Указать одно значение для отступов со всех сторон
- Указать два значения для отступов по горизонтали (слева и справа) и вертикали (сверху и снизу)
- Указать четыре значения для отступов для каждой из сторон

Установка в коде:

```
var stackLayout = new StackLayout
{
    Padding = new Thickness(0, 20, 0, 0),
    Children = {
        new BoxView { Color = Color.Green,
Margin = new Thickness (20) },
        new BoxView { Color = Color.Blue,
Margin = new Thickness (10, 25) },
        new BoxView { Color = Color.Red, Margin
= new Thickness (0, 20, 15, 5) }
    }
};
Content = stackLayout;
```

Определение в XAML:

```
<StackLayout Padding="0,20,0,0">  
    <BoxView Color="Green" Margin="20" />  
    <BoxView Color="Blue" Margin="10, 15" />  
    <BoxView Color="Red" Margin="0, 20, 15,  
5" />  
    </StackLayout>
```


В отношении отступа надо учитывать следующую вещь: так как на устройствах с iOS верхняя панель занимает 20 единиц, то для iOS желательно устанавливать отступ сверху размером как минимум в 20 единиц, чтобы текст метки не налезил на панель.

В этом случае можно глобально установить отступ сверху для всех платформ в 20 или более единиц. Либо можно задать отступ непосредственно только для iOS:

```
Label header = new Label() { Text = "Привет из  
Xamarin Forms" };  
    this.Content = header;  
    if(Device.RuntimePlatform == Device.iOS)  
        Padding = new Thickness(0, 20, 0, 0);
```

с помощью значения **Device.RuntimePlatform** можем получить текстовую метку текущей платформы, на которой запущено приложение, и сравнить ее. Константа "Device.iOS" фактически хранит значение "iOS". То есть, если текущая платформа - iOS, установить определенный отступ.

Выравнивание по горизонтали и вертикали

Все элементы, используемые при создании интерфейса, наследуются от класса `View`, который определяет два свойства `HorizontalOptions` и `VerticalOptions`. Они управляют выравнивание элемента соответственно по горизонтали и по вертикали.

В качестве значения они принимают структуру **LayoutOptions**. Данная структура имеет ряд свойств, которые хранят объекты опять же **LayoutOptions**:

- **Start**: выравнивание по левому краю (выравнивание по горизонтали) или по верху (выравнивание по вертикали)
- **Center**: элемент выравнивается по центру
- **End**: выравнивание по правому краю (выравнивание по горизонтали) или по низу (выравнивание по вертикали)
- **Fill**: элемент заполняет все пространство контейнера
- **StartAndExpand**: аналогичен опции **Start** с применением растяжения
- **CenterAndExpand**: аналогичен опции **Center** с применением растяжения
- **EndAndExpand**: аналогичен опции **End** с применением растяжения
- **FillAndExpand**: аналогичен опции **Fill** с применением растяжения

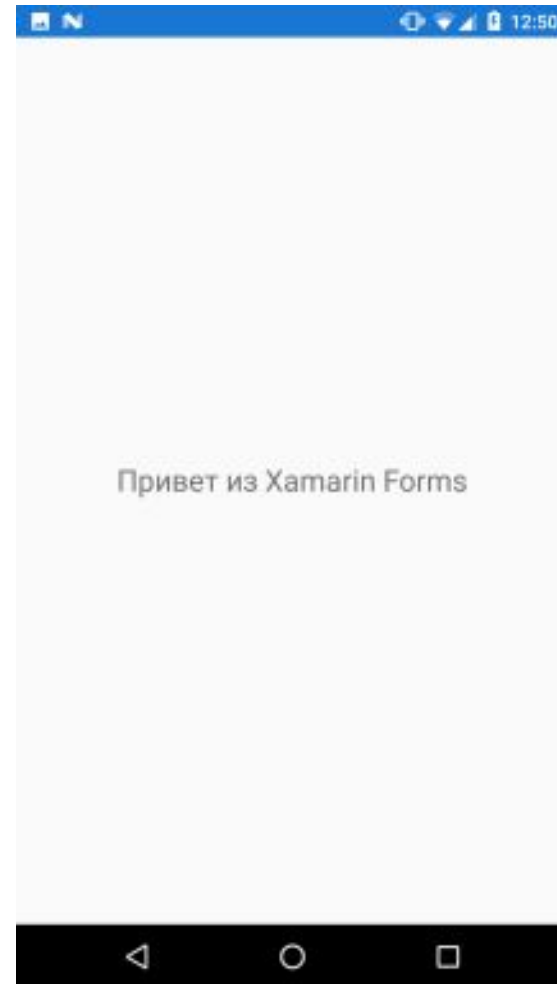
в коде C#:

```
Label header = new Label() { Text  
= "Привет из Xamarin Forms" };  
header.VerticalOptions =  
LayoutOptions.Center;  
header.HorizontalOptions =  
LayoutOptions.Center;
```

В

хамл:

```
<Label Text="Привет из  
Xamarin Forms"  
VerticalOptions="Center"  
HorizontalOptions="Center" />
```



Выравнивание текста внутри элемента

Выравнивание текста по горизонтали и вертикали задается с помощью

свойств **HorizontalTextAlignment** и **VerticalTextAlignment** соответственно. В качестве значения эти свойства принимают одну из констант перечисления **TextAlignment**:

- **Start**: текст выравнивается по левому краю по горизонтали или по верху по вертикали
- **Center**: текст выравнивается по центру
- **End**: текст выравнивается по правому краю по горизонтали или по низу по вертикали

- на C#:

```
Label header = new Label() { Text = "Привет из  
Xamarin Forms" };
```

```
header.HorizontalTextAlignment =  
TextAlignment.Center;
```

```
header.VerticalTextAlignment =  
TextAlignment.Center;
```

- В xaml:

```
<Label Text="Привет из Xamarin Forms"  
VerticalTextAlignment="Center"  
HorizontalTextAlignment="Center" />
```

Работа с цветом

За установку цвета фона и текста элементов отвечают свойства `BackgroundColor` и `TextColor` соответственно. В качестве значения они принимают структуру `Color`.

```
Label header = new Label() { Text = "Привет из  
Xamarin Forms" };
```

```
header.HorizontalTextAlignment =  
TextAlignment.Center;
```

```
header.VerticalTextAlignment =  
TextAlignment.Center;
```

```
header.BackgroundColor = Color.Teal;
```


Или в XAML:

```
<Label Text="Привет из  
Xamarin Forms"  
HorizontalTextAlignmen  
t="Center"  
VerticalTextAlignment="  
Center"  
  
BackgroundColor="Blue"  
TextColor="Yellow" />
```



Кроме встроенных констант типа `Color.Red` также для установки цвета можно указать и другие значения, используя структуру `Color`.

Для этого надо передать значение для одной из компонент красного, зеленого, синего цветов, смесь которых даст финальный цвет. Передаваемое значение имеет тип `double` и должно находиться в диапазоне от `0.0` до `1.0`.

Чтобы передать эти значения, можно использовать один из конструкторов структуры `Color`:

- `new Color(double grayShade)`: устанавливает тон серого цвета
- `new Color(double r, double g, double b)`: устанавливает компоненты красного, зеленого и синего
- `new Color(double r, double g, double b, double a)`: добавляет еще один параметр - `a`, который передает прозрачность и имеет значение от `0.0` (полностью прозрачный) до `1.0` (не прозрачный)

```
Label header = new Label() { Text = "Привет из  
Xamarin Forms" };  
header.BackgroundColor = new Color(0.9, 0.9, 0.8);  
//rgb
```

Также для установки цвета можно использовать ряд статических методов:

- `Color.FromHex(string hex)`: возвращает объект `Color`, созданный по переданному в качестве параметра шестнадцатеричному значению
- `Color.FromRgb(double r, double g, double b)`: возвращает объект `Color`, для которого также устанавливаются компоненты красного, зеленого и синего
- `Color.FromRgba(int r, int g, int b)`: аналогичен предыдущей версии метода, только теперь компоненты красного, зеленого и синего имеют целочисленные значения от 0 до 255
- `Color.FromRgba(double r, double g, double b, double a)`: добавляет параметр прозрачности со значением от 0.0 (полностью прозрачный) до 1.0 (не прозрачный)
- `Color.FromRgba(int r, int g, int b, int a)`: добавляет параметр прозрачности со значением от 0 (полностью прозрачный) до 255 (не прозрачный)
- `Color.FromHsla(double h, double s, double l, double a)`: устанавливает последовательно параметры `h` (hue - тон цвета), `s` (saturation - насыщенность), `l` (luminosity - яркость) и прозрачность.

```
Label header = new Label() { Text = "Привет из  
Xamarin Forms" };  
header.TextColor = Color.FromRgba(255, 0, 0,  
160);
```

В xaml

```
<Label Text="Привет из Xamarin Forms"  
BackgroundColor="#a7a7aa" TextColor="Red" />
```

Стилизация текста

Для стилизации вывода мы можно использовать ряд свойств:

- **FontFamily**: устанавливает применяемое семейство шрифтов
- **FontSize**: устанавливает высоту шрифта
- **FontAttributes**: устанавливает дополнительные визуальные эффекты шрифта - выделение жирным или курсивом

Для установки семейства используемых шрифтов свойству **FontFamily** в качестве значения передается название шрифта. Однако при установке шрифта надо учитывать, что данный шрифт должен поддерживаться на всех используемых платформах.

- установка в коде:

```
Label header = new Label() { Text = "Xamarin  
Forms in Arial" };  
header.FontFamily = "Arial";
```

- Установка в XAML:

```
<Label Text="Xamarin Forms in Arial"  
FontFamily="Arial" />
```

Свойство `FontSize`

Для установки высоты шрифта используется значение типа `double`:

```
Label header = new Label() { Text = "Xamarin  
Forms" };  
header.FontSize = 26;
```

Установка в XAML:

```
<Label Text="Xamarin Forms" FontSize="26" />
```


В классе **Device** определен статический метод **GetNamedSize()**, с помощью которого можно решить, какой лучше шрифт использовать на том или ином устройстве.

Этот метод в качестве первого параметра требует одну из констант из перечисления **NamedSize**:

- Default
- Micro
- Small
- Medium
- Large
- Subtitle
- Title
- Body
- Caption
- Header

Все эти константы представляют разные размеры шрифтов от маленького до большого, характерные непосредственно для платформы, на которой запускается приложение.

В качестве второго параметра передается тип элемента, для которого устанавливается шрифт:

```
Label header = new Label() { Text = "Привет из  
Xamarin Forms" };  
this.Content = header;  
header.FontSize =  
Device.GetNamedSize(NamedSize.Large,  
typeof(Label));
```

Установка в XAML:

```
<Label Text="Xamarin Forms" FontSize="Large" />
```

FontAttributes

Позволяет выделить текст жирным или курсивом. Для этого он принимает значение из перечисления `FontAttributes`:

- **Bold**: выделение жирным
- **Italic**: выделение курсивом

```
Label header = new Label() { Text = "Привет из  
Xamarin Forms" };  
header.FontAttributes = FontAttributes.Bold;
```

```
<Label Text="Xaml в Xamarin"  
FontAttributes="Bold" />
```

Форматирование текста

Свойство **FormattedText** допускает сложное форматирование.

Свойство **FormattedText** хранит не просто строку текста, а объект типа **FormattedString**, который инкапсулирует объекты типа **Span**. Каждый объект **Span** как раз и представляет кусок некоторым образом стилизованного текста. Для стилизации объекта **Span** применяются все те же свойства:

- Text
- FontFamily
- FontSize
- FontAttributes
- ForegroundColor
- BackgroundColor

текст со сложным форматированием в коде:

```
Label header = new Label();
    this.Content = header;
    FormattedString formattedString = new FormattedString();
    formattedString.Spans.Add(new Span
    {
        Text = "Сегодня ",
        FontSize = Device.GetNamedSize(NamedSize.Large,
typeof(Label))
    });
    formattedString.Spans.Add(new Span
    {
        Text = "хорошая",
        FontAttributes = FontAttributes.Bold
    });
    formattedString.Spans.Add(new Span
    {
        Text = " погода!",
        ForegroundColor=Color.Red
    });
    header.FormattedText = formattedString;
    header.VerticalTextAlignment = TextAlignment.Center;
    header.HorizontalTextAlignment = TextAlignment.Center;
```

В XAML:

```
<Label VerticalTextAlignment="Center"  
HorizontalTextAlignment="Center">  
  <Label.FormattedText>  
    <FormattedString>  
      <Span Text="Сегодня " FontSize="Large"  
/>  
      <Span Text="хорошая "  
FontAttributes="Bold" />  
      <Span Text="погода!"  
ForegroundColor="Red" />  
    </FormattedString>  
  </Label.FormattedText>  
</Label>
```

Android status bar with icons for back, home, and recent apps, along with signal strength, Wi-Fi, and battery indicators, and the time 13:29.

Сегодня хорошая **погода!**

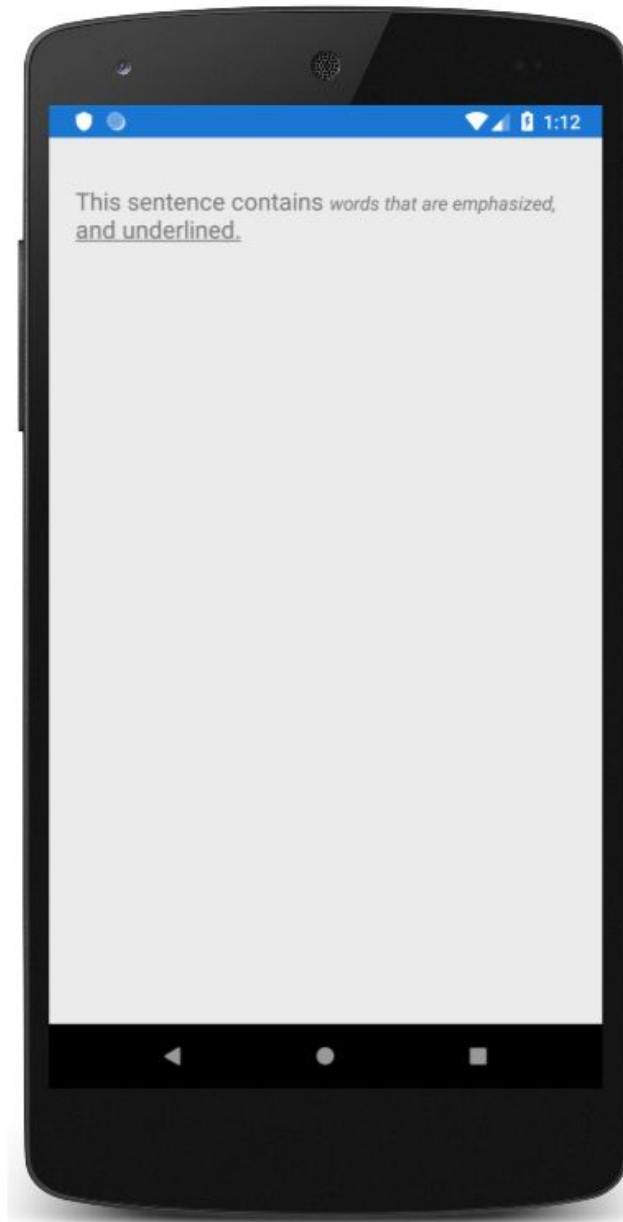


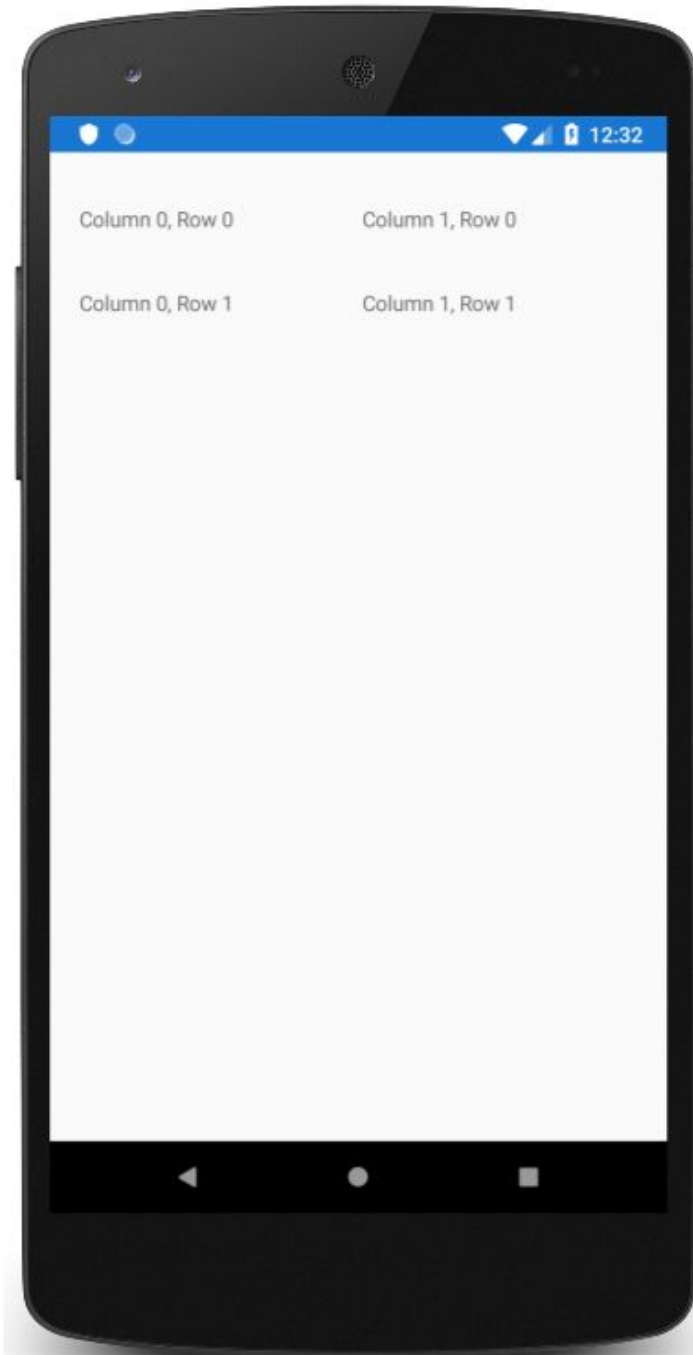
Перевод строки

Для переноса текста на новую строку используется значение "\n".

```
header.Text= "Его пример другим наука;\n"+  
    "Но, боже мой, какая скука\n" +  
    "С больным сидеть и день и ночь,\n" +  
    "Не отходя ни шагу прочь!\n";
```







КНОПКИ

```
StackLayout stackLayout = new StackLayout();

    Button button = new Button
    {
        Text = "Нажми!",
        FontSize =
Device.GetNamedSize(NamedSize.Large, typeof(Button)),
        BorderWidth = 1,
        HorizontalOptions = LayoutOptions.Center,
        VerticalOptions =
LayoutOptions.CenterAndExpand
    };
    button.Clicked += OnButtonClicked;
    stackLayout.Children.Add(button);
    this.Content = stackLayout;
```

Для кнопки можно задать обработчик нажатия для события `Clicked`.

```
private void OnButtonClicked(object sender,
System.EventArgs e)
{
    Button button = (Button)sender;
    button.Text = "Нажато!";
    button.BackgroundColor = Color.Red;
}
```

Обработчик во многих аналогичен стандартным обработчикам в `Windows Forms`.

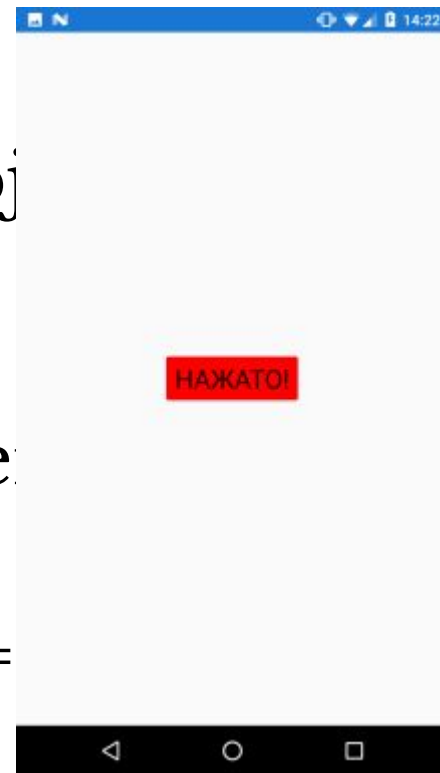
Он принимает два параметра: объект типа `object` (источника события) и `System.EventArgs` (аргумент события, хранящий некоторую дополнительную информацию).

Аналогичная кнопка в xaml:

```
<StackLayout>  
  <Button Text = "Нажми!" FontSize="Large"  
  BorderWidth="1"  
    HorizontalOptions="Center"  
    VerticalOptions="CenterAndExpand"  
    Clicked="OnButtonClicked" />  
</StackLayout>
```

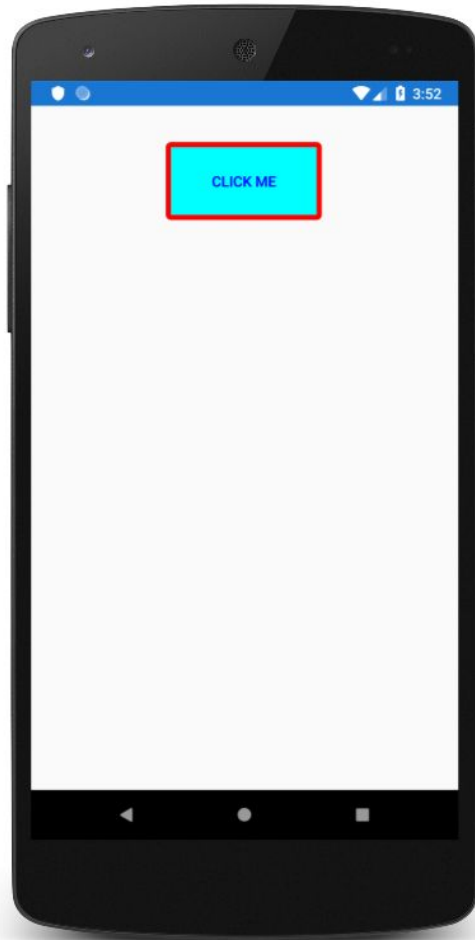
И тогда в файле отдельного кода MainPage.xaml.cs еще надо прописать обработчик OnButtonClicked:

```
private void OnButtonClicked(object sender, System.EventArgs e)
{
    Button button = (Button)sender;
    button.Text = "Нажато!";
    button.BackgroundColor =
}
```



Кнопка имеет много различных свойств, из которых следует выделить следующие:

- **FontFamily:** шрифт, который используется для текста на кнопке
- **FontSize:** размер текста на кнопке
- **FontAttributes:** выделение жирным или курсивом текста на кнопке
- **TextColor:** цвет шрифта
- **BorderColor:** цвет границы
- **BorderWidth:** ширина границы
- **CornerRadius:** радиус границы
- **Image:** позволяет задать изображение на кнопке



Текстовые поля

В Xamarin Forms текстовые поля представлены несколькими классами:

- **Label:** текстовая метка для вывода текста
- **Entry:** однострочное текстовое поле
- **Editor:** многострочное текстовое поле

Label

Label представляет обычную текстовую метку, которая выводит информацию с помощью свойства Text. Label удобен для создания заголовков и меток к элементам ввода.

Для обработки нажатия на заголовок метки можно воспользоваться специальным классом **TapGestureRecognizer**, который позволяет обрабатывать нажатия.

TapGestureRecognizer представляет специальный класс, который позволяет распознать нажатия. С помощью свойства **NumberOfTapsRequired** можно установить, сколько нажатий необходимо.

```
Label label = new Label
{
    VerticalTextAlignment =
TextAlignment.Center,
    HorizontalTextAlignment =
TextAlignment.Center,
    Text = "Welcome to Xamarin Forms!",
    FontSize =
Device.GetNamedSize(NamedSize.Large,
typeof(Label))
};

TapGestureRecognizer tapGesture = new
TapGestureRecognizer
{
    NumberOfTapsRequired = 2
};
```

```
int count = 0; // счетчик нажатий
tapGesture.Tapped += (s, e) =>
{
    count++;
    if (count % 2 == 0)
    {
        label.BackgroundColor = Color.Black;
        label.TextColor = Color.White;
    }
    else
    {
        label.TextColor = Color.Black;
        label.BackgroundColor = Color.White;
    }
    label.Text = $"Вы нажали {count} раз";
};
label.GestureRecognizers.Add(tapGesture); // для связи текста с меткой нужно
добавить объект tapGesture в коллекцию

Content = label;
```

по двойному нажатию
будет генерироваться
событие **Tapped**, в
обработчике которого
переключаются цвет
текста и фона метки

Текстовое поле Entry

Entry представляет текстовое поле для ввода однострочной информации:

```
Label textLabel;
Entry loginEntry, passwordEntry;
public MainPage()
{
    StackLayout stackLayout = new StackLayout();

    loginEntry = new Entry { Placeholder = "Login" };
    loginEntry.TextChanged+=loginEntry_TextChanged;

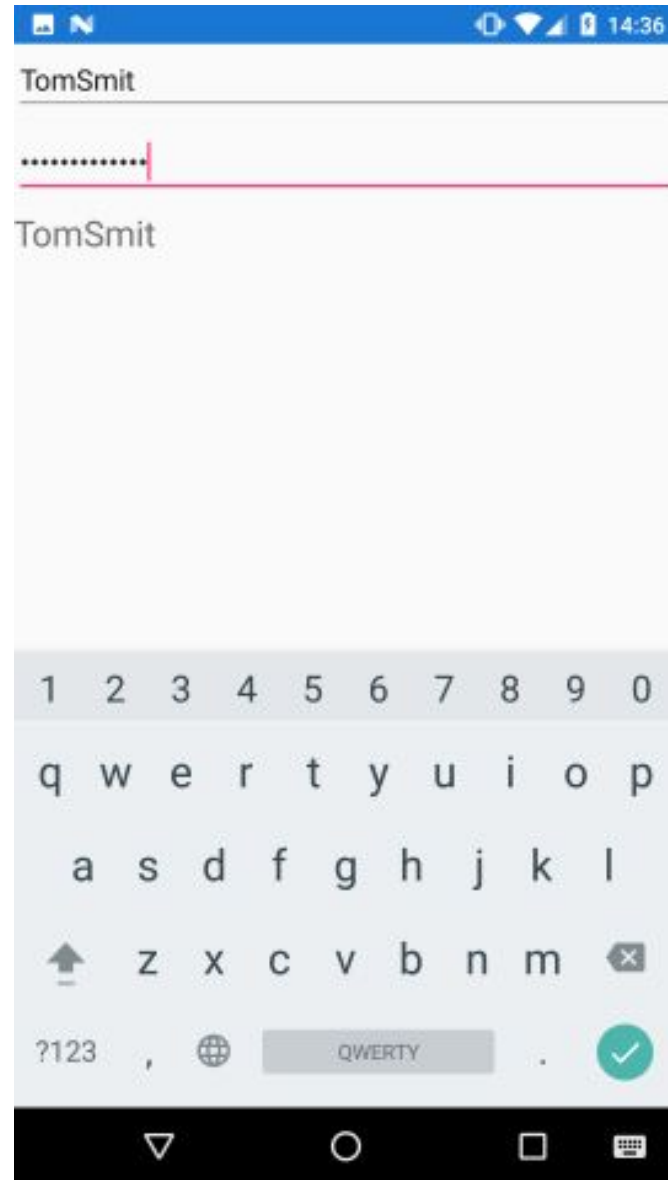
    passwordEntry = new Entry
    {
        Placeholder = "Password",
        IsPassword = true
    };
}
```

```
textLabel = new Label { FontSize =  
Device.GetNamedSize(NamedSize.Large, typeof(Label)) };
```

```
    stackLayout.Children.Add(loginEntry);  
    stackLayout.Children.Add(passwordEntry);  
    stackLayout.Children.Add(textLabel);  
    this.Content = stackLayout;  
}
```

```
private void loginEntry_TextChanged(object sender,  
TextChangedEventArgs e)  
{  
    textLabel.Text = loginEntry.Text;  
}
```

```
<StackLayout>
  <Entry x:Name="loginEntry"
  TextChanged="loginEntry_TextC
hanged" />
  <Entry
x:Name="passwordEntry"
Placeholder = "Password"
IsPassword = "True" />
  <Label x:Name="textLabel"
FontSize="Large" />
</StackLayout>
```



Основные свойства элемента **Entry**, которые можно использовать:

- **Text**: введенный в поле текст
- **TextColor**: цвет вводимого текста
- **Placeholder**: текст-заменитель, который отображается в поле до ввода и исчезает при начале печати
- **IsPassword**: при значении **true** указывает, то поле будет предназначено для ввода пароля, и поэтому все вводимые символы будут заменяться звездочкой
- Также класс **Entry** определяет два события:
- **TextChanged**: возникает при вводе символов в поле
- **Completed**: возникает при завершении ввода

Формат клавиатуры

Среди свойств текстового поля ввода наиболее интересным является свойство **Keyboard**. Оно позволяет задать формат клавиатуры для ввода символов. Например, если необходимо ввести в поле какую-то числовую информацию, то гораздо было бы проще с точки зрения юзабилити предоставить пользователю сразу числовую раскладку клавиатуры. Данное свойство принимает одно из значений перечисления **Keyboard**:

- Default
- Text
- Chat
- Url
- Email
- Telephone
- Numeric

Каждое значение предоставляет свою раскладку клавиатуры, предназначенную для ввода определенной информации.

Например, `entry.Keyboard=Keyboard.Telephone` - при вводе нам автоматически представляется раскладка, содержащая только те знаки, которые используются при наборе телефонного номера.

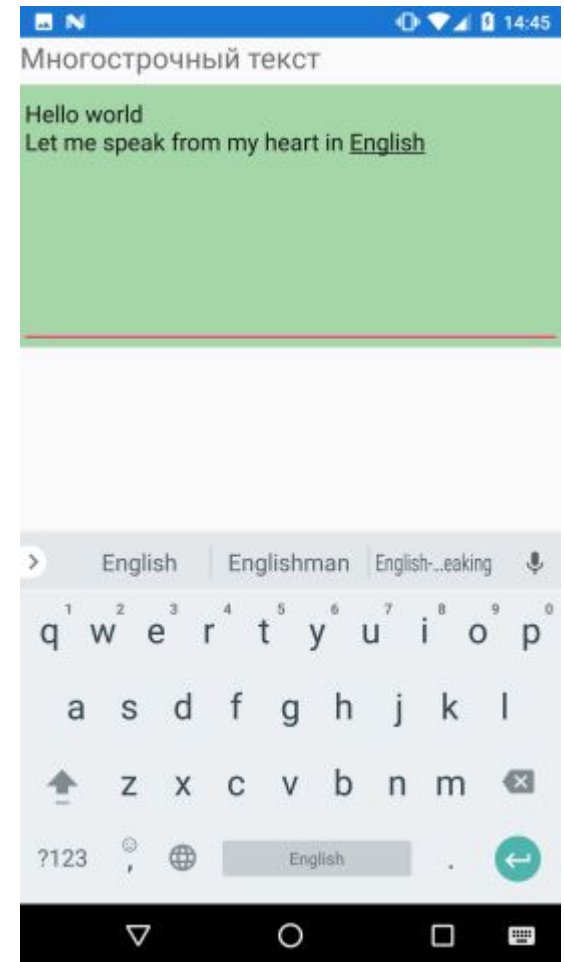
Editor

В отличие от **Entry Editor** представляет собой многострочное поле ввода, но принципы его работы аналогичны. Он также имеет все те же свойства и события.

```

StackLayout stackLayout = new
StackLayout();
Editor textEditor = new Editor {
HeightRequest=200,
BackgroundColor=Color.Teal};
Label textLabel = new Label
{
    Text="Многострочный текст",
    FontSize=
Device.GetNamedSize(NamedSize.Large,
typeof(Label))
};
stackLayout.Children.Add(textLabel);
stackLayout.Children.Add(textEditor);
this.Content = stackLayout;
<StackLayout>
    <Label Font="26" Text="Многострочный
ввод" />
    <Editor BackgroundColor="#a5d6a7"
HeightRequest="200" />
</StackLayout>

```



Контейнер **Frame**

Контейнер **Frame**, как правило используется для оформления или создания фона для вложенного элемента.

Среди свойств класса **Frame** следует выделить следующие:

- **BorderColor**: представляет цвет границы фрейма с помощью структуры **Color**.
- **CornerRadius**: представляет радиус границы фрейма в виде значения типа **float**.
- **HasShadow**: хранит значение типа **bool**, которое указывает, будет ли фрейм отбрасывать тень.

Фрейм может вмещать только один элемент. Например, создание фрейма в XAML:

```
<Frame>  
  <Label Text="Hello Xamarin" />  
</Frame>
```

в коде C#:

```
Frame frame = new Frame  
{  
    Content = new Label { Text = "Hello Xamarin" }  
};
```

BoxView

BoxView представляет прямоугольник. Чаще всего **BoxView** используется для создания окрашенных областей, либо в качестве декоративного примитивного графического оформления к другим элементам.

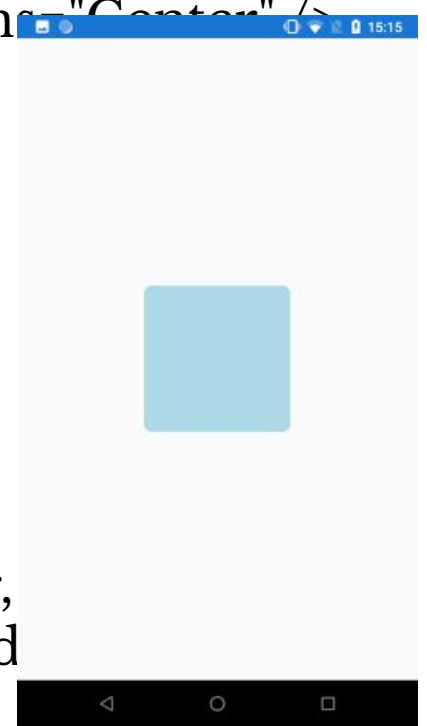
Основные свойства класса `BoxView`:

- **Color**: представляет цвет элемента в виде структуры `Color`.
- **CornerRadius**: представляет радиус границы `BoxView` в виде значения типа `float`.
- **WidthRequest**: представляет ширину элемента (по умолчанию равна 40 единицам).
- **HeightRequest**: представляет высоту элемента (по умолчанию равна 40 единицам).


```
<StackLayout>
  <BoxView Color="LightBlue" WidthRequest="150"
HeightRequest="150" CornerRadius="8"
  HorizontalOptions="Center" VerticalOptions="Center" />
</StackLayout>
```

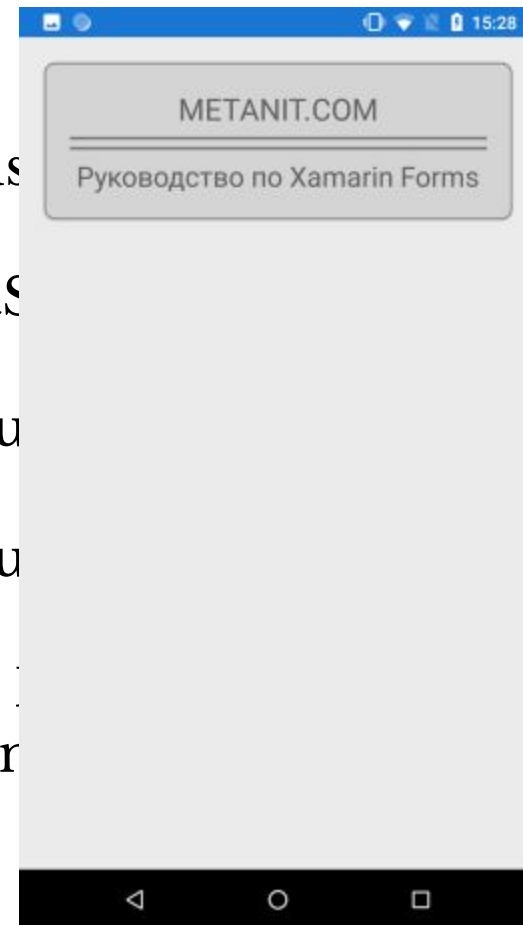
```
BoxView boxView = new BoxView
{
    Color = Color.LightBlue,
    CornerRadius = 8,
    WidthRequest = 150,
    HeightRequest = 150,
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.CenterAnd
};
```

```
StackLayout stackLayout = new StackLayout() { Children = {
boxView }};
Content = stackLayout;
```



Пример применение BoxView. (двойная линия)

```
<StackLayout Padding="20">
  <Frame BorderColor="Gray"
    BackgroundColor="#e1e1e1" CornerRadius="10">
    <StackLayout>
      <Label Text="METANIT.COM" FontSize="24"
        HorizontalOptions="Center" />
      <BoxView Color="Gray" HeightRequest="1"
        HorizontalOptions="Fill" />
      <BoxView Color="Gray" HeightRequest="1"
        HorizontalOptions="Fill" />
      <Label Text="Руководство по Xamarin Forms"
        FontSize="Large" HorizontalOptions="Center" />
    </StackLayout>
  </Frame>
</StackLayout>
```



- В C#:

```
Label label1 = new Label
{
    Text = "METANIT.COM",
    FontSize =
Device.GetNamedSize(NamedSize.Title, typeof(Label)),
    HorizontalOptions = LayoutOptions.Center
};
Label label2 = new Label
{
    Text = "Руководство по Xamarin Forms",
    FontSize =
Device.GetNamedSize(NamedSize.Large, typeof(Label)),
    HorizontalOptions = LayoutOptions.Center
};
BoxView boxView1 = new BoxView
{
    Color = Color.Gray,
    HeightRequest = 2,
    HorizontalOptions = LayoutOptions.Fill
};
```

```
BoxView boxView2 = new BoxView
{
    Color = Color.Gray,
    HeightRequest = 2,
    HorizontalOptions = LayoutOptions.Fill
};
StackLayout innerStackLayout = new StackLayout { Children = {
label1, boxView1, boxView2, label2 } };
Frame frame = new Frame
{
    Content = innerStackLayout,
    BorderColor = Color.Gray,
    BackgroundColor = Color.FromHex("#e1e1e1"),
    CornerRadius = 8
};

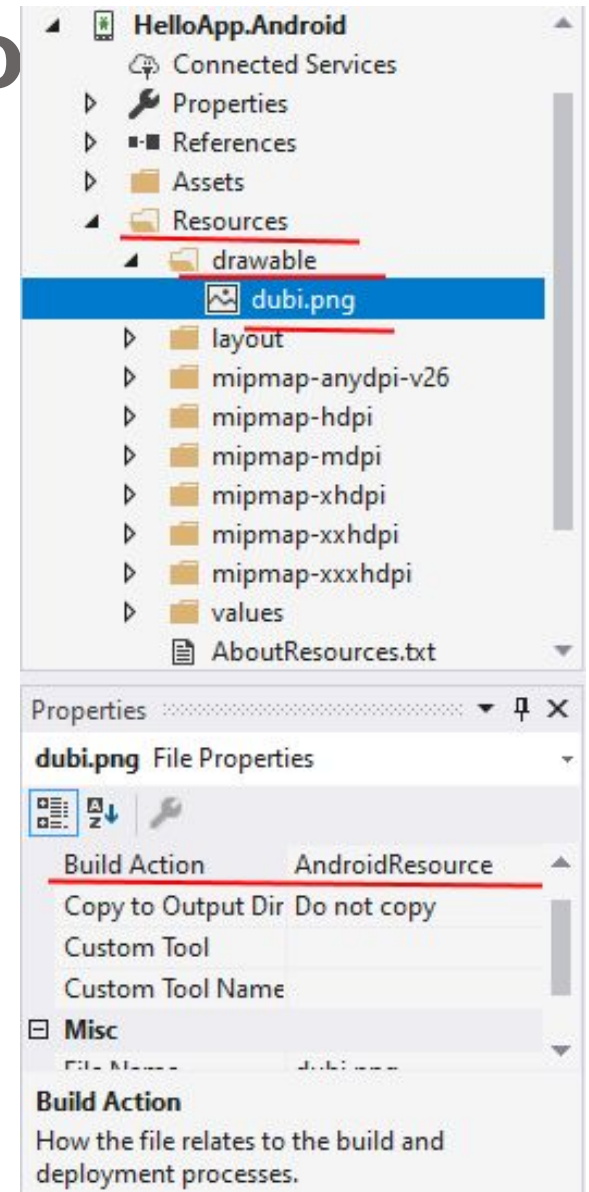
StackLayout stackLayout = new StackLayout() { Children = {
frame }, Padding = 20};
Content = stackLayout;
```

Работа с изображениями. Элемент **Image**

Для вывода изображение имеется элемент **Image**. Но перед выводом изображения его надо добавить в проекты, причем в проект для каждой отдельной ос.

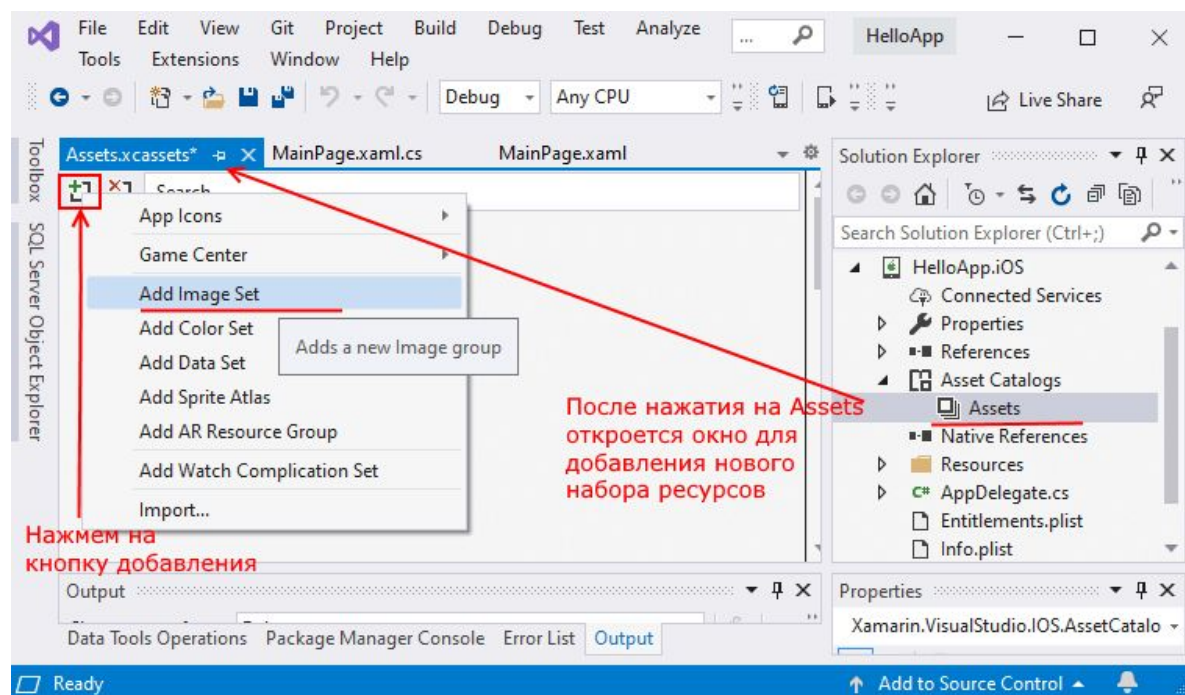
Локальные изображения Добавление для Android

В проекте для Android
добавить файл изображения
В
папку **Resources/Drawable**.
А в окне **Properties**
установить у этого
изображения **Build Action:**
AndroidResource.

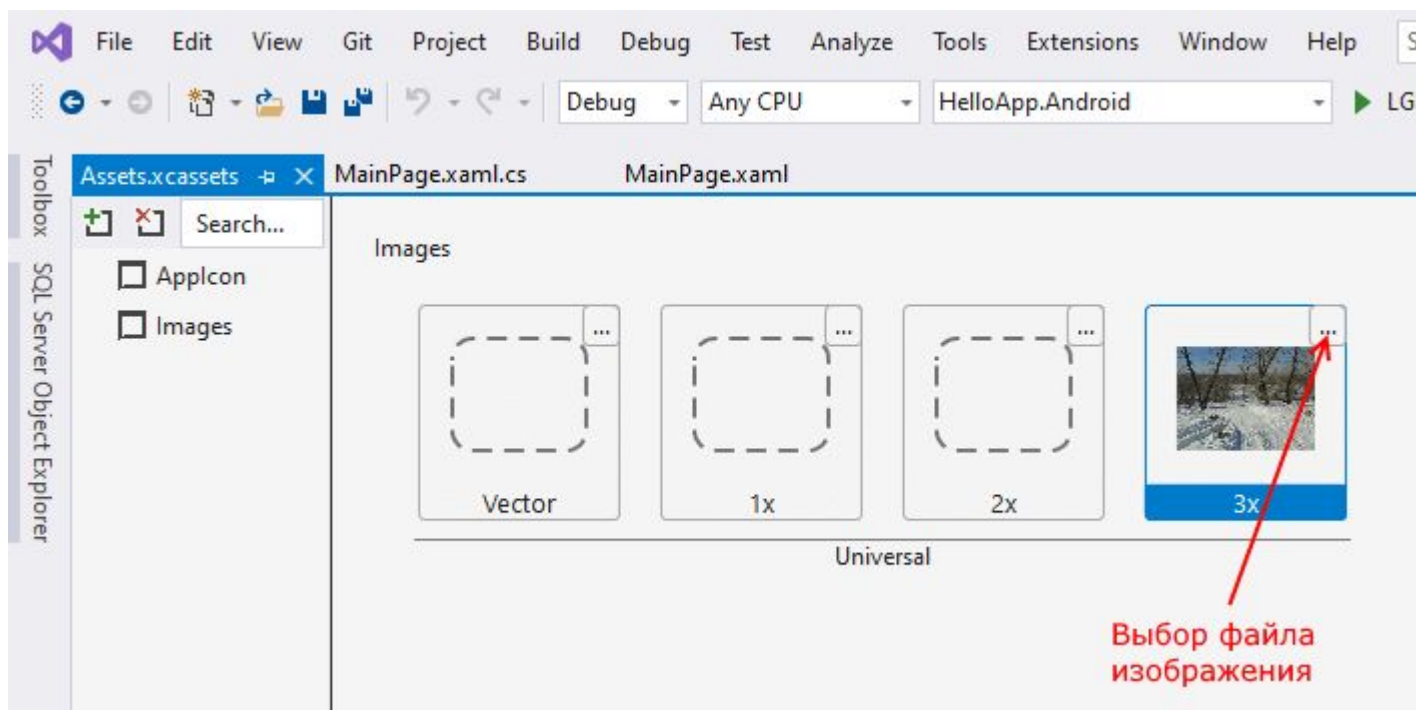


Добавление для iOS

В проекте для iOS перейти к папке **Assets Catalogs**. По умолчанию в ней должен быть каталог **Assets**. Нажать на него.



Откроется окно с прямоугольными областями для эскизов изображений. Нажать в одной из прямоугольных областей на кнопку в правом верхнем углу и открывшемся после этого диалоговом окне выбрать нужный файл изображения:



Доступ к изображению

простейший код вывода изображения в элемент Image

C#:

```
Image image = new Image { Source =  
"dubi.png" };  
this.Content = image;
```

XAML:

```
<Image Source="dubi.png" />
```



Размеры изображений

При использовании изображений надо учитывать размеры устройств.

На разных платформах есть свои принципы создания изображений для устройств с разной шириной экрана.

В Android применяется следующий подход: здесь файл надо положить в одну из подпапок в каталоге **Resources**:

drawable-hdpi: для устройств с расширением 240 DPI

drawable-xhdpi: для устройств с расширением 320 DPI

drawable-xxhdpi: для устройств с расширением 480 DPI

drawable-xxxhdpi: для устройств с расширением 640 DPI

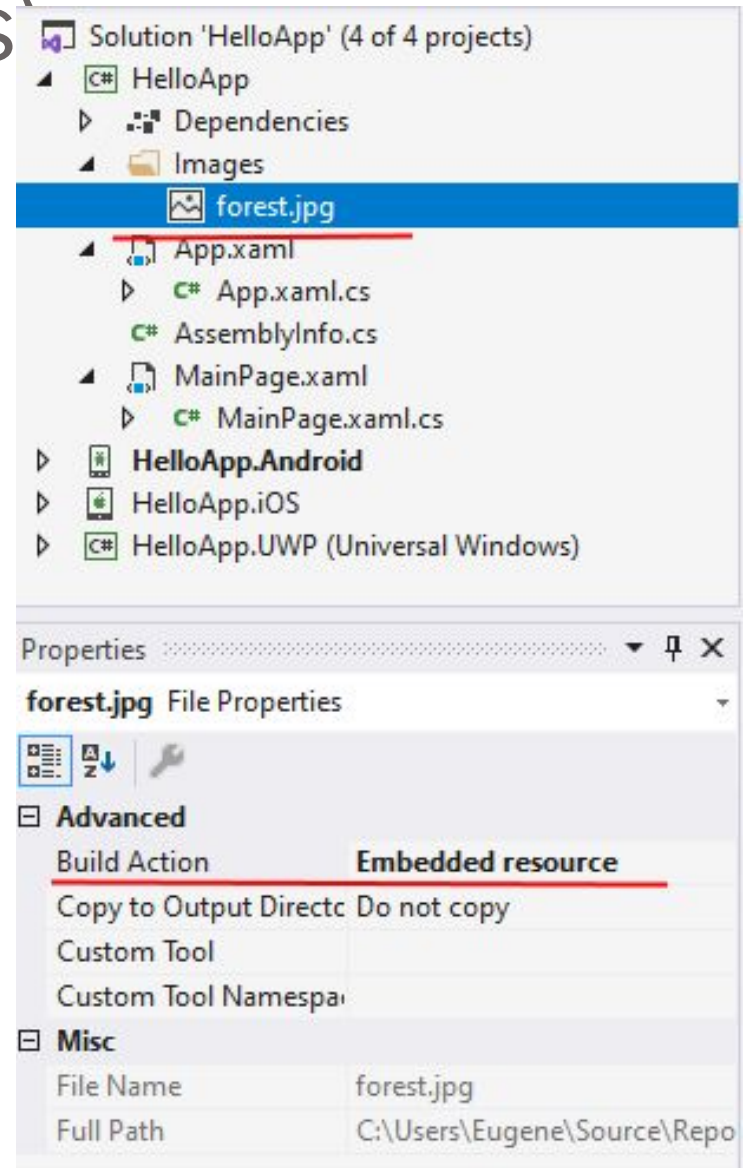
К примеру, если у нас есть файл "wild.png", то для разных размеров можно определить несколько версий файла и положить их в соответствующую папку/

Встроенные изображения (embedded images)

В отличие от выше рассмотренных локальных изображений добавляются непосредственно в разделяемую сборку в качестве ресурса.

То есть надо добавить файл только в главный проект.

Например, создать в главном проекте новый каталог `Images` и добавить в него файл изображения:



После добавления изображения в панели свойств для поля **Build Action** установить значение **Embedded Resources**.

Вывод изображения в коде C#:

```
Image image = new Image();  
    image.Source = ImageSource.FromResource  
("HelloApp.Images.forest.jpg");  
    Content = image;
```

Для получения ресурса изображения применяется метод `ImageSource.FromResource()`. Обратите внимание на путь, который в него передается. Этот путь начинается с названия проекта, то есть `HelloApp`. Дальше идет путь к изображению внутри проекта. Название проекта и название папок в этом пути отделяются точками.

Для XAML по умолчанию подобная возможность отсутствует, и надо вручную писать специальное расширение для XAML, которое позволит транслировать строковый путь в нужный нам объект. Для этого добавить в проект следующий класс:

```
[ContentProperty("Source")]
public class ImageResourceExtension : IMarkupExtension
{
    public string Source { get; set; }

    public object ProvideValue(IServiceProvider
serviceProvider)
    {
        if (Source == null)
        {
            return null;
        }
        var imageSource = ImageSource.FromResource(Source);

        return imageSource;
    }
}
```

Применить класс для загрузки изображения в коде хамл:

```
<Image Source = "{local:ImageResource  
HelloApp.Images.forest.jpg}" />  
</ContentPane>
```

Загрузка из сети

Xamarin также поддерживает загрузку из сети:

```
Image image = new Image();  
image.Source = new UriImageSource  
{  
    CachingEnabled = false,  
    Uri = new  
System.Uri("http://www.someserver/someimage.png  
")  
};
```

Кроме `url` изображения также можно задать параметры кэширования. Выражение `CachingEnabled = false` отключает кэширование. Но мы также можем включить его и установить период кэширования:

```
Image image = new Image();
image.Source = new UriImageSource
{
    CachingEnabled = true,
    CacheValidity = new
System.TimeSpan(2,0,0,0),
    Uri = new
System.Uri("http://www.someserver/someimage
.png")
};
```

Параметр `CacheValidity` указывает, сколько будет действовать кэширование - в данном случае 2 дня. Без установки параметра по умолчанию кэширование длится 24 часа.

Свойство Aspect

Позволяет задать принцип масштабирования изображения при его выводе на экран. Это свойство в качестве значения принимает одну из констант из одноименного перечисления **Aspect**:

AspectFit: значение по умолчанию. Если стандартное изображение не вписывается в экран (например, его ширина больше ширины экрана), то оно масштабируется с сохранением аспектного отношения (отношение ширины к длине)

Fill: растягивает изображение по ширине или длине без сохранения аспектного отношения

AspectFill: сохраняет аспектное отношение, но вырезает из него ту часть, которая вписывается в экран

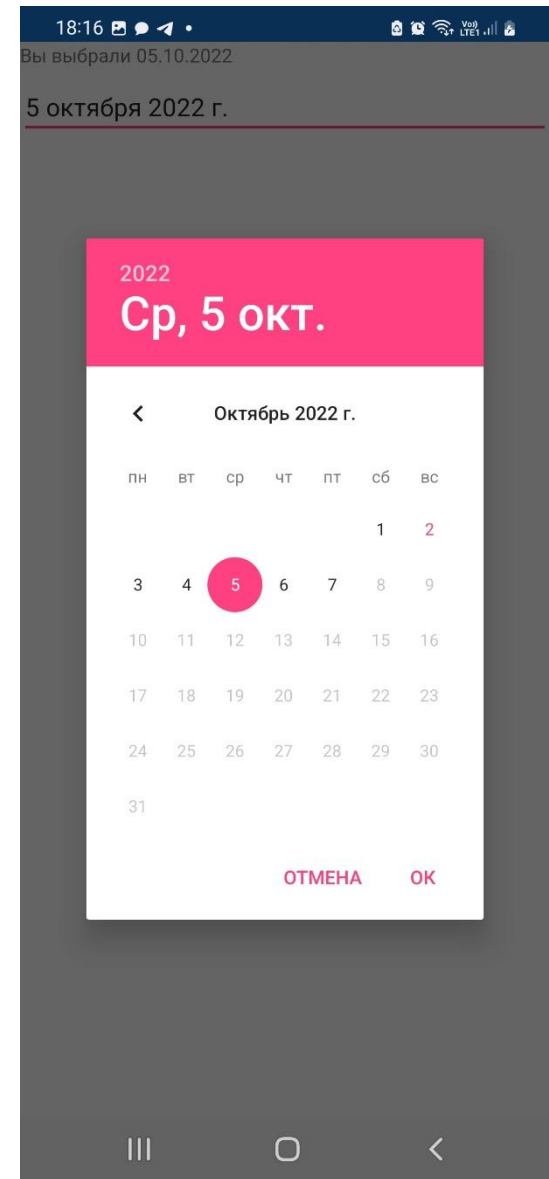
Выбор даты и времени. DatePicker и TimePicker

- **DatePicker** предназначен для выбора дат. Для его управления могут использоваться следующие свойства:
 - **MaximumDate**: максимальная возможная дата, по умолчанию равна 31 декабря 2100
 - **MinimumDate**: минимальная возможная дата, по умолчанию равна 1 января 1900
 - **Date**: выбранная дата, по умолчанию равна значению `DateTime.Today`
 - **Format**: определяет формат даты

```
Label label;
DatePicker datePicker;
public MainPage()
{
    label = new Label { Text = "Выберите дату" };
    datePicker = new DatePicker
    {
        Format = "D",
        MaximumDate = DateTime.Now.AddDays(5),
        MinimumDate = DateTime.Now.AddDays(-5)
    };
    datePicker.DateSelected += datePicker_DateSelected;
    StackLayout stack = new StackLayout { Children = { label,
datePicker } };
    this.Content = stack;
}

private void datePicker_DateSelected(object sender,
DateChangedEventArgs e)
{
    label.Text = "Вы выбрали " +
e.NewDate.ToString("dd/MM/yyyy");
}
```

Но при создании элемента надо учитывать, что в конечном счете Xamarin использует стандартные механизмы для отображения этого элемента на каждой конкретной мобильной платформе.



При установке дат в XAML необходимо передавать значение в формате, которое бы будет понятно для `DateTime.Parse()`. Наиболее простой способ заключается в использовании краткого формата даты "MM/dd/yyyy". Так, аналогичный пример в `xaml` выглядел бы следующим образом:

```
<Label x:Name="label" Text="Выберите дату" FontSize="Medium" />
```

```
    <DatePicker Format="D"
    DateSelected="datePicker_DateSelected">
```

```
    <DatePicker.MinimumDate>26/09/2022</DatePicker.Minimu
    mDate>
```

```
        <DatePicker.MaximumDate>6/10/2022
    </DatePicker.MaximumDate>
```

```
    </DatePicker>
```

И в этом случае нам надо задать обработчик события выбора даты с файле кода:

```
private void datePicker_DateSelected(object sender,
    DateChangedEventArgs e)
```

```
{
```

TimePicker

TimePicker представляет элемент управления для отображения времени:

```
Label label;
    TimePicker timePicker;
public MainPage()
{
    //InitializeComponent();
    label = new Label { Text = "Выберите время" };
    timePicker = new TimePicker() { Time = new
System.TimeSpan(17, 0, 0)};
    timePicker.PropertyChanged +=
TimePicker_PropertyChanged;

    StackLayout stack = new StackLayout { Children = {
label, timePicker } };
    this.Content = stack;
}
```

Если необходимо отследить выбор времени в `TimePicker`, то можно подписаться на событие `PropertyChanged`.

```
private void  
TimePicker_PropertyChanged(object  
sender,  
System.ComponentModel.PropertyChanged  
EventArgs e)  
{  
    if (args.PropertyName == "Time")  
    {  
        label.Text = "Вы выбрали " +  
timePicker.Time;  
    }  
}
```

Конкретное отображение TimePickera также будет зависеть от конкретной платформы. Например, на Android элемент будет выглядеть так:



Создание TimePicker в xaml:

```
<Label x:Name="label" Text="Выберите дату"  
FontSize="Medium" />  
    <TimePicker x:Name="timePicker" Time="17:00:00"  
PropertyChanged="TimePicker_PropertyChanged"></Time  
ePicker>  
    </StackLayout>  
</ContentPage>
```

Выпадающий список Picker

Визуально Picker представляет собой обычное текстовое поле, по нажатию на которое открывается список для выбора, что-то наподобие выпадающего списка:

Picker содержит список для выбора, а отследить выбранный элемент можно с помощью обработчика события `SelectedIndexChanged`.



```
Label header;
Picker picker;
public MainPage()
{
    header = new Label
    {
        Text = "Выберите язык",
        FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label))
    };

    picker = new Picker
    {
        Title = "Язык"
    };
    picker.Items.Add("C#");
    picker.Items.Add("JavaScript");
    picker.Items.Add("Java");
    picker.Items.Add("PHP");
    picker.SelectedIndexChanged += picker_SelectedIndexChanged;
    this.Content = new StackLayout { Children = { header, picker } };
}
void picker_SelectedIndexChanged(object sender, EventArgs e)
{
    header.Text = "Вы выбрали: " + picker.Items[picker.SelectedIndex];
}
```

В XAML

```
<Label x:Name="header" Text="Языки программирования"
FontSize="Large" />
  <Picker x:Name="picker"
SelectedIndexChanged="picker_SelectedIndexChanged">
  <Picker.Items>
    <x:String>C#</x:String>
    <x:String>C/C++</x:String>
    <x:String>JavaScript</x:String>
    <x:String>PHP</x:String>
  </Picker.Items>
</Picker>
```

```
void picker_SelectedIndexChanged(object sender, EventArgs
e)
{
  header.Text = "Вы выбрали: " +
picker.Items[picker.SelectedIndex];
}
```

Stepper

Stepper позволяет устанавливать числовое значение:

Label header;

```
public MainPage()
{
    header = new Label
    {
        Text = "Stepper",
        FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
        HorizontalOptions = LayoutOptions.Center
    };

    Stepper stepper = new Stepper
    {
        Minimum = 0,
        Maximum = 10,
        Increment = 0.1,
        HorizontalOptions = LayoutOptions.Center,
        VerticalOptions = LayoutOptions.CenterAndExpand
    };
    stepper.ValueChanged += OnStepperValueChanged;
    this.Content = new StackLayout { Children = { header, stepper } };
}
```

```
private void OnStepperValueChanged(object sender,  
ValueChangedEventArgs e)  
{  
    header.Text = String.Format("Выбрано: {0:F1}",  
e.NewValue);  
}
```

Stepper позволяет установить ряд свойств, который настраивают его поведение:

Minimum: устанавливает минимальное значение

Maximum: устанавливает максимальное значение

Increment: устанавливает шаг изменения значения

Для отслеживания изменения значения можно обработать событие **ValueChanged**



Stepper в xaml:

```
<Label x:Name="header" Text="Stepper"
FontSize="Large" />
  <Stepper Minimum="0" Maximum="10"
Increment="0.1"
ValueChanged="OnStepperValueChanged" />
</StackLayout>
```

А в файле связанного кода прописать обработчик
OnStepperValueChanged:

```
private void OnStepperValueChanged(object sender,
ValueChangedEventArgs e)
{
    if (header != null)
        header.Text = String.Format("Выбрано: {0:F1}",
e.NewValue);
}
```


Slider

Slider представляет собой горизонтальный ползунок и во многих аспектах он похож на **Stepper**. Среди его свойств можно выделить следующие:

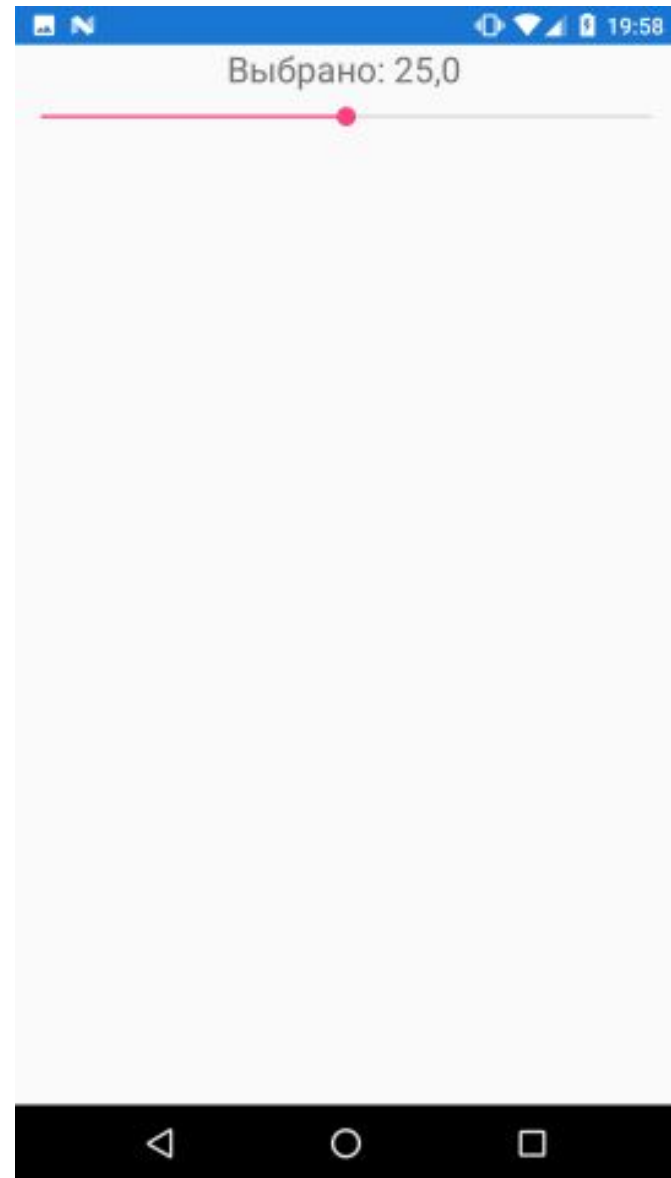
- **Minimum**: устанавливает минимальное значение
- **Maximum**: устанавливает максимальное значение
- **Value**: текущее значение
- **ThumbColor**: цвет указателя текущего значения
- **MinimumTrackColor**: цвет ползунка до указателя значения
- **MaximumTrackColor**: цвет ползунка после указателя значения

определение в коде C#

```
Label header;  
    public MainPage()  
    {  
        header = new Label  
        {  
            Text = "Slider",  
            FontSize = Device.GetNamedSize(NamedSize.Large,  
typeof(Label)),  
            HorizontalOptions = LayoutOptions.Center  
        };  
        Slider slider = new Slider  
        {  
            Minimum = 0,  
            Maximum = 50,  
            Value = 30  
            ThumbColor=Color.DeepPink,  
            MinimumTrackColor=Color.DeepPink,  
            MaximumTrackColor=Color.Gray  
        };  
        slider.ValueChanged += slider_ValueChanged;  
        this.Content = new StackLayout { Children = { header, slider } };  
    }  
}
```

```
void slider_ValueChanged(object sender,  
ValueChangedEventArgs e)  
    {  
        header.Text = String.Format("Выбрано: {0:F1}",  
e.NewValue);  
    }
```

Slider также позволяет установить минимальное и максимальное, а также текущее значение. И также изменение значения можно обработать с помощью обработки события `ValueChanged`



Аналогичный слайдер в xaml:

```
<Label x:Name="header" Text="Slider" FontSize="Large" />  
    <Slider Minimum="0" Maximum="50" Value="30"  
ValueChanged="slider_ValueChanged"  
        MinimumTrackColor="DeepPink"  
MaximumTrackColor="Gray" ThumbColor="DeepPink" />  
</StackLayout>
```

И в файле связанного кода пропишем обработчик `slider_ValueChanged`:

```
void slider_ValueChanged(object sender,  
ValueChangedEventArgs e)  
{  
    if(header!=null)  
        header.Text = String.Format("Выбрано: {0:F1}",  
e.NewValue);  
}
```

Переключатель **Switch**

Элемент **Switch** представляет переключатель, который может находиться в двух состояниях: включенном и выключенном.

Среди свойств класса **Switch** стоит выделить следующие:

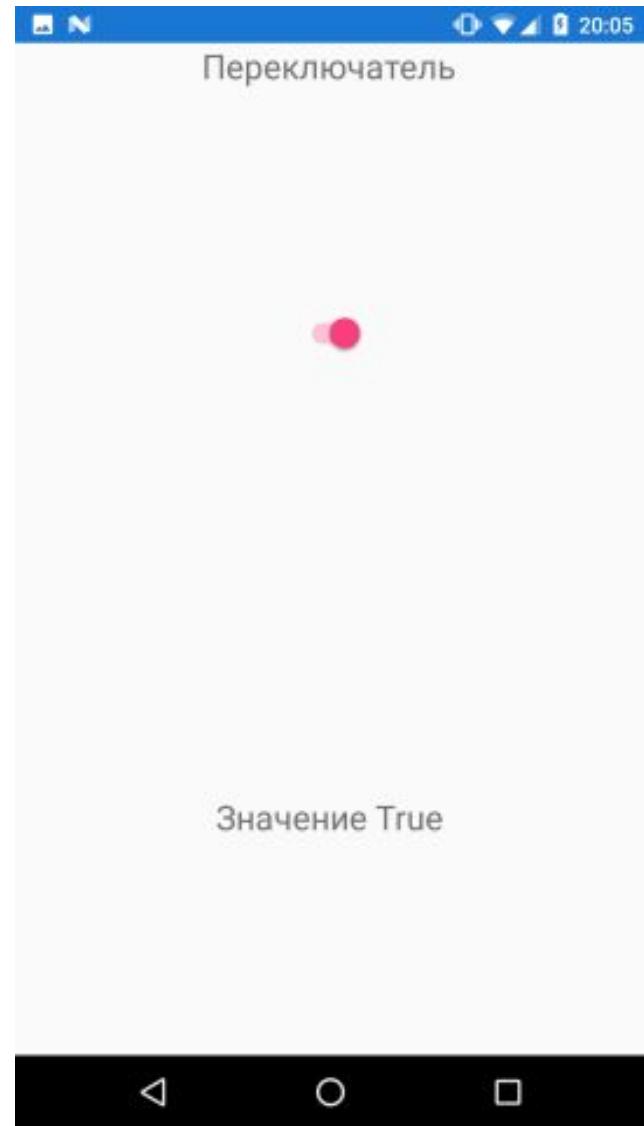
- **IsToggled**: указывает, находится ли **Switch** во включенном состоянии (значение `true`) или выключенном (значение `false`)
- **ThumbColor**: цвет кнопки переключателя
- **OnColor**: цвет переключателя во включенном состоянии

В коде C#:

```
Label label;
public MainPage()
{
    Label header = new Label
    {
        Text = "Переключатель",
        FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
        HorizontalOptions = LayoutOptions.Center
    };
    Switch switcher = new Switch
    {
        IsToggled = true,
        HorizontalOptions = LayoutOptions.Center,
        VerticalOptions = LayoutOptions.CenterAndExpand
    };
    switcher.Toggled += switcher_Toggled;
    label = new Label
    {
        FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
        HorizontalOptions = LayoutOptions.Center,
        VerticalOptions = LayoutOptions.CenterAndExpand
    };
    this.Content = new StackLayout { Children = { header, switcher, label } };
}
```

```
private void switcher_Toggled(object sender,  
ToggledEventArgs e)  
{  
    label.Text = $"Значение {e.Value}";  
}
```


Если надо установить переключатель в определенное состояние, то применяется свойство `IsToggled`. По умолчанию оно имеет значение `false`. Чтобы отследить смену состояния, мы можем обработать событие `Toggled`



Аналог в xaml:

```
<Label Text="Переключатель" FontSize="Large"  
HorizontalOptions="Center" />
```

```
  <Switch IsToggled="true"  
VerticalOptions="CenterAndExpand"  
  HorizontalOptions="Center" Toggled=  
"switcher_Toggled" />
```

```
  <Label x:Name="label" FontSize="Large"  
    HorizontalOptions = "Center" VerticalOptions =  
"CenterAndExpand" />
```

- <https://learn.microsoft.com/ru-ru/xamarin/get-started/tutorials/local-database/?tabs=vswin&tutorial-step=2>