Working with layouts

Layouts allow you to arrange and group controls within pages in a software application's user interface. Choosing a layout when designing a user interface requires the designer to know how controls should be arranged within the pages displayed during the software application's runtime.

This lesson will cover the following layouts: 1. Grid Layout

2. Frame Layout
3. StackLayout
4. ContentView Layout
5. ScrollView Layout
6. AbsolutLayout
7. RelativeLayout
8. FlexLayout

Let's consider the features of working with a Grid layout, which allows you to place controls in the cells of a table created inside the layout. Thus, each cell can contain several controls or layouts at once.

The Grid type layout has the following properties:

1. The Column property defines the column number in the table. Column numbering starts with 0. The default value is 0.
2. The ColumnDefinitions property defines the properties for the newly created column in the table.
3. The ColumnSpacing property determines the distance between columns.
tables. The default value of this property is 6 units.
4. The ColumnSpan property determines the number of columns to span.
within one line. The default value of this property is 1.
5. The Row property defines the row number in the table. Row numbering starts with 0. The default value of the property is 0.
6. The RowDefinitions property defines the properties for the generated new row in the table.

7. The RowSpacing property determines the distance between rows.
tables. The default value of this property is 6 units.
8. The RowSpan property determines the number of rows to be combined into within a single column. The default value of this property is 1.

To examine the features of working with the Grid layout, a GridDemo project is created for a multiplatform software application using the "Empty" template. You need to go to the "Solution Explorer" window, select and open the MainPage.xaml file and type the XAML program code below. The program code provides for the formation of a Grid layout, in which all three rows, each of which has 2 columns, contain the BoxView

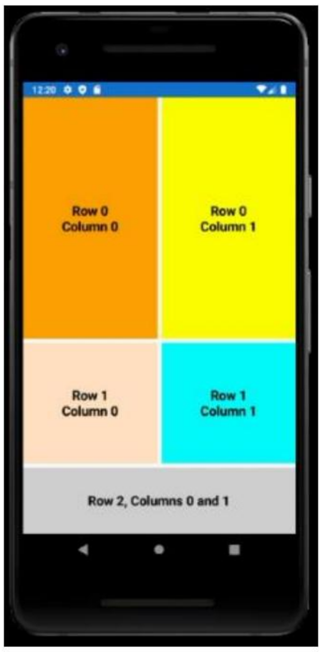
and Label controls. Each column has

equal width. The third row combines two columns. The BoxView control in the third row spans both columns. Multiple controls can be placed in a single cell.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://
        schemas.microsoft.com/winfx/2009/xaml" x:Class="GridDemo.MainPage">
        <Grid>

      <Grid.RowDefinitions>
         <RowDefinition Height="2*" />
         <RowDefinition />
         <RowDefinition Height="100" /> </
      Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
         <ColumnDefinition />
         <ColumnDefinition /> </
      Grid.ColumnDefinitions> <BoxView
      Color="Orange" /> <Label Text="Row
      0 Column 0"

            HorizontalTextAlignment="Center"
            FontSize="20" TextColor="Black"
            FontAttributes="Bold" HorizontalOptions="Center"
            VerticalOptions="Center" />
      <BoxView Grid.Column="1" Color="Yellow" /> <Label
      Grid.Column="1"
            Text="Row 0
                Column 1"
            HorizontalTextAlignment="Center"
            FontSize="20" TextColor="Black"
            FontAttributes="Bold"
            HorizontalOptions="Center"
            VerticalOptions="Center" />
      <BoxView Grid.Row="1" Color="Bisque" /> <Label
      Grid.Row="1"
            Text="Row 1
                    Column 0"
            HorizontalTextAlignment="Center"
            FontSize="20" TextColor="Black"
            FontAttributes="Bold"
            HorizontalOptions="Center"
            VerticalOptions="Center" />
      <BoxView Grid.Row="1"
             Grid.Column="1" Color="Aqua" /> <Label
      Grid.Row="1"
            Grid.Column="1"
```

```
                        Text="Row 1
                       Column 1"
                       HorizontalTextAlignment="Center"
                       FontSize="20" TextColor="Black"
                       FontAttributes="Bold"
                       HorizontalOptions="Center"
                       VerticalOptions="Center" />
                <BoxView Grid.Row="2" Grid.ColumnSpan="2"
                        Color="LightGray" />
                <Label Grid.Row="2" Grid.ColumnSpan="2"
                       Text="Row 2, Columns 0 and 1"
                        HorizontalTextAlignment="Center"
                       FontSize="20" TextColor="Black"
                       FontAttributes="Bold"
                       HorizontalOptions="Center"
                       VerticalOptions="Center" />
            </Grid>
        </ContentPage>
```

To demonstrate the Grid layout, we first build and then run a software application to demonstrate the use of the Grid layout. The page displays BoxView and Label controls located in cells of a table formed using the Grid layout.



Next, we will move on to studying the features of working with Frame and StackLayout layouts. Frame layouts are used to create borders around a control that is inside the layout. The Frame layout has the following properties:

1. The BorderColor property defines the border color of a Frame layout.
2. The CornerRadius property defines the radius of the layout corner rounding.

3. The HasShadow property determines whether there is a shadow around the layout.

The StackLayout layout allows you to place controls or layouts within it vertically or horizontally next to each other, depending on the value of the Orientation property. The Spacing property controls the spacing between controls within the layout and has a default value of 6.

To study the features of working with Frame and StackLayout layouts, the StackLayoutDemo project is created for a multiplatform software application using the Empty template. You need to go to the Solution Explorer window, select and open the MainPage.xaml file, and then type the program code below. The program code describes the creation of a StackLayout layout, in which three Label controls and six Frame layouts are located vertically one under another.

Each Frame layout covers a StackLayout layout contained within it, inside which a BoxView control for displaying color and a Label control, which displays the color names in the BoxView control, are located horizontally. The page background color, font color, size, writing style, and text arrangement within the Label controls, as well as the arrangement of controls within the layout, are configured. The BorderColor property of the Frame layout is also used to configure the thickness of the outlines surrounding the layouts.

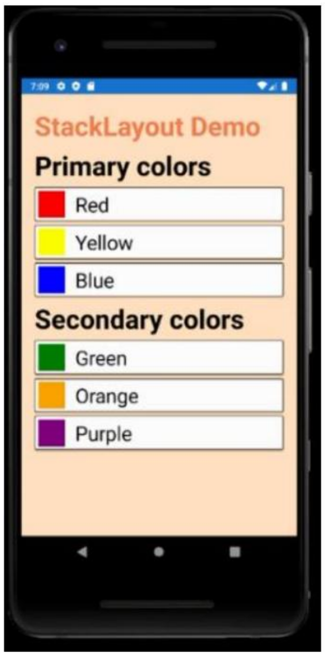
```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
          x:Class="StackLayoutDemo.MainPage"
          BackgroundColor="Bisque"
          Title="StackLayout Demo">

  <StackLayout Margin="20">
    <Label Text="StackLayout Demo"
        FontSize="40"
        FontAttributes="Bold"
        TextColor="Coral"/>
    <Label Text="Primary colors"
        FontSize="40"
        FontAttributes="Bold"
        TextColor="Black"/>
    <Frame BorderColor="Black"
        Padding="5">
      <StackLayout Orientation="Horizontal"
            Spacing="15">
        <BoxView Color="Red" />
        <Label Text="Red"
            FontSize="30"
            TextColor="Black"
```

```
               VerticalOptions="Center" />
     </StackLayout>
</Frame>
<Frame BorderColor="Black"
       Padding="5">
     <StackLayout Orientation="Horizontal"
                  Spacing="15">
          <BoxView Color="Yellow" />
          <Label Text="Yellow"
               FontSize="30"
               TextColor="Black"
               VerticalOptions="Center" />
     </StackLayout>
</Frame>
<Frame BorderColor="Black"
       Padding="5">
     <StackLayout Orientation="Horizontal"
                  Spacing="15">
          <BoxView Color="Blue" />
          <Label Text="Blue"
               FontSize="30"
               TextColor="Black"
               VerticalOptions="Center" />
     </StackLayout>
</Frame>
<Label Text="Secondary colors"
       FontSize="40"
       FontAttributes="Bold"
       TextColor="Black"/>
<Frame BorderColor="Black"
       Padding="5">
     <StackLayout Orientation="Horizontal"
                  Spacing="15">
          <BoxView Color="Green" />
          <Label Text="Green"
               FontSize="30"
               TextColor="Black"
               VerticalOptions="Center" />
     </StackLayout>
</Frame>
<Frame BorderColor="Black"
       Padding="5">
     <StackLayout Orientation="Horizontal"
                  Spacing="15">
          <BoxView Color="Orange" />
```

```
            <Label Text="Orange"
                  FontSize="30"
                  TextColor="Black"
                  VerticalOptions="Center" />
          </StackLayout>
        </Frame>
        <Frame BorderColor="Black"
              Padding="5">
          <StackLayout Orientation="Horizontal"
                      Spacing="15">
            <BoxView Color="Purple" />
            <Label Text="Purple"
                  FontSize="30"
                  TextColor="Black"
                  VerticalOptions="Center" />
          </StackLayout>
        </Frame>
      </StackLayout>
    </ContentPage>
```

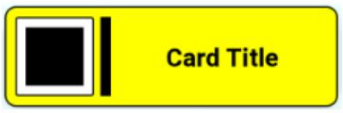
To demonstrate working with Frame and StackLayout layouts We need to first compile and then run the software application.

The user interface displays three Label controls and six Frame controls inside a StackLayout, each defining a line around the outline of the StackLayout inside the Frame. Each StackLayout inside the Frame contains a BoxView and a Label control, stacked horizontally.



Next, let's look at the specifics of working with a ContentView layout, which contains one child element and is typically used to create templates for custom and reusable controls.

To work with a ContentView type layout, let's create a project ContViewDemo for a Xamarin.Forms multiplatform software application using the "Empty" template. Next, using the previously described sequence of actions, a "Content Page" object named Card is created. Next, the created Card.xaml file is opened and the following XAML program code is entered. Using the XAML program code, a template of a custom control in the form of a card is created, which has the appearance of a rectangle with rounded corners and in which the color is displayed on the left side of the rectangle, and the text is placed on the right side.



In the code, the ContentView layout contains a custom control template and sets the x:Name attribute value to this. This value can be used to access the control that will be created based on the template using Binding.

The control template, called Card, declares the following properties:

The CardTitle property specifies the text displayed in the control;
The CBColor property specifies the border color of the template element.
controls and color of the separator line;
The TarColor property specifies the color of the square displayed by the BoxView control on the left side of the template control;

The CardColor property specifies the background color of the template control;
The ColText property specifies the color of the text in the template control.
The ContentView layout contains a Frame layout, which is a rectangle with rounded edges and a background color specified by the property
CardColor, and the border color specified by the CBColor property. In addition, the indents from the layout borders are set, and the value of the flag for the presence of a shadow around the layout is set. Inside the Frame layout is a Grid layout. Inside the Grid layout is a table with one row and three columns. The first column contains the Frame control with the BoxView control. By setting the Margin value, you can achieve the presence of a border around the BoxView control. The color that the BoxView control is filled with is set by the TarColor property. The entire second column is occupied by the BoxView control, which acts as a separator between the columns. The color of this control is set using

CBColor properties. The third cell contains a Label control, which displays the text that is set using the CardTitle property. The text color in the Label control is set by the ColText property. Binding allows you to bind the properties of Frame layouts and the BoxView and Label controls to the properties

CardTitle, CBColor, TarColor, CardColor, ColText.

```
<ContentView xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:d="http://xamarin.com/schemas/2014/forms/design"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
                                        compatibility/2006"
         x:Name="this"
         x:Class="ContViewDemo.Card">
   <Frame BindingContext="{x:Reference this}"
          BackgroundColor="{Binding CardColor,
                                 FallbackValue='Yellow'}"
          BorderColor="{Binding CBColor,
                              FallbackValue='Black'}"
        CornerRadius="10"
        HasShadow="True"
        Padding="10"
        VerticalOptions="Center"
        HorizontalOptions="Center">
      <Grid>
        <Grid.RowDefinitions>
          <RowDefinitionHeight="75" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="75" />
          <ColumnDefinition Width ="10" />
          <ColumnDefinition Width="200" />
        </Grid.ColumnDefinitions>
        <Frame IsClippedToBounds="True"
            BorderColor="{Binding CBColor,
                               FallbackValue='Black'}"
            HeightRequest="70"
            WidthRequest="70"
            HasShadow="False"
            HorizontalOptions="Center"
            VerticalOptions="Center">
          <BoxView BackgroundColor ="{Binding TarColor,
                                         FallbackValue='Black'}"
              Margin="-10"
              WidthRequest="50"
              HeightRequest="50"/>
            </Frame>
            <BoxView Grid.Column="1"
                   BackgroundColor="{Binding CBColor,
                                             FallbackValue='Black'}"
                   WidthRequest="10"
                   VerticalOptions="Fill" />
            <Label Grid.Column="2"
                  Text="{Binding CardTitle,
                                      FallbackValue='Card Title'}"
                  TextColor="{Binding ColText,
                                           FallbackValue='Red'}"
                  FontAttributes="Bold"
                  FontSize="24"
                  VerticalTextAlignment="Center"
                  HorizontalTextAlignment="Center" />
        </Grid>
    </Frame>
</ContentView>
```

Next, you need to open the Card.xaml.cs file and type the C Sharp program code given below. The program code creates the CardTitle, CBColor, TarColor, CardColor, ColText properties for the Card template control. Using BindableProperty, you can bind the properties of the control created on the basis of the Card template to the properties of this template.

To get and set the values of bindable properties for a control created based on the Card template, use the GetValue and SetValue methods.

```
using System.ComponentModel;
using Xamarin.Forms;
namespaceContViewDemo
{
    [DesignTimeVisible(true)]
    public partial class Card: ContentView
    {
        public static readonly BindableProperty
            CardTitleProperty = BindableProperty.Create(
                        nameof(CardTitle),
                        typeof(string),
                        typeof(Card));
        public static readonly BindableProperty
            CBColorProperty = BindableProperty.Create(
                         nameof(CBColor),
                         typeof(Color),
                         typeof(Card),
                    Color.Black);
public static readonly BindableProperty
    TarColorProperty =




BindableProperty.Create( nameof(TarColor), typeof(Color), typeof(Card), C
    CardColorProperty =




BindableProperty.Create( nameof(CardColor), typeof(Color), typeof(Card)
    ColTextProperty =





BindableProperty.Create( nameof(ColText), typeof(Color), typeof(Card), C
    get => (string)GetValue(Card.CardTitleProperty);
    set => SetValue(Card.CardTitleProperty, value);
}

public Color CBColor
{
    get => (Color)GetValue(Card.CBColorProperty);
    set => SetValue(Card.CBColorProperty, value);

} public Color TarColor
{
    get => (Color)GetValue(Card.TarColorProperty);
    set => SetValue(Card.TarColorProperty, value);

} public Color CardColor
{
    get => (Color)GetValue(Card.CardColorProperty);
    set => SetValue(Card.CardColorProperty, value);

} public ColorColText
{
    get => (Color)GetValue(Card.ColTextProperty);
    set => SetValue(Card.ColTextProperty, value);
}
```

```
        public Card()
        {
           InitializeComponent();
        }
     }
  }
```

Next, select the MainPage.xaml file in the Solution Explorer dialog box, open it, and type the XAML code below. The code for the ContentPage type page includes a reference

to controls - the namespace of the user control, that is, to the control template, the code for which is provided in the Card.xaml file.

The MainPage page contains a StackLayout layout, inside which there is a Label control for displaying the page title, as well as three custom controls that are instances of the Card template.

Each time the template is accessed, the program code passes the values of the bound properties CardTitle, CBColor, TarColor, CardColor, ColText to the template. These properties were bound to the template earlier in the program code provided in the Card.xaml and Card.xaml.cs files.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:d="http://xamarin.com/schemas/2014/forms/design"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
                                                          compatibility/2006"
         xmlns:controls="clr-namespace:ContViewDemo"
         x:Class="ContViewDemo.MainPage"
         Padding="10, 50"
         BackgroundColor="Azure">
    <StackLayout>
      <Label
           Margin="20"
           Text="Content View Demo"
           FontSize="30" FontAttributes="Bold"
           TextColor="Black"
           HorizontalOptions="Center"
           HorizontalTextAlignment="Center"
           VerticalTextAlignment="Center"/>

        <controls:Card CardTitle="Red"
                   CBColor="Black"
                   TarColor="Red"
```

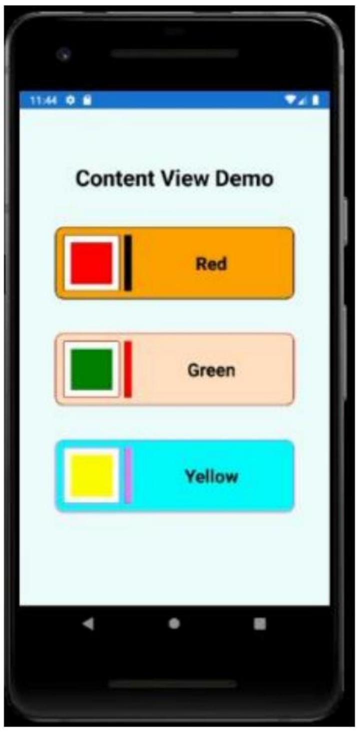
```
                        CardColor="Orange"
                        ColText="Black"/>
                <controls:Card CardTitle="Green"
                        CBColor="Red"
                        TarColor="Green"
                        CardColor="Bisque"
                        ColText="Black"/>
                <controls:Card CardTitle="Yellow"
                        CBColor="Violet"
                        TarColor="Yellow"
                        CardColor="Aqua"
                        ColText="Black"/>
            </StackLayout>
        </ContentPage>
```

To demonstrate working with the ContentView layout, you must first compile and then run the software application.

Upon startup, the UI displays three controls based on the Card template implemented using the ContentView layout. As the UI is built, the code accesses the control template three times with

new bindable property values.

As a result, three images are displayed on the mobile device screen. different instances of a template control.



Next, we will consider the features of working with a ScrollView layout, which can be used to scroll controls located inside this layout. Very often, inside a layout of the type

ScrollView is placed in a StackLayout type layout. However, it is impossible to display all the controls located inside the StackLayout type layout on the device screen. To specify the scrolling direction (horizontal, vertical or both), you must set the corresponding value of the Orientation property of the ScrollView type layout.

To consider the features of working with the ScrollView layout, the ScrViewDemo project is created for a multiplatform software application using the Empty template. You need to go to the Solution Explorer dialog box, select and open the MainPage.xaml file and type the program code below in it. In the program code, the ScrollView layout contains the StackLayout layout. The StackLayout layout contains four Frame layouts, which create the borders of the Grid layout located inside it. Each Frame layout has a background color, border color, corner radius, and shadow around the layout. The Grid layout has three rows and two columns. The first column of the first row contains a round two-color emblem formed using the Frame layout and the BoxView control. The second column of the first row contains the Label control with the displayed text. The entire second row, in which the columns are combined, is occupied by the BoxView control, which acts as a separator for the first and third rows. The third row, which combines two columns, contains a Label control that also displays text.

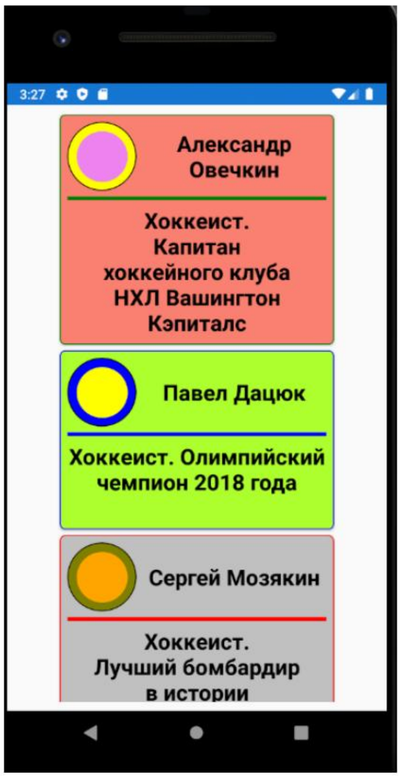
```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:d="http://xamarin.com/schemas/2014/forms/design"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
                                                    compatibility/2006"
         xmlns:controls="clr-namespace: ScrViewDemo "
         x:Class="ScrViewDemo.MainPage"
         Title="ScrollView Demo"
         Padding="10">
  <ScrollView>
    <StackLayout>
      <Frame
        BackgroundColor="Salmon"
        BorderColor="Green" CornerRadius="5"
        HasShadow="True" Padding="8"
        VerticalOptions="Center"
        HorizontalOptions="Center">
        <Grid>
          <Grid.RowDefinitions>
            <RowDefinitionHeight="75" />
            <RowDefinitionHeight="4" />
            <RowDefinition Height="Auto" />
          </Grid.RowDefinitions>
```

```
            <Grid.ColumnDefinitions>
               <ColumnDefinition Width="75" />
               <ColumnDefinition Width="200" />
            </Grid.ColumnDefinitions>
            <Frame BorderColor="Black"
                    BackgroundColor="Yellow"
                    CornerRadius="38" HeightRequest="60"
                    WidthRequest="60"
                    HorizontalOptions="Center"
                    HasShadow="False"
                    VerticalOptions="Center">
               <BoxView
                    Margin="-10" CornerRadius="38"
                    BackgroundColor="Violet"/> </
            Frame>
            <Label Grid.Column="1"
                    Text="Alexander Ovechkin"
                    FontAttributes="Bold"
                    FontSize="24" TextColor="Black"
                    VerticalTextAlignment="Center"
                    HorizontalTextAlignment="Center" /> <BoxView
            Grid.Row="1"
                     Grid.ColumnSpan="2"
                     BackgroundColor="Green"
                     HeightRequest="2"
                     HorizontalOptions="Fill" />
            <Label Grid.Row="2" Grid.ColumnSpan="2"
                    Text="Hockey player.
                              Captain of the NHL hockey club
                              Washington Capitals"
                    FontAttributes="Bold"
                    FontSize="24" TextColor="Black"
                    HorizontalTextAlignment="Center"
                    VerticalTextAlignment="Start"
                    VerticalOptions="Fill"
                    HorizontalOptions="Fill" />
         </Grid>
      </Frame>
      <Frame
            BackgroundColor="GreenYellow"
            BorderColor="Blue" CornerRadius="5"
            HasShadow="True" Padding="8"
            VerticalOptions="Center"
            HorizontalOptions="Center">
         <Grid>
```

```
<Grid.RowDefinitions>
    <RowDefinition Height="75" />
    <RowDefinition Height="4" />
    <RowDefinition Height="Auto" /> </
Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="75" />
    <ColumnDefinition Width="200" /> </
Grid.ColumnDefinitions> <Frame
BorderColor="Black"
           BackgroundColor="Blue"
           CornerRadius="38"
           HeightRequest="60"
           WidthRequest="60"
           HorizontalOptions="Center"
           HasShadow="False"
           VerticalOptions="Center">
    <BoxView
          Margin="-10"
          CornerRadius="38"
          BackgroundColor="Yellow"/> </
Frame>
<Label Grid.Column="1"
           Text="Pavel Datsyuk"
           FontAttributes="Bold"
           FontSize="24"
           TextColor="Black"
           VerticalTextAlignment="Center"
           HorizontalTextAlignment="Center" /> <BoxView
Grid.Row="1"
             Grid.ColumnSpan="2"
             BackgroundColor="Blue"
             HeightRequest="2"
             HorizontalOptions="Fill" /> <Label
Grid.Row="2"
           Grid.ColumnSpan="2"
           Text="Hockey player. Captain of the KHL Avtomobilist
                    hockey club"
           FontAttributes="Bold"
           FontSize="24"
           TextColor="Black"
           HorizontalTextAlignment="Center"
           VerticalTextAlignment="Start"
           VerticalOptions="Fill"
           HorizontalOptions="Fill" />
    </Grid> </
Frame>
<Frame
         BackgroundColor="Silver"
         BorderColor="Red"
         CornerRadius="5"
         HasShadow="True"
         Padding="8"
         VerticalOptions="Center"
         HorizontalOptions="Center"> <Grid>

    <Grid.RowDefinitions>
        <RowDefinition Height="75" />
        <RowDefinition Height="4" />
        <RowDefinition Height="Auto" /> </
      Grid.RowDefinitions> <Grid.
      ColumnDefinitions>
        <ColumnDefinition Width="75" />
        <ColumnDefinition Width="200" /> </
      Grid.ColumnDefinitions> <Frame
      BorderColor="Black"
                BackgroundColor="Olive"
                CornerRadius="38"
                HeightRequest="60"
                WidthRequest="60"
                HorizontalOptions="Center"
                HasShadow="False"
                VerticalOptions="Center">
        <BoxView
               Margin="-10"
               CornerRadius="38"
               BackgroundColor="Orange"/> </
      Frame>
      <Label Grid.Column="1"
                Text="Sergey Mozyakin"
                FontAttributes="Bold"
                FontSize="24"
                TextColor="Black"
                VerticalTextAlignment="Center"
                HorizontalTextAlignment="Center" /> <BoxView
      Grid.Row="1"
                  Grid.ColumnSpan="2"
                  BackgroundColor="Red"
                  HeightRequest="2"
                  HorizontalOptions="Fill" />
        <Label Grid.Row="2"
                    Grid.ColumnSpan="2"
                    Text="Hockey player.
                      The best scorer in the history of
                      Russian hockey"
                    FontAttributes="Bold"
                    FontSize="24"
                    TextColor="Black"
                    HorizontalTextAlignment="Center"
                    VerticalTextAlignment="Start"
                    VerticalOptions="Fill"
                    HorizontalOptions="Fill" /> </Grid>
    </Frame>
<Frame

BackgroundColor="Aqua"
         BorderColor="Brown"
         CornerRadius="5"
         HasShadow="True"
         Padding="8"
         VerticalOptions="Center"
         HorizontalOptions="Center"> <Grid>

    <Grid.RowDefinitions>
            <RowDefinition Height="75" />
            <RowDefinition Height="4" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="75" />
            <ColumnDefinition Width="200" /> </
        Grid.ColumnDefinitions> <Frame
        BorderColor="Black"
                    BackgroundColor="Red"
                    CornerRadius="38"
                    HeightRequest="60"
                    WidthRequest="60"
                    HorizontalOptions="Center"
                    HasShadow="False"
                    VerticalOptions="Center">
            <BoxView
                   Margin="-10"
                   CornerRadius="38"
                   BackgroundColor="Gold"/>
        </Frame>
```

```
                    <Label Grid.Column="1"
                           Text="Vladislav Tretyak"
                           FontAttributes="Bold"
                           FontSize="24"
                           TextColor="Black"
                           VerticalTextAlignment="Center"
                           HorizontalTextAlignment="Center" />
                    <BoxView Grid.Row="1"
                             Grid.ColumnSpan="2"
                             BackgroundColor="Red"
                             HeightRequest="2"
                             HorizontalOptions="Fill" />
                    <Label Grid.Row="2"
                           Grid.ColumnSpan="2"
                           Text="Hockey player.
                            The best goalie in history in the
                            history of world hockey"
                           FontAttributes="Bold"
                           FontSize="24"
                           TextColor="Black"
                           HorizontalTextAlignment="Center"
                           VerticalTextAlignment="Start"
                           VerticalOptions="Fill"
                           HorizontalOptions="Fill" />
                </Grid>
            </Frame>
        </StackLayout>
    </ScrollView>
</ContentPage>
```

To demonstrate the work with the ScrollView layout, you need to build and run the software application. When you run the software application, you can see that all four Frame layouts do not fit on the screen of the mobile device. The ScrollView layout with the help of scrolling allows you to organize the viewing of information in all Frame layouts.


3:27
Александр Овечкин
Хоккеист. Капитан хоккейного клуба НХЛ Вашингтон Кэпиталс
Павел Дацюк
Хоккеист. Олимпийский чемпион 2018 года
Сергей Мозякин
Хоккеист. Лучший бомбардир в истории

Below we consider the features of working with the AbsoluteLayout layout, which is designed to place controls within the layout with the indication of absolute values of coordinates and sizes. Also, to place controls within the layout, values proportional to the values of parameters characterizing the size and location of the layout can be specified. The AbsoluteLayout layout has the following properties:

1. The LayoutBounds property determines the location and size of a control that is placed within a layout. The default value of this property is (0, 0, AutoSize, AutoSize).

2. The LayoutFlags property determines the use of property values that characterize the size and position of the layout to obtain different proportions when placing controls within the layout. This property can take the following values:

The None value specifies that the layout's width and height values will be interpreted as absolute values (this is the default value for this property);

The XProportional value specifies that the value of x will be
be interpreted as proportional;
The YProportional value specifies that the y value will be
be interpreted as proportional;
The WidthProportional value specifies that the width value will be
be interpreted as proportional;
The HeightProportional value specifies that the height value will be
be interpreted as proportional;

The PositionProportional value specifies that the x and y values will be
be interpreted as proportional;

The SizeProportional value specifies that the width and height values will be
be interpreted as proportional;

The value All specifies that all values (x, y, width, height) will be
be interpreted as proportional.

To work with a layout of the AbsoluteLayout type, a project is created
AbsLayoutDemo for a multiplatform software application using the "Empty" template. You need to go to the "Solution Explorer" dialog box, select and open the MainPage.xaml file and type the program code given below.

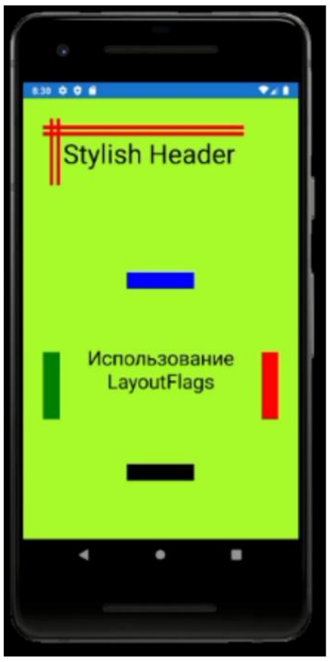
The first half of the code creates a logo of four intersecting lines, which are created using four BoxView controls and a Label control. The position of each BoxView control is determined by the first two absolute values specified as parameters in the LayoutBounds property. The size of each control is determined by the third and fourth parameters in the LayoutBounds property.

In the second half of the code, four BoxView controls and one Label control are positioned within the layout using proportional values (the LayoutFlags property is set to PositionProportional), with the controls sized using absolute values. Thus, the first two values specified in the LayoutBounds property for the BoxView and Label controls determine how the controls are positioned using proportional values.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="AbsLayoutDemo.MainPage"
             Title="AbsolutLayout Demo"
             Padding="10"
             BackgroundColor="GreenYellow">
   <AbsoluteLayout Margin="20">
      <BoxView Color="Red"
               AbsoluteLayout.LayoutBounds="0, 10, 300, 5" />
      <BoxView Color="Red"
               AbsoluteLayout.LayoutBounds="0, 20, 300, 5" />
      <BoxView Color="Red"
               AbsoluteLayout.LayoutBounds="10, 0, 5, 100" />
      <BoxView Color="Red"
               AbsoluteLayout.LayoutBounds="20, 0, 5, 100" />
      <Label Text="Stylish Header"
             FontSize="40"
             TextColor="Black"
```

```
        AbsoluteLayout.LayoutBounds="30, 25" />
    <BoxView Color="Blue"
         AbsoluteLayout.LayoutBounds="0.5,0.4,100,25"
        AbsoluteLayout.LayoutFlags="PositionProportional" />
    <BoxView Color="Green"
         AbsoluteLayout.LayoutBounds="0,0.7,25,100"
        AbsoluteLayout.LayoutFlags="PositionProportional" />
    <BoxView Color="Red"
         AbsoluteLayout.LayoutBounds="1,0.7,25,100"
        AbsoluteLayout.LayoutFlags="PositionProportional" />
    <BoxView Color="Black"
         AbsoluteLayout.LayoutBounds="0.5,0.9,100,25"
        AbsoluteLayout.LayoutFlags="PositionProportional" />
    <Label Text="Using LayoutFlags"
        FontSize="30"
        TextColor="Black"
        HorizontalOptions="Center"
        HorizontalTextAlignment="Center"
        AbsoluteLayout.LayoutBounds="0.5,0.65,300,80"
        AbsoluteLayout.LayoutFlags="PositionProportional" />
  </AbsoluteLayout>
</ContentPage>
```

To demonstrate how to work with the AbsolulLayout type layout, a software application is launched. After the software application is launched, an emblem with text is displayed at the top of the user interface "Stylish Header" using absolute positioning of controls, and the four rectangles labeled "Using LayoutFlags" at the bottom of the UI are achieved using proportional positioning.



Next, we consider working with a layout of the RelativeLayout type, which is used to place controls within a layout using relative or absolute values of parameters that characterize the sizes of controls. In this case, to obtain the values of the parameters, the values that characterize the sizes of the layout.

relative are used

The RelativeLayout type layout has the following properties for setting values that characterize the location and size of controls:

The XConstraint property specifies the relative X-axis location for control within the layout, taking into account the size of the layout;

The YConstraint property specifies the relative Y-axis position for control within the layout, taking into account the size of the layout;

The WidthConstraint property specifies a relative width value for a control located within a layout, taking into account the size of the layout;

The HeightConstraint property specifies a relative height value for a control located within a layout, taking into account the size of the layout;

The BoundsConstraint property specifies the relative position and size of a control within a layout, taking into account its size.
layout.

In XAML, the ConstraintExpression markup extension is used to organize the absolute positioning of controls. This markup extension is used to associate the position and size of an element with the property values that characterize the RelativeLayout layout or the control relative to which the placement is performed. The ConstraintExpression class has the following properties:

The Constant property defines a constant component to specify the location or size of the control;

The ElementName property specifies the name of the user interface element relative to which another user interface element is placed;

The Factor property specifies the scaling factor used to determine the location and size of the control being placed relative to the control specified in the property.

ElementName (by default this property has a value of 1);

The Property property specifies the name of the property in the Source element, used when placing a control;

The ConstraintType property specifies the type of relative positioning.
control element.

The ConstraintType property can take the following values:

The RelativeToParent value indicates that the control is placed relative to the "parent" element within which the control is located;

the RelativeToView value indicates that the control is placed relative to some other, non-parent control;

The Constant value indicates that there is a constant component to the control's position or size.

To consider the features of working with the RelativeLayout type layout, the RelatLayoutDemo project is created for a multiplatform software application using the "Empty" template. After creating the project, you need to go to the Solution Explorer window, select and open the file

MainPage.xaml and then type the XAML code shown below.

In the first part of the program code, using absolute values of parameters characterizing the placement and sizes of control elements, horizontal lines of a double frame are formed along the page borders by using four BoxView controls.

In the second part of the program code, using absolute values of parameters characterizing the placement and sizes of control elements, a BoxView control with rounded corners is formed, as well as a Label control located on top of it with the text "AbsolutLayoutDemo".

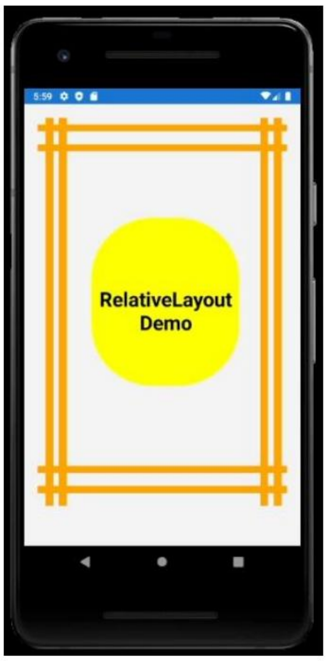
The position of each BoxView control, as well as the Label control, is determined using absolute values specified in the XConstraint and YConstraint properties. The size of each BoxView control is determined using absolute values specified in the WidthConstraint and HeightConstraint properties.

In the third part of the program code, vertical lines of a double frame along the page borders are formed using relative values of parameters characterizing the placement and sizes of control elements. The lines are formed using four BoxView controls. In this case, the relative values characterizing the position and sizes of BoxView controls are set using the ConstraintExpression markup extension.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="RelatLayoutDemo.MainPage"
              BackgroundColor="WhiteSmoke">
    <RelativeLayout Margin="20">
        <BoxView Color="Orange"
                 RelativeLayout.XConstraint="0"
                 RelativeLayout.YConstraint="10"
                 RelativeLayout.WidthConstraint="400"
                 RelativeLayout.HeightConstraint="10" />
        <BoxView Color="Orange"
                 RelativeLayout.XConstraint="0"
                 RelativeLayout.YConstraint="40"
                 RelativeLayout.WidthConstraint="400"
        RelativeLayout.HeightConstraint="10" />
<BoxView Color="Orange"
        RelativeLayout.XConstraint="0"
        RelativeLayout.YConstraint="520"
        RelativeLayout.WidthConstraint="400"
        RelativeLayout.HeightConstraint="10" />
<BoxView Color="Orange"
        RelativeLayout.XConstraint="0"
        RelativeLayout.YConstraint="550"
        RelativeLayout.WidthConstraint="400"
        RelativeLayout.HeightConstraint="10" />
<BoxView Color="Yellow"
        RelativeLayout.XConstraint="80"
        RelativeLayout.YConstraint="150"
        RelativeLayout.WidthConstraint="220"
        RelativeLayout.HeightConstraint="250"
        CornerRadius="90"/>
<Label Text="RelativeLayoutDemo"
      HorizontalTextAlignment="Center"
      FontSize="30" TextColor="Black"
      FontAttributes="Bold"
      RelativeLayout.XConstraint="5"
      RelativeLayout.YConstraint="250" />
<BoxView Color="Orange"
        RelativeLayout.XConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
            Property=Width,
            Constant=-360}"
        RelativeLayout.YConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
            Property=Height,
            Constant=-630}"
        RelativeLayout.WidthConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
            Property=Width,
            Constant=-360}"
        RelativeLayout.HeightConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
            Property=Height,
            Constant=-30}"/>
<BoxView Color="Orange"
```

```
        RelativeLayout.XConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
            Property=Width,
            Constant=-340}"
        RelativeLayout.YConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
            Property=Height,
            Constant=-630}"
        RelativeLayout.WidthConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
            Property=Width,
            Constant=-360}"
        RelativeLayout.HeightConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
            Property=Height,
            Constant=-30}"/>
<BoxView Color="Orange"
        RelativeLayout.XConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
            Property=Width,
            Constant=-20}"
        RelativeLayout.YConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
            Property=Height,
            Constant=-630}"
        RelativeLayout.WidthConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
            Property=Width,
            Constant=-360}"
        RelativeLayout.HeightConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
            Property=Height,
            Constant=-30}"/>
<BoxView Color="Orange"
        RelativeLayout.XConstraint=
        "{ConstraintExpression
            Type=RelativeToParent,
                Property=Width,
                Constant=-40}"
            RelativeLayout.YConstraint=
            "{ConstraintExpression
                Type=RelativeToParent,
                Property=Height,
                Constant=-630}"
            RelativeLayout.WidthConstraint=
            "{ConstraintExpression
                Type=RelativeToParent,
                Property=Width,
                Constant=-360}"
            RelativeLayout.HeightConstraint=
            "{ConstraintExpression
                Type=RelativeToParent,
                Property=Height,
                Constant=-30}"/>
    </RelativeLayout>
</ContentPage>
```

To demonstrate how to work with the RelativeLayout layout, you need to build the solution and then run the software application if there are no errors. The placement and sizes of the four horizontal frame lines are obtained using the absolute values of the properties that characterize the BoxView controls. The placement and sizes of the four vertical frame lines are obtained using the relative values of the properties of the BoxView controls, which depend on the values of the properties of the RelativeLayout layout. The placement and sizes of the BoxView and Label controls located in the middle of the page are obtained using the absolute values of the properties that characterize the BoxView and Label controls.



Next, we move on to studying the features of working with the FlexLayout type layout, which, like the StackLayout type layout, can arrange the location of the controls contained in it horizontally and vertically. At the same time, the FlexLayout type layout can also arrange the placement of controls if they do not fit in a row (column). In this case, the size, location, and alignment of the controls are controlled. The FlexLayout type layout can also adapt the display of controls to different screen sizes.

The FlexLayout layout has the following properties:

1. The Direction property determines the direction in which controls are arranged within the layout. The default value of the property is Row, which means that controls are arranged horizontally within the layout (by rows). In this case, the horizontal axis is the main axis, and the vertical axis is the secondary axis. Assigning the Column value to the property causes controls to be arranged vertically within the layout (by columns). If controls are arranged by columns, then the layout's main axis is vertical, and the secondary axis is

horizontal axis.

Thus, the Direction property can take the following values:

The Column value specifies the placement of controls on top down vertically within the layout;

The ColumnReverse value specifies the layout of the controls from bottom to top vertically within the layout;

The Row value specifies the arrangement of controls from left to right horizontally within the layout;

RowReverse value specifies the layout of controls from right to left horizontally within the layout.

2. The AlignItems property specifies the order in which the elements are placed. The AlignItems property can take the following values:

The Stretch value specifies whether the control is stretched vertically or horizontally depending on the direction of the secondary axis (this value is set by default);

The Center value sets the control to be centered. relative to the auxiliary axis;

the Start value specifies the position of the control on the left (horizontal) or top (vertical) depending on the direction of the secondary axis;

The End value specifies the position of the control to the right (horizontal) or bottom (vertical), depending on the direction of the minor axis.

3. The AlignContent property is similar to the AlignItems property, but is used to work with rows or columns as a whole, rather than working with

with individual controls. This property can take the following values:

The Center value specifies that the row group will be centered in the layout;

the End value specifies that the row group will be placed at the end of the parent element;

the SpaceAround value specifies that there will be equal space between the lines, and half as much space between the top and bottom lines and the top and bottom edges of the parent element, respectively;

the SpaceBetween value specifies that the top and bottom rows will be positioned along the top and bottom edges of the parent element, respectively, and that the remaining rows will be equally spaced;

the SpaceEvenly value indicates that there will be the same distance between all lines and the same distance before the first line and after the last line;

the Start value specifies that the row group will be placed at the start of the parent element;

The Stretch value specifies that the row group will be stretched from the beginning to the end of the parent element.

4. The JustifyContent property specifies the order in which elements are displayed on the main axis. The JustifyContent property can have the following values:

the Start value corresponds to the position of the control on the left (horizontal) or top (vertical) depending on the direction of the main axis (this value is set by default);

the End value corresponds to the position of the control on the right (horizontal) or at the bottom (vertical) depending on the direction of the main axis;

the Center value corresponds to the position of the control in the center horizontally or vertically, depending on the direction of the main axis;

The SpaceBetween value sets the spacing between controls;

The SpaceAround value sets the space around each control;

SpaceEvenly value - sets equal space between controls, as well as before the first and last element in a row, or above the first and after the last element in a column.

5. The Wrap property causes the control to wrap to the next row or column (depending on the layout of the controls) if it does not fit in the row or column. The property can take the following values:

The NoWrap value (the default option) corresponds to not wrapping the control to the beginning of the next row or column (in this case, the size of the controls is reduced so that they fit in the row or column);

The Wrap value corresponds to the presence of a wrapping of the control element in beginning of the next row or column;

The Reverse value ('wrap-reverse' in XAML) corresponds to the presence of move the control to the end of the next row or column;

6. The Order property allows you to change the order in which controls are arranged within a layout. Typically, controls are arranged within a layout in the order in which they are specified in the code using the Children property. The default value of this property is 0. If the Order property of the first control within a layout is set to a value that is lower than that of the other controls within the layout, the control will be displayed as the first item in a row or column. Similarly, a control will be displayed last in a row or column if the Order property of the control is set to a value that is higher than that of the other controls.

7. The Basis property specifies the width of controls when they are laid out horizontally, or the height of controls when they are laid out vertically. The property value can be specified either in absolute values or as a percentage of the layout size. The default value of the Basis property is FlexBasis.Auto, which means that the previously specified width or height of the control is used.

8. The Grow property specifies how to distribute space between controls. The Grow property has a default value of 0. If a positive value is specified, space on the main axis is allocated to the control and other controls that have positive Grow property values. Space on the main axis will be allocated proportionally to the property values that the controls have.

9. The Shrink property specifies which child elements receive priority when displaying their full sizes. The default value of the property is minus 1, but the property value must be greater than or equal to 0. The Shrink property is taken into account if the Wrap property value is NoWrap and the total width of the row or column with controls is greater than the width or height of the FlexLayout. When the property value is 0, the full size of the control is displayed in the layout. When the property value is greater than 0, the size of the controls is compressed when they are displayed on the device screen.

To consider the features of working with the FlexDemo layout, a project is created for a Xamarin.Forms multiplatform software application using the "Empty" template. First, you need to go to the "Solution Explorer" dialog box and create the EnumPicker class using the sequence of actions discussed earlier. Next, you need to open the created EnumPicker.cs file and type the C Sharp program code given below.

The code creates a template for the EnumPicker control, which is referenced by the code in the MainPage.xaml file (created later). The EnumPicker control is a child of the Picker control, which is used to select a value from a list.

The EnumType property is created and referenced by the program code in the MainPage.xaml file. The program code describes how to work with the property. EnumType if the value in the EnumPicker control that the user is currently working with has changed.

The oldValue and newValue properties of the EnumPicker control, which is associated with the FlexLayout layout properties, are passed the old and selected new values received from one of the EnumPicker controls (these controls are defined in the MainPage.xaml file, and the user will be working with one of them at the current time).

The data source from which the values of the oldValue and newValue properties come is specified using the ItemsSource property of the picker object. The picker object receives data from the Picker control that is defined in the MainPage.xaml file and with which the user is currently working.

The program code uses conditional statements to check the old and new values received from the EnumPicker controls listed in the MainPage.xaml file if the oldValue and newValue properties change. The program code also declares the EnumType property.

```
using System;
using System.Reflection;
using Xamarin.Forms;
namespaceFlexDemo
{ class EnumPicker: Picker
  {
   public static readonly BindableProperty EnumTypeProperty =
       BindableProperty.Create("EnumType",
                                               typeof(Type),
                                               typeof(EnumPicker),
              propertyChanged: (bindable, oldValue, newValue) =>
              {
                 EnumPicker picker = (EnumPicker)bindable;
                 if (oldValue != null)
                 {
                    picker.ItemsSource = null;
                 }
                 if (newValue != null)
                 {
                    if (!((Type)newValue).GetTypeInfo().IsEnum)
                       throw new ArgumentException
                      ("EnumPicker: EnumType property must be
                                                    enumeration type");
                    picker.ItemsSource = Enum.GetValues((Type)newValue);
                        }
                      });
                  public Type EnumType
                  {
                    set => SetValue(EnumTypeProperty, value);
                    get => (Type)GetValue(EnumTypeProperty);
                  }
                }
              }
```

Next, you need to open the MainPage.xaml file and type the XAML program code given below. The program code inside the

ContentPage type page contains a Grid type layout, inside which a table of two rows is marked up. The first row contains the Grid layout. The second row contains the layout

A FlexLayout to which a layout is bound using a BindingContext

Grid from the first row. The Label controls placed in the layout are styled using Style and Setter elements so that the Label control is vertically centered.

Inside the Grid layout, the first row contains another, internal Grid layout, which creates six rows and two columns.

In the first row of the internal Grid layout, the first column contains

label that displays the text "Number of Labels:". The second column of the first row of this layout contains a StackLayout with horizontally arranged controls, which are Label and Stepper. The Label control is bound to the Stepper control using the BindingContext binding. The value selected in the Stepper control will be converted to a text value and displayed in the Label control. The value selected using the Stepper control shows the number of Label controls placed inside the FlexLayout. When the value selected using the Stepper control changes, the OnStepperChanged event handler is triggered, which is given in the C Sharp program code in the MainPage.xaml.cs file, which will be created

Later.

In the second row of the internal Grid layout, the first column contains a label that displays the name of the "Direction:" property. In the second column of the second row, there is an EnumPicker element created based on the EnumPicker class from the EnumPicker.cs file. In this case, the element

EnumPicker is bound to the Direction property of the FlexLayout using Binding .

In the third row of the internal Grid layout, the first column contains a label that displays the name of the "Wrap:" property.

In the second column of the second row there is an EnumPicker element, which is bound to the Wrap property of the FlexLayout using Binding.

In the fourth row of the internal Grid layout, the first column contains a label that displays the name of the "JustifyContent:" property. In the second column of the second row, there is an element
EnumPicker, which is bound to the JustifyContent property of the FlexLayout using Binding.

In the fifth row of the internal Grid layout, the first column contains a label that displays the name of the "AlignItems:" property. In the second column of the second row, there is an EnumPicker element that is bound to the AlignItems property of the FlexLayout layout.

In the sixth row of the internal Grid layout, the first column contains a label that displays the name of the "AlignContent:" property. In the second column of the second row, there is an EnumPicker element that is bound to the AlignContent property of the FlexLayout.

The EnumPicker control displays a list of values for the FlexLayout property that you are currently working with (these are the Direction, Wrap, JustifyContent, AlignItems, AlignContent properties).

Thus, five EnumPicker controls are created in this file, intended to display the values of the FlexLayout layout properties. At the same time, it is envisaged that the number of Label controls displayed in the user

interface when the software application is launched is three. The initial values of the FlexLayout layout properties, which will be in effect when the software application is launched, are also set.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
          xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
           xmlns:local="clr-namespace:FlexDemo"
           x:Class="FlexDemo.MainPage">
  <Grid Margin="10, 0">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinitionHeight="*" />
    </Grid.RowDefinitions>
    <!-- Control Panel -->
    <Grid BindingContext="{x:Reference flexLayout}"
       Grid.Row="0">
      <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="*" /> </
Grid.ColumnDefinitions>
<Grid.Resources>
    <ResourceDictionary>
        <Style TargetType="Label"> <Setter

            Property="VerticalOptions" Value=" Center" /> </Style> </

    ResourceDictionary> </
Grid.Resources> <Label
Text="Number of Labels:"
       Grid.Row="0" Grid.Column="0" />
<StackLayout Orientation="Horizontal"
             Grid.Row="0" Grid.Column="1">

    <Label
     Text="{Binding Source={x:Reference numberStepper},
                         Path=Value,
                         StringFormat='{0:F0}'}" /> <Stepper
    x:Name="numberStepper"
             Minimum="0"
             Maximum="99"
             Increment="1"
             Value="3"
             ValueChanged="OnStepperChanged" /> </
StackLayout>
<Label Text="Direction:"
       Grid.Row="1" Grid.Column="0" />
<local:EnumPicker EnumType="{x:Type FlexDirection}"
                  SelectedItem="{Binding Direction}"
                  Grid.Row="1" Grid.Column="1" /> <Label
Text="Wrap:"
       Grid.Row="2" Grid.Column="0" />
<local:EnumPicker EnumType="{x:Type FlexWrap}"
                  SelectedItem="{Binding Wrap}"
                  Grid.Row="2" Grid.Column="1" />
<Label Text="JustifyContent:"
       Grid.Row="3" Grid.Column="0" />
<local:EnumPicker EnumType="{x:Type FlexJustify}"
                  SelectedItem="{Binding JustifyContent}"
                  Grid.Row="3" Grid.Column="1" />
            <Label Text="AlignItems:"
                Grid.Row="4" Grid.Column="0" />
            <local:EnumPicker EnumType=
                                        "{x:Type FlexAlignItems}"
                        SelectedItem="{Binding AlignItems}"
                        Grid.Row="4" Grid.Column="1" />
            <Label Text="AlignContent:"
                Grid.Row="5" Grid.Column="0" />
            <local:EnumPicker EnumType=
                                      "{x:Type FlexAlignContent}"
                        SelectedItem="{Binding AlignContent}"
                        Grid.Row="5" Grid.Column="1" />
          </Grid>
          <!—FlexLayout Layout -->
          <FlexLayout x:Name="flexLayout"
                 BackgroundColor="AliceBlue"
                 Grid.Row="1" />
        </Grid>
      </ContentPage>
```

Next, you need to open the MainPage.xaml.cs file and type the C Sharp program code given below.

The code specifies an array of colors to select the color of the text in the Label controls placed in the FlexLayout. The digitsText and decadeText arrays are used to display the text with the number in the Label controls.

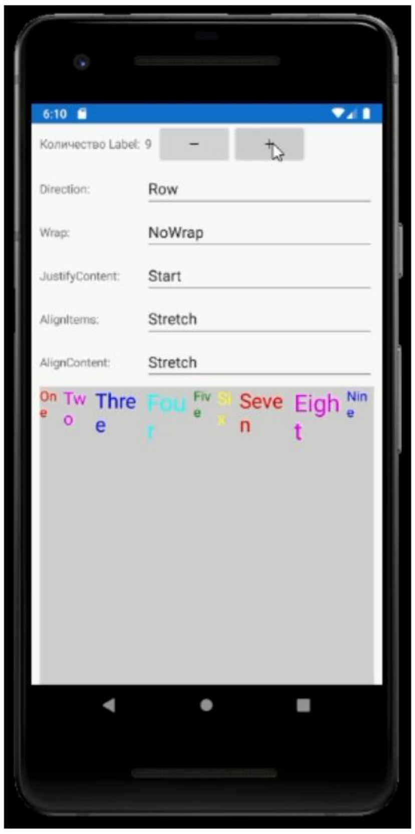
The OnStepperChanged event handler is referenced from the code in MainPage.xaml. The input parameter is the number of Label controls selected by the Stepper control named numberStepper, which is defined earlier in MainPage.xaml.

As an object that the handler will act on OnStepperChanged is a FlexLayout. The numberStepper.Value parameter creates a count value that specifies the number of Label controls inside the FlexLayout. The placement of the Label controls is organized using the Children property and a loop.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;
```

```
namespace FlexDemo {

    public partial class MainPage : ContentPage { static

    Color[] colors = {Color.Red, Color.Magenta,
                    Color.Blue, Color.Cyan, Color.Green, Color.Yellow };
        static string[] digitsText = { "", "One", "Two", "Three",
       "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten",
       "Eleven", "Twelve", "Thirteen", "Fourteen",
       "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"}; static string[]
        decadeText = { "", "", "Twenty", "Thirty",
           "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety"};
        public MainPage()

        { InitializeComponent();
            OnStepperChanged(flexLayout, new
           ValueChangedEventArgs(0, numberStepper.Value)); } void

        OnStepperChanged(object sender,
                                              ValueChangedEventArgs args)
        {
            int count = (int)args.NewValue; while
            (flexLayout.Children.Count > count) {

   flexLayout.Children.RemoveAt(flexLayout.Children.Count - 1);

            } while (flexLayout.Children.Count < count) {

                int number = flexLayout.Children.Count + 1; string text
                = ""; if (number <
                20) {

                    text = digitsText[number];

                }

                else { text = decadeText[number / 10] +
                             (number % 10 == 0 ? "" : "-") +
                                  digitsText[number % 10];
                }
                Label label = new Label {

                    Text = text,
                    FontSize = 16 + 4 * ((number - 1) % 4), TextColor
                    = colors[(number - 1) % colors.Length],
                        BackgroundColor = Color.LightGray
                    };
                    flexLayout.Children.Add(label);
                }
            }
        }
    }
```

To demonstrate how to use the FlexLayout layout, you must first build the software application and then run it. Once the software application is running, the control panel is displayed at the top of the user interface, and the layout is displayed at the bottom.
FlexLayout. The control panel displays a Stepper control, which allows you to change the number of Label controls placed in the FlexLayout. The control panel also contains five EnumPicker controls, which allow you to set the values of the previously discussed FlexLayout properties. When the software application starts, three Label controls are displayed by default. The values of the FlexLayout properties displayed when the software application starts were previously set in the program code in the MainPage.xaml file. The control panel is used to select the number of Label elements placed within the layout. Then, by selecting the values of the layout properties, the arrangement of the Label controls on the user interface is controlled.



Количество Label: 9

Direction:	Row
Wrap:	NoWrap
JustifyContent:	Start
AlignItems:	Stretch
AlignContent:	Stretch

On e Tw o Thre e Fou r Fiv e Si x Seve n Eigh t Nin e