Работа с макетами

Макеты позволяют упорядочивать и группировать элементы управления внутри страниц в пользовательском интерфейсе программного приложения. Выбор макета при проектировании пользовательского интерфейса требует от разработчика знания того, как должны располагаться элементы управления внутри страниц, отображаемых в процессе работы программного приложения.

На данном занятии будут рассмотрены следующие макеты:

1. Макет Grid
2. Макет Frame
3. Макет StackLayout
4. Макет ContentView
5. Макет ScrollView
6. Макет AbsolutLayout
7. Макет RelativeLayout
8. Макет FlexLayout

Рассмотрим особенности работы с макетом типа Grid, который позволяет размещать элементы управления в ячейках таблицы, создаваемой внутри макета. Таким образом, в каждой ячейке может содержаться сразу несколько элементов управления или макетов.

Макет типа Grid имеет следующие свойства:

1. Свойство Column определяет номер колонки в таблице. Нумерация столбцов начинается с 0. Значение по умолчанию равно 0.
2. Свойство ColumnDefinitions определяет свойства для сформированной новой колонки в таблице.
3. Свойство ColumnSpacing определяет расстояние между столбцами таблицы. Значение этого свойства по умолчанию равно 6 единицам.
4. Свойство ColumnSpan определяет количество объединяемых столбцов в пределах одной строки. Значение этого свойства по умолчанию равно 1.
5. Свойство Row определяет номер строки в таблице. Нумерация строк начинается с 0. Значение свойства по умолчанию равно 0.
6. Свойство RowDefinitions определяет свойства для сформированной новой строки в таблице.
7. Свойство RowSpacing определяет расстояние между строками таблицы. Значение этого свойства по умолчанию равно 6 единицам.
8. Свойство RowSpan определяет количество объединяемых строк в пределах одного столбца. Значение этого свойства по умолчанию равно 1.

Для рассмотрения особенностей работы с макетом типа Grid создается проект GridDemo для мультиплатформенного программного приложения с помощью шаблона «Пустой». Необходимо зайти в окно «Обозреватель решений», выбрать и открыть файл MainPage.xaml и набрать программный код XAML, приведенный ниже.

В программном коде предусматривается формирование макета типа Grid, в котором во всех трех строках, в каждой из которых по 2 столбца, расположены элементы управления BoxView и Label. Каждый столбец имеет

одинаковую ширину. В третьей строке объединяются два столбца. Элемент управления BoxView в третьей строке захватывает оба столбца. В одной ячейке могут быть размещены несколько элементов управления.

```
<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>
```

Для демонстрации работы с макетом Grid необходимо сначала осуществить сборку, а затем запустим программное приложение для демонстрации использования макета типа Grid. На странице отображаются элементы управления BoxView и Label, расположенные в ячейках таблицы, образованной с помощью макета Grid.

Далее перейдем к изучению особенностей работы с макетами типов Frame и StackLayout. Макеты типа Frame используются для создания границ вокруг элемента управления, который находится внутри макета. Макет типа Frame имеет следующие свойства:

1. Свойство BorderColor определяет цвет границы макета типа Frame.
2. Свойство CornerRadius определяет радиус закругления угла макета.

3. Свойство HasShadow определяет наличие тени вокруг макета.

Макет StackLayout позволяет размещать внутри себя элементы управления или макеты друг под другом по вертикали или рядом друг с другом по горизонтали в зависимости от значения свойства Orientation. Свойство Spacing регулирует интервал между элементами управления внутри макета и по умолчанию имеет значение, равное 6.

Для изучения особенностей работы с макетами типа Frame и StackLayout создается проект StackLayoutDemo для мультиплатформенного программного приложения с помощью шаблона «Пустой». Необходимо зайти в окно «Обозреватель решений», выбрать и открыть файл MainPage.xaml, а далее набрать программный код, приведенный ниже.

В программном коде рассмотрено создание макета StackLayout, в котором вертикально друг под другом расположены три элемента управления Label и шесть макетов типа Frame. Каждый макет типа Frame охватывает содержащийся внутри него макет типа StackLayout, внутри которого по горизонтали расположены элемент управления BoxView для отображения цвета и элемент управления Label, в котором отображается названия цвета в элементе управления BoxView. Произведена настройка цвета фона страницы, цвета, размера шрифта, стиля написания и расположения текста внутри элементов управления Label, а также расположения элементов управления внутри макета. Также с помощью свойства BorderColor макета Frame произведена настройка толщины контуров, окружающих макеты.

```
<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>
```

Для демонстрации работы с макетами типа Frame и StackLayout необходимо сначала собрать, а затем запустим программное приложение.

В пользовательском интерфейсе отображаются находящиеся внутри макета StackLayout три элемента управления Label и шесть макетов Frame, каждый из которых, задает линию по контуру макета StackLayout, находящегося внутри макета Frame. Каждый макет StackLayout внутри макета Frame содержит элементы управления BoxView и Label, располагающиеся горизонтально.

Далее рассмотрим особенности работы с макетом типа ContentView, который содержит один дочерний элемент и обычно используется для создания шаблонов нестандартных и многократно используемых элементов управления.

Для работы с макетом типа ContentView создадим проект ContViewDemo для мультиплатформенного программного приложения Xamarin.Forms с использованием шаблона «Пустой». Далее с использованием описанной ранее последовательности действий создается объект «Страница содержимого» с именем Card. Далее открывается созданный файл Card.xaml и производится набор приведенного ниже программного кода XAML. С помощью программного кода XAML создается шаблон нестандартного элемента управления в виде карты, которая имеет вид прямоугольника со скругленными углами и в которой с левой стороны в прямоугольнике отображен цвет, а с правой стороны размещен текст.

В программном коде макет ContentView содержит шаблон нестандартного элемента управления и задает значение атрибута x:Name, равное this. Это значение можно использовать для доступа с помощью привязки Binding к элементу управления, который будет создаваться на основе шаблона.

В шаблоне элемента управления, который называется Card объявляются следующие свойства:

свойство CardTitle задает текст, отображаемый в элементе управления;

свойство CBColor задает цвет границы шаблонного элемента управления и цвет линии разделителя;

свойство TarColor задает цвет квадрата, отображаемого с помощью элемента управления BoxView с левой стороны шаблонного элемента управления;

свойство CardColor задает цвет фона в шаблонном элементе управления;

свойство ColText задает цвет текста в шаблонном элементе управления.

В макете ContentView содержится макет Frame, представляющий собой прямоугольник со скругленными краями, цветом фона, задаваемым свойством CardColor, и цветом границы, задаваемой свойством CBColor. Кроме этого, заданы отступы от границ макета, и задано значения признака для наличия тени вокруг макета. Внутри макета Frame расположен макет Grid. Внутри макета Grid находится таблица, в которой одна строка и три столбца. В первом столбце находится элемент управления Frame элементом управления BoxView. За счет задания значения Margin можно добиться наличия окантовки вокруг элемента управления BoxView. Цвет, которым заполнен элемент управления BoxView, задается свойством TarColor. Весь второй столбец занимает элемент управления BoxView, который выполняет роль разделителя между столбцами. Цвет этого элемента управления задается с помощью

свойства CBColor. В третьей ячейке находится элемент управления Label, с помощью которого отображается текст, который задается при помощи свойства CardTitle. Цвет текста в элементе управления Label задается свойством ColText. С помощью Binding реализуется возможность привязки свойств макетов Frame и элементов управления BoxView и Label к свойствам 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>
        <RowDefinition Height="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>
```

Далее необходимо открыть файл Card.xaml.cs и набрать там программный код Си Шарп, приведенный далее. В программном коде создаются свойства CardTitle, CBColor, TarColor, CardColor, ColText для шаблонного элемента управления Card. С помощью BindableProperty осуществляется возможность привязки свойств элемента управления, создаваемого на базе шаблона Card, к свойствам этого шаблона.

Для получения и задания значений привязываемых свойств для элемента управления, создаваемого на основе шаблона Card, используются методы GetValue и SetValue.

```
using System.ComponentModel;
using Xamarin.Forms;
namespace ContViewDemo
{
  [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),
                Color.Black);
public static readonly BindableProperty
   CardColorProperty = BindableProperty.Create(
                nameof(CardColor),
                typeof(Color),
                typeof(Card),
                Color.White);
public static readonly BindableProperty
   ColTextProperty = BindableProperty.Create(
                nameof(ColText),
                typeof(Color),
                typeof(Card),
                Color.Red);
public string CardTitle
{
   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 Color ColText
{
   get => (Color)GetValue(Card.ColTextProperty);
   set => SetValue(Card.ColTextProperty, value);
}
```

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

Далее следует выбрать в диалоговом окне «Обозреватель решений» файл MainPage.xaml, открыть его и набрать программный код XAML, приведенный далее.

В программном коде для страницы типа ContentPage добавлена ссылка на controls - пространство имен пользовательского элемента управления, то есть, на шаблон элемента управления, программный код которого приведен в файле Card.xaml.

Страница MainPage содержит макет StackLayout, внутри которого находятся элемент управления Label для вывода названия страницы, а также три нестандартных элемента управления, являющихся экземплярами шаблона Card.

При каждом обращении к шаблону программный код передает шаблону значения привязываемых свойств CardTitle, CBColor, TarColor, CardColor, ColText. Привязка этих свойств к шаблону была произведена ранее в программном коде, приведенном в файлах Card.xaml и Card.xaml.cs.

```
<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"
```

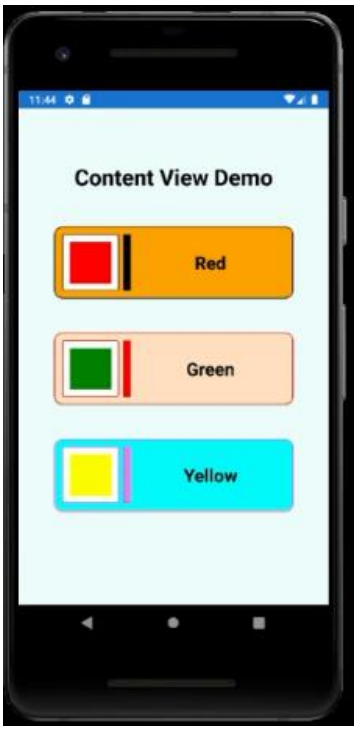
```
                CardColor="Orange"
                ColText="Black"/>
        <controls:Card CardTitle="Green"
                CBColor="Red"
                TarColor="Green"
                CardColor="Bisque"
                ColText="Black"/>
         <controls:Card CardTitle="Yellow"
           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

Для демонстрации работы с макетом ContentView необходимо сначала собрать, а затем запустить программное приложение.

После запуска на пользовательском интерфейсе отображаются три элемента управления, созданные на основе шаблона Card, реализованного с помощью макета ContentView.

В процессе построения пользовательского интерфейса программный код три раза обращается к шаблону элемента управления с новыми значениями привязываемых свойств.

В результате на экране мобильного устройства отображаются три отличающихся друг от друга экземпляра шаблонного элемента управления.



Далее рассмотрим особенности работы с макетом типа ScrollView, с помощью которого можно выполнять прокрутку элементов управления, расположенных внутри данного макета. Очень часто внутри макета типа

ScrollView размещается макет типа StackLayout. При этом на экране устройства невозможно отобразить все элементы управления, расположенные внутри макета типа StackLayout. Для задания направления прокрутки (горизонтальная, вертикальная или в обе стороны) необходимо установить соответствующее значение свойства Orientation макета типа ScrollView.

Для рассмотрения особенностей работы с макетом типа ScrollView создается проект ScrViewDemo для мультиплатформенного программного приложения с помощью шаблона «Пустой». Необходимо зайти в диалоговое окно «Обозреватель решений», выбрать и открыть файл MainPage.xaml и набрать в нём программный код, приведенный далее. В программном коде в макете ScrollView находится макет StackLayout. В макете StackLayout находятся четыре макета Frame, который создает границы находящегося внутри него макета Grid. У каждого макета Frame заданы цвет фона, цвет границы, радиус закругления углов, а также наличие тени вокруг макета. В макете Grid заданы три строки и два столбца. В первом столбце первой строки располагается круглая двухцветная эмблема, образованная с помощью макета Frame и элемента управления BoxView. Во втором столбце первой строки расположен элемент управления Label с отображаемым текстом. Всю вторую строку, в которой объединены столбцы, занимает элемент управления BoxView, который выполняет функции разделителя первой и третьей строки. В третьей строке, в которой объединены два столбца, расположен элемент управления Label, в котором также выводится текст.

```
<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>
              <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="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="Александр Овечкин"
               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="Хоккеист.
                     Капитан хоккейного клуба
                     НХЛ Вашингтон Кэпиталс"
               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="Павел Дацюк"
        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="Хоккеист. Капитан хоккейного клуба
                КХЛ Автомобилист"
        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="Сергей Мозякин"
               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="Хоккеист.
                 Лучший бомбардир в истории
                 российского хоккея"
               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="Владислав Третьяк"
                       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="Хоккеист.
                        Лучший вратарь в истории
                        в истории мирового хоккея"
                       FontAttributes="Bold"
                       FontSize="24"
                       TextColor="Black"
                       HorizontalTextAlignment="Center"
                       VerticalTextAlignment="Start"
                       VerticalOptions="Fill"
                       HorizontalOptions="Fill" />
            </Grid>
        </Frame>
    </StackLayout>
</ScrollView>
</ContentPage>
```

Для демонстрации работы с макетом типа ScrollView необходимо собрать и запустить программное приложение. При запуске программного приложения видно, что все четыре макета типа Frame не помещаются на экране мобильного устройства. Макет типа ScrollView с помошью прокрутки позволяет организовать просмотр информации во всех макетах типа Frame.

Далее рассматриваются особенности работы с макетом AbsoluteLayout, который предназначен для размещения элементов управления внутри макета с указанием абсолютных значений координат и размеров. Также для размещения элементов управления внутри макета могут указываются и значения, пропорциональные значениям параметров, характеризующих размеры и расположение макета. Макет AbsoluteLayout имеет следующие свойства:

1. Свойство LayoutBounds определяет расположение и размер элемента управления, который размещается внутри макета. Значение этого свойства по умолчанию равно (0, 0, AutoSize, AutoSize).

2. Свойство LayoutFlags определяет использование значений свойств, характеризующие размеры и положение макета, для получения различных пропорций при размещении элементов управления внутри макета. Данное свойство может принимать следующие значения:

значение None указывает, что значения ширины и высоты макета будут интерпретироваться как абсолютные значения (это значение данного свойства по умолчанию);

значение XProportional указывает, что значение x будет интерпретироваться как пропорциональное;

значение YProportional указывает, что значение y будет интерпретироваться как пропорциональное;

значение WidthProportional указывает, что значение width будет интерпретироваться как пропорциональное;

значение HeightProportional указывает, что значение height будет интерпретироваться как пропорциональное;

значение PositionProportional указывает, что значения x и y будут интерпретироваться как пропорциональные;

значение SizeProportional указывает, что значения width и height будут интерпретироваться как пропорциональные;

значение All указывает, что все значения (x, y, width, height) будут интерпретироваться как пропорциональные.

Для работы с макетом типа AbsoluteLayout создается проект AbsLayoutDemo для мультиплатформенного программного приложения с помощью шаблона «Пустой». Необходимо зайти в диалоговое окно «Обозреватель решений», выделить и открыть файл MainPage.xaml и набрать программный код, приведенный далее.

В первой половине программного кода предусматривается формирование эмблемы из четырех пересекающихся линий, которые выполнены с помощью четырех элементов управления BoxView и элемента управления Label. Положение каждого элемента управления BoxView определяется с помощью первых двух абсолютных значений, указанных в качестве параметров в свойстве LayoutBounds. Размер каждого элемента управления определяется с помощью третьего и четвертого параметра в свойстве LayoutBounds.

Во второй половине программного кода четыре элемента управления BoxView и один элемент управления Label размещаются внутри макета с использованием пропорциональных значений (значение свойства LayoutFlags равно PositionProportional), при этом размеры элементов управления установлены с помощью абсолютных значений. Таким образом, первые два значения, указанные в свойстве LayoutBounds для элементов управления BoxView и Label, определяют расположение элементов управления с использованием пропорциональных значений.

```
<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="Использование LayoutFlags"
            FontSize="30"
            TextColor="Black"
            HorizontalOptions="Center"
            HorizontalTextAlignment="Center"
            AbsoluteLayout.LayoutBounds="0.5,0.65,300,80"
            AbsoluteLayout.LayoutFlags="PositionProportional" />
    </AbsoluteLayout>
</ContentPage>
```

Для демонстрации работы с макетом типа AbsolulLayout запускается программное приложение. После запуска программного приложения в верхней части пользовательского интерфейса отображается эмблема с текстом «Stylish Header» с использованием абсолютного позиционирования элементов управления, а четыре прямоугольника с надписью: «Использование LayoutFlags» в нижней части пользовательского интерфейса получены с использованием пропорционального позиционирования.

Далее рассматривается работа с макетом типа RelativeLayout, который используется для размещения элементов управления внутри макета с использованием относительных или абсолютных значений параметров, характеризующих размеры элементов управления. При этом для получения относительных значений параметров используются значения, характеризующие размеры макета.

Макет типа RelativeLayout имеет следующие свойства для задания значений, характеризующих расположение и размеры элементов управления:

свойство XConstraint задает относительное расположение по оси X для элемента управления внутри макета, с учетом размера макета;

свойство YConstraint задает относительное расположение по оси Y для элемента управления внутри макета, с учетом размера макета;

свойство WidthConstraint задает относительное значение ширины для элемента управления, расположенного внутри макета, с учетом размера макета;

свойство HeightConstraint задает относительное значение высоты для элемента управления, расположенного внутри макета, с учетом размера макета;

свойство BoundsConstraint задает относительное расположение и размер для элемента управления, расположенного внутри макета, с учетом размера макета.

В XAML для организации абсолютного расположения элементов управления используется расширение разметки ConstraintExpression. Это расширение разметки используется для связи расположения и размера элемента со значениями свойств, характеризующими макет RelativeLayout или элемент управления, относительно которого производится размещение. При этом класс ConstraintExpression имеет следующие свойства:

свойство Constant определяет постоянную составляющую для указания расположения или размера элемента управления;

свойство ElementName определяет имя элемента пользовательского интерфейса, относительно которого производится размещение другого элемента пользовательского интерфейса;

свойство Factor определяет значение коэффициента масштабирования при определении расположения и размеров размещаемого элемента управления относительно элемента управления, указанного в свойстве ElementName (по умолчанию это свойство имеет значение 1);

свойство Property определяет имя свойства в элементе Source, используемом при размещении элемента управления;

свойство ConstraintType определяет тип относительного расположения элемента управления.

Свойство ConstraintType может принимать следующие значения:

значение RelativeToParent показывает, что размещение элемента управления производится относительно «родительского» элемента, внутри которого располагается элемент управления;

значение RelativeToView показывает, что – размещение элемента управления производится относительно какого-то другого, не «родительского» элемента управления;

значение Constant показывает, что имеется постоянная составляющая при выборе расположения или размера элемента управления.

Для рассмотрения особенностей работы с макетом типа RelativeLayout создается проект RelatLayoutDemo для мультиплатформенного программного приложения с помощью шаблона «Пустой». После создания проекта необходимо зайти в окно Обозреватель решений», выбрать и открыть файл MainPage.xaml и далее набрать программный код XAML, приведенный далее.

В первой части программного кода с использованием абсолютных значений параметров, характеризующих размещение и размеры элементов управления, формируются горизонтальные линии двойной рамки по границам страницы за счет использования четырех элементов управления BoxView.

Во второй части программного кода с использованием абсолютных значений параметров, характеризующих размещение и размеры элементов управления, производится формирование элемента управления BoxView со скругленными углами, а также расположенного поверх него элемента управления Label с текстом «AbsolutLayoutDemo».

Расположение каждого элемента управления BoxView, а также элемента управления Label определяется с использованием абсолютных значений, указанных в свойствах XConstraint и YConstraint. Размер каждого из элементов управления BoxView определяется с помощью абсолютных значений, указанных в свойствах WidthConstraint и HeightConstraint.

В третьей части программного кода с использованием относительных значений параметров, характеризующих размещение и размеры элементов управления, формируются вертикальные линии двойной рамки по границам страницы. Линии формируются с помощью четырех элементов управления BoxView. При этом установка относительных значений, характеризующих положение и размеры элементов управления BoxView производится с использованием расширения разметки ConstraintExpression.

```
<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>
```

Для демонстрации работы с макетом типа RelativeLayout необходимо собрать решение, а затем, при отсутствии ошибок, запустить программное приложение. Размещение и размеры четырех горизонтальных линий рамки получены с использованием абсолютных значений свойств, характеризующих элементы управления BoxView. Размещение и размеры четырех вертикальных линий рамки получены с использованием относительных значений свойств элементов управления BoxView, которые зависят от значений свойств макета RelativeLayout. Размещение и размеры для элементов управления BoxView и Label, находящихся в середине страницы, получены с использованием абсолютных значений свойств, характеризующих элементы управления BoxView и Label.

Далее переходим к изучению особенностей работы с макетом типа FlexLayout, который так же, как и макет типа StackLayout, может упорядочивать расположение содержащихся в нём элементов управления по горизонтали и вертикали. При этом, макет типа FlexLayout может также упорядочивать размещение элементов управления в том случае, если они не помещаются в строке (столбце). В этом случае производится контроль размеров, расположения и выравнивания элементов управления. Также макет типа FlexLayout может адаптировать отображение элементов управления под различные размеры экрана.

Макет FlexLayout обладает следующими свойствами:

1. Свойство Direction определяет направление расположения элементов управления внутри макета. Значение свойства по умолчанию – Row, что означает, что элементы управления расположены внутри макета по горизонтали (по строкам). В этом случае главной является горизонтальная ось, а вспомогательной – вертикальная ось. Присвоение свойству значения Column приводит к тому, что элементы управления будут расположены внутри макета по вертикали (по столбцам). Если элементы управления расположены по столбцам, то у макета основная ось вертикальная, а вспомогательная – горизонтальная ось.

Таким образом, свойство Direction может принимать следующие значения:

значение Column задает расположение элементов управления сверху вниз по вертикали внутри макета;

значение ColumnReverse задает расположение элементов управления снизу вверх по вертикали внутри макета;

значение Row задает расположение элементов управления слева направо по горизонтали внутри макета;

значение RowReverse задает расположение элементов управления справа налево по горизонтали внутри макета.

2. Свойство AlignItems указывает на порядок размещения элементов. Свойство AlignItems может принимать следующие значения:

значение Stretch задает растяжение элемента управления по вертикали или горизонтали в зависимости от направления вспомогательной оси (это значение установлено по умолчанию);

значение Center задает расположение элемента управления по центру относительно вспомогательной оси;

значение Start задает расположение элемента управления слева (по горизонтали) или сверху (по вертикали) в зависимости от направления вспомогательной оси;

значение End задает расположение элемента управления справа (по горизонтали) или снизу (по вертикали) в зависимости от направления вспомогательной оси.

3. Свойство AlignContent является аналогом свойства AlignItems, но используется для работы со строками или столбцами в целом, а не для работы

с отдельными элементами управления. Данное свойство может принимать следующие значения:

значение Center указывает, что группа строк будет находиться в центре макета;

значение End указывает, что группа строк будет помещена у конца родительского элемента;

значение SpaceAround указывает, что между строками будет одинаковое расстояние, а между верхней и нижней строками и верхним и нижним краями родительского элемента, соответственно, будет вдвое меньшее расстояние;

значение SpaceBetween указывает, что верхняя и нижняя строки будут располагаться по верхнему и нижнему краям родительского элемента соответственно, а между остальными строками будет одинаковое расстояние;

значение SpaceEvenly указывает, что между всеми строками будет одинаковое расстояние и такое же расстояние будет перед первой строкой и после последней;

значение Start указывает, что группа строк будет помещена у начала родительского элемента;

значение Stretch указывает, что группа строк будет растянута от начала до конца родительского элемента.

4. Свойство JustifyContent указывает на порядок вывода элементов на главной оси. Свойство JustifyContent может иметь следующие значения:

значение Start соответствует расположению элемента управления слева (по горизонтали) или сверху (по вертикали) в зависимости от направления главной оси (это значение установлено по умолчанию);

значение End соответствует расположению элемента управления справа (по горизонтали) или внизу (по вертикали) в зависимости от направления главной оси;

значение Center соответствует расположению элемента управления по центру по горизонтали или по вертикали в зависимости от направления главной оси;

значение SpaceBetween устанавливает интервалы между элементами управления;

значение SpaceAround устанавливает пробелы вокруг каждого элемента управления;

значение SpaceEvenly - устанавливает равное пространство между элементами управления, а также перед первым и последним элементом в строке или над первым и после последнего элемента в столбце.

5. Свойство Wrap приводит к переносу элемента управления в следующую строку или столбец (в зависимости от расположения элементов управления) в том случае, если он не помещается в строке или столбце. Свойство может принимать следующие значения:

значение NoWrap (вариант по умолчанию) соответствует отсутствию переноса элемента управления в начало следующую строку или столбец (при этом производится уменьшение размеров элементов управления для того, чтобы они уместились в строке или столбце);

значение Wrap соответствует наличию переноса элемента управления в начало следующей строки или столбца;

значение Reverse («wrap-reverse» в XAML) соответствует наличию переноса элемента управления в конец следующей строки или столбца;

6. Свойство Order позволяет изменить порядок, в котором располагаются элементы управления в макете. Обычно элементы управления располагаются внутри макета в том порядке, в котором они заданы в программном коде при помощи свойства Children. Значение свойства по умолчанию равно 0. Если свойству Order первого элемента управления внутри макета присвоено значение, меньшее, чем у других элементов управления внутри макета, то элемент управления будет отображаться как первый элемент в строке или столбце. Аналогично, элемент управления отображается последним в строке или столбце, если свойству Order элемента управления будет задано значение большее, чем у других элементов управления.

7. Свойство Basis задает ширину элементов управления, когда они располагаются по горизонтали, или высоту элементов управления, когда они располагаются по вертикали. Значение свойства может быть указано либо в абсолютных значениях, либо в процентах от размера макета. Значением свойства Basis по умолчанию является значение FlexBasis.Auto, которое означает, что используются заданные ранее значения ширины или высоты элемента управления.

8. Свойство Grow указывает, как распределять пространство между элементами управления. Свойство Grow имеет значение по умолчанию 0. Если задано положительное значение свойства, то пространство на главной оси выделяется элементу управления и другим элементам управления, для которых установлены положительные значения свойства Grow. Пространство по главной оси будет выделяться пропорционально значениям свойства, имеющихся у элементов управления.

9. Свойство Shrink указывает, какие дочерние элементы получают приоритет при отображении их полных размеров. Значение свойства по умолчанию - минус 1, но значение свойство должно быть больше или равно 0. Свойство Shrink учитывается, если значение свойства Wrap имеет значение NoWrap, а суммарная ширина строки или столбца с элементами управления больше, чем ширина или высота макета FlexLayout. При значении свойства, равного 0, в макете отображаются полные размеры элемента управления. При значении свойства больше 0 происходит сжатие размеров элементов управления при их отображении на экране устройства.

Для рассмотрения особенностей работы с макетом типа FlexDemo создается проект для мультиплатформенного программного приложения Xamarin.Forms с использованием шаблона «Пустой». Сначала необходимо зайти в диалоговое окно «Обозреватель решений» с помощью рассмотренной ранее последовательности действий создать класс EnumPicker. Далее необходимо открыть созданный файл EnumPicker.cs и набрать программный код Си Шарп, приведенный далее.

В программном коде создается шаблон элемента управления EnumPicker, на который ссылается программный код из файла MainPage.xaml (будет создан позже). Элемент управления EnumPicker является дочерним от элемента управления Picker, используемого для выбора значения из списка.

Создается свойство EnumType, на которое ссылается программный код из файла MainPage.xaml. В программном коде описана работа со свойством EnumType в случае изменения значения в элементе управления EnumPicker, с которым работает пользователь в текущий момент времени.

Свойствам oldValue и newValue элемента управления EnumPicker, связанному со свойствами макета FlexLayout, передаются старое и выбранное новое значения, полученные от одного из элементов управления EnumPicker (эти элементы определены в файле MainPage.xaml и с одним из них пользователь будет работать в текущий момент времени).

Источник данных, из которого поступают значения свойств oldValue и newValue, задаются с помощью свойства ItemsSource объекта picker. В объект picker передаются данные из того элемента управления Picker, который определен в файле MainPage.xaml и с которым пользователь работает в текущий момент времени.

В программном коде с помощью условных операторов производится проверка старых и новых значений, полученных от элементов управления EnumPicker, приведенных в файле MainPage.xaml, в случае изменения значений свойств oldValue и newValue. Также в программном коде объявлено свойство EnumType.

```
using System;
using System.Reflection;
using Xamarin.Forms;
namespace FlexDemo
{  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);
        }
    }
}
```

Далее необходимо открыть файл MainPage.xaml и набрать там программный код XAML, приведенный далее.

В программном коде внутри страницы типа ContentPage содержится макет типа Grid, внутри которого размечена таблица из двух строк. В первой строке располагается макет Grid. Во второй строке располагается макет FlexLayout, к которому с помощью привязки BindingContext привязан макет Grid из первой строки. К элементам управления Label, размещаемым в макете, с помощью элементов Style и Setter устанавливается стиль, при котором расположение элемента управления Label по вертикали производится по центру.

Внутри макета Grid в первой строке расположен другой, внутренний, макет Grid, в котором создаются шесть строк и два столбца.

В первой строке внутреннего макета Grid в первом столбце расположена метка, в которой отображается текст «Количество Label:». Во втором столбце первой строки этого макета расположен макет StackLayout с горизонтальным расположением элементов управления, в качестве которых используются Label и Stepper. При этом элемент управления Label с помощью привязки BindingContext привязан к элементу управления Stepper. Значение, которое выбрано в элементе управления Stepper, будет преобразовано в текстовое значение и отображено в элементе управления Label. Значение, выбранное при помощи элемента управления Stepper, показывает количество элементов управления Label, размещаемых внутри макета FlexLayout. При изменении значения, выбранного с помощью элемента управления Stepper, производится срабатывание обработчика событий OnStepperChanged, который приведен в программном коде Си Шарп в файле MainPage.xaml.cs, который будет создан позже.

Во второй строке внутреннего макета Grid в первом столбце расположена метка, в которой отображается название свойства «Direction:». Во втором столбце второй строки расположен элемент EnumPicker, созданный на основе класса EnumPicker из файла EnumPicker.cs. В данном случае элемент EnumPicker за счет использования Binding привязан к свойству Direction макета FlexLayout.

В третьей строке внутреннего макета Grid в первом столбце расположена метка, в которой отображается название свойства «Wrap:». Во

втором столбце второй строки расположен элемент EnumPicker, который с помощью Binding привязан к свойству Wrap макета FlexLayout.

В четвертой строке внутреннего макета Grid в первом столбце расположена метка, в которой отображается название свойства «JustifyContent:». Во втором столбце второй строки расположен элемент EnumPicker, который с помощью Binding привязан к свойству JustifyContent макета FlexLayout.

В пятой строке внутреннего макета Grid в первом столбце расположена метка, в которой отображается название свойства «AlignItems:». Во втором столбце второй строки расположен элемент EnumPicker, который привязан к свойству AlignItems макета FlexLayout.

В шестой строке внутреннего макета Grid в первом столбце расположена метка, в которой отображается название свойства «AlignContent:». Во втором столбце второй строки расположен элемент EnumPicker, который привязан к свойству AlignContent макета FlexLayout.

С помощью элемента управления EnumPicker отображается список значений того свойства макета FlexLayout, с которым производится в данный момент работа (это свойства Direction, Wrap, JustifyContent, AlignItems, AlignContent).

Таким образом, в данном файле созданы пять элементов управления EnumPicker, предназначенных для отображения значений свойств макета FlexLayout.

При этом, предусмотрено, что количество элементов управления Label, отображаемых в пользовательском интерфейсе при запуске программного приложения, равно трем. Также установлены начальные значения свойств макета FlexLayout, которые будут действовать в момент запуска пограммного приложения.

```
<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" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <!-- Панель управления -->
        <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="Количество Label:"
       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>
        <!—Макет FlexLayout -->
        <FlexLayout x:Name="flexLayout"
                BackgroundColor="AliceBlue"
                Grid.Row="1" />
    </Grid>
</ContentPage>
```

Далее необходимо открыть файл MainPage.xaml.cs и набрать программный код Си Шарп, приведенный далее.

В программном коде задается массив цветов colors для выбора цвета текста в элементах управления Label, размещаемых в макете FlexLayout. Массивы digitsText и decadeText предназначены для отображения текста с номером в элементах управления Label.

Приводится обработчик события OnStepperChanged, на который производится ссылка из программного кода в файле MainPage.xaml. В качестве входного параметра используется количество элементов управления Label, выбранное с помощью элемента управления Stepper с именем numberStepper, который задан ранее в файле MainPage.xaml.

В качестве объекта, на который будет воздействовать обработчик OnStepperChanged является макет FlexLayout. С помощью параметра numberStepper.Value создается значение count, задающее количество элементов управления Label внутри макета FlexLayout. Размещение элементов управления Label организуется с помощью свойства Children и с использованием цикла.

```
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);
        }
    }
}
```

Для демонстрации работы с макетом типа FlexLayout необходимо сначала собрать программное приложение, а затем запустить его. После запуска программного приложения в верхней части пользовательского интерфейса отображается панель управления, а в нижней части – макет FlexLayout. В панели управления отображается элемент управления Stepper, с помощью которого можно изменять количество элементов управления Label, размещаемых в макете FlexLayout. Также в панели управления расположены пять элементов управления EnumPicker, позволяющие задавать значения рассмотренных ранее свойств макета типа FlexLayout. При запуске программного приложения по умолчанию отображается три элемента управления Label. При этом значения свойств макета типа FlexLayout, отображаемого при запуске программного приложения, установлены ранее в программном коде в файле MainPage.xaml. С помощью панели управления производится выбор количества элементов Label, размещаемых внутри макета. Далее за счет выбора значений свойств макета производится управление расположением элементов управления Label на пользовательском интерфейсе.