# Set StartPage

using System.Linq;

using System.Reflection.Emit;

using System.Text;

using Xamarin.Forms;

namespace HelloApp

{

    public class Page1 : ContentPage

    {

        public Page1()

        {

            Content = new StackLayout

            {

                Children = {

                    new Label { Text = "Hello Page" }

                }

            };

        }

    }

}

namespace HelloApp

{

    public partial class App : Application

    {

        public App()

        {

            InitializeComponent();

            MainPage = new Page1();

        }

        //...........

    }

}

# DifficultProperty

Кроме простых свойств, которые могут устанавливаться в с помощью простой строки, например, <Label Text="Hello World" />, в XAML могут применяться сложные или комплексные свойства. В таких случаях свойство класса может принимать в качестве значения какой-нибудь сложный объект, который в свою очередь имеет некоторый набор свойств. Сложные свойства определяются в формате:

|  |  |
| --- | --- |
| 1  2  3 | <Элемент.Свойство>      <Сложный\_объект />  </Элемент.Свойство> |

Например, у класса Label есть свойство Style, которое в качестве значения принимает сложный объект одноименного класса Style:

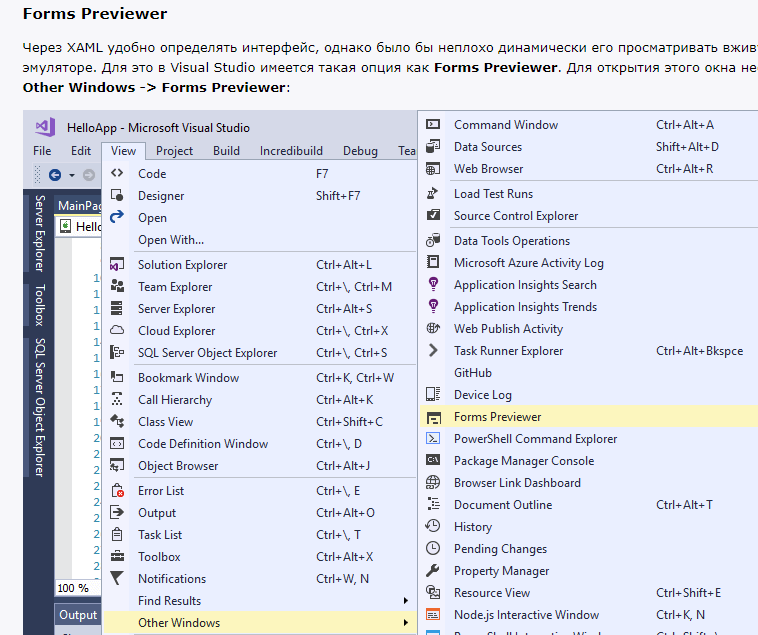
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <Label Text="Hello" HorizontalOptions="Center" VerticalOptions="Center">      <Label.Style>        <Style TargetType="Label">          <Setter Property="TextColor" Value="Red" />          <Setter Property="BackgroundColor" Value="Silver" />        </Style>      </Label.Style>  </Label> |

Элемент Style устанавливает стилевые свойства метки через вложенные элементы Setter.

Также простые свойства можно представить в виде сложных, например вместо VerticalOptions="Center" написать:

|  |  |
| --- | --- |
| 1  2  3 | <Label.VerticalOptions>      <LayoutOptions>Center</LayoutOptions>  </Label.VerticalOptions> |

# FormsPreview



# X:Static

# x:Static

**x:Static** позволяет привязать к атрибуту значение константы или статической переменной. Например, в коде C# определим константу или статическую переменную:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | using Xamarin.Forms;    namespace HelloApp  {      public partial class MainPage : ContentPage      {          public const string HEADER = "Xamarin";            public static double staticVar = 20;            public MainPage()          {              InitializeComponent();            }      }  } |

Тогда мы сможем использовать их значения следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?xml version="1.0" encoding="utf-8" ?>  <ContentPage xmlns="<http://xamarin.com/schemas/2014/forms>"               xmlns:x="<http://schemas.microsoft.com/winfx/2009/xaml>"               xmlns:local="clr-namespace:HelloApp"               x:Class="HelloApp.MainPage">        <Label Text="{x:Static local:MainPage.HEADER}"           FontSize="{x:Static local:MainPage.staticVar}"           VerticalOptions="Center" HorizontalOptions="Center" />  </ContentPage> |

# x:String x:Type

# x:Array и x:Type

Расширение разметки **x:Array** позволяет определить массив данных. А расширение **x:Type** позволяет указать тип данных. Например, воспользуемся элементом ListView для вывода содержимого массива:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <?xml version="1.0" encoding="utf-8" ?>  <ContentPage xmlns="<http://xamarin.com/schemas/2014/forms>"               xmlns:x="<http://schemas.microsoft.com/winfx/2009/xaml>"               xmlns:local="clr-namespace:HelloApp"               x:Class="HelloApp.MainPage">      <ListView>      <ListView.ItemsSource>        <x:Array Type="{x:Type x:String}">          <x:String>iPhone 7</x:String>          <x:String>Samsung Galaxy S7</x:String>          <x:String>HP Elite x3</x:String>        </x:Array>      </ListView.ItemsSource>    </ListView>    </ContentPage> |

В данном случае в определении массива атрибут Type="{x:Type x:String}" указывает, что массив будет содержать данные типа String.

# IMarkupExtension

Иногда может возникнуть необходимость создать какое-то свое расширение для XAML. В этом случае нам надо создать класс, который будет реализовать интерфейс **IMarkupExtension**.

К примеру, мы хотим чтобы через XAML можно было установить для фона элемента цвет из четырех компонент - ARGB. В этом случае мы можем создать следующий класс:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | using System;  using Xamarin.Forms;  using Xamarin.Forms.Xaml;    namespace HelloApp  {      public class ARGBColor : IMarkupExtension      {          public int Alpha { get; set; }          public int Red { get; set; }          public int Green { get; set; }          public int Blue { get; set; }            public object ProvideValue(IServiceProvider serviceProvider)          {              return Color.FromRgba(Red, Green, Blue, Alpha);          }      }  } |

Затем используем этот класс в XAML:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <?xml version="1.0" encoding="utf-8" ?>  <ContentPage xmlns="<http://xamarin.com/schemas/2014/forms>"               xmlns:x="<http://schemas.microsoft.com/winfx/2009/xaml>"               xmlns:local="clr-namespace:HelloApp"               x:Class="HelloApp.MainPage">    <Button Text="Xamarin Forms"           FontSize="Large"           HorizontalOptions="Center"           VerticalOptions="Center"             TextColor="{local:ARGBColor Alpha=255, Red=0, Green=77, Blue=64}"             BackgroundColor="{local:ARGBColor Alpha=255, Red=128, Green=203, Blue=196}">    </Button> |

# StackLayout

Аналогично в XAML мы могли бы написать:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <?xml version="1.0" encoding="utf-8" ?>  <ContentPage xmlns="<http://xamarin.com/schemas/2014/forms>"               xmlns:x="<http://schemas.microsoft.com/winfx/2009/xaml>"               x:Class="HelloApp.MainPage">      <StackLayout x:Name="stackLayout">      <StackLayout.Children>        <Label Text="Первая метка" />        <Label Text="Вторая метка" />      </StackLayout.Children>    </StackLayout>  </ContentPage> |

Либо даже сократить:

|  |  |
| --- | --- |
| 1  2  3  4 | <StackLayout x:Name="stackLayout">      <Label Text="Первая метка" />      <Label Text="Вторая метка" />  </StackLayout> |

Поскольку коллекция Children представляет собой обычный список, то он поддерживает операции по управлению элементами. В частности, мы можем динамически добавить новые элементы. Например:

|  |  |
| --- | --- |
| 1 | stackLayout.Children.Add(new Label { Text = "Третья метка" }); |

Хотя в данном случае коллекция элементов рассматривалась на примере контейнера StackLayout, но тот же принцип работы с элементами будет характерен и для других элементов компоновки.

# ScrollView

# ScrollView

При создании стека или любого другого элемента компоновки может сложиться ситуация, когда не все элементы будут помещаться на экране. В этом случае необходимо создать прокрутку с помощью элемента **ScrollView**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | class MainPage : ContentPage  {      public MainPage()      {          StackLayout stackLayout = new StackLayout();          for (int i = 1; i < 20; i++)          {              Label label = new Label              {                  Text = "Метка " + i,                  FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label))              };              stackLayout.Children.Add(label);          }          ScrollView scrollView = new ScrollView();          scrollView.Content = stackLayout;          this.Content = scrollView;      }  } |

Или в xaml:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?xml version="1.0" encoding="utf-8" ?>  <ContentPage xmlns="<http://xamarin.com/schemas/2014/forms>"                         xmlns:x="<http://schemas.microsoft.com/winfx/2009/xaml>"                         xmlns:local="clr-namespace:HelloApp"                         x:Class="HelloApp.MainPage">  <ScrollView>      <StackLayout>        <Label Text="Метка 1" FontSize="Large" />        <Label Text="Метка 2" FontSize="Large" />        <!--..................................................-->        <Label Text="Метка 20" FontSize="Large" />      </StackLayout>    </ScrollView>  </ContentPage> |

# AbsoluteLayout

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="<http://xamarin.com/schemas/2014/forms>"

             xmlns:x="<http://schemas.microsoft.com/winfx/2009/xaml>"

             x:Class="HelloApp.MainPage">

  <AbsoluteLayout>

    <BoxView Color="Red" AbsoluteLayout.LayoutBounds=".1, .2, 50, 80"

        AbsoluteLayout.LayoutFlags="PositionProportional" />

    <BoxView Color="Green" AbsoluteLayout.LayoutBounds="1, .2, 50, 80"

        AbsoluteLayout.LayoutFlags="PositionProportional" />

    <BoxView Color="Blue" AbsoluteLayout.LayoutBounds=".4, .8, .2, .2"

        AbsoluteLayout.LayoutFlags="All" />

 </AbsoluteLayout>

</ContentPage>

# Grid

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

             xmlns:local="clr-namespace:HelloApp"

             x:Class="HelloApp.MainPage">

  <Grid>

    <Grid.ColumnDefinitions>

      <ColumnDefinition Width="\*" />

      <ColumnDefinition Width="\*" />

    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>

      <RowDefinition Height="\*" />

      <RowDefinition Height="\*" />

    </Grid.RowDefinitions>

    <BoxView Color="Red" Grid.Column="0" Grid.Row="0" />

    <BoxView Color="Blue" Grid.Column="1" Grid.Row="0" />

    <BoxView Color="Yellow" Grid.Column="0" Grid.Row="1" Grid.ColumnSpan="2" />

  </Grid>

</ContentPage>

# Resources

### Определение ресурсов

В качестве ресурса можно определить любой объект. Все ресурсы помещаются в объект **ResourceDictionary**. У каждого визуального объекта, например, ContentPage или Button, имеется свойство **Resources**, которое как раз хранит объект ResourceDictionary. Например, определим несколько ресурсов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <?xml version="1.0" encoding="utf-8" ?>  <ContentPage xmlns="<http://xamarin.com/schemas/2014/forms>"               xmlns:x="<http://schemas.microsoft.com/winfx/2009/xaml>"               x:Class="StylesApp.MainPage">    <ContentPage.Resources>      <ResourceDictionary>        <Color x:Key="textColor">#004D40</Color>        <Color x:Key="backColor">#80CBC4</Color>        <x:String x:Key="fontSize">Large</x:String>      </ResourceDictionary>    </ContentPage.Resources>    <StackLayout>      <Button Text="iOS" TextColor="{StaticResource Key=textColor}"        BackgroundColor="{StaticResource Key=backColor}"  FontSize="{StaticResource Key=fontSize}" />      <Button Text="Android" TextColor="{StaticResource Key=textColor}"        BackgroundColor="{StaticResource Key=backColor}"  FontSize="{StaticResource Key=fontSize}" />    </StackLayout>  </ContentPage> |

Каждый ресурс должен иметь ключ, задаваемый с помощью атрибута x:Key. Это своего рода уникальный идентификатор ресурса. Например:

|  |  |
| --- | --- |
|  | <Color x:Key="textColor">#004D40</Color> |

Чтобы обратиться к этому ресурсу в коде, надо использовать расширение **StaticResource**:

|  |  |
| --- | --- |
|  | TextColor="{StaticResource Key=textColor}" |

# Styles

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="StylesApp.MainPage">

<ContentPage.Resources>

<ResourceDictionary>

<Style x:Key="buttonStyle" TargetType="Button">

<Setter Property="TextColor" Value="#004D40" />

<Setter Property="BackgroundColor" Value="#80CBC4" />

<Setter Property="FontSize" Value="Large" />

</Style>

</ResourceDictionary>

</ContentPage.Resources>

<StackLayout>

<Button Text="iOS" Style="{StaticResource buttonStyle}" />

<Button Text="Android" Style="{StaticResource buttonStyle}" />

</StackLayout>

</ContentPage>

Стиль создается как ресурс с помощью объекта Style и, как любой другой ресурс, он обязательно должен иметь ключ. Атрибут TargetType указывает, к какому типу оносится стиль. В данном случае это тип Button.

С помощью коллекции Setters определяется группа свойств, входящих в стиль. В нее входят объекты Setter, которые имеют следующие свойства:

Property: указывает на свойство, к которому будет применять данный сеттер. При этом свойство должно представлять тип BindableProperty

Value: собственно значение свойства

# Triggers

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="StylesApp.MainPage">

<ContentPage.Resources>

<ResourceDictionary>

<Style x:Key="entryStyle" TargetType="Entry">

<Style.Triggers>

<Trigger Property="Entry.IsFocused" Value="True" TargetType="Entry">

<Setter Property="TextColor" Value="Red" />

</Trigger>

</Style.Triggers>

</Style>

</ResourceDictionary>

</ContentPage.Resources>

<StackLayout>

<Entry FontSize="Large" Style="{StaticResource Key=entryStyle}" />

</StackLayout>

</ContentPage>

Стиль может содержать несколько триггеров, и все они определяются в элементе Style.Triggers.

У каждого триггера устанавливаются три свойства:

Property: свойство, на изменение которого должен отслеживать триггер

Value: значение свойства, при котором должен срабатывать триггер

TargetType: тип объектов, к которым применяется триггер

То есть триггер в данном случае будет срабатывать, когда свойство IsFocused элемента Entry приобритет значение true.

Работа триггера будет заключаться в установки рда свойств элемента Entry. Здесь у Entry устанавливается красный текст:

1

<Setter Property="TextColor" Value="Red" />

Таким образом, при получении фокуса сработает триггер, который окрасит текст в красный цвет.

# BindingContext

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="HelloApp.MainPage">

<StackLayout>

<Label x:Name="label"

BindingContext="{x:Reference Name=entryBox}"

Text="{Binding Path=Text}"

FontSize="Large" />

<Entry x:Name="entryBox" />

</StackLayout>

</ContentPage>

В XAML действует тот же принцип. Для установки привязки для объекта цели задается свойство BindingContext. Но теперь его значение имеет другую форму: {x:Reference Name=entryBox}. После расширения разметки x:Reference идет свойство Name и его значение - имя элемента источника. То есть в данном случае источником выступает объект с именем entryBox.

Далее устанавливается сама привязка к свойству:

1

Text="{Binding Path=Text}"

# BindableProperty

public class TagLabel : Label

{

    public static readonly BindableProperty TagProperty  =

        BindableProperty.Create("Tag", // название обычного свойства

            typeof(string), // возвращаемый тип

            typeof(TagLabel), // тип,  котором объявляется свойство

            "0"// значение по умолчанию

        );

    public string Tag

    {

        set

        {

            SetValue(TagProperty, value);

        }

        get

        {

            return (string)GetValue(TagProperty);

        }

    }

}

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="<http://xamarin.com/schemas/2014/forms>"

             xmlns:x="<http://schemas.microsoft.com/winfx/2009/xaml>"

             xmlns:local="clr-namespace:HelloApp;assembly=HelloApp"

             x:Class="HelloApp.MainPage">

    <StackLayout>

        <local:TagLabel x:Name="tagLabel"

           BindingContext="{x:Reference Name=entryBox}"

           Text="{Binding Path=Text}"

           Tag="{Binding Path=Text}"

           FontSize="Large" />

        <Entry x:Name="entryBox" />

        <Label x:Name="label"

           BindingContext="{x:Reference Name=tagLabel}"

           Text="{Binding Path=Tag}"

           FontSize="Large" />

    </StackLayout>

</ContentPage>

# Binding

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="HelloApp.MainPage">

<StackLayout>

<Entry x:Name="entry1"

BindingContext="{x:Reference Name=entry2}"

Text="{Binding Path=Text}" />

<Entry x:Name="entry2" />

</StackLayout>

</ContentPage>

Так как по умолчанию стандартным значением привязки для свойства Text объекта Entry является TwoWay, то при вводе в первое или второе текстовое поле, все изменения будут автоматически отображаться в другом поле.

Но если нам такое поведение не нужно, и мы хотим использовать именно односороннюю привязку от источника к цели, то мы можем изменить выражение привязки следующим образом:

1

2

<Entry x:Name="entry1" BindingContext="{x:Reference Name=entry2}"

Text="{Binding Path=Text, Mode=OneWay}" />

# Converter

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <ContentPage xmlns="<http://xamarin.com/schemas/2014/forms>"               xmlns:x="<http://schemas.microsoft.com/winfx/2009/xaml>"               xmlns:local="clr-namespace:HelloApp;assembly=HelloApp"               x:Class="HelloApp.MainPage">    <ContentPage.Resources>      <ResourceDictionary>        <local:DateTimeToLocalDateConverter x:Key="dateConverter" />      </ResourceDictionary>    </ContentPage.Resources>    <StackLayout>      <Label Text="{Binding Source={x:Reference Name=datePicker},  Path=Date}" />      <Label Text="{Binding Source={x:Reference Name=datePicker},                          Path=Date,                          Converter={StaticResource dateConverter}}" />      <DatePicker x:Name="datePicker" Format="D"  />    </StackLayout>  </ContentPage> |

Так как класс конвертера располагается в пространстве имен текущего проекта, то, чтобы класс был доступен в xaml, надо подключить текущее пространство имен.

|  |  |
| --- | --- |
|  | xmlns:local="clr-namespace:HelloApp;assembly=HelloApp" |

В моем служит пространством имен является HelloApp, и оно проецируется на суффикс local

Затем в ресурсах создается объект привязки с ключом dateConverter.

И далее в самом выражении привязки мы подключем данный ресурс в качестве конвертера:

|  |  |
| --- | --- |
|  | Converter={StaticResource dateConverter} |

# INotifyPropertyChanged

И чтобы уведомить цель привязки об изменении значений источник привязки должен реализовать интерфейс **INotifyPropertyChanged**. Для этого изменим класс Phone следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | using System.ComponentModel;    namespace HelloApp  {      public class Phone : INotifyPropertyChanged      {          private string title;          private string company;          private int price;            public string Title          {              get { return title; }              set              {                  if (title != value)                  {                      title = value;                      OnPropertyChanged("Title");                  }              }          }          public string Company          {              get { return company; }              set              {                  if (company != value)                  {                      company = value;                      OnPropertyChanged("Company");                  }              }          }          public int Price          {              get { return price; }              set              {                  if (price != value)                  {                      price = value;                      OnPropertyChanged("Price");                  }              }          }            public event PropertyChangedEventHandler PropertyChanged;          public void OnPropertyChanged(string prop = "")          {              if (PropertyChanged != null)                  PropertyChanged(this, new PropertyChangedEventArgs(prop));          }      }  } |

При этом в коде самой страницы MainPage ничего менять не надо. И если теперь мы нажмем на кнопку, то элемент Label мгновенно отреагирует на изменение свойства Price объекта Phone.

Аналогичный пример в XAML:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | <?xml version="1.0" encoding="utf-8" ?>  <ContentPage xmlns="<http://xamarin.com/schemas/2014/forms>"               xmlns:x="<http://schemas.microsoft.com/winfx/2009/xaml>"               xmlns:local="clr-namespace:HelloApp;assembly=HelloApp"               x:Class="HelloApp.MainPage">    <ContentPage.Resources>      <ResourceDictionary>        <local:Phone x:Key="phone" Title="iPhone 7" Company ="Apple" Price="56000" />      </ResourceDictionary>    </ContentPage.Resources>    <Grid BindingContext="{StaticResource phone}">      <Grid.ColumnDefinitions>        <ColumnDefinition Width="0.8\*" />        <ColumnDefinition Width="1.1\*" />        <ColumnDefinition Width="1.1\*" />      </Grid.ColumnDefinitions>      <Grid.RowDefinitions>        <RowDefinition Height="50" />        <RowDefinition Height="50" />        <RowDefinition Height="50" />      </Grid.RowDefinitions>      <Label Grid.Column="0" Grid.Row="0" Text="Модель" />      <Label Grid.Column="1" Grid.Row="0" Text="Компания"  />      <Label Grid.Column="2" Grid.Row="0" Text="Цена"   />        <Label Grid.Column="0" Grid.Row="1" Text="{Binding Path=Title}" />      <Label Grid.Column="1" Grid.Row="1" Text="{Binding Path=Company}"  />      <Label Grid.Column="2" Grid.Row="1" Text="{Binding Path=Price}"   />        <Button Text="Обновить" Grid.Column="2" Grid.Row="2" Clicked="UpdateButton\_Clicked" />    </Grid>  </ContentPage> |

Здесь объект Phone определяется как статический ресурс. Затем мы можем задать его как контекст для всего элемента Grid, а в отдельных элементах Label прописать привязку к свойствам этого объекта.

А в классе связанного кода пропишем обработчик кнопки, который будет менять значение свойства Price:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | using System;  using Xamarin.Forms;    namespace HelloApp  {      public partial class MainPage : ContentPage      {          public MainPage()          {              InitializeComponent();          }            private void UpdateButton\_Clicked(object sender, EventArgs e)          {              var phone = this.Resources["phone"] as Phone;              phone.Price += 4000;          }      }  } |