Добавление событий

Последнее обновление: 02.10.2016

В прошлой теме в элемент HeaderView были добавлены свойства. Теперь добавим также и события. В частности, событие нажатия.

Для вызова события нам необязательно писать какой-то код в самом элементе. В нем мы можем только определить нужные события, а затем их генерировать в рендерере.

Так, определим в классе HeaderView событие `TappedOrClickEvent`:

```
using System;
using Xamarin.Forms;

namespace CustomRendererApp
{
    public class HeaderView : View
    {
        public event EventHandler TappedOrClickEvent;
        public void FireClick(EventArgs e)
        {
            if (this.TappedOrClickEvent != null)
                this.TappedOrClickEvent(this, e);
        }

        public static readonly BindableProperty TextProperty =
            BindableProperty.Create("Text", typeof(string), typeof(HeaderView), string.Empty);
        public string Text
        {
            set
            {
                SetValue(TextProperty, value);
            }
            get
            {
                return (string)GetValue(TextProperty);
            }
        }

        public static readonly BindableProperty TextColorProperty =
            BindableProperty.Create("TextColor", typeof(Color), typeof(HeaderView),
Color.Default);
        public Color TextColor
```

```
        {
            set
            {
                SetValue(TextColorProperty, value);
            }
            get
            {
                return (Color)GetValue(TextColorProperty);
            }
        }
    }
}
```

На странице используем это событие:

```
<?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:CustomRendererApp;assembly=CustomRendererApp"
             x:Class="CustomRendererApp.MainPage">
  <StackLayout>
    <local:HeaderView x:Name="headerView1" Text="Simple HeaderView"
                      TappedOrClickEvent="ChangeText"></local:HeaderView>
  </StackLayout>
</ContentPage>
```

А в файле кода страницы пропишем обработчик:

```
using System;
using Xamarin.Forms;

namespace CustomRendererApp
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();

        }
        int clicks = 0;
        private void ChangeText(object sender, EventArgs e)
        {
            HeaderView hView = sender as HeaderView;
            hView.Text = $"{++clicks} clicks";
        }
    }
}
```

Теперь изменим рендерер для Android:

```
using Xamarin.Forms;
using Xamarin.Forms.Platform.Android;
using Android.Util;
using Android.Widget;
using CustomRendererApp;
using CustomRendererApp.Droid;
using System.ComponentModel;

[assembly: ExportRenderer(typeof(HeaderView), typeof(HeaderViewRenderer))]
namespace CustomRendererApp.Droid
{
    public class HeaderViewRenderer : ViewRenderer<HeaderView, TextView>
    {
        protected override void OnElementChanged(ElementChangedEventArgs<HeaderView> args)
```

```
        {
            base.OnElementChanged(args);
            if (Control == null)
            {
                // создаем и настраиваем элемент
                TextView textView = new TextView(Context);
                textView.SetTextSize(ComplexUnitType.Dip, 28);
                textView.Touch += TextView_Touch;
                // устанавливаем элемент для класса из Portable-проекта
                SetNativeControl(textView);
                // установка свойств
                if (args.NewElement != null)
                {
                    SetText();
                    SetTextColor();
                }
            }
        }

        private void TextView_Touch(object sender, TouchEventArgs e)
        {
            if (Element != null)
                Element.FireClick(System.EventArgs.Empty);
        }

        // изменения свойства
        protected override void OnElementPropertyChanged(object sender,
PropertyChangedEventArgs e)
        {
            base.OnElementPropertyChanged(sender, e);

            if (e.PropertyName == HeaderView.TextColorProperty.PropertyName)
            {
                SetTextColor();
            }
            else if (e.PropertyName == HeaderView.TextProperty.PropertyName)
            {
                SetText();
            }
        }
        private void SetText()
        {
            Control.Text = Element.Text;
        }
        private void SetTextColor()
        {
            Android.Graphics.Color andrColor = Android.Graphics.Color.Gray;

            if (Element.TextColor != Xamarin.Forms.Color.Default)
            {
                Xamarin.Forms.Color color = Element.TextColor;
                andrColor = Android.Graphics.Color.Argb(
                    (byte)(color.A * 255),
                    (byte)(color.R * 255),
                    (byte)(color.G * 255),
                    (byte)(color.B * 255));
            }
            Control.SetTextColor(andrColor);
        }
    }
}
```

Ключевым моментом для обработки события здесь является обработка события Touch нативного элемента TextView. В его обработчике по сути мы переадресуем вызовы элементу HeaderView:

```
Element.FireClick(System.EventArgs.Empty);
```

Это общая схема обработки событий, которая работает вне зависимости от события элемента: в обработчиках событий нативного элемента мы генерируем события элемента из Xamarin.Forms.

Также изменим код рендерера для iOS:

```
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;
using UIKit;
using CustomRendererApp;
using CustomRendererApp.iOS;
using System.ComponentModel;

[assembly: ExportRenderer(typeof(HeaderView), typeof(HeaderViewRenderer))]
namespace CustomRendererApp.iOS
{
    public class HeaderViewRenderer : ViewRenderer<HeaderView, UILabel>
    {
        UITapGestureRecognizer tapGestureRecognizer;
        protected override void OnElementChanged(ElementChangedEventArgs<HeaderView> args)
        {
            base.OnElementChanged(args);
            if (Control == null)
            {
                UILabel uilabel = new UILabel
                {
                    Font = UIFont.SystemFontOfSize(25)
                };
                tapGestureRecognizer = new UITapGestureRecognizer(() => { OnHeaderViewTapped();
});
                uilabel.AddGestureRecognizer(tapGestureRecognizer);
                SetNativeControl(uilabel);
            }
            if (args.NewElement != null)
            {
                SetText();
                SetTextColor();
            }
        }

        private void OnHeaderViewTapped()
        {
            if(Element!=null)
            {
                Element.FireClick(System.EventArgs.Empty);
            }
        }

        protected override void OnElementPropertyChanged(object sender,
PropertyChangedEventArgs e)
        {
            base.OnElementPropertyChanged(sender, e);

            if (e.PropertyName == HeaderView.TextColorProperty.PropertyName)
            {
                SetTextColor();
            }
            else if (e.PropertyName == HeaderView.TextProperty.PropertyName)
            {
                SetText();
            }
        }
        private void SetText()
        {
```

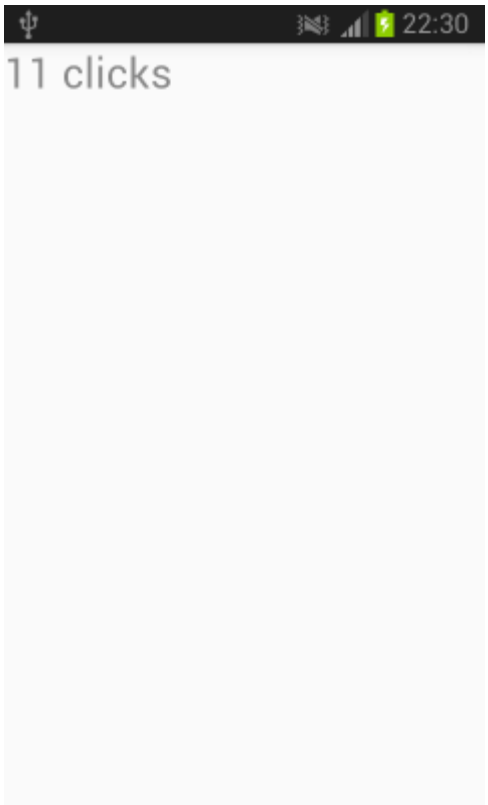
```
            Control.Text = Element.Text;
        }
        private void SetTextColor()
        {
            UIColor iosColor = UIColor.Gray;

            if (Element.TextColor != Xamarin.Forms.Color.Default)
            {
                Xamarin.Forms.Color color = Element.TextColor;
                iosColor = UIColor.FromRGBA(
                    (byte)(color.R * 255),
                    (byte)(color.G * 255),
                    (byte)(color.B * 255),
                    (byte)(color.A * 255));
            }
            Control.TextColor = iosColor;
        }
    }
}
```

Здесь для обработки нажатия используется класс **UITapGestureRecognizer**.

И изменим код рендерера для UWP:

```
using Xamarin.Forms.Platform.UWP;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media;
using CustomRendererApp;
using CustomRendererApp.UWP;
using System.ComponentModel;

[assembly: ExportRenderer(typeof(HeaderView), typeof(HeaderViewRenderer))]
namespace CustomRendererApp.UWP
{
    public class HeaderViewRenderer : ViewRenderer<HeaderView, TextBlock>
    {
        protected override void OnElementChanged(ElementChangedEventArgs<HeaderView> args)
        {
            base.OnElementChanged(args);
            if (Control == null)
            {
                TextBlock textBlock = new TextBlock
                {
                    FontSize = 28
                };
                textBlock.Tapped += TextBlock_Tapped;
                SetNativeControl(textBlock);
                if (args.NewElement != null)
                {
                    SetText();
                    SetTextColor();
                }
            }
        }

        private void TextBlock_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
        {
            if(Element!=null)
            {
                Element.FireClick(System.EventArgs.Empty);
            }
        }
```

```
        protected override void OnElementPropertyChanged(object sender,
PropertyChangedEventArgs e)
        {
            base.OnElementPropertyChanged(sender, e);
            if (e.PropertyName == HeaderView.TextColorProperty.PropertyName)
            {
                SetTextColor();
            }
            else if (e.PropertyName == HeaderView.TextProperty.PropertyName)
            {
                SetText();
            }
        }

        private void SetText()
        {
            Control.Text = Element.Text;
        }
        private void SetTextColor()
        {
            Windows.UI.Color winColor = Windows.UI.Colors.Black;

            if (Element.TextColor != Xamarin.Forms.Color.Default)
            {
                Xamarin.Forms.Color color = Element.TextColor;
                winColor = Windows.UI.Color.FromArgb(
                    (byte)(color.A * 255),
                    (byte)(color.R * 255),
                    (byte)(color.G * 255),
                    (byte)(color.B * 255));
            }
            Control.Foreground = new SolidColorBrush(winColor);
        }
    }
}
```



Назад **Содержание** **Вперед**

Ярослав Орлов • год назад
Добрый вечер. Разложите мне, пожалуйста, по полочкам в чем отличие данного раздела про HeaderView и предыдущего про создание ContentView?
Ведь по факту в предыдущем разделе мы могли реализовать свой контрол с своими свойствами и событиями.
• Ответить • Поделиться ›

Metanit Модератор ➚ Ярослав Орлов • год назад
да, можно реализовать свой контроллл различными способами. Но в ContentView там использовались уже готовые элементы, здесь же создаются новые элементы почти с нуля, ну и во-вторых, здесь больший акцент идет на рендереры, а не на собственно создание элементов
• Ответить • Поделиться ›

Ярослав Орлов ➚ Metanit • год назад

Так какова роль рендеров если по факту мы в них описуем ту же самую логику? С нуля то писать круто, но там мы тоже пишем с нуля накидывая разные кнопки и поля, только выглядит эстетичней чем сдесь, я не до конца понимаю смысл всей массы этого когда в рендерах? Результат абсолютно один и тот же. Если можно, напишите плюсы и минусы обоих способов, может это прольет свет на ситуацию. Взять тот же HeaderView при помощи рендера мы всеровно его связали с TextView, точно так же могли бы на готовую XAML страничку кинуть его и уже манипулировать им.

^ v • Ответить • Поделиться ›

Metanit Модератор ➚ Ярослав Орлов • год назад

а если вам надо создать элемент, который отличается от стандартных? Типа эллипса, морской звезды и т.д., которого нет в стандартной библиотеке?

^ v • Ответить • Поделиться ›

Ярослав Орлов ➚ Metanit • год назад

А если при помощи XAML мы нарисуем скажем бордер любой формы, то мы не сможем с инициировать на нем события наследовав его от какого-то класса типа Control? Что бы у фигуры данной формы были такие события как Click, Move, Enter и т.д.?

^ v • Ответить • Поделиться ›

Metanit Модератор ➚ Ярослав Орлов • год назад

от какого бы вы не наследовали элемента, вам все равно надо как-то отрисовывать этот элемент, рендерить. Для этого и надо создавать рендерер. А бордер любой формы в xaml - не xamarin отвечает за отрисовку элементов, а нативные платформы. Без них вы никакой бордер любой формы не сделаете

^ v • Ответить • Поделиться ›

Ярослав Орлов ➚ Metanit • год назад

Вот я создам по умолчанию ContentView Form, на этой разметке при помощи графических возможностей сделаю ту же звездочку и уже в классе C# опишу события и все остальное к этому нарисованому объекту. Ведь в рендерере мы так же связывали HeaderView с TextView. Или возможности XAML в Xamarin меньше чем в WPF?

^ v • Ответить • Поделиться ›

Metanit Модератор ➚ Ярослав Орлов • год назад

пример с HeaderView - это просто показательный пример в упрощенном виде. Но контроллы могут быть и значительно сложнее. Вы не понимаете, что Xamarin Forms в визуальной части - это просто надстройка над нативными платформами.
Вы сначала создайте эту звездочку...

^ v • Ответить • Поделиться ›

Ярослав Орлов ➚ Metanit • год назад

Спасибо большое за доступный ответ, сделал для себя пометки и все понял. Спасибо за Ваш безумнно полезный ресурс и Ваш труд.

 • Ответить • Поделиться ›

ТАКЖЕ НА **METANIT.COM**

Django | Статические файлы

1 комментарий • 21 день назад

Аватар **Pavel Korolecky** — Отличный, а самое главное — очень актуальный материал! По свежей Django колоссально мало

Go | LiteIDE

1 комментарий • 3 месяца назад

Аватар **Jonger Sender** — спасибо