



## Добавление свойств

Последнее обновление: 02.10.2016



В прошлой теме был создан новый элемент HeaderView. Но на данный момент он выглядит слишком просто:

```
public class HeaderView : View
{
}
```

В нем пока нет никакой функциональности. Теперь изменим его, добавив пару свойств:

```
using Xamarin.Forms;

namespace CustomRendererApp
{
    public class HeaderView : View
    {
        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);
            }
        }
    }
}
```

```

    }
}
}

```

Класс определяет два свойства: Text и TextColor. Фактически данные свойства выступают в качестве обертки над [BindableProperty](#). Тем самым мы сможем использовать свойства в механизме привязки.

На странице определим пару элементов HeaderView:

```

<?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 Text="Simple HeaderView"></local:HeaderView>
        <local:HeaderView Text="Colored HeaderView" TextColor="Red"></local:HeaderView>
    </StackLayout>
</ContentPage>

```

Затем нам надо изменить код рендереров для каждой платформы, чтобы через свойства в HeaderView можно было установить параметры визуализации.

Изменим код рендера для Android следующим образом:

```

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);

                // устанавливаем элемент для класса из Portable-проекта
                SetNativeControl(textView);
                // установка свойств
                if (args.NewElement != null)
                {
                    SetText();
                    SetTextColor();
                }
            }
        }
        // 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);
    }
}

```

Во время вызова метода `OnElementChanged()` ожидается, что объект из `Xamarin.Forms` (в нашем случае `HeaderView`) уже создан, а его свойства установлены. Хотя как правило так и происходит, но это не обязательно должно происходить. С помощью передаваемого в качестве параметра объект `ElementChangedEventArgs` мы можем получить тот элемент `Xamarin.Forms`, для которого создается нативный объект `TextView`. Если свойство `NewElement` не равно значению `null`, то значит объект `HeaderView` создан, а его свойства установлены. В этом случае свойство `Element` из `ViewRenderer` ссылается на тот же объект, что и `NewElement`. И мы можем передать значения его свойств в создаваемый нативный объект.

Для установки значений здесь используются два дополнительных метода `SetText()` и `SetTextColor()`.

При использовании свойств важно, чтобы при изменении их значений у объекта `Xamarin Forms` автоматически менялись также и значения свойств нативного объекта.

Если свойство элемента из `Xamarin.Forms` представляет `BindableProperty`, то любое его изменение вызовет событие `PropertyChanged`. Поэтому рендерер также будет уведомлен об изменении, и в этом случае у `ViewRenderer` будет вызван метод **`OnElementPropertyChanged()`**.

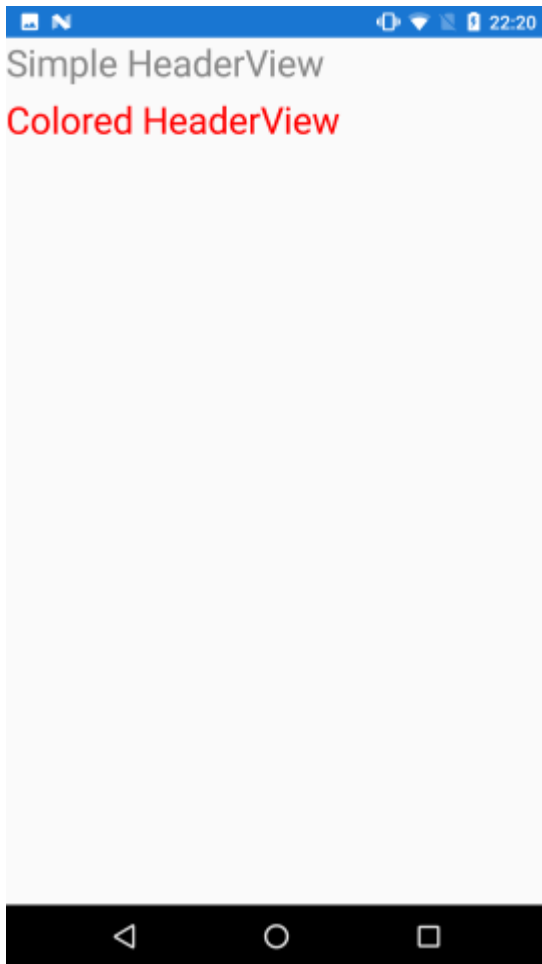
В методе `OnElementPropertyChanged()` с помощью передаваемого параметра `PropertyChangedEventArgs` мы можем получить изменяемое свойство:

```

if (e.PropertyName == HeaderView.TextColorProperty.PropertyName)
{
    //.....
}

```

Если мы запустим приложение, то к обоим элементам `HeaderView` будут применены те свойства, которые мы для них указали:



Далее изменим реализацию рендерера в проекте 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>
    {
        protected override void OnElementChanged(ElementChangedEventArgs<HeaderView> args)
        {
            base.OnElementChanged(args);
            if (Control == null)
            {
                UILabel uilabel = new UILabel
                {
                    Font = UIFont.SystemFontOfSize(25)
                };
                SetNativeControl(uilabel);
            }
            if (args.NewElement != null)
            {
                SetText();
                SetTextColor();
            }
        }
        protected override void OnElementPropertyChanged(object sender,
            PropertyChangedEventArgs e)
        {
            if (e.PropertyName == HeaderView.TextProperty.PropertyName)
            {
                SetText();
            }
            if (e.PropertyName == HeaderView.TextColorProperty.PropertyName)
            {
                SetTextColor();
            }
        }
    }
}
```

```

    {
        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 аналогичный код, отличаются только конкретные классы, используемые при рендеринге.

И также изменим код рендера 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
                };

                SetNativeControl(textBlock);
                if (args.NewElement != null)
                {
                    SetText();
                    SetTextColor();
                }
            }
        }
    }
}

```

```
    }  
    }  
}  
  
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);  
}  
}
```

Таким образом, мы можем определить в кастомном элементе свойства и затем использовать их для рендеринга этого элемента.



Яндекс.Директ

Сильный мороз в Минской области?

yandex.by

[Назад](#) [Содержание](#) [Вперед](#)[Sign up for free](#)

1 Комментарий metanit.com

1 Войти ▾

Рекомендовать Поделиться

Лучшее в начале ▾



Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ

ИЛИ ЧЕРЕЗ DISQUS

Имя

**Andrey Lysenko** • 2 месяца назад

В каком месте к PropertyChanged нового элемента происходит привязка обработчика OnElementPropertyChanged? за кулисами автоматически при регистрации bindable свойства?:

```
public static readonly BindableProperty TextProperty =
    BindableProperty.Create("Text", typeof(string), typeof(HeaderView), string.Empty);
```

^ | ▾ • Ответить • Поделиться ›

ТАКЖЕ НА METANIT.COM

## Django | Параметры представлений

2 комментария • 19 дней назад

**AP** — С чего это взял? страницы оканчиваются на 3.3.php

## Kotlin | Переменные

2 комментария • 3 месяца назад

**Иван Китаев** — Подскажите, как мы можем ввести переменные с клавиатуры?String Вводится через функцию readLine(), а другие

## Go | Соответствие интерфейсу

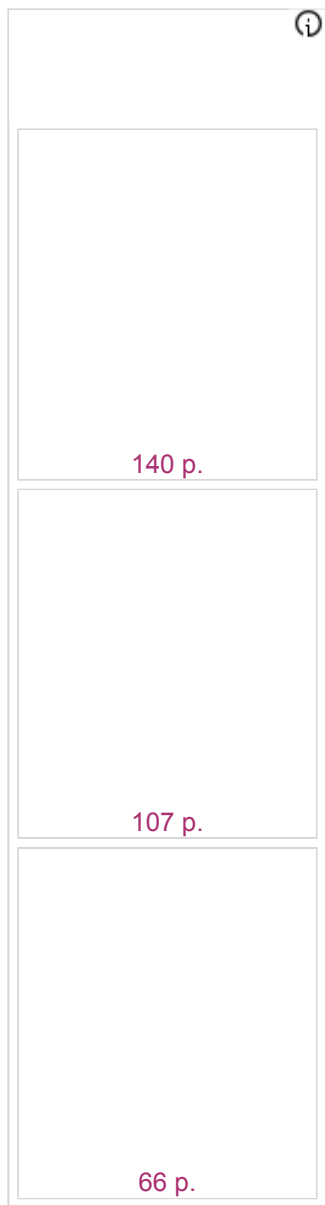
5 комментариев • 2 месяца назад

**Metanit** — менять необязательно

## Go | Visual Studio Code

1 комментарий • 3 месяца назад

**Алексей Сентюрин** — Для полноты не добавьте пару строк про дебаггинг?) Подписаться Добавить Disqus на свой сайт [Добавить Disqus](#) [Добавить](#) Конфиденциальность



140 p.

107 p.

66 p.

[Вконтакте](#) | [Twitter](#) | [Google+](#) | [Канал сайта на youtube](#) | [Помощь сайту](#)

Контакты для связи: metanit22@mail.ru

Copyright © metanit.com, 2012-2017. Все права защищены.