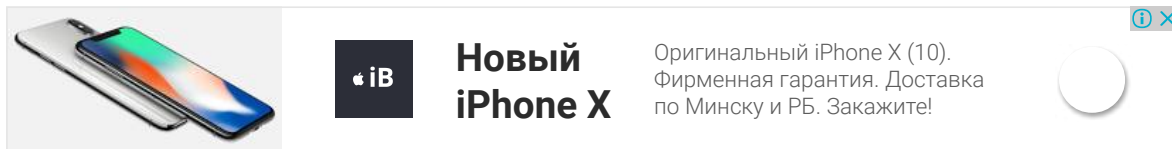




Триггеры

Последнее обновление: 13.08.2016



Триггеры в Xamarin Forms позволяют декларативно задать некоторые действия, которые выполняются при изменении свойств визуального объекта.

Триггеры свойств

Простые триггеры свойств определяются как элементы стиля с помощью объекта Trigger. Они следят за значением свойств и в случае их изменения с помощью объекта Setter устанавливают значение других свойств.

Например, пусть по переходу к текстовому полю Entry текст в нем приобретает красный цвет:

```
<?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:

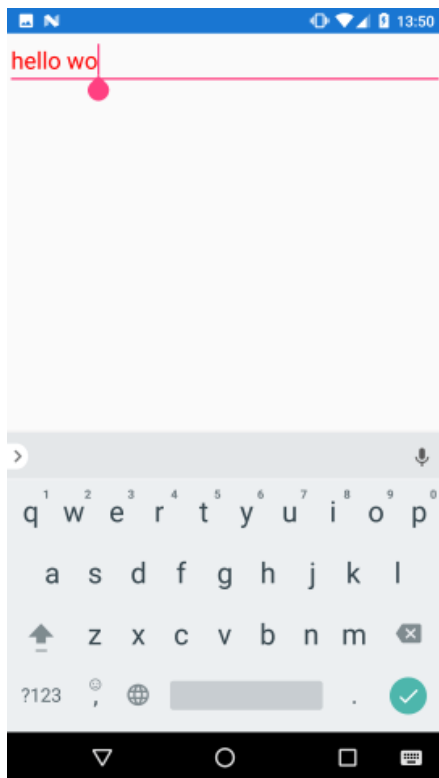
- **Property:** свойство, на изменение которого должен отслеживать триггер
- **Value:** значение свойства, при котором должен срабатывать триггер
- **TargetType:** тип объектов, к которым применяется триггер

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

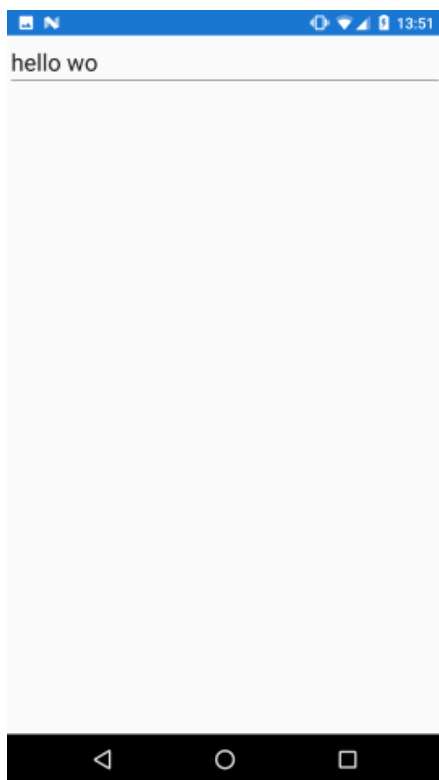
Работа триггера будет заключаться в установке ряда свойств элемента Entry. Здесь у Entry устанавливается красный текст:

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

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



Зато, когда мы установим фокус в другом месте приложения на какой-то другой элемент, то триггер уже действовать не будет, а текст в Entry приобретет свой стандартный цвет:



Триггеры свойств в коде C#

Определим триггеры свойств в коде C#:

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        Entry entry = new Entry();

        // определяем триггер для элемента Entry
        var trigger = new Trigger(typeof(Entry))
        {
```

```

        Property = Entry.IsFocusedProperty,
        Value = true
    };
    // установка зеленого фона
    trigger.Setters.Add(new Setter
    {
        Property = Entry.BackgroundColorProperty,
        Value = Color.Green
    });
    // установка белого цвета текста
    trigger.Setters.Add(new Setter
    {
        Property = Entry.TextColorProperty,
        Value = Color.White
    });
    // добавляем триггер
    entry.Triggers.Add(trigger);

    Content = new StackLayout
    {
        Children = { entry }
    };
}
}

```

Вначале определяем триггер с помощью объекта `Trigger`. Через свойство `Property` указываем свойства элемента `Entry`, которое будет отслеживаться. А с помощью свойства `Value` устанавливаем значение, при получении которого сработает триггер.

```

var trigger = new Trigger(typeof(Entry))
{
    Property = Entry.IsFocusedProperty,
    Value = true
};

```

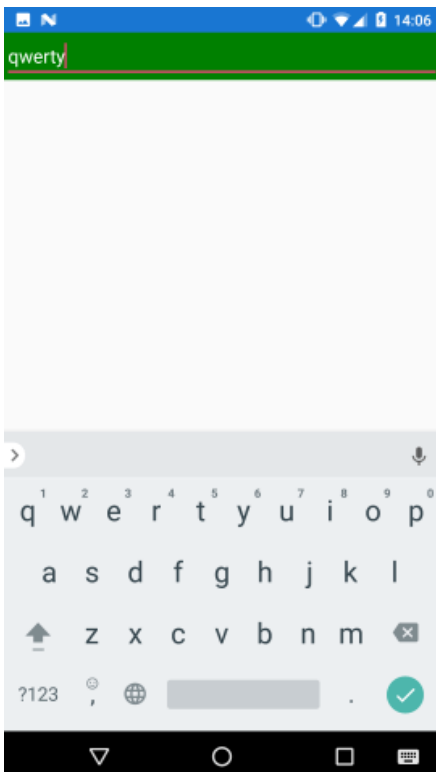
То есть когда свойство `IsFocused` получит значение `true`, сработает триггер.

Далее через коллекцию `Setters` определяем свойства и их значения, которые будут устанавливаться во время действия триггера - установка зеленого цвета фона и белого цвета для введенных символов.

В конце триггер добавляется в коллекцию `Triggers` элемента `Entry`:

```
entry.Triggers.Add(trigger);
```

В итоге при получении фокуса тестовое поле будет окрашиваться в зеленый цвет, а введенные символы - в белый:



Но пойдем чуть дальше и добавим к триггеру дополнительные действия. Действие в триггере - это обычный класс, который наследуется от класса **TriggerAction** и который реализует метод `Invoke()`. Так, определим следующий класс:

```
public class FocusTriggerAction : TriggerAction<Entry>
{
    protected override void Invoke(Entry sender)
    {
        if (sender.IsFocused)
            sender.FadeTo(1);
        else
            sender.FadeTo(0.5);
    }
}
```

Поскольку это действие будет применяться к элементу Entry, то базовый класс типизируется типом Entry. В метод Invoke в качестве параметра передается элемент Entry, к которому применяется триггер. В методе Invoke проверяем, имеет ли текстовое поле фокус, и если имеет, то делаем его непрозрачным, иначе делаем его наполовину прозрачным.

Теперь используем это действие:

```
public class MainPage : ContentPage
{
    public MainPage()
    {
        Entry entry = new Entry();

        var trigger = new Trigger(typeof(Entry))
        {
            Property = Entry.IsFocusedProperty,
            Value = true
        };
        trigger.Setters.Add(new Setter
        {
            Property = Entry.BackgroundColorProperty,
            Value = Color.Green
        });
        trigger.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Color.White
        });
        trigger.EnterActions.Add(new FocusTriggerAction());
        trigger.ExitActions.Add(new FocusTriggerAction());

        entry.Triggers.Add(trigger);

        Content = new StackLayout
        {
            Children = { entry }
        };
    }
}
```

Класс Trigger определяет два специальных свойства: **EnterActions** (хранит действия, которые применяются при срабатывании триггера) и **ExitActions** (хранит действия, которые выполняются, когда триггер перестает действовать). Причем в обе эти коллекции-свойства передается FocusTriggerAction, что позволит включить действие при включении триггера и отключать при отключении триггера.

Триггеры событий

Триггеры событий вызываются в ответ на события элемента. Но чтобы использовать триггер событий, нам надо вначале создать действие, которое будет вызываться триггером. .

Итак, добавим в наш проект следующий класс:

```
public class EntryValidation : TriggerAction<Entry>
{
    protected override void Invoke(Entry sender)
    {
        int number;
        if (!Int32.TryParse(sender.Text, out number))
            sender.BackgroundColor = Color.Red;
        else
            sender.BackgroundColor = Color.Default;
    }
}
```

В данном случае мы будем ожидать ввод только цифр в поле Entry. Если же будут введены нецифровые символы, то поле окрасится в красный цвет.

Теперь применим это действие:

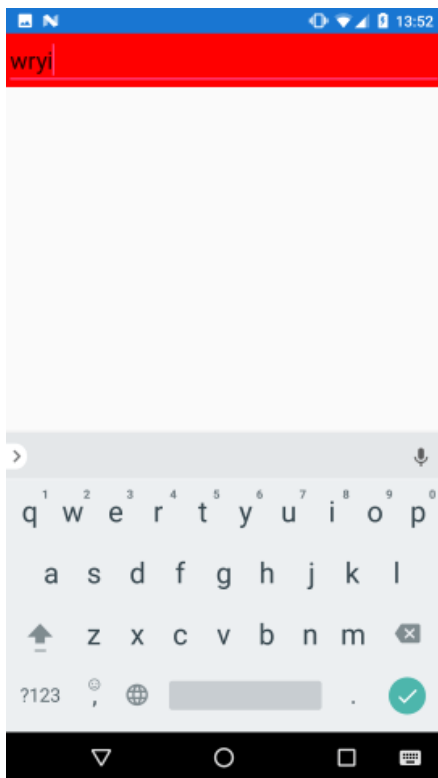
```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:StylesApp;assembly:StylesApp"
  x:Class="StylesApp.MainPage">
  <ContentPage.Resources>
    <ResourceDictionary>
      <Style x:Key="entryStyle" TargetType="Entry">
        <Style.Triggers>
          <EventTrigger Event="TextChanged">
            <local:EntryValidation />
          </EventTrigger>
        </Style.Triggers>
      </Style>
    </ResourceDictionary>
  </ContentPage.Resources>
  <StackLayout>
    <Entry FontSize="Large" Style="{StaticResource Key=entryStyle}" />
  </StackLayout>
</ContentPage>
```

Триггер событий также определяется в элементе `Style.Triggers`, только теперь он представлен объектом **EventTrigger**.

Атрибут `Event` этого объекта указывает на событие, при возникновении которого будет вызываться триггер. В данном случае это событие `TextChanged`, то есть изменение текста в поле `Entry`.

И в самом `EventTrigger` определяется наше действие `EntryValidation`.

В итоге, если мы введем в поле нецифровые символы, то это поле окрасится в красный цвет:



Триггеры событий в коде

У визуальных элементов есть свойство **Triggers**, которое мы можем использовать для добавления триггера:

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        Entry entry = new Entry();
        var trigger = new EventTrigger()
        {
            Event = "TextChanged"
        };
        trigger.Actions.Add(new EntryValidation());
        entry.Triggers.Add(trigger);

        Content = new StackLayout
        {
```

```
        Children = { entry }  
    };  
}  
}
```

Вначале собственно создается триггер - объект `EventTrigger`, и с помощью свойства `Event` у него устанавливается событие.

Далее в коллекцию `Actions` в триггере добавляется объект `EntryValidation`, а затем сам триггер добавляется в коллекцию `Triggers` у элемента `Entry`.

Результат работы будет аналогичен действию кода в `xaml`.

CUBA Platform - Open Source Java Web Framework

Design UI and data model visually, develop in any Java IDE cuba-platform.com



[Назад](#) [Содержание](#) [Вперед](#)



Тачка DGM GT-1081 ▾

21vek.by



Яндекс.Директ

Кабель ВВГНГ-LS здесь! ▾

 spectrinvest.by



Яндекс.Директ

1 Комментарий metanit.com

1 Войти ▾

 Рекомендовать  Поделиться

Лучшее в начале ▾



Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ

ИЛИ ЧЕРЕЗ DISQUS **Andrey Lysenko** • месяц назад



Почему в триггеры свойств вручную надо добавлять EnterTriggerAction и Exit... если они требуются с конструкторами по-умолчанию в триггерах свойств, где логика? в XAML все лаконично и закулисами интерпретатор с компилятором все делают без этих действий, зачем же в коде прописывать бессмысленные шаблонные строчки. Мое мнение что триггерами свойств можно управлять через эти два свойства-действие? верно ли что можно добавлять всякие примочки через них, может анимацию к примеру?

  • [Ответить](#) • [Поделиться](#) >

ТАКЖЕ НА METANIT.COM



Vue.js | Локальные и глобальные фильтры

1 комментарий • 3 месяца назад

 **eugene81** — Можно ли создать фильтр для фильтрации массивов?




C# и .NET | Соккрытие методов

3 комментариев • 2 месяца назад

 **Везнич** — пример, чтоб видно разницу между new и
 `override class Water { public virtual void Drink() {
Console.WriteLine("water life"); } } class Tea : Water { public`



Angular в ASP.NET Core | Маршрутизация

2 комментариев • 2 месяца назад

 **Zaur YakubOff** — Если создал проект на основе
 `Asc.NetCore MVC` куда нужно вставить `<base href=""/>`
???

C++ | Указатель на функцию как возвращаемое значение

1 комментарий • 4 месяца назад

 **Даниил Данилов** — Не знал что можно декораторы
 делать как JS Подписаться  Добавить Disqus на свой сайт [Добавить Disqus](#) [Добавить](#)  Конфиденциальность



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

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