BindableObject и BindableProperty

Последнее обновление: 16.06.2016

Для поддержки привязки данных Xamarin Forms определяет класс **BindableObject**. Все остальные визуальные элементы, как кнопки, метки, текстовые поля, контейнеры компоновки и так далее наследуются от этого класса.

Отличительной особенностью класса BindableObject является то, что он содержит специальные типы свойств **BindableProperty**. обычные свойства по сути представляют обертку над BindableProperty. В .NET есть похожая концепция - свойства зависимости (dependency property), которые имеют похожее назначение.

Например, в Xamarin Forms есть класс Label, который является потомком класса BindableObject и у которого есть свойство `Text`. Через это свойство мы можем установить текст метки. Но в реальности это свойство выглядит следующим образом:

```
public static readonly BindableProperty TextProperty =
    BindableProperty.Create("Text", typeof(string), typeof(Label),
                default(string), propertyChanged: OnTextPropertyChanged);

public string Text
{
    get { return (string)GetValue(TextProperty); }
    set { SetValue(TextProperty, value); }
}
```

Как правило, для каждого свойства BindableProperty создается обертка-обычное свойство. И название BindableProperty обычно имеет название обычного свойства + суффикс `Property`. Например, Text и TextProperty или TextColor и TextColorProperty.

Поэтому, если, допустим, у нас есть элемент Label, и мы хотим присвоить ему некоторый текст, то мы можем сдеать это двумя способами:

```
Label label = new Label();
// 1 способ - обычное свойство
label.Text = "Hello";
// 2 способ - через BindableProperty
label.SetValue(Label.TextProperty, "Hello Xamarin");
```

Для установки значения свойства через BindableProperty у объекта BindableObject вызывается метод `SetValue()`. В качестве первого параметра в метод передается само свойство (то есть в данном случае Label.TextProperty). Второй параметр передает значение для этого свойства.

Аналогично для получения значения свойства мы также можем применять два способа:

```
// 1 способ - через обычное свойство
string text = label.Text;
// 2 способ - через BindableProperty
text = label.GetValue(Label.TextProperty);
```

Таким образом, определяются BindableObject и BindableProperty.

Создание свойства BindableProperty

Допустим, мы хотим определить свое свойство BindableProperty в каком-то своем классе. Например, мы хотим расширить функционал класса Label, чтобы он включал некоторый тег, который присваивается метке. Для этого своздадим свой класс, производный от Label:

```
public class TagLabel : Label
```

```
{
    public static readonly BindableProperty TagProperty  =
        BindableProperty.Create("Tag", // название обычного свойства
            typeof(string), // возвращаемый тип
            typeof(TagLabel), // тип,  котором обьявляется свойство
            "0"// значение по умолчанию
        );
    public string Tag
    {
        set
        {
            SetValue(TagProperty, value);
        }
        get
        {
            return (string)GetValue(TagProperty);
        }
    }
}
```

Данный класс располагается в главном проекте решения.

Для определения свойства BindableProperty используется метод **BindableProperty.Create()**. Этот метод возвращает объект BindableProperty и принимает в данном случае четыре параметра по порядку:

- Имя обычного свойства, которое будет оберткой. В данном случае свойство будет называться "Tag"
- Возвращаемый тип свойства. В данном случае тип string
- Название типа, в котором объявляется это свойство. Здесь тип TagLabel
- Значение по умолчанию. Здесь строка "0"

Это не все возможные параметры. Другие перегруженные версии метода BindableProperty.Create() могут принимать еще шесть параметров по порядку:

- `defaultBindingMode` - режим привязки
- `validateValue` - метод, который проверяет новое значение на корректность
- `propertyChanged` - метод, который вызывается при изменении свойства
- `propertyChanging` - метод, который вызывается перед изменением свойства
- `coerceValue` - метод корректировки нового значения
- `defaultValueCreator` - метод-генератор значения по умоланию

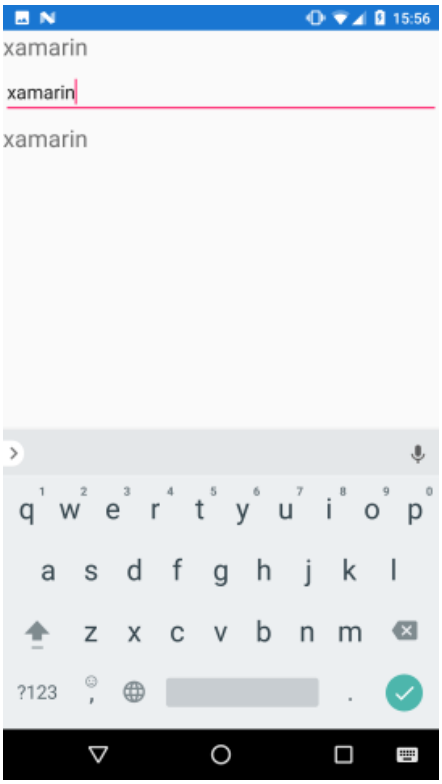
После определения класса и свойства они могут участвовать в привязке данных. Так, пусть у нас будет следующий код страницы:

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        TagLabel tagLabel = new TagLabel
        {
            FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label))
        };
        Entry entry = new Entry();
        // Устанавливаем привязку
        // источник привязки - entry, цель привязки - tagLabel
        tagLabel.BindingContext = entry;
        // Связываем свойства источника и цели
        tagLabel.SetBinding(TagLabel.TagProperty, "Text");
        tagLabel.SetBinding(TagLabel.TextProperty, "Text");

        Label label = new Label
        {
            FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label))
        };

        label.BindingContext = tagLabel;
        // устанавливаем привязку к свойству Tag объекта tagLabel
        label.SetBinding(Label.TextProperty, "Tag");
        StackLayout stackLayout = new StackLayout()
        {
            Children = { tagLabel, entry, label}
        };
```

```
        Content = stackLayout;
    }
}
```

Итак, здесь объект нашего класса TagLabel привязан к объекту entry. Причем сразу два свойства - Text и Tag у объекта TagLabel привязаны к свойству Text объекта Entry.

Также здесь есть простой объект Label, свойство Text которого привязано к свойству Tag объекта TagLabel. Поэтому при вводе символов в текстовое поле Entry синхронно будет изменяться значение свойства Tag у tagLabel, а это в свою очередь вызовет изменение свойства Text у простого объекта Label.



Таким образом, определив свое свойство по типу BindableProperty впоследствии мы сможем осуществлять к нему привязку.

Определение аналогичной функциональности в XAML-коде:

```
<?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>
```


Назад Содержание Вперед