



## Объект Binding

Последнее обновление: 16.06.2016



# CUBA Platform - Open Source Java Web Framework

Design UI and data model visually, develop in any Java IDE [cuba-platform.com](http://cuba-platform.com)

Свойство `BindingContext` не является единственным способом для установки источника привязки. Также мы можем инкапсулировать все опции привязки в объекте **Binding**. Например:

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        Label label = new Label
        {
            FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label))
        };
        Entry entry = new Entry();

        // определяем объект привязки: Source - источник, Path - его свойство
        Binding binding = new Binding { Source = entry, Path = "Text" };
        // установка привязки для свойства TextProperty
        label.SetBinding(Label.TextProperty, binding);

        StackLayout stackLayout = new StackLayout()
        {
            Children = {label, entry }
        };
        Content = stackLayout;
    }
}
```

Здесь у объекта `Binding` настраиваются ряд свойств. В частности, свойство `Source` указывает на источник привязки, а свойство `Path` - на свойство источника. В данном случае источником служит объект `Entry`, а свойством - его свойство `Text`.

Далее в метод `SetBinding()` в качестве второго параметра передается объект `Binding`, инкапсулирующий все опции привязки.

В итоге мы получаем то же самое, что и в прошлой теме: свойство `Text` объекта `Label` привязано к свойству `Text` объекта `Entry`.

Тоже самое можно прописать и в XAML:

```
<?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" FontSize="Large" Text="{Binding Source={x:Reference Name=entry}, Path=Text}" />
        <Entry x:Name="entry" />
    </StackLayout>
</ContentPage>
```

Таким образом, нам необязательно использовать свойство `BindingContext` для установки привязки к источнику. Более того использование объекта `Binding` имеет свои преимущества. В частности, с помощью `BindingContext` мы можем установить только один источник привязки для какого-либо элемента. Но что если мы хотим, чтобы одно свойство у `Label` было привязано к одному элементу, другое свойство - к другому элементу? В этом случае можно использовать только вышеописанный способ.

Также мы можем одновременно устанавливать и `BindingContext`, и использовать объект `Binding` для определения опций привязки, и в этом случае объект `Binding` будет переопределять `BindingContext`, если, к примеру, в `BindingContext` указан один

источник привязки, а в объекте Binding другой.

## Режим привязки

Режим привязки определяет в какую сторону будет работать привязка. Все возможные режимы задаются с помощью значений из перечисления **BindingMode**:

- Default: значение по умолчанию. Для разных элементов и свойств может отличаться
- OneWay: при изменении в источнике изменяется цель
- OneWayToSource: при изменении в цели привязки изменяется источник (то есть обратная привязка)
- TwoWay: изменения в источнике воздействуют на цель привязки, а изменения в цели воздействуют на источник (то есть двусторонняя привязка)

По умолчанию стандартным значением для всех элементов и их свойств является значение OneWay, то есть односторонняя привязка от источника к цели. Однако ряд элементов и их свойств по умолчанию используют двустороннюю привязку:

### Элемент

#### Свойство

Stepper

Value

Switch

IsToggled

Entry

Text

Editor

Text

SearchBar

Text

DatePicker

Date

TimePicker

Time

Например, пусть у нас два связанных текстовых поля:

```
<?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>
```

```
</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 одностороннюю привязку от источника к цели, то мы можем изменить выражение привязки следующим образом:

```
<Entry x:Name="entry1" BindingContext="{x:Reference Name=entry2}"
      Text="{Binding Path=Text, Mode=OneWay}" />
```

Для задания режима привязки в коде C# надо установить свойство Mode у объекта Binding:

```
Entry entry1 = new Entry();
Entry entry2 = new Entry();

Binding binding = new Binding { Source = entry2, Path = "Text", Mode= BindingMode.OneWay };
entry1.SetBinding(Label.TextProperty, binding);
```

## StringFormat

Свойство **StringFormat** класса Binding позволяет дополнительно отформатировать значение:

```
<?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 FontSize="Large" Text="{Binding Source={x:Reference Name=slider},
                                           Path=Value,
                                           StringFormat='Установлено: {0}'}" />
    <Slider x:Name="slider" Minimum="0" Maximum="50" Value="30" />
  </StackLayout>
</ContentPage>
```



Свойству StringFormat передается некоторая строка в одинарных кавычках, которая указывает, как будет форматироваться значение. Плейсхолдер {0} указывает на начальное значение, которое передается от привязки.



JOIN IN

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





Яндекс.Директ

0 Комментариев metanit.com

1 Войти ▾

Рекомендовать
 Поделиться

Лучшее в начале ▾



Начать обсуждение...

ВОЙТИ С ПОМОЩЬЮ

ИЛИ ЧЕРЕЗ DISQUS ?

Имя

Прокомментируйте первым.

ТАКЖЕ НА METANIT.COM

### C++ | Указатель на функцию как возвращаемое значение

1 комментарий • 4 месяца назад



Даниил Данилов — Не знал что можно декораторы делать как JS

### Vue.js | Валидация props

1 комментарий • 4 месяца назад



eugene81 — Подключена минимизированная версия библиотеки - сообщения об ошибках не показываются. При подключении не сжатой - видны

### Go | Соответствие интерфейсу

1 комментарий • 23 дня назад



Py — Еще давай. Не останавливайся, пожалуйста. Это гениальный язык. Он, конечно, не божественен, как шарп, но гениальный.


### Go | Сервер. Обработка подключений

2 комментариев • 4 дня назад



Metanit — Да


Подписаться
 Добавь Disqus на свой сайт
 Добавить Disqus
 Добавить
 Конфиденциальность



## Voltage Level Translators

From Texas  
Instruments

View Products,  
Design Tools,  
Software &  
Technical  
Documents & Get  
Support.



[ti.com/Logic/Voltage](http://ti.com/Logic/Voltage)

---

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

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