



Aula 07 – Programação de Dispositivos Móveis

Professor: Me. Wellington Tuler Moraes
Curso: ADS – IFSP – Campus Cubatão

Binding

- O Binding do Xamarin.Forms é, na verdade, uma abreviação – o nome completo é Data Binding.
- Data Binding*, segundo a Wikipedia, é uma técnica para, de forma simples, ligar duas propriedades de um software de forma que as informações de uma sejam acessadas na outra.
- É uma propriedade que permite a ligação de dois objetos, mudanças em um causará mudanças no outro.

Projeto - MainPage

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
3             xmlns:local="clr:namespace=MonkeyHubApp"
4             x:Class="MonkeyHubApp.MainPage">
5
6     <Label Text="{Binding Descricao}"
7           VerticalOptions="Center"
8           HorizontalOptions="Center" />
9
10 </ContentPage>
  
```

Projeto – MainPage.cs

```

1 using MonkeyHubApp.ViewModel;
2 using System;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Text;
6 using System.Threading.Tasks;
7 using Xamarin.Forms;
8
9 namespace MonkeyHubApp
10 {
11     [XamlCompilation(XamlCompilationOptions.Compile)]
12     public partial class MainPage : ContentPage
13     {
14         public MainPage()
15         {
16             InitializeComponent();
17             BindingContext = new MainViewModel();
18         }
19     }
20 }
  
```

MainViewModel

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MonkeyHubApp.ViewModel
8 {
9     public class MainViewModel
10     {
11         public string Descricao { get; set; }
12
13         public MainViewModel() {
14             Descricao = "Olá Mundo, eu estou aqui!";
15         }
16     }
17 }
  
```

Resultado



Mudança

```
using System.Threading.Tasks;

namespace MonkeyHubApp.ViewModel
{
    2 referências
    public class MainViewModel
    {
        2 referências
        public string Descricao { get; set; }

        1 referência
        public MainViewModel() {
            Descricao = "Olá Mundo, eu estou aqui!";

            Task.Delay(3000).ContinueWith(t =>
            {
                Descricao = "Meu texto mudou!";
            });
        }
    }
}
```

Problema

- Com o uso da estrutura usando o código lambda a seguir:


```
Task.Delay(3000).ContinueWith(t =>
{
    Descricao = "Meu texto mudou!";
});
```
- O texto da tela deveria mudar, porém infelizmente ele não muda, pois a tela não é notificada desta mudança.
- Se fizermos um processo de Debug podemos constatar que a ViewModel de fato altera a propriedade.

Solução

- Informar a página XAML sobre a mudança de estado da propriedade.
- O valor foi alterado depois de “setado” o valor do Binding.
- Empregar uma interface para notificar esta mudança.
- Principal Interface: INotifyPropertyChanged.

Implementar a Interface

```
MainViewModel.cs  App.xaml.cs  MainPage.xaml  MainPage.xaml.cs
MonkeyHubApp  MonkeyHubApp.ViewModel.MainViewM - MainViewModel()

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using System.ComponentModel;
7
8 namespace MonkeyHubApp.ViewModel
9 {
10     2 referências
11     public class MainViewModel : INotifyPropertyChanged
12     {
13         public event PropertyChangedEventHandler PropertyChanged;
14         public string Descricao { get; set; }
```

Notificar a mudança

```
MainViewModel.cs  App.xaml.cs  MainPage.xaml  MainPage.xaml.cs
MonkeyHubApp  MonkeyHubApp.ViewModel.MainViewM - PropertyChanged

9
10 public class MainViewModel : INotifyPropertyChanged
11 {
12     public event PropertyChangedEventHandler PropertyChanged;
13     2 referências
14     public string Descricao { get; set; }
15
16     1 referência
17     public MainViewModel() {
18         Descricao = "Olá Mundo, eu estou aqui!";
19
20         Task.Delay(3000).ContinueWith(t =>
21         {
22             Descricao = "Meu texto mudou!";
23             PropertyChanged.Invoke(this, new PropertyChangedEventArgs("Descricao"));
24         });
25     }
26 }
```

Passa a propriedade como parâmetro

Verificar se é nulo

```
MainViewModel.cs  App.xaml.cs  MainPage.xaml  MainPage.xaml.cs
MonkeyHubApp  MonkeyHubApp.ViewModel.MainViewM - MainViewModel()

9
10 {
11     2 referências
12     public class MainViewModel : INotifyPropertyChanged
13     {
14         public event PropertyChangedEventHandler PropertyChanged;
15         public string Descricao { get; set; }
16
17         1 referência
18         public MainViewModel() {
19             Descricao = "Olá Mundo, eu estou aqui!";
20
21             Task.Delay(3000).ContinueWith(t =>
22             {
23                 Descricao = "Meu texto mudou!";
24                 if (PropertyChanged != null) {
25                     PropertyChanged.Invoke(this, new PropertyChangedEventArgs("Descricao"));
26                 }
27             });
28         }
29     }
30 }
```

Resultado



Elvis Operator

- É um operador binário (?) que executa o restante da operação se o conteúdo da propriedade for diferente de nulo (verdadeiro). Similar ao ternário.

```
Descricao = "Novo texto mudou!";
if (PropertyChanged != null) {
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Descricao"));
}
```

```
Descricao = "Novo texto mudou!";
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Descricao"));
```

Diretiva nameof()

- Visa diminuir os erros de digitação do programador.
- Melhora o código por ser fortemente tipada, não permite erros do nome da variável.

```
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(Descricao)))
```

propfull

- Atalho para criar propriedades.
- Digitar propfull + tab + tab.

```
propfull
propfull
Trecho de código para propriedade e campo subjacente.
Observação: pressione Tab duas vezes para inserir o trecho 'propfull'.
```

```
private int myVar;

public int MyProperty
{
    get { return myVar; }
    set { myVar = value; }
}
```

Melhorando a notificação

- Colocaremos as notificações de mudança dentro do código do método set da propriedade descrição.

```
public string Descricao {
    get { return _descricao; }
    set {
        _descricao = value;
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(Descricao)));
    }
}
```

View-to-View Bindings

- Data Binding pode ser definido como um link de duas views em uma mesma página.
- Neste caso, usamos o BindingContext de um objeto alvo usando a extensão markup x:Reference

Código Exemplo

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/winfx/2006/xaml"
              xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
              x:Class="Sample.SliderBindingPage"
              Title="Slider Bindings Page">

    <StackLayout>
        <Label text="ROTATION"
               BindingContext="{x:Reference Name=slider}"
               Rotation="{Binding Path=Value}"
               FontAttributes="Bold"
               FontSize="Large"
               HorizontalOptions="Center"
               VerticalOptions="CenterAndExpand" />

        <Slider x:Name="slider"
                Maximum="360"
                VerticalOptions="CenterAndExpand" />

        <Label BindingContext="{x:Reference slider}"
               text="{Binding Value,
                           StringFormat='The angle is {0:F0} degrees'}"
               FontAttributes="Bold"
               FontSize="Large"
               HorizontalOptions="Center"
               VerticalOptions="CenterAndExpand" />
    </StackLayout>
</ContentPage>
```

Exemplo

- O Slider possui um atributo chamado x:Name que é referenciado por dois labels view usando a extensão x:Reference

BindingContext="{x:Reference Name=slider}"

...

BindingContext="{x:Reference slider}"

Exemplo

- A markup tag Binding possui muitas propriedades como a BindingBase e a classe Binding.
- O ContentProperty para Binding é o "path=", por exemplo:

Rotation="{Binding Path=Value}"

...

Text="{Binding Value,
StringFormat='The angle is {0:F0}
degrees'}"