Aula 07 – Programação de Dispositivos Móveis

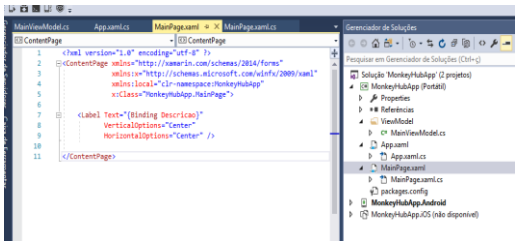
Professor: Me. Wellington Tuler Moraes

Curso: ADS – IFSP – Campus Cubatão

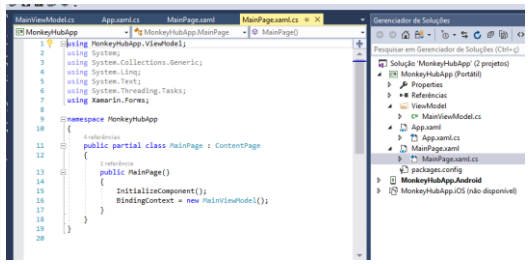
Binding

- 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.

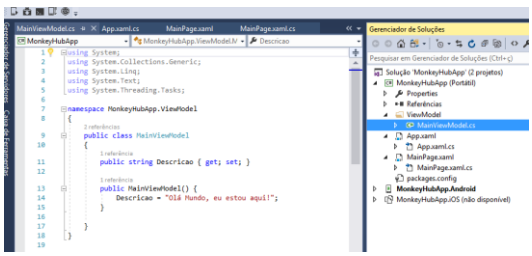
Projeto - MainPage



Projeto – MainPage.cs



MainViewModel



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 Biding.
- Empregar uma interface para notificar esta mudança.
- Principal Interface: INotifyPropertyChanged.

Implementar a Interface

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ComponentModel;

namespace MonkeyHubApp.ViewModel
{
    2 referências
    public class MainViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        public string Descricao { get; set; }
    }
```

Notificar a mudança

```
public class MainViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    public string Descricao { get; set; }

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

        Task.Delay(3000).ContinueWith(t =>
        {
            Descricao = "Meu texto mudou!";
            PropertyChanged.Invoke(this, new PropertyChangedEventArgs("Descricao") );
        });
    }
```

Passa a propriedade como parâmetro

Verificar se é nulo

```
public class MainViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    public string Descricao { get; set; }

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

        Task.Delay(3000).ContinueWith(t =>
        {
            Descricao = "Meu texto mudou!";
            if (PropertyChanged != null) {
                PropertyChanged.Invoke(this, new PropertyChangedEventArgs("Descricao"));
            }
        });
    }
```

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

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

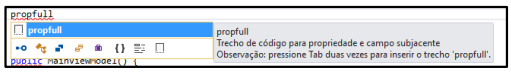
Diretiva nameof()

- 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 porpfull + tab + tab.




```
private int myVar;

0 referências
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 extenção markup x:Reference

Código Exemplo

```
<?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="XamlSamples.SliderBindingsPage"
             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 marckup tag Binding possui muitas proprieddes 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'}"