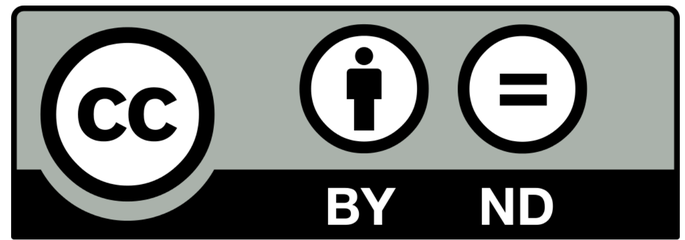
programação cross-platform com

xamarin

daniel ferreira
@dfsdaniel

Licença

- **Compartilhar:**
  - Você tem o direito de copiar e redistribuir o material em qualquer suporte ou formato para qualquer fim, mesmo que comercial.
- **Atribuição:**
  - Você deve dar o crédito apropriado, prover um link para a licença e indicar se mudanças foram feitas. Você deve fazê-lo em qualquer circunstância razoável, mas de maneira alguma que sugira ao licenciante a apoiar você ou o seu uso.
- **Sem derivações:**
  - Se você remixar, transformar ou criar a partir do material, você não pode distribuir o material modificado.
- **Licença CC BY-ND 4.0:**
  - https://creativecommons.org/licenses/by-nd/4.0/deed.pt_BR



Markup Extensions

Markup Extensions

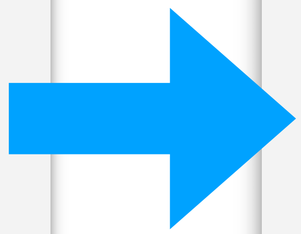
- Serve como uma “extensão” do XAML.
- Permite utilizar valores nas propriedades que não necessariamente são do tipo **string (ou formato textual)**.
- Permite acessar valores de propriedades e classes em tempo de execução.
- São classes que implementam a interface **IMarkupExtension** (possui um único método ProvideValue).
- Todas possuem o sufixo **"Extension"** no nome (porém não precisa explicitar no uso).

- **XAML Padrão:**
  - x:Static
  - x:Reference
  - X:Type
  - X:Null
  - x:Array
- **WPF:**
  - StaticResource
  - DynamicResource
  - Binding
- **Xamarin:**
  - ConstraintExpression

x:Static

- Permite que o valor de uma propriedade do XAML seja de uma **propriedade estática** de alguma classe.
- Possui apenas a propriedade **Member**.

```
<Label Text="Just some text"
    BackgroundColor="Accent"
    TextColor="Black"
    FontAttributes="Italic"
    VerticalOptions="Center" />
```

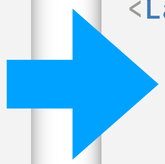


```
<Label Text="Just some text">
    <Label.BackgroundColor>
        <x:StaticExtension Member="Color.Accent" />
    </Label.BackgroundColor>
    <Label.TextColor>
        <x:StaticExtension Member="Color.Black" />
    </Label.TextColor>
    <Label.FontAttributes>
        <x:StaticExtension Member="FontAttributes.Italic" />
    </Label.FontAttributes>
    <Label.VerticalOptions>
        <x:StaticExtension Member="LayoutOptions.Center" />
    </Label.VerticalOptions>
</Label>
```

x:Static

- O sufixo "Extension" pode ser omitido.
- É possível utilizar a sintaxe de atributo utilizando chaves { }.
  - É a forma mais comum de se utilizar Markup Extensions.
  - **Member** é a **ContentProperty** da classe, então pode ser omitida também.

```
<Label Text="Just some text"
    BackgroundColor="{x:Static Member=Color.Accent"
    TextColor="{x:Static Member=Color.Black"
    FontAttributes="{x:Static Member=FontAttributes.Italic"
    VerticalOptions="{x:Static Member=LayoutOptions.Center" />
```



```
<Label Text="Just some text"
    BackgroundColor="{x:Static Color.Accent"
    TextColor="{x:Static Color.Black"
    FontAttributes="{x:Static FontAttributes.Italic"
    VerticalOptions="{x:Static LayoutOptions.Center" />
```

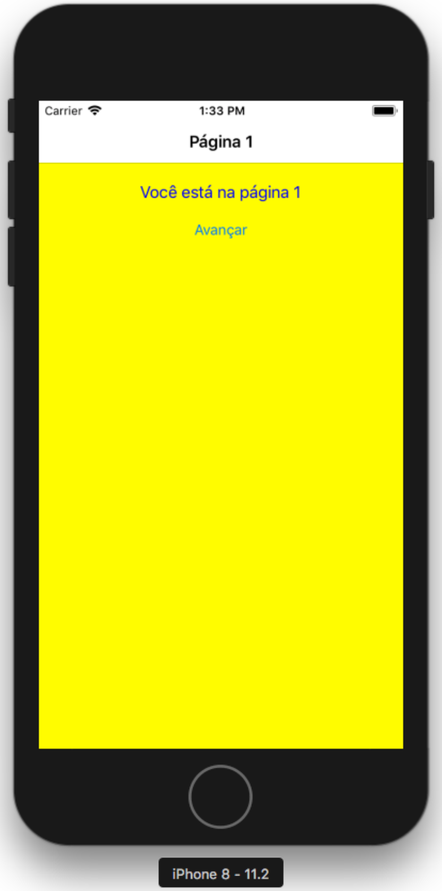
Utilizando as Chaves { }

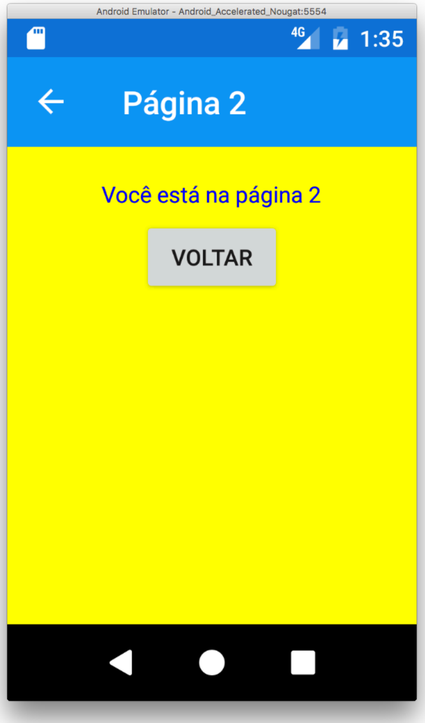
- No XAML, qualquer valor entre chaves é interpretado como um **markup extension**.
- Caso o valor do texto precise exibir exatamente as chaves, é necessário iniciar o valor com “{}".

```
<Label Text="{}{Texto aparecendo entre chaves}" />
```

Exercício

- Crie uma aplicação que possui 2 telas.
- As duas telas devem possuir **cor de fundo amarela, texto centralizado no meio de cor azul e negrito** indicando em qual tela o usuário está e botões de navegação para avançar ou voltar para outras telas.
- As cores de fundo e do texto **precisam ser compartilhadas** entre as telas para evitar repetição de código.





Resources Dictionary

- Dicionário de recursos (chave/valor).
- **Resources** podem ser compartilhados por:
  - Todos os elementos da aplicação;
  - Todos os elementos de uma mesma página;
  - Todos os elementos de um escopo específico.
- Todo **VisualElement** no Xamarin possui a propriedade Resources (do tipo **ResourceDictionary**);
- Todo resource no dicionário precisa ter um **key (string)** unicamente identificada dentro da coleção;
- Para acessar os resources dentro dos dicionários, utilizamos o markup extension **StaticResource**.

StaticResource

- **x:Static** permite apenas acessar métodos e propriedades estáticas das classes.
- **StaticResource** permite acessar resources dentro de uma lista do tipo **ResourceDictionary**.
- Um resource pode ser qualquer objeto.

```
<ContentPage.Resources>
    <ResourceDictionary>
        <x:Double x:Key="borderWidth">3</x:Double>
    </ResourceDictionary>
</ContentPage.Resources>
```

```
<ContentPage.Resources>
    <ResourceDictionary>
        <Color x:Key="textColor"
               x:FactoryMethod="FromRgb">
            <x:Arguments>
                <x:Double>0</x:Double>
                <x:Double>1</x:Double>
                <x:Double>0.5</x:Double>
            </x:Arguments>
        </Color>
    </ResourceDictionary>
</ContentPage.Resources>
```

```
<ContentPage.Resources>
    <ResourceDictionary>
        <OnPlatform x:Key="backgroundColor"
                    x:TypeArguments="Color"
                    Android="#404040" />
    </ResourceDictionary>
</ContentPage.Resources>
```

```
<ContentPage.Resources>
    <ResourceDictionary>
        <Color x:Key=“textColor">Red</Color>
    </ResourceDictionary>
</ContentPage.Resources>
```

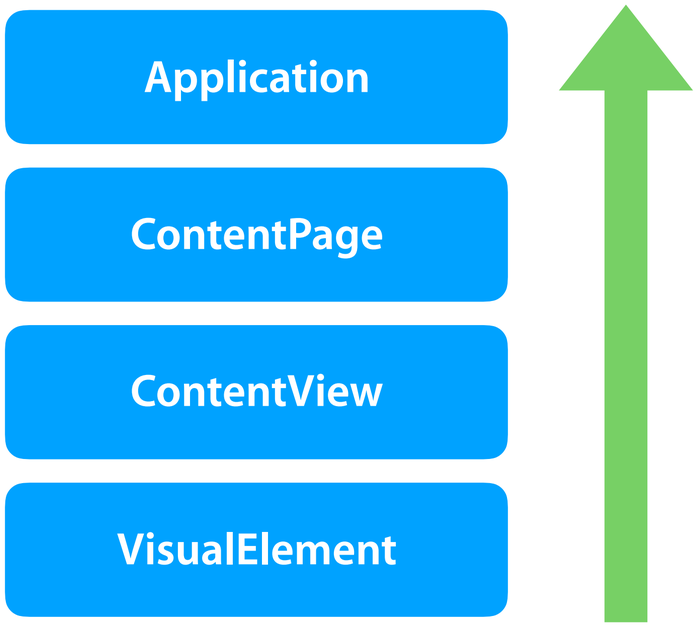
StaticResource

- Para utilizar **StaticResource**, basta utilizar a propriedade **Key** (**ContentProperty**).
- É possível utilizar com a sintaxe de elemento, ou com a sintaxe de atributo em conjunto com a sintaxe entre chaves { }.

```
<Button>
    <Button.BackgroundColor>
        <StaticResourceExtension Key="backgroundColor" />
    </Button.BackgroundColor>
</Button>
```

```
<Button Text="Botão centralizado"
        HorizontalOptions="{StaticResource horzOptions}" />
```

Hierarquia de Dicionários



Criando sua Biblioteca

- Para criar suas próprias bibliotecas, existe um template no Visual Studio que facilita esta tarefa:
- Botão direito na solução > Add New Project > Multiplatform > **Class Library (Xamarin Forms)**
- Para importar no seu projeto:
  - Botão direito na pasta References > **Edit References**.
  - **Importante!!**
    - Necessário que sua biblioteca seja iniciada pelo menos uma vez no seu código C#.
    - Geralmente fazemos isso no **App.cs**.
      - Chamamos um método **Init()** que não faz nada, apenas para instanciar a biblioteca.
- Para importar no XAML:

Criando Markup Extensions

- É possível criar sua própria **markup extension**.
- Basta implementar a interface **IMarkupExtension**.
  - Possui apenas um método **ProvideValue**.

```
public object ProvideValue(IServiceProvider serviceProvider)
{
    return Color.FromHsla(H, S, L, A);
}
```

```
<StackLayout>
    <BoxView>
        <BoxView.Color>
            <exLib:HslColorExtension H="0" S="1" L="0.5" />
        </BoxView.Color>
    </BoxView>
    <BoxView Color="{exLib:HslColorExtension H=0, S=0, L=0.5}" />
</StackLayout>
```

Platform Specific API

Shared Access Project (SAP)

- Relembrando:
  - Ele é todo copiado para dentro de cada projeto específico no momento do build.
  - Tem acesso direto as APIs e classes dentro de cada projeto.
- Utilizar SAP é uma forma mais direta e simples de acessar chamadas específicas de cada plataforma.

Diretivas Pré-Compilação

- Disponível apenas no projetos do tipo SAP (Shared Access Project)
- Condicionais:
  - #if
  - #elif
  - #else
  - #endif
- Símbolos:
  - __IOS__
  - __ANDROID__
  - WINDOWS_UWP
  - WINDOWS_APP
  - WINDOWS_PHONE_APP

Diretivas Pré-Compilação

- Disponível apenas no projetos do tipo SAP (Shared Access Project)
- Condicionais:
  - #if
  - #elif
  - #else
  - #endif
- Símbolos:
  - __IOS__
  - __ANDROID__
  - WINDOWS_UWP
  - WINDOWS_APP
  - WINDOWS_PHONE_APP

```
using Xamarin.Forms;
using System;

#if __IOS__
using UIKit;

#elif __ANDROID__
using Android.OS;

#endif
```

Diretivas Pré-Compilação

```
#if __IOS__
    UIDevice device = new UIDevice();
    modelLabel.Text = device.Model;
    versionLabel.Text = String.Format("{0} {1}", device.SystemName, device.SystemVersion);

#elif __ANDROID__
    modelLabel.Text = String.Format("{0} {1}", Build.Manufacturer, Build.Model);
    versionLabel.Text = Build.VERSION.Release.ToString();

#endif
```

Classes Paralelas

- Uma forma mais elegante de acessar classes nos projetos específicos de cada plataforma.
- Basta que o **nome das classes e seus namespaces** sejam os mesmos em cada projeto específico.
  - O nome dos métodos também precisam ser os mesmos.

Classes Paralelas

iOS

```
using System;
using UIKit;

namespace PlatInfoSap
{
    public class PlatformInfo
    {
        UIDevice device = new UIDevice();

        public string GetModel()
        {
            return device.Model;
        }

        public string GetVersion()
        {
            return String.Format("{0} {1}",
                                 device.SystemName,
                                 device.SystemVersion);
        }
    }
}
```

Android

```
using System;
using Android.OS;

namespace PlatInfoSap
{
    public class PlatformInfo
    {
        public string GetModel()
        {
            return String.Format("{0} {1}",
                                 Build.Manufacturer,
                                 Build.Model);
        }

        public string GetVersion()
        {
            return Build.VERSION.Release;
        }
    }
}
```