Ministério da Educação

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO **CAMPUS CUBATÃO**

Matéria: PDMI – ADS 671 – Professor: Wellington Tuler Moraes

Nome:_____ Data: __/__/_____

Nome:_____ Data: __/__/_____

Fonte: https://docs.microsoft.com/pt-br/xamarin/xamarin-forms/xaml/xaml-basics/get-started-with-xaml?tabs=vswin

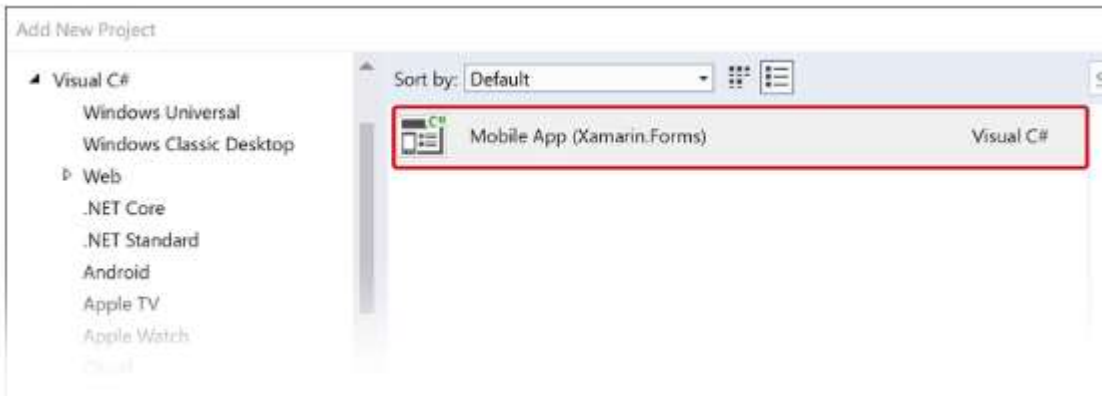
Parte 1. Guia de Introdução com XAML

Em um aplicativo xamarin. Forms, XAML é usado principalmente para definir o conteúdo visual da página e trabalha em conjunto com um arquivo de code-behind c#.

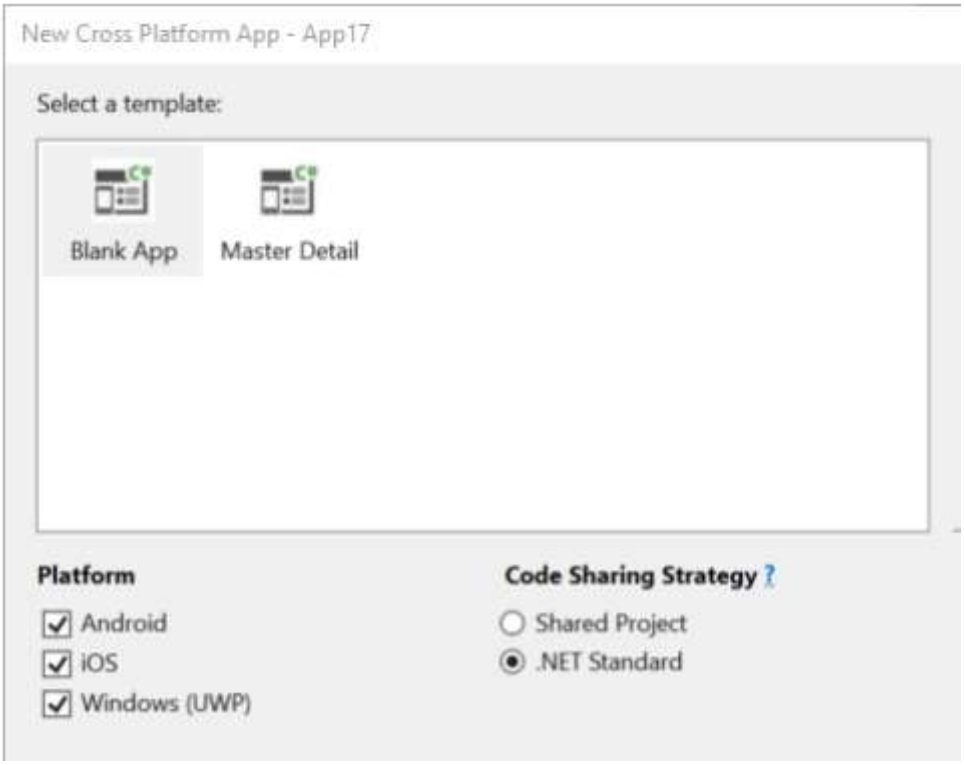
O arquivo code-behind fornece suporte de código para a marcação. Juntos, esses dois arquivos contribuem para uma nova definição de classe que inclui exibições de filho e inicialização de propriedade. Dentro do arquivo XAML, classes e propriedades são referenciadas com atributos e elementos XML e links entre a marcação e código são estabelecidas.

Criando a solução

Para começar a editar o arquivo XAML primeiro, use o Visual Studio ou o Visual Studio para Mac para criar uma nova solução xamarin. Forms.



No Windows, use o Visual Studio para selecionar **arquivo > Novo > projeto** no menu. No **novo projeto**caixa de diálogo, selecione **Visual C# > plataforma cruzada** à esquerda e, em seguida, **aplicativo móvel (xamarin. Forms)** da lista no centro.



Selecione um local para a solução, dê a ele um nome de **XamlSamples** (ou que preferir) e pressione **Okey**.

Na próxima tela, selecione a **aplicativo em branco** modelo e o **.NET padrão** estratégia de compartilhamento de código:

Quatro projetos são criados na solução: o **XamlSamples** biblioteca .NET padrão, **XamlSamples.Android**, **XamlSamples.iOS**e a plataforma Universal do Windows solução, **XamlSamples.UWP**.

Depois de criar o **XamlSamples** solução, você talvez queira testar seu ambiente de desenvolvimento, selecionando vários projetos de plataforma como o projeto de inicialização de solução e criar e implantar o aplicativo simple criado por o modelo de projeto em emuladores de telefone ou dispositivos reais.

A menos que você precisa para escrever código específico da plataforma, compartilhado **XamlSamples** o projeto de biblioteca .NET padrão é onde você

vai gastar praticamente todo o seu tempo programação.Esses artigos não serão de risco fora do projeto.

Anatomia de um arquivo XAML

Dentro de **XamlSamples** biblioteca .NET padrão são um par de arquivos com os seguintes nomes:

- **App**, o arquivo XAML; e
- **App.XAML.CS**, *c# por trás do código* arquivo associado ao arquivo XAML.

Você precisará clicar na seta ao lado de **App** para ver o arquivo code-behind.

Ambos **App** e **App.xaml.cs** contribuem para uma classe denominada `App` que deriva de `Application`. A maioria das outras classes com arquivos XAML contribuem para uma classe que deriva de `ContentPage`; esses arquivos usam XAML para definir o conteúdo visual de uma página inteira. Isso é verdadeiro para os outros dois arquivos a **XamlSamples** projeto:

- **MainPage. XAML**, o arquivo XAML; e
- **MainPage.xaml.cs**, o arquivo de code-behind c#.

O **MainPage. XAML** arquivo tem a seguinte aparência (embora a formatação pode ser um pouco diferente):

XAML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:XamlSamples"
             x:Class="XamlSamples.MainPage">

    <StackLayout>
        <!-- Place new controls here -->
        <Label Text="Welcome to Xamarin Forms!"
               VerticalOptions="Center"
               HorizontalOptions="Center" />
```

```
    </StackLayout>

</ContentPage>
```

O namespace XML duas (`xmlns`) declarações consultem URIs, a primeira aparentemente no site de web do Xamarin e a segunda da Microsoft. Não se preocupe em que ponto os URIs para a verificação. Não há nada lá. Eles são simplesmente os URIs de propriedade Xamarin e Microsoft e funcionam basicamente como identificadores de versão.

A primeira declaração de namespace XML significa que marcas definidas dentro do arquivo XAML sem prefixo referem-se para as classes em xamarin. Forms, por exemplo `ContentPage`. A segunda declaração de namespace define um prefixo de `x`. Isso é usado para vários elementos e atributos que são intrínsecos XAML em si e que é suportado por outras implementações do XAML. No entanto, esses elementos e atributos são um pouco diferentes dependendo do ano inserido no URI. Xamarin. Forms oferece suporte a especificação de XAML 2009, mas não todas.

O `local` declaração de namespace permite acessar outras classes do projeto de biblioteca .NET padrão.

No final da primeira marca, o `x` prefixo é usado para um atributo chamado `Class`. Porque o uso deste `x` prefixo é praticamente universal para o namespace XAML, atributos XAML, como `Class` quase sempre são chamados de `x:Class`.

O `x:Class` atributo especifica um nome de classe totalmente qualificado do .NET: o `MainPage` classe no `XamlSamples` namespace. Isso significa que esse arquivo XAML define uma nova classe chamada `MainPage` no `XamlSamples` namespace que deriva de `ContentPage` — a marca no qual o `x:Class` atributo aparece.

O `x:Class` atributo só pode aparecer no elemento raiz de um arquivo XAML para definir uma classe derivada do c#. Esta é a classe de somente nova definida no arquivo XAML. Em vez disso, tudo que aparece no arquivo XAML é instanciado a partir de classes existentes-las simplesmente e inicializado.

O **MainPage.xaml.cs** arquivo tem esta aparência (além de não utilizados `using` diretivas):

C#

```
using Xamarin.Forms;

namespace XamlSamples
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

O `MainPage` classe derivada de `ContentPage`, mas observe o `partial` definição da classe. Isso sugere que deve ser outra definição de classe parcial para `MainPage`, mas onde ele está? E o que é isso `InitializeComponent` método?

Quando o Visual Studio compila o projeto, ele analisa o arquivo XAML para gerar um arquivo de código c#.Se você examinar o **XamlSamples\XamlSamples\obj\Debug** diretório, você encontrará um arquivo chamado **XamlSamples.MainPage.xaml.g.cs**. O 'g' significa gerado. Essa é outra definição de classe parcial de `MainPage` que contém a definição do `InitializeComponent` chamado a partir do método a `MainPage`construtor. Esses dois parcial `MainPage` definições de classe podem ser compiladas juntos. Dependendo se o XAML é compilado ou não, o arquivo XAML ou um formato binário do arquivo XAML é inserido no executável.

Em tempo de execução, o código nas chamadas de projeto da plataforma específica um `LoadApplication` método, passando a ela uma nova instância do `App` classe na biblioteca do .NET padrão. O `App` cria uma instância do construtor da classe `MainPage`. Chama o construtor da classe `InitializeComponent`, que, em seguida, chama o `LoadFromXaml` método que extrai o arquivo XAML (ou seu binário compilado) da biblioteca do .NET padrão. `LoadFromXaml` inicializa todos os objetos definidos no arquivo XAML, conecta todos juntos em relações pai-filho, anexa os manipuladores de eventos definidos em código para eventos definidos no arquivo XAML e define a árvore resultante de objetos, como o conteúdo da página.

Embora normalmente não é necessário gastar muito tempo com arquivos de código gerado, às vezes, exceções de tempo de execução são geradas no código em arquivos gerados, portanto você deve estar familiarizado com eles.

Quando você compilar e executar esse programa, o `Label` elemento aparece no centro da página, como sugere o XAML. As três plataformas da esquerda para a direita são UWP, Android e iOS:

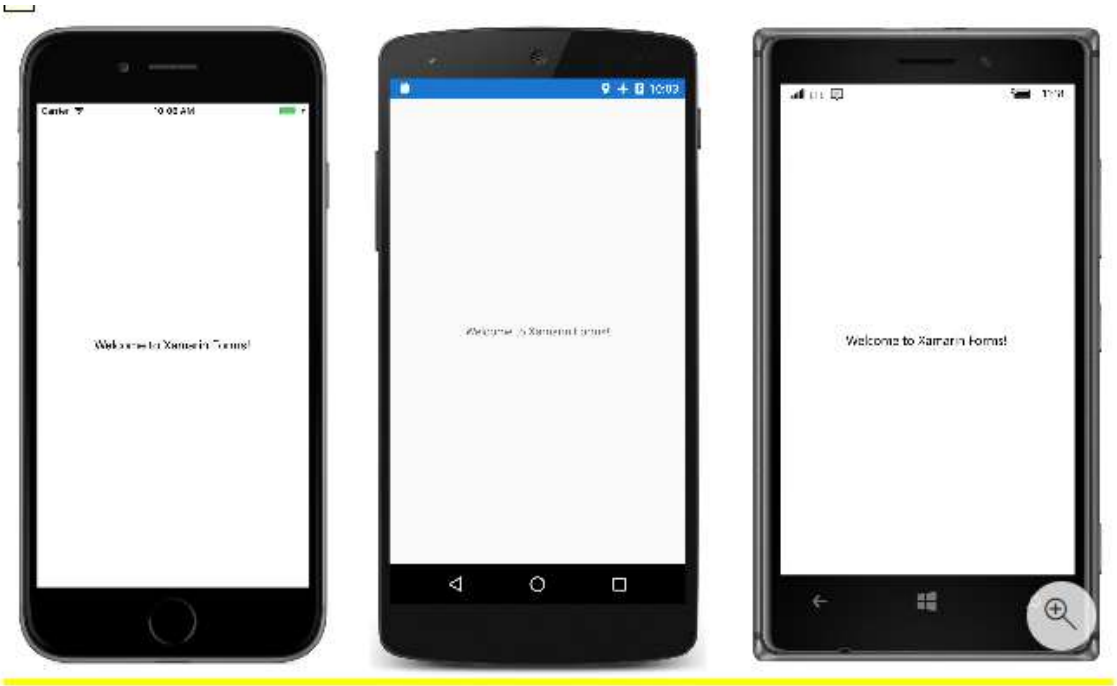
Ministério da Educação

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO **CAMPUS CUBATÃO**

Matéria: PDMI – ADS 671 – Professor: Wellington Tuler Moraes

Nome:_____ Data: __/__/_____

Nome:_____ Data: __/__/_____



Para obter visuais mais interessantes, tudo o que você precisa é mais interessante XAML.

Para obter visuais mais interessantes, tudo o que você precisa é mais interessante XAML.

Adicionando novas páginas XAML

Para adicionar outros baseados em XAML `ContentPage` classes ao seu projeto, selecione o **XamlSamples** biblioteca .NET padrão do projeto e invocar o **projeto > Adicionar Novo Item** item de menu. À esquerda do **Adicionar Novo Item** caixa de diálogo, selecione **Visual C#** e **xamarin. Forms**.Selecione a lista **página de conteúdo** (não **página de conteúdo (c#)**, que cria uma página de código somente ou **exibição de conteúdo**, que não é uma página). Nomeie a página, por exemplo, **HelloXamlPage.xaml**:

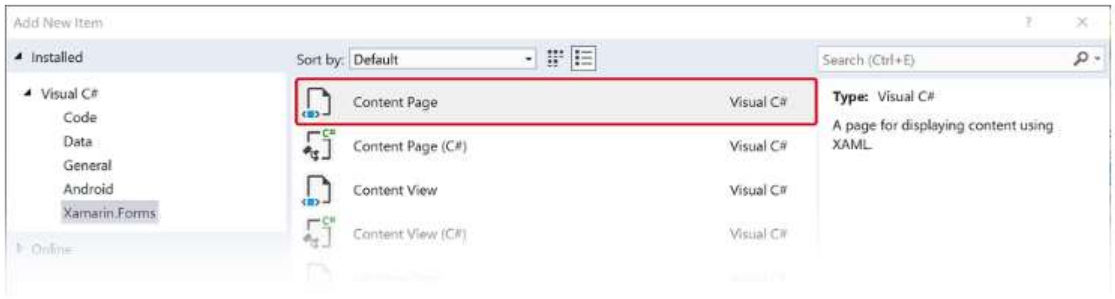
Ministério da Educação

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO **CAMPUS CUBATÃO**

Matéria: PDMI – ADS 671 – Professor: Wellington Tuler Moraes

Nome:_____ Data: __/__/_____

Nome:_____ Data: __/__/_____



Dois arquivos são adicionados ao projeto, **HelloXamlPage.xaml** e o arquivo code-behind **HelloXamlPage.xaml.cs**.

Conteúdo da página de configuração

Editar o **HelloXamlPage.xaml** arquivos de forma que as marcas somente são as de `ContentPage` e `ContentPage.Content`:

XAML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="XamlSamples.HelloXamlPage">
    <ContentPage.Content>

    </ContentPage.Content>
</ContentPage>
```

O `ContentPage.Content` marcas fazem parte da sintaxe exclusivo do XAML. Primeiro, eles podem parecer ser um XML inválido, mas são válidos. O período não é um caractere especial em XML.

O `ContentPage.Content` marcas são chamadas *elemento property* marcas. `Content` é uma propriedade de `ContentPage` e geralmente é

definido como uma única exibição ou um layout com modos de exibição do filho. Normalmente propriedades se tornam atributos em XAML, mas seria difícil de configurar um `Content` de atributo para um objeto complexo. Por esse motivo, a propriedade é expressa como um elemento XML que consiste em nome de classe e o nome da propriedade separados por um período. Agora o `Content` propriedade pode ser definida entre o `ContentPage.Content` marcas, como este:

XAML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="XamlSamples.HelloXamlPage"
             Title="Hello XAML Page">
    <ContentPage.Content>

        <Label Text="Hello, XAML!"
               VerticalOptions="Center"
               HorizontalTextAlignment="Center"
               Rotation="-15"
               IsVisible="true"
               FontSize="Large"
               FontAttributes="Bold"
               TextColor="Blue" />

    </ContentPage.Content>
</ContentPage>
```

Observe também que uma `Title` atributo foi definido na marca raiz.

Neste momento, a relação entre classes, propriedades e XML deve ser evidente: xamarin. Forms de uma classe (como `ContentPage` ou `Label`) aparece no arquivo XAML como um elemento XML. Propriedades de classe — incluindo `Title` na `ContentPage` e sete propriedades de `Label` — geralmente aparecem como atributos XML.

Existem muitos atalhos para definir os valores dessas propriedades. Algumas propriedades são tipos de dados básicos: por exemplo, o `Title` e `Text` propriedades são do tipo `String`, `Rotation` é do tipo `Double`, e `IsVisible` (que é `true` por padrão e é definido aqui apenas para ilustração) é do tipo `Boolean`.

O `HorizontalTextAlignment` é de propriedade do tipo `TextAlignment`, que é uma enumeração. Para uma propriedade de qualquer tipo de enumeração, tudo o que você precisa fonte é um nome de membro.

Para propriedades de tipos mais complexos, porém, conversores são usadas para analisar o XAML. São as classes em xamarin. Forms que derivam de `TypeConverter`. Muitos são classes públicas, mas alguns não são.Para este arquivo XAML específico, várias dessas classes desempenham uma função em segundo plano:

- `LayoutOptionsConverter` para o `VerticalOptions` propriedade
- `FontSizeConverter` para o `FontSize` propriedade
- `ColorTypeConverter` para o `TextColor` propriedade

Esses conversores regem a sintaxe permitida das configurações de propriedade.

O `ThicknessTypeConverter` pode lidar com um, dois ou quatro números separados por vírgulas. Se um número for fornecido, ele se aplica a todos os quatro lados. Com dois números, a primeira é à esquerda e o preenchimento à direita e a segunda é superior e inferior. Quatro números estão na ordem esquerda, superior, direita e inferior.

O `LayoutOptionsConverter` pode converter os nomes dos campos estáticos públicos do `LayoutOptions`estrutura para valores do tipo `LayoutOptions`.

O `FontSizeConverter` pode lidar com um `NamedSize` membro ou um tamanho de fonte numérico.

O `ColorTypeConverter` aceita os nomes de campos estáticos públicos do `Color` estrutura ou valores hexadecimais, com ou sem um canal alfa, precedido por um sinal numérico (#). Aqui está a sintaxe sem um canal alfa:

```
TextColor="#rrggbb"
```

Cada uma das letras pouco é um dígito hexadecimal. Aqui está como um canal alfa é incluído:

```
TextColor="#aarrggbb">
```

Para o canal alfa, tenha em mente que FF é completamente opaco e 00 é totalmente transparente.

Dois outros formatos permitem que você especifique apenas um dígito hexadecimal para cada canal:

`TextColor="#rgb"` `TextColor="#argb"`

Nesses casos, o dígito é repetido para formar o valor. Por exemplo, #CF3 é a cor RGB CC-FF-33.

Navegação de página

Quando você executa o **XamlSamples** programa, o `MainPage` é exibido. Para ver o novo `HelloXamlPage` você pode definir que, como a inicialização de novo, página de **App.xaml.cs** arquivo ou navegue para a nova página de `MainPage`.

Para implementar a navegação, primeiro altere o código no **App.xaml.cs** construtor para que um `NavigationPage` objeto é criado:
C#

```
public App()
{
    InitializeComponent();
    MainPage = new NavigationPage(new MainPage());
}
```

No **MainPage.xaml.cs** construtor, você pode criar um simples `Button` e usar o manipulador de eventos para navegar até `HelloXamlPage`:
C#

```
public MainPage()
{
    InitializeComponent();

    Button button = new Button
```

```
    {
        Text = "Navigate!",
        HorizontalOptions = LayoutOptions.Center,
        VerticalOptions = LayoutOptions.Center
    };

    button.Clicked += async (sender, args) =>
    {
        await Navigation.PushAsync(new HelloXamlPage());
    };

    Content = button;
}
```

Definindo o `Content` propriedade da página substitui a configuração do `Content` propriedade no arquivo XAML. Quando você compila e implanta a nova versão deste programa, um botão é exibido na tela.Pressioná-lo navega para `HelloXamlPage`. Aqui está a página resultante no iPhone, Android e UWP:

Você pode navegar de volta para `MainPage` usando o **< volta** botão no iOS, usando a seta para a esquerda na parte superior da página ou na parte inferior do telefone no Android, ou na seta à esquerda na parte superior da página no Windows 10.

Fique à vontade para experimentar o XAML para as diferentes maneiras renderizar o `Label`. Se você precisar inserir quaisquer caracteres Unicode em texto, você pode usar a sintaxe XML padrão. Por exemplo, para colocar a saudação inteligente aspas, use:

```
<Label Text="&#x201C;Hello, XAML!&#x201D;" … />
```

XAML e interações de código

O **HelloXamlPage** exemplo contém um único `Label` na página, mas isso está correto. A maioria dos `ContentPage` conjunto derivados de `Content` classificar de propriedade para um layout de algumas, como

um `StackLayout`. O `Children` propriedade o `StackLayout` é definido para ser do tipo `IList<View>` , mas é realmente um objeto do tipo `ElementCollection<View>`, e que a coleção pode ser preenchida com várias exibições ou outros layouts. Em XAML, essas relações pai-filho são estabelecidas com a hierarquia XML normal. Aqui está um arquivo XAML para uma nova página chamada **XamlPlusCodePage**:

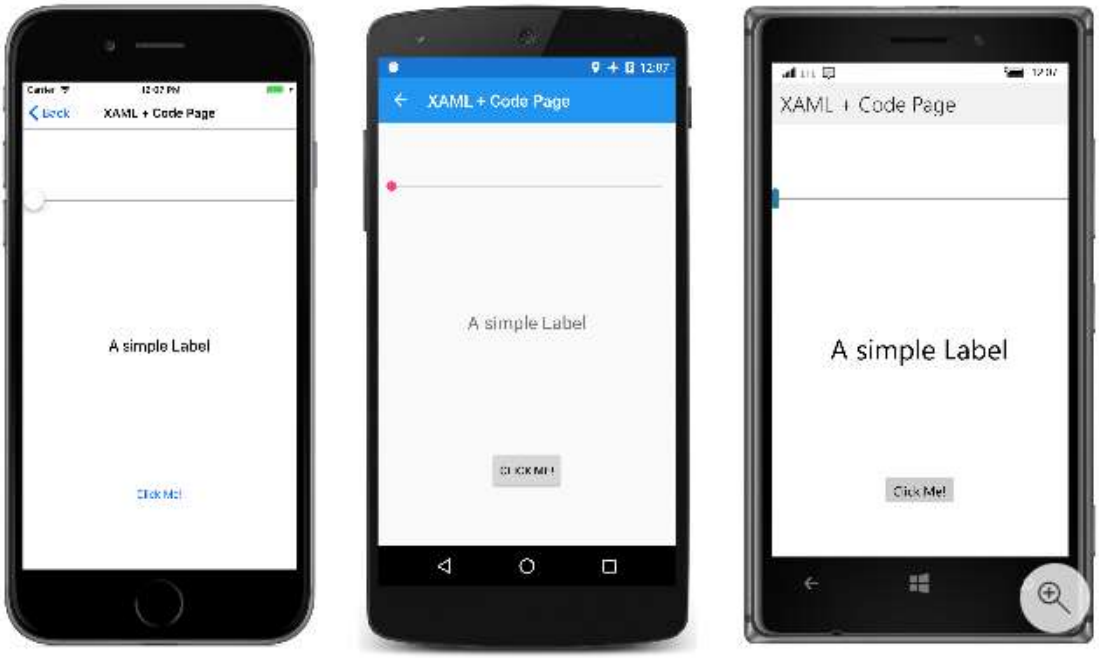
XAML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="XamlSamples.XamlPlusCodePage"
             Title="XAML + Code Page">
    <StackLayout>
        <Slider VerticalOptions="CenterAndExpand" />

        <Label Text="A simple Label"
               Font="Large"
               HorizontalOptions="Center"
               VerticalOptions="CenterAndExpand" />

        <Button Text="Click Me!"
                HorizontalOptions="Center"
                VerticalOptions="CenterAndExpand" />
    </StackLayout>
</ContentPage>
```

Esse arquivo XAML é sintaticamente completo e aqui está o que se parece:



No entanto, você provavelmente serão considere este programa seja funcionalmente deficiente. Talvez o `Slider` deve para causar o `Label` para exibir o valor atual e o `Button` destina-se provavelmente para fazer algo dentro do programa.

Como você verá no parte 4. Noções básicas de associação de dados, o trabalho de exibir um `Slider` valor usando um `Label` podem ser manipulados inteiramente XAML com uma associação de dados. Mas é útil ver a solução de código primeiro. Mesmo assim, tratar o `Button` clique definitivamente requer código. Isso significa que o arquivo de code-behind para `XamlPlusCodePage` deve conter manipuladores para o `ValueChanged` eventos do `Slider` e `Clicked` eventos do `Button`. Vamos adicioná-los:

C#

```
namespace XamlSamples
{
    public partial class XamlPlusCodePage
```

```
    {
        public XamlPlusCodePage()
        {
            InitializeComponent();
        }

        void OnSliderValueChanged(object sender, ValueChangedEventArgs
args)
        {

        }

        void OnButtonClicked(object sender, EventArgs args)
        {

        }
    }
}
```

Esses manipuladores de eventos não precisa ser público.

No arquivo XAML, o `Slider` e `Button` marcas precisam incluir atributos para o `ValueChanged` e `Clicked` que referenciam esses manipuladores de eventos:

XAML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="XamlSamples.XamlPlusCodePage"
             Title="XAML + Code Page">
    <StackLayout>
        <Slider VerticalOptions="CenterAndExpand"
                ValueChanged="OnSliderValueChanged" />

        <Label Text="A simple Label"
               Font="Large"
               HorizontalOptions="Center"
               VerticalOptions="CenterAndExpand" />
```

```
        <Button Text="Click Me!"
                HorizontalOptions="Center"
                VerticalOptions="CenterAndExpand"
                Clicked="OnButtonClicked" />
    </StackLayout>
</ContentPage>
```

Observe que a atribuição de um manipulador de um evento tem a mesma sintaxe atribuindo um valor a uma propriedade.

Se o manipulador para o `ValueChanged` evento o `Slider` usará o `Label` para exibir o valor atual, o manipulador deve fazer referência a esse objeto de código. O `Label` precisa de um nome, que é especificado com o `x:Name` atributo.

XAML

```
<Label x:Name="valueLabel"
       Text="A simple Label"
       Font="Large"
       HorizontalOptions="Center"
       VerticalOptions="CenterAndExpand" />
```

O `x` prefixo o `x:Name` atributo indica que esse atributo é intrínseco à XAML.

O nome atribuído para o `x:Name` atributo tem as mesmas regras que os nomes de variável c#. Por exemplo, ele deve começar com uma letra ou um sublinhado e conter sem espaços.

Agora o `ValueChanged` manipulador de eventos pode definir o `Label` para exibir o novo `Slider` valor. O novo valor está disponível de argumentos do evento:

C#Copiar

```
void OnSliderValueChanged(object sender, ValueChangedEventArgs args)
{
    valueLabel.Text = args.NewValue.ToString("F3");
```

```
}
```

Ou, o manipulador foi possível obter o `Slider` objeto que está gerando este evento a partir de `sender` argumento e obter o `Value` propriedade do:
C#Copiar

```
void OnSliderValueChanged(object sender, ValueChangedEventArgs args)
{
    valueLabel.Text = ((Slider)sender).Value.ToString("F3");
}
```

Ao executar o programa pela primeira vez o `Label` não exibe o `Slider` valor porque o `ValueChanged` evento ainda não foi acionado. Mas qualquer manipulação do `Slider` faz com que o valor a ser exibido:

Agora o `Button`. Vamos simular uma resposta a uma `Clicked` evento exibindo um alerta com o `Text` do botão. O manipulador de eventos pode ser convertido com segurança o `sender` argumento para um `Button` e, em seguida, acessar suas propriedades:

C#Copiar

```
async void OnButtonClicked(object sender, EventArgs args)
{
    Button button = (Button)sender;
    await DisplayAlert("Clicked!",
        "The button labeled '" + button.Text + "' has been clicked",
        "OK");
}
```

O método está definido como `async` porque o `DisplayAlert` método é assíncrono e deve ser precedido com o `await` operador, que retorna quando o método é concluído. Como esse método obtém o `Button`acionamento do evento do `sender` argumento, o mesmo manipulador pode ser usado para vários botões.

Você viu que um objeto definido em XAML pode disparar um evento que é tratado no arquivo code-behind e que o arquivo code-behind pode acessar um objeto definido em XAML usando o nome atribuído a ele com o `x:Name` atributo. Estas são as duas maneiras fundamentais que interagem de código e XAML.

Algumas ideias adicionais sobre como funciona XAML pode ser obtidas examinando o recém-gerado **XamlPlusCode.xaml.g.cs arquivo**, que agora inclui qualquer nome atribuído a qualquer `x:Name` atributo como um campo particular. Aqui está uma versão simplificada do arquivo:

C#

```
public partial class XamlPlusCodePage : ContentPage {

    private Label valueLabel;
```

```
    private void InitializeComponent() {
        this.LoadFromXaml(typeof(XamlPlusCodePage));
        valueLabel = this.FindByName<Label>("valueLabel");
    }
}
```

A declaração do campo permite que a variável deve ser livremente usado em qualquer lugar dentro de `XamlPlusCodePage` arquivo de classe parcial em sua jurisdição. Em tempo de execução, o campo é atribuído depois que o XAML foi analisado. Isso significa que o `valueLabel` campo é `null` quando o `XamlPlusCodePage` construtor começa mas válido após `InitializeComponent` é chamado.

Depois de `InitializeComponent` retorna controle volta para o construtor, os visuais da página foram construídos como se tivesse foi instanciados e inicializados no código. O arquivo XAML não desempenha nenhuma função na classe. Você pode manipular esses objetos na página de qualquer forma que você quer, por exemplo, adicionando modos de exibição para o `StackLayout`, ou a configuração do `Content`propriedade da página para algo totalmente. Você pode "percorrer a árvore" Examinando o `Content`propriedade da página e os itens a `Children` coleções de layouts. Você pode definir propriedades em modos de exibição acessados dessa maneira, ou atribuir manipuladores de eventos-los dinamicamente.

Fique à vontade. É a página e XAML é apenas uma ferramenta para criar o seu conteúdo.