

# Laboratório

---

## *Compartilhando código entre projetos multiplataforma*

Versão: 1.0.0  
Outubro de 2016



[Miguel Muñoz Serafín](#)

@msmdotnet



## CONTEÚDO

### INTRODUÇÃO

#### EXERCÍCIO 1: EXAMINANDO AS OPÇÕES PARA COMPARTILHAR CÓDIGO

Tarefa 1. Criar uma aplicação Xamarin.Forms com o modelo SAP.

Tarefa 2. Examinar os arquivos de código gerados.

#### EXERCÍCIO 2: UTILIZANDO O VIEW LABEL PARA MOSTRAR TEXTO

Tarefa 1. Criar uma solução Xamarin.Forms PCL.

Tarefa 2. Incluir espaço (preenchimento) na página (Solução 1).

Tarefa 3. Incluir espaço (preenchimento) apenas para iOS em projetos SAP (Solução 2).

Tarefa 4. Incluir espaço (preenchimento) apenas para iOS em projetos PCL ou SAP (Solução 3).

Tarefa 5. Centralizar a **Label** dentro da página (Solução 4).

Tarefa 6. Centralizar o texto dentro da **Label** (Solução 5).

### SUMÁRIO

# Introdução

---

Quando você criou a solução **HelloXamarinForms** no Visual Studio, teve a opção de selecionar 2 tipos principais de modelos:

- Xamarin.Forms Portable
- Xamarin.Forms Shared

A primeira opção cria uma biblioteca de classes portátil (PCL), enquanto a segunda opção cria um projeto de Recursos portátil (SAP), que consiste unicamente de arquivos de código compartilhado.

Neste laboratório, vamos discutir as várias opções para compartilhar código em uma aplicação multiplataforma e explorar o *View **Label*** com as considerações que devemos ter no momento de mostrar o texto em uma aplicação multiplataforma desenvolvida com Xamarin.Forms.

## Objetivos

Após a conclusão deste laboratório, os participantes serão capazes de:

- Descrever como compartilhar o código através de projetos PCL.
- Descrever como compartilhar o código através dos projetos SAP.
- Utilizar o componente **Label** para exibir o texto em uma aplicação Xamarin.Forms.

## Requisitos

Para a realização deste laboratório é necessário contar com o seguinte:

- Um computador de desenvolvimento com sistema Windows 10 e Visual Studio 2015 Community, Professional ou Enterprise com a plataforma Xamarin.
- Um computador Mac com a plataforma Xamarin.

Tempo estimado para completar este laboratório: **60 minutos**.

# Exercício 1: Examinando as opções para compartilhamento de código.

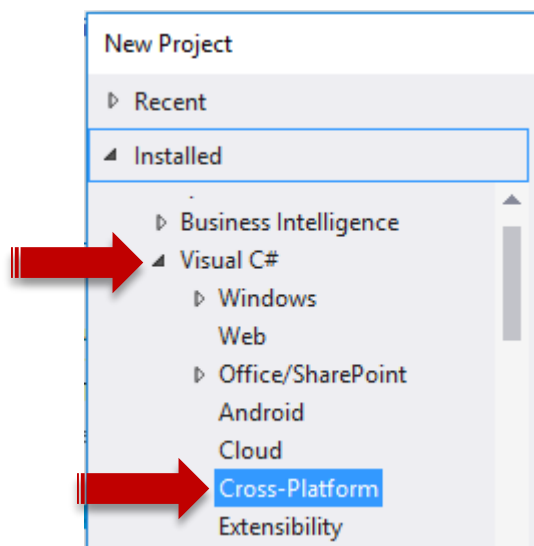
---

Neste exercício, você terá a oportunidade de explorar as vantagens e desvantagens das duas opções disponíveis para compartilhar código em uma aplicação multiplataforma Xamarin.Forms.

## Tarefa 1. Criar uma aplicação Xamarin.Forms com o modelo SAP.

A solução criada no laboratório anterior usou o modelo PCL. Agora é hora de criar uma segunda solução usando o modelo SAP.

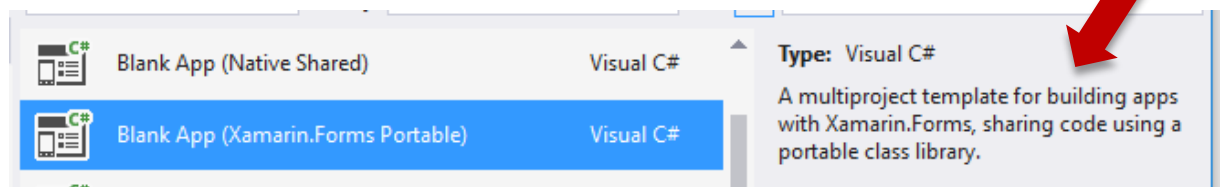
1. Selecione a opção **File > New > Project** no Visual Studio.
2. No painel esquerdo da janela **New Project** selecione **Visual C# > Cross-Platform**.



3. No painel central da janela **New Project** você pode ver diferentes modelos de solução disponíveis incluindo 5 de Xamarin.Forms:
  - a. Blank App (Xamarin.Forms Portable)
  - b. Blank App (Xamarin.Forms Shared)
  - c. Blank Xaml App (Xamarin.Forms Portable)
  - d. Blank Xaml App (Xamarin.Forms Shared)
  - e. Class Library (Xamarin.Forms)



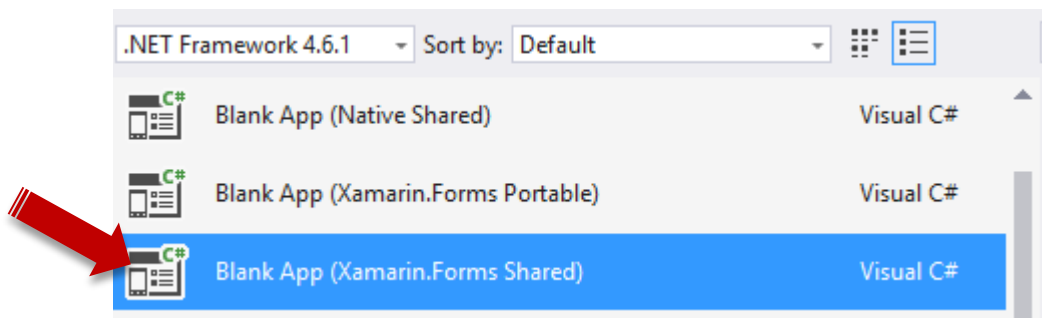
Clique em cada um dos modelos para ver sua descrição no painel da direita.



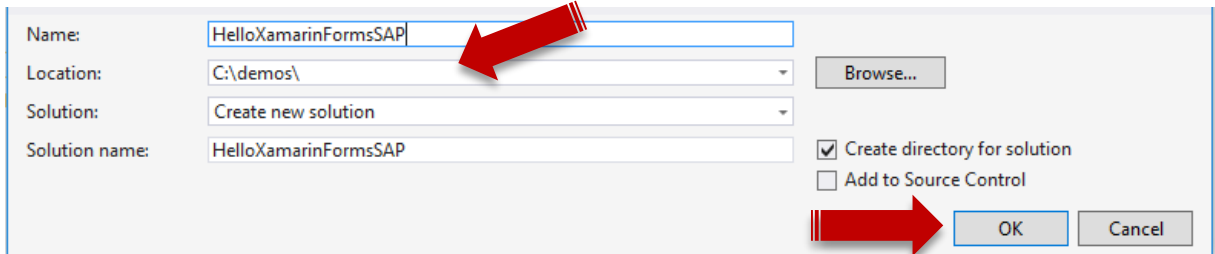
O termo “**Portable**” neste contexto se refere a uma Biblioteca de Classes Portátil (Portable Class Library – PCL). Todo o código comum de aplicativos se converte em uma biblioteca DLL que é referenciado por todos os projetos de plataforma individual.

O termo “**Shared**” neste contexto se refere a um Projeto de Recursos Compartilhados (Shared Asset Project – SAP). Este projeto contém arquivos de código e outro tipo de arquivos que são compartilhados com os projetos de cada plataforma, essencialmente, tornar-se parte de cada projeto em cada plataforma.

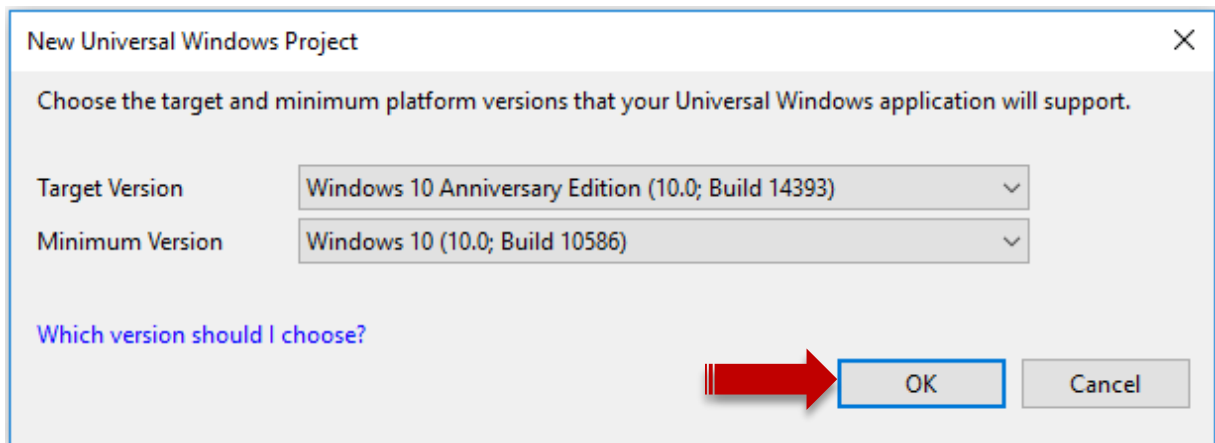
4. Selecione o modelo **Blank App (Xamarin.Forms Shared)**.



5. Forneça o nome, local e clique em **OK** para criar o projeto.

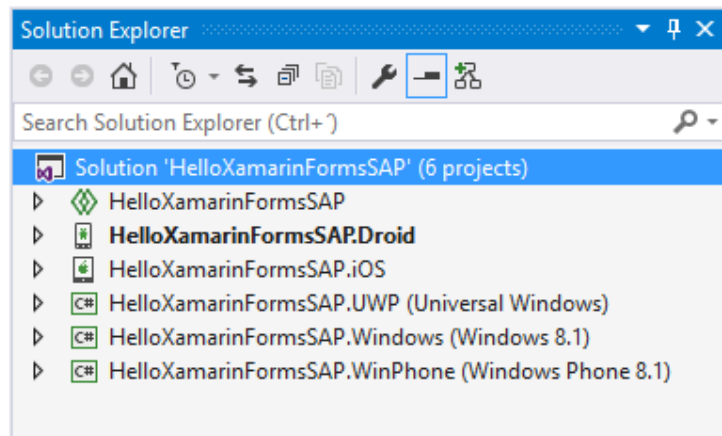


6. Clique no botão **OK** da caixa de diálogo **New Universal Windows Project** para aceitar as versões sugeridas para a aplicação UWP que será criada.



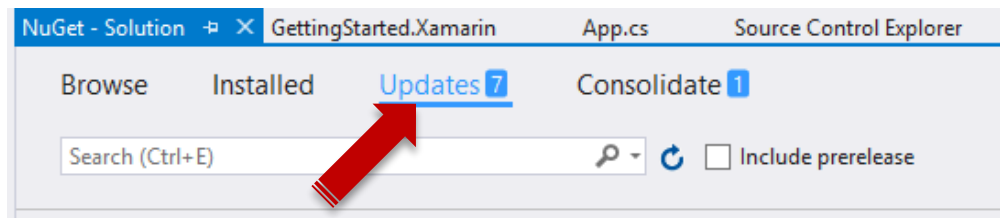
Seis projetos são criados. Para uma solução chamada **HelloXamarinFormsSAP**, esses projetos são:

- Um projeto PCL chamado **HelloXamarinFormsSAP** que é referenciado por outros 5 projetos.
- Um projeto de aplicação para Android chamado **HelloXamarinSAP.Droid**.
- Um projeto de aplicação para iOS chamado **HelloXamarinSAP.iOS**.
- Um projeto de aplicação para a UWP de Windows 10 e Windows Mobile 10 chamado **HelloXamarinFormsSAP.UWP**.
- Um projeto de aplicação para Windows 8.1 chamado **HelloXamarinFormsSAP.Windows**.
- Um projeto de aplicação para Windows Phone 8.1 chamado **HelloXamarinFormsSAP.WinPhone**.

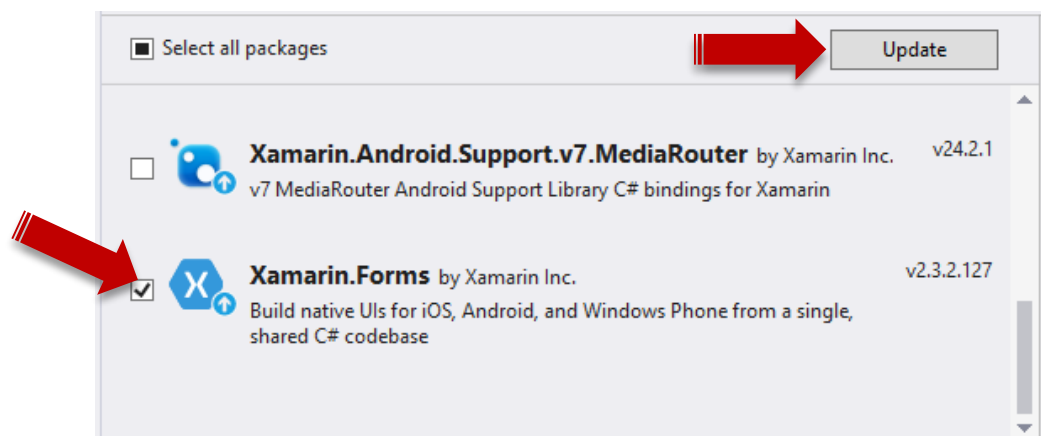


Quando você cria uma nova solução Xamarin.Forms, as bibliotecas Xamarin.Forms e outras bibliotecas auxiliares são automaticamente transferidas a partir do gerenciador de pacotes NuGet. Visual Studio armazena essas bibliotecas em um diretório chamado pacotes dentro da solução. No entanto, a versão específica da biblioteca do Xamarin.Forms que é baixado é especificado dentro do modelo da solução e, portanto, uma nova versão pode estar disponível.

7. Selecione a opção **Manage NuGet Packages for Solution** no menu contextual do nome da solução.
8. Clique na opção **Updates** para ver as atualizações disponíveis.



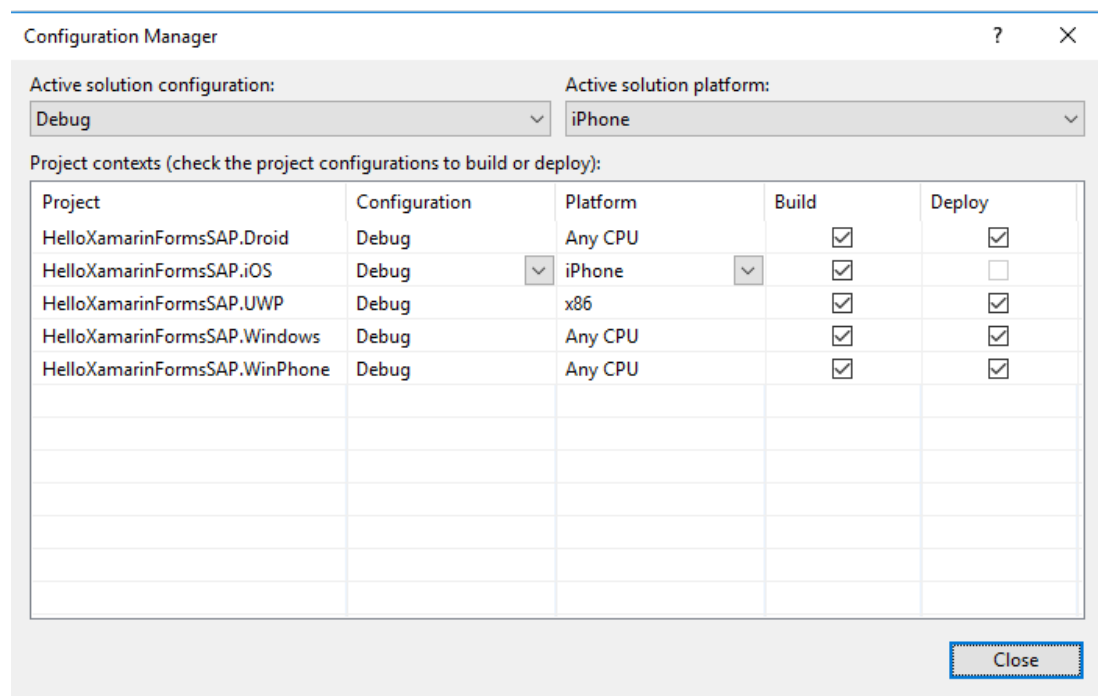
9. Selecione o pacote **Xamarin.Forms** e clique em **Update** para iniciar a atualização.



Você provavelmente será solicitado para aceitar as alterações feitas e reiniciar o Visual Studio para concluir a instalação.

10. Uma vez que a atualização for concluída, selecione a opção **Build > Configuration Manager**.
11. Na caixa de diálogo **Configuration Manager** você vai ver o projeto SAP e os outros 5 projetos de aplicação.

Verifique se a caixa de verificação **Build** está selecionada para todos os projetos e que a caixa de seleção **Deploy** está selecionada para todos os projetos de aplicação (a menos que a caixa seja cinza).



Observe que na coluna **Platform**:

- O projeto SAP estabeleceu a única opção disponível **Any CPU**.
- O projeto Android estabeleceu a única opção disponível **Any CPU**.
- O projeto iOS pode ter os valores **iPhone** ou **iPhoneSimulator** dependendo de como estaremos testando o aplicativo.
- O projeto UWP pode ter o valor **x86** para implantar o aplicativo para o desktop do Windows ou para um emulador. Também pode ter o valor **ARM** para implantar o aplicativo a um telefone.
- O projeto do Windows pode levar o valor **x86** para implantar o aplicativo para o desktop do Windows ou para um emulador. Também pode ter o valor **ARM** para implantar o aplicativo para um tablet. Você pode até ter o valor **x64** para implantar o



aplicativo para plataformas de 64 bits ou valor **Any CPU** para implantar o aplicativo no desktop do Windows, emulador, tablet ou plataformas de 64 bits.

- O projeto WinPhone pode ter o valor **x86** para implantar o aplicativo para um emulador, o valor **ARM** para implantar o aplicativo para um telefone real ou valor **Any CPU** para implantar a aplicação, tanto no emulador quanto no telefone real.

12. Clique em **Close** para fechar a caixa de diálogo **Configuration Manager**.

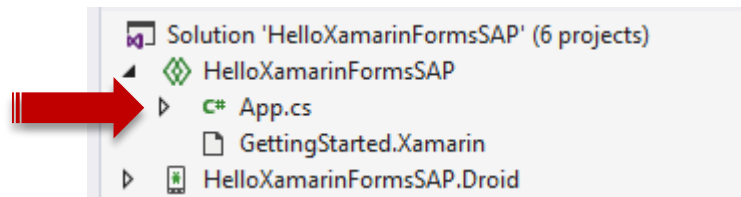
13. Teste a operação de cada um dos projetos de cada plataforma. Você verá a mensagem **"Welcome to Xamarin Forms!"** em cada plataforma.

## Tarefa 2. Examinar os arquivos de código gerados.

O projeto SAP é o projeto que receberá a maior parte de nossa atenção quando estamos escrevendo uma aplicação Xamarin.Forms. Em algumas circunstâncias, o código neste projeto poderia exigir algum código especial de certas plataformas.

Como você pode ver, tudo parece o mesmo entre uma solução PCL e uma solução SAP. Aqui estão algumas diferenças.

1. A primeira diferença que notamos é que o projeto SAP contém apenas um item: o arquivo **App.cs** (além do arquivo `GettingStarted.Xamarin` que é informativo).



Tanto em projetos PCL como nos projetos SAP, o código é compartilhado com as 5 aplicações, mas de maneiras diferentes.

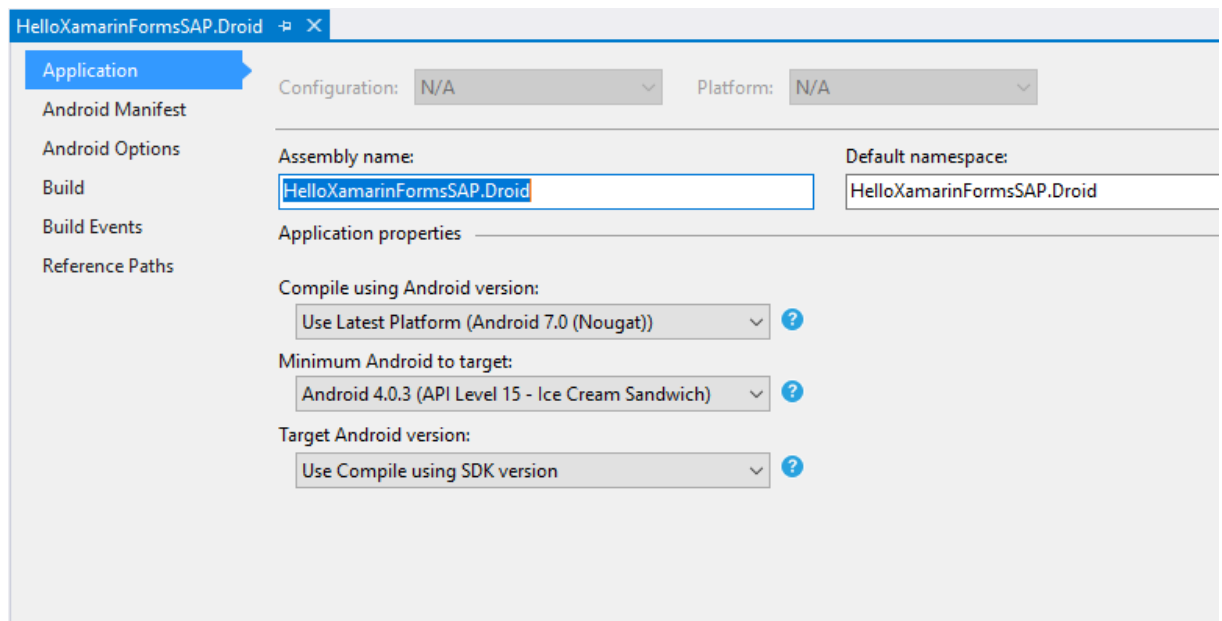
- Com a aproximação do projeto PCL, todo o código comum é encapsulado dentro de uma biblioteca DLL que cada projeto de aplicativos faz referências e links em tempo de execução.
- Com a aproximação do projeto SAP, com os arquivos de código comuns são realmente incluídos com cada um dos 5 projetos em tempo de compilação.

Por padrão, o projeto SAP tem apenas um arquivo chamado **App.cs**, mas é realmente como se o projeto SAP não existisse e que, em vez disso, havia 5 cópias diferentes desse arquivo, uma em cada projeto de aplicativo.

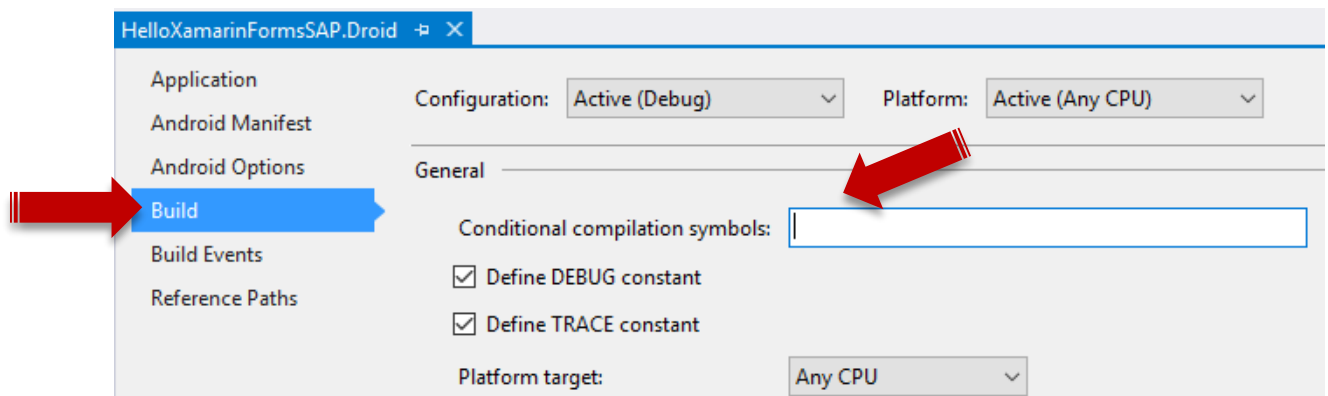
Alguns problemas sutis (e não tão sutis) podem manifestar-se com a abordagem da biblioteca compartilhada, por exemplo, projetos Android e iOS têm acesso a mais ou menos a mesma versão do .NET, mas não é a mesma versão do .NET que usam os projetos do Windows. Isto significa que qualquer classe .NET que o código partilhado acessa poderia ser um pouco diferente dependendo da plataforma. Este é o caso de algumas classes de espaço de E/S de arquivos no espaço de nomes System.IO.

Nós podemos compensar isso através do uso de diretivas de compilador do C#, em particular a diretiva `#if` e `#elif`. Os projetos gerados pelo modelo Xamarin.Forms, os vários projetos de implementação definem símbolos que podem ser usados com estas diretivas. Vamos ver o que esses símbolos são.

2. Selecione a opção **Properties** do menu de contexto do projeto Android. A folha de propriedades do projeto será exibida.

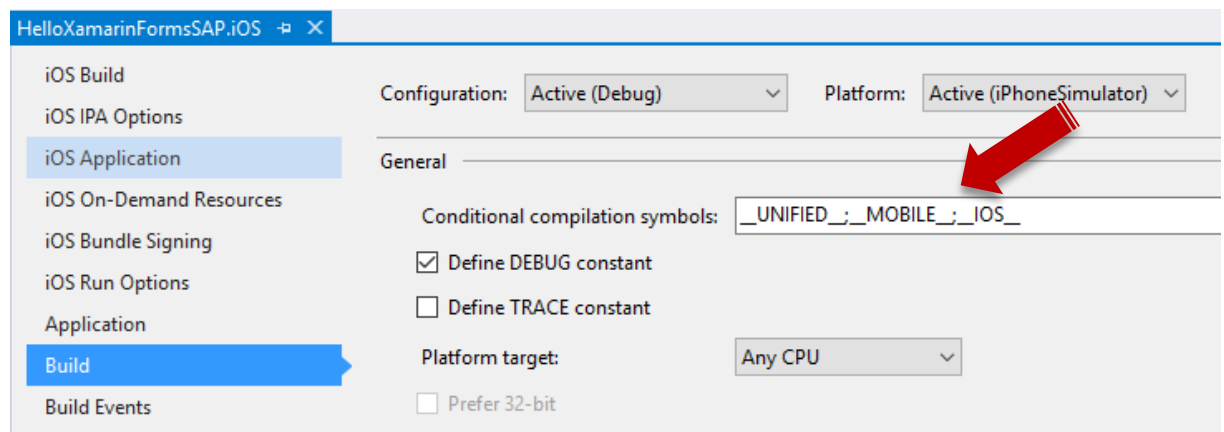


3. Selecione a opção **Build** e observe o valor do campo **Conditional compilation symbols**. Poderá notar que o campo aparece vazio, no entanto, o identificador `__ANDROID__` e vários identificadores `__ANDROID_nn__` (onde `nn` representa cada nível de API Android suportada) são definidos.



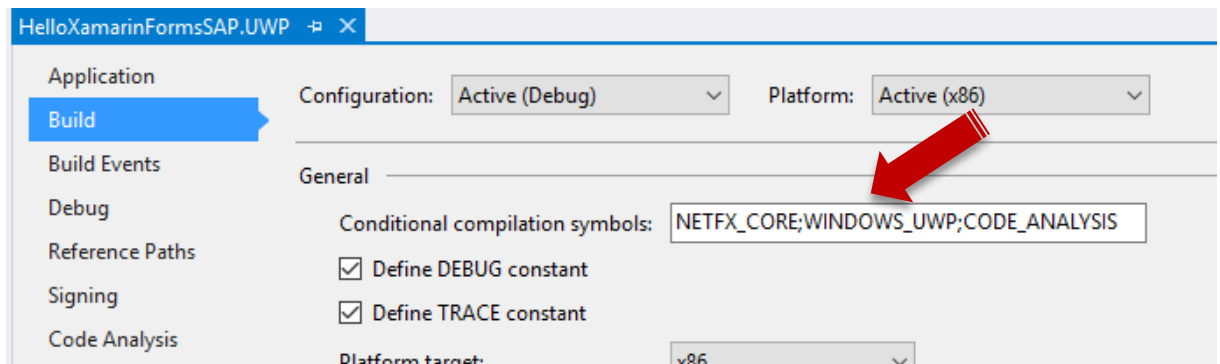
4. Selecione a opção **Properties** do menu contextual do projetos iOS. A folha de propriedades do projeto será exibida.
5. Selecione a opção **Build** e observe o valor do campo **Conditional compilation symbols**. Você notará a lista de 3 símbolos definidos que são separados por ponto e vírgula:

- \_\_UNIFIED\_\_
- \_\_MOBILE\_\_
- \_\_IOS\_\_



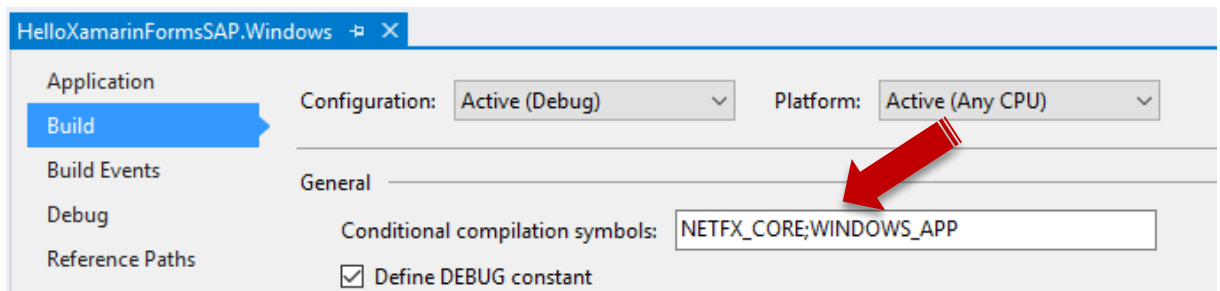
6. Selecione a opção **Properties** do menu contextual do projeto UWP. A folha de propriedades do projeto será exibida.
7. Selecione a opção **Build** e observe o valor do campo **Conditional compilation symbols**. Você pode observar os seguintes símbolos:

- NETFX\_CORE
- WINDOWS\_UWP
- CODE\_ANALYSIS



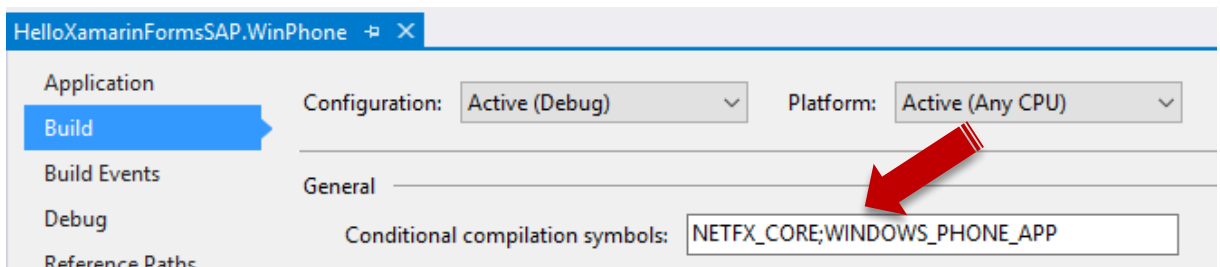
8. Selecione a opção **Properties** do menu contextual do projeto Windows. A folha de propriedades do projeto será exibida.
9. Selecione a opção **Build** e observe o valor do campo **Conditional compilation symbols**. Você pode observar os seguintes símbolos:

- NETFX\_CORE
- WINDOWS\_APP



10. Selecione a opção **Properties** o menu de contexto do projeto do Windows Phone. A folha de propriedades do projeto será exibida.
11. Selecione a opção **Build** e observe o valor do campo **Conditional compilation symbols**. Você pode observar os seguintes símbolos:

- NETFX\_CORE
- WINDOWS\_PHONE\_APP



12. Abra o arquivo **App.cs** do projeto SAP.

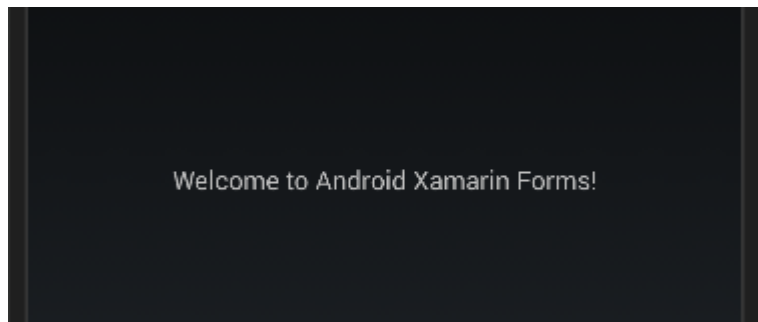
13. Substitua a linha

```
Text = "Welcome to Xamarin Forms!"
```

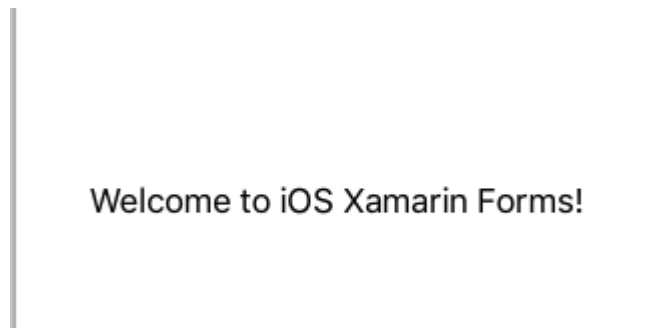
pelo seguinte código:

```
#if __IOS__
    Text = "Welcome to iOS Xamarin Forms!"
#elif __ANDROID__
    Text = "Welcome to Android Xamarin Forms!"
#elif WINDOWS_UWP
    Text = "Welcome to Windows UWP Xamarin Forms!"
#elif WINDOWS_APP
    Text = "Welcome to Windows Xamarin Forms!"
#elif WINDOWS_PHONE_APP
    Text = "Welcome to Windows Phone Xamarin Forms!"
#endif
```

14. Implante o aplicativo em alguns emulador Android. Você vai ver que a mensagem apropriada é exibida para Android.



15. Implante o aplicativo em qualquer emulador iOS. Você vai ver que a mensagem apropriada é exibida para iOS.



16. Implante o aplicativo em um emulador Windows 10 Mobile. Você verá a mensagem apropriada para UWP mostrado.



17. Implante o aplicativo do Windows 8.1 no simulador. Você verá a mensagem apropriada para o aplicativo do Windows é mostrado.



18. Exibe o aplicativo do Windows Phone em um emulador. Você verá a mensagem apropriada para o aplicativo do Windows Phone.



Como você deve ter notado, pela diretivas de compilação C#, arquivos de codeshare podem executar código específico para cada plataforma ou acessar as classes específicas de cada plataforma, incluindo classes por projetos individuais de cada plataforma. Você também pode definir seus próprios símbolos de compilação condicional, se desejar.

Diretivas do compilador são sem sentido em projetos PCL. Um projeto de PCL é totalmente independente das 5 plataformas como identificadores em cada projeto de plataformas não estão presentes no momento em que o projeto é compilado em PCL.

O conceito de PCL originalmente surgiu porque cada plataforma que usa .NET, hoje, utiliza um subconjunto diferente de .NET. Se você deseja criar uma biblioteca que possa ser utilizada por diferentes plataformas .NET, precisa usar apenas as partes comuns dos subconjuntos .NET.

Uma PCL é projetada para ser de ajuda para conter código que pode ser usado em vários (mas específicas) plataformas .NET. Consequentemente, uma determinada PCL contém uma espécie de

bandeiras incorporados indicando plataformas de apoio. A PCL usada em uma aplicação Xamarin.Forms deve suportar as seguintes plataformas:

- .NET Framework 4.5
- Windows 8
- Windows Phone 8.1
- Xamarin.Android
- Xamarin.iOS
- Xamarin.iOS (Classic)

Se precisarmos de um comportamento específico de uma plataforma na PCL, não poderíamos usar as diretivas de compilação de C#, porque estes só agem em tempo de compilação. Precisamos de algo que funciona em tempo de execução, como a classe `Device` de Xamarin.Forms.

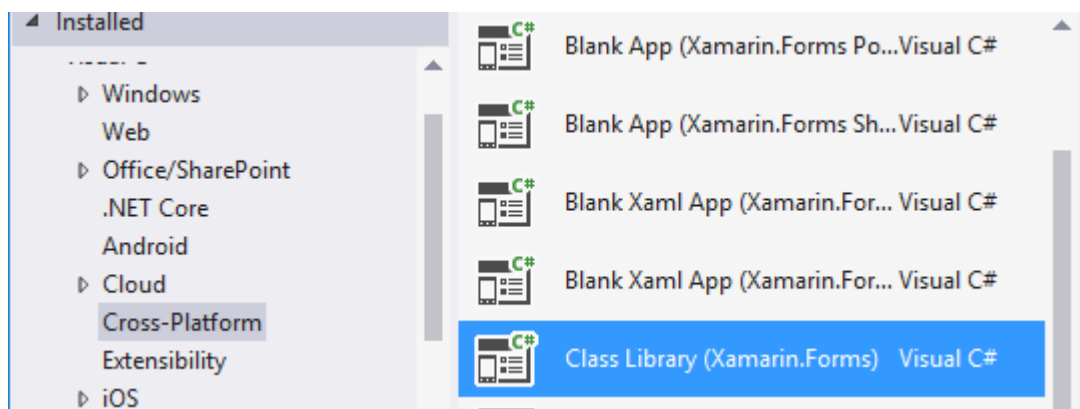
Uma PCL Xamarin.Forms pode acessar outras PCLs que suportem as mesmas plataformas, no entanto, você não pode acessar diretamente as classes definidas em projetos de aplicativos individuais. Deveríamos exigir esta funcionalidade, Xamarin.Forms fornece uma classe chamada **DependencyService** que nos permite acessar um código específico da plataforma de uma PCL seguindo uma metodologia.

## PCL ou SAP?

PCL é a estratégia recomendada para Xamarin.Forms e é a preferida por muitos programadores que trabalham com Xamarin.Forms há muito tempo. No entanto, a estratégia SAP também é suportado e também tem os seus seguidores.

## Qual escolher?

É possível ter ambos os tipos de projetos na mesma solução Xamarin.Forms. Se nós criamos uma solução Xamarin.Forms com um projeto SAP, podemos adicionar um novo projeto PCL e a solução selecionando o modelo **Class Library (Xamarin.Forms)**.



Os projetos e aplicações podem acessar tanto o projeto SAP como o projeto PCL. O projeto PCL também pode acessar os arquivos do projeto SAP.



## Exercício 2: Utilizando o View Label para mostrar o texto.

---

Neste exercício, vamos explorar o **View Label** para mostrar texto em uma aplicação multiplataforma desenvolvida em Xamarin.Forms.

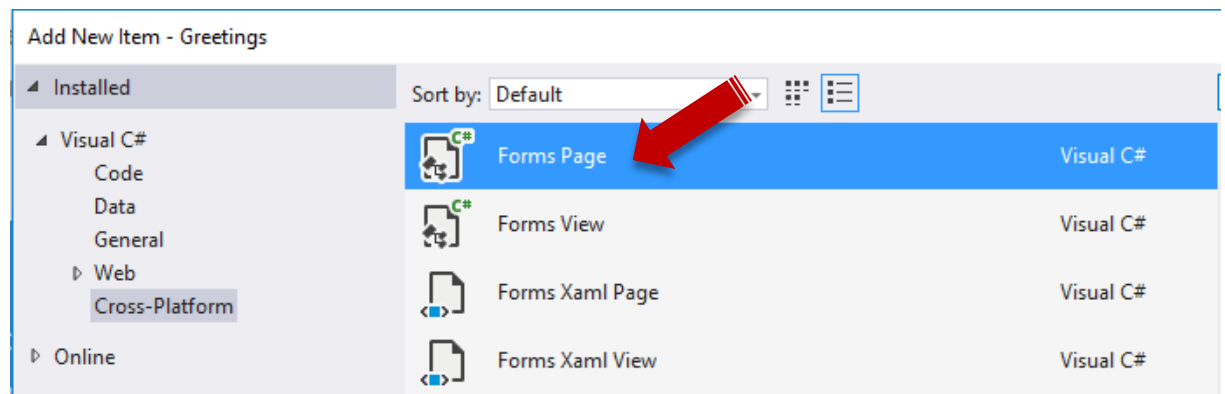
### Tarefa 1. Criar uma solução Xamarin.Forms PCL.

Nesta tarefa criaremos uma nova solução Xamarin.Forms PCL chamada **Greetings** utilizando o mesmo processo descrito anteriormente para criar uma solução Xamarin.Forms utilizando o modelo **Blank App (Xamarin.Forms.Portable)**.

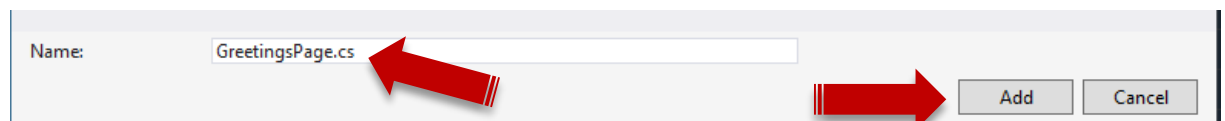
1. Crie uma nova aplicação multiplataforma chamada **Greetings** utilizando o modelo **Blank App (Xamarin.Forms.Portable)**.
2. Esta solução será estruturada como um programa típico Xamarin.Forms, o que significa que definirá uma nova classe resultante de **ContentPage**. Portanto, devemos agregar um novo arquivo ao projeto **Greetings**.

Selecione **Add > New Item** no menu contextual do projeto PCL Greetings.

3. Na janela **Add New Item**, selecione a janela **Forms Page**.



4. Coloque o nome **GreetingsPage.cs** e clique em **Add** para adicionar o novo elemento.



5. Observe o código da classe **GreetingsPage**. Você notará que a classe deriva de **ContentPage**. Como a classe **ContentPage** se encontra no espaço de nomes Xamarin.Forms, podemos observar que uma diretiva **using** inclui esse espaço de nomes.

A classe é definida como pública, mas não é realmente necessário, pois não pode ser acessada diretamente fora do projeto Greetings.

```
public class GreetingsPage : ContentPage
{
    public GreetingsPage()
    {
        Content = new StackLayout
        {
            Children = {
                new Label { Text = "Hello Page" }
            }
        };
    }
}
```

É na classe `GreetingsPage` (e outras como esta) onde investirá grande parte de seu tempo no início de programação com Xamarin.Forms. Para programas de uma única página, esta classe pode conter o código exclusivo do aplicativo que você precisa para escrever. Claro, poderíamos agregar classes adicionais para o projeto se precisarmos dela também.

Tradicionalmente, os programas de uma única página, recomenda-se que a classe que deriva de **ContentPage** tem o mesmo nome do aplicativo, mas com o sufixo **Page**. Essa convenção de nomenclatura é sugerido, mas não obrigatória.

6. Elimina todo o código do construtor **GreetingsPage** e as diretivas **using** para que o arquivo se pareça com o seguinte.

```
using System;
using Xamarin.Forms;

namespace Greetings
{
    public class GreetingsPage : ContentPage
    {
        public GreetingsPage()
        {
        }
    }
}
```

7. Adicione o seguinte código dentro do construtor para instanciar um **Label**, defina sua propriedade **Text** e atribuir a ocorrência **Label** na propriedade **Content** que **GreetingsPage** herda de **ContentPage**.

```
public GreetingsPage()
{
    var MyLabel = new Label();
    MyLabel.Text = "Greetings, Xamarin.Forms!";
}
```

```
        this.Content = MyLabel;
    }
```

8. Abra o arquivo **App.cs** do projeto PCL.
9. Elimine o código do construtor. O código da classe será similar a seguinte.

```
public class App : Application
{
    public App()
    {
    }

    protected override void OnStart()
    {
        // Handle when your app starts
    }

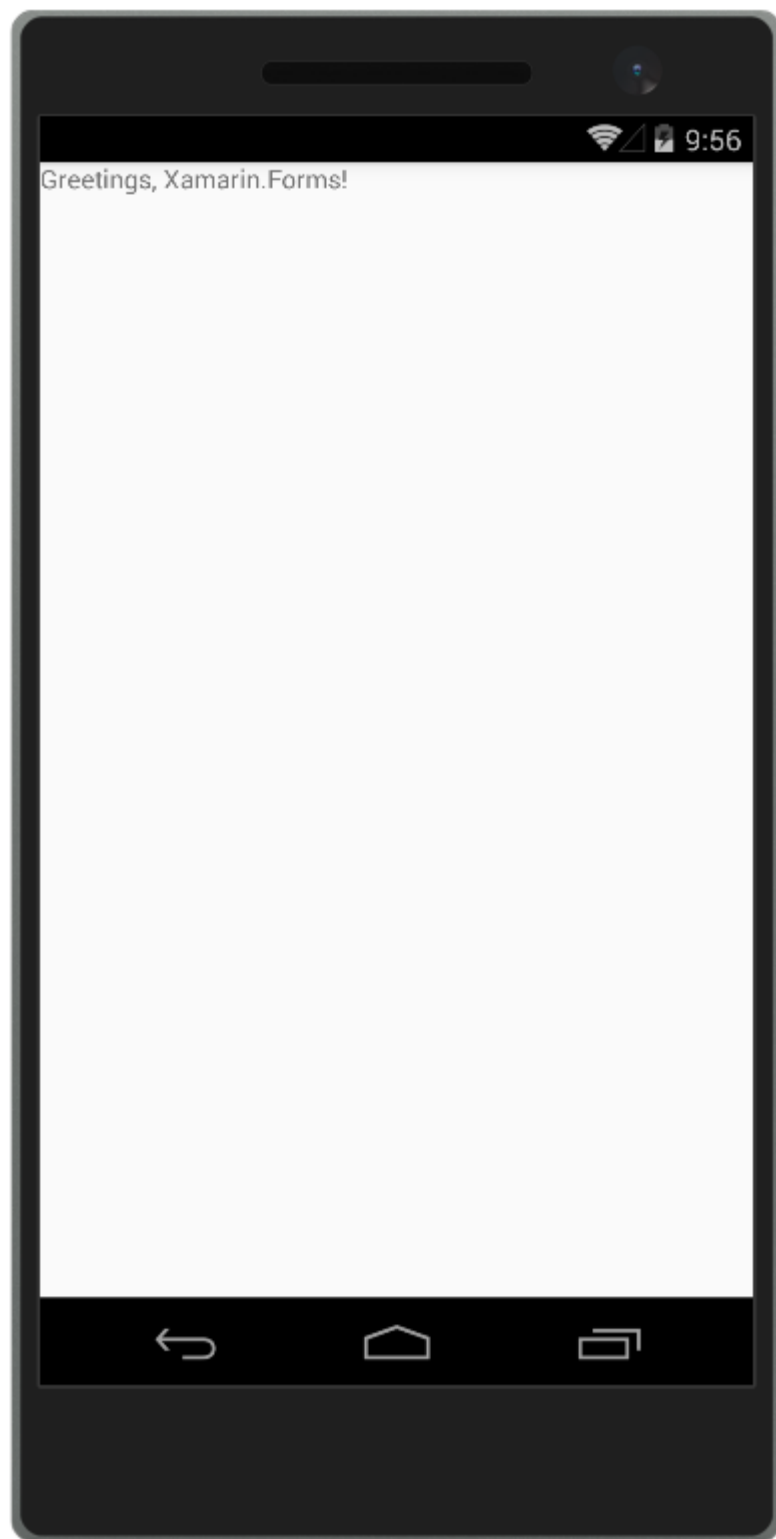
    protected override void OnSleep()
    {
        // Handle when your app sleeps
    }

    protected override void OnResume()
    {
        // Handle when your app resumes
    }
}
```

10. Adicione o seguinte código dentro do construtor para estabelecer a propriedade **MainPage** o valor de uma instância da classe **GreetingsPage**.

```
public App()
{
    MainPage = new GreetingsPage();
}
```

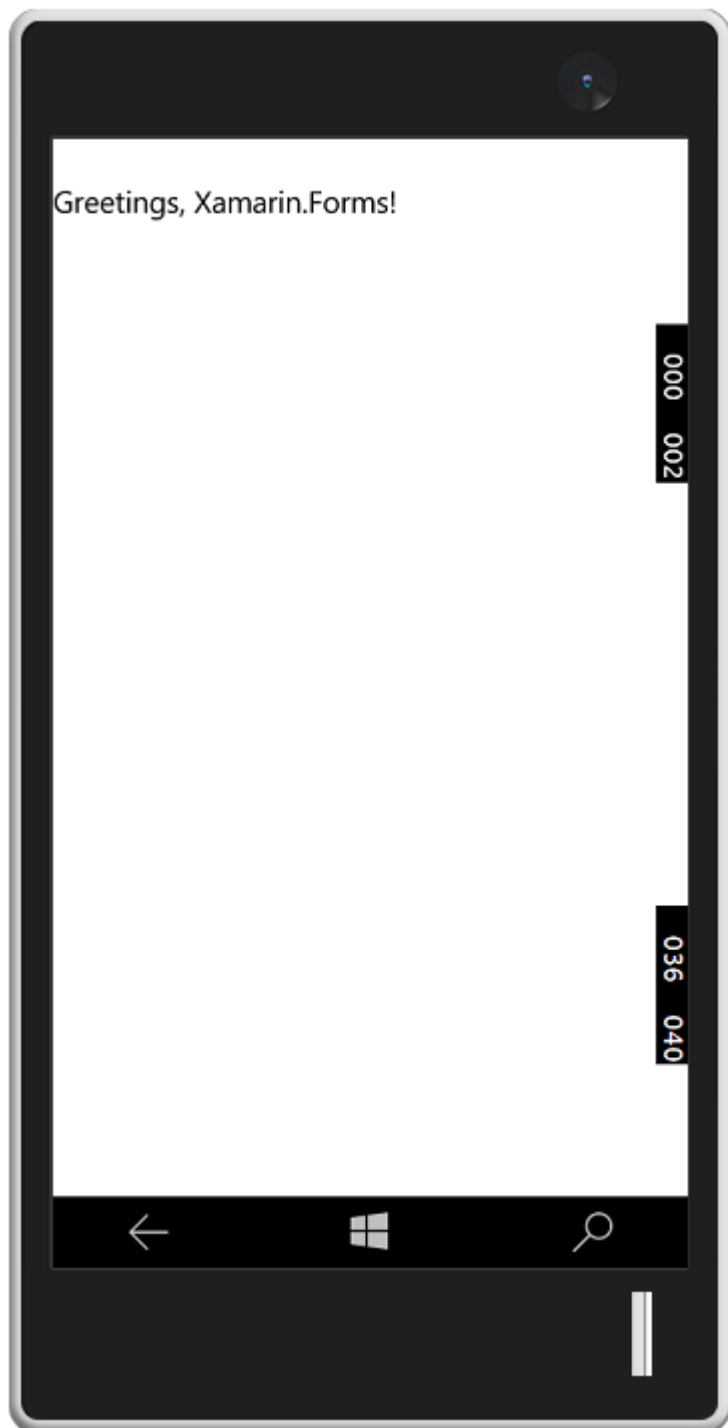
11. Compile e execute a aplicação em emuladores Android, iOS e Windows 10 Mobile.  
As imagens seguintes mostram a aplicação nos 3 emuladores.



**Android**



iOS

**Windows 10 Mobile**

Definitivamente a versão mais decepcionante desta aplicação Greetings é a versão iOS. Você pode achar que a aplicação divide a tela com a barra de status na parte superior. Qualquer coisa que a aplicação mostre na parte superior da página ocupará o mesmo espaço que a barra de status, ao menos que a aplicação faça algo para evitar isso.

Como posso resolver este problema? Há 4 maneiras de resolver este problema (ou 5, se usarmos um projeto SAP).

## Tarefa 2. Incluir espaço (preenchimento) na página (Solução 1).

A classe **Page** define uma propriedade chamada **Padding** que marca uma área em torno do perímetro interno da página e onde o conteúdo não pode ser localizado. A propriedade **Padding** é do tipo **Thickness**, uma estrutura que define quatro propriedades chamadas **Left**, **Top**, **Right** e **Bottom**. É recomendado lembrar essa ordem, pois é a ordem que definiremos as propriedades no construtor da classe **Thickness** assim como em XAML. A estrutura **Thickness** também define construtores para estabelecer a mesma quantidade de espaço em 4 lados ou para estabelecer a mesma quantidade de espaço no lado esquerdo e direito ou superior e inferior.

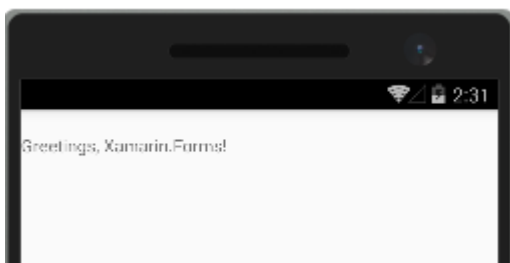
Com uma pequena pesquisa na Internet podemos encontrar que a barra de status do iOS tem uma altura de 20 unidades.

1. Adicione o seguinte código ao construtor da classe **GreetingsPage** para deixar um espaço de 20 unidades no topo da página.

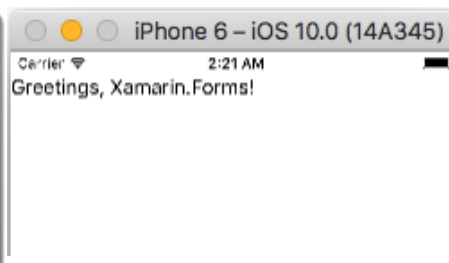
```
public GreetingsPage()
{
    var MyLabel = new Label();
    MyLabel.Text = "Greetings, Xamarin.Forms!";
    this.Content = MyLabel;

    Padding = new Thickness(0, 20, 0, 0);
}
```

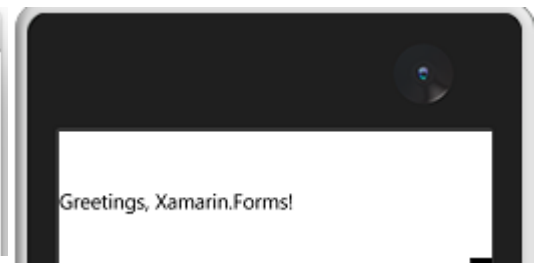
2. Execute o aplicativo nos 3 emuladores para ver o resultado.



**Android**



**iOS**



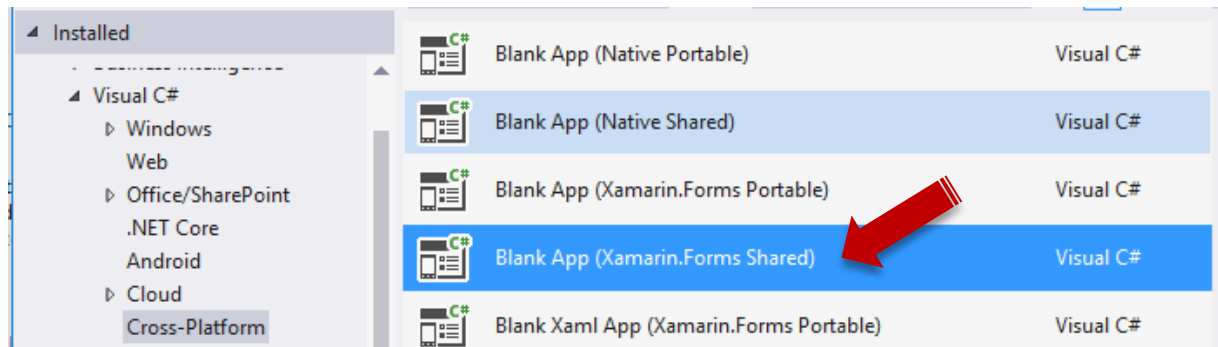
**Windows 10 Mobile**

Você pode notar que quando definir o valor na propriedade **Padding** de **ContentPage** resolverá o problema da sobreposição de texto com a barra de status do iOS, mas também fornece o mesmo espaço não é necessário no Android e Windows Mobile. Existe maneira de estabelecer o espaço apenas no iPhone?

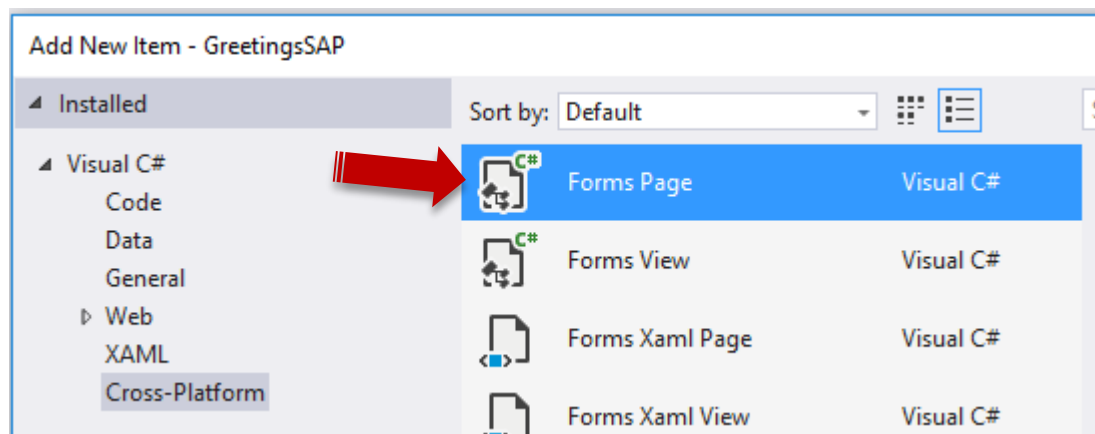
**Tarefa 3. Incluir espaço (preenchimento) para iOS apenas em projetos SAP (Solução 2).**

Uma das vantagens da utilização de projetos SAP é que as classes no projeto são extensões dos projetos das aplicações, portanto, podemos usar diretivas de compilação condicional.

1. Abra uma nova instância do Visual Studio.
2. Crie o projeto **GreetingsSAP** utilizando o modelo **Blank App (Xamarin.Forms Shared)**.



3. Selecione **Add > New Item** no menu contextual do projeto SAP.
4. Na janela **Add New Item**, selecione o modelo **Forms Page**.



5. Escreva o nome **GreetingsSAPPage.cs** e clique em **Add** para adicionar o novo elemento.
6. Modifique o código do arquivo **GreetingsSAPPage.cs** para que seja semelhante à classe **Greetings** do projeto PCL.

```
using System;
using Xamarin.Forms;

namespace GreetingsSAP
{
    public class GreetingsSAPPage : ContentPage
    {
        public GreetingsSAPPage ()
        {
            var MyLabel = new Label();
        }
    }
}
```



```
        MyLabel.Text = "Greetings, Xamarin.Forms!";  
        this.Content = MyLabel;  
  
        Padding = new Thickness(0, 20, 0, 0);  
    }  
}
```

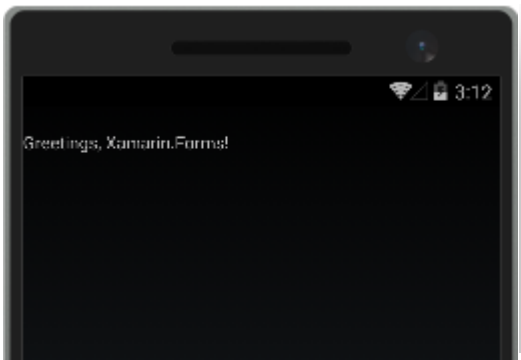
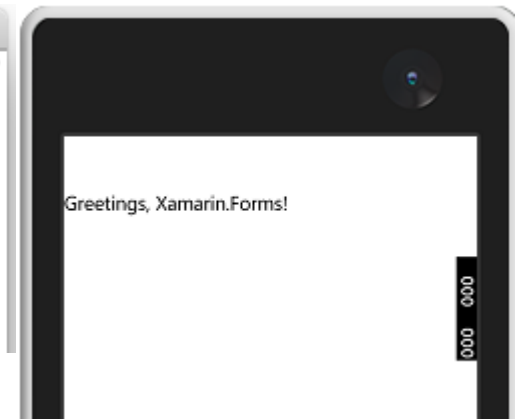
7. Abra o arquivo **App.cs** do projeto SAP.
8. Elimine o código de construtor. O código de classe será parecido com este.

```
public class App : Application  
{  
    public App()  
    {  
    }  
  
    protected override void OnStart()  
    {  
        // Handle when your app starts  
    }  
  
    protected override void OnSleep()  
    {  
        // Handle when your app sleeps  
    }  
  
    protected override void OnResume()  
    {  
        // Handle when your app resumes  
    }  
}
```

9. Adicione o seguinte código dentro do construtor para estabelecer a propriedade **MainPage** o valor de uma instância da classe **GreetingsSAPPage**.

```
public App()  
{  
    MainPage = new GreetingsSAPPage();  
}
```

10. Compile e execute a aplicação nos emuladores Android, iOS e Windows 10 Mobile. As imagens seguintes mostram a aplicação nos 3 emuladores.

**Android****iOS****Windows 10 Mobile**

Neste momento, ambas soluções PCL e SAP têm o mesmo problema que adiciona espaço na parte superior da página no Android e Windows Mobile.

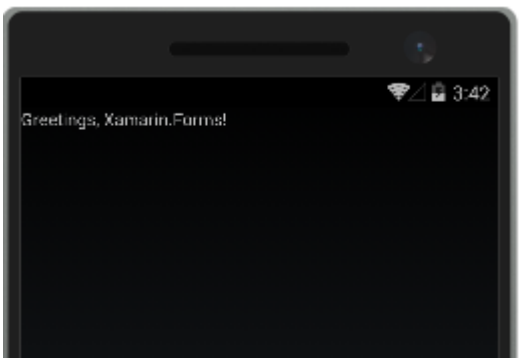
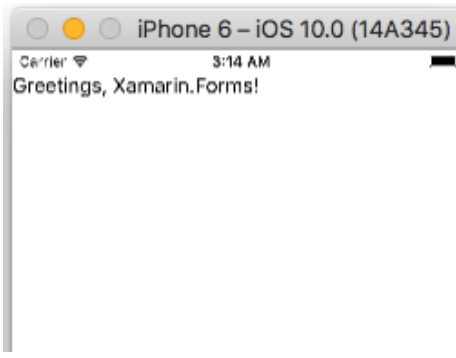
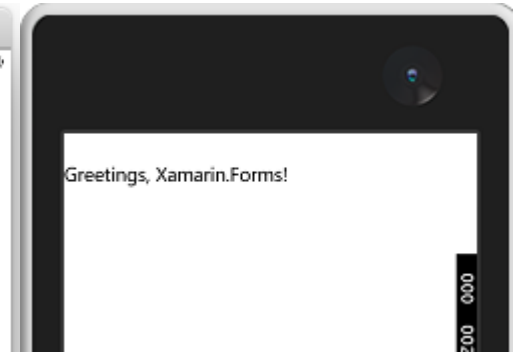
11. Modifique o código do construtor da classe **GreetingsSAPPage** para incorporar uma diretiva de compilação que faça com que o código que define o valor a propriedade **Padding** seja apenas compilado no projeto iOS. O código será similar ao seguinte.

```
public GreetingsSAPPage ()
{
    var MyLabel = new Label();
    MyLabel.Text = "Greetings, Xamarin.Forms!";
    this.Content = MyLabel;

    #if __IOS__
        Padding = new Thickness(0, 20, 0, 0);
    #endif
}
```

12. Compile e execute a aplicação nos emuladores Android, iOS e Windows 10 Mobile.

As imagens seguintes mostram a aplicação nos 3 emuladores.

**Android****iOS****Windows 10 Mobile**

Observe que, como a diretiva **#if** refere-se ao símbolo de compilação condicional `__IOS__`, a propriedade **Padding** é definida somente para o projeto iOS.

No entanto, estes símbolos de compilação condicional afetam apenas a compilação do programa, por isso que eles não têm efeito sobre uma solução PCL. Existe alguma maneira para que um projeto de PCL possa incluir diferentes valores **Padding** para diferentes plataformas?

#### **Tarefa 4. Incluir espaço (preenchimento) apenas para iOS em projetos PCL ou SAP (Solução 3).**

A classe estática **Device**, inclui várias propriedades e métodos que permitem que o código resolva as diferenças entre dispositivos em tempo de execução em uma maneira fácil e simples.

- A propriedade **Device.OS** retorna um membro da enumeração **TargetPlatform** que indica o tipo de sistema operacional que está trabalhando: Android, iOS, Other, Windows ou WinPhone.
- A propriedade **Device.Idiom** retorna um membro da enumeração **TargetIdiom** que indica o tipo de dispositivo que está trabalhando: Desktop, Phone, Tablet ou Unsupported.

É possível utilizar essas duas propriedades em instruções **if** e **else** ou em um bloco **switch** para executar um código específico de uma plataforma particular.

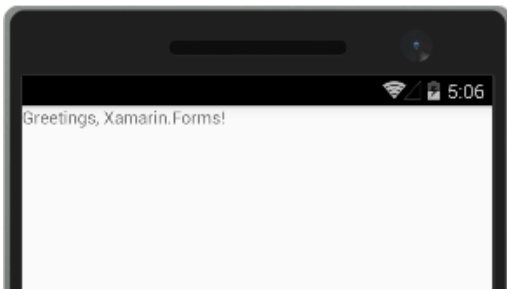
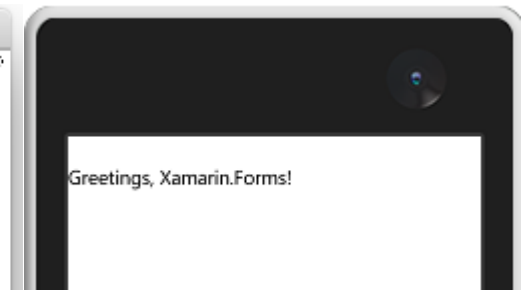
Dois métodos chamados **OnPlatform** proporcionam soluções mais elegantes:

- O método genérico estático **Device.OnPlatform<T>** leva 3 argumentos de tipo T, o primeiro para iOS, o segundo para Android e o terceiro para plataformas Windows.
  - O segundo método estático **Device.OnPlatform** tem 4 argumentos de tipo **Action** também na ordem iOS, Android e Windows com um 4º argumento para um valor padrão.
1. Abra o arquivo **GreetingsPage** do projeto PCL.
  2. Substitua a linha de código `Padding = new Thickness(0, 20, 0, 0);` pelo seguinte código para definir o valor `Padding` para os dispositivos iPhone.

```
Padding = Device.OnPlatform<Thickness>(  
    new Thickness(0, 20, 0, 0),  
    new Thickness(0),  
    new Thickness(0));
```

O primeiro argumento **Thickness** é para iOS, o Segundo é para Android e o terceiro é para Windows.

13. Compile e execute a aplicação em emuladores Android, iOS e Windows 10 Mobile.  
As imagens seguintes mostram a aplicação nos 3 emuladores.

**Android****iOS****Windows 10 Mobile**

Pode notar que o **Padding** foi aplicado apenas ao dispositivo iOS.

O código anterior também poderia ser da seguinte maneira. Por quê?

```
Padding = new Thickness(0, Device.OnPlatform(20, 0, 0), 0, 0);
```

Podemos até mesmo usar a versão de **Device.OnPlatform** com argumentos **Action**.

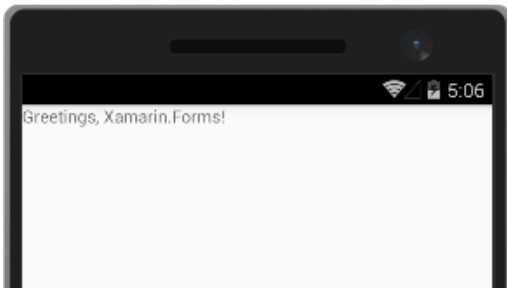
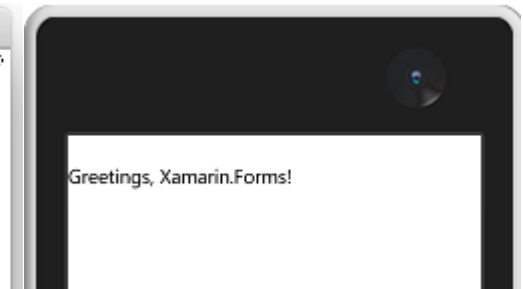
3. Modificar a linha de código anterior com o seguinte.

```
Device.OnPlatform(() =>
{
    Padding = new Thickness(0, 20, 0, 0);
});
```

Os argumentos do método **OnPlatform** são **null** por padrão, então podemos estabelecer apenas o valor do primeiro argumento a ser processado em iOS, como mostrado no código anterior.

14. Compile e execute a aplicação em emuladores Android, iOS e Windows 10 Mobile.

As imagens seguintes mostram a aplicação nos 3 emuladores.

**Android****iOS****Windows 10 Mobile**

Você pode notar que o aplicativo funciona como esperado porque a instrução que define o **Padding** é executado apenas quando a aplicação está em execução no iOS. É claro que através da criação de um único argumento no método **Device.OnPlatform** faria o código compreensível para algumas pessoas que precisam de lê-lo. Algo que poderia ajudar a tornar

o código mais claro é definir o nome do parâmetro que precede o argumento como no exemplo a seguir.

```
Device.OnPlatform(iOS: () =>
{
    Padding = new Thickness(0, 20, 0, 0);
});
```

A sintaxe acima torna explícito que a instrução é executada apenas no iOS.

O método **Device.OnPlatform** é muito conveniente e tem a vantagem de trabalhar em ambos os projetos PCL e SAP. No entanto, você não pode acessar as APIs dentro de plataformas individuais. Para fazer isso vamos precisar **DependencyService**.

### Tarefa 5. Centralizar o Label dentro da página (Solução 4).

O problema do texto sobre a barra de status do iOS ocorre apenas porque a apresentação padrão do texto é feita na parte superior esquerda. É possível centralizar o texto na página?

Xamarin.Forms suporta vários recursos que facilitam o projeto sem a necessidade de o programa para realizar cálculos complexos que envolvem tamanhos e coordenadas. A classe **View** define duas propriedades chamadas **HorizontalOptions** e **VerticalOptions** que especificam como o **View** deve ser posicionado em relação ao seu recipiente (neste caso **ContentPage**). Estas duas propriedades estão tipo **LayoutOptions**, uma estrutura muito importante em Xamarin.Forms.

Geralmente utilizamos a estrutura **LayoutOptions** especificando um dos 8 campos públicos estáticos que apenas definem leitura:

- Center
- CenterAndExpand
- End
- EndAndExpand
- Fill
- FillAndExpand
- Start
- StartAndExpand

A estrutura **LayoutOptions** também define duas propriedades de instância que nos permitem criar valor com as mesmas combinações:

- Uma propriedade **Alignment** de tipo **LayoutAlignment** que é uma enumeração com 4 membros: **Center**, **End**, **Fill** e **Start**.
- Uma propriedade **Expands** de tipo **bool**.

Para a propriedade **HorizontalOptions**, a palavra **Start** significa esquerda e **End** significa direita.

Para a propriedade **VerticalOptions**, a palavra **Start** significa acima e **End** significa abaixo.



Para mais informações sobre a estrutura **LayoutOptions** você pode consultar o seguinte link:

**Xamarin.Forms.LayoutOptions Structure**

<https://developer.xamarin.com/api/type/Xamarin.Forms.LayoutOptions/>

Vamos ver como posicionar a **Label** no centro da página.

1. Modifique o código do construtor da classe **GreetingsPage** da seguinte forma.

```
public class GreetingsPage : ContentPage
{
    public GreetingsPage()
    {
        var MyLabel = new Label();
        MyLabel.Text = "Greetings, Xamarin.Forms!";
        this.Content = MyLabel;

        MyLabel.HorizontalOptions = LayoutOptions.Center;
        MyLabel.VerticalOptions = LayoutOptions.Center;
    }
}
```

2. Compile e execute a aplicação nos emuladores Android, iOS e Windows 10 Mobile. Você vai notar que o texto aparece centralizado na página.

As imagens seguintes mostram a aplicação nos 3 emuladores.

**Android****iOS**



**Windows 10 Mobile**

**Tarefa 6. Centralizar o texto dentro da Label (Solução 5).**

O View **Label** foi projetado para exibir o texto em uma linha ou parágrafo. Muitas vezes, é necessário controlar a forma como as linhas de texto são alinhadas horizontalmente. O View **Label** define a propriedade **HorizontalTextAlignment** para esse fim e também define a propriedade **VerticalTextAlignment** para posicionar o texto verticalmente. Ambas as propriedades são definidas para o valor de um membro da enumeração **TextAlignment** que tem os membros **Start**, **Center** e **End**.



Para esta última solução ao problema da barra de status de iOS vamos configurar os valores das propriedades **HorizontalTextAlignment** e **VerticalTextAlignment** do View **Label**.

1. Modifique o código do construtor da classe **GreetingsPage** da seguinte maneira.

```
public class GreetingsPage : ContentPage
{
    public GreetingsPage()
    {
        var MyLabel = new Label();
        MyLabel.Text = "Greetings, Xamarin.Forms!";
        this.Content = MyLabel;

        MyLabel.HorizontalTextAlignment = TextAlignment.Center;
        MyLabel.VerticalTextAlignment = TextAlignment.Center;
    }
}
```

2. Compile e execute a aplicação nos emuladores Android, iOS e Windows 10 Mobile. Você vai notar que o texto aparece centralizado na página.

Visualmente, o resultado com apenas uma linha de texto é o mesmo que centralizar com as propriedades **HorizontalOptions** e **VerticalOptions** no entanto essas técnicas são ligeiramente diferentes.

## Resumo

---

Neste laboratório você analisou as diferentes opções para compartilhar código em uma aplicação multiplataforma e explorou o View **Label** para exibir texto em uma aplicação multiplataforma desenvolvida com Xamarin.Forms. Conheceu também algumas das alternativas disponíveis ao exibir texto para ter a mesma aparência nas distintas plataformas.

É momento de começar a trabalhar com mais elementos para construir a interface de usuário em uma aplicação Xamarin.Forms.

Quando tiver terminado este laboratório, publique a seguinte mensagem no Twitter e Facebook:

***Acabei o #Lab03 da #MaratonaXamarin e conheço as opções para compartilhar código em Xamarin!***