

Laboratorio

Compartiendo código entre proyectos multiplataforma

Versión: 1.0.0
Octubre de 2016



[Miguel Muñoz Serafín](#)

@msmdotnet



CONTENIDO

INTRODUCCIÓN

EJERCICIO 1: EXAMINANDO LAS OPCIONES PARA COMPARTIR CÓDIGO

Tarea 1. Crear una aplicación Xamarin.Forms con la plantilla SAP.

Tarea 2. Examinar los archivos de código generados.

EJERCICIO 2: UTILIZANDO EL VIEW LABEL PARA MOSTRAR TEXTO

Tarea 1. Crear una solución Xamarin.Forms PCL.

Tarea 2. Incluir espacio (padding) en la página (Solución 1).

Tarea 3. Incluir espacio (padding) solo para iOS en proyectos SAP (Solución 2).

Tarea 4. Incluir espacio (padding) únicamente para iOS en proyectos PCL o SAP (Solución 3).

Tarea 5. Centrar el **Label** dentro de la página (Solución 4).

Tarea 6. Centrar el texto dentro del **Label** (Solución 5).

RESUMEN

Introducción

Cuando creaste la solución **HelloXamarinForms** en Visual Studio, tuviste la opción de seleccionar 2 principales tipos de plantillas:

- Xamarin.Forms Portable
- Xamarin.Forms Shared

La primera opción crea una Biblioteca de Clases portable (PCL) mientras que la segunda opción crea un Proyecto de Recursos Portable (SAP) consistente únicamente de archivos de código compartido.

En este laboratorio, analizaremos las distintas opciones para compartir código en una aplicación multiplataforma y exploraremos el *View **Label*** con las consideraciones que debemos tomar al momento de mostrar texto en una aplicación multiplataforma desarrollada con Xamarin.Forms.

Objetivos

Al finalizar este laboratorio, los participantes serán capaces de:

- Describir la manera de compartir código a través de proyectos PCL.
- Describir la manera de compartir código a través de proyectos SAP.
- Utilizar el elemento **Label** para mostrar texto en una aplicación Xamarin.Forms.

Requisitos

Para la realización de este laboratorio es necesario contar con lo siguiente:

- Un equipo de desarrollo con sistema operativo Windows 10 y Visual Studio 2015 Community, Professional o Enterprise con la plataforma Xamarin.
- Un equipo Mac con la plataforma Xamarin.

Tiempo estimado para completar este laboratorio: **60 minutos**.

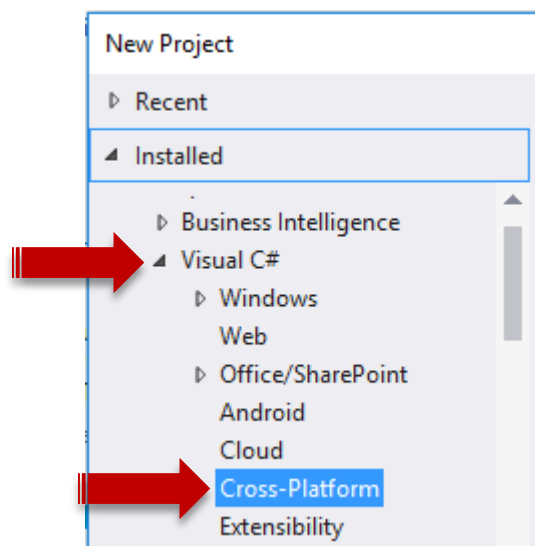
Ejercicio 1: Examinando las opciones para compartir código.

En este ejercicio tendrás la oportunidad de explorar las ventajas y desventajas de las dos opciones disponibles para compartir código en una aplicación multiplataforma Xamarin.Forms.

Tarea 1. Crear una aplicación Xamarin.Forms con la plantilla SAP.

La solución creada en el laboratorio anterior utilizó la plantilla PCL. Ahora es momento de crear una segunda solución utilizando la plantilla SAP.

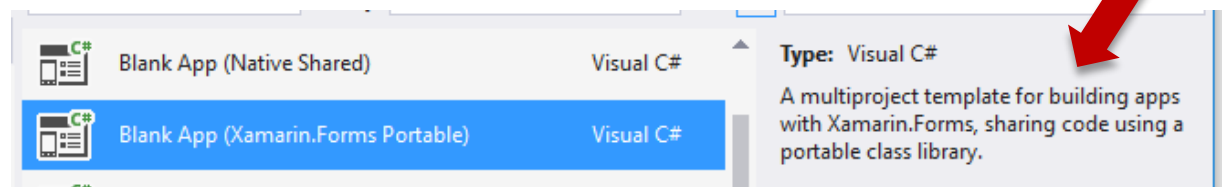
1. Selecciona la opción **File > New > Project** desde Visual Studio.
2. En el panel izquierdo de la ventana **New Project** selecciona **Visual C# > Cross-Platform**.



3. En el panel central de la ventana **New Project** puedes ver distintas plantillas de solución disponibles incluyendo 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)



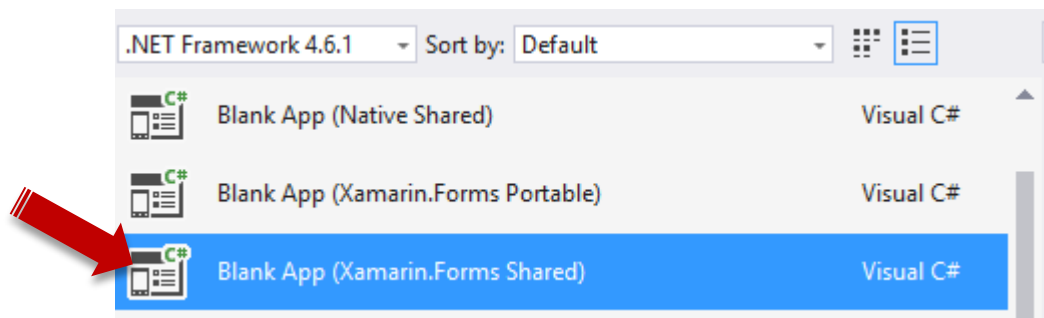
Haz clic en cada una de las plantillas para ver su descripción en el panel de la derecha.



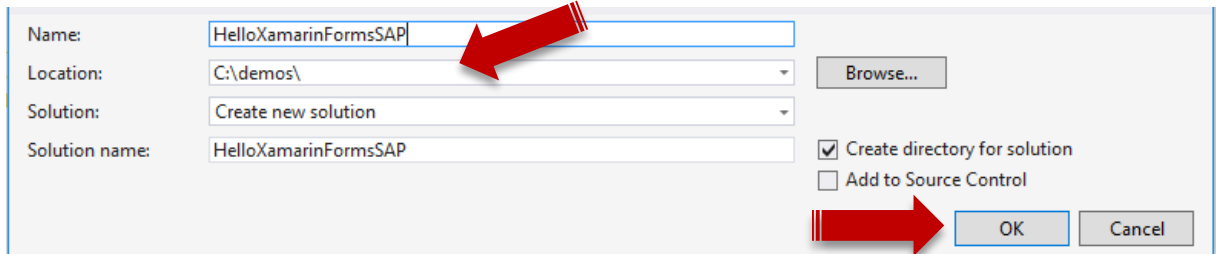
El término “**Portable**” en este contexto se refiere a una Biblioteca de Clases Portable (Portable Class Library – PCL). Todo el código común de la aplicación se convierte en una biblioteca DLL que es referenciada por todos los proyectos de plataforma individuales.

El término “**Shared**” en este contexto se refiere a un Proyecto de Recursos Compartidos (Shared Asset Project – SAP). Este proyecto contiene archivos de código y otro tipo de archivos que son compartidos con los proyectos de cada plataforma, volviéndose esencialmente parte de cada proyecto de cada plataforma.

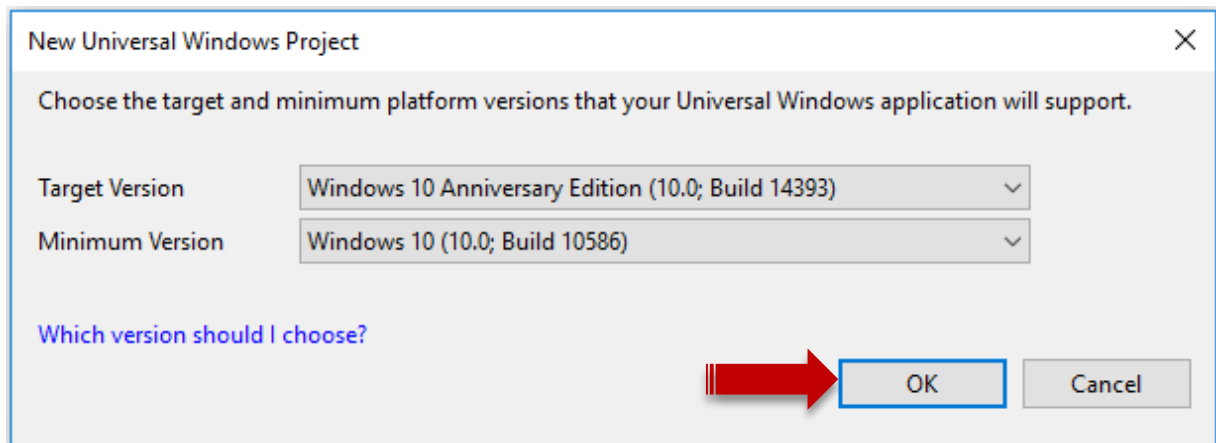
4. Selecciona la plantilla **Blank App (Xamarin.Forms Shared)**.



5. Proporciona el nombre, ubicación y haz clic en **OK** para crear el proyecto.

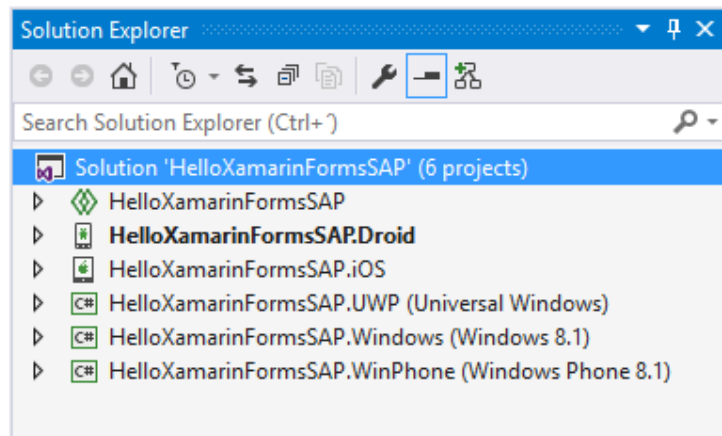


6. Haz clic en el botón **OK** del cuadro de diálogo **New Universal Windows Project** para aceptar las versiones sugeridas para la aplicación UWP que será creada.



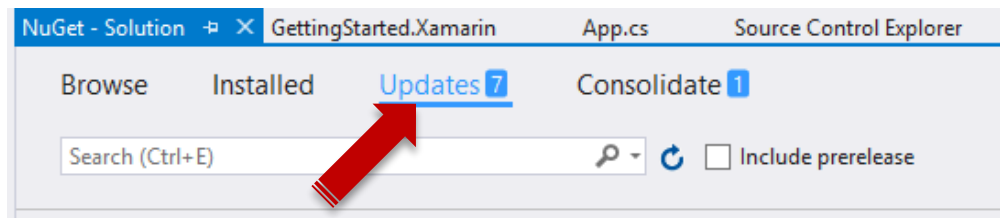
Seis proyectos son creados. Para una solución llamada **HelloXamarinFormsSAP**, estos proyectos son:

- Un proyecto PCL llamado **HelloXamarinFormsSAP** que es referenciado por los otros 5 proyectos.
- Un proyecto de aplicación para Android llamado **HelloXamarinSAP.Droid**.
- Un proyecto de aplicación para iOS llamado **HelloXamarinSAP.iOS**.
- Un proyecto de aplicación para la UWP de Windows 10 y Windows Mobile 10 llamado **HelloXamarinFormsSAP.UWP**.
- Un proyecto de aplicación para Windows 8.1 llamado **HelloXamarinFormsSAP.Windows**.
- Un proyecto de aplicación para Windows Phone 8.1 llamado **HelloXamarinFormsSAP.WinPhone**.

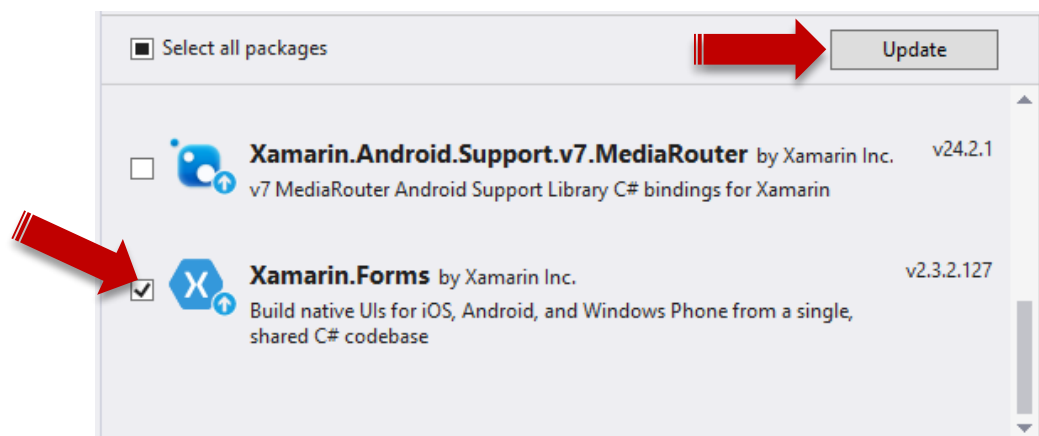


Cuando creas una nueva solución Xamarin.Forms, las bibliotecas Xamarin.Forms y otras bibliotecas auxiliares son descargadas automáticamente desde el administrador de paquetes NuGet. Visual Studio almacena esas bibliotecas en un directorio llamado **packages** dentro del directorio de la solución. Sin embargo, la versión particular de la biblioteca Xamarin.Forms que es descargada es especificada dentro de la plantilla de la solución y, por lo tanto, una nueva versión podría estar disponible.

7. Selecciona la opción **Manage NuGet Packages for Solution** desde el menú contextual del nombre de la solución.
8. Haz clic en la opción **Updates** para ver las actualizaciones disponibles.



9. Selecciona el paquete **Xamarin.Forms** y haz clic en **Update** para iniciar la actualización.

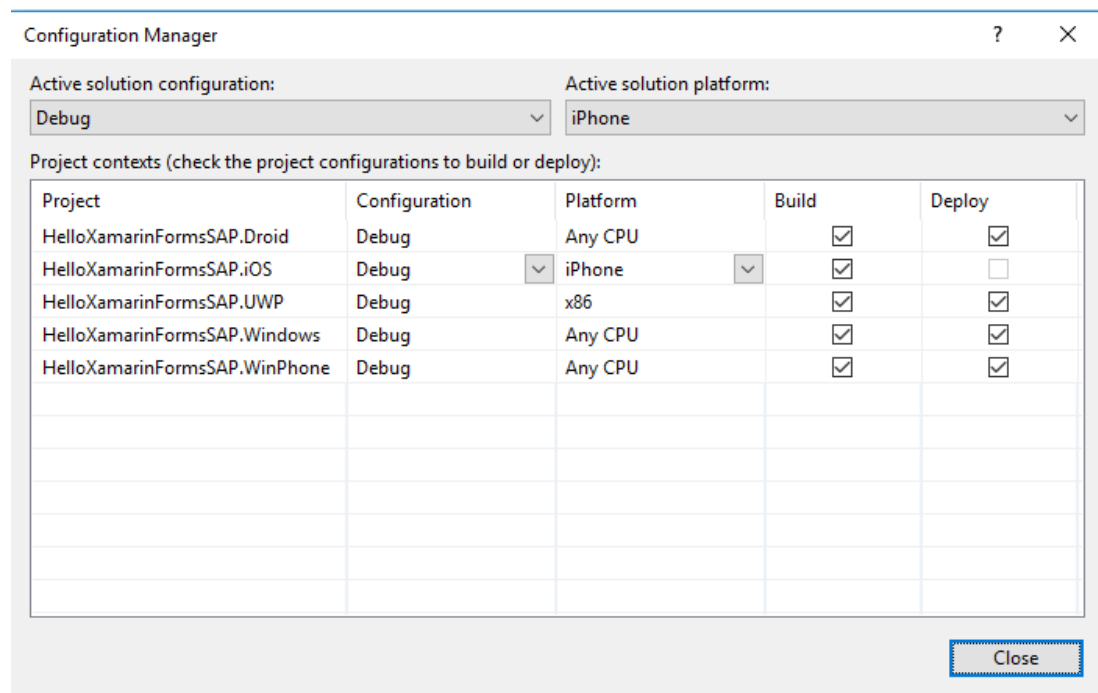


Es probable que te sea solicitado aceptar los cambios a realizar y reiniciar Visual Studio para concluir la instalación.

10. Una vez que haya concluido la actualización selecciona la opción **Build > Configuration Manager**.

11. En la ventana de diálogo **Configuration Manager** podrás ver el proyecto SAP y los otros 5 proyectos de aplicaciones.

Asegúrate que la casilla de verificación **Build** este seleccionada para todos los proyectos y que la casilla **Deploy** este seleccionada para todos los proyectos de aplicación (a menos que la casilla esté gris).



Observa que en la columna **Platform**:

- El proyecto SAP tiene establecido el único valor disponible **Any CPU**.
- El proyecto Android tiene establecido el único valor disponible **Any CPU**.
- El proyecto iOS puede tomar los valores **iPhone** o **iPhoneSimulator** dependiendo de cómo estaremos probando la aplicación.
- El proyecto UWP puede tomar el valor **x86** para desplegar la aplicación hacia el escritorio de Windows o hacia un emulador. También puede tomar el valor **ARM** para desplegar la aplicación hacia un teléfono.
- El proyecto Windows puede tomar el valor **x86** para desplegar la aplicación hacia el escritorio de Windows o hacia un emulador. También puede tomar el valor **ARM** para desplegar la aplicación hacia una tableta. Incluso puede tomar el valor **x64** para

desplegar la aplicación hacia plataformas de 64 bits o el valor **Any CPU** para desplegar la aplicación en el escritorio de Windows, emulador, tableta o plataformas de 64 bits.

- El proyecto WinPhone puede tomar el valor **x86** para desplegar la aplicación hacia un emulador, el valor **ARM** para desplegar la aplicación hacia un teléfono real o el valor **Any CPU** para desplegar la aplicación en ambos, emulador y teléfono real.

12. Haz clic en **Close** para cerrar la ventana de dialogo **Configuration Manager**.

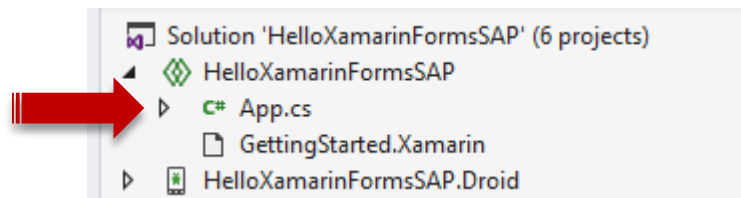
13. Prueba el funcionamiento de cada uno de los proyectos de cada plataforma. Podrás ver el mensaje **"Welcome to Xamarin Forms!"** en cada plataforma.

Tarea 2. Examinar los archivos de código generados.

El proyecto SAP es el proyecto que recibirá la mayor parte de nuestra atención cuando estemos escribiendo una aplicación Xamarin.Forms. En algunas circunstancias el código en este proyecto podría requerir algo de código particular de ciertas plataformas.

Tal y como pudiste observar, todo parece igual entre una solución PCL y una solución SAP. Veamos algunas diferencias.

1. La primera diferencia que podemos notar es que el proyecto SAP contiene únicamente un elemento: el archivo **App.cs** (además del archivo `GettingStarted.Xamarin` que es informativo).



Tanto en los proyectos PCL como en los proyectos SAP, el código es compartido con las 5 aplicaciones, pero de diferentes maneras.

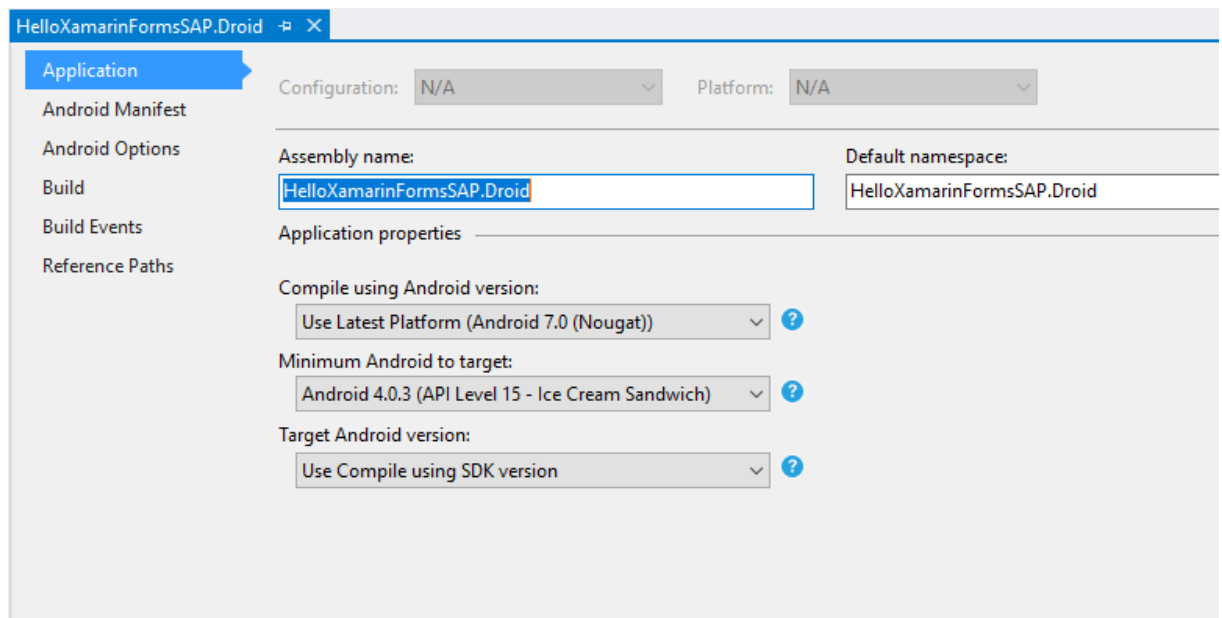
- Con el enfoque del proyecto PCL, todo el código común es encapsulado dentro de una biblioteca DLL que cada proyecto de aplicación hace referencia y enlaza en tiempo de ejecución.
- Con el enfoque del proyecto SAP, los archivos con el código común, son realmente incluidos con cada uno de los 5 proyectos en tiempo de compilación.

De manera predeterminada, el proyecto SAP tiene únicamente un solo archivo llamado **App.cs**, pero realmente es como si el proyecto SAP no existiera y que en lugar de eso existieran 5 diferentes copias de ese archivo, una en cada proyecto de aplicación.

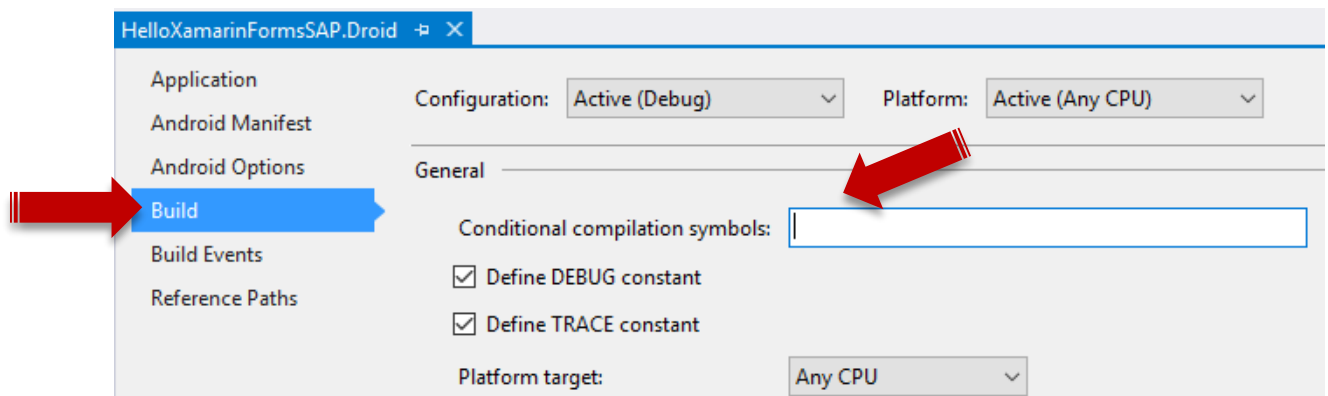
Algunos problemas sutiles (y no tan sutiles) pueden manifestarse con el enfoque de biblioteca compartida, por ejemplo, los proyectos Android y iOS tienen acceso a más o menos la misma versión de .NET, pero no es la misma versión de .NET que utilizan los proyectos de Windows. Esto significa que cualquier clase .NET que el código compartido accede podría ser algo diferente dependiendo de la plataforma. Este es el caso de algunas clases de E/S de archivos en el espacio de nombres System.IO.

Podemos compensar esta situación a través del uso de las directivas de compilación de C#, particularmente de la directiva **#if** y **#elif**. En los proyectos generados por la plantilla Xamarin.Forms, los distintos proyectos de aplicación definen símbolos que podemos utilizar con esas directivas. Veamos cuáles son esos símbolos.

2. Selecciona la opción **Properties** del menú contextual del proyecto Android. La hoja de propiedades del proyecto te será mostrada.

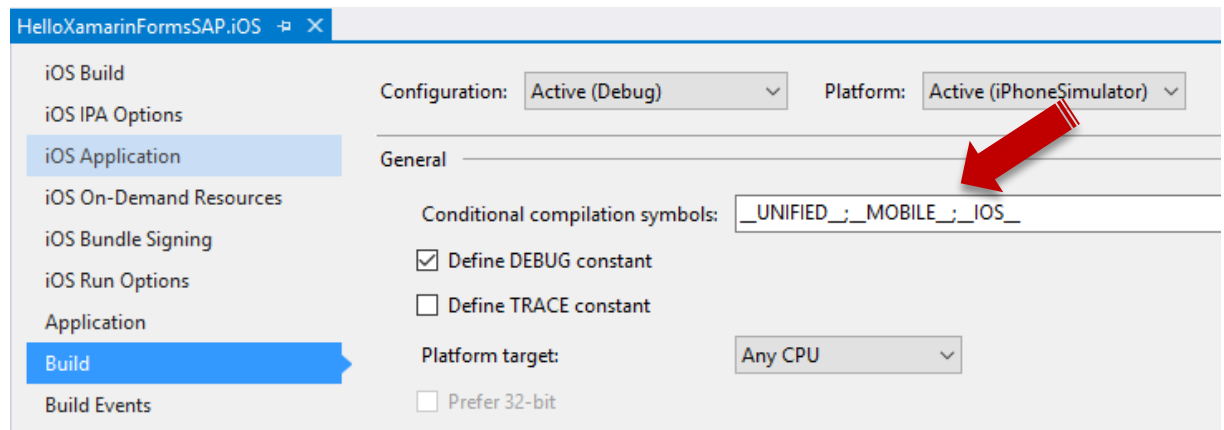


3. Selecciona la opción **Build** y observa el valor del campo **Conditional compilation symbols**. Podrás notar que el campo aparece vacío, sin embargo, el identificador **__ANDROID__** y varios identificadores **__ANDROID_nn__** (donde **nn** representa cada nivel de API Android soportada) se encuentran definidos.



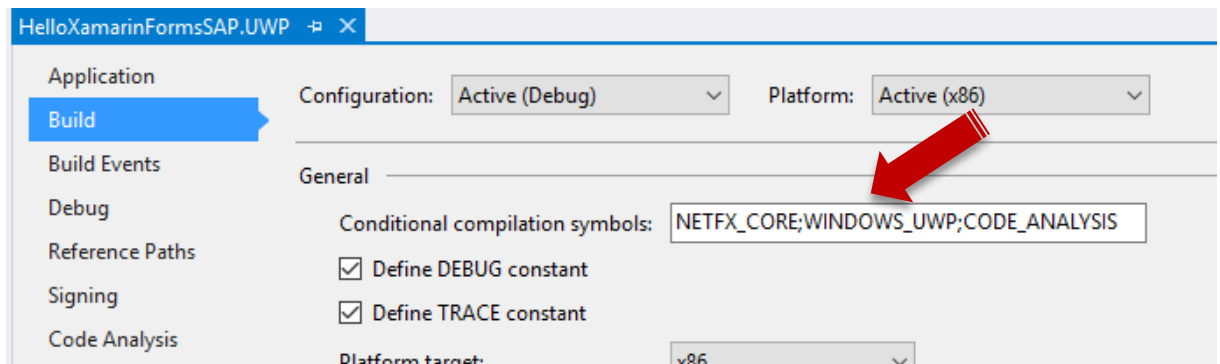
4. Selecciona la opción **Properties** del menú contextual del proyecto iOS. La hoja de propiedades del proyecto te será mostrada.
5. Selecciona la opción **Build** y observa el valor del campo **Conditional compilation symbols**. Podrás notar la lista de 3 símbolos definidos que se encuentran separados por punto y coma:

- __UNIFIED__
- __MOBILE__
- __IOS__



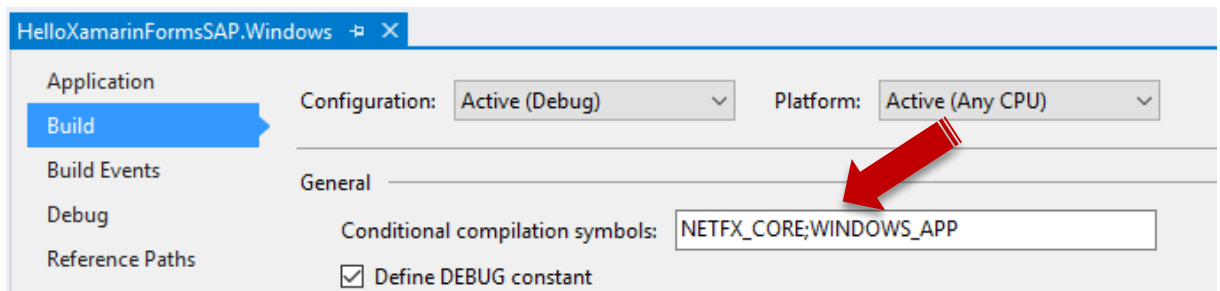
6. Selecciona la opción **Properties** del menú contextual del proyecto UWP. La hoja de propiedades del proyecto te será mostrada.
7. Selecciona la opción **Build** y observa el valor del campo **Conditional compilation symbols**. Podrás notar los siguientes símbolos:

- NETFX_CORE
- WINDOWS_UWP
- CODE_ANALYSIS



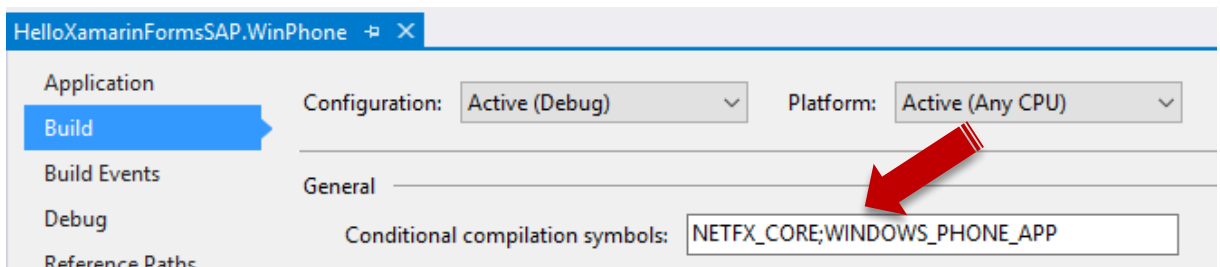
8. Selecciona la opción **Properties** del menú contextual del proyecto Windows. La hoja de propiedades del proyecto te será mostrada.
9. Selecciona la opción **Build** y observa el valor del campo **Conditional compilation symbols**. Podrás notar los siguientes símbolos:

- NETFX_CORE
- WINDOWS_APP



10. Selecciona la opción **Properties** del menú contextual del proyecto Windows Phone. La hoja de propiedades del proyecto te será mostrada.
11. Selecciona la opción **Build** y observa el valor del campo **Conditional compilation symbols**. Podrás notar los siguientes símbolos:

- NETFX_CORE
- WINDOWS_PHONE_APP



12. Abre el archivo **App.cs** del proyecto SAP.

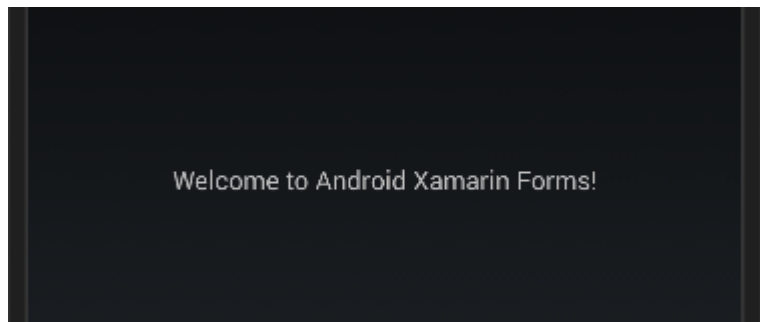
13. Remplaza la línea

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

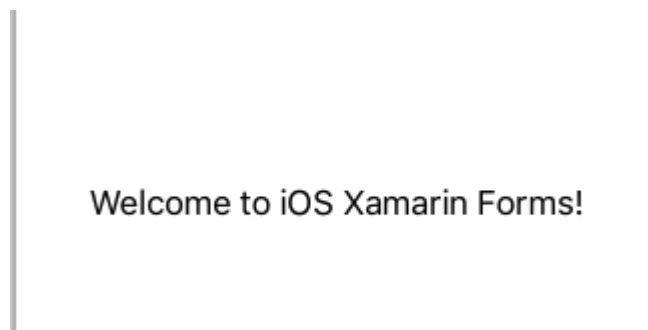
por el siguiente código:

```
#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. Despliega la aplicación en algún emulador Android. Podrás ver que se muestra el mensaje apropiado para Android.



15. Despliega la aplicación en algún emulador iOS. Podrás ver que se muestra el mensaje apropiado para iOS.



16. Despliega la aplicación en algún emulador Windows 10 Mobile. Podrás ver que se muestra el mensaje apropiado para UWP.



17. Despliega la aplicación Windows 8.1 en el simulador. Podrás ver que se muestra el mensaje apropiado para la aplicación Windows.



18. Despliega la aplicación Windows Phone en un emulador. Podrás ver que se muestra el mensaje apropiado para la aplicación Windows Phone.



Como habrás notado, mediante las directivas de compilación de C#, los archivos de código compartido pueden ejecutar código específico de cada plataforma o acceder a clases específicas de cada plataforma, incluyendo clases en los proyectos individuales de cada plataforma. También puedes definir tus propios símbolos de compilación condicional si así lo deseas.

Las directivas de compilación no tienen sentido en proyectos PCL. Un proyecto PCL es enteramente independiente de las 5 plataformas ya que los identificadores en los proyectos de cada plataforma no están presentes en el momento en que el proyecto PCL es compilado.

El concepto de PCL originalmente surgió debido a que cada plataforma que utiliza .NET, en la actualidad, utiliza un subconjunto diferente de .NET. Si deseas crear una biblioteca que pueda ser utilizada por diferentes plataformas .NET, necesitas utilizar únicamente las partes comunes de esos subconjuntos .NET.

Una PCL es concebida para ser de ayuda al contener código que es utilizable en múltiples (pero específicas) plataformas .NET. En consecuencia, una PCL particular, contiene una especie de

banderas incrustadas que indican las plataformas que soporta. Una PCL utilizada en una aplicación Xamarin.Forms debe soportar las siguientes plataformas:

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

Si llegáramos a necesitar un comportamiento específico de una plataforma en la PCL, no podríamos utilizar las directivas de compilación de C# debido a que estas solo actúan en tiempo de compilación. Necesitaríamos de algo que trabaje en tiempo de ejecución, tal como la clase **Device** de Xamarin.Forms.

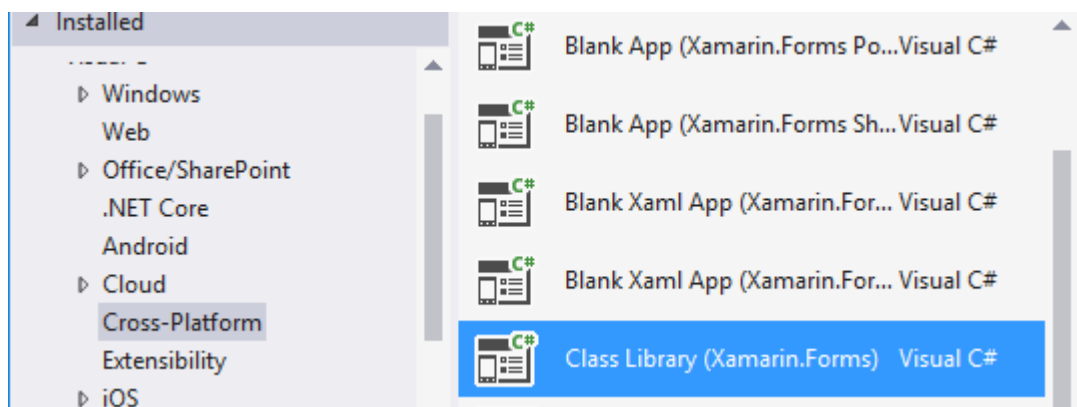
Una PCL Xamarin.Forms puede acceder a otras PCLs que soporten las mismas plataformas, sin embargo, no puede acceder directamente a clases definidas en los proyectos de aplicación individuales. Si llegáramos a requerir esa funcionalidad, Xamarin.Forms proporciona una clase llamada **DependencyService** que nos permite acceder a código específico de una plataforma desde una PCL siguiendo una metodología.

¿PCL o SAP?

PCL es la estrategia recomendada para Xamarin.Forms y es la preferida por muchos programadores que han estado trabajando con Xamarin.Forms desde hace tiempo. Sin embargo, la estrategia SAP, es también soportada y también tiene sus seguidores.

¿Cuál seleccionar?

Es posible tener los dos tipos de proyectos en la misma solución Xamarin.Forms. Si hemos creado una solución Xamarin.Forms con un proyecto SAP, podemos agregar un nuevo proyecto PCL a la solución seleccionando la plantilla **Class Library (Xamarin.Forms)**.



Los proyectos de las aplicaciones pueden acceder tanto al proyecto SAP como al proyecto PCL. El proyecto PCL también puede acceder a los archivos del proyecto SAP.

Ejercicio 2: Utilizando el View Label para mostrar texto.

En este ejercicio exploraremos el *View **Label*** para mostrar texto en una aplicación multiplataforma desarrollada con Xamarin.Forms.

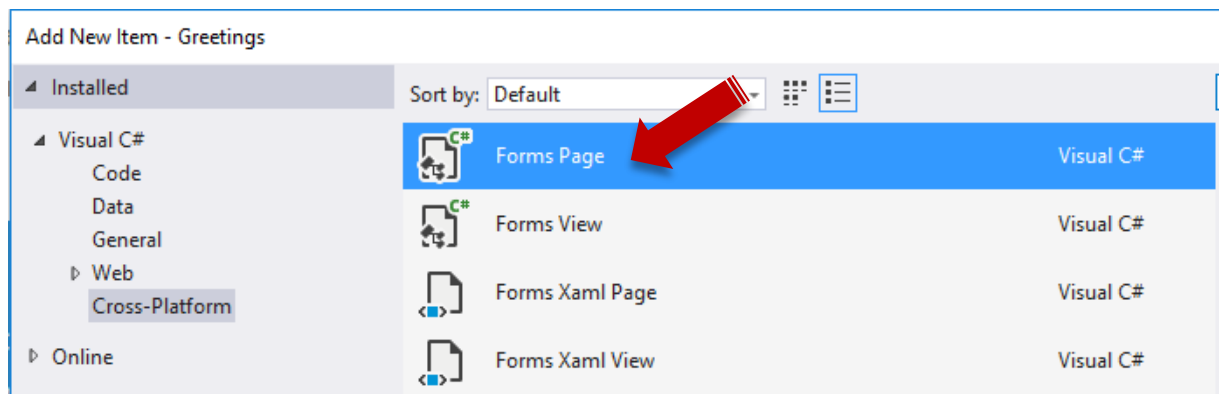
Tarea 1. Crear una solución Xamarin.Forms PCL.

En esta tarea crearemos una nueva solución Xamarin.Forms PCL llamada **Greetings** utilizando el mismo proceso descrito anteriormente para crear una solución Xamarin.Forms utilizando la plantilla **Blank App (Xamarin.Forms.Portable)**.

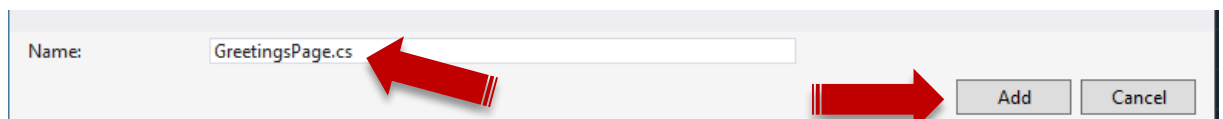
1. Crea una nueva aplicación multiplataforma llamada **Greetings** utilizando la plantilla **Blank App (Xamarin.Forms.Portable)**.
2. Esta solución será estructurada como un programa típico Xamarin.Forms, lo que significa que definirá una nueva clase que derive de **ContentPage**. Por lo tanto, debemos agregar un nuevo archivo al proyecto **Greetings**.

Selecciona **Add > New Item** desde el menú contextual del proyecto PCL Greetings.

3. En la ventana **Add New Item**, selecciona la plantilla **Forms Page**.



4. Asigna el nombre **GreetingsPage.cs** y haz clic en **Add** para agregar el nuevo elemento.



5. Observa el código de la clase **GreetingsPage**. Podrás notar que la clase deriva de **ContentPage**. Debido a que la clase **ContentPage** se encuentra en el espacio de nombres Xamarin.Forms, puedes observar que una directiva **using** incluye ese espacio de nombres.

La clase está definida como pública pero realmente no es necesario debido a que no será accedida directamente fuera del proyecto Greetings.

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

Es en la clase `GreetingsPage` (y otras como esta) donde invertirás gran parte de tu tiempo en tus inicios de programación con `Xamarin.Forms`. Para programas de una sola página, esta clase podría contener el único código de la aplicación que necesitaríamos escribir. Por supuesto, podríamos agregar clases adicionales al proyecto si así lo necesitamos.

Tradicionalmente, en programas de una sola página, es recomendable que la clase que deriva de **ContentPage** tenga el mismo nombre de la aplicación, pero con el sufijo **Page**. Esta convención de nombres es sugerida pero no obligatoria.

6. Elimina todo el código del constructor **GreetingsPage** y las directivas **using** para que el archivo se vea similar a lo siguiente.

```
using System;
using Xamarin.Forms;

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

7. Agrega el siguiente código dentro del constructor para instanciar un **Label**, establecer su propiedad **Text** y asignar la instancia **Label** a la propiedad **Content** que **GreetingsPage** hereda de **ContentPage**.

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

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

8. Abre el archivo **App.cs** del proyecto PCL.
9. Elimina el código del constructor. El código de la clase será similar al siguiente.

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

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

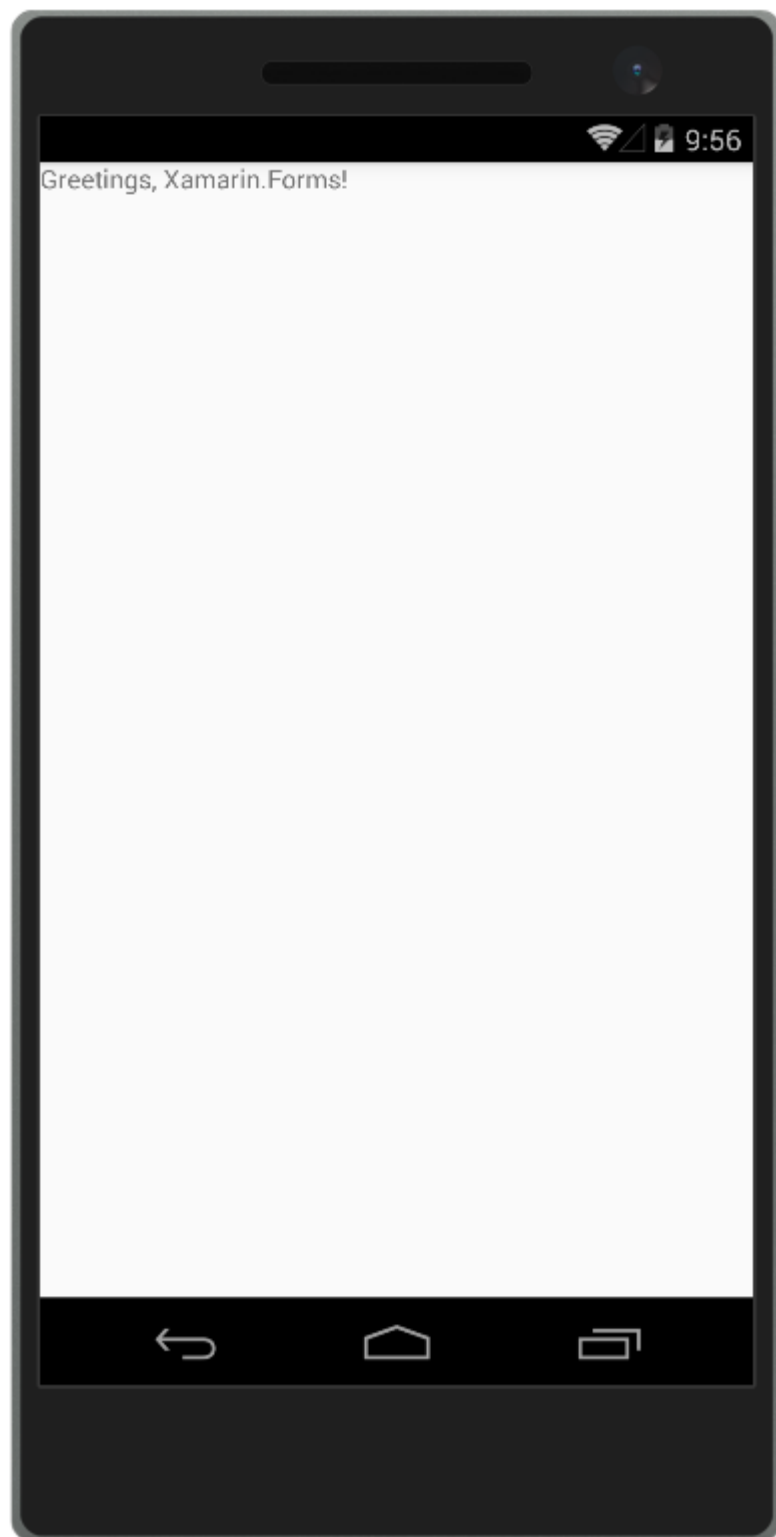
    protected override void OnSleep()
    {
        // Handle when your app sleeps
    }

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

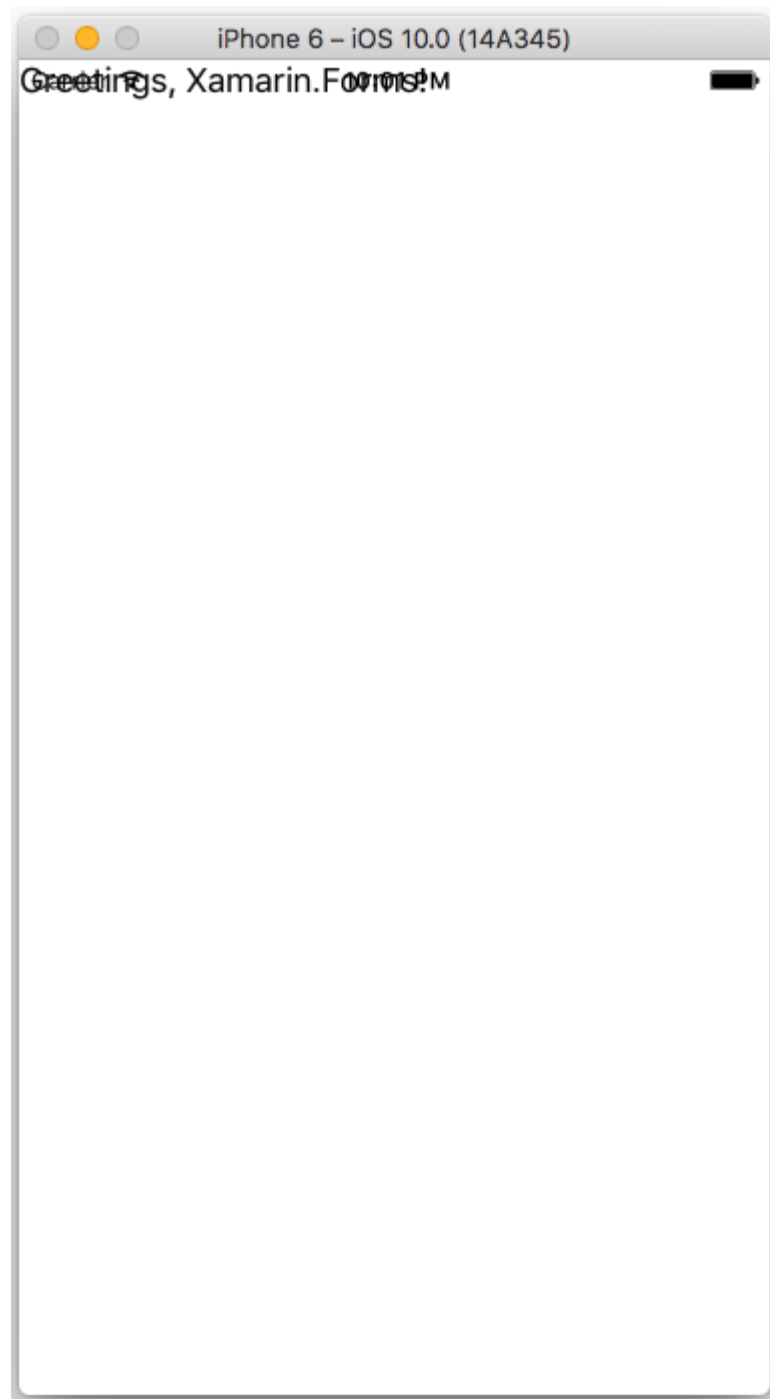
10. Agrega el siguiente código dentro del constructor para establecer a la propiedad **MainPage** el valor de una instancia de la clase **GreetingsPage**.

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

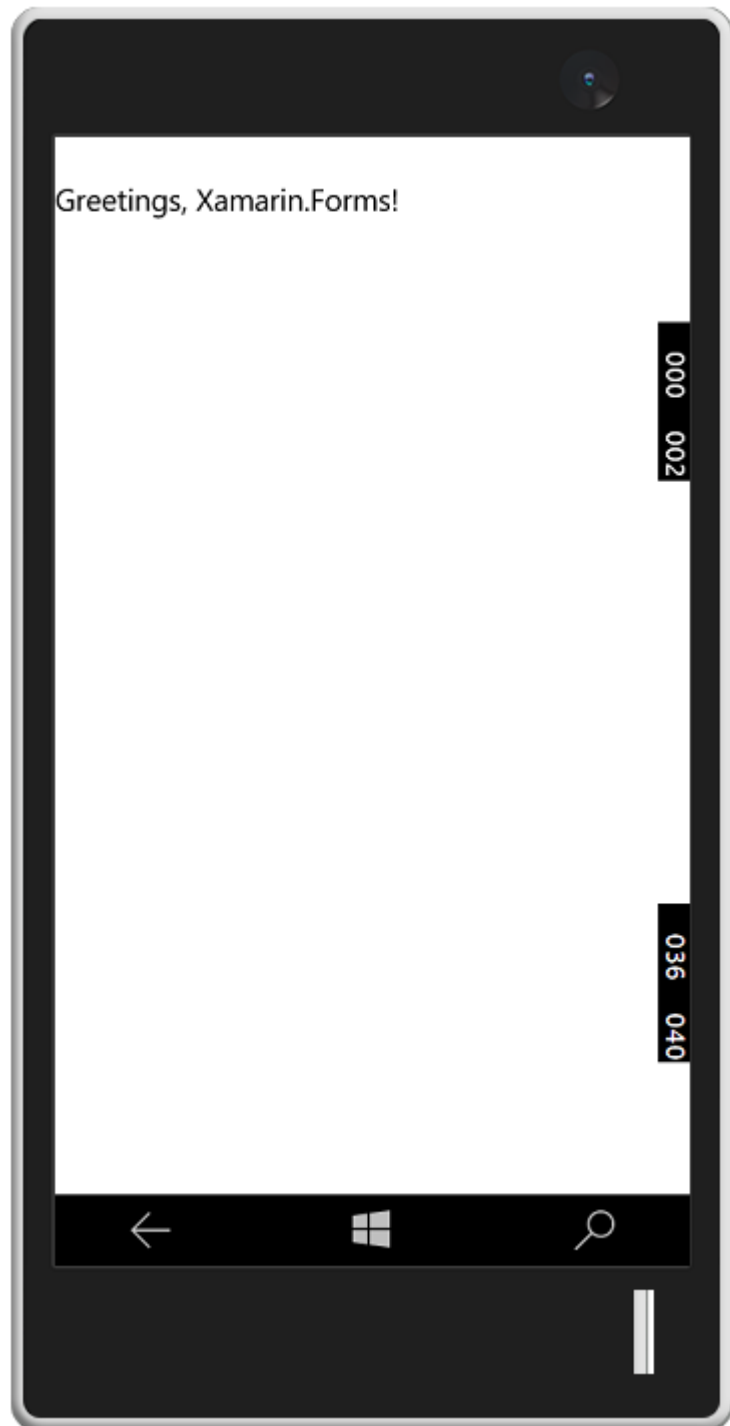
11. Compila y ejecuta la aplicación en emuladores Android, iOS y Windows 10 Mobile. Las imágenes siguientes muestran la aplicación en los 3 emuladores.



Android



iOS



Windows 10 Mobile

Definitivamente la versión más decepcionante de esta aplicación Greetings es la versión iOS. Puedes darte cuenta que la aplicación comparte la pantalla con la barra de estado en la parte superior. Cualquier cosa que la aplicación muestre en la parte superior de la página ocupará el mismo espacio que la barra de estado a no ser que la aplicación haga algo para evitarlo.

¿Cómo poder resolver este problema? Existen 4 formas de resolver este problema (o 5 si utilizamos un proyecto SAP).

Tarea 2. Incluir espacio (padding) en la página (Solución 1).

La clase **Page** define una propiedad llamada **Padding** que marca un área alrededor del perímetro interior de la página y en el cual el contenido no puede ubicarse. La propiedad **Padding** es de tipo **Thickness**, una estructura que define cuatro propiedades llamadas **Left**, **Top**, **Right** y **Bottom**. Es recomendable recordar ese orden debido a que ese es el orden en que definiremos las propiedades en el constructor de la clase **Thickness** así como en XAML. La estructura **Thickness** también define constructores para establecer la misma cantidad de espacio sobre los 4 lados o para establecer la misma cantidad de espacio a la izquierda y derecha o en la parte superior e inferior.

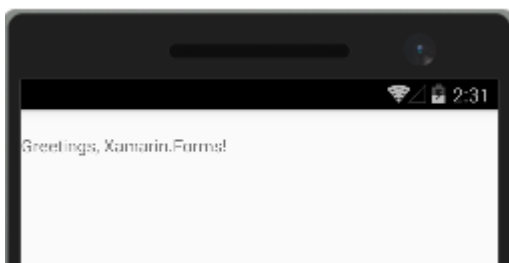
Con una pequeña búsqueda en Internet podremos encontrar que la barra de estatus de iOS tiene una altura de 20 unidades.

1. Agrega el siguiente código al constructor de la clase **GreetingsPage** para dejar un espacio de 20 unidades en la parte superior de la página.

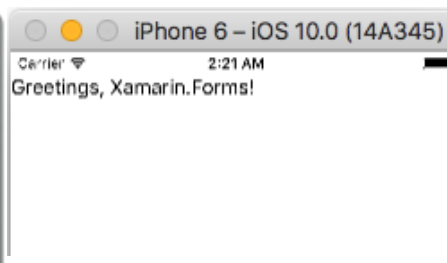
```
public GreetingsPage()
{
    var MyLabel = new Label();
    MyLabel.Text = "Greetings, Xamarin.Forms!";
    this.Content = MyLabel;

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

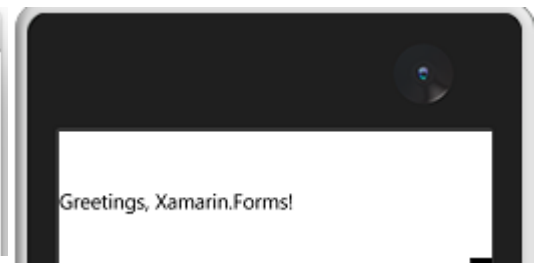
2. Ejecuta la aplicación en los 3 emuladores para ver el resultado.



Android



iOS



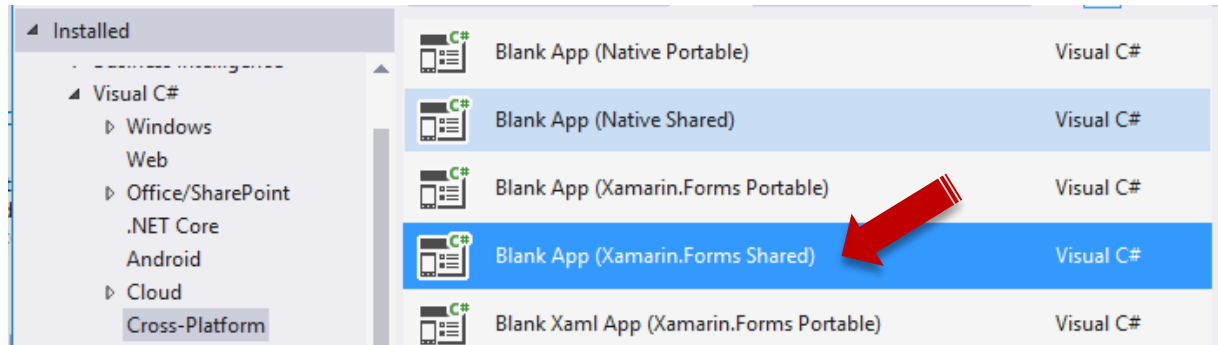
Windows 10 Mobile

Puedes notar que al establecer el valor en la propiedad **Padding** de **ContentPage** resuelves el problema del solapamiento de texto con la barra de estado de iOS, pero también establece el mismo espacio que no es requerido en Android y Windows Mobile. ¿Habrá forma de establecer el espacio únicamente en iPhone?

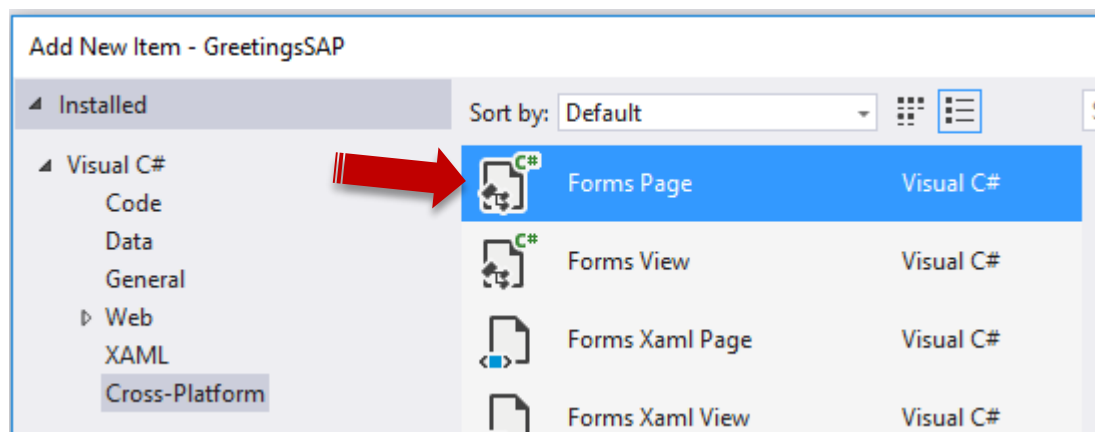
Tarea 3. Incluir espacio (padding) solo para iOS en proyectos SAP (Solución 2).

Una de las ventajas del uso de los proyectos SAP es que las clases en el proyecto SAP son extensiones de los proyectos de las aplicaciones, por lo tanto, podemos utilizar directivas de compilación condicional.

1. Abre una nueva instancia de Visual Studio.
2. Crear el proyecto **GreetingsSAP** utilizando la plantilla **Blank App (Xamarin.Forms Shared)**.



3. Selecciona **Add > New Item** desde el menú contextual del proyecto SAP.
4. En la ventana **Add New Item**, selecciona la plantilla **Forms Page**.



5. Asigna el nombre **GreetingsSAPPage.cs** y haz clic en **Add** para agregar el nuevo elemento.
6. Modifica el código del archivo **GreetingsSAPPage.cs** para que sea similar al de la clase **Greetings** del proyecto 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);  
    }  
}
```

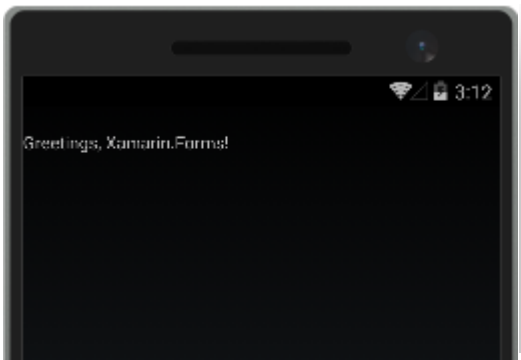
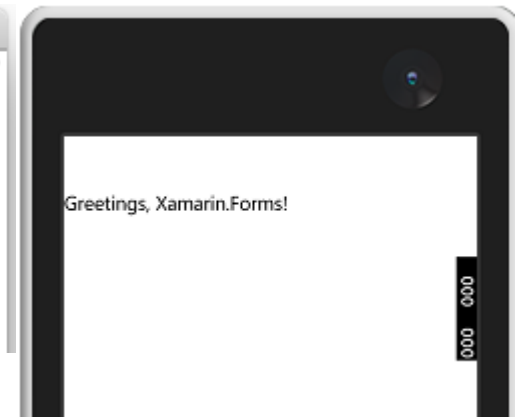
7. Abre el archivo **App.cs** del proyecto SAP.
8. Elimina el código del constructor. El código de la clase será similar al siguiente.

```
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. Agrega el siguiente código dentro del constructor para establecer a la propiedad **MainPage** el valor de una instancia de la clase **GreetingsSAPPage**.

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

10. Compila y ejecuta la aplicación en emuladores Android, iOS y Windows 10 Mobile.
Las imágenes siguientes muestran la aplicación en los 3 emuladores.

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

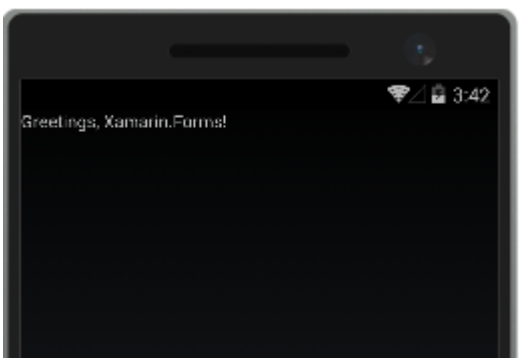
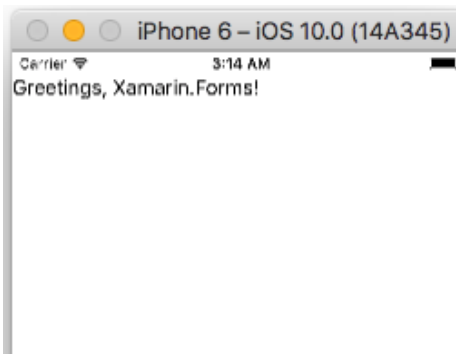
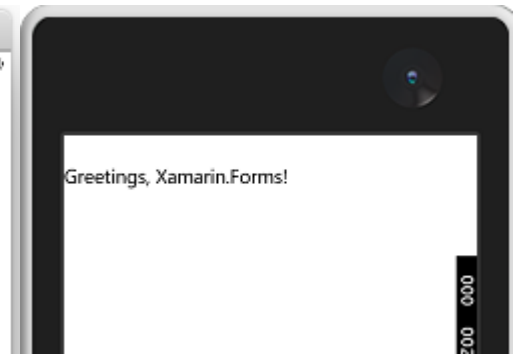
En este momento, ambas soluciones PCL y SAP tienen el mismo problema que agrega espacio en la parte superior de la página en Android y Windows Mobile.

11. Modifica el código del constructor de la clase **GreetingsSAPPage** para incorporar una directiva de compilación que haga que el código que establece el valor a la propiedad **Padding** solo sea compilado en el proyecto iOS. El código será similar al siguiente.

```
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. Compila y ejecuta la aplicación en emuladores Android, iOS y Windows 10 Mobile. Las imágenes siguientes muestran la aplicación en los 3 emuladores.

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

Nota que debido a que la directiva **#if** hace referencia al símbolo de compilación condicional **__IOS__**, la propiedad **Padding** es establecida únicamente para el proyecto iOS.

Sin embargo, esos símbolos de compilación condicional afectan solo a la compilación del programa, por lo que no tienen efecto en una solución PCL. ¿Habría alguna forma para que en un proyecto PCL se puedan incluir diferentes valores **Padding** para diferentes plataformas?

Tarea 4. Incluir espacio (padding) únicamente para iOS en proyectos PCL o SAP (Solución 3).

La clase estática **Device**, incluye diversas propiedades y métodos que permiten al código hacer frente a las diferencias entre dispositivos en tiempo de ejecución de una manera fácil y simple.

- La propiedad **Device.OS** devuelve un miembro de la enumeración **TargetPlatform** que indica el tipo de sistema operativo en el que está trabajando: Android, iOS, Other, Windows o WinPhone.
- La propiedad **Device.Idiom** devuelve un miembro de la enumeración **TargetIdiom** que indica el tipo de dispositivo en el que está trabajando: Desktop, Phone, Tablet o Unsupported.

Es posible utilizar esas dos propiedades en instrucciones **if** y **else** o en un bloque **switch** para ejecutar código específico de una plataforma particular.

Dos métodos llamados **OnPlatform** proporcionan soluciones más elegantes:

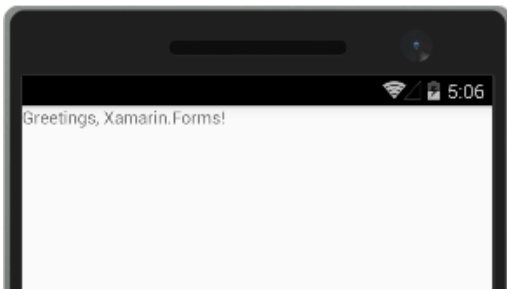
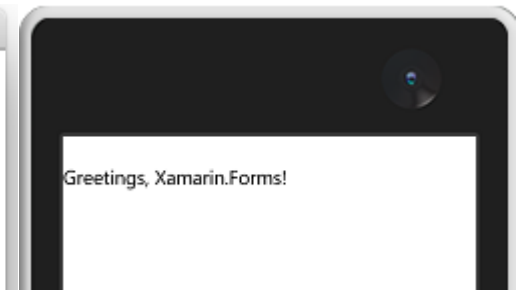
- El método genérico estático **Device.OnPlatform<T>** toma 3 argumentos de tipo T, el primero para iOS, el segundo para Android y el tercero para plataformas Windows.
- El segundo método estático **Device.OnPlatform** tiene 4 argumentos de tipo **Action** también en el orden iOS, Android y Windows con un 4º argumento para un valor por defecto.

1. Abre el archivo **GreetingsPage** del proyecto PCL.
2. Reemplaza la línea de código **Padding = new Thickness(0, 20, 0, 0);** por el siguiente código para establecer el valor **Padding** para únicamente los dispositivos iPhone.

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

El primer argumento **Thickness** es para iOS, el Segundo es para Android y el tercero es para Windows.

13. Compila y ejecuta la aplicación en emuladores Android, iOS y Windows 10 Mobile.
Las imágenes siguientes muestran la aplicación en los 3 emuladores.

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

Puedes notar que el Padding únicamente fue aplicado al dispositivo iOS.

El código anterior también podría quedar de la siguiente manera. ¿Por qué?

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

Incluso, podríamos utilizar la versión de **Device.OnPlatform** con argumentos **Action**.

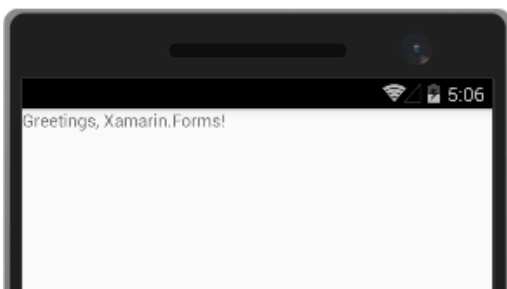
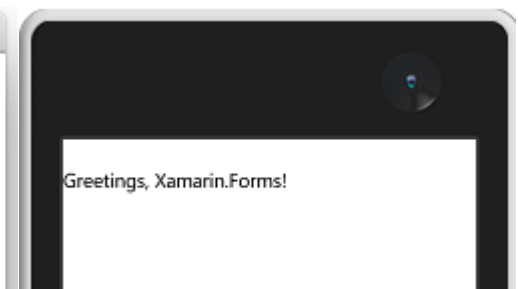
3. Modifica la línea de código anterior por lo siguiente.

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

Los argumentos del método **OnPlatform** son **null** de manera predeterminada por lo que podemos establecer únicamente el valor del primer argumento para que sea procesado en iOS, tal y como se muestra en el código anterior.

14. Compila y ejecuta la aplicación en emuladores Android, iOS y Windows 10 Mobile.

Las imágenes siguientes muestran la aplicación en los 3 emuladores.

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

Puedes notar que la aplicación funciona como es esperado debido a que la instrucción que establece el Padding se ejecuta solamente cuando la aplicación está corriendo sobre iOS. Por supuesto que al establecer un solo argumento en el método **Device.OnPlatform** haría que el código sea poco entendible para personas que necesiten leerlo. Algo que podría ayudar a

que el código sea más claro es estableciendo el nombre del parámetro precediendo el argumento como en el siguiente ejemplo.

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

La sintaxis anterior hace explícito que la instrucción se ejecuta solamente en iOS.

El método **Device.OnPlatform** es muy práctico y tiene la ventaja de trabajar en ambos proyectos PCL y SAP. Sin embargo, no puede acceder a las APIs dentro de las plataformas individuales. Para hacer esto necesitaremos **DependencyService**.

Tarea 5. Centrar el Label dentro de la página (Solución 4).

El problema del texto encimándose sobre la barra de estado de iOS ocurre solo porque la presentación predeterminada del texto se realiza sobre la parte superior izquierda. ¿Será posible centrar el texto en la página?

Xamarin.Forms soporta diversas características que facilitan el diseño sin requerir que el programa realice cálculos complejos envolviendo tamaños y coordenadas. La clase **View** define dos propiedades llamadas **HorizontalOptions** y **VerticalOptions** que especifican la forma en que el **View** debe ser posicionado en relación a su contenedor (en este caso **ContentPage**). Estas dos propiedades son de tipo **LayoutOptions**, una estructura muy importante en Xamarin.Forms.

Generalmente utilizamos la estructura **LayoutOptions** especificando uno de los 8 campos públicos estáticos de solo lectura que define:

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

La estructura **LayoutOptions** también define dos propiedades de instancia que nos permiten crear un valor con esas mismas combinaciones:

- Una propiedad **Alignment** de tipo **LayoutAlignment** que es una enumeración con 4 miembros: **Center**, **End**, **Fill** y **Start**.
- Una propiedad **Expands** de tipo **bool**.

Para la propiedad **HorizontalOptions**, la palabra **Start** significa izquierda y **End** significa derecha.

Para la propiedad **VerticalOptions**, la palabra **Start** significa arriba y **End** significa abajo.



Para obtener más información sobre la estructura **LayoutOptions** puedes consultar el siguiente enlace:

Xamarin.Forms.LayoutOptions Structure

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

Veamos como posicionar el **Label** en el centro de la página.

1. Modifica el código del constructor de la clase **GreetingsPage** de la siguiente manera.

```
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. Compila y ejecuta la aplicación en emuladores Android, iOS y Windows 10 Mobile. Podrás notar que el texto aparece centrado en la página.

Las imágenes siguientes muestran la aplicación en los 3 emuladores.

**Android****iOS**



Windows 10 Mobile

Tarea 6. Centrar el texto dentro del Label (Solución 5).

El View **Label** está diseñado para mostrar texto de una línea o un párrafo. Frecuentemente es necesario controlar la forma en que las líneas de texto son alineadas horizontalmente. El View **Label** define la propiedad **HorizontalTextAlignment** para ese propósito y también define la propiedad **VerticalTextAlignment** para posicionar el texto verticalmente. Ambas propiedades son establecidas con el valor de un miembro de la enumeración **TextAlignment** que tiene los miembros **Start**, **Center** y **End**.

Para esta última solución al problema de la barra de estado de iOS vamos a configurar los valores de las propiedades **HorizontalTextAlignment** y **VerticalTextAlignment** del View **Label**.

1. Modifica el código del constructor de la clase **GreetingsPage** de la siguiente manera.

```
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. Compila y ejecuta la aplicación en emuladores Android, iOS y Windows 10 Mobile. Podrás notar que el texto aparece centrado en la página.

Visualmente, el resultado con una sola línea de texto es lo mismo que centrar con las propiedades **HorizontalOptions** y **VerticalOptions** sin embargo esas técnicas son ligeramente diferentes.

Resumen

En este laboratorio analizaste las distintas opciones para compartir código en una aplicación multiplataforma y exploraste el View **Label** para mostrar texto en una aplicación multiplataforma desarrollada con Xamarin.Forms. Conociste también algunas de las alternativas disponibles al mostrar texto para tener la misma apariencia en las distintas plataformas.

Es momento de empezar a trabajar con más elementos para construir la interfaz de usuario en una aplicación Xamarin.Forms.

Cuando hayas finalizado este laboratorio publica el siguiente mensaje en Twitter y Facebook:

¡He finalizado el #Lab03 del #XamarinDiplomado y conozco las opciones para compartir código en Xamarin!