



Xamarin.Forms XAML

Manual de estudiante



Miguel Muñoz Serafín



Xamarin.Forms

Módulo XAML

Manual de estudiante

Primera edición

Julio de 2018

Soporte técnico:
soporte@mail.ticapacitacion.com

Contenido

Acerca del módulo.....	4
Objetivos	4
Lección 1 Conceptos básicos de XAML.....	5
Objetivos de la lección	5
Conceptos básicos de XAML.....	6
Laboratorio Examinando la anatomía de un archivo XAML.....	8
Laboratorio Agregando nuevas páginas XAML	14
Laboratorio Agregando navegación entre páginas.....	18
Laboratorio Interactuando con código C#.....	23
Laboratorio Elementos de Propiedad (Property Elements)	31
Laboratorio Propiedades adjuntas (Attached Properties)	36
Propiedades de Contenido.....	41
Laboratorio Manejando diferencias de plataforma con OnPlatform	43
Extensiones de marcado XAML.....	48
Laboratorio Utilizando extensiones de marcado XAML para compartir recursos	49
Laboratorio Utilizando la extensión de marcado x:Static	57
Extensiones de marcado estándar	63
Propiedades en las extensiones de marcado XAML.....	64
Laboratorio Enlace de datos (Data Binding).....	65
Laboratorio Enlaces View-a-View.....	71
Laboratorio El Modo de enlace	75
Laboratorio Enlaces y colecciones.....	79
Laboratorio Convertidores de valores de enlace	86
Compilación XAML	90
Laboratorio Xamarin Live Reload	91
Laboratorio XAML Previewer para Xamarin.Forms.....	95
Lección 2 Extensiones de marcado XAML.....	97
Objetivos de la lección	97
Consumiendo Extensiones de Marcado XAML	98



Laboratorio Creando Extensiones de Marcado XAML	101
Laboratorio Creando una extensión de marcado para acceder a Bitmaps.....	105
Lección 3 Argumentos, Propiedades y Diccionarios.....	111
Objetivos de la lección	111
Laboratorio Pasando argumentos a un Constructor	112
Laboratorio Invocando métodos Factory.....	115
Especificando un argumento de un tipo genérico	118
Propiedades enlazables.....	119
Laboratorio Creando y consumiendo una propiedad enlazable	120
Propiedades Adjuntas	127
Laboratorio Creando y consumiendo Propiedades adjuntas	128
Diccionarios de Recursos.....	136
Laboratorio Creando y consumiendo un Diccionario de Recursos	137
Lección 4 XAML Standard (Preview).....	146
Objetivos de la lección	146
Los Controles XAML Standard (Preview).....	147

Acerca del módulo

Módulo XAML forma parte de una colección de módulos relacionados con Xamarin.Forms. El módulo está dirigido a desarrolladores que están incursionando en el mundo de desarrollo de aplicaciones móviles multiplataforma utilizando Xamarin.Forms.

Este módulo presenta una introducción a XAML, el lenguaje de marcado declarativo que puede ser utilizado para definir interfaces de usuario. Se describe la sintaxis básica de XAML así como las extensiones de marcado XAML que incluyen el enlace de datos.

El módulo describe el uso de los atributos XAML para pasar argumentos a constructores no predeterminados, para invocar métodos y para especificar el tipo de un argumento genérico. Se explica el propósito de las *Propiedades Enlazables (Bindable Properties)*, *Propiedades Adjuntas (Attached Property)* y el uso de *Diccionarios de Recursos*.

Finalmente, en este módulo se describen los pasos para experimentar con *XAML Standard* en Xamarin.Forms.

El contenido de este módulo concentra la información que puede ser encontrada en el sitio con la documentación oficial de Xamarin (<https://docs.microsoft.com/en-us/xamarin/>), pero estructurada en un formato que pretende facilitar y agilizar el proceso de aprendizaje de XAML en Xamarin.Forms.

Objetivos

Al finalizar este módulo, los participantes contarán con las habilidades y conocimientos para:

- Describir qué es XAML.
- Describir la sintaxis básica de XAML.
- Describir la forma en que las extensiones de marcado XAML permiten extender el poder y flexibilidad de XAML.
- Crear y consumir extensiones de marcado XAML.
- Describir la forma de pasar argumentos en XAML.
- Describir la extensión de marcado para enlace de datos.
- Describir el propósito de las propiedades enlazables y las propiedades adjuntas.
- Describir el uso de Diccionarios de Recursos.
- Experimentar con XAML Standard en Xamarin.Forms.

Los temas que se cubren en este módulo son:

- Lección 1: Conceptos básicos de XAML.
- Lección 2: Extensiones de marcado XAML.
- Lección 3: Argumentos, Propiedades y Diccionarios.
- Lección 4: XAML Standard (Preview).



Lección 1

Conceptos básicos de XAML

En esta lección se describe la sintaxis básica de XAML - *eXtensible Application Markup Language* - (*Lenguaje de marcado de aplicaciones extensible*) que permite a los desarrolladores definir interfaces de usuario en aplicaciones Xamarin.Forms utilizando lenguaje declarativo XAML en lugar de lenguaje imperativo C#.

Esta lección describe el uso de las extensiones de marcado que extienden el poder de XAML y se muestra que, aunque no es obligatorio el uso de XAML en una aplicación Xamarin.Forms, suele ser más conciso y visualmente más coherente que el código equivalente en C#.

Objetivos de la lección

Al finalizar esta lección, los participantes podrán:

- Describir la sintaxis básica de Xamarin.
- Describir los beneficios que ofrecen las extensiones de marcado XAML.
- Describir el enlace de datos.
- Describir el proceso de compilación de XAML.
- Describir el propósito de la herramienta XAML Live Reload.
- Describir el propósito de la herramienta XAML Previewer para Xamarin.Forms.
- Describir los espacios de nombres XAML.

Conceptos básicos de XAML

XAML es un lenguaje basado en XML creado por Microsoft como una alternativa al código de programación imperativo para poder crear objetos, inicializarlos y organizarlos en jerarquías de padres e hijos. XAML ha sido adaptado a diversas tecnologías del .NET Framework principalmente en *Windows Presentation Foundation (WPF)* donde XAML es utilizado para definir el diseño de interfaces de usuario. XAML también ha sido adaptado a *Silverlight*, *Windows Runtime*, la *Plataforma Universal de Windows (UWP)* y *Xamarin.Forms*.

En el archivo XAML, el desarrollador de Xamarin.Forms puede definir interfaces de usuario utilizando *Controles (Views)*, *Diseños (Layouts)*, *Páginas (Pages)* o clases personalizadas.

El archivo XAML puede ser compilado o incrustado en el archivo ejecutable. En cualquier caso, la información XAML es analizada en tiempo de compilación para localizar objetos que hayan sido nombrados y nuevamente en tiempo de ejecución para instanciar e inicializar objetos y para establecer enlaces entre estos objetos y el código de programación.

XAML tiene varias ventajas sobre el código imperativo equivalente.

- El código XAML suele ser más compacto y legible que el código equivalente.
- La jerarquía inherente Padre e Hijo de XML permite a XAML imitar con mayor claridad visual la jerarquía Padre e Hijo de los objetos de la interfaz de usuario.
- El código XAML puede ser fácilmente escrito de forma manual por los programadores, pero también puede ser generado por herramientas visuales como *Microsoft Blend*.



Al momento de crear este entrenamiento no existe aún un Diseñador Visual para generar código XAML para aplicaciones Xamarin.Forms por lo que todo el código debe ser hecho a mano.

XAML también tiene desventajas, la mayoría relacionadas con limitaciones que son intrínsecas de los lenguajes de marcado:

- XAML no puede contener código imperativo. Todos los controladores de eventos deben ser definidos en un archivo de código.
- XAML no puede contener ciclos para procesamiento repetitivo. Sin embargo, varios objetos visuales de Xamarin.Forms pueden generar múltiples elementos como por ejemplo el objeto *ListView* que puede generar varios elementos dependiendo de los objetos en su colección *ItemsSource*.



- XAML no puede contener procesamiento condicional, sin embargo, un enlace de datos puede hacer referencia a un convertidor de enlace basado en código que permite un procesamiento condicional de forma eficaz.
- Generalmente, XAML no puede crear instancias de clases que no definen un constructor sin parámetros. Sin embargo, algunas veces existe un mecanismo para evitar esta restricción.
- Generalmente, XAML no puede invocar métodos. Sin embargo, algunas veces también existe un mecanismo para evitar esta restricción.
- Aún no existe un diseñador visual para generar código XAML en aplicaciones Xamarin.Forms.

XAML es básicamente XML, pero XAML tiene algunas características de sintaxis únicas. Las más importantes son:

- Elementos de Propiedad (*Property Element*) que permite asignar valores distintos de texto a atributos.
- Propiedades adjuntas.
- Extensiones de marcado.

Laboratorio

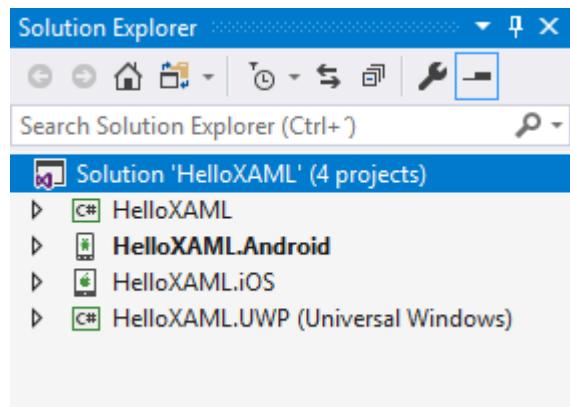
Examinando la anatomía de un archivo XAML

En una aplicación Xamarin.Forms, XAML se usa principalmente para definir el contenido visual de una página y funciona junto con un archivo de código subyacente (*code-behind*) C#.

El archivo de código subyacente proporciona soporte de código para el marcado. Juntos, los dos archivos contribuyen a una nueva definición de clase que incluye vistas hijas y la inicialización de propiedades. En el archivo XAML, se hace referencia a las clases y propiedades con elementos y atributos XML y se establecen los enlaces entre el código y el marcado.

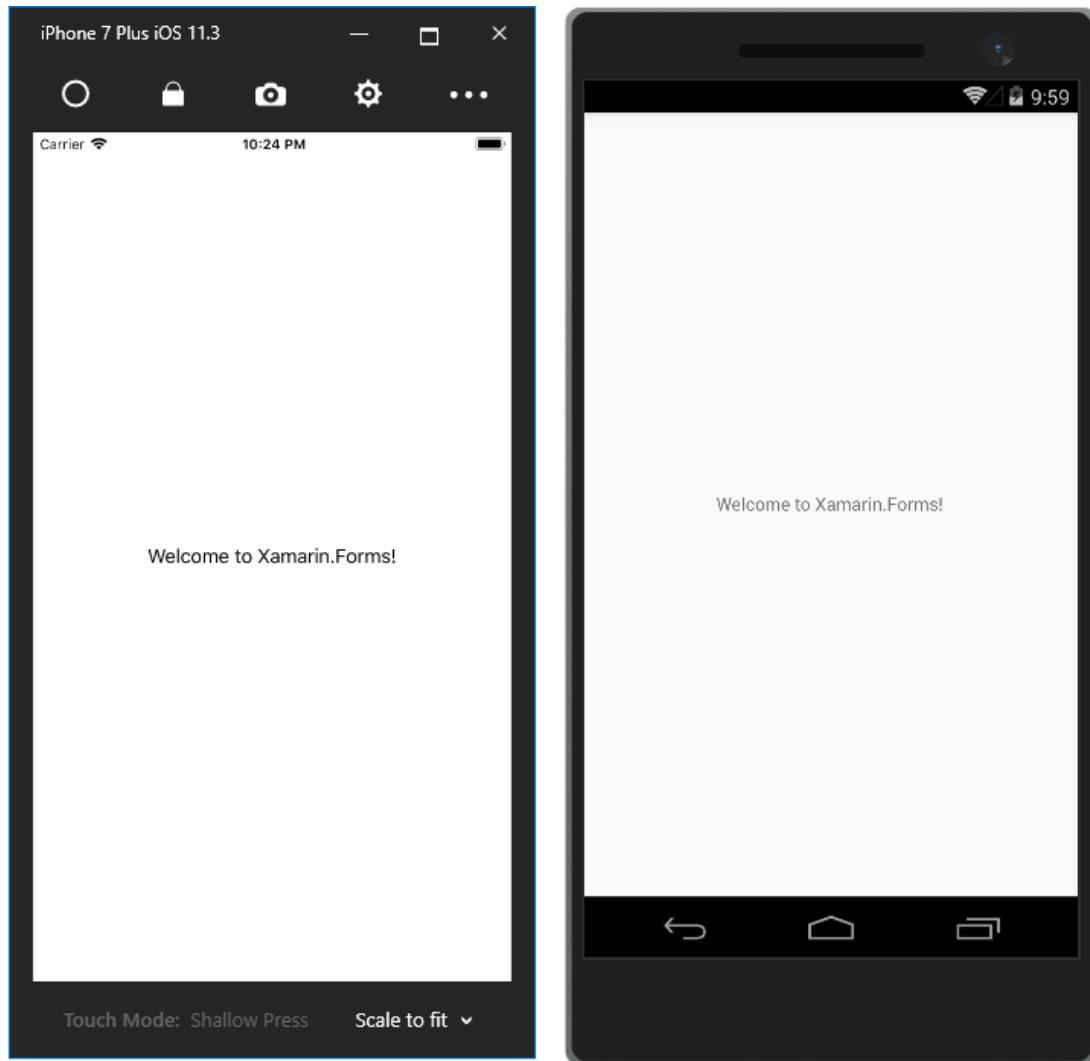
En este laboratorio examinaremos la anatomía de un archivo XAML.

1. Abre Visual Studio bajo el contexto del administrador.
2. Crea una nueva solución Xamarin.Forms vacía utilizando .NET Standard como la estrategia para compartir código y con destino para al menos 2 plataformas. El explorador de soluciones mostrará una estructura similar a la siguiente



3. Actualiza el paquete NuGet **Xamarin.Forms** a su versión más reciente.
4. Ejecuta tu aplicación para verificar que todo funcione correctamente.

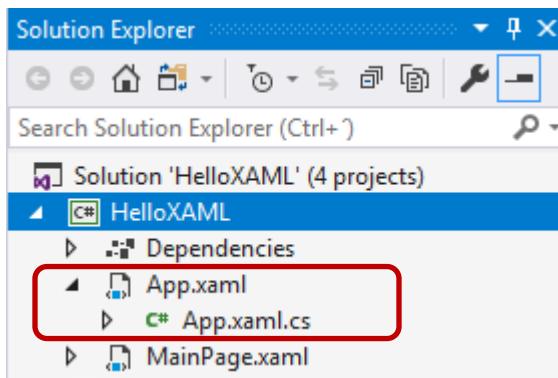
Las siguientes imágenes muestran la ejecución de la aplicación en iOS y Android.



Examinemos ahora la anatomía de un archivo XAML.

5. En el proyecto compartido existen dos archivos con los siguientes nombres:

- ***App.xaml***, un archivo XAML.
- ***App.xaml.cs***, un archivo de código subyacente C# asociado al archivo XAML.



Los dos archivos **App.xaml** y **App.xaml.cs** contribuyen una clase llamada **App**.

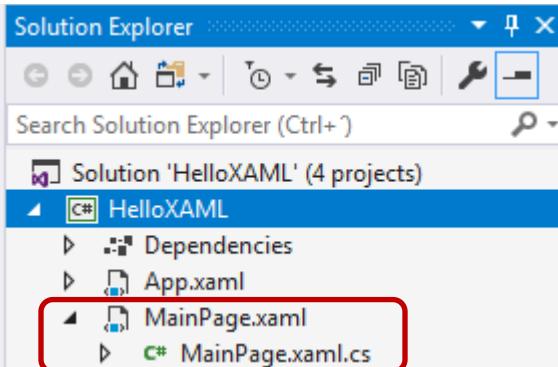
6. Haz clic sobre el archivo **App.xaml.cs** para abrirlo y observa la definición de la clase **App**.

```
public partial class App : Application
```

Puedes observar que la clase **App** deriva de la clase **Xamarin.Forms.Application**.

La mayoría de las clases de otros archivos XAML contribuyen una clase que deriva de **ContentPage**, estos archivos utilizan XAML para definir el contenido visual de una página. Tal es el caso de los otros dos archivos del proyecto:

- **MainPage.xaml**, un archivo XAML.
- **MainPage.xaml.cs**, un archivo de código subyacente C# asociado al archivo XAML.



Examinemos el contenido de un archivo XAML.

7. Haz clic sobre el archivo **MainPage.xaml** para abrirlo en el editor de código XAML.

Las dos primeras declaraciones de espacio de nombres XML (xmlns) hacen referencia a URIs.

```
xmlns="http://xamarin.com/schemas/2014/forms"  
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```



Aunque aparentemente apuntan a recursos de los sitios web xamarin.com y microsoft.com, realmente no hay algo en ese lugar. Son simplemente URIs propiedad de Xamarin y Microsoft que funcionan simplemente como identificadores de versión.

La primera declaración de espacio de nombres XML significa que las etiquetas definidas en el archivo XAML y que no tienen un prefijo, se refieren a clases en Xamarin.Forms, por ejemplo, las etiquetas **ContentPage**, **StackLayout** y **Label**.

La segunda declaración de espacio de nombres define un prefijo de **x**.

```
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

Este prefijo es utilizado por varios elementos y atributos que son intrínsecos del mismo XAML y que son soportados por otras implementaciones de XAML. Sin embargo, estos elementos y atributos son ligeramente diferentes dependiendo del año especificado en el URI. Xamarin.Forms soporta la especificación XAML 2009, pero no todo de ella.

8. Examina la siguiente línea de código.

```
xmlns:local="clr-namespace:HelloXAML"
```

La declaración del espacio de nombres **local**, nos permite obtener acceso a otras clases que sean definidas en el proyecto .NET Standard.

9. Examina la línea final de la etiqueta **ContentPage**.

```
x:Class="HelloXAML.MainPage"
```

Puedes notar que el prefijo **x** se utiliza para un atributo llamado **Class**. Dado que el uso del prefijo **x** es prácticamente universal para el espacio de nombres XAML, los atributos XAML como **Class** casi siempre se conocen como **x:Class**.

El atributo **x:Class** especifica un nombre de clase .NET completamente calificado, esto es, incluye el nombre de la clase y su espacio de nombres, en este caso **HelloXAML.MainPage**. Esto significa que este archivo XAML define una nueva clase llamada **MainPage** en el espacio de nombres **HelloXAML** que deriva de **ContentPage**, la etiqueta en la que el atributo **x:Class** aparece.

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

El atributo **x:Class** solo puede aparecer en el elemento raíz de un archivo XAML para definir una clase derivada C#. Ésta es la única nueva clase que es definida en el archivo XAML. Todo



lo demás que aparece en el archivo XAML son simplemente clases existentes que son instanciadas e inicializadas.

10. Abre el archivo **MainPage.xaml.cs**.

Puedes observar que la clase **MainPage** deriva de la clase **ContentPage** y que además es una clase **Partial**.

```
public partial class MainPage : ContentPage
```

Al ser una clase **Partial**, significa que puede existir otra definición de clase **Partial** para **MainPage**.

11. Observa el código del constructor.

Puedes observar la existencia de una instrucción que hace un llamado al método **InitializeComponent**.

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

¿Dónde está la otra definición de la clase **Partial MainPage**? ¿Dónde se encuentra definido el método **InitializeComponent**?

Encontremos las respuestas.

12. En el Explorador de Windows, abre la carpeta ... \obj\Debug\netstandard2.0 de tu proyecto compartido.

Local Disk (C:) > Demos > HelloXAML > HelloXAML > HelloXAML > obj > Debug > netstandard2.0		
Name	Type	Size
App.xaml.g.cs	CS File	1 KB
HelloXAML.AssemblyInfo.cs	CS File	1 KB
HelloXAML.AssemblyInfoInputs.cache	CACHE File	1 KB
HelloXAML.csproj.CoreCompileInputs.cache	CACHE File	1 KB
HelloXAML.csproj.FileListAbsolute.txt	Text Document	1 KB
HelloXAML.csprojAssemblyReference.cache	CACHE File	183 KB
HelloXAML.dll	Application extens	6 KB
HelloXAML.pdb	Program Debug D...	1 KB
MainPage.xaml.g.cs	CS File	1 KB



Cuando Visual Studio compila el proyecto, el archivo XAML es analizado para generar un archivo de código C# con el mismo nombre, pero con la extensión **xaml.g.cs**. La letra “**g**” es la sigla de “generated”.

13. Localiza y abre el archivo **MainPage.xaml.g.cs**.

Puedes notar que ahí se encuentra la otra definición de clase parcial **MainPage** que contiene la definición del método **InitializeComponent** invocado desde el constructor de la clase **MainPage**.

```
[global::Xamarin.Forms.Xaml.XamlFilePathAttribute("MainPage.xaml")]
public partial class MainPage : global::Xamarin.Forms.ContentPage {

    [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tas
    private void InitializeComponent() {
        global::Xamarin.Forms.Xaml.Extensions.LoadFromXaml(this, typeof(MainPage));
    }
}
```

Estas dos definiciones de clase parcial de **MainPage**, se compilan juntas. Dependiendo de si se compila el código XAML o no, el archivo XAML o el formato binario del archivo XAML compilado es incrustado en el archivo ejecutable.

En tiempo de ejecución, el código en el proyecto de cada plataforma particular invoca al método **LoadApplication**, pasándole una nueva instancia de la clase **App** definida en la biblioteca .NET standard.

El constructor de la clase **App** crea una instancia de la clase **MainPage**. El constructor de esa clase invoca al método **InitializeComponent**, el cual invoca al método **LoadFromXaml** que se encarga de extraer el archivo XAML (o su binario compilado) de la biblioteca .NET Standard.

El método **LoadFromXaml** Inicializa todos los objetos definidos en el archivo XAML, los conecta a todos en conjunto en relaciones Padre e hijo, adjunta los controladores de eventos definidos en el código a los eventos establecidos en el archivo XAML y establece el árbol de objetos resultante como el contenido de la página.

Aunque normalmente no es necesario manipular los archivos de código generados, en ocasiones, en tiempo de ejecución se producen excepciones en el código de estos archivos por lo que debemos estar familiarizado con ellos.

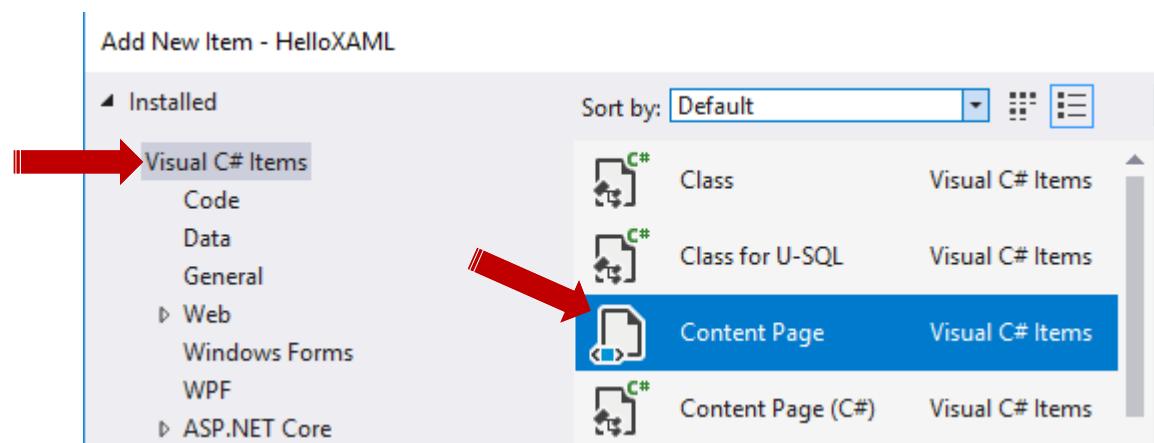
Laboratorio

Agregando nuevas páginas XAML

Una vez que conocemos la anatomía de un archivo XAML, es momento de aprender a agregar más páginas a nuestra aplicación.

En este laboratorio exploraremos la forma de agregar nuevas páginas a una aplicación Xamarin.Forms.

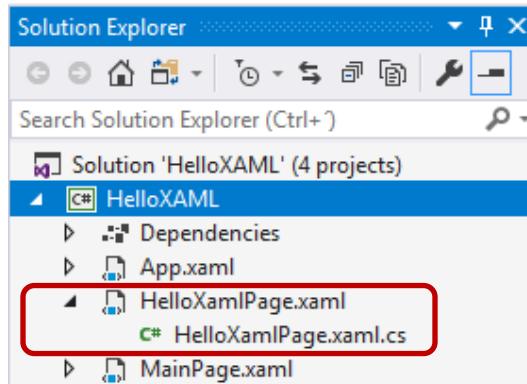
1. Abre la solución del laboratorio anterior en caso de que la hayas cerrado.
2. Selecciona la opción **Add > New Item...** del menú contextual del proyecto compartido Xamarin.Forms.
3. Selecciona la plantilla **Content Page**. Esta plantilla nos permite agregar una página para mostrar contenido utilizando XAML. La plantilla se basa en la clase **ContentPage**.



Puedes también notar la existencia de la plantilla **Content Page (C#)**. Esta plantilla nos permite agregar una página para mostrar contenido utilizando código C#.

Asigna un nombre a la página, por ejemplo, **HelloXamlPage.xaml**. El explorador de soluciones mostrará la página agregada.

Puedes notar que se han agregado 2 archivos al proyecto, **HelloXamlPage.xaml** y **HelloXamlPage.xaml.cs**.



4. Elimina el elemento **StackLayout**. El código será similar al siguiente.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="HelloXAML.HelloXamlPage">
    <ContentPage.Content>
        </ContentPage.Content>
    </ContentPage>
```

Las etiquetas **ContentPage.Content** son llamadas *Elemento de Propiedad (Property element)*. **Content** es una propiedad de **ContentPage** y generalmente se le establece un único elemento visual (*View*) o un *Layout* con *Views* hijos. Normalmente las propiedades de objetos se convierten en atributos en XAML, pero sería difícil asignar un objeto complejo a un atributo *Content*. Por esta razón, la propiedad es expresada como un elemento XML que consiste en el nombre de la clase y en el nombre de la propiedad separados por un carácter punto. El valor de la propiedad **Content** puede ser establecido dentro de las etiquetas **ContentPage.Content**.

5. Agrega el siguiente código dentro de las etiquetas **ContentPage.Content**. El código asigna un *View Label* como valor de la propiedad *Content* del objeto *ContentPage*.

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

6. Agrega el siguiente atributo a la etiqueta raíz **ContentPage** para establecer el valor de la propiedad **Title** de **ContentPage**.

```
Title="Hello XAML Page!"
```



En este momento podemos resumir que una clase de Xamarin.Forms como *ContentPage* o *Label* se representa en el archivo XAML como un elemento XML. Las propiedades de la clase como por ejemplo **Title** de *ContentPage* o **Text** del *Label* aparecen como atributos XML.

Algunas propiedades son tipos de datos básicos como las propiedades **Title** y **Text** que son de tipo **String**, **Rotation** es de tipo **Double**, y la propiedad **IsVisible** es de tipo **Boolean**.

La propiedad **HorizontalTextAlignment** es de tipo **TextAlignment**, que es una enumeración. Para una propiedad de cualquier tipo de enumeración, lo único que necesitamos proporcionar es el nombre de un miembro.

Para las propiedades de tipos más complejos se utilizan convertidores (*converters*) para que estos analicen el código XAML. Los convertidores son clases de Xamarin.Forms que derivan de la clase **TypeConverter**. Muchas son clases públicas pero otras no. Para nuestro ejemplo de código XAML agregado, algunas de estas clases *Converter* desempeñan un papel importante detrás de escena.

- **LayoutOptionsConverter** para la propiedad *VerticalOptions*
- **FontSizeConverter** para la propiedad *FontSize*
- **ColorTypeConverter** para la propiedad *TextColor*

Estos convertidores rigen la sintaxis permitida para los valores de la propiedad.

Por ejemplo, el convertidor **ThicknessTypeConverter** utilizado para definir márgenes de elementos puede manejar uno, dos o cuatro números separados por comas. Si un número es proporcionado, se aplica a los cuatro lados. Con dos números, el primero es el valor de la izquierda y de la derecha mientras que el segundo número es el valor de la parte superior e inferior. Cuando se especifican 4 números, el orden en que se aplican es Izquierda, Arriba, derecha y abajo.

El convertidor **LayoutOptionsConverter** puede convertir los nombres de los campos estáticos públicos de la estructura **LayoutOptions** a valores de tipo **LayoutOptions**.

El convertidor **FontSizeConverter** puede manejar un miembro de tipo **NamedSize** o un tamaño de fuente numérico.

El convertidor **ColorTypeConverter** acepta los nombres de los campos estáticos públicos de la estructura **Color** o los valores RGB hexadecimales, con o sin un canal alfa, precedido por un signo de número (#). Este es un ejemplo de la sintaxis sin un canal alfa:

TextColor="#rrggbb"

Para el canal alfa, debemos tomar en cuenta que **FF** es completamente opaco y **00** es totalmente transparente.

Existen otras dos sintaxis que nos permiten especificar un único dígito hexadecimal para cada canal:

TextColor="#rgb" *TextColor="#argb"*

En estos casos, el dígito es repetido para formar el valor. Por ejemplo, #CF3 es el color RGB #CCFF33.

7. Ejecuta la aplicación y observa el resultado. Puedes notar que el contenido de la página **MainPage** es mostrado en lugar de la nueva página. El siguiente paso será establecer la nueva página de inicio o navegar a la nueva página desde **MainPage**. Esto lo haremos en el siguiente laboratorio.

Laboratorio

Agregando navegación entre páginas.

Para mostrar una nueva página agregada al proyecto compartido Xamarin.Forms podemos establecer la nueva página como página de inicio en el archivo **App.xaml.cs** o navegar a la nueva página desde **MainPage**.

En este laboratorio exploraremos la forma de navegar hacia la nueva página agregada en el laboratorio anterior.

1. Abre el archivo **App.xaml.cs**.

Para implementar navegación entre páginas es necesario modificar el constructor de la clase **App** para crear un objeto **NavigationPage**.

2. Modifica el código del constructor de la clase **App** para establecer un nuevo objeto **NavigationPage** a la propiedad **MainPage**.

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

El objeto **NavigationPage** es un tipo **Page** que administra la navegación. En su constructor le podemos pasar la instancia de la página raíz.

Con este cambio, la página **MainPage** será mostrada al iniciar la aplicación, pero desde esa página podríamos tener un botón que nos permita navegar hacia la nueva página.

3. Abre el archivo **MainPage.xaml.cs**.

4. Agrega el siguiente código al final del constructor de la clase **MainPage** para crear un objeto **Button** sencillo.

```
Button NavigateButton = new Button  
{  
    Text = "¡Navegar!",  
    HorizontalOptions = LayoutOptions.Center,  
    VerticalOptions = LayoutOptions.Center  
};
```

Cuando el usuario toque el botón queremos navegar a la nueva página por lo que será necesario agregar un manejador del evento **Clicked** del elemento **Button**.

-
5. Al final del constructor de la clase **MainPage**, agrega el siguiente código que define el manejador del evento **Clicked** del objeto **Button**.

```
NavigateButton.Clicked += async (sender, e) =>
{
    await Navigation.PushAsync(new HelloXamlPage());
};
```

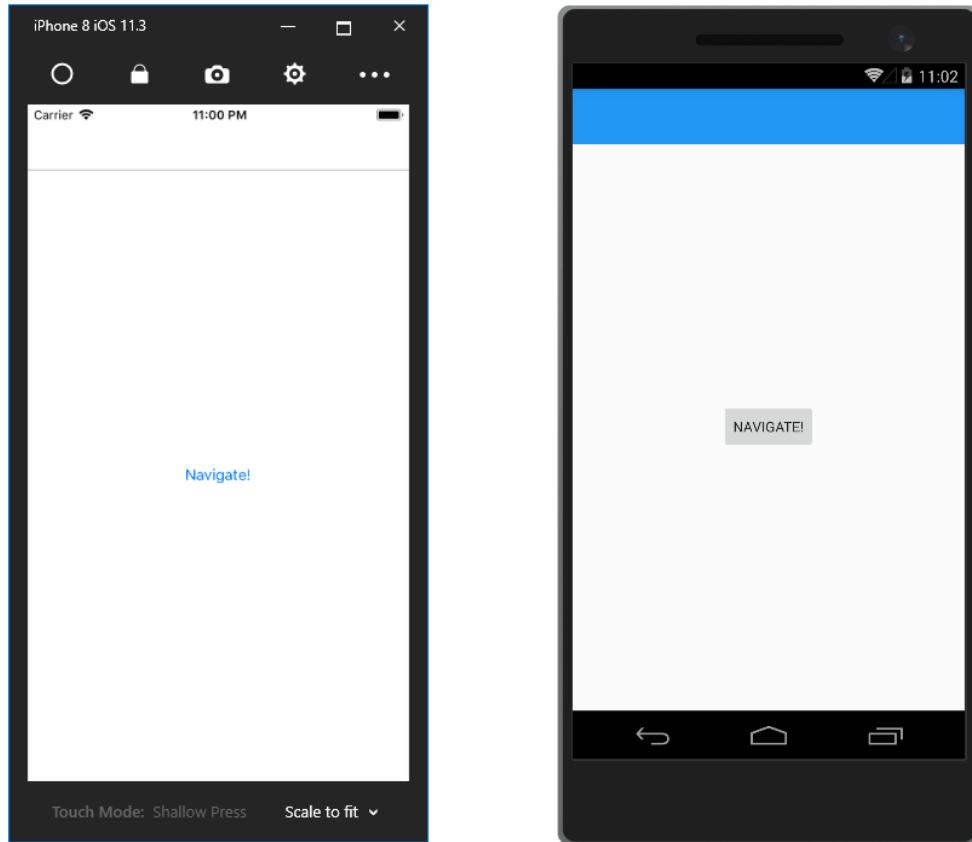
El siguiente paso será agregar el objeto **Button** a la jerarquía de elementos XAML de la página. Con fines demostrativos, agregaremos código que reemplace todo el contenido actual de **MainPage**. Actualmente **MainPage** tiene un elemento **StackLayout** como contenido.

6. Agrega el siguiente código al final del constructor de la clase **MainPage**.

```
Content = NavigateButton;
```

Al asignar un valor a la propiedad **Content** en el código C#, el valor establecido en la propiedad **Content** en el archivo XAML será remplazado.

7. Ejecuta la aplicación. Puedes notar que ahora aparece un botón en la pantalla en lugar del mensaje "Welcome to Xamarin.Forms!" definido en el archivo XAML como se muestra en las siguientes imágenes en iOS y Android respectivamente.



8. Toca el botón para navegar a la página **HelloXamlPage**. La página será mostrada como en las siguientes imágenes en iOS y Android respectivamente.