ÍNDICE

PREFACIO

Vivimos en la Era de la Transformación Digital, donde el World Economic Forum habla de que estamos transcurriendo la Cuarta Revolución Industrial[1]. Estas son denominaciones que representan el impacto de la tecnología en las personas, empresas, gobiernos y modelos de negocios, no solo por los cambios, desafíos y oportunidades que generan en cada uno de estos ámbitos, sino por la velocidad sin precedentes con la que estos cambios suceden.

Fenómenos como el poder del Cloud Computing, la disponibilidad y accesibilidad del acceso a Internet, la movilidad y proliferación de múltiples tipos de dispositivos móviles, sumados a los más recientes como la "Inteligencia Artificial", la realidad virtual y realidad aumentada, son factores de innovación, de creación nuevas oportunidades, disrupción de modelos de negocios y creación de nuevas experiencias para los usuarios.

Si hablamos de experiencias para los usuarios, los Smartphones son hoy el primer punto de comunicación e interacción rica de empresas y servicios con usuarios a través de Apps.

Hoy, los desarrolladores de Apps enfrentan el desafío de la heterogeneidad de dispositivos y tecnologías: iOS, Android y Windows componen el mercado de Smartphones. Desarrollar para múltiples plataformas es complejo y costoso. Conocer múltiples sistemas operativos y lenguajes de programación, adquirir los conocimientos necesarios y tenerlos actualizados, mantener múltiples versiones de

[1] https://www.weforum.org/agenda/2016/01/the-fourth-industrial-revolution-what-it-means-and-how-to-respond/

código y diferentes ambientes y administrar varias distribuciones y actualizaciones son parte de los retos a los que los desarrolladores deben enfrentarse.

Xamarin.Forms tiene un lugar de privilegio en este contexto, permitiendo desarrollar, mantener y administrar una línea base de código para iOS, Android y Windows, para luego generar aplicaciones nativas para cada plataforma, logrando que la App tenga la performance y experiencia en cada dispositivo, dando adicionalmente al desarrollador el poder de la reutilización, la productividad y la optimización en la calidad del código, y en consecuencia de la calidad de la aplicación.

Este libro aporta un enorme valor a la comunidad desarrolladora de habla hispana, permitiendo a desarrolladores tener una didáctica, sólida y completa referencia para poder tomar el máximo potencial de Xamarin.Forms y en consecuencia tener éxito en el desarrollo de aplicaciones móviles para múltiples plataformas.

Eduardo Mangarelli
Director de Tecnología e Innovación de Microsoft Latinoamérica
Ingeniero de Sistemas de la Universidad ORT de Uruguay
Profesor de la Universidad ORT de Uruguay
Inversor y Advisor de múltiples empresas

PRÓLOGO

A lo largo de varios años, he tenido la oportunidad de participar en una gran cantidad de proyectos de software dentro y fuera de México, para empresas privadas y gubernamentales. En los últimos dos años, muchos de esos proyectos han sido exitosas aplicaciones móviles multiplataforma de negocio construidas con Xamarin.Forms, por lo que de primera mano estoy convencido y he podido apreciar que Xamarin.Forms es una excelente opción tecnológica en el momento de decidir el rumbo por el que irán las soluciones móviles en tu empresa o institución.

El libro que tienes en tus manos es el resultado de varios meses de arduo trabajo y desvelo. El haber decidido escribir esta obra tiene la firme intención de inspirar a los lectores a construir con calidad todo tipo de aplicación móvil de negocio con Xamarin.Forms.

¿QUIÉN DEBERÍA LEER ESTE LIBRO?

Para que puedas sacar el máximo provecho, este libro presupone que ya tienes experiencia con el lenguaje de programación C# y la plataforma de desarrollo .NET con Visual Studio .NET. Es ideal –mas no un prerrequisito– contar con experiencia previa en el desarrollo de aplicaciones con XAML (WPF, Silverlight, Windows Phone o Universal Windows Platform). El libro está enfocado a desarrolladores amateurs que deseen aprender a construir aplicaciones multiplataforma con Xamarin.Forms, y a desarrolladores profesionales de empresas que deseen iniciar o perfeccionar su estrategia de aplicaciones móviles multiplataforma con esta fascinante tecnología.

EL PROYECTO

A lo largo de los capítulos 4 al 11, construimos una aplicación móvil multiplataforma para la captura de encuestas. Si bien la aplicación hace encuestas

acerca de cuál es el equipo de fútbol favorito de la gente, fácilmente podrías adaptar esta aplicación a cualquier otro tipo de encuesta con muy poco esfuerzo.

SOFTWARE UTILIZADO

Este libro está basado en la versión estable de Xamarin.Forms 2.3.3.180, Prism para Xamarin.Forms 6.3.0 pre-1 y sqlite-net-pcl 1.2.1. Además, se ha usado Visual Studio .NET 2015 Enterprise Update 3, en inglés. Muy probablemente, en el momento de estar leyendo estas líneas, ya tengas a tu alcance versiones más nuevas de los paquetes de NuGet requeridos, así como también ya puedas descargar Visual Studio .NET 2017 RTM.

ESTRUCTURA DEL LIBRO

El libro está organizado y estructurado de tal manera que pueda leerse en orden de inicio a fin, ya que gradualmente en cada capítulo (a partir del 4) implementamos funcionalidades adicionales y modificaciones en la aplicación de encuestas. Para sacar el máximo provecho te sugiero leer los capítulos en orden y sobre todo practiques el código relacionado con cada uno de ellos.

CÓDIGO FUENTE

Todo el código fuente de este libro, así como el script de la base de datos de encuestas, lo puedes descargar de https://github.com/rdiazconcha.

1 INTRODUCCIÓN A XAMARIN

¡Bienvenido al mundo de desarrollo profesional con Xamarin! Si estás leyendo esta primera página seguramente eres de los(as) que les gusta comenzar a entender a detalle las bases de la plataforma de desarrollo que usarás para construir fantásticas soluciones. Yo soy igual que tú, así que te invito a que comencemos desde el principio: ¿qué es Xamarin?

¿Qué es Xamarin?

Para poder comprender bien qué es Xamarin debemos remontarnos a la historia de .NET.

Si ya tienes algunos años el mundo de desarrollo de aplicaciones para el ecosistema Windows, recordarás que en el año 2000 Microsoft anunció la plataforma de desarrollo .NET, la cual fue promovida como una plataforma de desarrollo cuyo objetivo era poder ejecutar aplicaciones en una gran cantidad de dispositivos y sistemas operativos diversos "basada en estándares de Internet" (lo que significara en aquel momento dicha frase rimbombante). En diciembre de ese año, la Infraestructura de Lenguaje Común o CLI –parte fundamental y corazón de .NET– fue publicada como el estándar abierto ECMA-335, abriendo un gran abanico de oportunidades para aquellos que quisieran hacer una implementación independiente. Este estándar define cómo las aplicaciones escritas en múltiples lenguajes de alto nivel pueden ejecutar en diferentes ambientes, sin la necesidad de reescribir esas aplicaciones para tomar en consideración las características únicas de dichos entornos. Una persona llamada Miguel de Icaza, de la empresa Ximian, se inspiró en esta especificación estándar y se dio a la tarea determinar si era posible implementarla también en el ecosistema Linux.

La implementación de esta especificación en el ecosistema Linux se llamó Mono, y permitía a los desarrolladores de ese sistema operativo usar un nuevo y novedoso lenguaje de programación llamado C# para poder construir aplicaciones. En el año 2001 se lanzó Mono como proyecto de código fuente abierto. En el mes de agosto del año 2003, la empresa Ximian –junto con sus fundadores Miguel de Icaza y Nat Friedman– fue adquirida por Novell (sí, aquella empresa responsable de su famoso sistema operativo de red Novell Netware), y al siguiente año (tres años después de haber lanzado el proyecto) Mono 1.0 era liberado en junio de 2004, mientras que Microsoft poco tiempo después lanzaba la versión 2.0 del .NET Framework. Algunos meses después, sería lanzada la versión 3.0 (cuyo nombre de producto originalmente era WinFX), versión que incluía las tecnologías Windows Communication Foundation (WCF), Windows Workflow Foundation (WF) y Windows Presentation Foundation (WPF).

La historia continuó con las versiones 3.5, 4.0, 4.5, etcétera. Microsoft al ser una empresa gigante, con recursos financieros y de capital humano muy grande, tuvo un ciclo de innovación mucho más rápido y mucho más contundente que el proyecto de código fuente abierto Mono, cuyo avance e innovación estaba supeditado a los esfuerzos, tiempo y energía de un limitado número de personas. Por tal motivo, nunca estaría a la par Mono con .NET en términos de funcionalidad e innovación. No obstante, Mono había logrado lo que Microsoft únicamente había podido soñar: el crear y ejecutar código verdaderamente multiplataforma, ya que Mono no se concentró solamente en el sistema operativo Linux, sino que su alcance incluía otras tecnologías y dispositivos como Android, PlayStation 3, Wii, Mac OSX y iOS. A través de los proyectos MonoTouch y MonoDroid ahora era posible escribir y ejecutar aplicaciones escritas con el lenguaje C# en los sistemas operativos móviles iOS y Android, respectivamente.

Eran tiempos fabulosos en el mundo tecnológico, y más para los amantes de .NET y C#.

Y pues, como en la mayoría de historias en los negocios: un pez grande se come a otro más pequeño. En el año 2011 la empresa Novell es adquirida por la empresa Attachmate, produciendo una gran cantidad de recortes de proyectos y personal, entre ellos las personas fundadoras de la empresa Ximian, que unos años atrás Novell adquirió. Pocas semanas después del recorte, sería dada a conocer la buena noticia de que Mono seguiría teniendo soporte a través de Xamarin, una empresa recién fundada y conformada por la gran mayoría de personas que fueron despedidas por parte de Attachmate.

Xamarin comenzó como una startup, obteniendo inyección financiera por parte de inversionistas externos, creando un portafolio de productos y servicios sumamente atractivos. No obstante, como cualquier empresa que está empezando,

el flujo de dinero es importante, así que las tecnologías que alguna vez fueron de código fuente abierto se perfilaron y aterrizaron como productos con un modelo de pago por anualidad, a cambio de soporte técnico, actualizaciones frecuentes, extraordinarias herramientas y una creciente y vibrante comunidad de seguidores. MonoTouch se convirtió en Xamarin.iOS, MonoDroid se convirtió en Xamarin.Android y MonoMac se convirtió en Xamarin.Mac. Su entorno de desarrollo MonoDevelop derivó en Xamarin Studio, con soporte tanto para Mac OSX como Windows.

Había comenzado otra etapa en la contienda de las plataformas de desarrollo multiplataforma para dispositivos móviles.

Ciertamente, Microsoft siempre miró de reojo la progresiva evolución e innovación de Mono y las tecnologías relacionadas. En el mes de febrero 2016, Microsoft anunciaría que había llegado a un acuerdo con la empresa Xamarin para poder adquirirla. Esa noticia, aunque ya era esperada por parte de la comunidad de .NET, fue sorprendente. Pero lo que fue aún más sorprendente fue que Microsoft anunciaba que haría de Xamarin un producto sin costo, incluido incluso en Visual Studio .NET Community (su edición gratuita). Pero si eso era sorprendente, lo que a continuación declaró Microsoft nos dejó a todos mudos: haría de Xamarin un proyecto Open Source. Microsoft con esta adquisición cumplía su sueño y profecía: el hacer realmente de .NET una plataforma de desarrollo para construir aplicaciones que ejecuten en un sinfín de sistemas operativos y dispositivos.

El resto de la historia se sigue escribiendo y la estamos viviendo: yo al escribir estas líneas y tú, al estar leyéndolas.

Bueno ya, entonces ¿qué es Xamarin?

Xamarin permite a los desarrolladores de C# poder construir aplicaciones multiplataforma para los sistemas operativos móviles y dispositivos más importantes del mundo: Android y iOS, además de poder compartir código con aplicaciones del ecosistema de Windows. Una de las características más importantes de Xamarin es que nos permite construir aplicaciones nativas con Interfaz de Usuario y desempeño nativo, tal cual como si estuviéramos desarrollando con Java para Android o con Objective-C o Swift para iOS, pero reutilizando al máximo nuestros conocimientos en el lenguaje C# y en la plataforma de desarrollo .NET.

Técnicamente hablando, Xamarin expone o "refleja" las APIs de cada sistema operativo subyacente, para poderlas utilizar con .NET, el CLR y el lenguaje C#. Adicionalmente a esta característica bien recibida por los desarrolladores de C#, lo que diferencia a Xamarin es su capacidad de compartir una base de código que pueda ser reutilizada por diferentes plataformas.

Métodos para compartir el código

En Xamarin hay tres métodos para compartir el código: Bibliotecas de Clases Portables (o PCL por sus siglas en inglés), los proyectos de Código compartido y los proyectos de .NET Standard.

BIBLIOTECAS DE CLASES PORTABLES

La Biblioteca de Clases Portable (o PCL por su nombre en inglés "Portable Class Library") es el mecanismo más usado y sugerido para construir aplicaciones con Xamarin, ya que compila un ensamblado de tipo DLL y resulta una estrategia más limpia en el momento de construir aplicaciones. Este es el método que utilizaremos a lo largo de todo este libro.

CÓDIGO COMPARTIDO

Con este método se crea un "concentrador" en donde se colocan archivos de código y son automáticamente vinculados a los diferentes proyectos que constituyen la solución. En este caso, el código no compila a un DLL por separado, sino que compila como parte de la aplicación concreta nativa. Debido a esto, el código rápidamente se puede inundar con bucles #IF…#ENDIF, haciendo su lectura un tanto menos eficiente. En mi particular experiencia, este mecanismo no lo recomiendo, a menos que se tenga una excelente razón para ello y que no se pueda resolver a través de una Biblioteca de Clases Portable.

.NET STANDARD

El .NET Standard es un conjunto de APIs que cada runtime de .NET debe implementar. En el momento de estar escribiendo este libro, la versión más ubicua es la 1.6. Por sus cualidades actuales, este no será el mecanismo que usaré en el libro, sin embargo, en un corto plazo definitivamente será la estrategia recomendada cuando se construyan aplicaciones multiplataforma.

El siguiente diagrama muestra el diseño lógico de una aplicación construida con Xamarin:

El diagrama muestra la estrategia "clásica" de construcción de aplicaciones con Xamarin: tener un código común que incluya la lógica de negocio, servicios, acceso a datos, almacenamiento, etcétera., y los elementos de la Interfaz de Usuario y lógica específica para cada sistema operativo subyacente en cada proyecto de la plataforma concreta.

Gracias a este mecanismo, podemos obtener un gran porcentaje de código común entre las diferentes plataformas, por lo que el mantenimiento de código se reduce y se incrementa la velocidad en la implementación de nuevas características o correcciones.

Sin embargo, esta estrategia o modelo clásico de construcción de aplicaciones con Xamarin rápidamente se vuelve no tan eficiente y ágil, sobre todo cuando se trata de aplicaciones de negocio, privadas, que no necesariamente requieren interfaces de usuario sofisticadas o de alta complejidad.

Xamarin se dio cuenta de esto y en el año 2014 creó un framework llamado Xamarin.Forms el cual es una abstracción de los principales elementos visuales de la Interfaz de Usuario de cada sistema operativo, maximizando aún más el código compartido que podemos tener en nuestros proyectos.

FUNDAMENTOS DE XAMARIN.FORMS

¿Qué es Xamarin.Forms?

Xamarin.Forms es un framework multiplataforma de elementos visuales, desplegable a través de NuGet. Estos elementos visuales son una abstracción de los elementos visuales nativos de cada sistema operativo concreto donde ejecuta la aplicación. Xamarin.Forms nos permite compartir hasta el 100% de código multiplataforma, ya que a diferencia del método "clásico", el código compartido también incluye las vistas o Interfaz de Usuario de la aplicación.

¿Xamarin o Xamarin.Forms?

Una de las principales dudas que se tienen alrededor del desarrollo de aplicaciones con Xamarin es si utilizamos Xamarin o Xamarin.Forms, como si se tratasen de dos productos completamente diferentes entre sí. Esto es completamente falso, y sospecho que esta duda está basada en cómo se entendió el mensaje cuando se lanzó la plataforma por primera vez. He escuchado a personas recomendar evitar Xamarin.Forms como opción para el desarrollo de aplicaciones multiplataforma. ¡No podrían estar más equivocados! En realidad, Xamarin.Forms es únicamente la abstracción de algunos de los elementos más relevantes en la Interfaz de Usuario de los diversos sistemas operativos que soporta, pero sigue siendo Xamarin e incluso no podría ejecutar sin él.

Mi recomendación a todo desarrollador profesional o amateur, empleado o independiente, que desee construir soluciones móviles de negocio y/o empresariales

con Xamarin es que comience con Xamarin.Forms y no con Xamarin "clásico", ya que la reutilización de código es mucho mayor, la dificultad se reduce y por lo tanto el costo y el tiempo asociados para construir dichas soluciones también se ven reducidos.

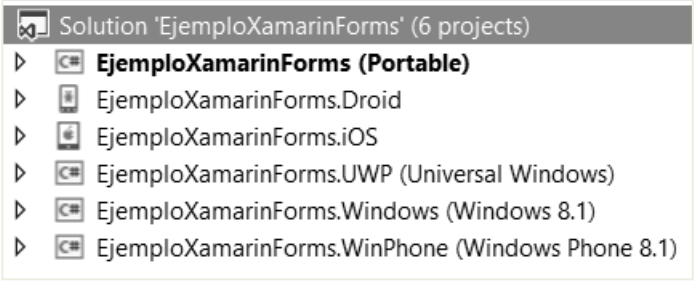
La siguiente tabla nos muestra una guía de cuándo utilizar Xamarin.Forms o cuándo utilizar Xamarin "clásico".

Xamarin.Forms	Xamarin.iOS / Xamarin.Android
• Apps de negocio • Apps de datos • Apps que requieran poca funcionalidad específica de cada plataforma • Apps donde compartir el código sea más importante que una IU sofisticada	• Apps que requieran interacción muy especializada • Apps que usen muchas APIs específicas de una plataforma • Apps en donde la IU sea más importante que compartir el código

ANATOMÍA DE UNA SOLUCIÓN

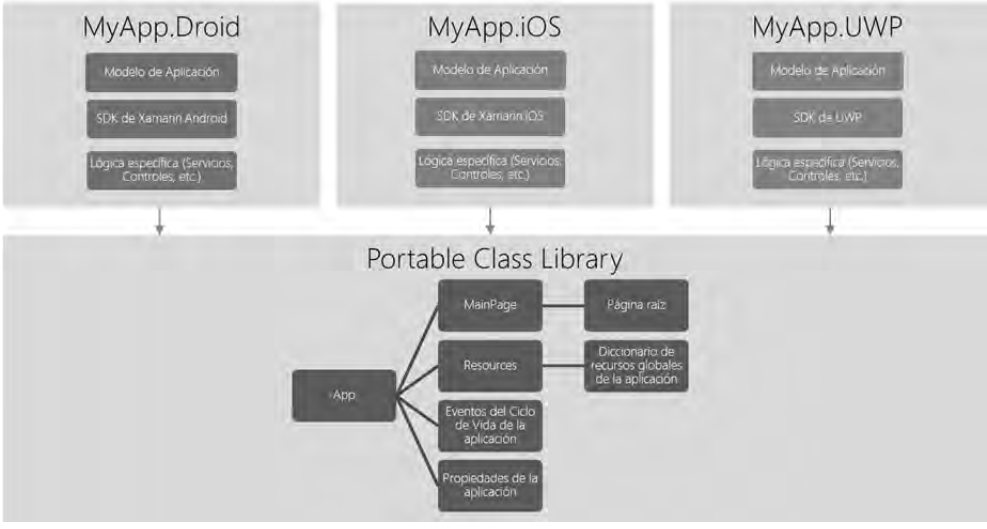
Cuando creamos una aplicación de Xamarin.Forms por medio de Visual Studio .NET usando la plantilla de Biblioteca de Clases Portables, Visual Studio crea diversos proyectos en la solución. El primer proyecto es la Biblioteca de Clases Portable la cual contendrá el código común y compartido multiplataforma que será usado por todo el resto de proyectos. Adicionalmente, Visual Studio .NET creará un proyecto por cada sistema operativo del que encuentre su SDK correctamente instalado y configurado. Ahora bien, nosotros podemos tener más control sobre qué proyectos exactamente crear, si creamos manualmente la solución y sus proyectos.

La siguiente figura muestra el Solution Explorer de Visual Studio .NET después de haber creado un proyecto llamado "EjemploXamarinForms", por medio de la plantilla "Blank Xaml App (Xamarin.Forms Portable)":



ARQUITECTURA DE XAMARIN.FORMS

Comprender a detalle la arquitectura de una aplicación de Xamarin.Forms es necesario para poder construir aplicaciones robustas y elegantes con esta tecnología. El siguiente diagrama muestra la arquitectura de una aplicación de Xamarin.Forms, la cual utiliza el método de Bibliotecas de Clases Portables para compartir el código y que apunta a tres sistemas operativos: Android, iOS y UWP.



Clase Application

Comenzamos hablando de la Biblioteca Clases Portable. Este proyecto contendrá la clase que es el punto de entrada para la aplicación de Xamarin.Forms. Esta clase (regularmente llamada App y contenida en el archivo físico App.cs) es de tipo Xamarin.Forms.Application.

La siguiente tabla describe los miembros más importantes de la clase App:

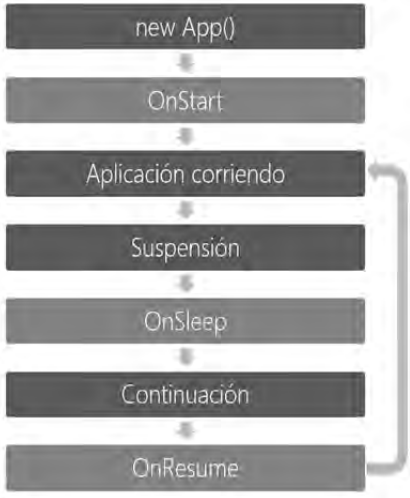
MainPage	Propiedad que representa la raíz visual de la aplicación
Resources	Propiedad de tipo KeyValuePair<string, object> que representa los recursos globales para la aplicación. Para más información acerca de los recursos, consulta el capítulo “El Lenguaje XAML” donde se describen con más detalle
OnStart()	Método virtual que se ejecuta cuando la aplicación se inicia por primera vez

OnSleep()	Método virtual que se ejecuta cuando la aplicación se está yendo a segundo plano
OnResume()	Método virtual que se ejecuta cuando la aplicación reanuda su ejecución después de haber estado en segundo plano
Properties	Propiedad de tipo IDictionary<string, object> en donde podemos almacenar cualquier tipo de objeto serializable para guardar el estado de la aplicación
SavePropertiesAsync()	Método que podemos ejecutar para guardar proactivamente en el almacenamiento local del dispositivo los objetos del diccionario Properties, y de esa manera evitar el riesgo de que no sean guardados ya sea porque ocurrió un error en la app o porque el dispositivo fue apagado en ese justo momento

Cada uno de los proyectos concretos referencia a la Biblioteca de Clases Portable, además de referenciar al SDK de Xamarin para cada sistema operativo. Es importante recordar que si bien Xamarin es una plataforma de desarrollo multiplataforma, cada sistema operativo tendrá sus particularidades, es decir, cada sistema operativo cuenta con un modelo de desarrollo específico.

Ciclo de vida

Como comentábamos anteriormente, el ciclo de vida de las aplicaciones con Xamarin.Forms está expuesto a través de los métodos OnStart(), OnSleep() y OnResume() de la clase Application. En cada método podemos escribir código que responda en el momento que el ciclo de vida tenga. Por ejemplo, si queremos escribir código que ejecute justo cuando inicia la aplicación, podemos ponerlo en el método OnStart(). De igual manera, si queremos ejecutar código en el momento que la aplicación se esté yendo a segundo plano, entonces lo pondríamos en el método OnSleep(), o si queremos ejecutar alguna lógica cuando la aplicación esté reanudando su ejecución después de haber estado en segundo plano, entonces ese código lo implementaríamos en el método OnResume(). El siguiente diagrama muestra el ciclo de vida completo de las aplicaciones con Xamarin.Forms:



Ejecutando Xamarin.Forms en cada proyecto concreto

Ejecutar Xamarin.Forms requiere que se cumplan los siguientes tres pasos:

1. Heredar de una clase base especial
2. Iniciar el motor de Xamarin.Forms
3. Cargar el objeto de tipo Application (App.cs) implementada en la Biblioteca de Clases Portable

A continuación describiremos con más detalle estos pasos requeridos para cada proyecto.

APLICACIÓN ANDROID

Para ejecutar una aplicación de Xamarin.Forms en Android, debemos realizar algunas modificaciones ligeras al proyecto de Xamarin.Android que haya creado la plantilla de Visual Studio .NET o que hayamos creado nosotros manualmente. En el proyecto de Android tenemos una clase llamada MainActivity, la cual es el punto de entrada de la aplicación. Es decir, esto es lo que primero que se interpreta del código cuando decidimos ejecutar nuestra app en el dispositivo. Para que Xamarin.Android ejecute Xamarin.Forms, esta clase MainActivity debe heredar de la clase base Xamarin.Forms.Platform.Android.FormsApplicationActivity. En el constructor, debemos además invocar el método Init() de la clase Xamarin.Forms.Forms y por último debemos ejecutar el método LoadApplication(), pasando como parámetro una instancia del objeto que representa la aplicación en la PCL. En el siguiente fragmento de código nos muestra la clase MainActivity una vez realizados los cambios necesarios para ejecutar Xamarin.Forms:

```
namespace MyApp.Droid
{
    [Activity(Label = "MyApp", Icon = "@drawable/icon",
MainLauncher = true, ConfigurationChanges =
ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
    public class MainActivity :
global::Xamarin.Forms.Platform.Android.FormsApplicationAc
tivity
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            global::Xamarin.Forms.Forms.Init(this,
bundle);
            LoadApplication(new App());
        }
    }
}
```

Nota: Es mejor idea usar la clase FormsAppCompatActivity, pero por cuestiones de sencillez en este momento usaremos FormsApplicationActivity. Más adelante en el libro, retomaremos este importante asunto.

APLICACIÓN IOS

En el proyecto de Xamarin.iOS, encontraremos una clase llamada AppDelegate la cual debe heredar de la clase base Xamarin.Forms.Platform.iOS.FormsApplicationDelegate. En el método FinishedLaunching(), debemos invocar el método Init() de la clase Xamarin.Forms.Forms e inmediatamente después debemos invocar el método LoadApplication(), pasando como parámetro el objeto que representa la aplicación en la PCL. El siguiente fragmento de código muestra estos cambios en la clase AppDelegate del proyecto de Xamarin.iOS:

```
namespace MyApp.iOS
{
    [Register("AppDelegate")]
    public partial class AppDelegate :
global::Xamarin.Forms.Platform.iOS.FormsApplicationDelega
te
    {
        public override bool
FinishedLaunching(UIApplication app, NSDictionary
options)
        {
            global::Xamarin.Forms.Forms.Init();
            LoadApplication(new App());
            return base.FinishedLaunching(app,
options);
        }
    }
}
```

APLICACIÓN UNIVERSAL WINDOWS PLATFORM (UWP)

En el proyecto de UWP, encontraremos la clase App en el archivo físico App.xaml.cs. Esta clase representa el punto de entrada de la aplicación UWP. En dicha clase, encontraremos el método OnLaunched() en donde debemos iniciar el motor de Xamarin.Forms a través de la ejecución del método Init() justo después de haber creado el objeto Frame y antes de establecerlo como la raíz visual de la ventana. Adicionalmente, la clase MainPage debe heredar de la clase base de WindowsPage, y en su constructor debemos de cargar el objeto que representa la aplicación en la PCL, a través de la ejecución del método LoadApplication().

El siguiente fragmento de código muestra los cambios necesarios tanto en la clase App como en MainPage del proyecto UWP:

```
//App.xaml.cs
 protected override void
OnLaunched(LaunchActivatedEventArgs e) {
    …
    rootFrame = new Frame();
```