

4TH EDITION

INTRODUCTION TO PROGRAMMING WITH XOJO



iOS

Addendum

RHINE, LEFEBVRE, PERLMAN

CONTENIDOS

a. Introducción

1. macOS y Xcode
 2. El Simulador iOS
 3. Controles y Eventos iOS
 4. Controles iOS en la Práctica
 5. Dispositivos y Ajustes de Compilación
-
6. Pantallas y Vistas
 7. iOS en la Práctica
 8. Depuración



Introducción

Bienvenido al Apéndice para de [*Introduction to Programming with Xojo!*](#)

Durante mucho tiempo sólo había dos formas de crear y desplegar apps para iPhone y iPad. Podías usar Xcode de Apple, siendo una herramienta muy potente, pero requería el tener que aprender uno de los lenguajes de programación de Apple. O también podías crear una app web con HTML y JavaScript, que funcionaba pero resultaba en una app que no se podía distribuir a través de la App Store y que, por lo general, ofrecía un peor rendimiento.

Se ha creado este apéndice para proporcionarte una introducción a la creación de apps para iPhone e iPad usando Xojo, lo que significa que puedes desplegar todos tus conocimientos y habilidades en Xojo para crear y desplegar apps en una de las principales plataformas móviles.

Para empezar hay algunas cuestiones que deben mencionarse. En primer lugar, Xojo sólo soporta actualmente iOS, de modo que no es posible crear apps nativas Xojo para Android u otras plataformas móviles por ahora. En segundo lugar, dado el modo en el que funcionan las Developer Tools de Apple, las apps iOS de Xojo sólo se pueden desarrollar en un Mac. Aprenderás más sobre esto en un momento.

Este apéndice es una introducción al desarrollo iOS con Xojo. Aprenderás como crear una app iOS y como probarla con el Simulador iOS de Apple. Otros temas, como la distribución de la app

en la App Store, están fuera del alcance de esta introducción pero hay algunos enlaces que te ayudarán con estas cuestiones avanzadas.

Este apéndice asume que ya has trabajado con la *Introducción a la Programación con Xojo* y que estás familiarizado con los fundamentos en el uso de Xojo.

1.1 macOS y Xcode

Como se indicaba anteriormente, desarrollar para iOS requiere de un Mac. Deberías de estar utilizando la última versión de macOS. También necesitarás instalar Xcode de Apple, el cual está disponible como descarga gratuita desde la Mac App Store.

El Programa de Desarrolladores de Apple cuenta con una membresía de pago, pero no es necesario que pagues para probar tus apps iOS. Si decides seguir adelante y desplegar tus apps en la App Store, entonces tendrás que invertir en una membresía de pago en dicho punto.

Xcode es el entorno de desarrollo recomendado por Apple. Es una herramienta muy potente, pero tienen una curva de aprendizaje elevada en comparación con Xojo. Sólo necesitarás usar Xcode, fundamentalmente, porque es la que incluye el Simulador iOS.

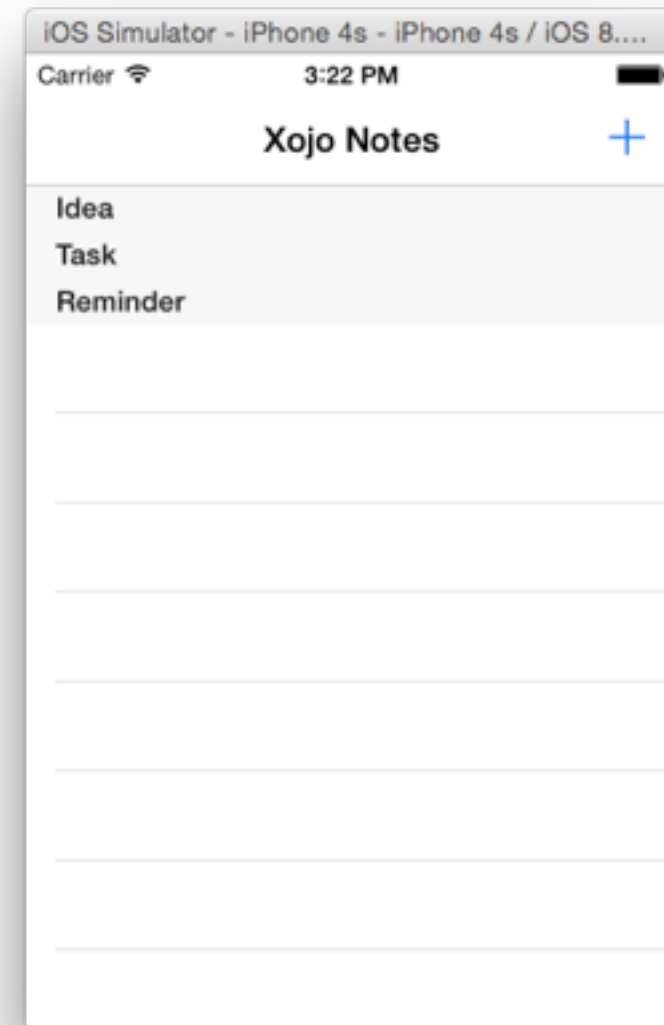
Ten en cuenta que la primera vez que ejecutes Xcode (tanto si lo ejecutas manualmente como si lo hace Xojo), tendrás que aceptar los términos de Apple.

1.2 El Simulador iOS

El Simulador iOS es la app macOS que ejecuta una versión reducida de iOS, la cual puedes usar para probar tus apps. Si recuerdas el modo en el que funcionan las apps desktop de Xojo, puedes hacer clic en el botón Run y ejecutar una versión temporal de tu app. En el caso de las apps iOS, esa versión temporal se ejecuta en el Simulador iOS.

En los primeros días de iOS, un desarrollador podía confiar en que cada iPhone tuviese el mismo tamaño y resolución de pantalla. Esto comenzó a cambiar en 2010 con la llegada de las pantallas retina y, en los últimos años, Apple ha publicado una variedad incluso más amplia de tamaños de pantalla y resoluciones.

Debido a esta variedad, querrás probar tu app en diferentes dispositivos. El Simulador iOS te permite configurar múltiples dispositivos iOS virtuales (tanto iPhone como iPad de varios tamaños), algo que aprenderás como hacer en la siguiente sección.



1.3 Controles iOS y Eventos

Antes de que comencemos con un proyecto de ejemplo rápido y sencillo, has de saber unas cuantas cosas sobre los controles en iOS. Estos son distintos de los controles desktop de varias formas significativas.

En primer lugar, dado que no hay apuntador de ratón en iOS, los controles tienen menos eventos. No hay evento `MouseEnter` o cualquier cosa relacionada con el ratón. La mayoría de los controles tienen eventos para `Open`, `Close` y `Action`.

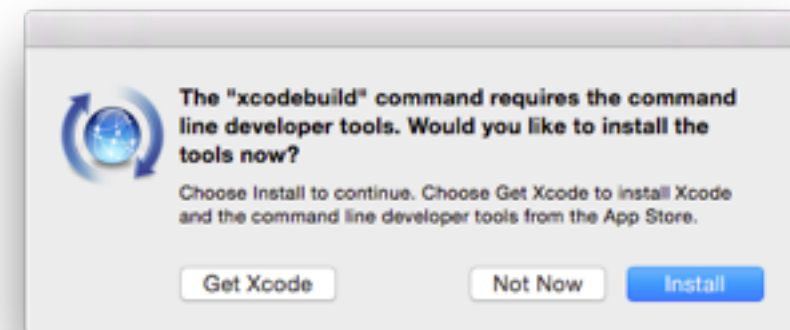
En segundo lugar, si bien Xojo sigue las coordenadas relacionadas con la posición del ratón en desktop, iOS no tiene dicha capacidad salvo que el dedo del usuario esté en contacto con la pantalla.

Por último, observarás que la lista de controles disponibles para iOS es mucho más reducida en comparación con desktop. Muchos de los controles desktop no tienen cabida en una pantalla táctil. Puede que se añadan más controles en futuras versiones de Xojo.

1.4 Controles iOS en la Práctica

Para crear una app iOS, empieza abriendo la aplicación Xojo. Te pedirá que selecciones una plantilla para un nuevo proyecto. Selecciona “iOS” y rellena los campos `Application Name`, `Company Name` y `Application Identifier`.

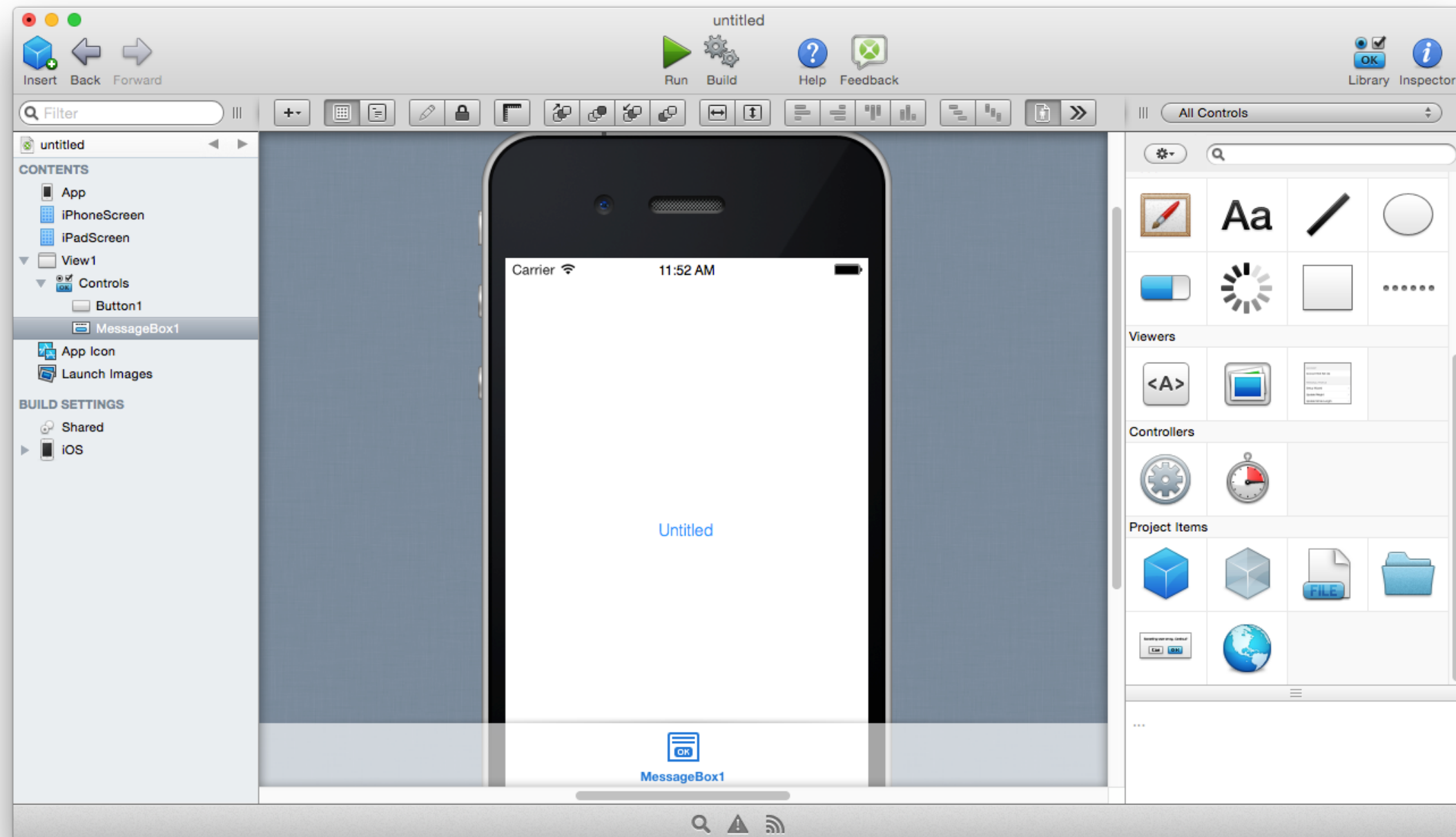
Observa que si Xcode no está instalado o debidamente configurado, entonces puede que veas un mensaje de error como este:



Asumiendo que todo está instalado, el texto introducido en `Application Name` puede ser el que quieras, así como el usado para `Company Name`. El campo `Application Identifier` debe estar, sin embargo, en el formato de “DNS inversa” de Apple. Por ejemplo, si una compañía fuese FooBar crease una app llamada

DreamCatcher, entonces el identificador podría ser com.foobar.dreamcatcher (como puedes ver, es prácticamente lo inverso a una dirección web, por lo que se denomina formato de DNS inversa).

Una vez que hayas completado los tres campo, pulsa OK. Se presentará la pantalla de bienvenida que te resultará muy parecida, pero distinta de forma sutil.



Verás la pantalla de un iPhone en Xojo en vez de la habitual pantalla de una ventana desktop. También observarás que el listado de controles no es tan amplio y variado como en desktop, tal y como se ha indicado anteriormente.

1) Busca el Botón en los controles y arrástralo a la View1.

Usa las guías para centrar el Botón en la Vista.

2) Busca el MessageBox en la lista de controles y arrástralo a la View1.

El control MessageBox se situará automáticamente en la parte inferior del tapiz, dado que se trata de un control no visual que sólo aparece cuando se necesita.

3) En el Inspector, cambiar la etiqueta del Botón para que sea “Say Hello.”

4) En el Inspector, cambia las siguientes propiedades en el MessageBox:

Message: “Nice to meet you!”

Title: “Hello, Mobile World!”

Left Button: “Cancel”

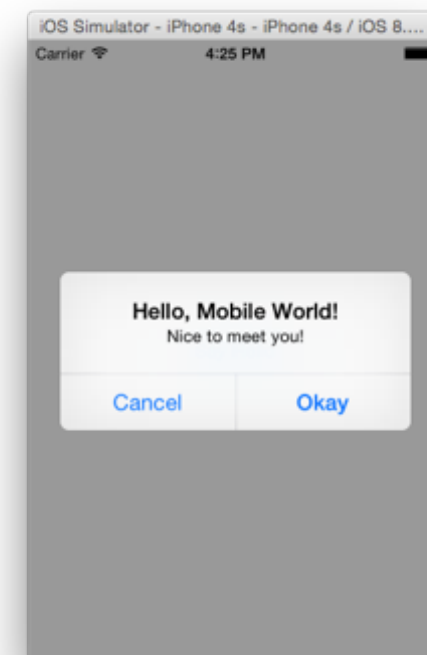
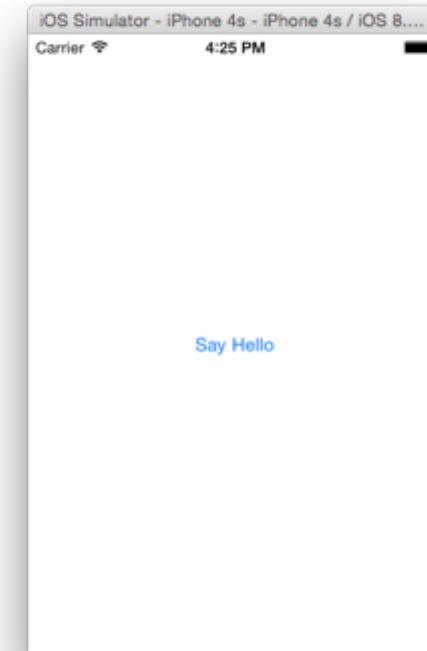
Right Button: “Okay”

5) Añade el siguiente código en el evento Action del Botón:

```
MessageBox1.Show
```

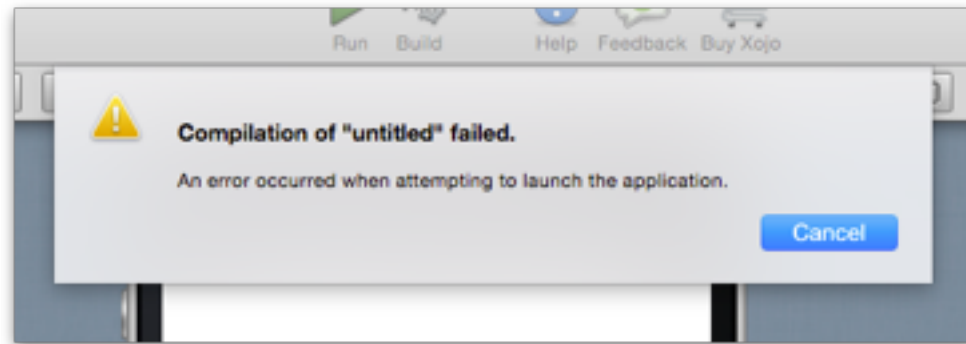
6) Ejecuta el proyecto.

Si tu ordenador está configurado correctamente, aparecerá el Simulador iOS y se ejecutará la app. Haz clic en el botón para ver como se muestra el mensaje “Hello, Mobile World”. Debería ser similar a la siguiente imagen.



7) Resolución de problemas

Si tu ordenador no está configurado correctamente, es posible que veas un mensaje de error. El siguiente mensaje de error indica que Xcode nunca se ha ejecutado (o bien que no se ha aceptado la licencia).



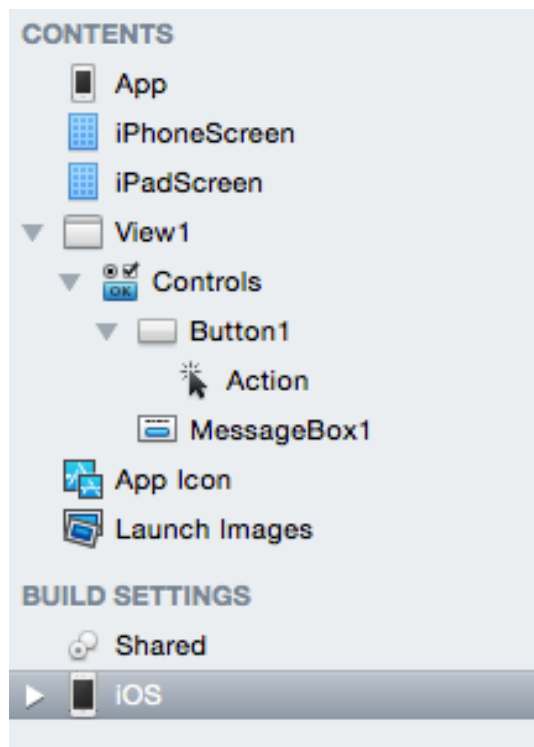
1.5 Dispositivos y Ajustes de Compilación

Como se indicaba anteriormente, puede usarse el Simulador iOS para probar una app en una variedad de dispositivos virtuales. La parte negativa es que tienes que configurar dichos dispositivos tu mismo.

Afortunadamente aun tienes el Simulador iOS ejecutándose gracias a la prueba de tu app "Hello Mobile World". Si es así, dirígete al menú Hardware del Simulador iOS. En Hardware, dirígete a Device y luego a Manage Devices. Esto ejecutará Xcode y mostrará la pantalla de gestión de dispositivos (Device Management).

En la pantalla Device Management, puedes crear tus propios dispositivos virtuales iOS. Los dispositivos hardware que elijas estarán limitados al hardware que soporte la actual versión de iOS de Apple, pero supondrán una forma fiable para determinar el modo en el que se presentará y comportará tu app en cada uno de ellos.

Cuando vuelvas a xojo, podrás elegir qué dispositivo utilizar para depurar en la sección iOS Build Settings del Navegador:



ID	
iOS App Name	My iOS App
File Types	Choose...
Build	
Bundle Identifier	com.bradrhine.myapp
Code Signing	
Team	None
Entitlements	...
Build For App Store	OFF
iOS Debugging	
Simulator Device	<input checked="" type="checkbox"/> iPhone 4s (iOS 8.1) <input type="checkbox"/> iPhone 5s (iOS 8.1) <input type="checkbox"/> iPhone 6 Plus (iOS 8.1) <input type="checkbox"/> iPhone 6 (iOS 8.1)

Cualquiera que sea el Dispositivo Simulador que elijas aquí, este será el que aparezca cuando ejecutes tu app iOS en el depurador, y tu app se escalará de acuerdo a dicho dispositivo.

1.6 Pantallas y Vistas

Una de las grandes diferencias entre el desarrollo para iOS y desktop se encuentra en la cantidad de información mostrada al usuario. En desktop tus usuarios ven una ventana, la cual es una porción de la pantalla.

En iOS no hay ventanas. El usuario no puede cambiar el tamaño de la interfaz y siempre está activa toda la pantalla. Dado que iOS sólo puede ejecutar una variedad de dispositivos y tamaños de pantalla, tu app ha de adaptarse a diferentes diseños y resoluciones. Esto es algo que se logra usando Screens (Pantallas) y Views (Vistas).

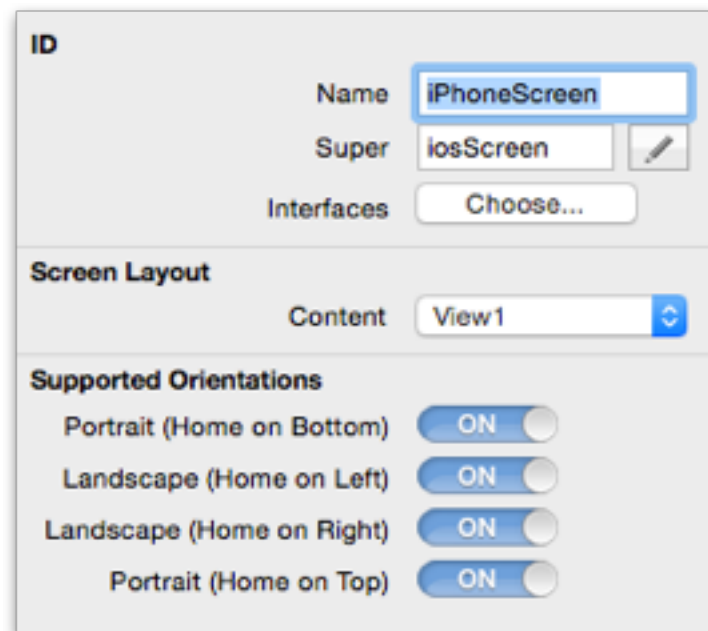
Resulta útil pensar en una Vista como la versión de una ventana en iOS. No es exactamente lo mismo, pero una Vista es donde construyes tu interfaz y defines cómo debe responder la app a las acciones del usuario, de forma similar a lo que ocurre en una ventana desktop.

Una Pantalla es algo un poco más quisquilloso, porque no hay una característica real en desktop con lo que se pueda comparar. En tu app iOS, defines una pantalla (Screen) por dispositivo y

orientación. Una app iOS en Xojo tendrá dos Pantallas por omisión: una pantalla iPhone y una pantalla iPad.

Si estás creando una app sólo para iPhone es seguro borrar la pantalla de iPad, y viceversa.

Recuerda que en una app desktop seleccionas la ventana que verá por omisión el usuario. En una app iOS seleccionas la Vista por omisión para el usuario, pero has de hacerlo para cada Pantalla:



Observa que al combinar tu Pantalla y Vista con las diferentes orientaciones disponibles puedes crear una interfaz muy personalizada para tu app iOS. Por ejemplo, alguien que usa tu app en un iPad en orientación apaisada puede ver una interfaz completamente distinta sobre la que ve alguien que usa la app en un iPad con la orientación vertical. Y puedes refinar aun más la

interfaz en función de que el usuario tenga un dispositivo iOS con el botón Inicio en la parte superior, inferior, a la derecha o izquierda.

Pero cambiar de una Vista a otra en iOS no resulta tan simple como mostrar una nueva Ventana en desktop. Dado que la vista actual llena por completo la interfaz de usuario, has de gestionar qué Vista está en primer plano. Esto queda demostrado de forma más sencill mediante un proyecto de ejemplo.

1) Crea un nuevo proyecto iOS.

Dado que se trata de un proyecto rápido usado como demostración, no has de preocuparte sobre el nombre de la Aplicación y el resto de detalles relacionados. Por omisión ya existirá una Vista llamada View1.

2) Crea una nueva Vista.

Ve al menú Insert y selecciona View. La nueva Vista se llamará View2 por omisión. En el Inspector, define la propiedad `NavigationBarVisible` de View2 a `True`. Esto proporcionará un botón Back a la View1.

3) Añade un `iOSButton` a View1.

Arrastra un `iOSButton` sobre View1 y sitúalo en el centro de la Vista. Cambia su Etiqueta (Caption) a "This is View1". Introduce el siguiente código en el evento Action del `iOSButton`:

```
Dim V As New View2
PushTo(V)
```

4) **Define la propiedad BackButtonTitle de la View1.**

Cada Vista tiene una propiedad llamada “BackButtonTitle”. Esta propiedad determina la etiqueta del botón que devolverá al usuario a la Vista. Define BackButtonTitle en la Vista a “Back”. Esto activará la barra de herramientas a nivel del sistema para View2 y añadirá un botón “Volver”. Recuerda que la etiqueta del botón Back en View2 está configurado por la propiedad BackButtonTitle de la View1.

5) **Añade una UILabel a View2.**

Arrastra una UILabel en View2 y sitúala en el centro de la Vista. Cambia su propiedad Text a “This is View2”.

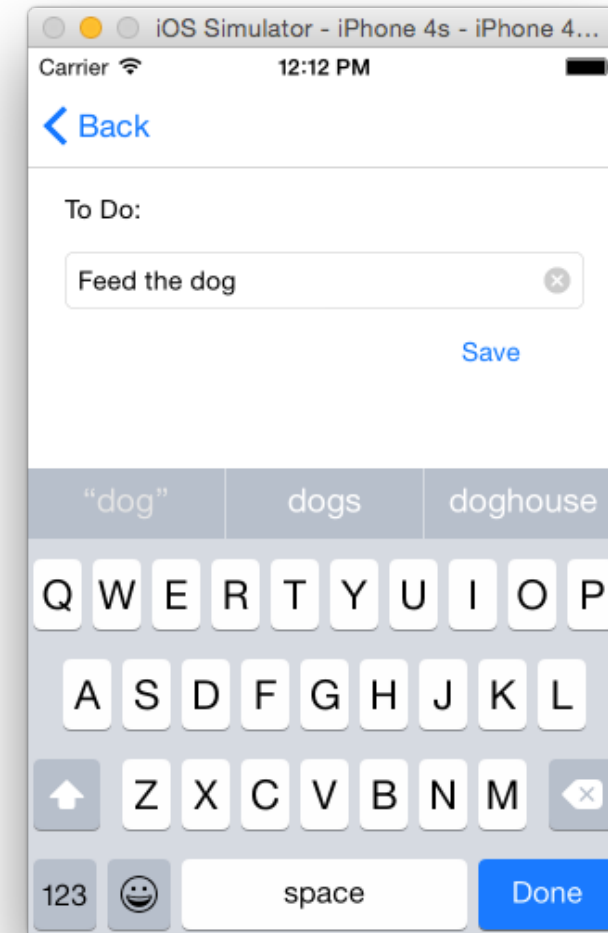
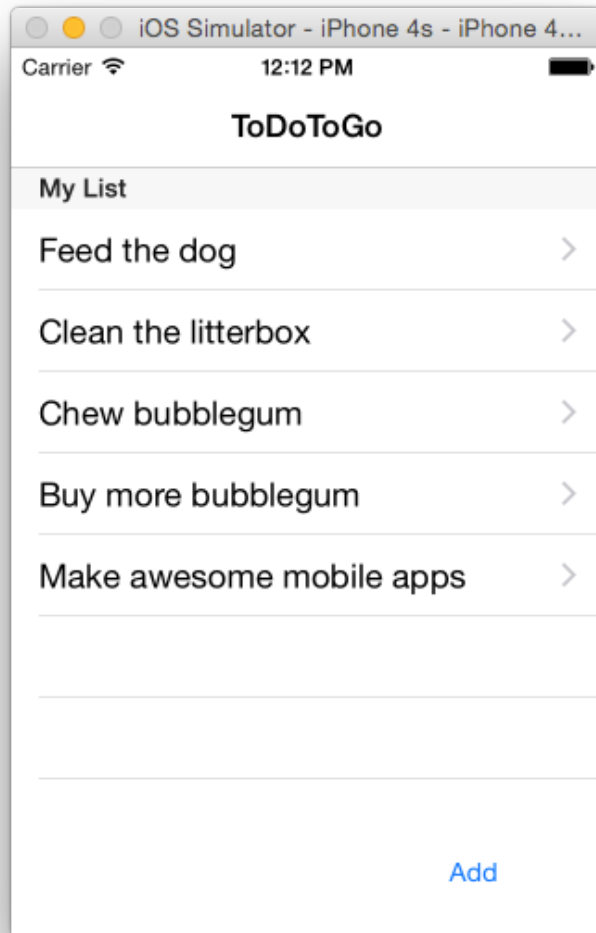
6) **Ejecuta el Proyecto.**

Experimenta con la navegación entre View1 y View2. Observa como la View2 se “desliza” sobre la View1 y luego se vuelve a “deslizar” de nuevo para desaparecer.

Gestionar estas interacciones entre Vistas es una parte crítica en el desarrollo de una app iOS genial, dado que da al usuario una sensación de “ubicación” en la interfaz. El método PushTo en particular causa que una View se “deslice” o “se ponga sobre” otra Vista, creando una sensación para el usuario de que dichas Vistas están apilada y que sus posiciones son relativas entre sí.

1.7 Manos a la obra con iOS

Tu proyecto de ejemplo para iOS es una variación del Gestor de Tareas creado en el Capítulo 5. Podrás añadir elementos a una lista de tareas, editar dichos elementos y borrarlos. La app terminada debería similar a esta:



1) Si aún no lo has hecho, ejecuta Xojo y crea una nueva aplicación iOS. Guárdala como “ToDoGo”.

Para empezar, crearás la interfaz, luego añadirás el código. En primer lugar cambia el nombre de tu Vista por defecto a “ListView”. Asegúrate de que el BackButtonTitle de ListView está definido a “Back” y que su propiedad NavigationBarVisible está configurada a True. Define el título del ListView a “ToDoGo”. En la Librería, añade dos controles al ListView: una iOSTable llamada “ToDoTable” y un botón llamado “AddButton”.

2) Añade un nuevo módulo al proyecto dirigiéndote al menú Insert y seleccionado Module.

El módulo debería nombrarse “ToDoList”. Añádele una nueva propiedad: ToDo(-1) As Text. Los paréntesis tras el nombre de la propiedad hacen que dicha propiedad sea un array (el valor -1 hace que sea un array vacío). Aquí es donde tu app guardará las tareas del usuario. Asegúrate de que el ámbito de la propiedad sea Global.

3) Añade un nuevo método al ListView.

Tu app necesitará una forma de mostrar el actual listado de elementos al usuario, lo que significa que has de añadir un método que itere el array de tareas y los ponga en la iOSTable. Añade un nuevo método a ListView llamado “PopulateTable”.

4) Añade este código al método PopulateTable:

```
Dim i As Integer
Dim cell As iOSTableCellData
ToDoTable.RemoveAll
ToDoTable.AddSection( "My List" )

For i = 0 To UBound( ToDo )
    cell = New iOSTableCellData
    cell.Text = ToDo( i )
    cell.AccessoryType = 7
        iOSTableCellData.AccessoryTypes.Disclosure
    ToDoTable.AddRow( 0, cell )
Next
```

Observa que al trabajar con iOSTable no es lo mismo que trabajar con un ListBox en una app desktop. Una diferencia importante es que la iOSTable tiene secciones, lo que se traduce en áreas discretas de la tabla, cada una de ellas con su propio título. Por ejemplo, si quisieras dividir tus tareas en categorías, podrías tener una sección en la iOSTable para cada categoría. Para mantener las cosas sencillas en este proyecto de ejemplo, el código anterior añade una sección llamada “Mi Lista” en la iOSTable.

La incorporación de datos en las celdas de una iOSTable también es muy distinto. Recuerda que en desktop sólo tenías que usar el método AddRow y pasar la string correspondiente. En iOS has de crear una nueva instancia de la clase iOSTableCellData y pasar dicha instancia al método AddRow de la iOSTable. Dicho método también toma un entero que indica a qué sección debería de pertenecer la celda. En este ejemplo, dado que sólo hay una sección, sólo tenemos que usar cero como el primer parámetro.

El tipo de dato iOSTableCellData tiene una propiedad que siempre debería de mostrar un valor: text. Esta es la información que se mostrará en la lista

y que podrá ver el usuario. El anterior método también asigna un `AccessoryType` a la `iOSTableCellData`. El `AccessoryType` proporciona una pista gráfica sobre el propósito de la celda. En este caso, el tipo `Disclosure` añade una pequeña flecha al final de la celda para indicar que el usuario puede tocar la celda para disparar una nueva interacción.

Por último, observa que antes de que rellenes la tabla con datos has de vaciarla de cualquier valor ya existente, de modo que se evite presentar al usuario múltiples copias del mismo dato.

5) **Añade al ListView un Event Handler para el evento Activate.**

Cuando el usuario vea la `ListView`, los datos habrán sido refrescados y actualizados, de modo que el manejador de evento `Activate` debería de incluir esta línea de código:

```
PopulateTable
```

6) **Añade una nueva Vista llamada “ToDoView.”**

Esta es la Vista donde el usuario introducirá y editará los elementos. Asegúrate de que su propiedad `NavigationBarVisible` está definida a `True`. Necesitará cuatro controles que puedes situar donde desees (usa la captura de pantalla como guía): Una etiqueta, un `TextField` llamado “`ToDoField`”, un botón llamado “`SaveButton`”, y un botón llamado “`DeleteButton`.” Las etiquetas de `SaveButton` y `DeleteButton` deberían de ser “`Save`” y “`Delete`”, respectivamente. Define la propiedad `Visible` del `DeleteButton` a `False`; pues sólo será necesario cuando el usuario esté editando un ítem, de modo que tu app sólo debería de mostrarlo cuando fuese necesario. Por último, añade una nueva propiedad a `ToDoView`: `ToDoItem` as `Text`.

7) **Añade un nuevo método a ToDoView llamado “Populate.”**

El método debería tomar un parámetro: `Item` as `Text`. El siguiente código debería tomar dicho valor y asignarlo a la propiedad `ToDoItem` de la Vista.

Cuando el usuario edite una tarea tu app tendrá que mostrar el texto de dicho ítem en el `ToDoField`. El método `Populate` se encargará de ello. Este es el código para dicho método:

```
ToDoItem = Item
ToDoField.Text = ToDoItem
DeleteButton.Visible = True
```

Este código define la propiedad `ToDoItem` de la `ToDoView` con el texto del ítem que el usuario quiere editar. También pone el valor en la propiedad `text` del `ToDoField`. Por último, hace que `DeleteButton` sea visible para que el usuario pueda eliminar la tarea de la lista.

8) **Añade el siguiente código al evento Action del AddButton en el ListView.**

Añade el manejador de evento `Action` al `AddButton` en `ListView`, y añade el siguiente código:

```
Dim V As New ToDoView
PushTo( V )
```

Este código debería de resultarte familiar del ejemplo de navegación que vimos al comienzo del capítulo. Simplemente muestra la `ToDoView` en la pantalla para permitir que el usuario cree una nueva tarea.

9) Añade el siguiente código al evento Action de ToDoTable en el ListView.

Añade el manejador de evento Action al AddButton en la ListView, y añade este código:

```
Dim item As Text
Dim V As New ToDoView
item = Me.RowData( section, row ).Text
V.Populate( item )
PushTo( V )
```

Este código se ejecuta cuando el usuario toca en una celda de la ToDoTable. Determina sobre qué tarea ha tocado el usuario y la pasa al método Populate del ToDoView. Observa que en vez de acceder simplemente al valor de texto para la fila seleccionada, tal y como harías en desktop, debes de acceder a este a través de la propiedad RowData.

10) Añade el siguiente código al evento Action de SaveButton en ToDoView.

Añade el manejador de evento Action al SaveButton en ToDoView, y añade este código:

```
If ToDoField.Text <> "" Then
    If ToDoItem <> "" Then
        Dim i As Integer
        For i = 0 To UBound( ToDo )
            If ToDo( i ) = ToDoItem Then
                ToDo( i ) = ToDoField.Text
            End If
        Next
    Else
        ToDo.Append( ToDoField.Text )
    End If
End If
Self.Close
```


Cuando el usuario toca SaveButton, tu app necesitará crear una nueva tarea o bien actualizar la existente. Para determinar la acción a realizar, este código comprueba si la propiedad ToDoItem del ToDoView contiene ya texto. Si no es así, entonces se puede asumir que el usuario está creando una nueva tarea. Esto se hace añadiendo la propiedad de texto de ToDofield al array ToDo.

Sin embargo, en el caso de que ToDoItem en el ToDoView ya contenga texto, entonces ha de actualizarse el ítem ya existente. Para ello, este código itera el array global ToDo buscando un elemento que coincida con el que se ha proporcionado a ToDoView mediante el método Populate. Si encuentra la correspondencia, actualiza dicho elemento en el array.

Cuando una tarea está completada, el método cierra la ventana y devuelve al usuario a la ListView.

11) Añade el siguiente código en el evento Action de DeleteButton en ToDoView.

Añade un manejador de evento Action al DeleteButton en ToDoView, y añade el siguiente código:

```
If ToDoItem <> "" Then
    Dim i As Integer
    For i = 0 To UBound( ToDo )
        If ToDo( i ) = ToDoItem Then
            ToDo.Remove( i )
        End If
    Next
End If
Self.Close
```

Cuando el usuario toque DeleteButton, la app iterará el array global ToDo, buscando un elemento que se corresponda con el proporcionado a

ToDoView mediante el método Populate. Si se encuentra una coincidencia, se borra del array y se devuelve al usuario al ListView.

12) Ejecuta el proyecto.

Debería de aparecer el Simulador iOS y se debería de lanzar tu app iOS. Prueba a añadir, editar y borrar algunas tareas.

Si tu proyecto no funciona, asegúrate de que has seleccionado un Dispositivo de Simulador en la sección iOS Build Settings de tu proyecto, y de que Xcode esté instalado correctamente.

1.8 Depuración

Incluso aunque pueda parecer que tu proyecto iOS esté botando entre varias aplicaciones antes de que lo veas en funcionamiento, la depuración es prácticamente igual que depurar una app desktop. Aun puedes definir puntos de parada y ejecutar el código paso a paso.

Para más información sobre depuración, consulta las secciones 1.5 y 2.6 en *Introducción a la Programación con Xojo*.