

# El entorno Xcode

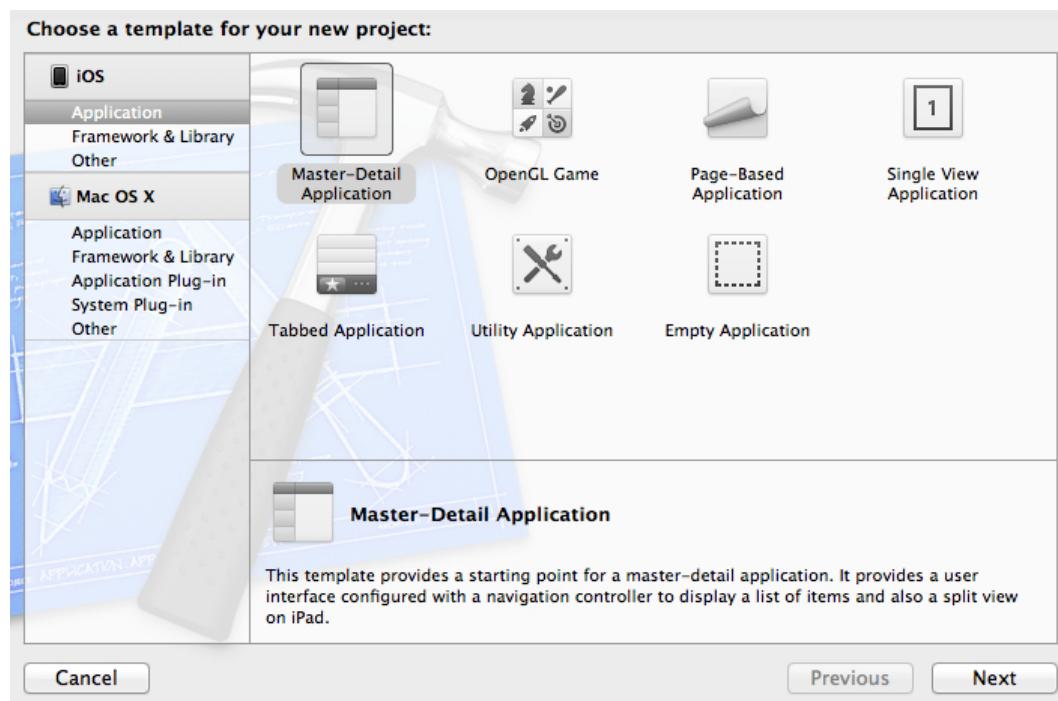
## Índice

1 Estructura del proyecto.....	6
2 Propiedades del proyecto.....	9
3 Configuraciones.....	14
4 Localización.....	15
5 Esquemas y acciones.....	17
6 Recursos y grupos.....	18
7 Interface Builder.....	20
8 Organizer.....	21
9 Repositorios SCM.....	23
9.1 Repositorios Git.....	23
9.2 Repositorios SVN.....	26
10 Snapshots.....	30
11 Ejecución y firma.....	31
11.1 Creación del par de claves.....	31
11.2 Perfil de provisionamiento.....	35

Para desarrollar aplicaciones iOS (iPod touch, iPhone, iPad) deberemos trabajar con una serie de tecnologías y herramientas determinadas. Por un lado, deberemos trabajar con el IDE Xcode dentro de una máquina con el sistema operativo MacOS instalado, cosa que únicamente podremos hacer en ordenadores Mac. Por otro lado, el lenguaje que deberemos utilizar es Objective-C, una extensión orientada a objetos del lenguaje C, pero muy diferente a C++. Además, deberemos utilizar la librería Cocoa Touch para crear una interfaz para las aplicaciones que pueda ser visualizada en un dispositivo iOS. Vamos a dedicar las primeras sesiones del módulo a estudiar fundamentalmente el entorno Xcode y el lenguaje Objective-C, pero introduciremos también algunos elementos básicos de Cocoa Touch para así poder practicar desde un primer momento con aplicaciones iOS.

Comenzaremos viendo cómo empezar a trabajar con el entorno Xcode. Vamos a centrarnos en la versión 4.2 de dicho IDE, ya que es la que se está utilizando actualmente y cuenta con grandes diferencias respecto a sus predecesoras.

Al abrir el entorno, lo primero que deberemos hacer es crear un nuevo proyecto con *File > New > New Project....*. En el asistente para la creación del proyecto nos dará a elegir una serie de plantillas de las que partir. En el lateral izquierdo vemos una serie de categorías de posibles plantillas, y en la parte central vemos las plantillas de la categoría seleccionada. Vemos además que las categorías se dividen en iOS y Mac OS X. Como es evidente, nos interesarán aquellas de la sección iOS.



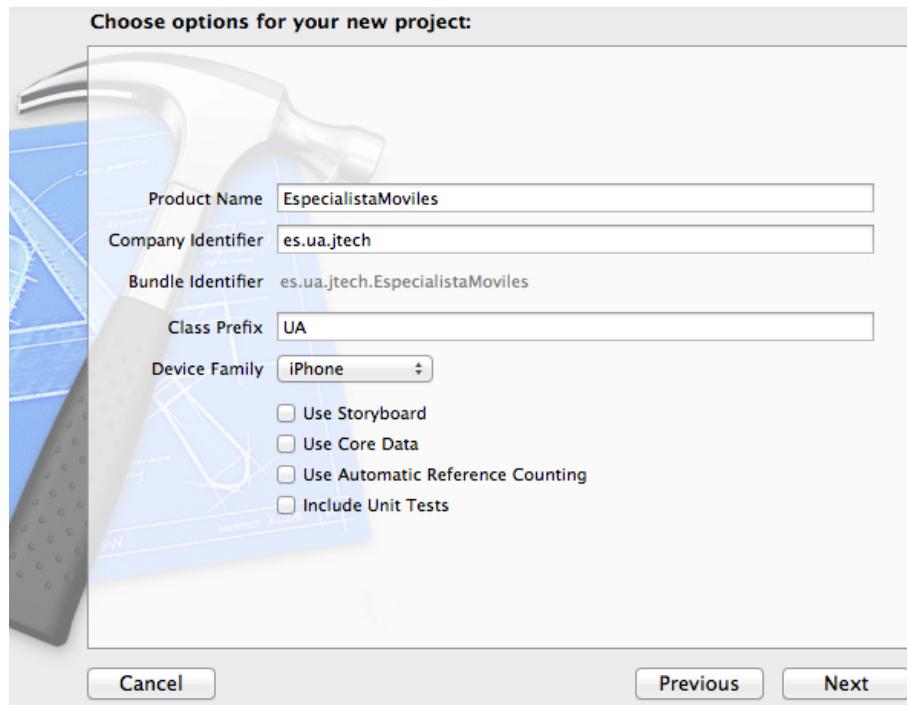
Tipos de proyectos

Las categorías que encontramos son:

- *Application*: Estos son los elementos que nos interesarán. Con ellos podemos crear diferentes plantillas de aplicaciones iOS. Encontramos plantillas para los estilos de navegación más comunes en aplicaciones iOS, que estudiaremos con mayor detenimiento más adelante, y también alguna plantilla básica como *Empty Application*, que no crea más componentes que una ventana vacía. Deberemos elegir aquella plantilla que más se ajuste al tipo de aplicación que vayamos a realizar. La más común en aplicaciones iOS es *Master-Detail Application* (en versiones anteriores de Xcode el tipo equivalente se llamaba *Navigation-based Application*), por lo que será la que utilicemos durante las primeras sesiones.
- *Framework & Library*: Nos permitirá crear librerías o *frameworks* que podamos utilizar en nuestras aplicaciones iOS. No se podrán ejecutar directamente como una aplicación, pero pueden ser introducidos en diferentes aplicaciones.
- *Other*: Aquí encontramos la opción de crear un nuevo proyecto vacío, sin utilizar ninguna plantilla.

Tras seleccionar un tipo de proyecto, nos pedirá el nombre del producto y el identificador de la compañía. El nombre del producto será el nombre que queramos darle al proyecto y a nuestra aplicación, aunque más adelante veremos que podemos modificar el nombre que se muestra al usuario. El identificador de la compañía se compone de una serie de cadenas en minúscula separadas por puntos que nos identificarán como desarrolladores. Normalmente pondremos aquí algo similar a nuestra URL escrita en orden inverso (como se hace con los nombres de paquetes en Java). Por ejemplo, si nuestra web es `jtech.ua.es`, pondremos como identificador de la compañía `es.ua.jtech`. Con el nombre del producto junto al identificador de la compañía compondrá el identificador del paquete (*bundle*) de nuestra aplicación. Por ejemplo, una aplicación nuestra con nombre `EspecialistaMoviles`, tendría el identificador `es.ua.jtech.EspecialistaMoviles`.

Podremos también especificar un prefijo para las clases de nuestra aplicación. En Objective-C no existen los paquetes como en lenguaje Java, por lo que como espacio de nombres para las clases es habitual utilizar un determinado prefijo. Por ejemplo, si estamos desarrollando una aplicación de la Universidad de Alicante, podríamos especificar como prefijo `UA`, de forma que todas las clases de nuestra aplicación llevarán ese prefijo en su nombre y nos permitirá distinguirlas de clases de librerías que estemos utilizando, y que podrían llamarse de la misma forma que las nuestras si no llevasen prefijo.



Datos del proyecto

Bajo los campos anteriores, aparece un desplegable que nos permite elegir a qué tipo de dispositivos vamos a destinar la aplicación: iPhone, iPad, o Universal. Las aplicaciones de iPhone se pueden ver en el iPad emuladas, pero en un recuadro del tamaño del iPhone. Las de iPad sólo son compatibles con el iPad, mientras que las aplicaciones Universales son compatibles con ambos dispositivos y además su interfaz de adapta a cada uno de ellos. Por ahora comenzaremos realizando aplicaciones únicamente para iPhone, y más adelante veremos cómo desarrollar aplicaciones para iPad y aplicaciones Universales.

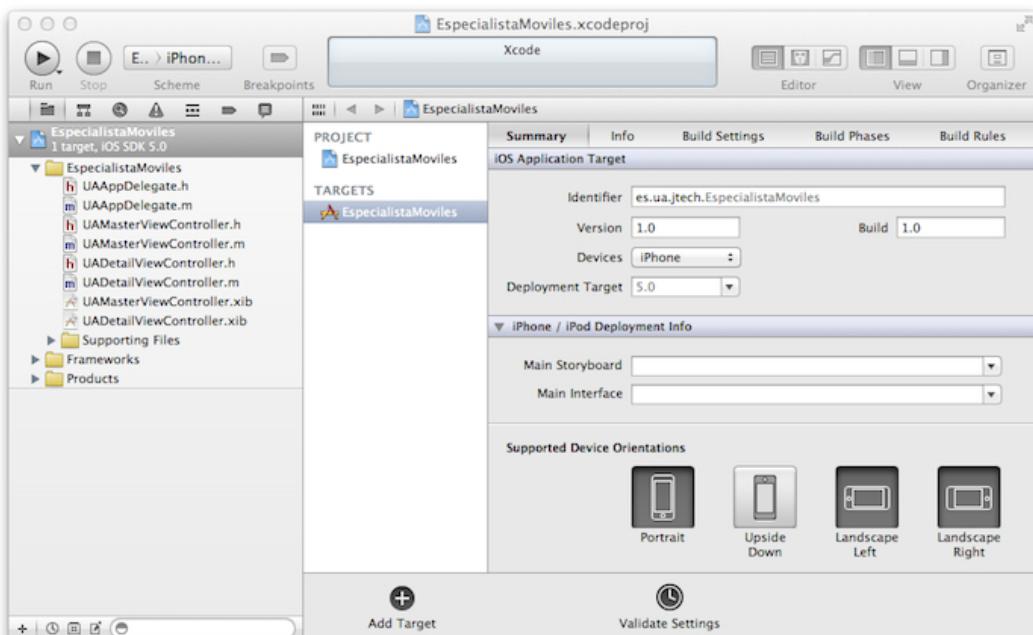
También tendremos cuatro *checkboxes* que para añadir *Storyboards*, *Core Data*, *Automatic Reference Counting* (ARC) y pruebas de unidad a nuestra aplicación. Por ahora vamos a desmarcarlos. Más adelante estudiaremos dichos elementos.

Tras llenar estos datos, pulsaremos *Next* y para finalizar tendremos que seleccionar la ubicación del sistema de archivos donde guardar el proyecto. Se creará un directorio con el nombre de nuestro proyecto en la localización que indiquemos, y dicho directorio contendrá todos los componentes del proyecto.

#### Nota

En la ventana de selección de la ubicación en la que guardar el proyecto nos dará también la opción de crear automáticamente un repositorio SCM local de tipo Git para dicho proyecto. Por el momento vamos a utilizar dicho repositorio, ya que lo único que tendremos que hacer es dejar marcada la casilla y Xcode se encargará de todo lo demás. Más adelante veremos como subir el proyecto a un repositorio remoto propio.

## El entorno Xcode



Aspecto del entorno Xcode

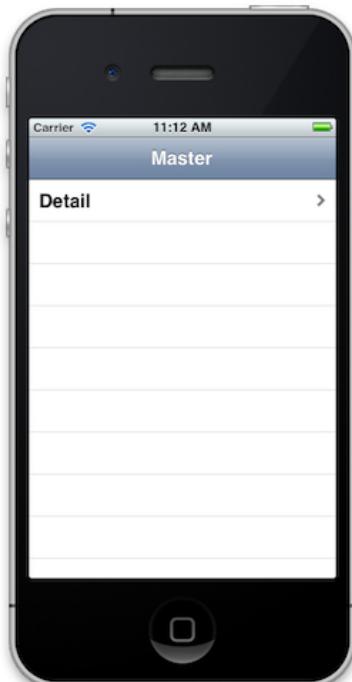
Una vez creado el proyecto, veremos la ventana principal del entorno Xcode. Podemos distinguir las siguientes secciones:

- **Navegador:** A la derecha vemos los artefactos de los que se compone el proyecto, organizados en una serie de grupos. Esta sección se compone de varias pestañas, que nos permitirán navegar por otros tipos de elementos, como por ejemplo por los resultados de una búsqueda, o por los resultados de la construcción del proyecto (errores y warnings).
- **Editor:** En la parte central de la pantalla vemos el editor en el que podremos trabajar con el fichero que esté actualmente seleccionado en el navegador. Dependiendo del tipo de fichero, editaremos código fuente, un fichero de propiedades, o la interfaz de la aplicación de forma visual.
- **Botones de construcción:** En la parte superior izquierda vemos una serie de botones que nos servirán para construir y ejecutar la aplicación.
- **Estado:** En la parte central de la parte superior veremos la información de estado del entorno, donde se mostrarán las acciones que se están realizando en un momento dado (como compilar o ejecutar el proyecto).
- **Opciones de la interfaz:** En la parte derecha de la barra superior tenemos una serie de botones que nos permitirán alterar el aspecto de la interfaz, mostrando u ocultando algunos elementos, como puede ser la vista de navegación (a la izquierda), la consola de depuración y búsqueda (abajo), y la barra de utilidades y ayuda rápida (derecha). También podemos abrir una segunda ventana de Xcode conocida como *Organizer*, que nos permitirá, entre otras cosas, gestionar los dispositivos con los que contamos o navegar por la documentación. Más adelante veremos dicha ventana con más detalle.

**Recomendación**

Resulta bastante útil mantener abierta la vista de ayuda rápida a la derecha de la pantalla, ya que conforme editamos el código aquí nos aparecerá la documentación de los elementos de la API sobre los que se encuentre el cursor.

En este momento podemos probar a ejecutar nuestro proyecto. Simplemente tendremos que pulsar el botón de ejecutar (en la parte superior izquierda), asegurándonos que junto a él en el cuadro desplegable tenemos seleccionado que se ejecute en el simulador del iPhone. Al pulsar el botón de ejecución, se mostrará el progreso de la construcción del proyecto en la parte superior central, y una vez finalizada se abrirá una ventana con un emulador de iPhone y se ejecutará nuestra aplicación en él, que actualmente será únicamente una pantalla con una barra de navegación y una lista vacía. En la barra de menús del simulador podemos encontrar opciones para simular diferentes condiciones del hardware, como por ejemplo los giros del dispositivo.



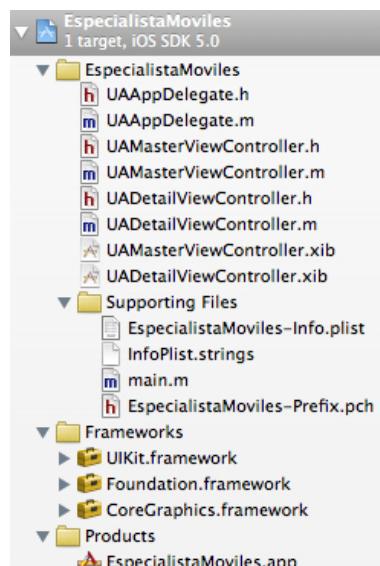
Simulador de iPhone

## 1. Estructura del proyecto

En la zona del navegador podemos ver los diferentes elementos que componen nuestro proyecto. Los tipos de ficheros que encontraremos normalmente en un proyecto Objective-C son los siguientes:

- .m: El código fuente Objective-C se guarda en ficheros con extensión .m, por lo que ese será el formato que encontraremos normalmente, aunque también se podrá utilizar código C (.c) estándar, C++ (.cpp, .cc), u Objective-C combinado con C++ (.mm).
- .h: También tenemos los ficheros de cabeceras (.h). Las clases de Objective-C se componen de un fichero .m y un fichero .h con el mismo nombre.
- .xib: También encontramos .xib (también conocidos como nib, ya que este es su formato una vez compilados) que son los ficheros que contienen el código de la interfaz, y que en el entorno se editarán de forma visual.
- .plist: Formato de fichero de propiedades que estudiaremos más adelante con mayor detenimiento, y que se utiliza para almacenar una lista de propiedades. Resultan muy útiles para guardar los datos o la configuración de nuestra aplicación, ya que resultan muy fáciles de editar desde el entorno, y muy fáciles de leer y de escribir desde el programa.
- .strings: El formato .strings define un fichero en el que se definen una serie de cadenas de texto, cada una de ellas asociada a un identificador, de forma que creando varias versiones del mismo fichero .strings tendremos la aplicación localizada a diferentes idiomas.
- .app: Es el resultado de construir y empaquetar la aplicación. Este fichero es el que deberemos subir a iTunes para su publicación en la App Store.

Estos diferentes tipos de artefactos se pueden introducir en nuestro proyecto organizados por grupos. Es importante destacar que los grupos no corresponden a ninguna organización en el disco, ni tienen ninguna repercusión en la construcción del proyecto. Simplemente son una forma de tener los artefactos de nuestro proyecto organizados en el entorno, independientemente del lugar del disco donde residan realmente. Los grupos se muestran en el navegador con el icono de una carpeta amarilla.

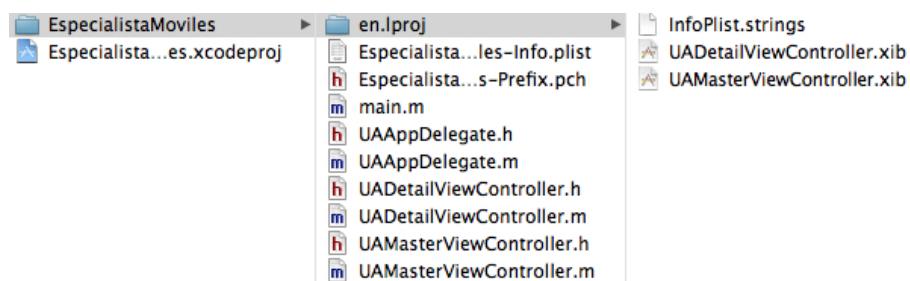


Configuración del proyecto creado

En la plantilla que nos ha creado encontramos los siguientes grupos de artefactos:

- **Fuentes** (están en un grupo con el mismo nombre del proyecto, en nuestro caso `EspecialistaMoviles`). Aquí se guardan todos los ficheros de código fuente que componen la aplicación (código fuente `.m`, cabeceras `.h` y ficheros de la interfaz `.xib`).
- **Soporte** (dentro del grupo de fuentes, en un subgrupo *Supporting files*). Aquí encontramos algunos recursos adicionales, como el fichero `main.m`, que es el punto de arranque de la aplicación, y que nosotros normalmente no deberemos tocar, el fichero `Proyecto-Prefix.pch`, que define un prefijo con una serie de *imports* que se aplicarán a todos los ficheros del proyecto (normalmente para importar la API de Cocoa Touch), `Proyecto-Info.plist`, que define una serie de propiedades de la aplicación (nombre, identificador, versión, ícono, tipos de dispositivos soportados, etc), y por último `InfoPlist.strings`, donde se externalizan las cadenas de texto que se necesitan utilizar en el fichero `.plist` anterior, para así poder localizar la aplicación fácilmente.
- **Frameworks**: Aquí se muestran los *frameworks* que la aplicación está utilizando actualmente. Por defecto nos habrá incluido `UIKit`, con la API de la interfaz de Cocoa Touch, `Foundation`, con la API básica de propósito general (cadenas, *arrays*, fechas, URLs, etc), y `CoreGraphics`, con la API básica de gráficos (fuentes, imágenes, geometría).
- **Products**: Aquí vemos los resultados de construir la aplicación. Veremos un fichero `EspecialistaMoviles.app` en rojo, ya que dicho fichero todavía no existe, es el producto que se generará cuando se construya la aplicación. Este fichero es el paquete (*bundle*) que contendrá la aplicación compilada y todos los recursos y ficheros de configuración necesarios para su ejecución. Esto es lo que deberemos enviar a Apple para su publicación en la App Store una vez finalizada la aplicación.

Sin embargo, si nos fijamos en la estructura de ficheros que hay almacenada realmente en el disco podemos ver que dichos grupos no existen:



Estructura de ficheros del proyecto

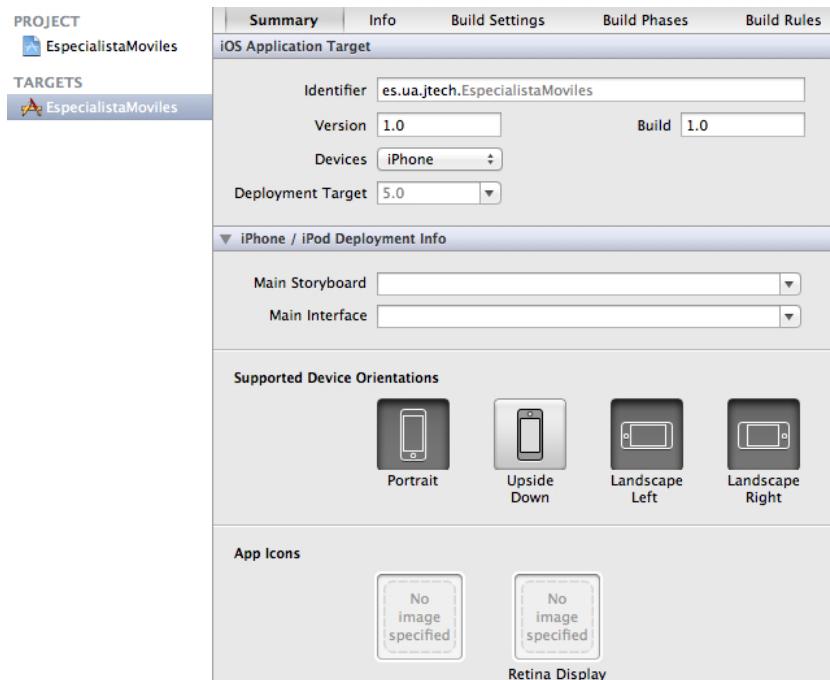
Vemos que tenemos en el directorio principal un fichero con extensión `xcodeproj`. Este es el fichero con la configuración de nuestro proyecto, y es el que deberemos seleccionar para abrir nuestro proyecto la próxima vez que vayamos a trabajar con él (haciendo doble-click sobre él se abrirá Xcode con nuestro proyecto). A parte de este fichero, tenemos un directorio con el nombre del proyecto que contiene todos los ficheros al

mismo nivel. La única excepción son los ficheros que hay en un directorio de nombre `en.lproj`. Esto se debe a que dichos ficheros están localizados, concretamente a idioma inglés como indica su directorio (`en`). Tendremos un subdirectorio como este por cada idioma que soporte nuestra aplicación (`es.lproj`, `ca.lproj`, etc). Es la forma en la que se trata la localización en las aplicaciones iOS, y lo veremos con más detalle a continuación.

## 2. Propiedades del proyecto

Si seleccionamos en el navegador el nodo raíz de nuestro proyecto, en el editor veremos una ficha con sus propiedades. Podemos distinguir dos apartados: *Project* y *Targets*. En *Project* tenemos la configuración general de nuestro proyecto, mientras que *Target* hace referencia a la construcción de un producto concreto. En nuestro caso sólo tenemos un *target*, que es la construcción de nuestra aplicación, pero podría haber varios. Por ejemplo, podríamos tener un target que construya nuestra aplicación para iPhone y otro que lo haga para iPad, empaquetando en cada una de ellas distintos recursos y compilando con diferentes propiedades. También tendremos dos *targets* si hemos creado pruebas de unidad, ya que tendremos un producto en el que se incluirán las pruebas, y otro en el que sólo se incluirá el código de producción que será el que publiquemos.

Si pinchamos sobre el *target*, en la pantalla principal veremos una serie de propiedades de nuestra aplicación que resultan bastante intuitivas, como la versión, dispositivos a los que está destinada, y versión mínima de iOS que necesita para funcionar. También podemos añadir aquí tanto los iconos como las imágenes para la pantalla de carga de nuestra aplicación, simplemente arrastrando dichas imágenes sobre los huecos correspondientes.



Configuración básica del target

Lo que no resulta tan claro es por ejemplo la diferencia entre versión y *build*. La primera de ellas indica el número de una *release* pública, y es lo que verá el usuario en la App Store, con el formato de tres enteros separados por puntos (por ejemplo, 1.2.1). Sin embargo, la segunda se utiliza para cada *build* interna. No es necesario seguir ningún formato dado, y puede coincidir o no con el número de versión. En muchas ocasiones se utiliza un número entero de *build* que vamos incrementando continuamente con cada iteración del proyecto, por ejemplo 1524. Otra diferencia existente entre ambos números es que podemos tener un número diferente de versión para distintos *targets*, pero la *build* siempre será la misma para todos ellos.

También vemos unos campos llamados *Main Storyboard* y *Main Interface*. Aquí se puede definir el punto de entrada de nuestra aplicación de forma declarativa, pudiendo especificar qué interfaz (fichero `xib`) o que *storyboard* se va a mostrar al arrancar la aplicación. En la plantilla crear ninguno de estos campos tiene valor, ya que la interfaz se crea de forma programática. En realidad el punto de entrada está en `main.m`, y podríamos modificarlo para tener mayor control sobre lo que la aplicación carga al arrancar, pero normalmente lo dejaremos tal como se crea por defecto. Más adelante veremos más detalles sobre el proceso de carga de la aplicación.

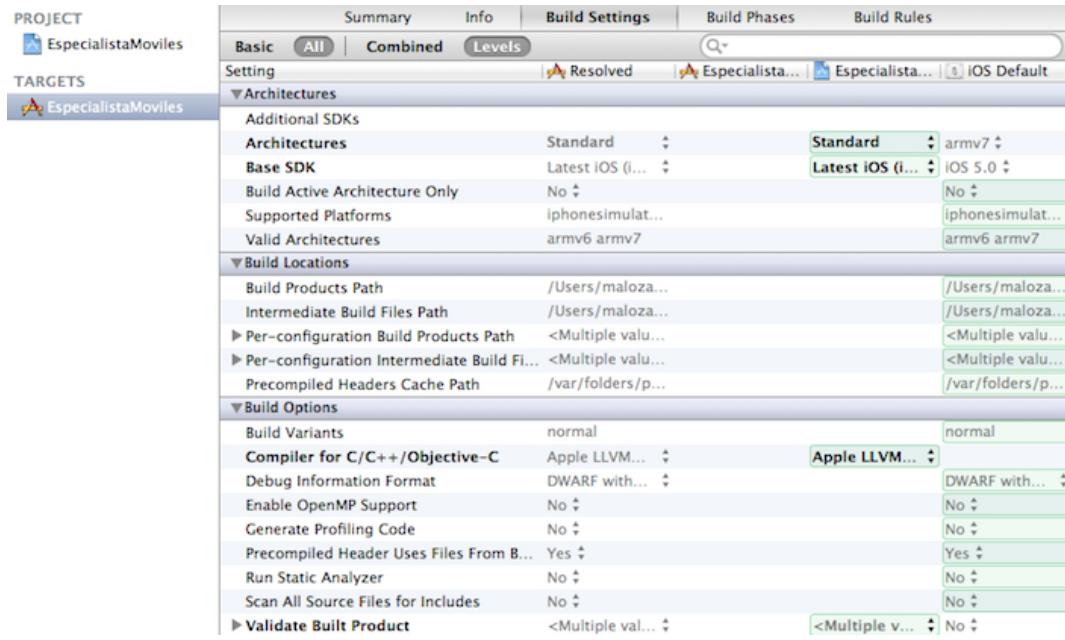
En la pestaña *Info* del *target* vemos los elementos de configuración anteriores, y algunos más, pero en forma de lista de propiedades. Realmente esta información es la que se almacena en el fichero `Info.plist`, si seleccionamos dicho fichero veremos los mismos datos también presentados como lista de propiedades. Por ejemplo la propiedad *Bundle display name* no estaba en la pantalla anterior, y nos permite especificar el nombre que

aparecerá bajo el ícono de nuestra aplicación en el iPhone. Por defecto tenemos el nombre del proyecto, pero podemos modificarlo por ejemplo para acortarlo y así evitar que el iPhone recorte las letras que no quepan. Si pinchamos con el botón derecho sobre el fichero `Info.plist` y seleccionamos *Open As > Source Code*, veremos el contenido de este fichero en el formato XML en el que se definen los ficheros de listas de propiedades, y los nombres internos que tiene cada una de las propiedades de configuración.

PROJECT	Summary	Info	Build Settings	Build Phases	Build Rules
EspecialistaMoviles					
TARGETS					
EspecialistaMoviles					
	Custom iOS Target Properties				
	Key	Type	Value		
	Bundle name	String	<code>\$(PRODUCT_NAME)</code>		
	Bundle identifier	String	<code>es.ua.jtech.\$(PRODUCT_NAME):rfc1034identifier</code>		
	InfoDictionary version	String	6.0		
	► Required device capabilities	Array	(1 item)		
	Bundle version	String	1.0		
	Executable file	String	<code>\$(EXECUTABLE_NAME)</code>		
	Application requires iPhone environment	Boolean	YES		
	► Icon files	Array	(0 items)		
	► Supported interface orientations	Array	(3 items)		
	Bundle display name	String	<code>\$(PRODUCT_NAME)</code>		
	Bundle OS Type code	String	APPL		
	Bundle creator OS Type code	String	????		
	Localization native development region	String	en		
	Bundle versions string, short	String	1.0		
	► Document Types (0)				
	► Exported UTIs (0)				
	► Imported UTIs (0)				
	► URL Types (0)				

Configuración completa del target

A continuación tenemos la pestaña *Build Settings*. Aquí se define una completa lista con todos los parámetros utilizados para la construcción del producto. Estos parámetros se pueden definir a diferentes niveles: valores por defecto para todos los proyectos iOS, valores generales para nuestro proyecto, y valores concretos para el *target* seleccionado. Se pueden mostrar los valores especificados en los diferentes niveles, y el valor que se utilizará (se coge el valor más a la izquierda que se haya introducido):



Parámetros de construcción del target

Hay un gran número de parámetros, por lo que es complicado saber lo que hace exactamente cada uno, y la mayor parte de ellos no será necesario que los modifiquemos en ningún momento. Normalmente ya se les habrá asignado por defecto un valor adecuado, por lo que aquí la regla de oro es **si no sabes lo que hace, ¡no lo toques!**. En esta interfaz podremos cambiar los valores para el *target*. Si queremos cambiar los valores a nivel del proyecto, deberemos hacerlo seleccionando el proyecto en el lateral izquierdo del editor y accediendo a esta misma pestaña.

La información más importante que podemos encontrar aquí es:

- *Base SDK*: Versión del SDK que se utiliza para compilar. Se diferencia de *Deployment Target* en que *Base SDK* indica la versión del SDK con la que se compila el proyecto, por lo que no podremos utilizar ninguna característica que se hubiese incluido en versiones posteriores a la seleccionada, mientras que *Deployment Target* indica la versión mínima para que nuestra aplicación funcione, independientemente de la versión con la que se haya compilado.
- *Code Signing*: Aquí podemos especificar el certificado con el que se firma la aplicación para probarla en dispositivos reales o para distribuirla. Más adelante veremos con más detalle la forma de obtener dicho certificado. Hasta que no lo tengamos, no podremos probar las aplicaciones más que en el emulador.

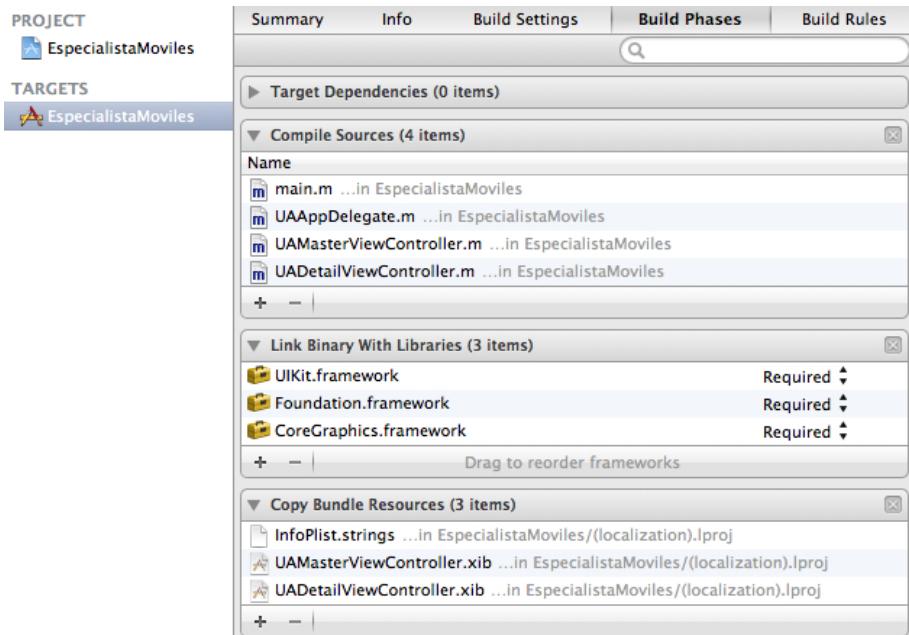
### Advertencia

Si seleccionamos un *Deployment Target* menor que el *Base SDK*, deberemos llevar mucho cuidado de no utilizar ninguna característica que se haya incluido en versiones posteriores al *Deployment Target*. Si no llevásemos cuidado con esto, un comprador con una versión antigua de iOS podría adquirir nuestra aplicación pero no le funcionaría. Lo que si que podemos hacer es

detectar en tiempo de ejecución si una determinada característica está disponible, y sólo utilizarla en ese caso.

Vemos también que para algunas opciones tenemos dos valores, uno para *Debug* y otro para *Release*. Estas son las configuraciones del proyecto, que veremos más adelante. Por ejemplo, podemos observar que para *Debug* se añade una macro `DEBUG` y se elimina la optimización del compilador para agilizar el proceso de construcción y hacer el código más fácilmente depurable, mientras que para *Release* se eliminan los símbolos de depuración para reducir el tamaño del binario.

La siguiente pestaña (*Build Phases*) es la que define realmente cómo se construye el *target* seleccionado. En ella figuran las diferentes fases de construcción del proyecto, y para cada una de ellas indica los ficheros que se utilizarán en ella:



Fases de construcción del target

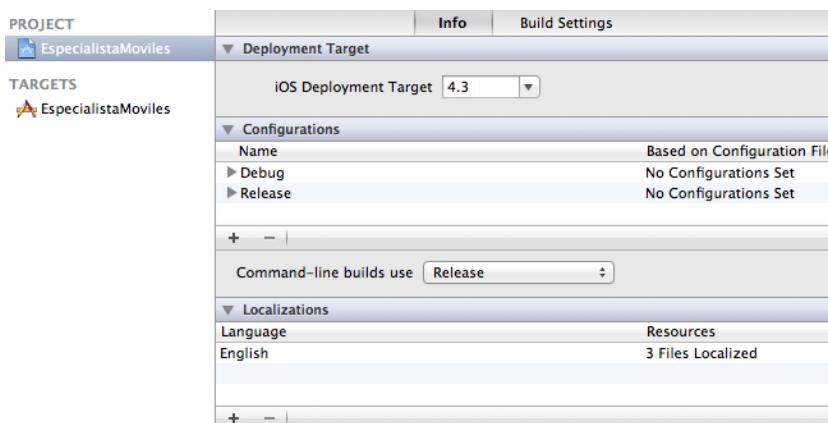
Podemos ver aquí los ficheros que se van a compilar, las librerías con las que se va a enlazar, y los recursos que se van a copiar dentro del *bundle*. Por defecto, cuando añadamos elementos al proyecto se incluirán en la fase adecuada (cuando sea código se añadirá a la compilación, los *frameworks* a *linkado*, y el resto de recursos a copia). Sin embargo, en algunas ocasiones puede que algún fichero no sea añadido al lugar correcto, o que tengamos varios *targets* y nos interese controlar qué recursos se incluyen en cada uno de ellos. En esos casos tendremos que editar a mano esta configuración. Es importante destacar que el contenido y estructura final del paquete (*bundle*) generado se define en esta pantalla, y es totalmente independiente a la organización del proyecto que vemos en el navegador.

Por último, en *Build Rules*, se definen las reglas para la construcción de cada tipo de recurso. Por ejemplo, aquí se indica que las imágenes o ficheros de propiedades simplemente se copiarán, mientras que los ficheros .xib deberán compilarse con el compilador de Interface Builder. Normalmente no tendremos que modificar esta configuración.

### 3. Configuraciones

Hemos visto que en la pantalla *Build Settings* en algunos de los parámetros para la construcción de la aplicación teníamos dos valores: *Debug* y *Release*. Éstas son las denominadas configuraciones para la construcción de la aplicación. Por defecto se crean estas dos, pero podemos añadir configuraciones adicionales si lo consideramos oportuno (normalmente siempre se añade una tercera configuración para la distribución de la aplicación, que veremos más adelante).

Podemos gestionar las configuraciones en la pestaña *Info* de las propiedades generales del proyecto:



Configuración general del proyecto

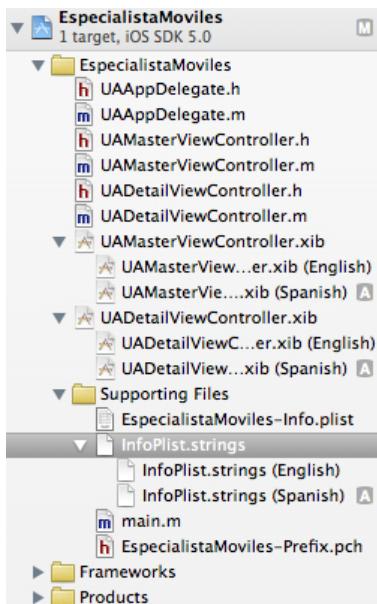
Podemos ver en la sección *Configurations* las dos configuraciones creadas por defecto. Bajo la lista, encontramos los botones (+) y (-) que nos permitirán añadir y eliminar configuraciones. Cuando añadimos una nueva configuración lo hacemos duplicando una de las existentes. Una práctica habitual es duplicar la configuración *Release* y crear una nueva configuración *Distribution* en la que cambiamos el certificado con el que se firma la aplicación.

Es importante diferenciar entre *targets* y *configuraciones*. Los *targets* nos permiten construir productos distintos, con diferentes nombre, propiedades, y conjuntos de fuentes y de recursos. Las configuraciones son diferentes formas de construir un mismo producto, cambiando las opciones del compilador, o utilizando diferentes firmas.

## 4. Localización

En la pantalla de propiedades generales del proyecto también vemos la posibilidad de gestionar las localizaciones de nuestra aplicación, es decir, los distintos idiomas en los que estará disponible. Por defecto sólo está en inglés, aunque con los botones (+) y (-) que hay bajo la lista podemos añadir fácilmente todas aquellas que queramos. Por ejemplo, podemos localizar también nuestra aplicación al español.

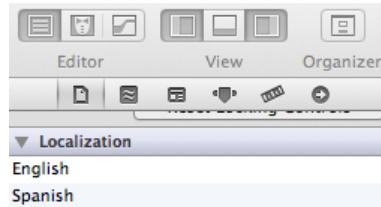
Por defecto localiza automáticamente los ficheros `xib` de la interfaz y los ficheros `strings` en los que se externalizan las cadenas de la aplicación. En el navegador veremos dos versiones de estos ficheros:



Ficheros localizados

En el disco veremos que realmente lo que se ha hecho es guardar dos copias de estos ficheros en los directorios `en.lproj` y `es.lproj`, indicando así la localización a la que pertenece cada uno.

Es posible, por ejemplo, que una interfaz no necesite ser localizada, bien por que no tiene textos, o porque los textos se van a poner desde el código. En tal caso, no es buena idea tener el fichero de interfaz duplicado, ya que cada cambio que hagamos en ella implicará modificar las dos copias. La gestión de la localización para cada fichero independiente se puede hacer desde el panel de utilidades (el panel de la derecha, que deberemos abrir en caso de que no esté abierto), seleccionando en el navegador el fichero para el que queremos modificar la localización:



Panel de utilidades de localización

De la misma forma, podemos también añadir localizaciones para ficheros que no estuviesen localizados. Por ejemplo, si tenemos una imagen en la que aparece algún texto, desde el panel de utilidades anterior podremos añadir una nueva localización y así tener una versión de la imagen en cada idioma.

Vemos también que por defecto nos ha creado (y localizado) el fichero `InfoPlist.strings` que nos permite localizar las propiedades de `Info.plist`. Por ejemplo imaginemos que queremos que el nombre bajo el icono de la aplicación cambie según el idioma. Esta propiedad aparece en `Info.plist` como *Bundle display name*, pero realmente lo que necesitamos es el nombre interno de la propiedad, no la descripción que muestra Xcode de ella. Para ver esto, una vez abierto el fichero `Info.plist` en el editor, podemos pulsar sobre él con el botón derecho y seleccionar *Show Raw Keys/Values*. De esa forma veremos que la propiedad que buscamos se llama `CFBundleDisplayName`. Podemos localizarla en cada una de las versiones del fichero `InfoPlist.strings` de la siguiente forma:

```
// Versión en castellano del fichero InfoPlist.strings
"CFBundleDisplayName" = "MovilesUA";
// Versión en inglés del fichero InfoPlist.strings
"CFBundleDisplayName" = "MobileUA";
```

Si ejecutamos esta aplicación en el simulador, y pulsamos el botón *home* para volver a la pantalla principal, veremos que como nombre de nuestra aplicación aparece *MobileUA*. Si ahora desde la pantalla principal entramos en *Settings > General > International > Language* y cambiamos el idioma del simulador a español, veremos que el nombre de nuestra aplicación cambia a *MovilesUA*.

Con esto hemos localizado las cadenas de las propiedades del proyecto, pero ese fichero sólo se aplica al contenido de `Info.plist`. Si queremos localizar las cadenas de nuestra aplicación, por defecto se utilizará un fichero de nombre `Localizable.strings`, que seguirá el mismo formato (cada cadena se pone en una línea del tipo de las mostradas en el ejemplo anterior: `"identificador" = "cadena a mostrar";`).

Podemos crear ese fichero `strings` pulsando sobre el botón derecho del ratón sobre el grupo del navegador en el que lo queramos incluir y seleccionando *New File ...*. Como tipo de fichero seleccionaremos *iOS > Resource > Strings File*, y le llamaremos `Localizable`. No deberemos olvidar añadir las localizaciones desde el panel de

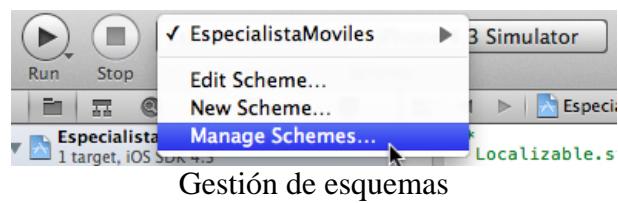
utilizadas. Más adelante veremos cómo acceder desde el código de la aplicación a estas cadenas localizadas.

## 5. Esquemas y acciones

Otro elemento que encontramos para definir la forma en la que se construye y ejecuta la aplicación son los esquemas. Los esquemas son los encargados de vincular los *targets* a las configuraciones para cada una de las diferentes acciones que podemos realizar con el producto (construcción, ejecución, pruebas, análisis estático, análisis dinámico, o distribución).

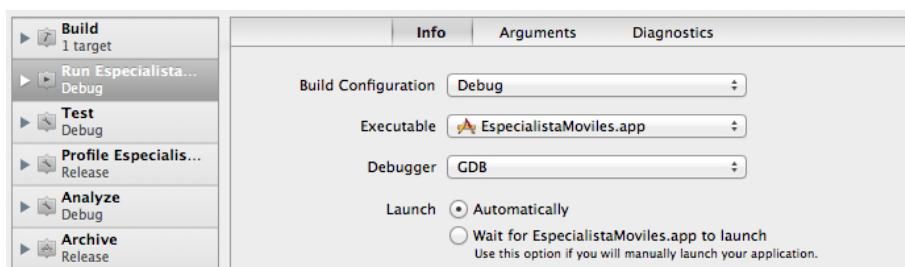
Por defecto nos habrá creado un único esquema con el nombre de nuestro proyecto, y normalmente no será necesario modificarlo ni crear esquemas alternativos, aunque vamos a ver qué información contiene para comprender mejor el proceso de construcción de la aplicación.

Para gestionar los esquemas debemos pulsar en el cuadro desplegable junto a los botones de ejecución de la aplicación:



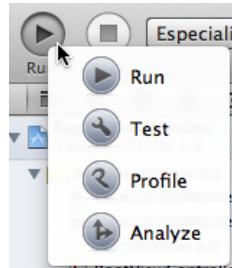
Gestión de esquemas

Si queremos editar el esquema actual podemos pulsar en *Edit Scheme ...*:



Edición de un esquema

Vemos que el esquema contiene la configuración para cada posible acción que realicemos con el producto. La acción *Build* es un caso especial, ya que para el resto de acciones siempre será necesario que antes se construya el producto. En esta acción se especifica el *target* que se va a construir. En el resto de acciones se especificará la configuración que se utilizará para la construcción del target en cada caso. Podemos ejecutar estas diferentes acciones manteniendo pulsado el botón izquierdo del ratón sobre el botón *Run*:



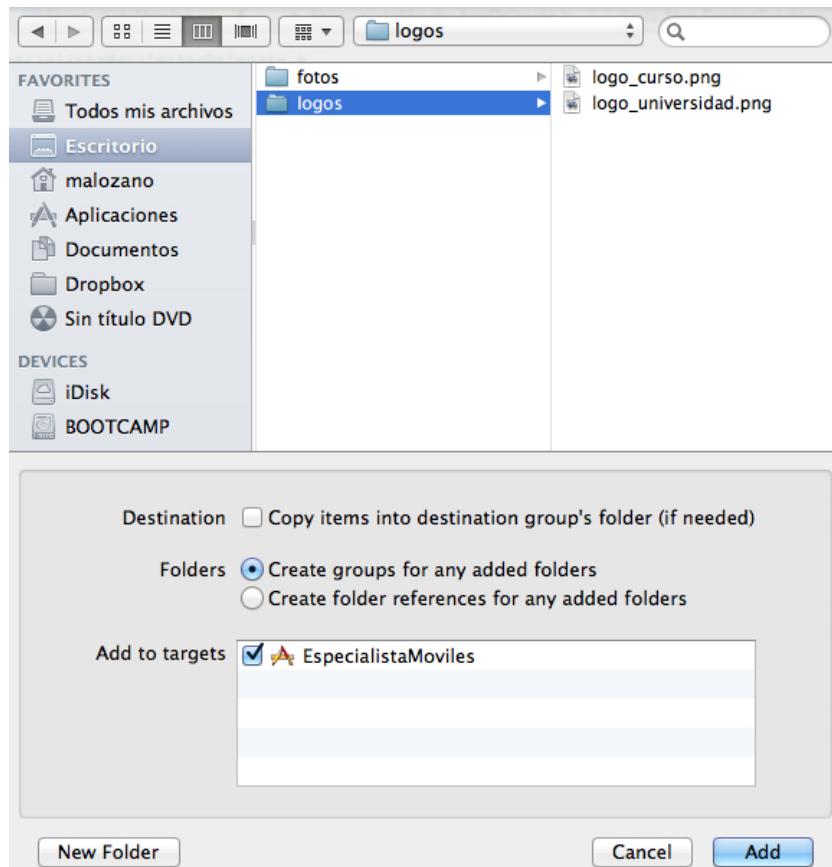
Ejecución de acciones

También pueden ser ejecutadas desde el menú *Product*, donde encontramos también la acción *Archive*, y la posibilidad de simplemente construir el producto sin realizar ninguna de las acciones anteriores. Las acciones disponibles son:

- *Build*: Construye el producto del *target* seleccionado en el esquema actual.
- *Run*: Ejecuta la aplicación en el dispositivo seleccionado (simulador o dispositivo real). Por defecto se construye con la configuración *Debug*.
- *Test*: Ejecuta las pruebas de unidad definidas en el producto. No funcionará si no hemos creado pruebas de unidad para el *target* seleccionado. Por defecto se construye con la configuración *Debug*.
- *Profile*: Ejecuta la aplicación para su análisis dinámico con la herramienta *Instruments*, que nos permitirá controlar en tiempo de ejecución elementos tales como el uso de la memoria, para poder así optimizarla y detectar posibles fugas de memoria. Más adelante estudiaremos esta herramienta con más detalles. Por defecto se construye con la configuración *Release*, ya que nos interesa optimizarla en tiempo de ejecución.
- *Analyze*: Realiza un análisis estático del código. Nos permite detectar construcciones de código incorrectas, como sentencias no alcanzables o posibles fugas de memoria desde el punto de vista estático. Por defecto se utiliza la configuración *Debug*.
- *Archive*: Genera un archivo para la distribución de nuestra aplicación. Para poder ejecutarla deberemos seleccionar como dispositivo destino *iOS Device*, en lugar de los simuladores. Utiliza por defecto la configuración *Release*, ya que es la versión que será publicada.

## 6. Recursos y grupos

Si queremos añadir recursos a nuestro proyecto, podemos pulsar con el botón derecho del ratón (*Ctrl-Click*) sobre el grupo en el que los queremos añadir, y seleccionar la opción *Add Files To "NombreProyecto" ...*. Tendremos que seleccionar los recursos del disco que queramos añadir al proyecto. Puede ser un único fichero, o un conjunto de ficheros, carpetas y subcarpetas.



Añadir un nuevo recurso

Bajo el cuadro de selección de fichero vemos diferentes opciones. La primera de ellas es un *checkbox* con la opción *Copy items into destination group's folder (if needed)*. Si lo seleccionamos, copiará los recursos que añadamos el directorio en el que está el proyecto (en caso de que no estén ya dentro). En caso contrario, simplemente añadirá una referencia a ellos. Resulta conveniente copiarlos para que así el proyecto sea autocontenido, y podamos transportarlo fácilmente sin perder referencias.

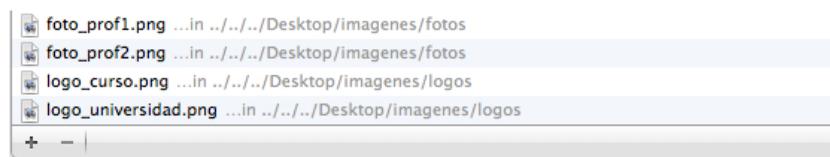
Bajo esta opción, nos da dos alternativas para incluir los ficheros seleccionados en el proyecto:

- *Create groups for any added folders*: Transforma las carpetas que se hayan seleccionado en grupos. Tendremos la misma estructura de grupos que la estructura de carpetas seleccionadas, pero los grupos sólo nos servirán para tener los recursos organizados dentro del entorno. Los grupos aparecerán como iconos de carpetas amarillas en el navegador.



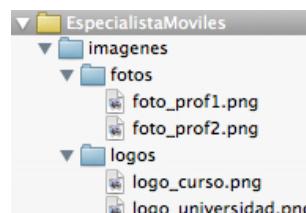
Grupo de recursos

Realmente todos los ficheros se incluirán en la raíz del paquete generado. Esto podemos comprobarlo si vamos a la sección *Build Phases* del *target* en el que hayamos incluido el recurso y consultamos la sección *Copy Bundle Resources*.



Copia de recursos en grupos

- *Create folder references for any added folders:* Con esta segunda opción las carpetas seleccionadas se incluyen como carpetas físicas en nuestro proyecto, no como grupos. Las carpetas físicas aparecen con el icono de carpeta de color azul.



Carpeta de recursos

En este caso la propia carpeta (con todo su contenido) será incluida en el paquete (*bundle*) generado. Si consultamos *Build Phases* veremos que lo que se copia al paquete en este caso es la carpeta completa.



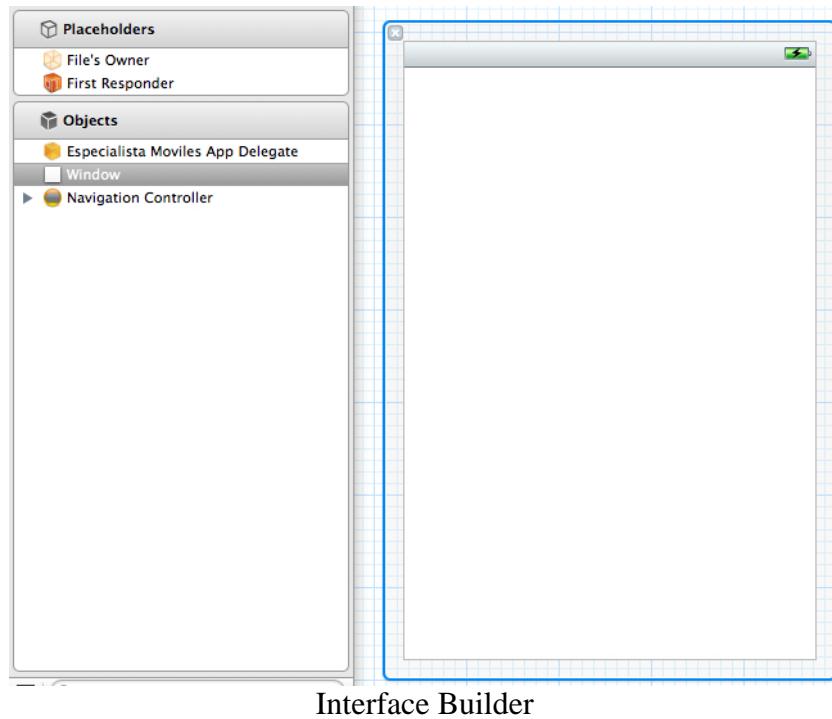
Copia de recursos en carpetas

Todos los recursos añadidos pueden ser localizados también como hemos visto anteriormente.

## 7. Interface Builder

Si en el navegador seleccionamos un fichero de tipo *xib*, en los que se define la interfaz, en el editor se abrirá la herramienta Interface Builder, que nos permitirá editar dicha

interfaz de forma visual. En versiones anteriores de Xcode Interface Builder era una herramienta independiente, pero a partir de Xcode 4 se encuentra integrado en el mismo entorno.

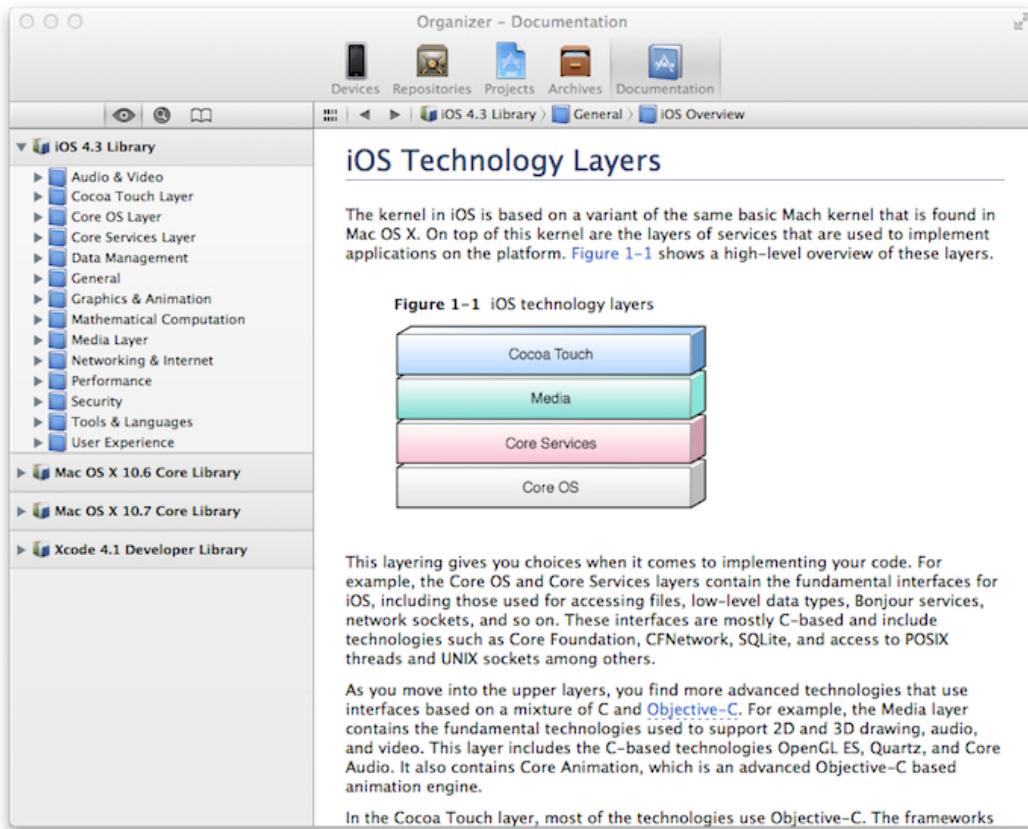


Veremos esta herramienta con más detalle cuando estudiemos la creación de la interfaz de las aplicaciones iOS.

## 8. Organizer

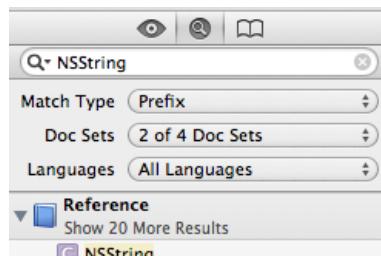
El *organizer* es una ventana independiente de la interfaz principal de Xcode desde donde podemos gestionar diferentes elementos como los dispositivos reales de los que disponemos para probar la aplicación, los repositorios de control de fuentes (SCM) que utilizamos en los proyectos, los archivos generados para la publicación de aplicaciones, y la documentación sobre las herramientas y las APIs con las que contamos en la plataforma iOS.

Podemos abrir *organizer* pulsando sobre el botón en la esquina superior derecha de la ventana principal de Xcode.



### Ventana del Organizer

El uso más frecuente que le daremos el *organizer* será el de consultar la documentación de la plataforma iOS. La navegación por la documentación se hará de forma parecida a como se haría en un navegador. Normalmente nos interesarán buscar un determinado ítem. Para ello, en la barra lateral izquierda podemos especificar los criterios de búsqueda:



### Búsqueda en la documentación

Es recomendable abrir los criterios avanzados de búsqueda y en *Doc Sets* eliminar los conjuntos de documentación de Mac OS X, ya que no nos interesarán para el desarrollo en iOS y así agilizaremos la búsqueda.

## 9. Repositorios SCM

Anteriormente hemos visto que Xcode nos pone muy fácil crear un repositorio Git para nuestro proyecto, ya que basta con marcar una casilla durante la creación del proyecto. Sin embargo, si queremos utilizar un repositorio remoto deberemos realizar una configuración adicional. Las opciones para trabajar con repositorios en Xcode son bastante limitadas, y en muchas ocasiones resulta poco intuitiva la forma de trabajar con ellas. Vamos a ver a continuación cómo trabajar con repositorios Git y SVN remotos.

Para gestionar los repositorios deberemos ir a la pestaña *Repositories* del *Organizer*. Ahí veremos en el lateral izquierdo un listado de los repositorios con los que estamos trabajando. Si hemos marcado la casilla de utilizar Git al crear nuestros proyectos, podremos ver aquí la información de dichos repositorios y el historial de revisiones de cada uno de ellos.

### 9.1. Repositorios Git

Si hemos creado un repositorio Git local para nuestro proyecto, será sencillo replicarlo en un repositorio Git remoto. En primer lugar, necesitaremos crear un repositorio remoto. Vamos a ver cómo crear un repositorio privado en BitBucket ([bitbucket.org](http://bitbucket.org)).

- En primer lugar, deberemos crearnos una cuenta en BitBucket, si no disponemos ya de una.
- Creamos desde nuestra cuenta de BitBucket un repositorio (*Repositories > Create repository*).
- Deberemos darle un nombre al repositorio. Será de tipo Git y como lenguaje especificaremos Objective-C.

The screenshot shows the BitBucket interface for creating a new repository. At the top, there's a navigation bar with the BitBucket logo, 'Explore', 'Dashboard', and 'Repositories'. Below it is a dark blue header bar with the text 'Create new repository' and 'Start from scratch.' In the main form area, there are several input fields and options:

- Owner:** A dropdown menu showing 'malozano'.
- Name (required):** An input field containing 'Prueba'. To its right are three small icons: a green square with a white gear, a red square with a white plus sign, and a yellow square with a white minus sign. Next to the name is a checked checkbox labeled 'Private'.
- Repository type:** A radio button group where 'Git' is selected, and 'Mercurial' is unselected.
- Project management:** A checkbox group where 'Issue tracking' is unselected and 'Wiki' is unselected.
- Language:** A dropdown menu showing 'Objective-C'.
- Description:** A large text area for entering a description, which is currently empty.
- Website:** An input field for a website URL, currently empty.
- Create repository:** A button at the bottom left of the form.

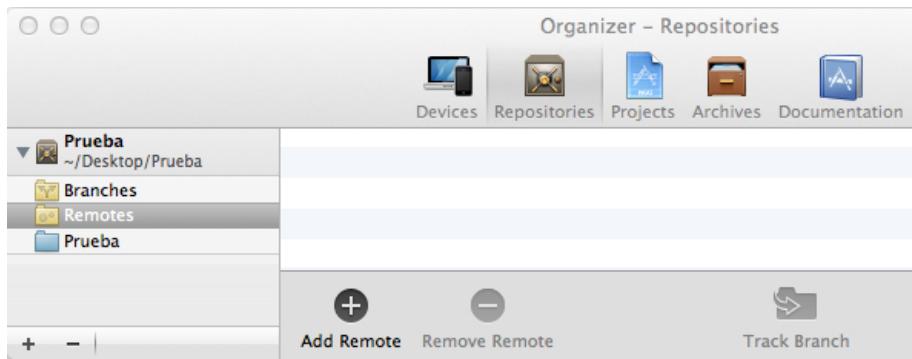
#### Creación de un repositorio BitBucket

- Una vez hecho esto, veremos el repositorio ya creado, en cuya ficha podremos encontrar la ruta que nos dará acceso a él.

The screenshot shows the BitBucket repository details page for 'Prueba'. At the top, there's a navigation bar with tabs: 'Overview' (which is active), 'Downloads (0)', 'Pull requests (0)', 'Source', 'Commits', and 'Admin'. Below the tabs, the repository owner is listed as 'malozano / Prueba'. Underneath, there's a section for cloning the repository with the text 'Clone this repository (size: 580 bytes): [HTTPS](https://malozano@bitbucket.org/malozano/prueba.git) / [SSH](ssh://git@bitbucket.org:7999/malozano/prueba.git) / [SourceTree](#)' and the command '\$ git clone https://malozano@bitbucket.org/malozano/prueba.git'.

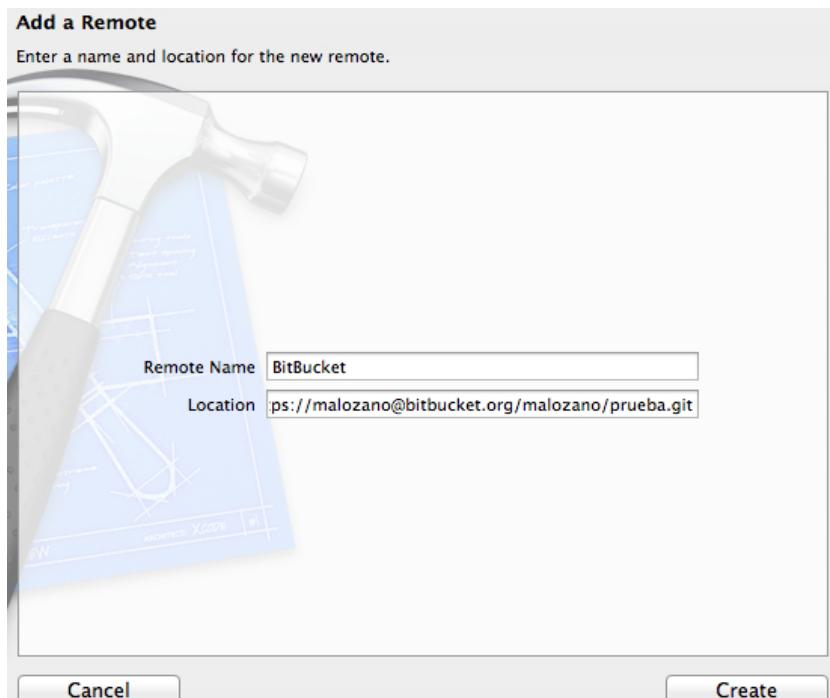
Ficha del repositorio en BitBucket

A continuación volvemos a Xcode, donde tenemos un proyecto ya creado con un repositorio Git local. Para vincular este repositorio local con el repositorio remoto que acabamos de crear, deberemos ir a la pestaña *Repositories* de la ventana del *Organizer*, donde veremos el repositorio local de nuestro proyecto.



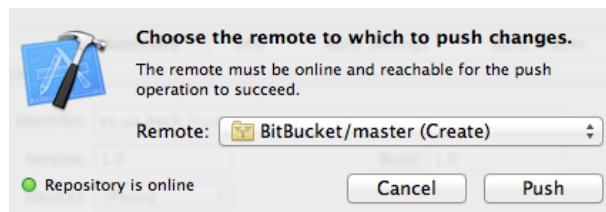
Repositorio en Organizer

Dentro de este repositorio local, veremos una carpeta *Remotes*. Entraremos en ella, y pulsaremos el botón *Add Remotes* de la parte inferior de la pantalla. Ahora nos pedirá un nombre para el repositorio remoto y la ruta en la que se puede localizar. Utilizaremos aquí como ruta la ruta de tipo HTTPS que vimos en la ficha del repositorio de BitBucket:



Añadir repositorio remoto

Una vez añadido el repositorio remoto, deberemos introducir el login y password que nos den acceso a él. Con esto ya podemos volver al proyecto en Xcode, y enviar los cambios al servidor. En primer lugar deberemos hacer un *Commit* de los cambios en el repositorio local, si los hubiera (*File > Source Control > Commit...*). Tras esto, ya podremos subir estos cambios al repositorio remoto (*Push*). Para ello, seleccionamos la opción *File > Source Control > Push...*, tras lo cual nos pedirá que indiquemos el repositorio remoto en el que subir los cambios. Seleccionaremos el repositorio que acabamos de configurar.



Push en repositorio remoto

Tras realizar esta operación en la web de BitBucket podremos ver el código que hemos subido al repositorio:

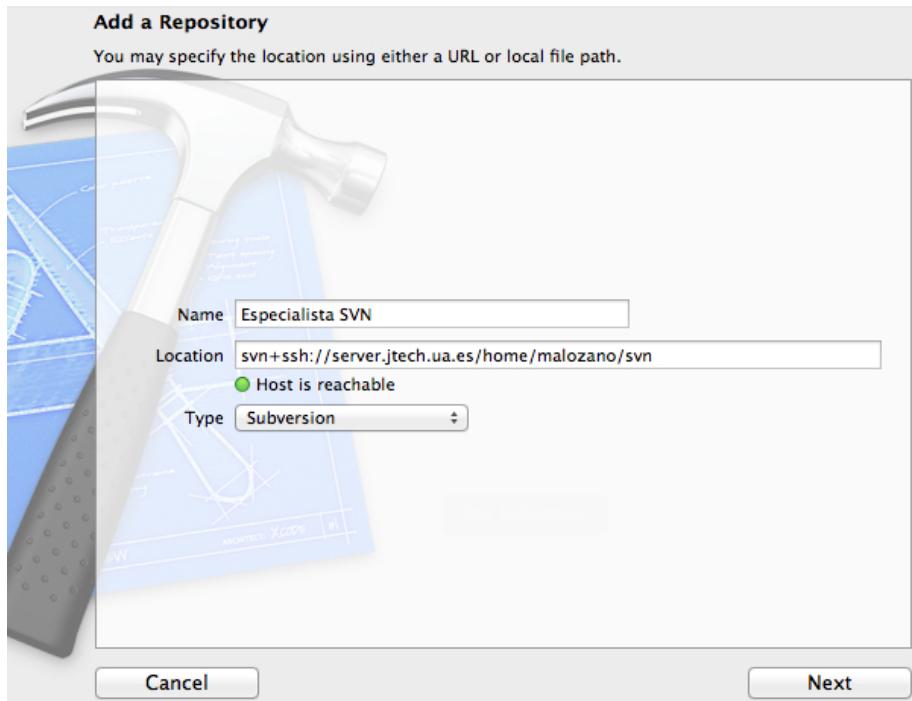
Código en BitBucket

## 9.2. Repositorios SVN

Vamos a ver la forma de acceder a un repositorio SVN remoto. Si queremos añadir un nuevo repositorio de forma manual deberemos pulsar el botón (+) que hay en la esquina inferior izquierda de la pantalla de repositorios de *Organizer*. Una vez pulsado el botón (+) y seleccionada la opción *Add Repository ...*, nos aparecerá la siguiente ventana donde introducir los datos de nuestro repositorio:

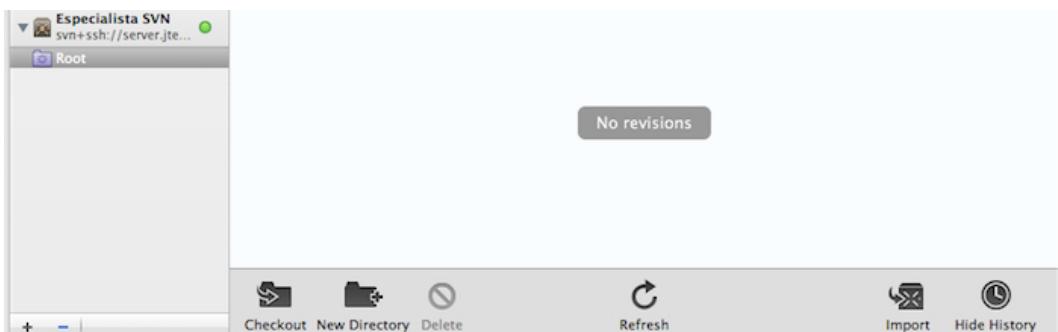
Le daremos un nombre y pondremos la ruta en la que se encuentra. Tras esto, pulsamos *Next*, y nos pedirá el login y password para acceder a dicho repositorio. Una vez

introducidos, nos llevará a una segunda pantalla donde nos pedirá las rutas de `trunk`, `branches`, y `tags` en el repositorio. Si el repositorio está todavía vacío, dejaremos estos campos vacíos y pulsaremos *Add* para añadir el nuevo repositorio.



Configurar un nuevo repositorio SVN

Una vez creado, lo veremos en el lateral izquierdo con un elemento *Root* que corresponderá a la raíz del repositorio. Si seleccionamos esta ruta raíz, en la zona central de la pantalla veremos su contenido y podremos crear directorios o importar ficheros mediante los botones de la barra inferior:



Acceso al repositorio SVN

Aquí podemos crear el *layout* del repositorio. Para repositorios SVN se recomienda que en el raíz de nuestro proyecto tengamos tres subdirectorios: `trunk`, `branches`, y `tags`. Podemos crearlos con el botón *New Directory*.

Una vez creados los directorios, pinchamos sobre el ítem con el nombre de nuestro repositorio en la barra de la izquierda e indicamos la localización de los directorios que acabamos de crear. Si ponemos bien las rutas junto a ellas aparecerá una luz verde:



Configuración del layout del repositorio

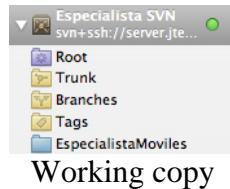
Además de *Root*, ahora veremos en nuestro repositorio también las carpetas de los elementos que acabamos de crear. El código en desarrollo se guardará en *trunk*, por lo que es ahí donde deberemos subir nuestro proyecto.

Por desgracia, no hay ninguna forma de compartir un proyecto de forma automática desde el entorno Xcode, como si que ocurre con Eclipse. De hecho, las opciones para trabajar con repositorios SCM en Xcode son muy limitadas y en muchas ocasiones en la documentación oficial nos indican que debemos trabajar desde línea de comando. Una de las situaciones en las que debe hacerse así es para subir nuestro proyecto por primera vez.

Vamos a ver una forma alternativa de hacerlo utilizando el *Organizer*, en lugar de línea de comando.

- Lo primero que necesitaremos es tener un proyecto creado con Xcode en el disco (cerraremos la ventana de Xcode del proyecto, ya que no vamos a seguir trabajando con esa copia).
- En el *Organizer*, entramos en el directorio *trunk* de nuestro repositorio y aquí seleccionamos *Import* en la barra inferior.
- Nos abrirá un explorador de ficheros para seleccionar los ficheros que queremos subir al repositorio, seleccionaremos el directorio raíz de nuestro proyecto (el directorio que contiene el fichero *xcodeproj*).
- Con esto ya tenemos el proyecto en nuestro repositorio SVN, pero si ahora abrimos el proyecto del directorio de nuestro disco local en el que está guardado, Xcode no reconocerá la conexión con el repositorio, ya que no es lo que se conoce como una *working copy* (no guarda la configuración de conexión con el repositorio SVN).
- Para obtener una *working copy* deberemos hacer un *checkout* del proyecto. Para ello desde el *Organizer*, seleccionamos la carpeta *Trunk* de nuestro repositorio, dentro de ella nuestro proyecto, y pulsamos el botón *Checkout* de la barra inferior. Nos pedirá un lugar del sistema de archivos donde guardar el proyecto, y una vez descargado nos preguntará si queremos abrirlo con Xcode.

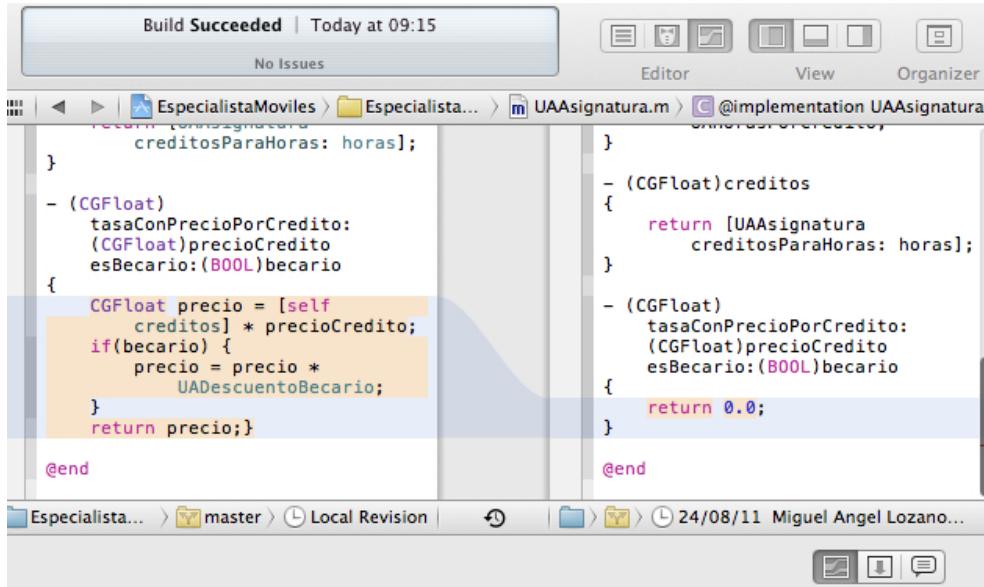
En los elementos del repositorio ahora veremos una carpeta azul con el nombre del proyecto descargado. Esta carpeta azul simboliza una *working copy* del proyecto. Al tener esta *working copy* reconocida, ya podremos realizar desde el mismo entorno Xcode operaciones de SVN en dicho proyecto.



Cada vez que hagamos un cambio en un fichero, en el navegador de Xcode aparecerá una a su lado indicando que hay modificaciones pendientes de ser enviadas, y de la misma forma, cuando añadamos un fichero nuevo, junto a él aparecerá una para indicar que está pendiente de ser añadido al repositorio. Para enviar estos cambios al repositorio, desde Xcode seleccionaremos *File > Source Control > Commit*.

Si ya tuviesemos una *working copy* válida de un proyecto en el disco (es decir, una copia que contenga los directorios `.svn` con la configuración de acceso al repositorio), podemos añadirla al *Organizer* directamente desde la vista de repositorios, pulsando el botón (+) de la esquina inferior izquierda y seleccionando *Add Working Copy....* Nos pedirá la ruta local en la que tenemos guardada la *working copy* del proyecto.

A parte de las opciones para el control de versiones que encontramos al pulsar con el botón derecho sobre nuestro proyecto en el apartado *Source Control*, en Xcode también tenemos el editor de versiones. Se trata de una vista del editor que nos permite comparar diferentes versiones de un mismo fichero, y volver atrás si fuese necesario. Para abrir este editor utilizaremos los botones de la esquina superior derecha de la interfaz, concretamente el tercer botón del grupo *Editor*. En el editor de versiones podremos comparar la revisión local actual de cada fichero con cualquiera de las revisiones anteriores.

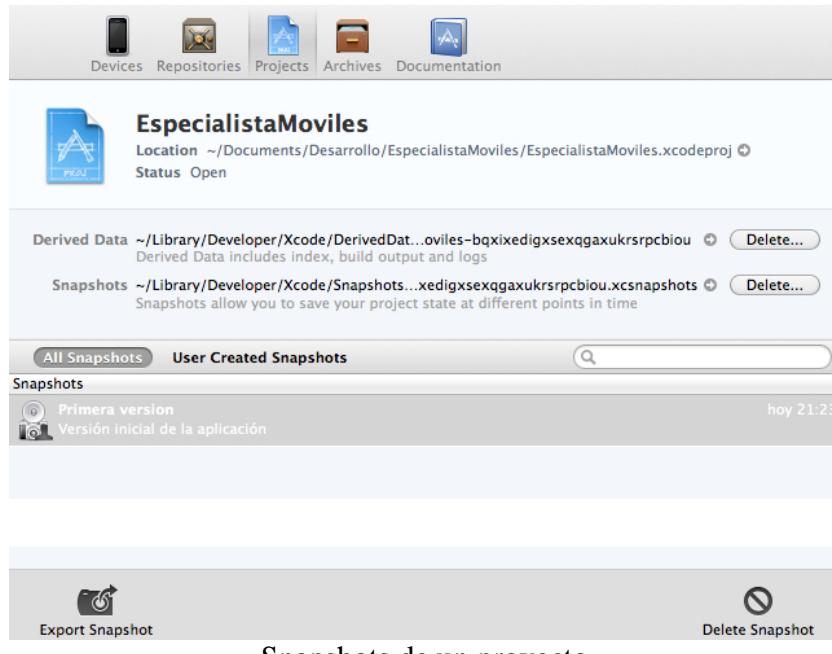


## 10. Snapshots

Una forma más rudimentaria de guardar copias de seguridad de los proyectos es la generación de *snapshots*. Un *snapshot* no es más que la copia de nuestro proyecto completo en un momento dado. Xcode se encarga de archivar y organizar estas copias.

Para crear un *snapshot* debemos seleccionar en Xcode *File > Create Snapshot ...*, para el cual nos pedirá un nombre y una descripción.

En la vista *Projects* del *Organizer* veremos para cada proyecto la lista de *snapshots* creados.



Snapshots de un proyecto

Desde aquí podremos exportar un *snapshot* para guardar el proyecto en el lugar del disco que especifiquemos tal como estaba en el momento en el que se tomó la instantánea, y así poder abrirlo con Xcode. En esta pantalla también podemos limpiar los directorios de trabajo del proyecto y borrar los *snapshots*.

Desde Xcode, también se puede restaurar un *snapshot* con la opción *File > Restore snapshot....* Tendremos que seleccionar el *snapshot* que queremos restaurar.

## 11. Ejecución y firma

Como hemos comentado anteriormente, para probar las aplicaciones en dispositivos reales necesitaremos un certificado con el que firmarlas, y para obtener dicho certificado tendremos que ser miembros de pago del *iOS Developer Program*, o bien pertenecer al *iOS University Program*. Para acceder a este segundo programa, se deberá contar con una cuenta Apple, y solicitar al profesor responsable una invitación para acceder al programa con dicha cuenta. El programa es gratuito, pero tiene la limitación de que no podremos publicar las aplicaciones en la App Store, sólo probarlas en nuestros propios dispositivos.

Vamos a ver a continuación cómo obtener dichos certificados y ejecutar las aplicaciones en dispositivos reales.

### 11.1. Creación del par de claves

El primer paso para la obtención del certificado es crear nuestras claves privada y pública.

Para ello entraremos en el portal de desarrolladores de Apple:

<http://developer.apple.com>

Accedemos con nuestro usuario, que debe estar dado de alta en alguno de los programas comentados anteriormente.

The screenshot shows the Apple Developer website's iOS Dev Center. At the top, there are links for Technologies, Resources, Programs, Support, and Member Center. Below that is a search bar. The main navigation bar includes links for iOS Dev Center, Mac Dev Center, and Safari Dev Center. A user profile is shown at the top right. The main content area features a "iOS SDK 4.3" section with links for Downloads and Getting Started Videos. To the right, there's a sidebar for the "iOS Developer Program" with links for iOS Provisioning Portal, Apple Developer Forums, and Developer Support Center. The overall layout is clean and professional.

### Developer Center

Desde este portal, accedemos al *iOS Provisioning Portal*.

The screenshot shows the iOS Provisioning Portal. On the left, a sidebar lists Home, Certificates, Devices, App IDs, and Provisioning. The main content area has a heading "Welcome to the iOS Provisioning Portal" and a message about the portal's purpose. It features a callout box with an exclamation mark icon, encouraging users to visit the Member Center for team, account, and program info. Below this, there's a section titled "Get your application on an iOS with the Development Provisioning Assistant", which includes an image of a smartphone and a developer tool icon. A "Launch Assistant" button is located at the bottom of this section. The overall design is modern and user-friendly.

### Provisioning Portal

Desde aquí podemos gestionar nuestros certificados y los dispositivos registrados para

poder utilizarlos en el desarrollo de aplicaciones. Vemos también que tenemos disponible un asistente para crear los certificados por primera vez. El asistente se nos irá dando instrucciones detalladas y se encargará de configurar todo lo necesario para la obtención de los certificados y el registro de los dispositivos. Sin embargo, con Xcode 4 el registro de los dispositivos se puede hacer automáticamente desde el *Organizer*, así que nos resultará más sencillo si en lugar del asistente simplemente utilizamos el *iOS Provisioning Portal* para obtener el certificado del desarrollador, y dejamos el resto de tareas a *Organizer*.

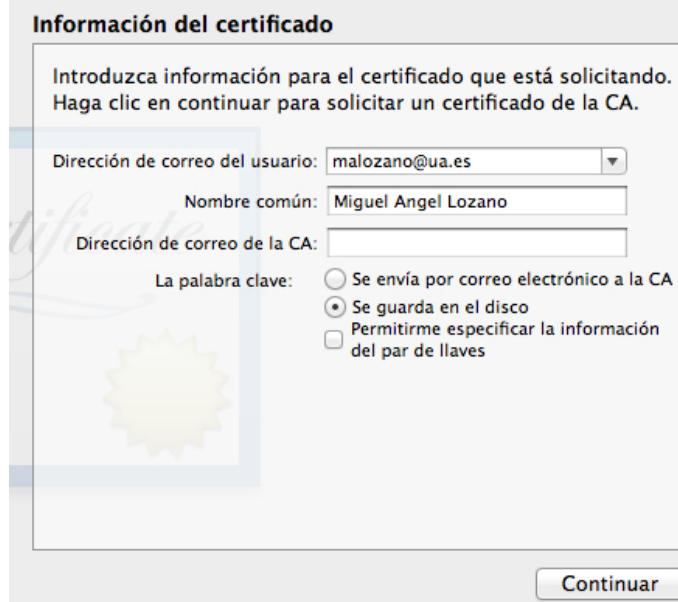
The screenshot shows the iOS Provisioning Portal interface. At the top, it says "Welcome, Miguel Angel Lozano Ortega | Edit Profile | Log out". Below that, it displays "Provisioning Portal : Universidad de Alicante (Dpto. de Ciencia de la Computacion e I.A.)" and a link to "Go to iOS Dev Center". On the left, there's a sidebar with links: Home, Certificates (which is selected and highlighted in blue), Devices, App IDs, and Provisioning. The main content area has tabs: Development (selected), History, and How To. Under "Development", it says "Current Development Certificates". It shows a table with one row: "Your Certificate". The table columns are Name, Provisioning Profiles, Expiration Date, Status, and Action. A note below the table says "You currently do not have a valid certificate" and a button labeled "Request Certificate". At the bottom, there's a note: "\*If you do not have the WWDR intermediate certificate installed, click here to download now."

### Gestión de certificados de desarrollador

Entraremos por lo tanto en *Certificates*, y dentro de la pestaña *Development* veremos los certificados con los que contamos actualmente (si es la primera vez que entramos no tendremos ninguno), y nos permitirá solicitarlos o gestionarlos. Antes de solicitar un nuevo certificado, descargaremos desde esta página el certificado WWDR, y haremos doble *click* sobre él para instalarlo en nuestros llaveros (*keychain*). Este certificado es necesario porque es el que valida los certificados de desarrollador que emite Apple. Sin él, el certificado que generemos no sería de confianza.

Una vez descargado e instalado dicho certificado intermedio, pulsaremos sobre el botón *Request Certificate* para comenzar con la solicitud de un nuevo certificado de desarrollador.

Nos aparecerán las instrucciones detalladas para solicitar el certificado. Deberemos utilizar la herramienta *Acceso a Llaveros* (*Keychain Access*) para realizar dicha solicitud, tal como explican las instrucciones. Abrimos la herramienta y accedemos a la opción del menú *Acceso a Llaveros > Asistente para Certificados > Solicitar un certificado de una autoridad de certificación ....*



### Solicitud de un certificado de desarrollador

Aquí tendremos que poner nuestro *e-mail*, nuestro nombre, e indicar que se guarde en el disco. Una vez finalizado, pulsamos *Continuar* y nos guardará un fichero `.certSigningRequest` en la ubicación que seleccionemos.

Con esto habremos generado una clave privada, almacenada en los llaveros, y una clave pública que se incluye en el fichero de solicitud de certificado. Volvemos ahora al *iOS Provisioning Portal*, y bajo las instrucciones para la solicitud del certificado vemos un campo para enviar el fichero. Enviaremos a través de dicho campo el fichero de solicitud (`.certSigningRequest`) generado, y en la página *Certificates > Development* aparecerá nuestro certificado como pendiente. Cuando Apple emita el certificado podremos descargarlo a través de esta misma página (podemos recargarla pasados unos segundos hasta que el certificado esté disponible).

Name	Provisioning Profiles	Expiration Date	Status	Action
Miguel Angel Lozano Ortega		Aug 23, 2012	Issued	<a href="#">Download</a> <a href="#">Revoke</a>

### Certificado listo para descargar

Una vez esté disponible, pulsaremos el botón *Download* para descargarlo y haremos doble *click* sobre él para instalarlo en nuestros llaveros. Una vez hecho esto, en *Acceso a*

*Llaveros > Inicio de sesión > Mis certificados* deberemos ver nuestro certificado con la clave privada asociada.



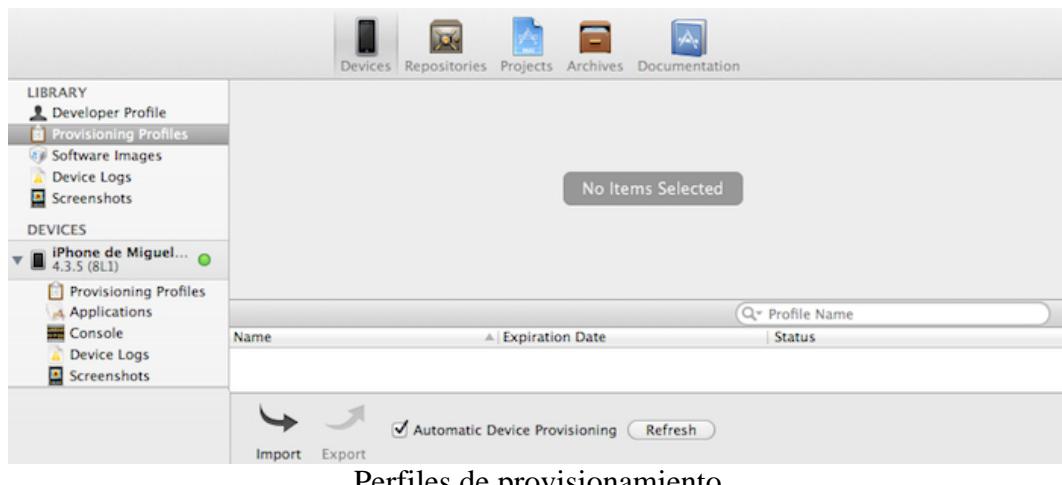
#### Importante

El certificado generado no podrá utilizarse en otros ordenadores para firmar la aplicación, ya que la clave privada está incluida en nuestros llaveros, pero no en el certificado, y será necesaria para poder firmar. Si queremos trabajar en otro ordenador tendremos que exportar la clave privada desde *Acceso a Llaveros*.

## 11.2. Perfil de provisionamiento

Una vez contamos con nuestro certificado de desarrollador instalado y con su clave privada asociada en nuestro llavero, podemos crear un perfil de provisionamiento para poder probar nuestras aplicaciones en nuestros dispositivos.

Para ello abriremos el *Organizer* e iremos a la sección *Devices*. Aquí entraremos en *Developer Profiles* y comprobaremos que ha reconocido correctamente nuestro perfil de desarrollador tras instalar el certificado. Ahora lo que necesitamos son perfiles de provisionamiento para poder instalar la aplicación en dispositivos concretos. Si entramos en *Provisioning Profiles* veremos que no tenemos ninguno por el momento, y nos aseguraremos que en la parte inferior tengamos marcado *Automatic Device Provisioning*.



Ahora conectaremos mediante el cable USB el dispositivo que queramos registrar y veremos que aparece en la barra de la izquierda. Pulsamos sobre el nombre del dispositivo en dicha barra y abrirá una pantalla con toda su información, entre ella, la versión de iOS y el identificador que se utilizará para registrarla en el *iOS Provisioning*

Portal.



Gestión de los dispositivos

Vemos que no tiene ningún perfil de provisionamiento disponible, pero abajo vemos un botón *Add to Portal* que nos permite registrar automáticamente nuestro dispositivo para desarrollo. Al pulsarlo nos pedirá nuestro *login* y *password* para acceder al portal de desarrolladores, y automáticamente se encargará de registrar el dispositivo, crear los perfiles de provisionamiento necesarios, descargarlos e instalarlos en Xcode.

#### Nota

Si estamos usando el *iOS University Program* al final del proceso, cuando esté obteniendo el perfil de distribución, nos saldrá un mensaje de error indicando que nuestro programa no permite realizar dicha acción. Esto es normal, ya que el perfil universitario no permite distribuir aplicaciones, por lo que la generación del perfil de distribución deberá fallar. Pulsaremos *Aceptar* y el resto del proceso se realizará correctamente.

Si ahora volvemos al *iOS Developer Portal* veremos que nuestro dispositivo se encuentra registrado ahí (sección *Devices*), y que contamos con un perfil de provisionamiento (sección *Provisioning*).

Ahora podemos volver a Xcode y ejecutar nuestra aplicación en un dispositivo real. Para ello seleccionamos nuestro dispositivo en el cuadro junto a los botones de ejecución, y pulsamos el botón *Run*.



Ejecutar en un dispositivo real

*El entorno Xcode*

