?npc l bc _ npmep_ k_p rsq npmng_q
_njga_agm l cq n_p_ gPhone c iPad

?njga_agm l cq gN f m l c c gN_b n_p_ Npg l agng_ l rcq

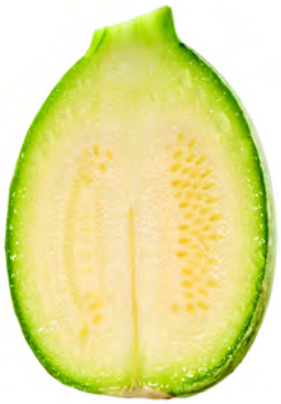
Dr. Rory Lewis

Prefacio de Ben Easton

Apress®

Traducción:
amacor, mhergon
Euki y un servidor (CO-2)

Aplicaciones iPhone e iPad para Principiantes



Dr. Rory Lewis

TRADUCCIÓN:
CO-2: Capítulos 1,2,4,5,7 y 9
amacor: Capítulos 3,6
mhergon: Capítulo 8
Euki: Capítulo 9
iPadForos.com

Apress®

Contenidos de un Vistazo

Contenidos

Prefacio: Acerca del Autor

"Rory y yo nos conocimos en Los Ángeles en 1983. Me recuerda a uno de los personajes de mi película favorita, Buckaroo Banzai—siempre yendo en seis direcciones a la misma vez. Si le paras y le preguntas qué está haciendo, te responderá con un sorprendente lujo de detalles. Disciplinado y amigable, tiene la habilidad de explicar lo altamente abstracto de un modo muy simple. Siempre cumple con lo que se propone, y te ayudaremos a hacer lo mismo.

Por Qué Congeniará con el Dr. Lewis

Mientras asistía a la Universidad de Siracusa como estudiante de Ingeniería Informática, Rory se las apañaba para asistir a clases y ganar dinero para mantener a su familia. En 1990, consiguió un trabajo en el campus como supervisor en los Laboratorios de Computación en el Colegio de Ingeniería LC Smith. A pesar de estar lidiando con temas propios del programa de Ingeniería Electrónica, siempre estaba en el Help Desk. Fue una experiencia algo desalentadora para Rory, ya que su trabajo se limitaba a ayudar a sus compañeros de estudios en dudas de equipo de laboratorio. Los compañeros le hacían cada vez preguntas más profundas y difíciles: *"Tío, has entendido la tarea de asignamiento de cálculo? Puedes ayudarme?!"*

Estos estudiantes asumieron que dado que Rory era Supervisor, conocía todas las respuestas. Temeroso y lleno de dudas sobre sí mismo, buscó la forma de ayudarles sin tener que revelar sus propias deficiencias y carencias. Rory aprendió a responder con un: *"Volvamos a lo elemental. Recuerdas que la semana pasada el profesor nos presentó una ecuación*…? Volviendo a lo básico, recordando lo aprendido, Rory empezó a desarrollar una técnica que, la mayoría de las veces, daba buenos resultados. En su último año, colas de estudiantes esperaban en el Help Desk a Rory, en las noches que le trocaba trabajar.

17 Años Después

Imagínese a un profesor alocado de pelo largo caminando por el campus de la Universidad de Colorado en Colorado Springs, vestido de profesor a la vieja usanza, pero con un toque bastante descuidado. Mientras camina hacia el Edificio de Ingeniería es recibido y saludado por estudiantes y profesores con sonrisas y saludos con la cabeza, asombrados viendo su vestimenta: chaqueta de tela, camiseta Grateful Dead, pantalones khaki y sandalias. Conforme camina por el hall del Departamento de Ciencias de Computación, se forman filas de estudiantes esperando a la puerta de su despacho, reminiscencia de aquellas colas que se formaban en el Help Desk en aquellos años en que los estudiantes le esperaban haciendo cola en el Help Desk del Laboratorio de Informática. Los estudiantes le saludan con un "Buenos días, Dr. Lewis!" . Muchos de estos estudiantes ni siquiera van a su clase, pero saben perfectamente que el Dr. Lewis los atenderá y les ayudará de todos modos.

Pasado—Presente—Futuro

El Dr. Lewis tiene tres estudios académicos. Obtuvo la Licenciatura en Ciencias de Computación en la Universidad de Siracusa, El Colegio LC Smith de Ingeniería es una de las mejores escuelas del país. AMD y Microsoft envían a sus empleados más aventajados a estudiar las tesis.

Una vez completada su Licenciatura (con especialidad en matemáticas de circuitos electrónicos en procesadores), ingresó es la Universidad de Derecho de Siracusa. Durante su primer verano, Fulbright & Jaworski, el bufete de abogados más extendido del país, reclutó a Rory para trabajar en su oficina de Austin, donde algunos de los abogados se especializan en litigios de patentes propiedad intelectual de altas tecnologías. Experiencia en tareas administrativas, Lewis trabajó en el famoso caso *AMD v. Intel* ; ayudando a evaluar los algoritmos de matemática de cirtutería de microprocesadores eléctricos.

Durante su segundo verano en la Universidad de Derecho, la firma Skjerven, Morrill, MacPherson, Franklin, & Friel—la otra empresa con la que trabajaron en el caso *AMD v. Intel*—reclutó Rory para trabajar en su departamento de Silicon Valley (San José y San Francisco). Después de varios años inmerso en los estudios de la Ley, recibió su Doctorado en Siracusa y empezó a darse cuenta que su verdadera pasión era las *matemáticas* de computación, no las ramificaciones legales del hardware y software. Prefería un ambiente creativo y de aprendizaje a la lucha y el litigio del derecho.

Después de tres años fuera de la academia, Rory Lewis se trasladó al sur para conseguir su Doctorado en Ciencias de Computación en la Universidad de North Carolina, en Charlotte, donde estudió bajo la tutela del Dr. Zbigniew W. Ras, conocido mundialmente por sus innovaciones en algoritmos, metodologías y bases de datos multimedia. Mientras preparaba su Doctorado, Lewis impartió cursos de ciencia de computación a estudiantes de ingeniería informática, así como cursos de comercio electrónico y cursos de programación para estudiantes MBA.

Una vez conseguido el Doctorado en Ciencias de Computación, Rory aceptó una plaza en la Universidad de Ciencias de Computación de Colorado Springs, donde su investigación se basó en las matemáticas computacionales de las neurociencias. Más recientemente, ha sido coautor de un gran ensayo de análisis matemático del génesis de la epilepsia en relación al hipotálamo. Sin embargo, con la llegada del revolucionario iPhone de Apple, una plataforma extraordinariamente flexible-y *comercial*— para miniaplicaciones, juegos y herramientas de computación personal, despertó en él un gran interés y empezó a experimentar y programar para su propio deleite. Una vez establecidos y asimilados todos los conocimientos, Lewis pensó en crear una clase para aplicaciones iPhone que podría incluir a estudiantes no-ingenieros. Con sus conocimientos como testador de betas para el iPhone, empezó a integrar los parámetros de la plataforma iPad en sus planes de estudios- incluso antes de la salida oficial en Abril de 2.010.

La clase fué un rotundo éxito y el boca a boca fue tremendamente positivo, tanto por parte de estudiantes como de colegas de profesión. Cuando se le presentó la oportunidad de convertir su curso en un libro que saldría publicado por la editorial Apress, el Dr. Lewis no la dejó escapar. Aceptó la oferta para convertir el curso, notas de clase, y vídeos en el libro que en estos momentos tienes en tus manos

Por Qué Escribir Este Libro?

Las razones que llevaron al Dr. Lewis a escribir este libro son las mismas razones por las que se planteó crear una clase para Ingenieros y no-Ingenieros: el desafío y el divertimento! Según Lewis, el iPhone y el iPad son"*...dos de los aparatos más poderosos, tecnológicamente avanzados y cools que jamás de han fabricado!*"

Está fascinado por el hecho de que, bajo esa pantalla de imagines en alta resolución y atractivos iconos, el iPhone y el iPad están creados bajo programación en *Objective-C,* un lenguaje de programación increíblemente difícil y avanzado. Cada vez más, Lewis fue solicitado por estudiantes y colegas que querían crear aplicaciones para el iPhone, para conocer su opinión acerca de sus ideas. Parecía que, con cada nueva versión del iPhone, por no hablar de la llegada del iPad, el interés por adquirir conocimientos para la programación de aplicaciones, crecía más y más. Ideas maravillosas e innovadoras, simplemente necesitaban el canal adecuado para que el mundo pudiera conocerlas.

Sin embargo, la gente que escribe libros sobre Objective-C lo hace para gente que tiene conocimientos avanzados de Java, C#, o C++, libros que no son de ayuda para la persona que no tiene estos conocimientos, pero que por el contrario sí que tiene una gran idea con respecto a una aplicación iPhone/iPad. Lewis decidió poner en marcha esta clase. Se dió cuenta que podría ser una Buena idea el usar su propias notas para la primera parte del curso, y luego explorar y profundizar en los recursos que esta plataforma nos ofrece.

Siguió adelante con esta idea. Lewis estaba muy impresionado con *Beginning iPhone 3 Development: Exploring the iPhone SDK*. Este best-seller era un libro de instrucciones de la Editorial Apress que fue escrito por Dave Mark and Jeff Lamarche. Lewis se convenció de que su libro podría proporcionar a sus lectores conocimientos y un alto nivel para programar fluidamente en todos los dispositivos multitouch de Apple.

El curso del Dr. Lewis fue un auténtico éxito, y después de una posterior conversación con un representante de Apress, Lewis mencionó que sólo había empezado a usar el libro a mitad de semestre, ya que tenía que dar a los estudiantes no provenientes de la carrera de ingeniería la base necesaria. El editor sugirió convertir estas notas y esbozos en un primer libro –un libro introductoria dedicando a gente sin conocimientos o de conocimientos muy bajos. En este momento, era sólo cuestión de tiempo y detalles –como la organización y la revisión de los vídeos instruccionales para ponerlos a disposición de otros no-ingenieros que estaban ilusionados en programar sus propias aplicaciones para iPhone y/o iPad.

Esta es la historia de cómo un profesor chiflado llegó a escribir este libro. Esperamos que te lleves este libro a casa y comiences. *Ármate con estos conocimientos y empieza ahora mismo a cambiar tu vida!*

Ben Easton
Autor, Profesor,, Editor

Acerca de Los Autores Ayudantes

Ben Easton, graduado en la Universidad de Washington y Lee. Posee una Licenciatura en Filosofía. Sus aficiones son la música, banca, navegación a vela, ala delta… La mayoría de su trabajo ha girado, de una manera u otra, en torno a la educación. Ben impartió clases durante 17 años, dedicando la mayoría de ellos a clases de matemáticas.

Recientemente su experiencia como implementador y probador de software despertó su afinidad por los temas técnicos. Escritor freelance, ha escrito varias historias de ciencia ficción y guiones de cine, así como diversos artículos en revistas y boletines. Reside en Austin, Texas, y en estos momentos está trabajando en su primera novela.

Kyle Roucis es un estudiante de la Universidad de Colorado en Colorado Springs, donde busca licenciarse en Ciencias de Computación y en Diseñó y Desarrollo de Juegos. Kyle fué el ayudante de el Dr. Lewis para la CS201, clase que probó todas las aplicaciones y metodologías presentadas en este libro. Kyle evaluó los códigos de los estudiantes de las lecciones de este libro y contribuyó con muchas grandes ideas, la cuales hicieron incluso que el Dr. Lewis cambiase incluso la forma de presentar ciertos temas. Kyle ha estado desarrollando aplicaciones para el iPhone y el iPod Touch desde que se presentó el primer SDK en Junio de 2007. La mayoría de su trabajo se ha basado en el iPhone. Kyle vive en Colorado Springs y espera crear su propio estudio de desarrollo de juegos con énfasis en programación de juegos para el iPhone, iPad y Mac.

Acerca del Revisor Técnico

Kristian Besley es programador y desarrollador de páginas web. Actualmente trabaja en educación y está especializado en juegos, interactividad y contextos dinámicos, principalmente Flash, PHP y .NET. También imparte clases de medios interactivos. Kristian ha trabajado como productor freelance para numerosos clientes, entre los que se incluyen la BBC, JISC, Welsh, Assembly Government, Pearson Education, y BBC Cymru.

Ha escrito varios libros para los Friends of ED, tales como la colección *Foundation Flash* series, *Flash MX Video*, *Flash ActionScript for Flash 8*, y *Learn Programming with Flash MX*. También ha sido colaborador de los increíbles libros *Flash Math Creativity* y ha escrito para la revista *Computer Arts.*

Kristian actualmente vive con su familia en Swansea, Gales, y posee un Galés bastante fluido.

Agradecimientos

Cuando llegué a América en 1981 con 20 años, no tenía experiencia, ni dinero, ni siguiera sabía cómo se utilizaban las cabinas de teléfonos americanas. Desde entonces he seguido un precioso camino hasta la concepción de este libro y mi empleo como Profesor Asistente en dos Universidades de Colorado. Me considero una persona afortunada, ya que he conocido a mucha gente maravillosa.

Primero, a mi mujer, Kera, que movió montañas para ayudarme con gráficos, comidas, dictados, animándome a seguir trabajando y manteniendo niveles sanitarios en nuestra casa. Gracias, Kera.

A mi madre, Adeline, que siempre me animó, incluso en los momentos más difíciles cuando casi dejo los estudios de Ingeniería Eléctrica. A mi hermana, Vici, que me mantiene con los pies en el suelo, y a mi difunto hermano Murray, un constante recordatorio de lo bonita y preciosa que es la vida. A Ketih y Nettie Lewis, quienes me enseñaron a utilizar las cabinas telefónicas de América. A Ben Easton, Brian Bucci y Dennos Donahue, quienes me acogieron en sus familias cuando no tenía a nadie.

Especial agradecimiento al Dr. Zbigniew Ras, mi tutor de doctorado, quién incluso llegó a ser un padre para mí, y al Dr. Terry Boult, mi mentor y socio en el programa de Licenciatura en Innovación en la UCCS.

Por ultimo pero no menos importante, a Clay Andres de Apress- quién me ayudó en todo el proceso de elaboración del libro y arriesgó su reputación convenciendo a un grupo de personas realmente inteligentes de que yo podía realmente escribir un libro como este.

Muchas gracias a todos vosotros!.

Prefacio

Lo Que *Este* Libro Hará Por *Usted*

Quieres aprender cómo programar para el iPad o iPhone, y seguro que te consideras una persona inteligente -pero cuando te has encontrado con líneas de código o instrucciones demasiado técnicas, has sentido que tu cabeza estaba a punto de estallar. ¿Se te ponen los ojos vidriosos cuando sigues instrucciones complicadas? Oyes una vocecita en tu cabeza que te susurra, *"Pero a dónde vas! el cerebro se quedó seis líneas atrás, pero incomprensiblemente sigues escaneando la página—creyendo que no eres tan lento cómo piensas. Genial!"*

A ver si esto te suena familiar... tienes una duda con algo bastante técnico y decides recurrir a Google para buscar la solución del problema. Pinchas sobre un enlace y alguien ha hecho exactamente la misma pregunta! Te emocionas mientras que la página carga, pero, por desgracia se trata de un foro, un chat donde geeks, gente con experiencia, habla con cierta base y postean inteligibles códigos). Localizas la pregunta que buscabas, que no es más que la duda que tienes, y esa pregunta va seguida de… demasiado tarde! Tu cerebro ha echado el cierre y te sientes frustrado y en tensión, como si tuvieras nudos en el estómago.

Te suena familiar?

¿Sí? Entonces este libro va dirigido a tí! Te imagino en una tienda de libros o en el aeropuerto, echando un vistazo a una revista, buscando algo de interés. Debido a lo acotado de este tema, imagino que podrás permitirte un iPhone, un Mac un coche y billetes de avión. Probablemente estés interesado en la floreciente industria de los dispositivos handhelds y en el tremendo ritmo de evolución de las memorias y microprocesadores… en cómo ideas fugaces pueden convertirse en plataformas de computación totalmente nuevas , en aplicaciones de software potentes, en herramientas útiles y juegos de inteligencia… quizá incluso hasta en la pasta! Te estarás preguntando si puedes entrar en acción haciendo uso de tu intelecto y experiencia técnica para servir a las masas.

Cómo puedo saber todo esto sobre tí?

Fácil! Después de todos estos años en los que he enseñado a estudiantes programación, he aprendido que si estás leyendo esto es porque eres lo suficientemente inteligente y lanzado para pisar el terreno de juego de la programación, y en especial para un dispositivo tan interesante como el iPhone y el iPad. Si te sientes identificado con la persona que he descrito líneas arriba, entonces te conozco. Nos presentaron el uno al otro hace ya largo tiempo.

Eres una persona inteligente pero que tiene "espasmos mentales" cuando lee un código complejo, incluso si tienes algunas nociones de programación. Incluso este es tu libro también en caso de que tengas una buena base de conocimientos en otros lenguajes de programación, pero lo que buscas es aprender de una manera sencilla a programar par el iPhone y el iPad. Puedo guiarte a través de cualquier atasco mental, ya que tengo bastante experiencia en guiar a las personas a través de obstáculos técnicos, ya sean reales o imaginarios. He hecho esto en miles de veces con mis estudiantes y mi metodología también funcionará contigo.

El Enfoque Que Utilizo

No intento explicar todo con detalle. Tampoco espero que comprendas cada línea de código. Lo único que haré es mostrarte, paso a paso, como llevar a cabo acciones clave. Mi enfoque es comprensivo y fácil de seguir, y me enorgullece la habilidad que poseo para enseñar a estudiantes y gente interesada un amplio espectro de conocimientos y habilidades.

Esencialmente, te guiaré, a tu propio ritmo, hasta el objetivo establecido de aprender a codificar, subir, y quizás incluso comercializar tu primera aplicación iPhone/iPad, ya sea simple o compleja. *Buenas noticias*: la mayoría de las aplicaciones descargadas no son nada complejas. Las más populares son simples herramientas de uso cotidiano y sentido común… encontrar tu coche en un parking, elaborar mejores listas de la compra, o hacer un seguimiento de tu evolución física. Sin embargo, cuando completes este libro, quizás quieras derivarte a otros libros de la editorial Apress y Friends of ED. Tienes un amplio abanico de posibilidades, y por el camino te iré guiando en el mejor modo de avanzar en tus conocimientos.

Quizás quieras leer un poco sobre mí para que te sientas seguro de la decisión de tomarme como guía en esta emocionante “app-ventura”.

Podrás divertirte y prosperar en este asombroso y mágico mundo.

Paz!

Rory A. Lewis, Doctor, JD

Capítulo 1

Antes De Empezar…

En este capítulo introductorio comprobaremos que tenemos todas las herramientas y accesorios necesarios para trabajar. Algunos de los lectores estaréis convencidos de que tenéis todo lo necesario, por lo que podéis pasar directamente al Capítulo 2 y empezar inmediatamente a trabajar con vuestro primer programa.

En cualquier caso, este capítulo es interesante para entender el por qué explico ciertas cosas y otras pasar de alto. Para aquellos que nunca habéis programado en Objetive –C esto será un reto –incluso para mis estudiantes de ingeniería que tienen conocimientos de Java, C y C#. Sin embargo, con una preparación adecuada y una mentalidad positiva, lo lograreis.

Os animo a que sigáis leyendo este capitulo. El tiempo que invertiréis en ello estará bien aprovechado, ya que nos dará confianza para la tarea que vamos a comentar. El Capítulo 1 ayudará a estructurar la forma en que tu cerebro asimila todos los contenidos que vamos a ir viendo.

Necesidades y Accesorios

A fin de programar para el iPhone y / o IPAD, y seguir con los ejercicios, tutoriales y ejemplos presentados en este libro, tendrás que prestar atención a unos requisitos mínimos:

- Macintosh basado en Intel con Leopard (OS X 10.5.3 o posterior ...)
  - Si el Mac es posterior a 2006, el equipo es suficientemente potente.
  - No necesitará el último modelo de Mac. En caso de que todavía no dispongas de uno, te sugiero un MacBook básico, sin lujos.
  - Si eres dueño de un Mac antiguo, prueba a añadir algo de RAM.
- Registrarse como desarrollador a través del iPhone / IPAD Software Development Kit (SDK).
  - Si eres estudiante, sería interesante que tu profesor esté al tanto de ello y puedas registrarte bajo el nombre de tu profesor.
  - En caso no seas estudiante, necesitas seguir los siguientes pasos para registrarte.

1. Carga la siguiente página: http://developer.apple.com/programs/iphone/, Obtendrás una página similar a la mostrada en la Figura 1-1. Selecciona el botón “Enrrol Now”.

Figura 1–1. Pincha sobre el botón “Enroll Now”.

2. Pincha el botón “Continue” (Ver Figura 1-2).

Foto 1–2. Selecciona el botón “Continue”..

3. La mayoría de vosotros eligireis la opción “I need to create a new account for …” (Crear nueva cuenta para Programa de Desarrollo Apple) (flecha 1 de Foto 1–3). A continuación, pincha sobre el botón “Continue” (Flecha 2 en Foto 1-3). En el caso que tengas registrada una cuenta, deberás escoger la opción “I currently have an Apple ID ...” (Tengo una cuenta Apple…) y pasar al punto 6, donde nos loguearemos y procederemos a descargar el SDK.)

Foto 1–3. Selecciona el botón “I need to create an Apple ID …”.

4. Lo más normal es que te des de alta como usuario individual, por lo que deberás ir a la parte izquierda de la pantalla y seleccionar el botón “Individual” (Ver Foto 1-4). Si te das de alta como Empresa, pincha sobre la opción “Company” a la derecha, sigue los pasos que se te marquen y pasa el punto 6.

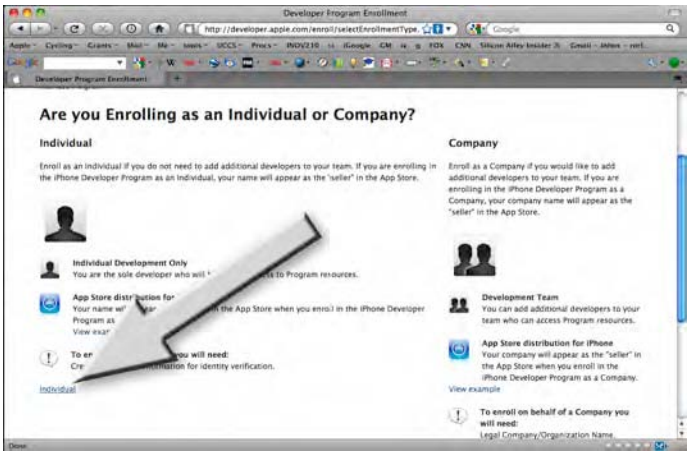


Foto 1–4. Selecciona la opción “Individual”..

5. Aquí introducirás toda tu información tal y como se puede ver en la Foto 1-5. Tendrás que abonar los 99 dólares del Programa Standard. Este paquete te proporcionará todas las herramientas, recursos y soporte técnico que necesitas (Si estás leyendo este libro, seguro que no querrás adquirir el Programa de Empresas, el cual vale 299 dólares y el cual está enfocado aplicaciones comerciales). Después de pagar, guarda tu Apple ID y tu nombre de usuario y la aplicación te mandará dicha confirmación de alta y datos a tu correo electrónico.

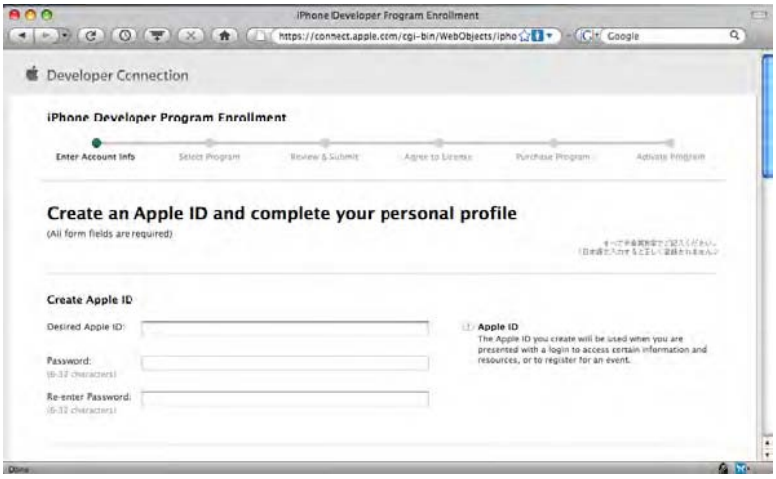


Foto 1–5. Introduce tus datos personales.

6. Utiliza tu Apple ID para acceder a la página principal de desarrollo iPhone/iPad. Desplázate al final de la página y descarga el SDK tal y como aparece en la Foto 1-6. Crea todos los iconos necesarios en tu escritorio. El SDK incluye un entorno de desarrollo integrado (IDE). Este entrono es una plataforma de programación que contiene un conjunto de herramientas, sub-aplicaciones y el código necesario que nos facilitará bastante el trabajo. Haremos gran uso del Xcode, Interface Builder, y del iPhone/iPad Simulator, por lo que sería conveniente que tuvieses estos accesos directos preparados en el escritorio y así tener un acceso más fácil.

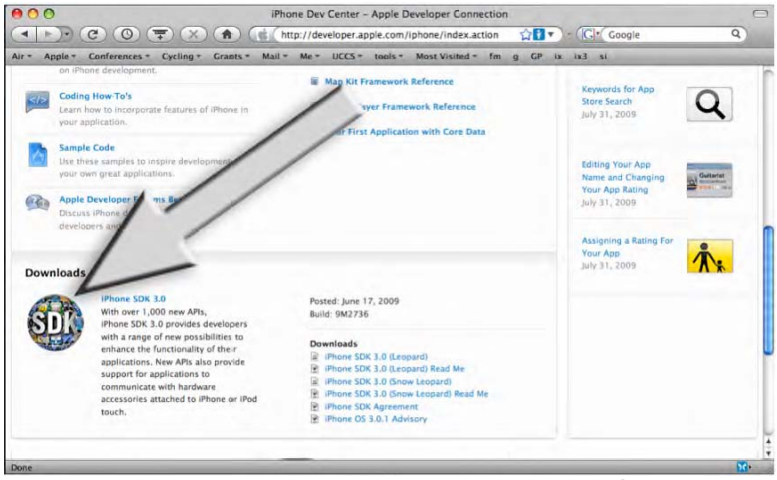


Figura 1–6. Una vez logueado como Desarrollador Registrado Apple, podrás acceder al final de la página y descargar el SDK.

7. Crea un acceso directo de Xcode en tu escritorio. Para ello accede a Macintosh HD ► Developer ► Applications ► Xcode.app y arrástralo a tu escritorio tal y como aparece en la Foto 1-7. Del mismo modo, crea otro acceso directo del Interface Builder seleccionando Macintosh HD ► Developer ► Applications ► Interface Builder.app y arrastrándolo al escritorio también. Por ultimo, crea otro acceso del iPhone/iPad Simulator en tu escritorio accediendo a Macintosh HD ► Developer ► Platforms ► iPhone/iPad Simulator Platform y arrastrándolo al mismo.

Foto 1–7. El Xcode, Interface Builder, y el iPhone/iPad Simulator—cerrados y cargados, preparados para funcionar!

NOTA: Siempre que haga referencia a "iPhone" "iPad," lo estaré hacienda por extensión para cualquier otro dispositivo iPhone o iPad, incluyendo el iPad Touch.

Qué es lo que no te enseñaré

Una vez instaladas las herramientas Interface Builder y el iPhone/iPad estamos preparados para empezar. Pero espera!. Primero debes saber hacia donde vamos.

Lo primero, déjame decirte hacia dónde NO vamos –aquello en lo que no nos vamos a centrar. No voy a intentar explicar cómo funciona o qué sentido tiene cada línea de programación. En lugar de ello daremos una aproximación al subsistema, indicando que partes o secciones de código te servirán en cada situación.

Si bien el objetivo de este libro es que aprendas, adquieras conocimientos globales y desarrolles habilidades, nos ocuparemos más de las moléculas que de los átomos o partículas subatómicas. Haremos hincapié en el reconocimiento de atributos generales, comportamientos y relaciones de código, de modo que no necesitará enredarse en minucias, símbolo por símbolo. Lo que haré es darte conocimientos para que puedas escoger aquellas áreas en las que te quieras especializar.

Ciencias de la Computación: Un paisaje amplio y diverso

Hagamos uso de una analogía: supongamos que el iPhone/iPad es un coche. La mayoría de nosotros conducimos de la misma manera que utilizamos el ordenador. Del mismo modo que no intentaría intentar explicar cómo funciona cada parte del coche si os estuviera enseñando a conducir, tampoco os voy a iniciar a la programación IPhone e IPad con conceptos de ingeniería informática desde el primer momento.

Incluso los grandes mecánicos que trabajan con coches todos los días raramente conocen los fundamentos físicos y electrónicos que hay detrás del motor de combustión, sin hacer mención a todos los sistemas auxiliares; ellos pueden conducir un coche, diagnosticar qué falla en el mismo, y usar sus herramientas y máquinas (incluyendo ordenadores) para repararlo óptimamente. Por similitud, grandes programadores que crean aplicaciones para el IPhone y el IPad raramente conocen los fundamentos de programación y los diseños de circutería de la plataforma Apple. Sin embargo, utilizan estas herramientas y tienen visión para vislumbrar un nuevo nicho en el amplio espectro de necesidades de aplicaciones , y pueden utilizar sus herramientas y aplicaciones –instaladas en los escritorios de sus ordenadores- para diseñar, codificar aplicaciones y difundirlas al mercado.

Siguiendo con esta analogía, la programación de iPhone o iPad es como jugar con el motor de tu coche- personalizando aquello que quieras que haga. Apple ha diseñado un motor de computación tan fantástico como lo es un motor V8. También ha facilitado un bonito chasis en el cual podemos modificar o reconstruir nuestro motor de computación. También existen restricciones sobre cómo podemos “tunear” nuestros coches iPhone/Ipad, y para aquellos que nunca hayan tuneado un coche, voy a demostrar cómo aprovechar al máximo las posibilidades creativas, al mismo tiempo que se respetan las mencionadas restricciones.

Voy a enseñarte, sin demasiado lujo de detalles, como hacer el cambio de aceite, ruedas, asientos y ventanas para convertirlo en un todo terreno, deportivo, o un coche pensado para la jungla. Cuando hayas dominado este libro, sabrás cómo enfocar una modificación del motor, la transmisión, el tren de tracción, la eficiencia del combustible o el sistema estéreo del coche.

Por Qué Existe El Purgatorio En Objetive-C

Asumo que nunca has trabajado en un coche, nunca te has manchado las manos de grasa y que quieres tunear uno de los coches más potentes del mundo –con un complejo motor V8. Voy a enseñarte a hacerlo y al mismo tiempo vamos a pasarlo bien.

Lo primero, necesitas aprender un poco sobre cómo puede llegar a trucarse un coche con motor V8, en este caso el IPad. En 1971, Steve Jobs y Steve Wozniak se conocieron y cinco años más tarde fundaron Apple, produciendo uno de los primeros ordenadores personales de éxito. En 1979, Jobs visitó Xerox PARC (Palo Alto Research Center), y consiguió una contrata con Xerox para un suministro de su nuevo proyecto llamado Lisa. Aunque el Alto no fue un producto comercial, fue el primer ordenador personal que utilizó un entorno de interfaz gráfica de usuario (GUI). Lisa fue el primer producto Apple con ratón y GUI.

A principios de 1985, Jobs perdió poder en el Consejo de Administración de Apple y salió de la compañía, pasando a fundar NeXT, que posteriormente sería comprada por Apple en 1997. Durante el tiempo que estuvo en NeXT, Steve Jobs cambió algunas de de las características fundamentales en el código Macintosh (Mac), en otras palabras, creó un basto pero hermoso lenguaje llamado Objective-C. El poder de este lenguaje de programación se basa en la habilidad y eficiencia en el uso de objetos. El cerebro de Jobs estaba en plena efervescencia creativa, y ese increíble código de lenguaje Objetive-C alcanzó altas cotas. Su inspiración pasó a las entrañas del Mac mediante la creación de un metalenguaje al que llamamos Cocoa. Un metalenguaje es un lenguaje usado para analizar o definir otro lenguaje. Objective-C sería la "bestia indomable", y Cocoa sería el lenguaje de domesticación de la bestia, o al menos , la jaula de la bestia.

Si eres un auténtico principiante en el mundo de la programación, no puedo esperar que entiendas entre las sutilezas y distinciones de los lenguajes de programación. Simplemente te estoy dando una visión histórica general sobre la cual basar tu experiencia. Lo que quiero con esto es que te quede claro que Objective-C y Cocoa son dos herramientas muy poderosas y ambas son relevantes para la Programación iPhone/iPad.

Houston, Tenemos Un Problema

Esta es la esencia del reto al que me intrigó, y condujo al diseño de mi curso original. ¿Cómo se puede enseñar a personas que no son estudiantes de ingeniería, tal vez como tú, algo que se le puede atragantar incluso a los mejores estudiantes de Ingeniería Informática? A nivel universitario, los estudiantes por regla general tienen nociones de introducción a la programación como C# ó C++.

Dicho esto, vamos profundizar en Objective-C! A veces te va a parecer lioso, otras veces lo asimilarás mejor. Habrá veces que tendrás que releer páginas o ver viedoejemplos varias veces, pero conseguirás asimilar conocimientos que de concepto son difíciles.

Sobre Cómo Visitaremos El Purgatorio De Vez En Cuando

Hay partes concretas de mis cursos en las que la mitad de la clase coge el concepto inmediatamente, un cuarto de ella tiene que repasar conceptos para hacerse con ellos y el último cuarto se agobian y lo desisten. Este tercer grupo deja la carrera de Ingeniería y opta por opciones más fáciles. Tengo identificadas esas partes del curso y no vamos a entrar en ellas. Repito, no las vamos a ver.

No te preocupes, no permitiré que metas la mano en un avispero (en lo relativo a Objetive-C) cuyo picotazo te lleve a la muerte. Tampoco voy a enseñarte aquellos conceptos que puedas encontrar complicados. No voy a explicar esto ahora. Simplemente asúmelos! Si te relajas y sigues mis ejemplos, podrás asimilar los conceptos de este libro con gran éxito.

Cuando te encuentres en uno de esos momentos difíciles, tómatelo con calma. Siempre puedes releer la sección o ver los video ejemplos. En la aventura de la programación IPhone/IPad no encontrarás conceptos que te hagan abandonar. Hay solo unas tres áreas críticas en el libro, las cuales las he redactado del modo más fácil que me ha sido posible. También hay blogs y foros de discusión donde puedes recurrir para aclarar problemas y compartir conocimientos con otras personas.

Mirando Hacia El Futuro… *Inicio del Desarrollo iPhone: Reflexiones para el SDK iPhone*

Algunos de vosotros querréis continuar haciendo pinitos en la programación para iPhone y iPad leyendo el libro de Dave Mark y Jeff Lamarche, B*eginning iPhone 3 Development* (Apress, 2009). Recuerda la analogía hecha en el capítulo anterior, en la que puede darse el caso de que queramos convertirnos en mecánicos de un automóvil con un complejo motor V8 montado en un chasis simple. Este libro asume que los lectores tienen conceptos sobre lo que es un carburador, un pistón y de que saben como montar ruedas de competición y llantas en los coches tuneados de sus amigos.

En otras palabras, en ese libro los autores asumen que entiendes los fundamentos de la programación orientada a objetos: que conoces qué objetos y variables hay, y que estás familiarizado con el lenguaje de programación Objective-C.

A diferencia de esto, yo asumo que el lector no sabe por ejemplo, cuáles son los conceptos “class”, “member o “void”. También asumo que no sabes cómo funciona el sistema de gestión de memoria en un iPhone/iPad, y lo que es más, de que nunca has tenido interés –hasta ahora-, en comprender lo que es una matriz o un SDK.

Qué Vamos A Aprender

He advertido que cuando los estudiantes se enfrentan a una clase difícil, esta se hace más sencilla si se le da una aplicación práctica y esta se crea con facilidad. En cada etapa de este proceso, pondré un ejemplo que puedas leer, ver, y asimilar con facilidad. Más tarde, volveremos a analizar parte de estos primeros pasos y entraremos en ellos con más detalle. Voy a explicar cómo llevar a cabo alguna tarea o acción por primera vez *sin ni siquiera comprenderla*. Luego, comparando la primera vez que pasamos por ello con modificaciones posteriores, conseguiremos aprender a ajustar el programa un poquito por aquí y un poquito por allá. De esta manera estaremos en el camino del aprendizaje –motivados e inspirados para absorber nuevas enseñanzas, métodos y trucos..

Aplicaciones Frescas y Entretenidas: Por Qué Enseño De Esta Manera

Habrás oído muchas veces cuál es el mejor método para recordar cosas: Hacerlas es mejor que verlas y escucharlas. Sé de sobra que a los estudiantes les gusta el humor – y adivinad que! Las personas recordamos historias divertidas y lecciones amenas mucho mejor que clases sosas y aburridas. Cuando los estudiantes trabajan en un código que es divertido y alocado, dedican mucho más tiempo y esfuerzo en el mismo.

Cuanta más atención mental empleamos en la solución de un problema, las conexiones neuronales de nuestro cerebro son más eficientes y numerosas. Cuantas más conexiones apliquemos, mayor facilidad tendremos para recordar conceptos –y lo más importante- desperdiciaremos un menor tiempo en métodos inefectivos.

Cuanto más tiempo empleemos en un tema concreto, más elementos de juicio tendremos para comprobar si una metodología de enseñanza es correcta o no. Verás que a medida que avancemos iremos empleando el humor para asimilar conceptos de computación y conceptos y métodos Objective-C sin que tengamos que hacer ningún esfuerzo consciente.

Es común que mis estudiantes contacten conmigo después de que les ponga algún trabajo difícil para casa. Primero me envían un mensaje preguntando si pueden contactar por Skype. Una noche en particular, estaba jugando al ajedrez con un amigo cuando recibí un mensaje preguntándome si estaba disponible. “Por supuesto”,

respondí.. Advertí a mi colega, quién también es profesor en la Universidad de North Carolina, de que algunos estudiantes iban a conectarse a Skype. Al conectarse, aparecieron cuatro de mis estudiantes de Ingeniería Electrónica, con los ojos bien abiertos y sonriendo. "*Hey, Dr. Lewis, finalmente lo conseguimos, pero amigo! Gracias al ultimo método con el que nos ha enseñado*."

Cuando terminamos la conversación, apagué mi Mac, eran las 12:30. Mi colega me preguntó, "Rory, yo nunca he llamado a un profesor a estas horas de la noche, y mucho menos después de media noche! ¿No deberían de plantear las dudas en horario de clases?!" Probablemente tuviese razón, pero después de pensar en ello durante un momento, le respondí "Soy feliz porque sé que están trabajando en lo que les he encomendado" Conforme íbamos preparando la siguiente partida de ajedrez, murmuró algo acerca de cómo podría sentirme a gusto con toda esa locura.

El tema está en que quiero que leas este libro entero. Quiero que trabajes en los ejemplos y que sientas euforia cada vez que termines una tarea. He hecho todo lo posible para hacer de esto un libro divertido. Si optas por seguir la estructura del mismo y comprometerte en el trabajo, este libro cambiará tu vida!.

Por cierto, ir avanzando con el libro te hará coger caché. La gente más allegada verá que tienes capacidad de crecimiento y será testigo de su transformación, y como resultado, le pedirán que les hagas aplicaciones para ellos.

Evangelizar a Tu Abuela… Lo que has programado es Crucial!

Es importante que el código complejo no te desanime. Hace exactamente dos minutos, un estudiante ha entrado a mi despacho –tan confuso que incluso no sabía transmitirme lo que no había entendido. Me ha dicho algo tipo "Mi matriz de Segundo orden trabaja bien en línea, pero no cuando lo hace como clase o método". Yo le he respondido "No, eso es demasiado complejo! Esta es una manera más fácil de decirlo…".

"Wow! Ya sé qué he hecho mal, Dr. Lewis. Gracias!" Ahora escribo esto, él se lo está explicando a sus dos compañeros que vinieron ayer e intentaron hacerme la misma pregunta. No te preocupes los conceptos que aparecen en estas preguntas –tales como "classes" y "methods," y otras entradas de códigos las veremos en este libro. ¡Cada cosa a su tiempo!

Si eres capaz de mantener los pies en el suelo y transformar conceptos complejos en ideas simples, podrás recordar las mías y dominarlas. Asimila este concepto y serás capaz de convertir tus ideas en código fuente – y quién sabe dónde llegarás! Por ello, estoy decidido a transmitirte la habilidad de convertir aquello que tu abuela en lenguaje de programación para IPhone e Ipad.

Cómo Funciona Todo Esto?

Antes de empezar nuestro primer programa en el Capítulo 2, es muy importante que estés dispuesto a volver atrás y saber qué hemos visto, dónde estamos ahora y dónde vamos a ir después. Mirando la Foto 1-8, puedes ver una sección gris que contiene dos iconos que representan el Mac OSX y el SDK, que incluye el Interface Builder y Xcode. Estos se explicarán en detalle más adelante, el 90% de este libro va tratando cada tema en su momento.

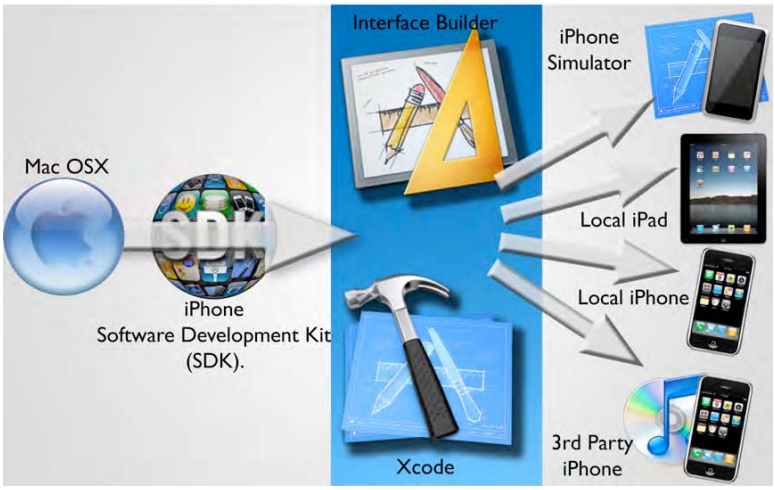


Foto 1–8. Esquema de aplicaciones de programación para iPhone e iPad. El SDK de IPhone e IPad está instalado sobre el SO Mac OS X 10.5.3 o posterior. Te enseñaré como usar el SDK's Xcode y el Interface Builder para crear aplicaciones. Una vez que crees una aplicación, hay cuatro formas de ejecutarlas: usando el Simulador IPhone/IPad, con tu iPhone conectado a tu Mac, con el IPad conectado al Mac, o descargando vía iTunes hacia otros iPhone/iPad.

La banda azul del medio es donde estamos en este momento. A la izquierda de esta área es dónde has estado. Tienes un Mac (comprado después de 2006 y un SO Mac OS X 10.5.3 o superior), y ya hemos completado el proceso de descarga del SDK para iPhone e iPad SDK (Fotos de la 1-1 a la 1–6). También hemos creado accesos directos al Interface Builder, Xcode, y al iPhone/iPad Simulator y los hemos colocado en nuestro escritorio (Foto 1–7). Ahí es donde estamos en este momento.

En el Capítulo 2, empezaremos a utilizar el Xcode y el Interface Builder para convertirte en un programador! Vamos a ejecutar todos los programas que hagamos compilándolos en una de las varias posibles opciones, las cuales están representadas a la derecha de la banda azul. La primera opción será el iPhone/iPad Simulator. La segunda será el tu iPhone y/o iPad conectados. Por ultimo, podemos usar iTunes para enviar tu Aplicación IPhone y/o Ipad a la App Store, donde la gente podrá comprarla o descargarla de un modo gratuito. Ahí es donde llegaremos.

Los dos objetos centrales de la Foto 1-8 son, como sabes, donde nos vamos a centrar en la mayor parte de tiempo en este libro. Usaremos Xcode para escribir en código, tal y como los grandes programadores hacen. Te enseñaré cómo funcionan todas sus funciones tales como la gestión de archivos, compilación, depuración y reporte de errores. El Interface Builder es la mejor forma que tiene Apple de permitirnos hacer drag and drop con objetos en nuestras aplicaciones iPhone/iPad. Si quieres un botón, por ejemplo, simplemente arrástralo y suéltalo donde quieres que quede en tu iPhone o iPad virtual.

Básicamente, utilizaremos Xcode para gestionar, escribir, ejecutar y repasar nuestra aplicación, para crear el contenido y la funcionalidad. Usaremos el Interface Builder para arrastrar y soltar elementos en nuestro interface hasta que la aplicación tenga el aspecto que queramos darle.

Después de haber integrado todos los objetos de interfaz con el código escrito en Xcode, podremos avanzar y ajustar los parámetros de acuerdo con la gestión y eficiencia de memoria, pero esto ya es avanzar ya mucho en nuestra historia.

Nuestra Hoja de Ruta: Uso del Xcode e Interface Builder

A menudo, diversos autores de libros de programación utilizan los mismos métodos anticuados. Primero presentan una simple aplicación “Hello Word”, para posteriormente pisar el acelerador e introducir al lector en códigos avanzados, lo que hace cundir el desanimo y que muchos lectores decidan dejarlo. Usando Objective-C (usado bajo Cocoa) junto con el SDK iPhone e iPad, he tenido que repensarme el proceso introductorio. Para ello he identificado tres problemas:

- Enseñarte a decir “Hello World” para posteriormente pasar a conceptos más avanzados y a las API, seria contraproducente.
- No tiene sentido elegir al azar una de las maneras de las que hay para decir “Hola al mundo” desde tu iPhone/iPad. Todas son necesarias tarde o temprano.
- Tratar de explicar una simple aplicación “Hello World” en Objective-C es algo más complicado de lo que parece y para lo que un usuario principiante está preparado, a no ser de que este proceso se vea por etapas o capas.

Mi solución para estos tres problemas es simple. Te enseñaré decir Hola al mundo desde tu iPhone/iPad no de una manera, ni de dos, sino de varias maneras diferentes. En cada una de ellas profundizaremos cada vez más.

Cada vez que viajes por el reino del Xcode, se te preguntará que tipo de vehículo quieres conducir. Un Jeep? Un coche de carreras? Un descapotable? Centrándonos en lo esencial, voy a enseñaros como “conducir” en Xcode. El objetivo es tener los suficientes conocimientos y confianza para tener acceso a cualquiera de estos vehículos. Por tanto, echemos un vistazo a lo que nos pueden ofrecer estos distintos vehículos. Por favor, sígame.

Preparándonos para nuestro primer proyecto iPhone/iPad

Dando por descontado que ya has descargado el SDK e instalado el Interface Builder, Xcode, y el Simulador iPhone/iPad, enciende tu Mac y pincha sobre el icono Xcode de tu escritorio. Tu pantalla debería ser muy parecida a la representada en la Foto 1–9. Arriba aparece una ventana de bienvenida a Xcode, incluyendo todos tus recursos para IPhone e IPad.

Foto 1–9. Después de hacer doble clic sobre el icono Xcode, verás la pantalla de “Bienvenido a Xcode”. Mantén la opción “Show at Launch” marcada.

Mira en la esquina inferior izquierda de la pantalla y asegúrate de que la opción Show at Launch permanece marcada. Aquí dispones de muchos recursos valiosos, los cuales tendrás siempre a mano. Te sugiero que después de que hayas terminado de leer el Capitulo 4, te tomes un poco de tiempo y explores estos recursos. Esta práctica te abrirá las puertas de la creatividad.

Sin empezar todavía un Nuevo proyecto, demos un paseo y comprobemos cuantos modelos debemos de "conducir". Para abrir un nuevo proyecto en Xcode, pulse simultáneamente Command + Shift + N . Esta combinación de tres teclas, representadas en la Fotografía 1-10 como (⌘⇧N), abrirá una ventana que muestra los distintos tipos de vehículos que puedes conducir en el mundo de Xcode..

La Foto 1–10 muestra 6 tipos de vehículos: Aplicaciones Navigation-based Application, Open GL ES Application, Tab Bar Application, Utility Application, View-based Application, y Window-based Application.

En los comienzos, la mayor parte de nuestros viajes en Xcode los haremos en los dos últimos estilos enumerados. Volviendo a hablar en términos de computación, View-based Application y Window-based Application son las estructuras que utilizaremos en el desarrollo básico.

No te preocupes, no he olvidado nuestra meta de crear una simple aplicación "Hello World". Te enseñare a Saludar al Mundo usando las seis opciones, y poco a poco te irás familiarizando con ellas.

Foto 1–10. Cuando escogemos un template para un Nuevo proyecto, tendrás la opción de escoger entre 6 estilos diferentes.

Los Screencasts Adicionales

Todas las fotos que aparecen en este libro han sido capturadas de mi pantalla y puedes consultarlas en un screencast. Por ejemplo, el ejemplo helloWorld_001 del Capítulo 2, puedes encontrarlo en:
`http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/002_helloWorld_ 002.htm`.

No es necesario repasar todo el screencast, ya que he incluido todas las instrucciones en el Capitulo 2. Sin embargo, he oído a estudiantes decir que algunas veces es divertido volver sobre lecciones que han escuchado. Estos videoejemplos están bastante condensados. Si deseas seguir aprendiendo y apoyándose en los screencast, tenga en cuenta estas recomendaciones:

- Para el video cuando vaya por delante tuya. Rebobina y vuelve atrás.
- Una vez que puedas completar un proyecto en su totalidad, guarda el screencast en otro directorio. A continuación vuelve a hacerlo con menos paradas hasta que lo domines … y compiles.
- Para plantearse retos, quizás un objetivo pueda ser que el ejecutes el código en el mismo tiempo que he empleado yo. En todo caso, como quiero que te sientas satisfechos con la programación a alto nivel, así que no lo tomes en el sentido estricto. Te recomiendo practicar todos los ejemplos en el libro.

Los Archivos PDF Incluidos

También incluyo una versión en PDF de las diapositivas que uso con mis estudiantes de la Universidad de Colorado. Estos archivos PDF –los cuales no son fundamentales pero sí suplementarios- muestran todas las diapositivas a partir de este capítulo. También hay enlaces para aquellos de vosotros que queráis profundizar en cualquier material que no esté recogido en este libro.

NOTA: Puedes acceder a videos y material adicional en el sitio www.apress.com.

Fingiendo no saber: El Arte de la De-Ofuscación

Antes de entrar en materia, me gustaría reiterar que voy a enseñarte cómo programar sabiendo únicamente lo esencial. A medida que avancemos, explicaré conceptos un poco más profundos, aunque no llegaremos a los mismos hasta que tengamos bien claros los conceptos básicos. Esta es una nueva forma de enseñar y he tenido un gran éxito con ella.

Probablemente pensarás que estoy completamente loco, pero aún así te pido que sigas mis instrucciones. Si te surge alguna duda de la que no encuentres solución en ese momento en el libro, créeme si te digo que no es importante en ese momento. ¡ Vamos paso a paso!.

Vamos paso a paso…

Este libro es muy detallado, y aunque incluso pongo a vuestra disposición video tutoriales para los ejercicios, no necesitarás hacer uso de ellos. Puedes leer únicamente el libro, sin conexión a internet, y todo lo que necesites lo podrás encontrar en estas páginas.

Ahora que hemos terminado de comprobar tus parámetros de sistema, nos hemos dado de alta como Desarrollador Apple, hemos descargado el SDK, descomprimido las herramientas esenciales y configurado nuestro escritorio, ya estamos preparados para pasar al Capítulo dos y empezar a crear código.

Pasa la página!

Capítulo 2

Despegue!

El primer programa que vamos a intentar, tal y como mencionamos en el Capítulo 1, será un básico y genérico programa tipo “Hello World”. Debo aclarar, sin embargo, que como todo lo concerniente en el contexto Objective-C, no necesariamente será sencillo.

Nuestra primera aventura con este conjunto de herramientas será decir “Hello” al mundo desde la aplicación template View-based en Xcode. Con posterioridad, podremos decir “Hello” al mundo desde la aplicación template Navigation-based, primero de un modo muy básico y posteriormente con varias modificaciones.

Además de la información presente en este libro, la que incluye varios pantallazos, también te facilito una serie de screencasts, los cuales podrás encontrar en mi página web. En este capítulo trabajaremos tres ejemplos, los cuales puedes encontrar en los siguientes enlaces:

http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/001_helloWorld_001.htm
http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/002_helloWorld_002.htm
http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/003_helloWorld_003.htm

1. Antes de abrir Xcode, primero cierra todos los programas, de modo que puedas aprovechar al máximo la capacidad de tu equipo. Presiona Command + Tab y luego Command + Q para cerrarlo todo hasta que únicamente quede el Finder en tu pantalla. Localiza el icono Xcode en tu Escritorio y ábrelo. A continuación aparecerá la pantalla de Bienvenida vista en el capítulo 1. Ver Foto 2-1.

Foto 2–1. Clica sobre el icono Xcode de tu escritorio para arrancar la aplicación. Te aparecerá la Pantalla de Bienvenida a la que hacíamos referencia en el Capitulo 1.

2. Abre un nuevo proyecto en Xcode. Hay dos maneras de hacerlo: usando combinaciones de teclado o mediante el uso del ratón. Te recomiendo encarecidamente que uses las combinaciones de teclado, ya que te ahorrará tiempo y parecerás un auténtico Profesional. Ten en cuenta que la mejor manera de no conseguir trabajo como desarrollador de aplicaciones IPhone e iPad es la de usar el ratón en aquellas funciones que pueden hacerse mediante combinación de teclas. Usando el teclado, pulsa simultáneamente Command + Shift + N. Esta combinación de tres teclas aparece en la Foto 2–2 como ⌘⇧N. (Usando el ratón deberás abrir un Nuevo Proyecto accediendo así: File ► New Project.) Ahora la pantalla debería de mostrar el Nuevo Proyecto tal y como se aprecia en la Foto 2–2.

Foto 2–2. Selecciona el template View-based Application para crear un Nuevo Proyecto.

NOTA: Mi icono template View-based Application apareció marcado por defecto, el tuyo puede que no. Independientemente de esto, pincha sobre él y guarda el Nuevo proyecto en tu escritorio como helloWorld_001.

3. Tan pronto hayas guardado el proyecto en tu escritorio, Xcode inicia el archivo `helloWorld_001` tal y como podrás comprobar en la parte superior de la pantalla (ver Foto 2–3). Si todo esto te parece complicado, tranquilo y no te pongas nervioso. Esta es la manera que tiene Apple de organizar todo lo que usaremos para montar complejas aplicaciones. Por ahora sólo tienes que seguir adelante y dejar de lado todas las preguntas que se te puedan presentar. Tal y como se muestra en la Foto 2-3, haz doble clic para abrir la carpeta Classes (No te preguntes todavía qué cuál es la función de la carpeta “Classes”. Simplemente abre la carpeta y en su debido tiempo sabrás todo lo necesario acerca de esta carpeta.)

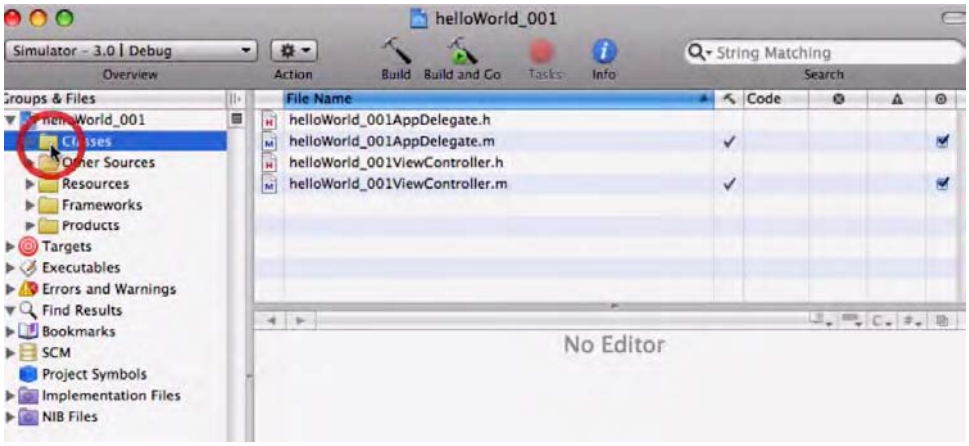


Foto 2–3. Clica el icono indicado para abrir el directorio Classes.

4. Como puedes ver en la foto 2–4, Xcode ha creado cuatro archivos, cada uno empezando por un prefijo que es idéntico al nombre de tu proyecto, `helloWorld_001`. Ahora vamos a seleccionar lo que es el llamado archivo interface; el cual tiene el nombre de tu proyecto seguido de `ViewController.h`. Esto generará el archivo `helloWorld_001ViewController.h`. A su debido tiempo explicaré que significa la extensión .h, y los detalles de lo que en estos momentos estamos haciendo y comprobarás como tiene sentido. Siguiente paso!

5. En la Foto 2–4, advertirás que hay varias líneas de color verde; estas líneas son comentarios internos, ya sean de un codificador o un programador a otro. Esos comentarios son "invisibles" para nuestro Mac y dado que ahora enfocamos nuestro interés en el código con el que trabaja el equipo, prestaremos atención a la línea siguiente a estos comentarios:

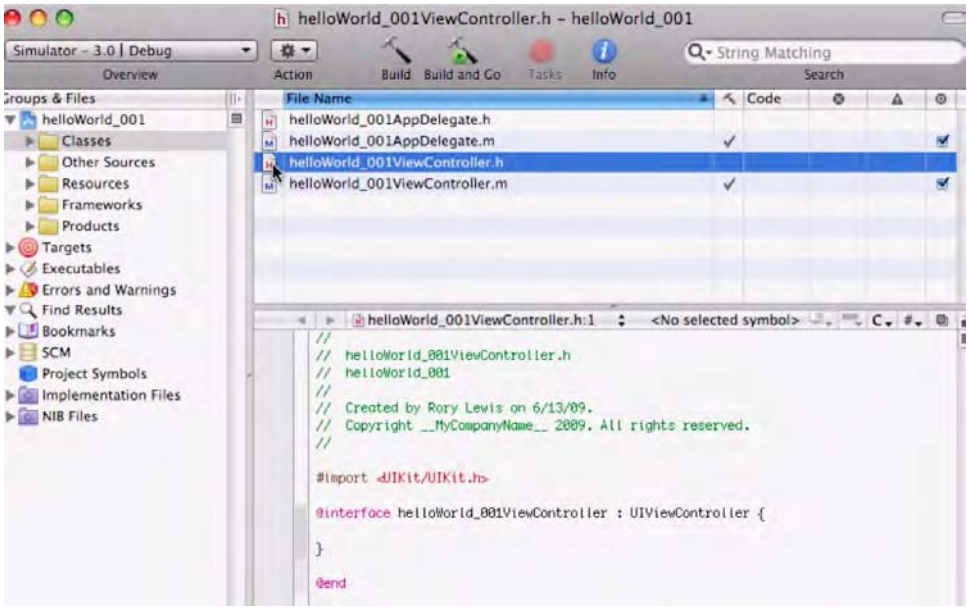


Foto 2–4. Abre el archivo. Así debe verse tu pantalla antes de escribir tus dos líneas de código.

```
#import <UIKit/UIKit.h>
```

Esta línea de código importa algo llamado UIKit framework. Esto facilita a tu aplicación elementos y combinaciones de teclas que harán más fácil el interface de usuario. Este comando nos acerca un código organizado en clases, una especie de paquetes de sub-rutinas preprogramadas. El uso del UIKit, nos evitará escribir o rescribir un montón de código y podremos aprovechar una serie de paquetes preprogramados para diseñar nuestra aplicación. Estas classes facilitan la presentación de los datos al usuario y la respuesta de entradas del usuario.

NOTA: Tenga en cuenta que haremos referencia al término "User Interface" con las iniciales UI.

Después de esta línea nos encontramos con:

```
@interface helloWorld_001ViewController : UIViewController {
```

Dos cosas a destacar. La primera, el símbolo @ es una orden de Xcode, la cual transforma nuestro código en una acción, y tiene algo esencia e importante que anunciar. Llamamos "directiva" a cualquier orden que comience con esa @. Esta @directiva indica a Xcode que tenemos algo que decir concerniente a "helloWorld_001," incluyendo aquello que vamos a decir encorsetado entre par'enteis {}.
Mira a tu pantalla, a la derecha, a continuación del par'entesis inicial, {. No hay nada! Todavía no le hemos marcado nada —verdad? Sabemos lo que queremos hacer, y es decir "Hello" al mundo des de nuestro dispositivo iPhone/iPad. Ahora que nuestra atención está centrada en el compilador, en concreto en la directiva @interface, queremos indicar que primero haremos uso del Interface Builder, IBOutlet, para decir algo … y que ese algo ser'a escrito UILabel.

NOTA: En lo sucesivo, representaremos el t'ermino "Interface Builder" con las iniciales IB.

Para ello, vas a incluir la siguiente línea de comando después del primero de los paréntesis:

```
IBOutlet UILabel *label;
```

No olvides incluir el punto y coma ";" al final de la línea. Este símbolo le dice al compilador que has terminado de hablar por el momento. Y no te preocupes acerca del símbolo * que aparece justo antes de la palabra "label." Iremos a ello más tarde.
Tu código debería quedar como la línea que os pongo a continuación. También puedes ver la Foto 2-5.

```
//
// helloWorld_001ViewController.h
// helloWorld_001
//
// Created by Rory Lewis on 6/13/09.
// Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface helloWorld_001ViewController : UIViewController {
        IBOutlet UILabel *label;
}
@end
```

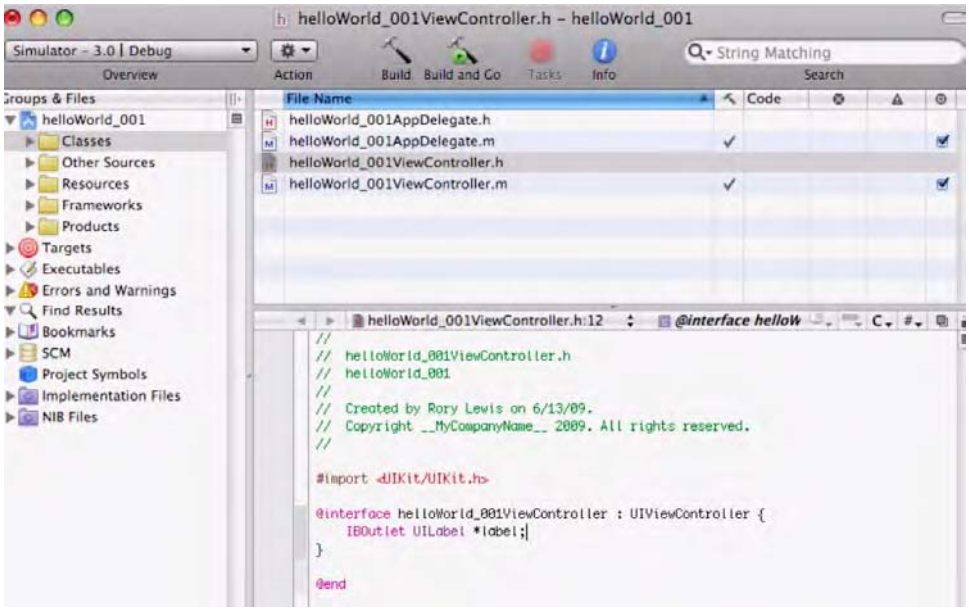


Foto 2–5. Aquí puedes apreciar cómo queda tu archivo después de haber incluido el código.

NOTA: Si quieres saber más acerca de detalles técnicos busca el slideshow pertinente en mi web `http://www.rorylewis.com`.

6. Ahora que hemos introducido nuestra primera línea de código, necesitamos incluir en el compilador una cosa más. En concreto, queremos incluir un botón en nuestro iPhone o iPad que Apple ya ha codificado para nosotros. Vamos a añadir una `IBAction`, la cual hará referencia al "hello." Para ello, incluiremos la siguiente línea después del paréntesis cerrante.

```
-(IBAction)hello:(id)sendr;
```

De nuevo, comprueba que has incluido las comillas al final de la línea. Tampoco te preocupes por el hyphen (el símbolo menos). Volveremos a él más tarde.

Ahora, marca la línea tal y como podrás ver en la Foto 2-6. Usando el teclado, presiona Command + C. Esta útil combinación de teclas aparece en la Foto 2-6 como ⌘C. Con ello se consigue copiar esta línea en el portapapeles (También puedes usar el mouse y acceder al comando Edit ► Copy.)
Nuestras instrucciones quedan así:

```
//
// helloWorld_OO1ViewController.h
// helloWorld_OO1
//
// Created by Rory Lewis on 6/13/09.
// Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface helloWorld_OO1ViewController : UIViewController { IBOutlet
UILabel *label;
}
-(IBAction)hello:(id)sendr;
@end
```

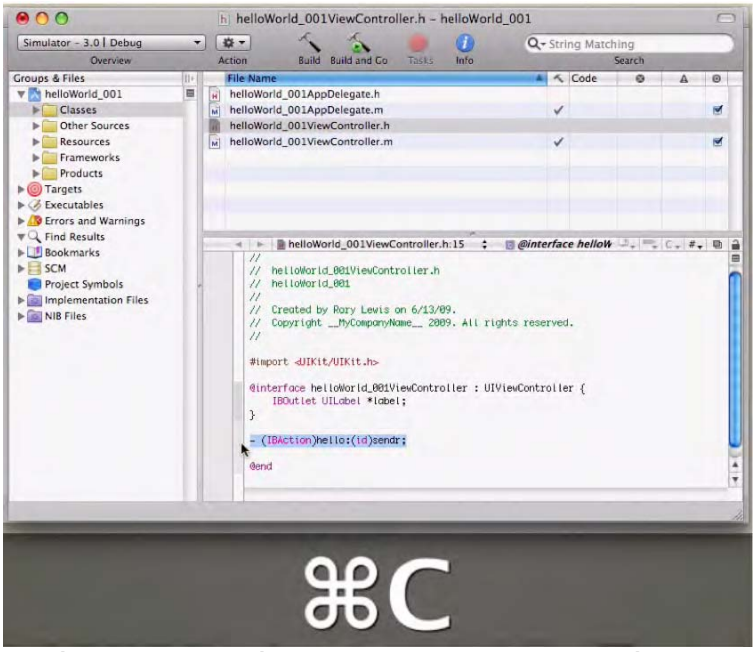


Foto 2–6. Copia el código marcando la línea indicada y usando la combinación de teclas.

7. Hemos terminado con este archivo, guárdalo usando la combinación de teclas Command + S, tal y como aparece en la Foto 2–7 (⌘S). Este es el método más adecuado para guardar, bastante más cómodo que con el ratón.

8. Como puedes apreciar en la Foto 2–8, hemos terminado de trabajar el archivo `helloWorld_001ViewController.h`. Ahora vamos a trabajar con otro archivo, en concreto el `helloWorld_001ViewController.m`. Este archivo tiene un nombre idéntico al anterior, a diferencia de la extensión del mismo, que únicamente presenta una tecla.

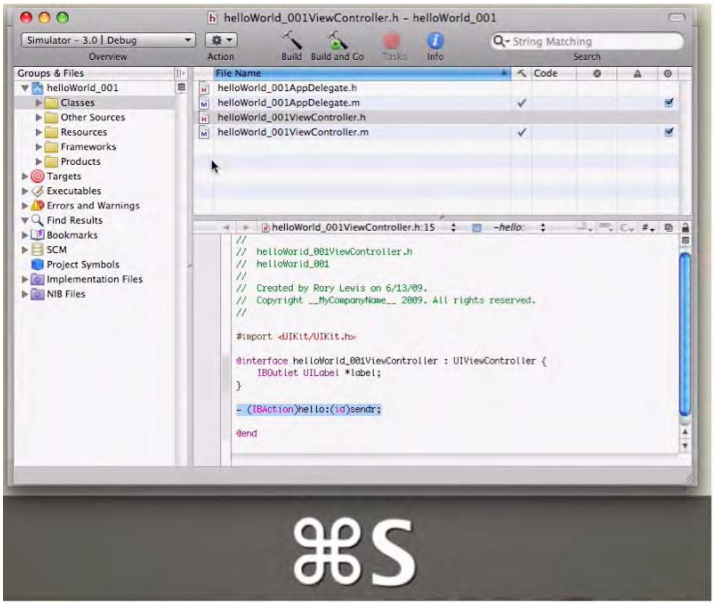


Foto 2–7. Grava el archivo usando la combinación de teclas detallada.

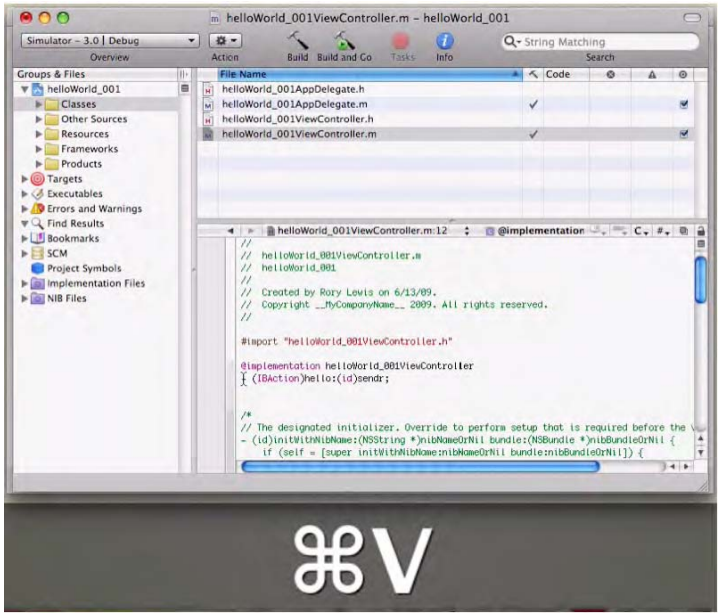


Foto 2–8. Añade a tu nuevo archivo la información, pegando la línea de código guardada (archivo anterior) en el área indicada.

Permitidme hacer diferenciación entre estos dos archivos, el de la extensión.h, y el otro con la extensión .m.

The `ViewController` gestiona las interacciones que tu código tiene con el display, y también gestiona las interacciones del usuario con tu código. Contiene una vista pero esa vista no es una vista del mismo. El `ViewController` es una clase, sobre la cual únicamente sólo tendrás un conocimiento mínimo sobre ella. Lo que quiero que te quede claro es que cada clase es formada por dos archivos: el archivo de cabecera (o header file) (.h) y el archivo de implementación (implementation file) (.m).

Me gustaría que esta parte la leyeses en voz alta. No te preocupes si estás en la biblioteca! De acuerdo? “Le decimos a la computadora en el archivo de cabecera qué tipo de comandos ejecutaremos en el archivo de implementación.”

Repetimos, esta vez en el contexto de nuestro código: “Le decimos a la computadora en el archivo `helloWorld_001ViewController.h` qué tipo de comandos ejecutaremos en el archivo `helloWorld_001ViewController.m`.”

Bien, admítelo: no ha estado del todo mal!

Ya hemos hecho la primera parte de nuestro trabajo. Ahora procederemos a la implementación del archivo y a la ejecución de comandos. Para hacer esto abriremos `helloWorld_001ViewController.m`, situándonos debajo de las líneas verdes – es decir, de todos los comentarios.

Presiona Command + V para pegar la línea que copiamos en el paso anterior, cuando estábamos trabajando con el archivo de cabecera.:

```
-(IBAction)hello:(id)sendr;
```

Repasa la Foto 2–6, si es necesario. Esta combinación de teclas puedes verla en la Foto 2–7 como ⌘V. (Por supuesto, también puedes usar el ratón accediendo al menú Edit ► Paste, en todo caso, te vuelvo a insistir en que deberías coger el hábito de usar las combinaciones de teclas!)

```
//
// helloWorld_001ViewController.m
// helloWorld_001
//
// Created by Rory Lewis on 6/13/09.
// Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import "helloWorld_001ViewController.h"

@implementation helloWorld_001ViewController
-(IBAction)hello:(id)sendr;
```

9. Como puedes ver en la Foto 2–9, Quiero que borres el punto y coma del final de la línea que acabas de pegar, el cual se utilizaba para avisar al compilador de que hemos "dejado de hablar" (declaración de una acción).

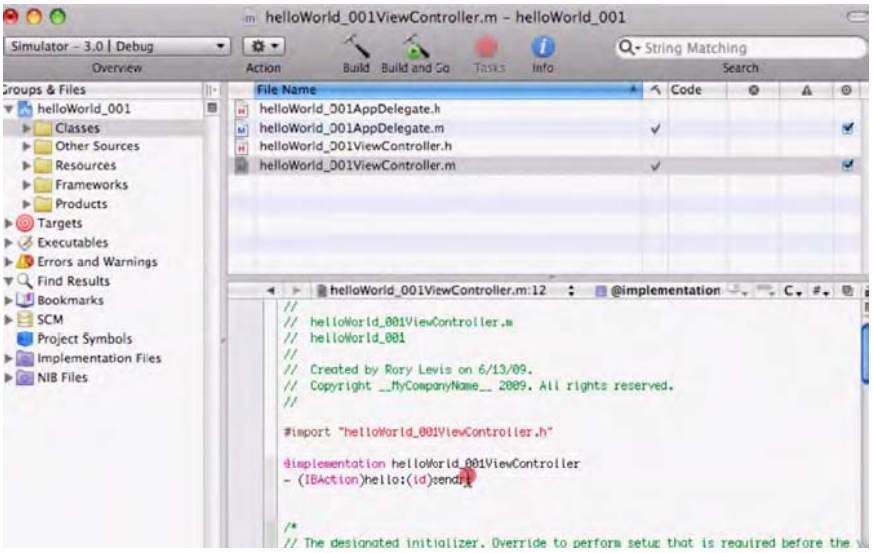


Foto 2–9. Borra el punto y coma señalado con un punto rojo en la foto.

En el archivo de implementación no queremos declarar una acción, queremos *implementarla*. Así queda el código antes de insertar los siguientes comandos que vamos a utilizar para ello:

```
//
// helloWorld_001ViewController.m
// helloWorld_001
//
// Created by Rory Lewis on 6/13/09.
// Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import "helloWorld_001ViewController.h"

@implementation helloWorld_001ViewController
-(IBAction)hello:(id)sendr
```

10. Una vez borrado el punto y coma posterior al término "sendr", procederemos a insertar un comando para cumplir con nuestro objetivo. Queremos enlazar la etiqueta que hemos creado (ver paso 6) con los medios que podrán proporcionar el texto a la pantalla del dispositivo. Para ello introduce el siguiente código inmediatamente después del término "sendr":

```
{label.text = @"Hello World!";}
```

Presiona Intro después del corchete abriente, y pulsa Intro otra vez después del punto y coma. El código tiene que quedar así:

```
//
// helloWorld_001ViewController.m
// helloWorld_001
//
// Created by Rory Lewis on 6/13/09.
// Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import "helloWorld_001ViewController.h"

@implementation helloWorld_001ViewController
-(IBAction)hello:(id)sendr{
    label.text = @"Hello World!";
}
```

Repasa la Foto 2–10 para ver los resultados.

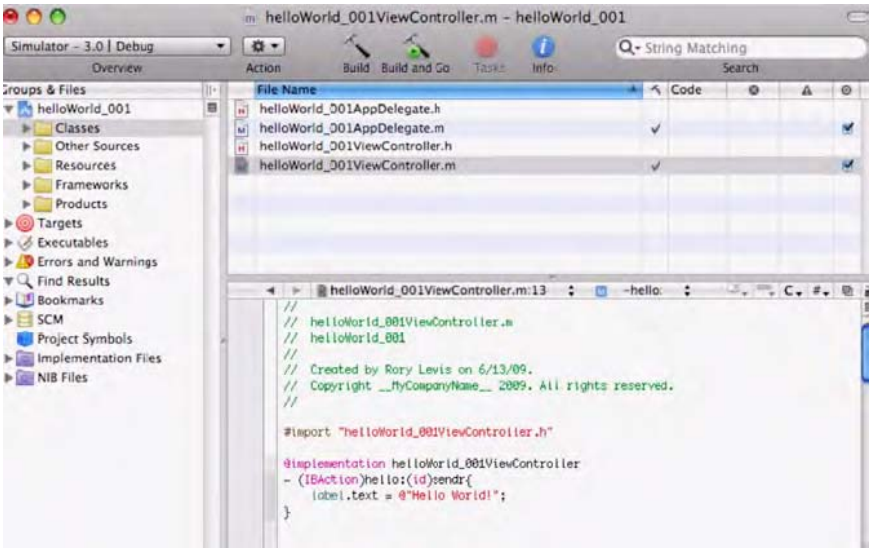


Foto 2–10. Reemplaza el punto y coma con un paréntesis abriente, introduce tu código y en la siguiente línea introduce el paréntesis cerrante.

11. Usando la combinación de teclas Command + S, guarda el archivo de implementación. Mira la Foto 2–11.

12. Ahora, procederemos a abrir el Interface Builder.ow, it's time to open Interface Builder. En lugar de abrir la aplicación y buscar el archivo que necesitamos, simplemente haremos doble clic en el archivo para arrancar automáticamente el Interface Builder. El archivo que necesitamos se encuentra la carpeta Resources. Una vez que hayas abierto esta carpeta, te encontrarás con el archivo `ViewController.xib`.

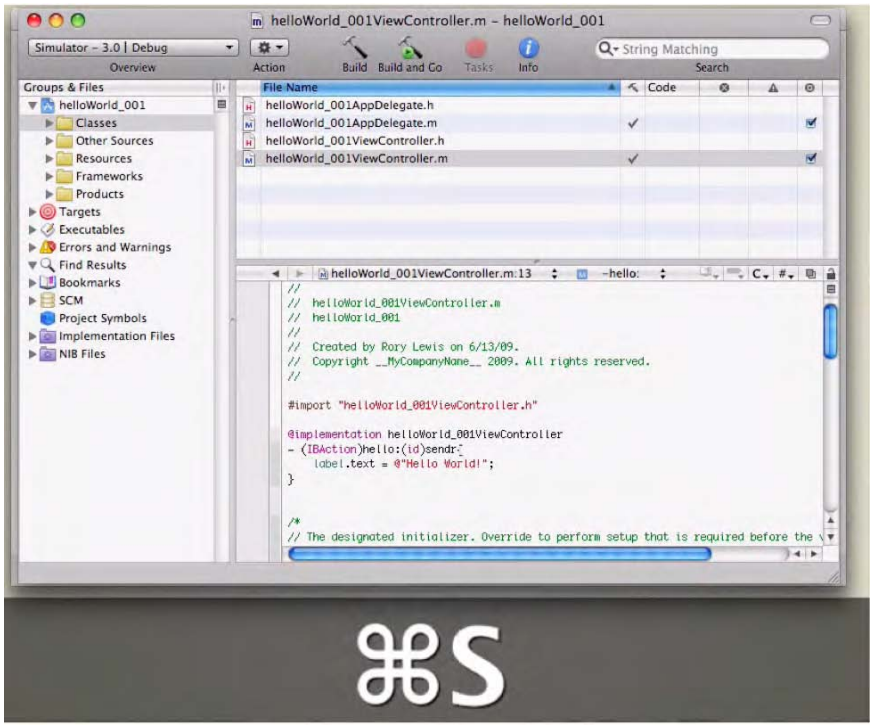


Foto 2–11. Guarda tu archivo de implementación.

A esto lo llamaremos *View Controller Nib File*. Lo denomino de esta manera porque es importante el ir aprendiendo algo de jerga de programación. Cuando un programador dice "Abre tu view controller nib file," está diciendo que vayas a la carpeta Resources y abras el archivo `ViewController.xib`.

El nombre que le dimos a nuestro proyecto precederá al texto en el archivo `ViewController.xib`. Como llamamos a nuestro proyecto `helloWorld_001`, el Intterface Builder insertará este nombre antes de `ViewController.xib`, obteniendo el nombre completo `helloWorld_001ViewController.xib`, tal y como puedes ver en la Foto 2–12. Haz clic sobre `helloWorld_001ViewController.xib` para abrir el IB.

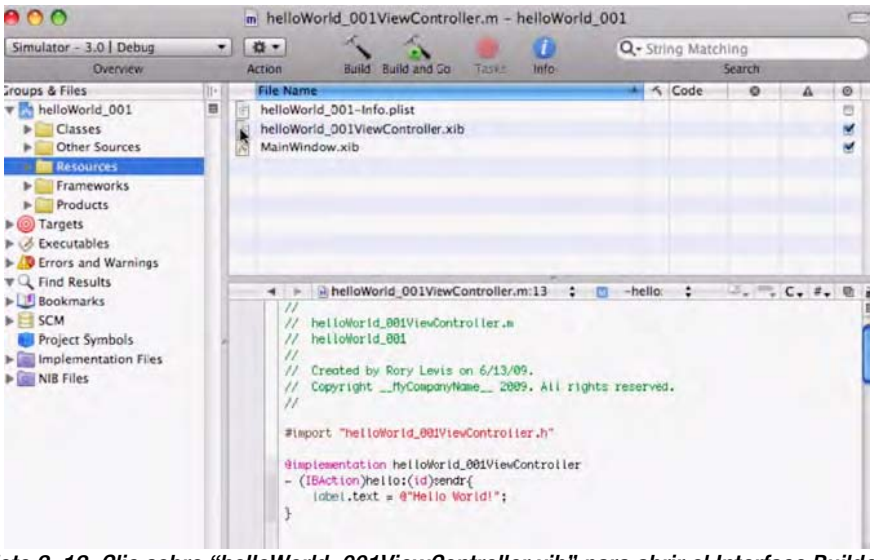


Foto 2–12. Clic sobre "helloWorld_001ViewController.xib" para abrir el Interface Builder.

13. Una vez abierto el Interface Builder, obtendremos un entorno de trabajo parecido a la Foto 2–13. Antes de ir ordenando nuestras ventanas de trabajo de un modo parecido al mío, necesitamos asegurarnos de que nuestra *Librería(Library)* está abierta. Para realizar esto, usa la combinación de teclas ⌘⇧L (o ve al menú Tools, y pincha sobre

Library). Comprueba si la ventana “Library”, probablemente situada en la parte izquierda, está presente en tu pantalla.

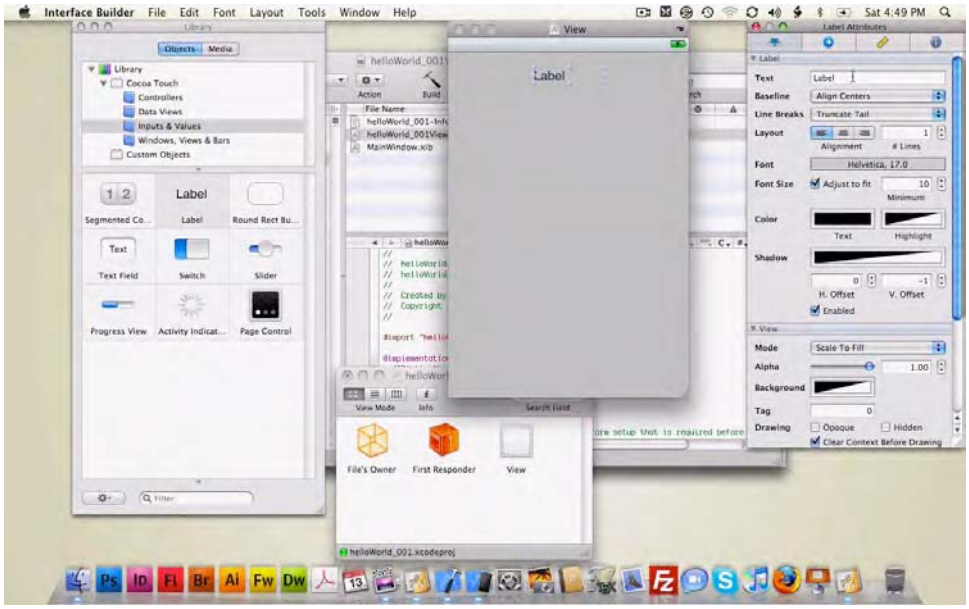


Foto 2–13. Una vez abierto el Interface Builder, asegúrate que la ventana Library está abierta.

Queremos arrastrar una etiqueta que en el argot común se denomina View window. Usaremos este mismo término en el libro, aunque realmente no es un concepto realmente preciso. SI estás interesado en forma adecuada de referirse a esta ventana de un modo técnico, podría decirse: arrastremos una etiqueta a nuestro archivo “Hello World 001 View Controller nib” file. En la Foto 2–13, este archivo está representado por un frame (o ventana) que tiene la etiqueta “View” en su encabezado.

Si lo crees conveniente, adecua el tamaño de la ventana. Ahora, en las Opciones de la ventana Label (a la derecha), borra el texto que aparece por defecto (“Label”) en el campo superior. Aquí es donde vamos a escribir la frase “Hello World!”

Advierte que puedes centrar el texto seleccionando el botón Center en la línea “Layout” (cuatro líneas más debajo de la línea de Texto).

14. Ahora vamos a explorar el resto de opciones y parámetros que están disponibles en esta ventana. Tenemos muchas herramientas para modificar la apariencia de nuestras aplicaciones. Presta especial atención a cada uno de los detalles y aprendiendo el uso de cada uno de ellos, algún día podremos conseguir colocar una de nuestras aplicaciones en la lista de Aplicaciones Más Descargadas de la AppStore!. Arrastra un botón a nuestra ventana View. Puedes expandir el marco del botón haciendo clic sobre la espina inferior del mismo, y sin soltar el botón ir arrastrando al gusto, tal y como puedes ver en la Foto 2–14. Queremos crear un mayor espacio para dar cabida al texto. Este botón será el cual pulse el usuario para que la aplicación responda “Hello World!”, por lo que en este botón necesitamos el espacio necesario para incluir el texto “Press Me” (Clícame).

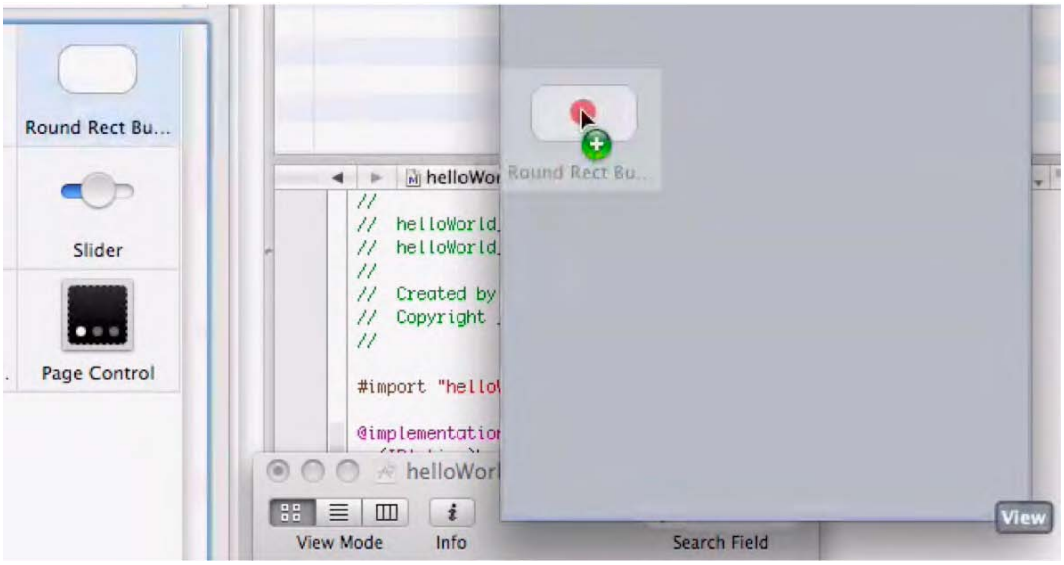


Foto 2–14. Arrastra un botón a tu ventana View.

15. Tal y como aparece en la Foto 2–15, ahora podemos introducir las palabras "Press Me" en el campo Title (Título) de la ventana Button Attributes (Atributos del botón). También podemos alinear el botón y la etiqueta con cualquier otro y centrarlos en el contexto de la pantalla. Esto lo podemos hacer seleccionando la pestaña Ruler (Regla) donde tenemos los controles de tamaño y alineamiento localizados en la parte superior derecha de esta ventana. Una vez seleccionado, desplázate hacia abajo hasta la línea Aligment y selecciona el icono Align Vertical Centers (Segundo desde la derecha), ves una línea más abajo y clica sobre Align Horizontal Center en Container a la derecha del todo. Esto hará que etiqueta y botón se centren entre ellos, y al mismo tiempo que se centre con el interface.

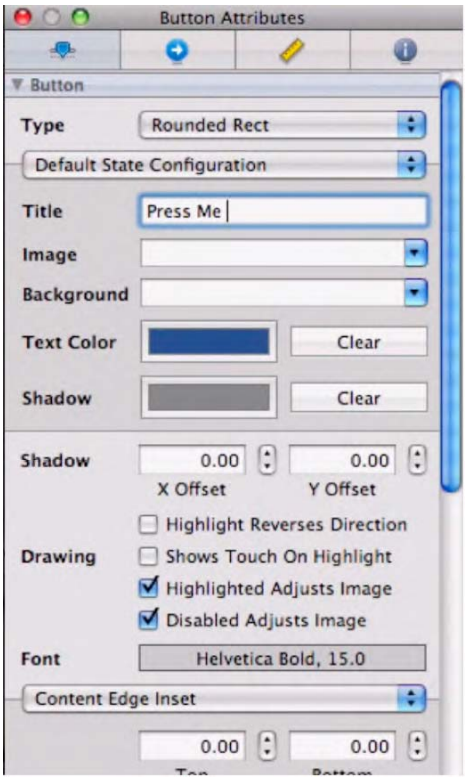


Foto 2–15. Introduce el texto "Press Me" en el campo Title en la ventana Button Attributes.

16. Desplázate a la ventana `helloWorld_001ViewController.xib` tal y como se representa en la Foto 2–16. Esta ventana contiene tres iconos: File's Owner, First Responder, y View. Ve al icono File's Owner y relaciónalo con la etiqueta presionando la tecla Control (representada mediante el símbolo ^) y pinchando sobre el icono File's Owner. Ahora desplaza el cursor del ratón a la etiqueta de texto en la ventana View, tal y como aparece en la Foto 2–16. Ten en cuenta que la "fishing line" está siendo enlazada desde el icono File's Owner hasta tu Text Label. Esto indica que la conexión que deseas crear está operativa. (Si no aparece una línea, significaría que no se ha establecido la conexión.) Cuando nos aproximemos a las inmediaciones de Text Label, veremos un cuadro de opción negro expandirse en Text Label, conteniendo la palabra "Label." Recordemos que en el paso 5 referenciamos el código que aparece como `*label`. Bien, ahora estamos viendo el mecanismo de funcionamiento. Aquí es donde queremos enlazar nuestro File's Owner con Label. Por tanto, arrastra la línea de enlace con la palabra "Label" contenida en el cuadro de opción negro, tal y como aparece en la Foto 2–16.

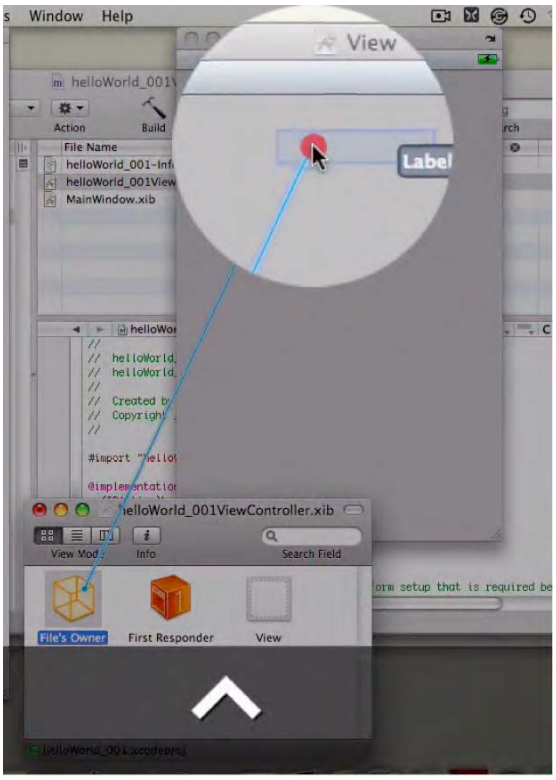


Foto 2–16. Crea una conexión desde el icono "File's Owner".

17.. Tómate un respiro y piensa acerca de lo que acabamos de hacer. Vamos a ver si podemos ir encajando las piezas de este puzzle. Quiero que te tomes un respiro, repases cuidadosamente cuatro Fotos y posteriormente vuelvas a este punto.

Primero repasa la Foto 2–5, y luego ve a la Foto 2–10. Piensa en la palabra "label." Ahora ve adelante y revisa las Fotos 2–17 y 2–18 … y sigue pensando en el término "label." De acuerdo? Hasta dentro de un rato...

Foto 2–17. Conecta File's Owner con tu Text Label.

Bien… Terminaste? Qué has advertido?
Espero que hayas advertido los pasos que estamos llevando a cabo en este proceso que nos va a llevar al destino, el cual has podido entrever en la Foto 2–18.

Recuerdas cómo en el Paso 10 introdujiste `{label.text = @"Hello World!";}`, y luego, en el Paso 16 conectaste el icono File's Owner con el Text Label? Recuerdas cómo cambiamos la Text Label para decir "Hello World!"? Probablemente estés pensando, "Vale, lo recuerdo vagamente, pero te olvidas de la Foto 2–5!"

Empezamos esta relación de eventos en el Paso 5, que dio lugar a la Foto 2–5. Establecimos una conexión entre el Outlet y la Label al introducir la línea `IBOutletUILabel *label;` … y el resto de piezas fueron encajando una a una a partir de ahí. Me hago cargo de que probablemente no hayas establecido en tu mente una conexión global de todo, pero estoy satisfecho de que al menos alguno de estos puntos sí que tengan significado y relación para ti.

Estamos en el Bosque de Objective-C y has empezado a ver ciertas conexiones interesantes. Ahora se nos presenta un detalle interesante que tenemos que clarificar y tener en cuenta. Te doy una pista… Qué es lo que pasa con el botón que reza "*Press Me*"?

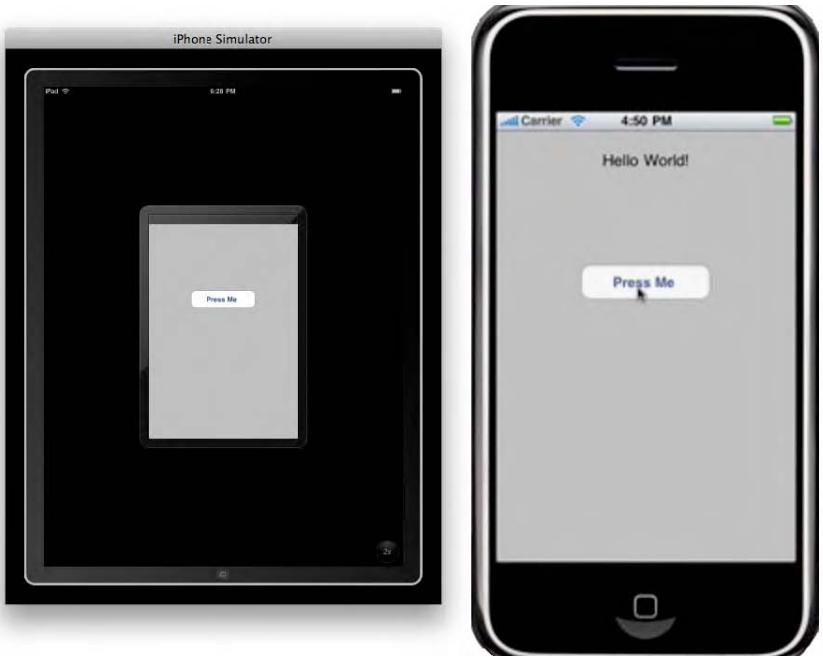


Foto 2–18. (izquierda) El botón "Press Me" representado en el iPad Simulator (derecha), podemos comprobar el resultado de pulsar el botón en el iPhone Simulator.

18. Del mismo modo que estableciste una conexión en el Paso 16, desde el icono File´s Owner a tu Text Label, ahora necesitarás conectar el botón que tiene el texto “Press Me” con el File’s Owner. Ve al icono Button y conéctalo con File's Owner presionando la tecla Control (representada como ^ en la foto) y haz clic en el icono Button. Ahora, lleva el cursor hasta el File’s Owner, (Foto 2–19). Una vez más podrás ver tu línea de enlace o “fishing line” dibujada desde el icono Button hasta File’s Owner. Esto indica que la conexión está establecida.

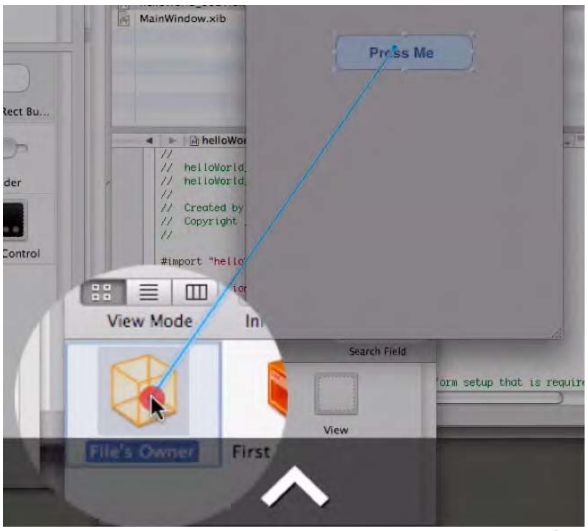


Foto 2–19. En el Interface Builder, comprueba que se ha establecido conexión entre el icono File’s Owner y la Ventana View.

19. Al acercarte a las inmediaciones del icono File’s Owner icon, advertirás que aparece otro cuadro de opción negro con la palabra “hello.” Recuerda que en el Paso 6 introdujimos – `(IBAction)hello:(id)sendr`?
El Sr. Hello está diciendo “No crees que deberías arrastrar esa línea de enlace hacia ***mí***? Recuerda – Seré yo quien avise a esa etiquetita que al marcarla hay que decir *Hello World*!”

Por tanto, debes de seguir arrastrando la línea de enlace hasta que conecte la etiqueta “hello” en el cuadro negro desplegable, tal y como aparece en la Foto 2–20.
Guarda tu trabajo: ⌘S.

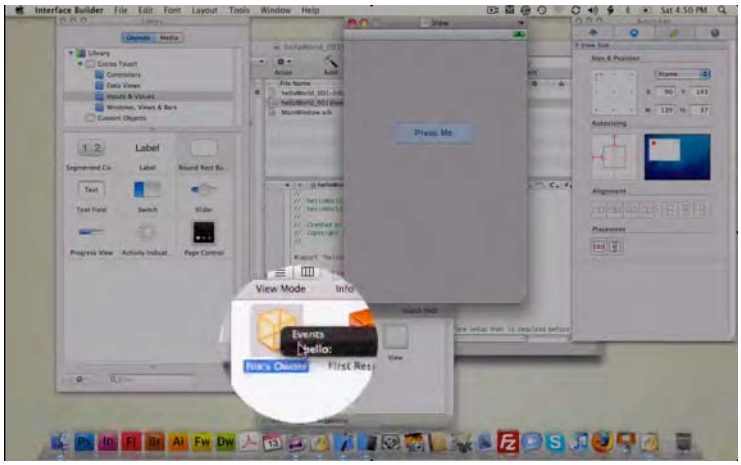


Foto 2–20. En el Interface Builder, abre la lista desplegable desde el icono File’s Owner.

20. El último paso consiste en compilar el código. Esto lo haremos presionando ⌘↩ (Command + Run), tal y como aparece en la Foto 2–21. Ahora la computador transformará todo el código en lenguaje de programación, posteriormente lo pasará a unos y ceros, para finalmente transformarlo en lenguaje que pueda ser entendido por los Simuladores iPhone e iPad … y por supuesto también por nuestros dispositivos iPhone e iPad.

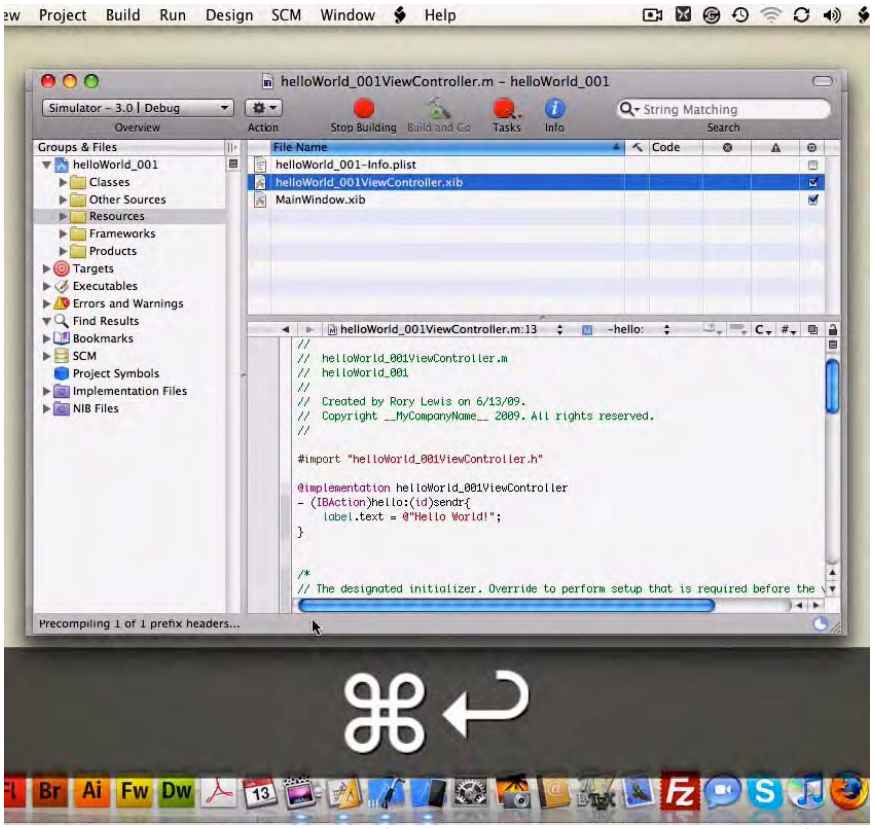


Foto 2–21. Vuelve a Xcode y usa la combinación de teclas indicadas para compilar el código.

Después de que se complete la compilación, vamos a probar nuestra aplicación. Presiona sobre el botón “Press Me” y obtendrás tu resultado. La Foto 2–22 muestra el iPad Simulator en modo Pantalla Completa, antes y después de pulsar el botón.
Acabas de completar nuestra primera aplicación para iPhone e iPad.. ***Felicidades***!

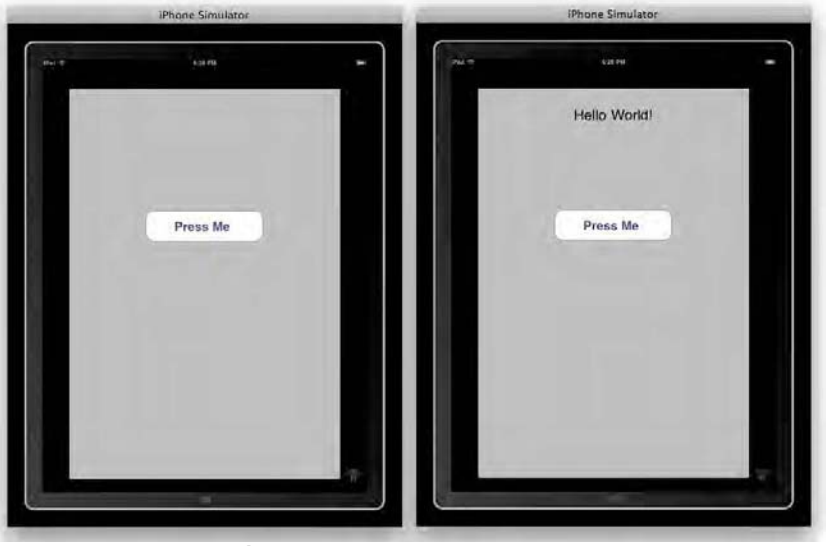


Foto 2–22. (izquierda) Precioso botón en iPad en modo pantalla completa; (derecha) Pincha sobre él … Hello World!

helloWorld_002 – en Navigation-based Application

En nuestro primer programa, "`helloWorld_001`," dijimos "Hello" al mundo desde el Template View-based Application en Xcode. Ahora empezaremos con el segundo ejemplo, `helloWorld_002`, el cual será creado mediante el Template Navigation-based Application template en Xcode.

Antes de empezar con el siguiente método, debes guardar `helloWorld_001` en el directorio que estimes oportuno. Crea una carpeta en tu directorio Documents llamada *My Programs*, y guarda el archivo `helloWorld_001` arrastrándolo a esa carpeta. A continuación, con un escritorio despejado y vacío, cierra todos los programas. Presiona Command + Tab y luego Command + Q para cerrar todo hasta que el Finder aparezca en pantalla.

Tal y como hiciste en tu primer ejemplo, lanza Xcode y abre un nuevo proyecto usando la combinación de teclas: ⌘⇧N. La pantalla debería mostrar el asistente de Nuevo Proyecto tal y como se muestra en la Foto 2-23. Podrás apreciar que tu Template View-based Application template estaba marcado por defecto dado que fue el método utilizado en el último ejemplo. Ahora clica sobre el icono Navigation-based Application, y guarda el Nuevo archivo en tu escritorio con el nombre "`helloWorld_002`."

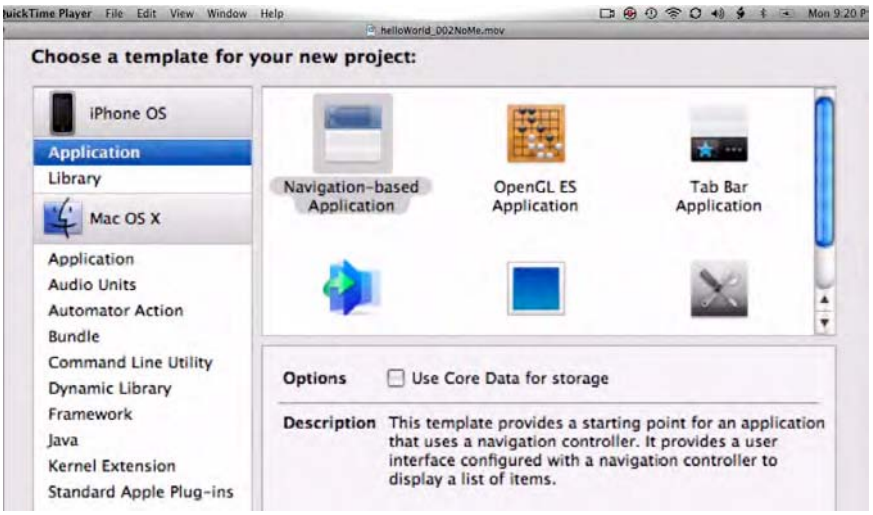


Foto 2–23. Abre Xcode, selecciona el Template Navigation-based, y guarda el Nuevo proyecto en tu escritorio.

1. Una vez guardado el proyecto en el escritorio, Xcode inicializa un archivo llamado `helloWorld_002` con un grupo llamado Classes, tal y como vistes en el ejemplo anterior (ver Foto 2–24). Comprobaremos que Template navigation-based está formado por dos pares de subarchivos: un header y main AppDelegate, junto con un RootViewController header y main. Al hacer clic sobre el archivo `RootViewController.h` (header), advertirás que obtenemos lo que llamamos una subclase de UITableViewController, que se ocupa de controlar una serie de herramientas necesarias para la visualización de tablas en una aplicación iPhone/iPad. Estas Tablas son utilizadas mucho más de lo que la mayoría de la gente percibe. Ya que utilizaremos estas tablas para muchas aplicaciones, pienso que deberíamos aprender ha decir "Hello World!" en este contesto. De hecho, ya tenemos todo lo que necesitamos para llamar a una tabla en blanco.

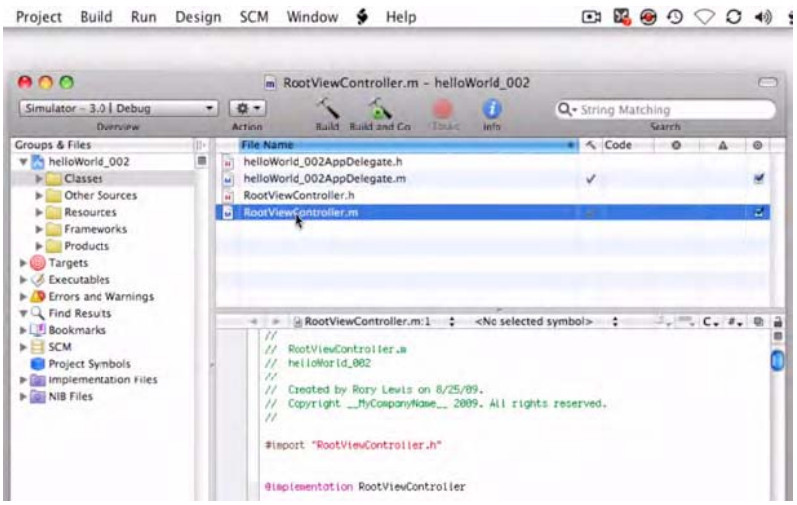


Foto 2–24. Abre el directorio Classes y clica en el archivo RootViewController.m para examinar el código repetitivo preexistente.

Recordemos que al final del primer ejemplo (Foto 2–22), compilamos nuestro código presionando ⌘↩. Vimos cómo la computadora convirtió todo el código que habíamos introducido en una aplicación. Bien, ahora vamos a hacer esto de nuevo, pero sin introducir una sola línea de código.

En Xcode, seguimos adelante presionando ⌘↩. Vea la Foto 2–25.

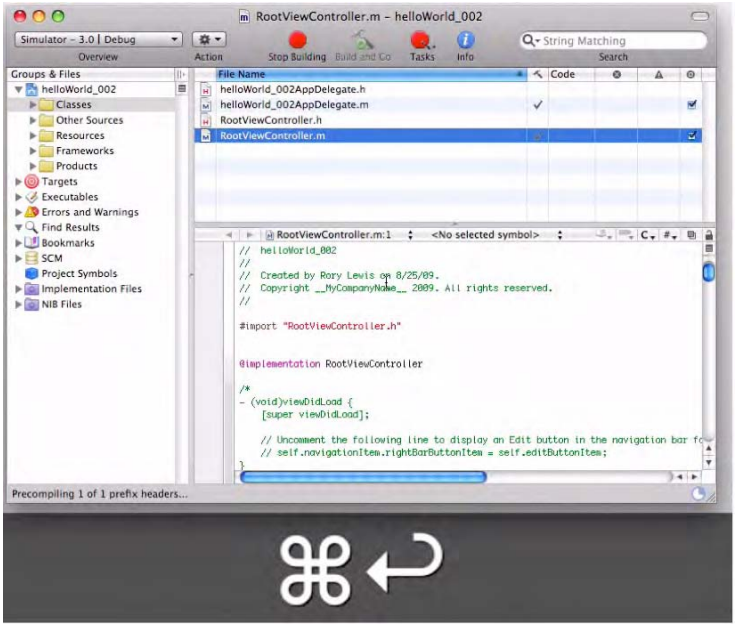


Foto 2–25. En Xcode, usamos las combinaciones de teclas para compilar y ejecutar nuestro programa—a pesar de no haber introducido ni una sola línea de código

Como puedes comprobar en la Foto 2–26, es posible compilar pequeñas porciones de código preprogramado, pero este tipo de compilación carece de elementos específicos. El modo vista de tabla está desplegado, pero evidentemente el equipo no tiene ni idea de lo que queremos decir o mostrar, por lo que tenemos que indicárselo.

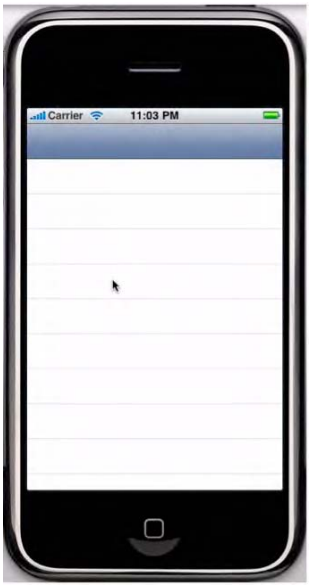


Foto 2–26. Ejecutando el programa llamamos al Simulador iPhone e iPad, donde podremos apreciar una Tabla perfectamente conformada si nada dentro.

2. El simulador iPhone/iPad tiene varias poderosas estructuras que han sido preparadas con antelación, y estamos viendo una de las típicas proyecciones vía el Template de la Navigation-based Application. Que piensas que puede provocar que esto suceda?

Respuesta: el `UITableViewController`, que contiene código que en su debido momento examinaremos. De momento, lo archivamos en misterios sin resolver.

Recordemos que tenemos que cumplir con un objetivo y ese objetivo es usar una línea de la tabla para decir “Hello World!” Vamos a ello.

3. Primero cerramos el Simulador iPhone/iPad mediante la combinación de teclas ⌘Q. Volvemos a Xcode, en concreto al archivo de implementación RootViewController, que va a ser donde introduciremos nuestro código (ver Foto 2–27).

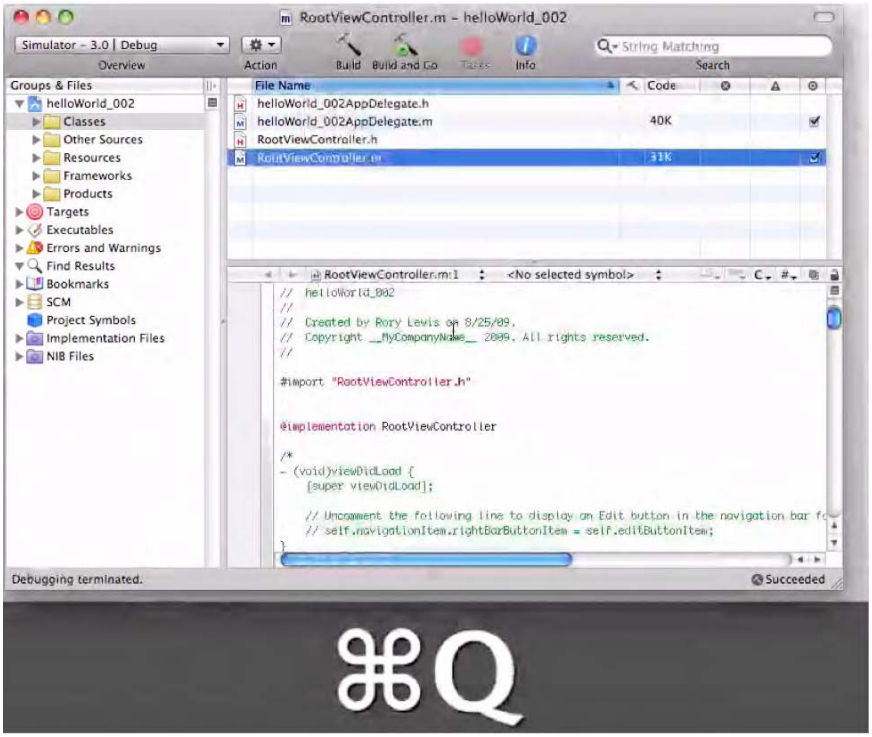


Foto 2–27. Usando la combinación de teclas indicada, saldremos del Simulador y volveremos a Xcode, donde podremos empezar a introducir nuestro Nuevo código.

Cuando abrimos por primera vez `RootViewController.m`, nos encontramos con un montón de anotaciones en verde. Algunos de estos comentarios hacen referencia a funciones prediseñadas que los chicos de Apple generosamente han programado y nos las han facilitado, con el objetivo de que nos sean de utilidad para los programadores. De momento no vamos a hacer uso de ellas, pero más tarde sí que lo haremos.

Una de las cosas que vamos a hacer e sir a la sección que tiene que ver con el número de filas de nuestra tabla. Aquí haremos un pequeño cambio para lograr el objetivo de nuestro ejercicio. Ver Foto 2–28.

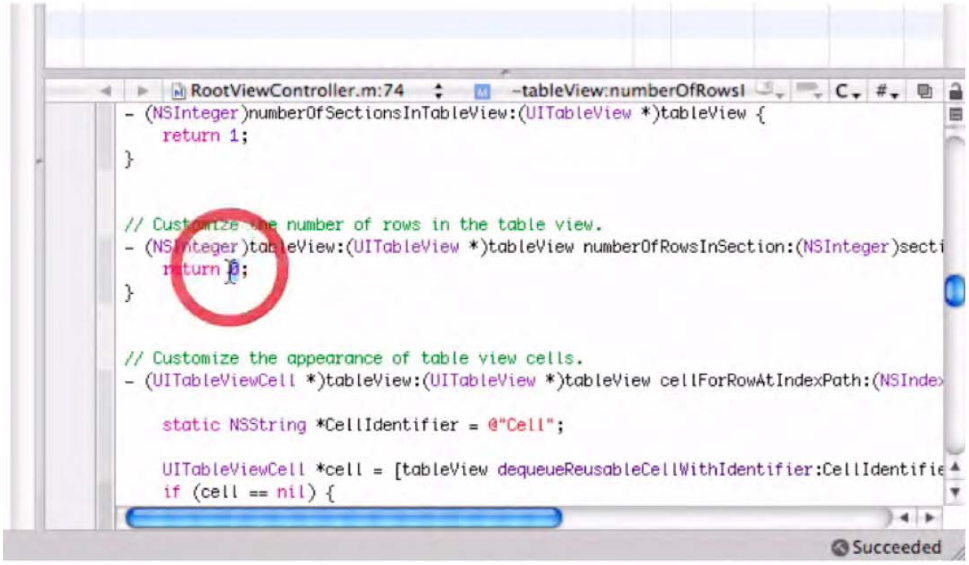


Foto 2–28. El círculo en rojo muestra el sitio donde tienes que cambiar el valor por defecto de 0 a 1.

4. Baja en la pantalla del archivo `RootViewController.m` hasta que veas el código que determina cuántas filas vamos a tener en nuestra tabla de aplicaciones. En esta simple aplicación únicamente necesitamos tener una línea, por lo que tendremos que decirle que no devuelva como valor cero el número de filas, sino que dicho valor tiene que ser uno. El círculo en rojo en la Foto 2–28 indica el lugar donde tenemos que realizar el cambio.

Por tanto borra el 0, y reemplázalo por un 1.

Cambia esto:

```
// Customize the number of rows in the table view.
          -(NSInteger)tableView:(UITableView
          *)tableViewnumberOfRowsInSection:(NSInteger)section {
return 0;
}

to this:
          -// Customize the number of rows in the table view-
          (NSInteger)tableView:(UITableView
          *)tableViewnumberOfRowsInSection:(NSInteger)section {
return 1;
     }
```

5. Ahora quiero que agregues un poco de código que cambiará las propiedades de la línea que vamos a manipular. Queremos decir “Hello World!” en una línea. Justo debajo de la línea donde hemos hecho el reemplazo en el paso anterior, podremos apreciar una

sección con un comentario en verde: “Customize the appearance of table view cells.” (Personalice la apariencia de la vista de tabla).

Baje hasta donde dice, “ConFoto the cell,” tal y como muestra la Foto 2–29. Subraye esa línea y bórrela.

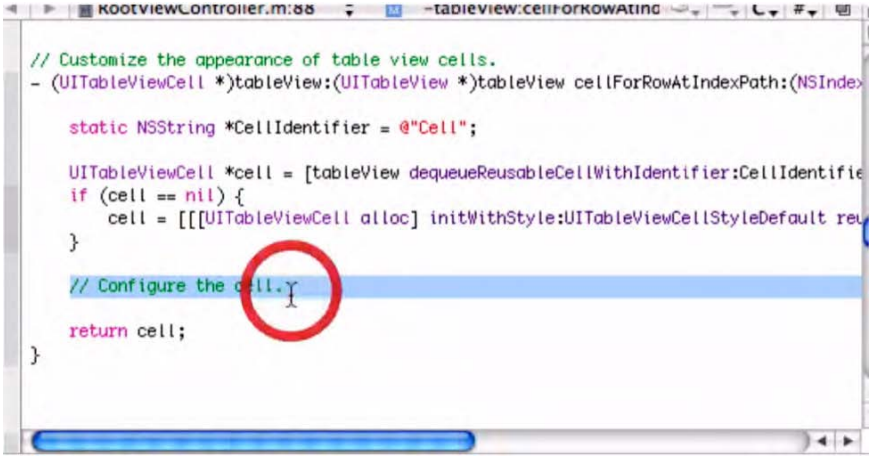


Foto 2–29. Borre esta línea de código por defecto e ingrese el comando deseado.

6. Ahora, en esta misma línea, escribiremos:

```
[cell.textLabelsetText”@"Hello World!"];
```

Necesitamos una celda que tiene el código de Apple necesario para insertar el texto en una etiqueta. Por otra parte queremos introducir la frase “Hello World!” para rellenar la misma. Este es el código necesario:

```
-// Customize the appearance of table view cells- (UITableViewCell
*)tableView:(UITableView
*)tableViewcellForRowAtIndexPath:(NSIndexPath *)indexPath {

staticNSString *CellIdentifier =”@"Ce”l";

UITableViewCell                              *cell                              =
[tableViewdequeueReusableCellWithIdentifier:CellIdentifier];
if (cell == nil) {
cell = [[[UITableViewCellalloc]
initWithStyle:UITableViewCellStyleDefaultreuseIdentifier:CellIdentifie
r] autorelease];
     }

          [cell.textLabelsetText”@"Hello World!"];
return cell;
```

La aplicación te preguntará si quieres guardar cambios una vez ejecutemos la combinación de teclas para compilar y ejecutar nuestro Nuevo código (⌘⮌). Ver Foto 2–30. Pulse Intro.

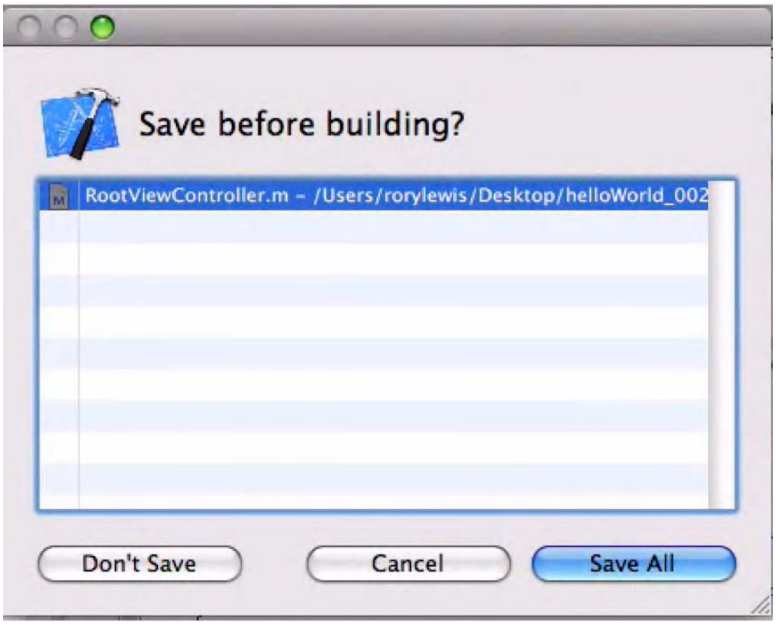


Foto 2–30. Una vez introducida la combinación de teclas para compilar el Nuevo código el programa te preguntará si quieres guardar tu trabajo.

7. Una vez finalizado este paso, vemos una tabla que contiene una línea con el texto deseado "Hello World!" (Ver Fotos 2–31 a 2–34).

Buen trabajo! Hemos terminado nuestra *segunda* aplicación iPhone/iPad .

Foto 2–31. Voilà … éxito!

Foto 2–32. El resultado es una sencilla tabla con la frase "Hello World!" en el primera línea, mostrada en Vista iPhone dentro del iPad.

Foto 2–33. Pincha en "Hello World!" y la primera línea se tornará activa.

Foto 2–34. Aquí vemos la primera línea subrayada y active, vista en Modo Pantalla completa del iPad.

helloWorld_003 – Modificando una Aplicación Navigation-based

En nuestro ultimo ejemplo de este capítulo, repetiremos los pasos seguidos en el ejemplo anterior, helloWorld_002, y haremos algunas pequeñas modificaciones. Selecciona la Aplicación Template Navigation-based otra vez y llama al proyecto helloWorld_003.

1. Desplázate hacia abajo en el archivo `RootViewController.m`, y tal y como hiciste en el ejemplo helloWorld_002, introduce esta línea de código:

```
[cell.textLabelsetText:@"Hello World!"];
```

Ahora el archivo RootView Controller tendrá el siguiente aspecto:

```
// Customize the appearance of table view cells.
-(UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
      static NSString *CellIdentifier = @"Cell";
      UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
      if (cell == nil) {
   cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier] autorelease];
       }
      [cell.textLabel setText:@"Hello World!"];
```

```
        return cell;
 }
```

2. Ahora quiero que centres el texto. Para hacerlo puedes escribir el siguiente código directamente, o puedes copiar y pegar la línea que acabas de insertar arriba, editándola de la siguiente manera:

[[cell textLabel] setTextAlignment:UITextAlignmentCenter];

Quedando de la siguiente manera:

```
// Customize the appearance of table view cells.
-(UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
      static NSString *CellIdentifier = @"Cell";
      UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
      if (cell == nil) {
   cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier] autorelease];
       }
      [[cell textLabel] setTextAlignment:UITextAlignmentCenter];
      [cell.textLabel setText:@"Hello World!"];
       return cell;
}
```

En esencia, este comando nos está diciendo, “Hey, celda con etiqueta de texto, `[cell textLabel]`, obedece la s instrucciones que te está dando tu superior, `setTextAlignment`, y obedece a la Comando de Interfaz de Usuario que se te acaba de dar”

Para la salida de la línea de texto esto se traduce, “Arriba del todo y céntrate, amigo!”

3. Pero espera, todavía no hemos terminado. En este pequeño ejercicio quiero que hagas el texto editable, hasta cierto punto. Vamos a colocar un botón Edit, sobre el cual si se presiona, le dará al usuario la posibilidad de borrar nuestro texto “Hello World!”.

Cómo hacemos esto? Vuelve al Paso 4 del ejemplo 2, donde mencioné que hay varias funciones que los programadores a menudo no utilizan. En este caso, es el momento de volver y sumergirnos en este punto para hacer un cambio.

Anteriormente estuvimos hablando sobre los comentarios que aparecían en pantalla. Si recuerdas, estaban representados en verde e incluían información o funciones que eran invisibles para el procesador. Antes, los comentarios venían introducidos por una doble barra //. Ahora hay otro estilo de comentarios, consistente en acotar los con un /* al inicio del comentario, y un */ al final del mismo.

Vuelve a la parte superior y encontrarás un código que dice “Uncomment the following line to display an Edit button in the navigation bar …” parecido a estos:

```
/*
-(void)viewDidLoad {
   [super viewDidLoad];

   // Uncomment the following line to display an Edit button in
the navigation bar for
this view controller.
   // self.navigationItem.rightBarButtonItem = self.editButtonItem;
}
*/
```

4. Necesitarás borrar los dos marcadores de comentario /* y */, aparte de borrar las líneas de comentarios que comiencen por //. Ahora, la sección quedará así:

```
-(void)viewDidLoad {
   [super viewDidLoad];

      }
```

Excelente! Ahora vamos a seguiremos de un modo análogo al como lo hicimos en el ejemplo helloWorld_002, empezando por guardar nuestro trabajo en el archivo RootView Controller. Usando la correspondiente combinación de teclas, guarda y compila el código.

Felicidades! Acabas de terminar tu tercera aplicación iPhone/iPad!

La Foto 2–35 muestra claramente los resultados de nuestro gran trabajo.

Foto 2–35. Con estas simples modificaciones, conseguiremos una tabla en blanco con un texto centrado arriba del todo, acompañado del botón Edit.

Tan simple como esto. Tenemos que tener en consideración los mecanismos que han permitido que el código preprogramado nos ayude y permita esta funcionalidad, a los cuales hemos accedido simplemente haciendo unos pocos cambios elementales en el archivo. Si pinchamos en el botón Edit, aparecerá un símbolo en rojo y blanco, produciéndose también que el texto del botón pase de “Edit” (Editar) a “Done” (Hecho) Ver Foto 2–36.

Foto 2–36. Pulsando el botón Edit activamos el comando "Delete" (Borrar) representado por un icono rojo y blanco. Ahora el texto del botón cambia a Done (Hecho).

Capítulo 3

¿Qué es Lo Que Nos Vamos a Encontrar?

Ahora que te has mojado programando sus tres primeras aplicaciones para iPhone y para iPad, quiero que te preguntes: *¿Hacia dónde voy*? La respuesta a esa pregunta la encontrarás en este capítulo.

Pospuse intencionadamente esta orientación hasta que hubiera hecho algo de programación. Con el regusto de código fresco en tu mente, ahora estás en disposición de apreciar la secuencia y la variedad de los retos que tenemos por delante. ¡Siga leyendo!

Foto 3–1. El autor con aplicaciones para iPhone e iPad para principiantes absolutos en ambos dispositivos.

Para ayudarle a orientarse, he dividido este capítulo general en tres secciones. En la Sección Uno (§I), explico por qué he escogido la hoja de ruta expuesta en los Capítulos 4 a 9, y doy una breve descripción de cada capítulo.

En la Sección Dos (§II), trato la relación entre el iPhone y el iPad (Foto 3–1), y cómo esta relación puede afectar a las materias que vamos a ver y ejercicios correspondientes, dando quizá una nueva perspectiva a sus prioridades creativas y a sus objetivos.

En la Sección Tres (§III), nos aventuramos en el hardware, y consideramos las estructuras más profundas que implica el código que ejecutan estos dispositivos. No es preciso que leas el §II o §III inmediatamente, pero te sugiero que al menos le eches un vistazo al §III –incluso si piensas que no te estás enterando de nada. Simplemente deja correr sus ojos sobre el texto. Probablemente descubrirás que no viene mal echar de vez en cuando una ojeada y releer estas líneas.

§I: El CAMINO POR DELANTE

Basé las clases que imparto sobre el iPhone/iPad en la Universidad de Colorado en los seis componentes más comunes de todas las aplicaciones para iPhone/iPad: navegación, acciones (con salidas simples o múltiples), cambio de vista en pantalla, interacciones táctiles sobre la misma, gestos, tablas y mapas. He aquí un esquema de cómo se abordan estos elementos

Capítulo 4: Introducción al Código

Botones & gráficos
Capítulo 5: Botones & etiquetas con gráficos múltiples

IBOutlets & UILabels
Capítulo 6: Switch View (Cambio de vista en pantalla) con gráficos múltiples

Tres enfoques diferentes
Capítulo 7: Arrastrar, rotar y escalar

Toques, gestos y sucesos
Capítulo 8: Vista de tabla, navegación y arrays

Designar una serie de tablas enlazadas
Capítulo 9: mapKit

Anotaciones, Vistas de Mapa y controles de mapa

Presentación del capítulo 4 – *Introducción al Código*

In el capítulo 4, haré que hagas lo que mis estudiantes hacen: ayudarme a aprender su nombre. Vas a manipular un retrato tuyo, trabajarás con él para aprender sobre botones y gráficos. Si no tienes un retrato tuyo, puedes descargar la foto que uso en el ejemplo (Figura 3-2). Cuando hayas acabado habrás pasado de la etapa de código “Hello World!” a poner botones para controlar gráficos. En este capítulo, usaremos botones y gráficos que interactúan con una sola vista.

Nuestro trabajo con botones y etiquetas nos evocará preguntas interesantes, y haré todo lo posible para contestar las más urgentes. Por ejemplo, los informáticos que desarrollaron “Cocoa Touch”, el código subyacente que controla como interactuamos los humanos con el iPhone y el iPad, usaron un concepto que llamaron Modelo-Vista-Controlador (Model-View-Controller) (MVC). Una parte esencial de este modelo es el código que forma la Interfaz Gráfica de Usuario (GUI). La GUI es la forma con que los usuarios se relacionan, hablan y se comunican con el ordenador. Profundizaremos en los componentes del MVC en la tercera sección de este capítulo: Una mirada por dentro (bajo el capó)

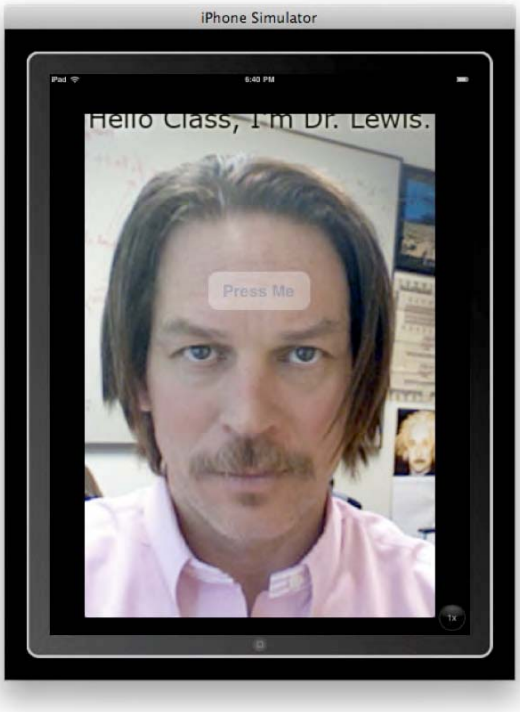


Foto 3–2. Hola, Soy el doctor Lewis.

Presentación del capítulo 5 –*Botones y etiquetas con gráficos múltiples*

Los botones y gráficos que interactúan con una vista están bien, pero ¿cómo hacemos que un botón interactúe con más de una vista? No se trata en realidad de hacer que un programa gráfico interactúe con vistas múltiples; se trata de configurar el programa, el que sea (un mapa, un juego, código para calcular) para que salte entre varias vistas de forma eficiente y sin esfuerzo. Este concepto es importante para toda la programación futura, y tu experiencia en este capítulo le servirá para el resto del libro.

El capítulo 5 no es sólo un paso adelante en términos de complejidad de código, también es una transición a un lenguaje más técnico. Pero no se apure –¡estará bien! Todo está expuesto con pasos discretos y analogías divertidas. Verá como hacemos un pequeño juego de manos al tomar una foto de una escena y superponer otra foto en ella (ver Figura 3-3). Esta técnica se usa constantemente en juegos, y se tocarán algunas ideas relacionadas en la discusión de INDIO al final de este capítulo.

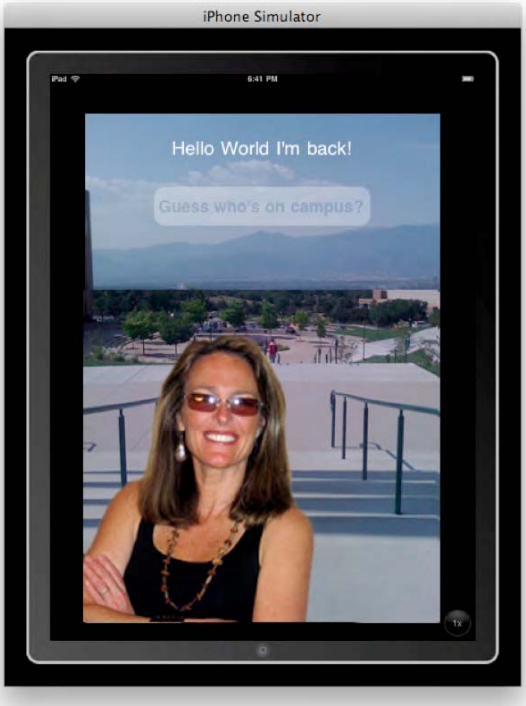


Figura 3–3. La imagen del iPad de la superposición de vistas: "¡Hello World, he vuelto!".

Presentación del capítulo 6 –*Cambio de vista con gráficos múltiples*

Este es el capítulo más largo del libro porque trabajamos tres ejemplos que hacen lo mismo pero de forma diferente. El capítulo 6 trata de pestañas y cambio de vista con pestañas –algo muy común en casi todas las aplicaciones para iPhone e iPad que he visto (ver Figura 3-4). En el primer ejemplo, codificamos los gráficos de forma larga. En el segundo ejemplo vemos lo fácilmente que funciona la forma simple – sin embargo también fracasa en cuanto a instruirnos o proporcionarnos la experiencia del código que subyace en las pestañas y los cambios de vista. Por tanto, usaremos el tercer ejemplo para componer un híbrido de los dos enfoques; Ganaremos la experiencia y resistencia de un método y el ahorro de tiempo que supone el otro.

El éxito que obtengas en el capítulo 6 te hará avanzar del estado de principiante al de auténtico programador-en-progreso. Al abordar algunos aspectos difíciles de la codificación, te habrá ganado el respeto de la comunidad de programadores para iPhone/iPad. La buena noticia es que estarás preparado y dispuesto para el próximo paso, porque habrás seguido el camino que he preparado sobre la base de mi experiencia educativa previa.

Gran parte de mi filosofía es que nos preparemos para retos futuros por medio de la mentalización y visualización. Esto requiere o implica organizar nuestras abstracciones en partes que podamos manipular y utilizar para otro fin, en cualquier otro contexto nuevo que surja. ¡Es cosa de flexibilidad! Por tanto, empieza ahora a imbuirte en estos ejercicios y retos futuros y a apreciar que no solo tratan de mostrar imágenes al azar al pulsar un botón.

Es vital que vea que cada foto representa una parte de código diferente, el cual le da al usuario de la aplicación acceso a una vista nueva, a un nuevo conjunto de opciones o a un nuevo nivel del programa. Usamos las fotos en esta parte del proceso de aprendizaje porque es más fácil que el código. También hace más fácil la solución de problemas; si la imagen no hace lo que se supone que tiene que hacer, sabes que tu código de "cambio de imagen" es el problema.

¡Prepárese para ser *catapultado a la estratosfera de la programación*!

Figura 3–4. Cambio fluido en la vista completa del iPad. ¡Ni una línea de código!.

Presentación del capítulo 7 *–Arrastrar, rotar y escalar.*

El desarrollar y aprovechar la potencia del Multi-Touch en el iPhone y el iPad puede ser increíble y eso es lo que vamos a aprender en este capítulo. La creación de una imagen que rote, escale y se mueva con un toque de dedos intuitivo confirma la potencia de estos dispositivos. (ver Figura 3-5)

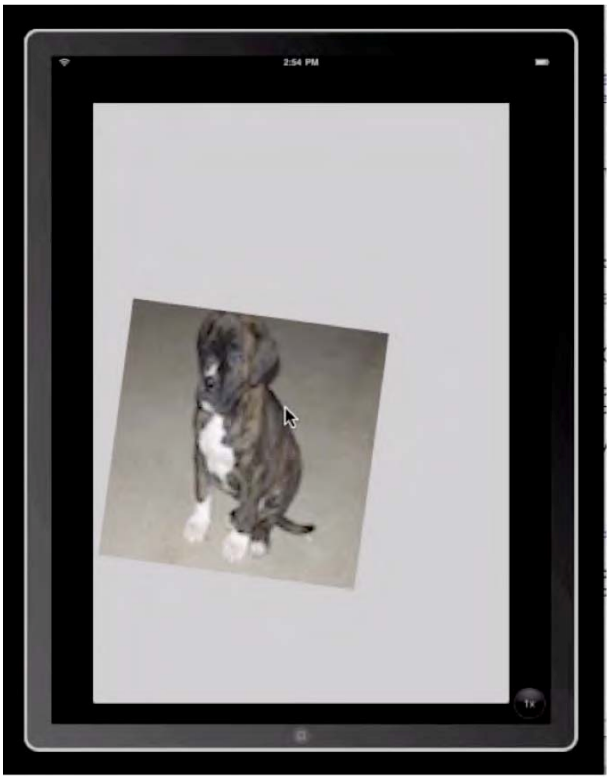


Figura 3–5. Los ejemplos funcionan tanto en el iPhone como en el iPad.

Piénselo. Antes del iPad y el iPhone, usábamos un ratón y un teclado. Ahora se trata de tocar. Hay un viejo acrónimo que algunos de vosotros probablemente conozcáis: WYSIWYG. Significa "what you see is what you get," –lo que ve es lo que hay, cuando un documento de un procesador de textos se imprime de la forma que esperas. Ahora, en el mundo Multi-Touch de Apple, vemos que se forma un nuevo paradigma –uno que refleja que tocar la pantalla de cierta forma logra una tarea natural y lógica de forma fácil e intuitiva. El ordenador le dirá, como el genio decía a Aladino, "Tu toque es mi orden".

En el Capítulo 7, le mostraré e iniciaré al código que interactúa con los diferentes tipos de toques y gestos que los usuarios emplean cuando ejecutan la aplicación. Si eres como mis antiguos alumnos, te encantará aprender la gramática que dirige las imágenes para que se muevan según nuestros toques, golpes, pellizcos, arrastres, etc.

Presentación del capítulo 8 *–Vista de tablas, navegación y matrices.*

La vista de tablas y la navegación son elementos esenciales en la inmensa mayoría de las aplicaciones. ¿Se ha preguntado como hace la App Store esa lista tan *cool* de aplicaciones? ¿Alguna vez ha necesitado tener una lista de cosas a mano, pero no está

seguro de qué aplicación usar? ¡No se preocupes más! Creará una aplicación con vista de tabla capaz de cambiar entre múltiples vistas (ver Figura 3-6). ¿No encuentra lo que quiere en la App Store? ¡Hágalo para usted –y para otros!

Aunque esto suena fantástico, hay un problema. Para ir más allá de lo esencial de la vista de tablas y la navegación, necesitas meter la cabeza en una bestia de la programación: la matriz.

Debido al grado de dificultad de este concepto de programación, generalmente se establece que las matrices no deberían enseñarse a los principiantes. No pensaba incluir este capítulo en el libro, pero como muchos de mis estudiantes querían utilizar tablas en sus proyectos finales, tuve que replantearme la estrategia. Decidí enseñar a mis estudiantes, y a los lectores de este libro, como trabajar con matrices y controlarlas, dando nociones básicas de cómo funcionan en realidad.

Por tanto, quiero considerar el Capítulo 8 como un capítulo opcional. Cuando lleguemos le guiaré a través del proceso de decidir si quiere aprender matrices, y entonces insistiré en que vaya directamente al capítulo 9. Cruzaremos el puente cuando lleguemos.

Figura 3–6. Tabla de vista creada en el capítulo 8.

Introducción al capítulo 9—*MapKit*

El capítulo 9 es mi favorito y pronostico que será el que más disfrutes tú también. Para cuando llegue allí, tendrá suficientes conocimientos básicos para disfrutar del MapKit – el código que usamos para interactuar con mapas en el iPhone y el iPad. Encontraremos nuestra localización en el mundo y nos daremos cuenta de lo diminutos, aunque inteligentes, que somos (Figura 3-7). Crear nuestras propias anotaciones, vista

de mapas y controles de mapas son solo algunos de los placeres que nos esperan.

También te llevaré a recorrer algunas aplicaciones excelentes existentes que usan las funciones de mapas, con la esperanza de que te inspiren y provoquen ideas para creaciones futuras. En ese sentido, compartiré algunos de los proyectos de mis alumnos que demuestran vivamente lo rápida e impresionantemente que se pueden aplicar los conocimientos adquiridos en el curso para producir aplicaciones a un nivel de sofisticación decente.

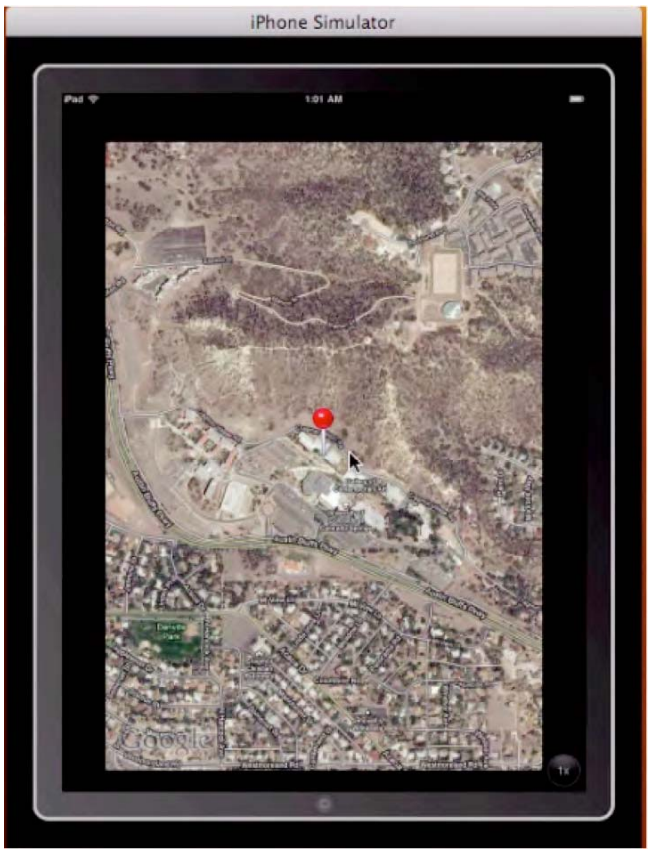


Figura 3–7. La aplicación MapKit que construiremos en el capítulo 9.

§II: EL IPHONE Y EL IPAD

Desde el principio, vamos a entender de qué va el iPad. De hecho, vamos a pensar lo que todos los genios informáticos de Apple hacían después del lanzamiento del iPad. En las primeras 24 horas de la salida del iPad, había muchas posibilidades de que los Maqueros se rascaran la cabeza protestando. ¿Qué hay aquí para mi? No hay nada nuevo. ¡Es un asco! La gente tenía la idea de que con sus Macs y sus iPhones tenían todo lo que necesitaban, y muchos de ellos concluyeron rápidamente que el iPad no serviría ninguna necesidad inmediata.

Bien, entonces, ¿quién es el público objetivo para el iPad?

Sabiendo por qué se desarrolló el iPad y para *quien*, entenderemos con qué tipo y estilo de aplicaciones tenemos más oportunidades de éxito. Pasé casi dos años siguiendo los

acontecimientos que llevaron a la presentación del iPad –desde la especulación inicial a si se llamaría iTablet o iSlate, a los informes anteriores a la salida al mercado, las noticias de actualizaciones- todo.

NOTA: El mercado objetivo del iPad NO recoge al usuario Mac. Tampoco es el usuario medio de iPhone.

Lo crea o no, el diseño del iPad está dirigido a usuarios de más edad que son nuevos en el mundo informático y no quieren gastar más de $1.000. Son gente que sólo saben encender y apagar el ordenador y usar email. El iPad también está dirigido a estudiantes jóvenes, en un intento de superar el Kindle. Uno de los principales fines de iPad es convertir en obsoleta la compra de libros de texto ofreciendo una librería digital, del mismo modo que iTunes ofrece música. Ahí lo tiene: las generaciones de usuarios viejos y jóvenes son el objetivo de mercado, en contraposición al grupo de informáticos estándar que normalmente compra dispositivos de alta tecnología.

En este libro le presento los conocimientos y técnicas necesarios para crear aplicaciones que funcionen en el iPhone y el iPad. Las imágenes de aplicaciones en el libro incluyen el simulador de iPad, que implica inherentemente el simulador de iPhone. Asegúrese de comprobar las diversas figuras que contienen imágenes de resultados y estudiar las diferencias entre el iPad y el iPhone. Es importante que sepa en qué difiere el iPad del iPhone.

¿Funcionan las aplicaciones tanto en el iPad como en el iPhone?

Todas las aplicaciones ilustradas funcionan tanto en el iPad como en el iPhone. Lo sé porque yo mismo la he probado –no solo en los simuladores, también en los dispositivos reales. Algunas de las imágenes podrían haberse hecho con archivos más grandes (y mayor resolución) si hubieran estado codificadas exclusivamente para el iPad, pero decidí que fueran compatibles con ambos dispositivos.

El mejor modo de saber si una aplicación que hagas funciona tanto en el iPhone como en el iPad es simplemente conectar su ipad a su Mac; iTunes mostrará las aplicaciones que funcionan en el iPad (Figura 3-8).

Figura 3–8. Muestra las aplicaciones para iPhone del autor que iTunes dice que son compatibles con iPad.

NOTA: Algunas de las aplicaciones del iPad—en vista normal—parecen un poco borrosas; es porque las imágenes se han estirado más del 100% de su tamaño ideal. Para los que estáis interesados en programar exclusivamente para iPad tendréis que aseguraros de que la resolución de la imagen se queda al 100% o por debajo.

Más espacio de pantalla

Las dimensiones físicas del iPad son algo más del doble de la altura del iPhone y ligeramente más del triple de ancho, aunque el grosor únicamente es de 1,1 milímetros más. El resultado es aproximadamente un incremento de 6 veces el área de pantalla y prácticamente la misma profundidad, por tanto es realmente un dispositivo mucho mayor, pero que aún así mantiene una estructura realmente delgada y ligera. Las dimensiones físicas, sin embargo, no son las mismas que las dimensiones de visualización.

NOTA: El display del iPhone contiene 320 x 480 píxeles; el display del iPhone tiene 768 x 1024 píxeles. Esto le confiere al iPad un espacio utilizable casi cinco veces superior al del iPhone.

Este espacio extra tiene implicaciones importantes. La más obvia es que se puede presentar más información al usuario en un momento dado. Esto significa que la perspectiva organizativa que usan los programadores en las aplicaciones existentes para el iPhone no siempre se pueden traducir directamente a la nueva y relativamente expansiva plataforma. También significa que la migración del iPhone al iPad implica classes –subrutinas preprogramadas- que no son compartidas por los dos dispositivos, y lograr esto puede requerir cierta especialización en el interior de las aplicaciones. Los cambios en el manejo de la interfaz de usuario cambian y las diferencias en la Interfaz de Programación de Aplicaciones (API) serán tratadas más tarde. Ahora consideremos las implicaciones del incremento de desarrollo que ostenta el iPad.

El iPad puede ejecutar aplicaciones del iPhone sin cambiar una coma. Es más, el iPad puede ejecutar estas aplicaciones al doble de su tamaño original usando una función lupa. Esto hace el que proceso de migración sea sencillo para los actuales desarrolladores de iPhone. Sin embargo, estamos tratando con algo que es mucho más que un iPhone sobre dimensionado. Hay muchas mejoras legítimas sobre el iPhone en el nuevo dispositivo –no solo un aumento de tamaño, también de cuantía. Por esta razón tenemos que maximizar estas mejoras y deslumbrar a los futuros usuarios.

Más espacio de pantalla significa más información y más regalo para la vista. Un usuario puede ahora ver muchos datos al mismo tiempo sin tener que navegar por la aplicación. Más espacio para videos, fotos, texto o gráficos de juegos significa más flexibilidad y longevidad para tu aplicación. La estructura del iPad soporta todas las características del iPhone incluyendo UIKit, Core Graphics, Media Kit y OpenGl ES. Sencillamente, son herramientas con las que se pueden construir un montón de aplicaciones. El incremento en potencia de proceso y la pantalla más grande significan que el iPad –y por tanto tus aplicaciones- pueden hacer significativamente más con el mismo conjunto de herramientas.

Este incremento en área de pantalla, con alta resolución mejorada, sensores multi-touch, también significa más espacio para que el usuario interactúe con la aplicación – con una articulación más fina. Ahora se pueden usar gestos complejos para navegar y manipular una aplicación de modos imposibles en la pantalla relativamente pequeña del iPhone. No te olvides de esto cuando tu aplicación vaya tomando forma.

NOTA: Los usuarios ahora pueden usar la mano completa, e incluso ambas manos para interactuar con elementos de una aplicación.

¿Qué gestos tienen aquí sentido? ¿Qué clase de imputs pueden provocar esto o eso? Además de la mayor capacidad de soportar entradas con gestos clave que representan atajos, el mayor tamaño de la pantalla posibilita la vista de barras de tareas y accesorios. Suministrar al usuario una barra de tareas ya no es tabú, y en realidad se podría fomentar para ayudar a los usuarios a beneficiarse de la interfaz aumentada del iPad.

Cuando se toman estos cambios de forma colectiva, nos lleva a considerar cuidadosamente los diseños de nuestras aplicaciones. La creatividad que usted y sus compañeros programadores ejercen, no sólo debe beneficiarse de la habilidad de mostrar cantidades de información mayores en un formato intuitivo y fácil de leer, debe estar al borde de una reorganización total y una transformación de la experiencia del usuario.

Master-Detail

Una de las características principales de iPad que no se encuentran en el iPhone es la interfaz Master-Detail. Lleva un tiempo en los sistemas operativos Mac, y ahora ha florecido en una forma nueva. El master-detail es una capa de interfaz que se sitúa en la parte superior de una lista o una tabla y que actúa como paso intermedio en lugar de un botón “adelante” o “atrás”. En el iPhone, se puede ver el patrón principal (Master) o el detalle (Detail), uno cada vez, pero nunca los dos al mismo tiempo.

Figura 3–9. Fíjese como la capa master-detail queda sobre el email en la orientación vertical y al lado en la orientación horizontal.

Esta capa de doble propósito permite al usuario, por ejemplo y en el caso de los emails, trabajar con el correo electrónico utilizando la Bandeja de Entrada como capa principal (Master) y el email en segundo plano (Detail). Como se puede ver en esta imagen de Apple (Figura 3-9), el iPad presenta el patrón Master-Detail en cualquier orientación.

Interfaz de usuario en el iPad

Algunas aplicaciones iPhone se escalan perfectamente cuando se ejecutan en iPads. Sin embargo, estas aplicaciones son probablemente minoría ya que las capacidades de entrada de datos y el espacio de visor del iPad cambian completamente el carácter fundamental de una aplicación. Para arreglar esto, necesitamos usar un diseño de interfaz de usuario especializado para presentar la aplicación en ambas plataformas.

NOTA: Advierta que aunque el iPad puede agrandar imágenes del iPhone, mis pruebas de las aplicaciones del iPad (como el 3 de abril del 2010) demostraron que las imágenes grandes no se convierten con fluidez.

Los nuevos elementos de interfaz introducidos con el iPad permiten un flujo de datos y una interacción con la aplicación altamente estilizada y potente. Los genios de Apple han hecho la mayor parte del trabajo por usted, suministrando unos elementos de interfaz increíblemente ricos y fáciles de usar, que te permiten centrarte en la funcionalidad de la aplicación.

Comprobando la plataforma

Todo esto es perfecto, pero obviamente hay cosas que el iPad puede hacer y el iPhone no puede –y viceversa. Ya sabemos que hay classes que usa el iPad que el iPhone no reconoce. Por tanto ¿qué hay que hacer para que nuestra aplicación se comporte correctamente dependiendo del dispositivo en el que se ejecuta? Esto no es, desafortunadamente, tan sencillo como quisiéramos, pero se puede hacer.

Por esta razón, no nos vamos a preocupar de comprobar la plataforma actual. En lugar de eso, simplemente cambiaremos los archivos nib para que hagan lo que necesitamos que hagan en el iPad. No se preocupe, -los nibs y el Interface Builder funcionan exactamente de la misma forma para el iPad y el iPhone.

De todas formas, si es absolutamente necesario que sepas que existen varias técnicas que se usan para comprobar la plataforma actual de una aplicación. Dichas técnicas implican la comprobación de la existencia de clases de plataformas específicas, funciones y características. Por ejemplo, podría comprobar la anchura de la pantalla para determinar la plataforma. Una aplicación para iPhone devolverá una anchura de pantalla de 320 pixels, mientras que una aplicación escrita para el iPad devolverá una anchura de pantalla de 768 pixels. Ciertamente, si la anchura de pantalla devuelta es menor de 768, sabemos que estamos trabajando para un iPhone o, quizás, un iPod Touch. Por supuesto, hay otras formas de comprobar la plataforma presente. También puede comprobar una función llamada `UI INTERFACE IDIOM ()`. Esta función puede devolver bien `UIInterfaceIdiomPhone` o `UIInterfaceIdiomPad`**.** Estos ejemplos se explican por sí mismos bastante bien, espero, y se pueden usar para determinar el dispositivo con precisión. De las muchas otras técnicas para determinar la plataforma actual, aprenderemos a usar la que nos convenga más en un contexto dado.

NOTA: Como se dijo anteriormente, cada dispositivo puede hacer cosas que el otro no puede. El iPad no puede hacer llamadas telefónicas, fotografías o grabar video. Caso a caso se debería pensar en aplicaciones que trataran de servicios de localización o funciones de medición de aceleración –porque aunque ambos dispositivos son portátiles y se pueden mover, el tamaño puede importar.

Compatibilidad

La salida del iPhone OS 3.2 ha cambiado muchas de las funcionalidades básicas a las que estábamos acostumbrados con nuestros iPhone iPod Touch. Sin embargo, no todo el mundo usa el 3.2, por tanto debemos asegurarnos de que nuestras aplicaciones son compatibles con versiones previas, específicamente 3.1.2 y 3.1.3. Por suerte, hacerlo es bastante simple. En nuestro nuevo proyecto para iPhone OS 3.2, solo necesitamos poner el "Base SDK" de los ajustes de nuestro proyecto a 3.1.2 o 3.1.3. y ejecutar la aplicación.

NOTA: Como todos sabemos, la desventaja de los rápidos avances tecnológicos es que siempre debemos de estar preparados para cualquier actualización. Por tanto, independientemente de cualquier compatibilidad presente, hemos de tener en cuenta que se producen actualizaciones cada 6 meses aproximadamente, por lo que las instrucciones deberán seguir funcionando al actualizar el SDK a la versión más reciente.

Si quiere ejecutar la aplicación en el simulador de iPad para ver qué aspecto tiene, haga clic en el botón emergente del ángulo superior izquierdo del Xcode. Desde ahí, escoja iPhone Simulator 3.2. (o superior). Esto llevará automáticamente la aplicación al simulador de iPad. Mientras la aplicación se ejecuta en el simulador de iPad, puede hacer zoom pulsando el botón 2x en la esquina inferior izquierda del simulador. Adicionalmente, puede hacer más zoom abriendo la ventana de menú, pasando el razón por el elemento Scale (escalar), y eligiendo la escala deseada.

§III: UNA MIRADA POR DENTRO

Hasta ahora hemos visto tres formas por las cuales podemos decir "¡Hello World!" desde el iPhone/iPad. El primer ejemplo desde la plantilla de aplicaciones basadas en vista (View-Based Aplication); los dos siguientes ejemplos (relacionados) se construyeron desde la plantilla de aplicaciones basadas en navegación (Navigation-Based). Lo más importante, hemos tenido la oportunidad de jugar con Xcode, Interface Builder y los simuladores de iPhone e iPad.

Sobre todo, el aspecto más importante del capítulo 2 era familiarizarte con el proceso creativo en el contexto de la programación de aplicaciones: empezar con una idea y acabar con un producto tangible y funcional. Varias veces te pedí que ignoraras el código pesado que pensé que le resultaría molesto o desalentador. También puede que hayas notado que cuando intentabas entender algo de este código espeso, tenía sentido de un forma extraña, caótica y maravillosa. Bien, según vayamos progresando, haremos del "caos" de lo desconocido, algo menos inquietante.

Antes de tratar este tema, permíteme aconsejarte el que te lo tomes con calma cuando lleguemos al Objetive-C, nuestro lenguaje de programación, ya todavía estoy por encontrar un programador avanzado que conozca todos los símbolos y comandos. Como en otras industrias, la gente tiende a adquirir muchos conocimientos en sus dominios específicos y a especializarse (por ejemplo, integrar Google Maps en un juego o una aplicación).

Una analogía que me gusta es esta: los mecánicos de coches solían ser capaces de desmontar un motor completamente y luego volver a montarlo –presumiblemente mejor de lo que estaba. Hoy en día, los mecánicos de coches están muy especializados, y sólo unos pocos saben como desmontar completamente y reconstruir un coche moderno específico. Nos hacemos expertos en motores híbridos de Ford, o en circuitos eléctricos de Toyota Prius, en los frenos de tambor que frenan trailers y así sucesivamente. ¡No hay nada malo en ello!

Esto es parecido a cómo estamos actuando. Acabamos de ensuciarnos las manos programando tres aplicaciones con *éxito*. Y ahora, si todo va de acuerdo al plan, caminaremos hacia un futuro rebosante de seguridad. Sé por experiencia que, en mis estudiantes, esa seguridad se puede desbaratar si están intimidados o abrumados por demasiada complejidad o detalles técnicos. He descubierto que los estudiantes pueden superar baches si saben dónde van y si saben que los tramos difíciles no se volverán demasiado peligrosos ni asustarán demasiado.

Ha dicho "¡Hola!"… ahora diremos, ¡INDIO!

Podemos dividir la mayoría de las aplicaciones para iPhone e iPad en cuatro diferentes funciones: **I**nteracción, **N**avegación **D**atos, e **I/O** (entrada/salida) (Interaction, Navigation, Data e I/O Imput Output). Hemos visto suficientes aplicaciones para saber que podemos interactuar con ellas; podemos navegar de una pantalla a otra; podemos manipular y usar datos; y podemos suministrar imputs (entrada) (teclear, pegar, hablar) y recibir outputs (imágenes, sonidos, textos, ¡*diversión*!)

Antes de que nos acerquemos otra vez a abordar un programa desde una de estas áreas específicas, tenemos que tener un mejor conocimiento de cómo funcionan los diferentes aspectos de la programación de iPhone/iPad, qué aspecto tienen y cómo se comportan. También tenemos que aprender sus limitaciones y los pros y contras en términos de la experiencia de usuario deseada o proyectada. Y, a causa de las diferencias tratadas arriba, si la aplicación es para el iPhone o el iPad.

Según ganas conocimientos elementales de dónde están las barreras y limitaciones, tu viaje a través de estos cuatro dominios, todos ellos parte de un vasto "bosque", serán más potentes y productivos. Una parte importante de mi trabajo es mostrarte cómo dirigirse sin peligro a través del Bosque INDIO. Algunas secciones del bosque son más desalentadoras que otras, pero la buena noticia es que conseguirás una bonita vista, desde un nivel alto, como desde un helicóptero. Después de nuestro tour aéreo, nos tiraremos en paracaídas al suelo del bosque, abriremos el Xcode y continuaremos explorando las sendas, los agujeros llenos de agua, y los atajos –para tachar las secciones innecesarias y estar al acecho de animales salvajes.

Modelo-Vista-Controlador

Como se mencionó previamente, los programadores que desarrollaron Cocoa Touch usaron un concepto conocido como Modelo-Vista-Controlador (MVC) como base para el código para las aplicaciones de iPhone e iPad. Esta es la idea básica.

Modelo: Este contiene los datos y las clases que hacen que la aplicación funcione. Es la parte del programa donde podrías encontrar secciones de código que te dije que ignoraras. Este código también puede contener objetos que representen elementos que podrías tener en tu aplicación (por ejemplo pinballs, dibujos animados, nombres en bases de datos, citas en tu calendario).

Vista: Esta es la combinación de todas las cosas buenas que los usuarios ven cuando usan tu aplicación. Es cuando el usuario interactúa con botones barras de desplazamiento, controles y otras experiencias que pueden sentir y apreciar. Aquí puedes tener una vista principal formada por un número de vistas diferentes.

Controlador: El controlador conecta el modelo y la vista sin perder de vista lo que el usuario está haciendo. Piensa en ello como en el plan estructural –la espina dorsal- de la aplicación. Es como coordinamos los botones pulsa el usuario y, si es necesario, como cambias una vista por otra, todo en respuesta a los datos, reacciones, entradas etc. del usuario.

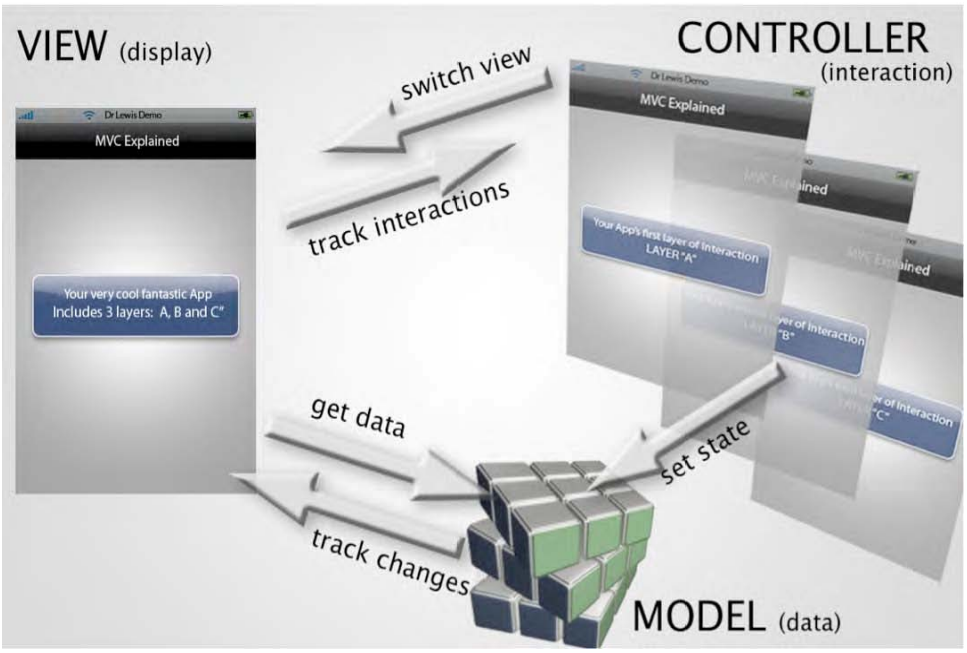


Foto 3-10. El Modelo, la Vista y el Controlador (MVC).

Considera el siguiente ejemplo como ilustrativo de cómo puedes usar el concepto de MVC para dividir la funcionalidad de la aplicación para iPhone/iPad en tres categorías distintas. La figura 3-10 muestra una representación de su aplicación; la he llamado "MVC Explicado". Podrás ver que la VISTA muestra una representación –una etiqueta- de "Su fantástica aplicación tiene 3 capas: A, B y C".

En la sección CONTROLADOR de la aplicación, vemos las tres capas individuales por separado, Capa "A", Capa "B" y Capa "C". Dependiendo de qué mecanismo de control clique el usuario en el domino VISTA, el visualizador que aprecia el usuario, el CONTROLADOR devuelve la respuesta apropiada –la vista siguiente de las tres capas preparadas.

Tu aplicación probablemente usará datos de algún tipo, y esta información se almacenará en la sección MODELO de su programa. Los datos podrían ser números de teléfono, puntuaciones de jugadores, localizaciones de GPS en un mapa, y así sucesivamente.

A medida que el usuario interactúa con la sección VISTA de la aplicación, podrá recuperar datos de su base de datos. Digamos que sus datos contienen el lugar donde el usuario aparcó el coche. Cuando el usuario pulse un botón concreto del programa, podría recuperar los datos GPS del MODELO. Si es un objetivo en movimiento, podría rastrear los cambios en la posición del usuario en relación a un coche en el aparcamiento. Finalmente, el CONTROLADOR podría cambiar el estado (o modo) de sus datos. Un estado podría mostrar números de teléfono, mientras que otro podría mostrar posiciones GPS o las diez puntuaciones más altas de un juego. El CONTROLADOR es también el lugar donde ocurren las animaciones. Lo que pasa en la animación puede afectar y quizá cambiar el estado del Modelo. Esto se podría hacer usando varias herramientas, como objetos UIkit, para controlar y animar cada capa, estado, etc.

Si esto suena complicado, recuerda que ya has hecho mucho de esto sin ni siquiera saberlo. En el ejemplo 1, hiciste que el usuario apretara un botón y emergió una etiqueta que decía “¡Hello World!” Esto demuestra que ya has construido una interacción con un ViewControler (Controlador de vista). Profundizaremos en estas posibilidades, por supuesto. En el capítulo 4 nos aventuraremos con más profundidad en el cuadrante de **I**nteracción del Bosque INDIO, y permitiremos al usuario añadir y borrar elementos de vista de tabla.

Cuando hagamos esto, haré lo posible por mantenerte centrado en el cuadro global en relación a las interacciones… via **N**avegación. Nuestro objetivo será que el usuario avance de información menos específica a más específica con cada vista nueva.

Capítulo 4

Una Introducción al Código

Antes de centrarnos en partes específicas del código y en la manera en que crearemos las aplicaciones recogidas en este capítulo, necesitamos dedicar un momento a la evaluación de nuestro enfoque. En este capítulo entraremos en material de un modo algo diferente a los capítulos anteriores.

He incluido pequeñas reseñas a cada paso, pero sin hacer detalle de las instrucciones previas. Encontrarás muchas similitudes ente las funciones y métodos que vamos a emplear y de las que ya hemos hecho uso. Es perfectamente comprensible el que tengas que volver hacia atrás y repasar instrucciones detalladas con anterioridad. El no redundar en ellas es en aras de la racionalización y el ahorro de espacio, haciendo de este libro un manual más manejable.

Otra diferencia que encontrarás es que, después de que hayas completado el programa que vamos a acometer, haremos una revisión del código. Esta sección la encontrarás al final de cada uno de los ejemplos más importantes de este libro y la sección se recogerá bajo el nombre “Ahondando en el Código”. En estos análisis, volveremos sobre conceptos ya escritos, ahondando y explicando procesos con un mayor nivel de detalle. Allí comentaremos a presentar términos más técnicos, los cuales se irán aplicando en futuros capítulos y para poder comunicarnos con otros programadores.

Siguiendo el ejemplo utilizado al principio del libro: Hasta el momento, te he enseñado a montarte en el coche, arrancarlo, pisar el acelerador y a girar el volante a medida que avanzamos. En este capítulo te guiaré de un modo parecido, pero con un modelo de automóvil diferente –con un menor número de instrucciones detalladas. Después de que lleguemos a nuestro destino, abriré el capó y te mostraré cómo, cuando pisas el acelerador, entra gasolina al motor, o verás cómo se accionan las pinzas al pisar el pedal de freno. En el Capítulo 8, analizaremos la cantidad de gasolina que entró en el motor al pisar el acelerador, y el cómo se calientan las pastillas de freno al accionar el pedal a una determinada velocidad, y así sucesivamente. Por supuesto, confío en que serás capaz de manejarlo!

Pasemos a nuestra siguiente aplicación. Estate preparado para volver hacia atrás conmigo y repasar determinadas líneas tan pronto nos vayamos centrando en ciertas partes del código, y así veremos como funciona todo en conjunto.

004_helloWorld: Botones con gráficos

Como indicamos en el Capítulo 3, nos vamos a embarcar en un viaje en el cual exploraremos las cuatro categorías de las aplicaciones iPhone e iPad, abreviadas bajo el término INDIO: Interaction, Navigation, Data, e I/O. Nuestro cuatro ejemplo es muy similar a las sencillas aplicaciones “Hello World!” creadas en capítulos anteriores, a excepción de que en este ejemplo se muestra como hacer algo que no es tan intuitivo como pueda parecer a primera vista – cómo manejar botones y gráficos.

Una de las maneras que tengo de asegurarme el aprendizaje de todos los nombres de mis estudiantes es el tener fotografías de todos ellos en uno de los ejercicios. A los estudiantes les pido que incluyan sus fotos como fondo en una aplicación “Hello World!”. Invariablemente todos lo ven como un ejercicio muy sencillo. Luego resulta que no solo existen problemas con el tamaño y la colocación de las imagines, sino que las mismas también tapan el código una vez que son colocadas sobre la pantalla.

En este proyecto, he creado una imagen tal y como les pido a mis estudiantes que hagan, y la he colocado como un wallpaper. Así que cuando colocan sus caras en el lugar apropiado, sus fotos me dicen “Hello” y me dicen quiénes son. Este ejercicio va paso a paso a través de este proceso, por lo que puedes hacer lo mismo con una foto tuya. Se complicará en algunos momentos, por lo que sigue el desarrollo cuidadosamente. Intenta no caer en la trampa de saltarte pasos pensando que tienes cierta práctica, ya que puedes acabar algo confuso si no sigues la explicación de un modo lineal.

Toma aire, relájate, y vamos paso a paso. Aprenderás a utilizar algunas herramientas muy útiles en este ejercicio. Todo lo que necesitas está en este libro: pantallazos e instrucciones paso a paso. Quiero recordarte que todos los pantallazos los tienes en mi screencrast disponible en:

```
http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/002_helloWo
rld_002.htm
```

1. Como siempre, empezamos con un escritorio, limpio. Cierra todos los programas usando las combinaciones Command + Tab y Command + Q hasta que sólo tengas en pantalla el Finder.

   Verás que he preparado una imagen mía como ejemplo para esta aplicación. Por tanto, cada vez que veas mi foto en el libro es como si vieses tu propia foto. Usa el programa que quieras para crear una imagen de 320 píxeles de ancho por 480 pixels de alto. Si no está conforme de un modo exacto tal y como se muestra en la Foto 4–1, tu iPhone puede mostrar efectos poco realistas, pudiendo incluso darse el caso de que no muestre la imagen. Por experiencia, limítate a trabajar con una imagen de exactamente 320 × 480 píxeles. Mas tarde, cuando tengamos barras en la parte superior, todo cambiará. Por el momento grábate en la cabeza el trabajar con formato de 320 × 480.

Foto 4–1. Crea una imagen tuya en un editor de imagen y guárdala bajo formato .png.

Como puedes ver en este pantallazo, he usado Photoshop. Puedes copiar todos los parámetros que he utilizado repasando la Foto 4-1. De este modo te asegurarás que tu imagen se ajustará de un modo correcto a la aplicación. Guarda el archivo de imagen bajo la extensión `.png`, llamándolo `helloworld.png`. Lo guardamos bajo esta extensión ya que los iPhones trabajan con extensiones `.png` —y no con extensiones tipo .jpg o similares. No te cuestiones esto. Simplemente acéptalo. Necesitas un programa que pueda crear, editar y guardar archivos `.png` .

Si no tienes Photoshop, o alguna herramienta de edición similar, y planeas programar con gráficos para dispositivos iPhones/iPad, te recomiendo el que te hagas con una copia. Una vez has creado tu imagen y la has guardado en el escritorio, cierra Photoshop (o programa equivalente). Llegado a este punto has de tener un escritorio vacío, a excepción de esta imagen creada.

2. Comencemos abriendo una Aplicación tipo View-based en Xcode. Abre Xcode, y presiona ⌘⇧N para abrir un archivo de Nuevo programa. En la Foto 4–2, puedes ver mi icono Aplicación View-based marcado por defecto. Si el tuyo no lo está, haz clic sobre el mismo.

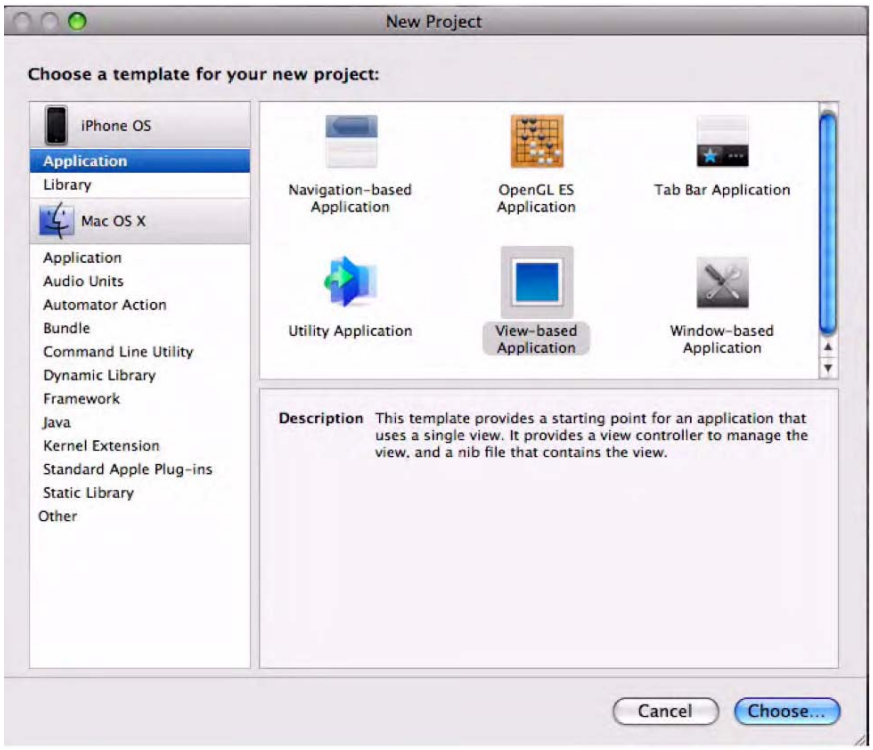
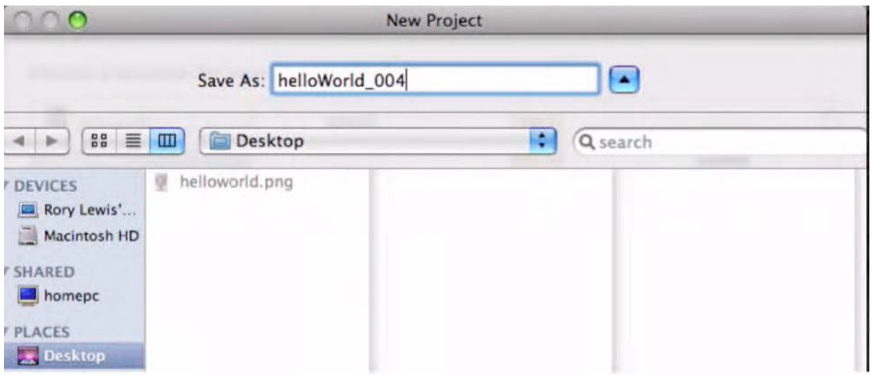


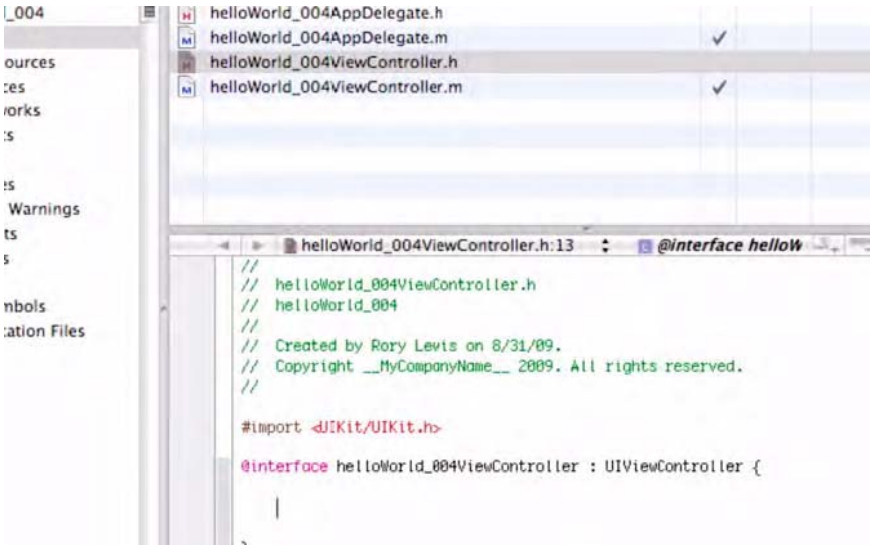
Foto 4–2. Después de presionar ⌘⇧N, crea una Aplicación View Based.

3. Guarda tu Aplicación View-based en el escritorio como `helloWorld_004`. (Foto 4-3)

4. Como se muestra en la Foto 4–4, una vez guardes tu proyecto en el escritorio, Xcode inicializa (instantiates) un proyecto bajo el nombre `helloWorld_004`, como podrás comprobar en la cabecera de la ventana. (Si no estás familiarizado con el termino "instantiate", echa un vistazo al apartado "Ahondando en el Código" al final de este capítulo.) Pinchando en el archivo raíz d tu `helloWorld_004`, encuentras el archivo `plist`, los ficheros `AppDelegate`, y los ficheros `ViewController` . Estos archivos van a ser tus compañeros, y conforme nos adentremos en este Nuevo ejemplo, te familiarizarás con ellos más y más.



F oto 4–3. Guarda tu Aplicación View-based en tu Escritorio con el nombre as helloWorld_004.



F oto 4–4. Así se ve tu archivo helloWorld_004ViewController antes de que introduzcas Nuevo código.

Repasando los ficheros, vemos que algunos terminan con la extensión `.h` y otros lo hacen con la extensión `.m`. La extensión `.m` representan los archivos de *Implementación* (*Implementation files*), y los archivos de extensión .h representan los archivos de *Cabecera* (*header files*). A esta altura, deberías saber que los Header files contienen classes, types, functions y contast declaration. Por el contrario, los Implementation files contienen las líneas que hacen uso de material precodificado. En este ejercicio vamos a trabajar con los archivos de extensión `.h`. Por ahora, esta información es suficiente, pasemos al siguiente paso.

5. Tal y como hicimos en el primer ejercicio, después de abrir el archivo de interface, nos desplazaremos hacia abajo en la pantalla hasta situarnos por debajo de #import <UIKit/UIKit.h>, comando por el cual importamos el UIKit framework. Sigue desplazándote hasta colocarte debajo de

```
@interface helloWorld_004ViewController : UIViewController {
```

y entonces presiona Intro para hacer una línea de espacio para el código que vamos a introducir en el siguiente paso.

6. Como antes, añadir debajo de la línea que queda debajo del corchete abriente (el carácter {) el siguiente comando:

```
IBoutlet UILabel *label;
```

Esto añade un Outlet a la etiqueta (label), justo como hicimos en ejemplos anteriores. No olvides el punto y coma (;) al final de la línea. Como sabes, el punto y coma le dice al compilador que has terminado de" hablar" por el momento.

7. Ahora necesitamos añadir una acción al botón. Introduce el corchete cerrante (}) y escribe lo siguiente:

```
-(IBAction)hello:(id)sendr;
```

Asegúrate de colocar el punto y coma al final de la línea. En el siguiente código, podrás apreciar cómo ha de quedar tu archivo.

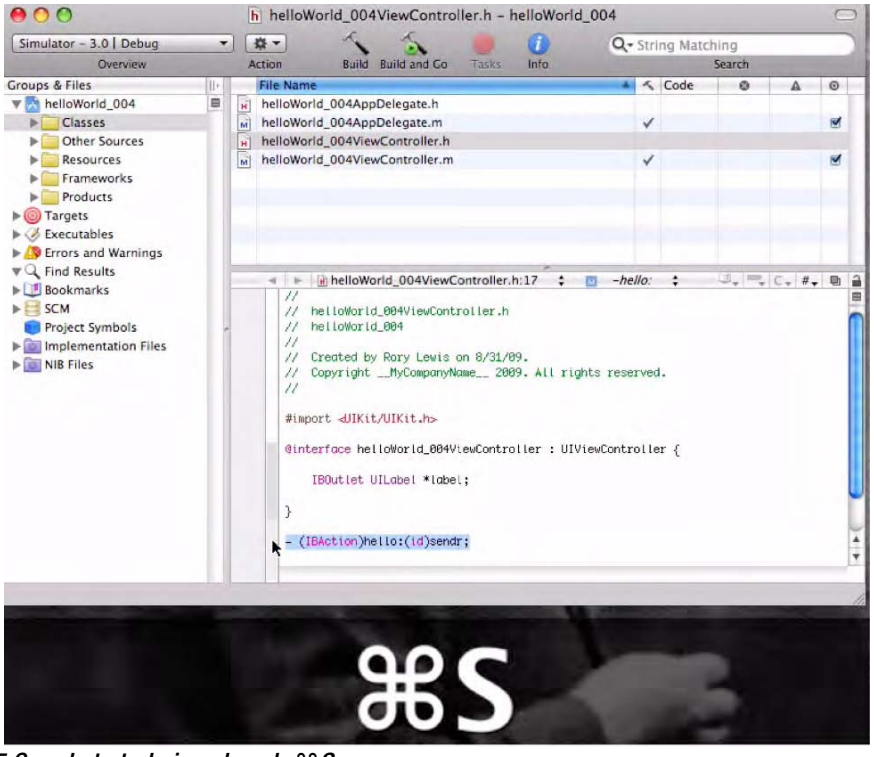
```
//
// helloWorld_004ViewController.h
// helloWorld_004
//
// Created by Rory Lewis on 6/13/09.
// Copyright __MyCompanyName__ 2009. All rights reserved.
//
#import <UIKit/UIKit.h>
@interface helloWorld_004ViewController : UIViewController {
      IBOutlet UILabel *label;
}
-(IBAction)hello:(id)sendr;
@end
```

Después de seleccionar la línea

```
-(IBAction)hello:(id)sendr;
```

Cópiala subrayándola y usando la combinación de teclas ⌘C.

8. Guarda tu archivo pulsando ⌘S. Foto 4–5.



F oto 4–5.Guarda tu trabajo pulsando ⌘S.

9. Ahora que hemos terminado con nuestro fichero header, pasamos a nuestro archivo de Implementación: `helloWorld_004ViewController.m`. Este archivo gestiona cómo tú código interactúa con el display. Abre este archivo, desplázate hacia abajo pasadas las líneas verdes pertenecientes a los comentarios e inserta la línea que acabas de copiar, colocándote en el sitio correcto y utilizando la combinación de teclas ⌘V.

```
#import "helloWorld_004ViewController.h"

@implementation helloWorld_004ViewController
*** THIS IS WHERE YOU PASTE ***
```

10. Borra el punto y coma del final de la línea que acabas de pegar y reemplázalo con un corchete abriente tal y como se muestra en la Foto 4–6.

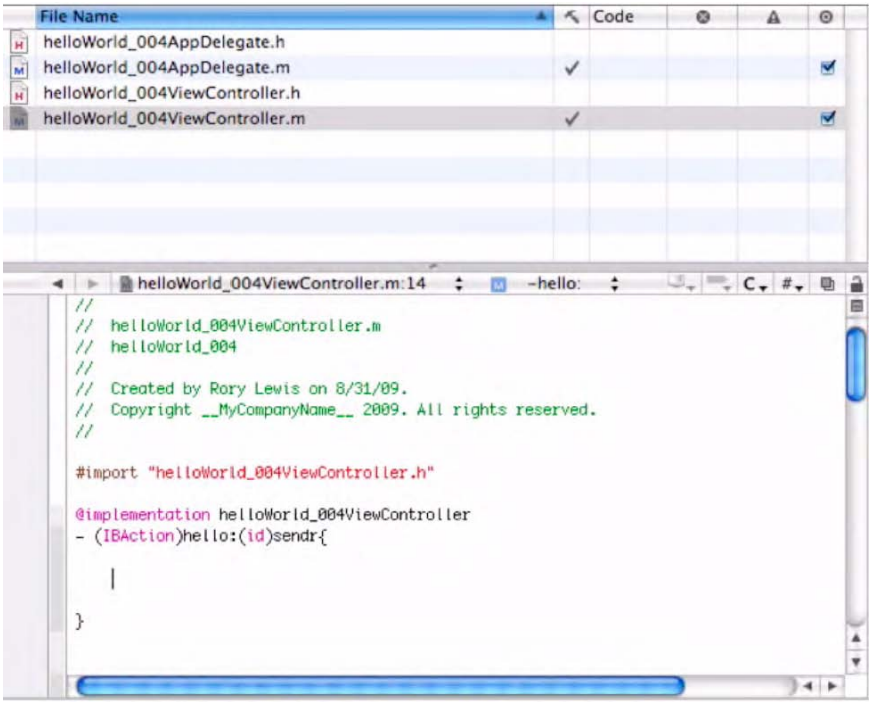


Foto 4–6. Este es el archivo de Implementación después de insertar la línea de código que copiamos e introduciendo el corchete abriente "{" en lugar del punto y coma que hemos eliminado

```
#import“helloWorld_004ViewController.h"

@implementation helloWorld_004ViewController

-(IBAction)hello:(id)sendr{
```

11. Después de borrar el punto y coma y reemplazarlo con el corchete abriente, ve a la siguiente línea e introduce este código

```
label.text = @"OK I'm repeating myself!";
```

seguido de un corchete cerrante en la línea de abajo. Ver Foto 4–7.

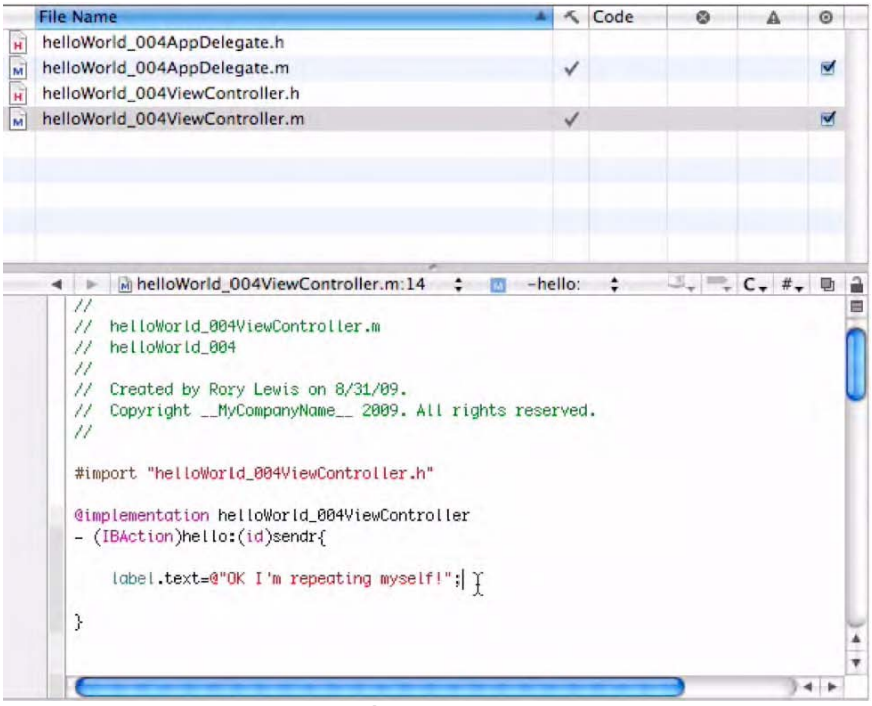
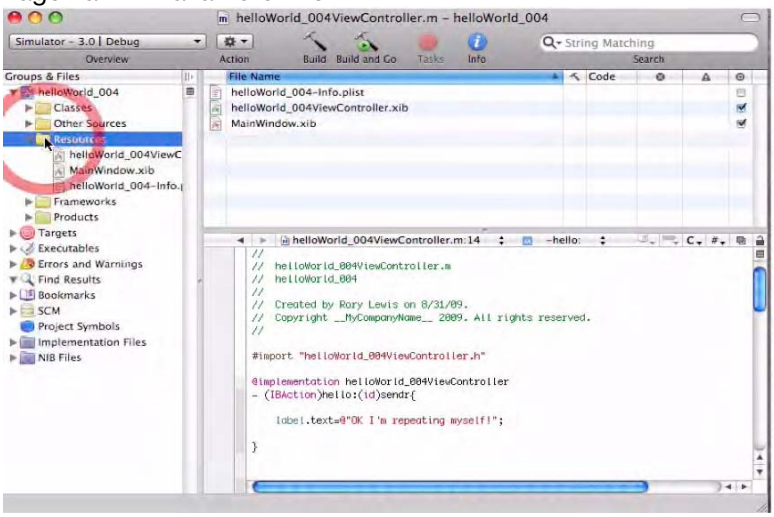


Foto 4–7. Introduce el comando que generará el texto "OK I'm repeating myself!" Ahora, guarda tu archivo usando la combinación de teclas correspondiente

```
#import "helloWorld_004ViewController.h"

@implementation helloWorld_004ViewController
-(IBAction)hello:(id)sendr{
label.text = @"OK I'm repeating myself!";
}
```

12. Ahora, guarda tu trabajo (⌘S).

13. Clica en el directorio `Resources`**.** Quiero que lo actives de modo que puedas guardar tu imagen ahí. Mira la Foto 4–8.



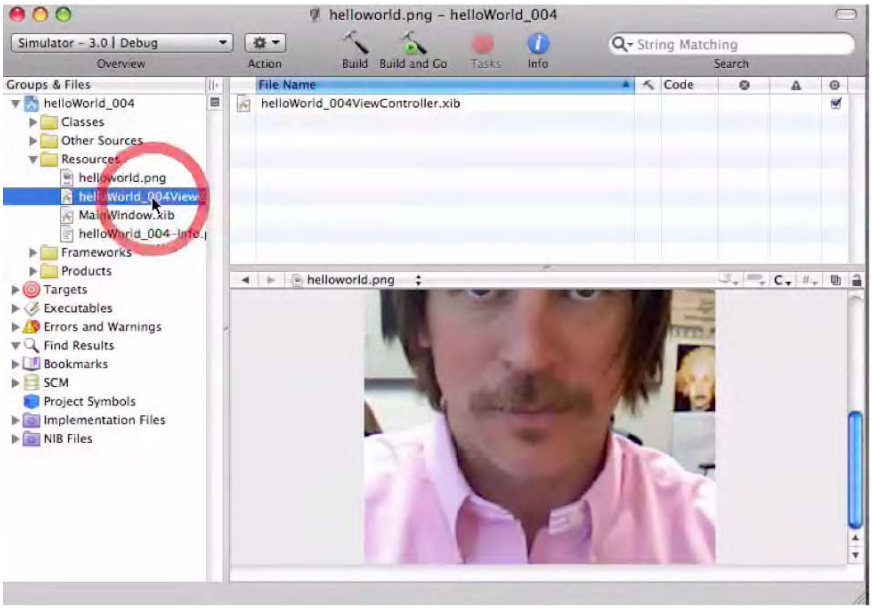
F oto 4–8. Clica en la carpeta Resoruces.

14. Localiza tu imagen `helloworld.png` en el escritorio y arrástrala hasta la carpeta Resources. Me gustaría que tomases el hábito de guardar tus imágenes en esta carpeta. Es un buen hábito que te ayudará a tenerlo todo organizado. Cuando empieces a trabajar con varias imágenes, probablemente querrás crear una carpeta dedicada a las mismas, asignándole por ejemplo el nombre de `Images`. Por el momento, vamos a ir colocando las imágenes en la mencionada carpeta `Resources`.

15. Marca la opción que reza: "Copy items into your destination group's folder.". De esta manera nos aseguramos que la imagen siga apareciendo en tu aplicación en caso de que cambies la localización del directorio o borres la imagen, ya que Xcode hará una copia de tu carpeta `Resources`.

Por ejemplo ahora tenemos la imagen guardada en el escritorio. Si la borrásemos o mandásemos este directorio por internet o a un iPhone sin marcar la casilla comentada, dejaría de ser parte de tu programa. Marcando esta opción garantizamos que la imagen `helloworld.png` que estamos guardando en el directorio `Resources` irá donde el programa vaya. Para completar este paso, clica el botón Add o presiona Intro.

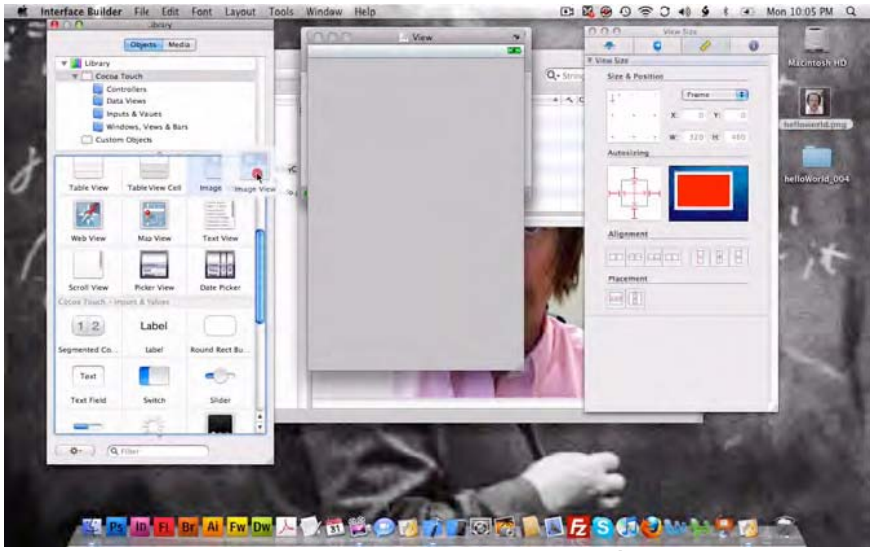
16. Una vez hayas abierto la carpeta `Resources`, haz doble clic en el archivo nib, tal y como se muestra en la Foto 4–9. Como ya sabrás, este es el archivo que tiene la extensión `.xib`—concretamente, el archivo `helloWorld_004ViewController.xib`—abriendo el archivo nib automáticamente se abre el Interface Builder, donde vamos a poder controlar y visualizar nuestra aplicación. Hacemos esto porque ha llegado el momento de conectar las fotos, botones y etiquetas al código que hemos escrito.



F oto 4–9. Para abrir el Interface Builder, clica en el archivo nib.

17. Pretendemos tener un fondo de pantalla (wallpaper) sobre el cual colocar los botones y etiquetas incluidos en nuestra aplicación. Esto significa que necesitamos un lugar donde pueda residir esta imagen. En el Interface Builder, puede colocar imágenes en Image Views. Desplázate hacia abajo y encontrarás el icono Image View. Arrastra la imagen al View frame, como se muestra en la Foto 4–10.

18. Queremos conectar nuestro `helloworld.png` con nuestra Imagen de vista de modo que la misma aparezca. La forma de conseguirlo es ir al apartado Information de la ventana File Attributes, abrir la ventana desplegable y seleccionar nuestra imagen.



F oto 4–10. Arrastra el icono Image View en nuestro View frame vacío.

Necesitamos asegurarnos de que nuestra imagen, `helloworld.png`, aparece en nuestro View frame. Si la imagen que hemos preparado no se muestra correctamente, necesitamos parar y repasar los pasos anteriores.

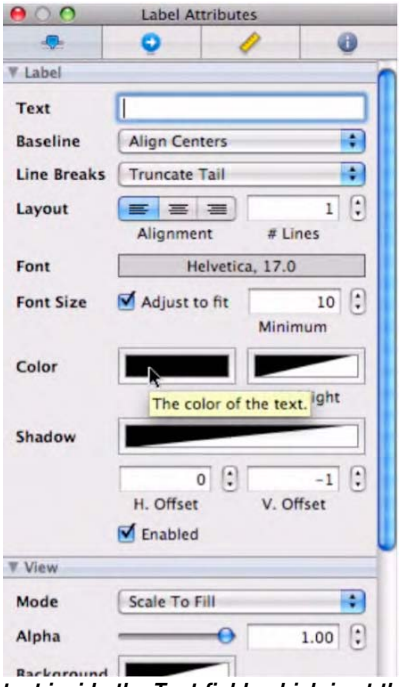
Si llegado a este punto tenemos problemas, tenemos que tener en cuenta que las posibles causas pueden ser que la imagen no haya sido guardada correctamente en la carpeta Resources, o que la misma no ha sido guardada bajo formato `.png`.

19. Vuelve a tu Library, desplázate hacia abajo en la pantalla hasta el icono Label, y arrástralo a t imagen. Puedes colocarlo donde quieras. En mi caso lo coloqué en mi cabeza. Foto 4–11.

Foto 4–11. Clica en la opción label para arrastrar una etiqueta a la imagen

.

20. Tal y como hicimos en ejercicios anteriores, necesitamos eliminar la etiqueta por defecto. Por tanto elimina el texto que contiene. Al mismo tiempo ves al apartado Color, clica sobre el box y cambia a una tonalidad diferente. Ver Foto 4–12.

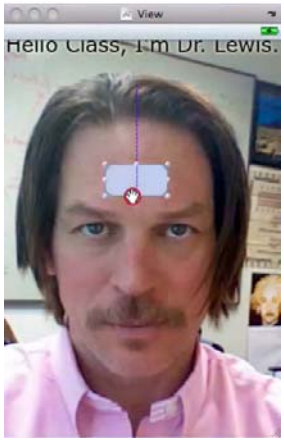


F oto 4–12. Remove the default text inside the Text field, which is at the top of the Label Attributes frame, and then insert "OK I'm repeating myself!"

Puedes cambiar el color del texto que va dentro de la etiqueta. En mi caso, cambié el color del texto ya que soy castaño. Si tu pelo es rubio, por ejemplo, puedes probar a escoger un color oscuro para hacer algo de contraste.

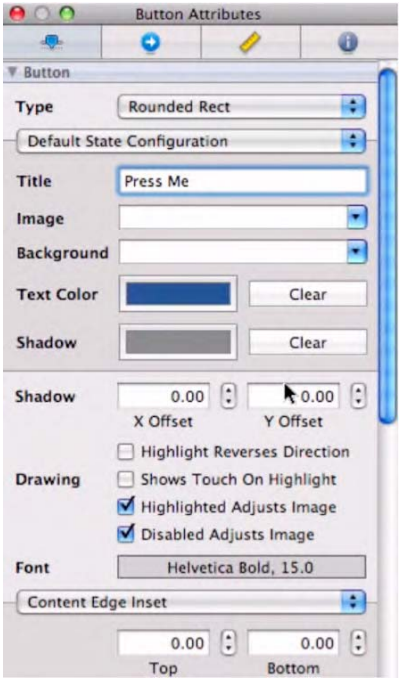
21. Ahora queremos expandir la etiqueta de modo que tenga un tamaño suficiente para contener nuestro texto. Arrastra uno de los dos (indicadores de tamaño) de la etiqueta para conseguir el tamaño en el que creas que cabrá el texto. En mi caso, el texto "OK I'm repeating myself!"

22. Tal y como se muestra en la Foto 4–13, necesitamos arrastrar un botón a la vista en pantalla. Cuando un usuario pulse el botón, se mostrará la etiqueta deseada.



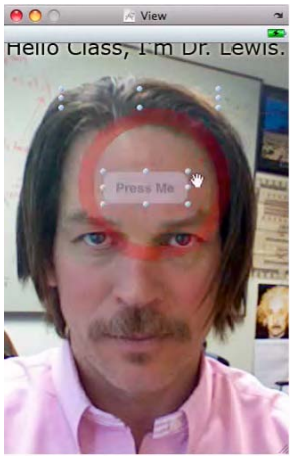
F oto 4–13. Arrastra un botón en el View frame.

23. Como se muestra en la Foto 4–14,también queremos cambiar el texto por defecto del botón. En Button Attributes, cambia el texto del campo Title field a “Press Me.”



F oto 4–14. borra el texto por defecto del campo Title, y escribe “Press Me.”

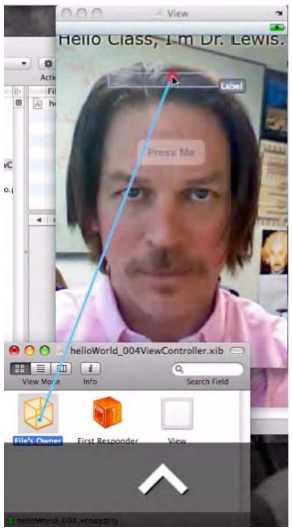
24. En el frame Attributes puedes también ajustar la transparencia, de modo que la misma deje entrever a través del botón. Ve al final de la ventana y cambia el slider Alpha al nivel que te parezca adecuado. El efecto puedes verlos en la Foto 4–15.



F oto 4–15. El botón semi-transparente ha sido seleccionado junto con la etiqueta para ajustar su alineamiento y emplazamiento.

25. Con el botón y la etiqueta seleccionados, alinéalos escogiendo la opción Placement que está al final del frame View Size.

26. A continuación, presiona la tecla ⌘ mientras clicas y arrastras el icono File's Owner icon a la etiqueta. El Interface Builder mostrará la conocida línea de enlace o "fishing line" para mostrar la relación entre ambos. Ver Foto 4–16.



F oto 4–16. arrastra el icono File's a la Label.

27. Como se muestra en la Foto 4–17, establece la conexión entre el File's Owner y la Etiqueta (Label) escogiendo en el menú desplegable.

F oto 4–17. Establece una conexión entre el File's Owner y la Label by seleccionándolo del menú desplegable Outlets: label.

28. Ahora presiona la tecla ⌘ mientras clicas y arrastras desde el Botón al icono File's Owner, tal y como se muestra en la Foto 4–18. Una vez estos elementos queden conectados, aparece el menú desplegable y puedes escoger el texto "OK I'm repeating myself!" Guarda tu proyecto (⌘S), y sal del Interface Builder (⌘Q).

F oto 4–18. Comando(_)-arrastra desde el Botón al icono File's Owner. Después se te presenta un menú desplegable donde podrás seleccionar el mensaje deseado. Guarda (_S) el archivo y sal (_Q) del Interface Builder.

29. Para terminar, introduce ⌘↩ de modo que puedas ejecutar el código. Cuando se lance tu aplicación, verás tu imagen (o la mía) con un botón semitransparente superpuesto. Sigue las instrucciones... pulsa el botón y comprueba el resultado. La Foto 4–19 muestra el iPhone Simulator, mientras que las Fotos 4–20 y 4–21 muestran el iPad Simulator.

Felicidades! Has integrado una interacción usuario/programador con algunos aspectos gráficos. Has sido capaz de salir airoso de este reto con una menor cantidad de instrucciones por mi parte.

Bien Hecho!

F oto 4–19. Clica el botón y... Funciona!

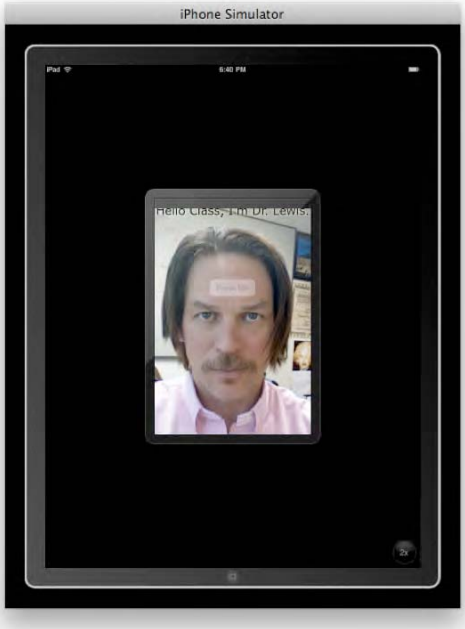


Foto 4–20. Vista del iPad Simulator en modo vista iPhone – antes de que el botón "2x" sea presionado.

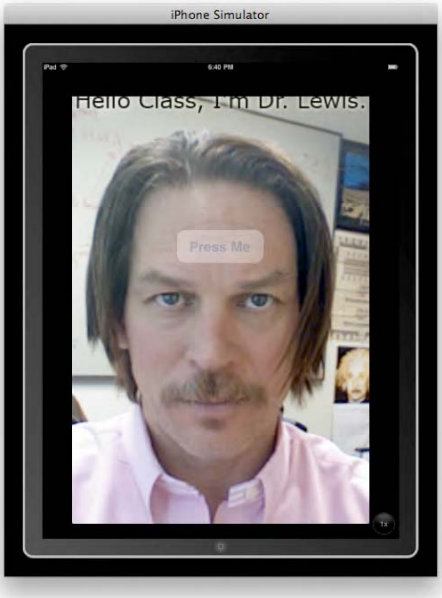


Foto 4–21. iPad Simulator en modo de vista a Toda Pantalla (2x) – antes de que el botón sea presionado. El botón "1x" abajo a la derecha sirve para cambiar a vista normal, correspondiente a la Foto 4-20.

Profundizando en el Código

En esta nueva sección del Capítulo, examinaremos varios conceptos mencionados pero que probablemente sigan envueltos en un halo de misterio par tí. Con la lectura de este capítulo conseguiremos *cierta* comprensión de conceptos, pero no la comprensión definitiva. Por la presente, te autorizo a que "lo cojas aunque sea parcialmente". Por supuesto, si consigues una plena comprensión de todo lo explicado en este capítulo, permíteme ser el primero en felicitarte.

Nibs, Zibs, and Xibs

Estos "items", estos elementos, son básicamente la misma cosa, pero la gente hace referencia a ellos de distintas maneras. Recientemente, en una Conferencia en Denver, "360iDev for iPhone Developers," fué curioso oír cómo varios conferenciantes se referían a los archivos `.xib` como "nibs" o "zibs.". La mayoría de los programadores prefieren decir archivos "nib". No importa el cómo te refieras a ellos, lo importante es que entiendas qué vamos a hacer con estos archivos, qué son y qué necesitas saber para poder entender cómo trabajan.

Recuerdas en el paso 16 cómo abrimos el Interface Builder? Correcto – se aperturó de un modo automático al clicar sobre el archivo nib. Una vez abierto, vistes que el archive contenía todo el código asociado con nuestros botones e imagines, y que, de hecho, esta información estaba contenida ahí mismo. De esa manera, cuando ejecutas la aplicación, todos los objetos y todos los enlaces asociados con los objetos están integrados de un modo correcto, pudiendo de modo "mágico" ir juntos y ofrecer al usuario la experiencia que ideaste.

Resulta que los archivos nib, cuando se examinan a nivel de Cocoa u Objective-C, contienen toda la información necesaria para activar los archivos de interfaz de usuario (UI files), transformándola en una obra de arte gráfica para iPhone o iPad. También es posible escoger archivos nib separados para crear interacciones más complejas, cosa que verás en el siguiente capítulo.

Toda la información que contienen estos archivos es colocada ahí para *crear una instance* de botones, etiquetas, imagines y demás elementos que hayas introducidos. Esta colección de comandos es colocada y guardada en el archive nib para convertirse en la Interfaz de Usuario. El código y los comandos cogidos en su conjunto se convierten en una realidad y son percibidos por el usuario –ya sea de un modo visto, oído o incluso sentido por el mismo.

Algunas veces hacemos uso del termino "instantiate" (instanciar) de un modo similar. Por ejemplo, cuando guardas por primera vez un proyecto, el computador instancia (instantiates) – hace real y te muestra la evidencia para – una entidad de proyecto en virtud de asignarle al mismo un cuerpo de subarchivos, por así decirlo. En helloWorld_004, pudiste ver cómo al proyecto se le dotó instantáneamente de "brazos y piernas" … dos archivos AppDelegate y dos archivos ViewController.

Decimos que hemos "creado una instancia (instance) de" algo cuando le hemos dicho al ordenador cómo y cuando utilizar algo de memoria y reservarla para algún proceso en particular, o conjunto de procesos, de forma que, cuando se cumplan una serie de parámetros, el usuario tenga una experiencia de estos datos (es decir, de todo lo que se

asigna en memoria). Algunas veces nos referimos a estas colecciones o archivos de descripciones y comando como clases (classes), methods (métodos) u objetos (objects). En esta sección de Profundizando en el Código, estos términos pueden parecer que vayan de la mano o que sean sinónimos, pero no es así. Conforme vayas leyendo, irás comprendiendo cada término y aprenderás a distinguirlos y utilizarlos cada uno en situaciones determinadas.

Cuando decimos que has creado una instancia (instance) de botones y etiquetas en tu archive nib, lo que estás diciendo en realidad de que cuando ejecutes tu código, una porción específica de la memoria del equipo, conocida por su dirección, se hará cargo de cosas con objeto de generar la Intefaz de usuario (UI) –la experiencia que vive el usuario al probar la app- que has diseñado. Cada vez que tu aplicación sea ejecutada en tu iPhone o iPad, el interfaz será recreado por los comandos orquestados que residen en los archivos nib.

Considera el archive nib con la acción mostrada en la Foto 4–13. Arrastraste un botón desde la Library a la ventana View, y por tanto creaste una instancia (instante) de este botón. Si alguien te preguntase qué significa esto, deberías mirarle a los ojos y decirle “ Al crear una instancia (instante) de este botón, he dado instrucciones al equipo para que reserve memoria en el correspondiente archivo `.xib`, la cual, una vez se ejecute la aplicación, aparecerá e interactuará con el usuario, tal y como hemos querido al diseñar la aplicación.”

Metodos (Methods)

El siguiente concepto que quiero que veas de un modo más profundo es el de los methods (métodos). Como hice con los nibs, sólo voy a darte una visión “desde las alturas” en este momento. Ya has usado methods en varias ocasiones por lo que simplemente voy a explicarte qué es lo que hiciste con los mismos.

Repasando las fotos de la 4–18 a la 4–21, recuerda cómo ⌘-arrastraste desde el botón al icono File´s Owner y, cuando apareció el menú desplegable, seleccionaste la frase “OK I’m repeating myself!” Adivina qué—Esto fue hecho mediante los methods!

Volviendo al paso 7, quiero que comprendas qué es lo que hiciste cuando copiaste y pegaste el comando:

```
-(IBAction)hello:(id)sendr;
```

Diste instrucciones al computador de asociar una acción con un botón. El primer símbolo en esa sección de código es un símbolo menos (–). Esto significa que la `IBAction` es algo que llamamos instance method. Por otra parte, si hubieses introducido el símbolo más (+) en vez de el menos: `+ (IBAction)`, habríamos llamado a un class method. El primer símbolo anuncia al procesador una instance, mientras que el segundo anuncia una class. Lo que estos dos elementos tienen en común es el method: IBAction. Además, únicamente sólo por el nombre, vemos que esta es una acción que se llevará a cabo en el Interface Builder.

Mientras tanto, echemos un vistazo a lo que sabemos sobre las instances. Volviendo al paso 9, reseñé que habíamos terminado nuestro trabajo en el archivo header, y que era el momento de trabajar en el archivo de Implementación (Implementation File), `helloWorld_004ViewController.m`. Este es el archivo que gestiona el código, y por eso te ordené que lo aperturases. Tuviste que insertar una línea de código en ese archivo (.m) la línea que traíamos ya copiada del archivo de cabecera o header file: -

```
(IBAction)hello:(id)sendr;
#import "helloWorld_004ViewController.h"
@implementation helloWorld_004ViewController
-(IBAction)hello:(id)sendr;
@end
```

Eliminaste el punto y coma del final de la línea y lo reemplazaste por un corchete abierto ({). Lo importante aquí es que el archivo (`.m`) contiene los comandos para el objeto y que estos comandos van en la parte de código que va desde **“**`@implementation"` y el comando **“**`@end"` que cierra el código. Aquí puedes verlo:

```
#import "helloWorld_004ViewController.h"

@implementation helloWorld_004ViewController
-(IBAction)hello:(id)sendr{

}

@end
```

Pero espera – Todavía no he explicado qué significa el término “method” (método). Bien, a ver si esto ayuda. Los Methods son los paquetes de código que el computador ejecuta cuando un input en particular es recibido y debe de ser procesado- como puede ser la pulsación de un botón.

Ponemos un ejemplo: un programador dice “Aquí tienes una aplicación que te ayudará a dibujar una bonita casa”. Esta información pertenece al Header. A continuación el programador introduce instrucciones específicas sobre cómo se diseñará la casa, cómo va a asentarse la misma, su orientación y el tipo de clima que vamos a colocar en el fondo, y demás elementos. “Dibuja una línea de horizonte de modo curvo a un tercio de la parte inferior de la pantalla, y en la mitad de la misma coloca un rectángulo de 4 × 7, el tejado será un trapecio con una base de …,” y demás instrucciones. Estas instrucciones específicas del diseño pertenecen al archivo de Implementación (implementation file), ya que describen las acciones reales – *el método* (method)- de dibujado de la casa.

Para conectar tu botón a un method llamado `"hello, "` añadiste el siguiente código:

```
-(IBAction)hello:(id)sendr;
```

Esto *creó una instancia (instance)* del método (method) `hello`. Entonces, creaste un espacio en la memoria para ejecutar el código que está dentro de ese method (método) `hello`.

Resumiendo:

1 Primero arrastraste un botón desde la Library a la ventana del documento y creaste una instancia (instante) de ese botón.

2 Luego ⌘-arrastraste desde el botón al icono File's Owner y conectaste el botón el método (method) `hello` que aparecía en el menú desplegable.

3 En el momento que introdujiste el código `- (IBAction)hello:(id)sendr;` creaste este `"hello"` method instance :

- El (-) significa que la `IBAction` es un méthod instance.
- `(IBAction)` *tipo (type)* que obtenemos
- `hello` es el *nombre (name)* del method (método).
- `(id)` es el tipo (type) de argumento.
- `sendr` es el *nombre (name)* del argumento

Esto es todo lo que vamos a ver de los methods por ahora. Vamos a ir repitiendo estos pasos de creación y utilización de methods en los siguientes capítulos, y con la práctica, el concepto te irá pareciendo cada vez más claro y nítido.

Capítulo 5

Botones y Etiquetas con Gráficos Múltiples

En este capítulo vamos a emprender juntos nuestro quito programa, y es el momento de pisar un poco el acelerador. Tal y como vimos en el Capítulo 4, podrás tener acceso a capturas de pantallas e implementar código si recuerdas la mayoría de detalles- pasos que han sido descritos repetidamente en ejemplos previos. Habrá menos fotos de cada uno de los pasos que veamos, trabajaremos más y recurriremos a información presentada en el Capítulo 4.

Como en el Capítulo 4, una vez que hayamos completado el programa, haremos una revisión de código en la sección "Profundizando en el Código" ("Digging The Code"). Inicialmente trataremos algunos de los mismos aspectos y conceptos que tratamos en el Capítulo anterior, y profundizaremos en código nuevo. No todo va a ser conceptos profundos, pero sí que expandiremos nuestros horizontes para considerar otros conceptos de computación que se unan a estos niveles más profundos de análisis.

Probablemente advertirás un cambio de estilo en el Capítulo 5, en el cual iremos dejando el lenguaje "elemental" utilizado en capítulos anteriores. Por lo pronto, vamos a coger ritmo – un poco más rápido y adquiriendo una terminología más técnica. En todo caso, si no pasa nada si no te haces con la totalidad de conceptos y técnica, ya que es perfectamente comprensible. Relájate y disfruta del siguiente ejemplo.

helloWorld_005: Una Aplicación View-Based

En este ejemplo, continuaremos ahondando en el concepto INDIO (Interaction, Navigation, Data, e I/O (input/output)). Nuestro código recuperará dos imagines, que por diferentes medios, interactuarán con el usuario. Este ejercicio nos acerca al aspecto I/O de INDIO, sobre el cual no habíamos ahondado porque hasta ahora no habíamos alcanzado este concepto.

Como un guerrero digital que se dirige por la senda del bosque de of Objective-C, tendrás que acostumbrarte a vadear ríos, cazar tigres, hacer fuego en la lluvia y muchas otras cosas más. Hasta ahora has aprendido a hacer fuego con pedernal y has saltado pequeños charcos.

En esta lección aprenderás a cruzar grandes ríos y cazar tigres y pronto seremos capaces de enfrentarnos a grandes demonios de I/O en el reino de INDIO.

En Segundo lugar- Siéntete a gusto contigo mismo! Estás profundamente inmerso en el Bosque de Objective-C. Si te pierdes, recuerda que podrás encontrar el “mapa en este link:

`http://www.rorylewis.com`

Foto 5–1. Crea tres imagines png: una capa inferior, otra superior y un icono de escritorio. Guárdalas en un escritorio limpio y organizado.

Preliminares

Como viene siendo habitual, empecemos con un escritorio limpio y solo cuatro iconos: nuestro Macintosh HD y tres archivos (mostrados como iconos en la Foto 5–1). Como estoy seguro de que ya habrás advertido, pienso que la organización es básica en la programación y esto pasa por tener un escritorio despejado, y por tanto quiero animaros a perfeccionar este concepto de la organización. Utilizando la correspondiente combinación de teclas, procedemos a cerrar todos los programas abiertos.

Te invito a descargar estas imágenes (`www.apress.com`) las cuales se convertirán el nos pilares de este proyecto. En todo caso sería interesante que preparases estas imagines por tu cuenta. De esta manera, desarrollarás un mayor interés y entusiasmo en esta tarea. Llegado a este punto tienes dos alternativas: descargar las imagines de la url facilitada o prepararlas tú mismo. Suponiendo que vas a hacer el esfuerzo de crear esos tres archivos de imagen, presta atención a las siguientes pautas.

El tamaño de la primera imagen será el estándar del iPhone, es decir 320 píxeles de ancho por 480 píxeles de alto. Esta imagen sera standard of 320 pixels in width by 480 pixels in height. Esta será la capa inferior de dos imagines, por lo que la llamaremos imagen o capa de fondo. Nuestra capa de fondo es una fotografía de las escaleras del Instituto de Ingeniería de UC-Colorado Springs. Usaremos esta imagen como fondo de otra imagen, en este caso, la de mi mujer, Kera. Cuando el programa se ejecute aparecerá la imagen de fondo indicada, y una vez pulsemos un botón, aparecerá la imagen de Kera en la parte superior de la escalera. Que bueno —Kera Lewis ha decidido volver a la Universidad!

Este es nuestro escenario. Se trata de una imagen familiar en la que de pronto alguien (o algo) inusual o inesperado aparece de repente. Esta imagen de fondo necesita tener un formato de 320 × 480 pixels, y necesita guardarse en formato `.png` (Foto 5–2).

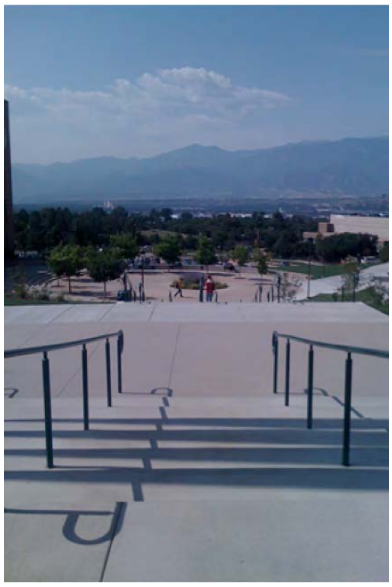


Foto 5–2. Imagen de fondo – o capa de fondo..

Para crear la segunda imagen, a la cual llamaremos capa superior, copiaremos esta primera imagen de fondo. Esta imagen la recortaremos para crear una nueva imagen con unas dimensiones exactas de 320 × 299 pixels. Sé que puede parecer una resolución poco habitual, pero confía en mí. Bien, hemos conseguido una copia más o menos cuadrada de las dos terceras partes de la sección inferior de la foto de fondo.

El siguiente paso es pegar en esta selección algún recorte de algo interesante o inusual. Con ello obtendremos algo como en la Foto 5–3: Kera Lewis, aparece superpuesta sobre la imagen de fondo. Esta capa superior también la guardaremos en formato `.png`.

F oto 5–3. Esta es la capa superior modificada, la cual se superpondrá a la capa de fondo.

Por tanto, el resultado final obtenido será el de un archivo de capa superior que ha sido elaborado mediante una sección inferior de la capa de fondo original, y superponiendo sobre esta una imagen o foto original. A estas alturas ya tendrás una idea de lo que queremos hacer: empezar con una imagen de fondo para posteriormente, al ejecutar una acción aparezca esta segunda capa superpuesta de un modo perfecto. Con esta acción conseguiremos que ese objeto o persona parezca que ha aparecido de la nada. Nuestra capa superior no invadirá la parte superior de la imagen de fondo; la cual estamos reservando para la aparición de un texto. Utilizamos esta forma de trabajo ya que el iPhone y el iPad no soportan transparencias `.png`.

La tercera imagen será un icono a tu elección En el Capítulo anterior, vimos como personalizar un icono. En mi caso, he tomado una parte de la cara de mi mujer y la he utilizado como archivo “icono” (Foto 5–4).

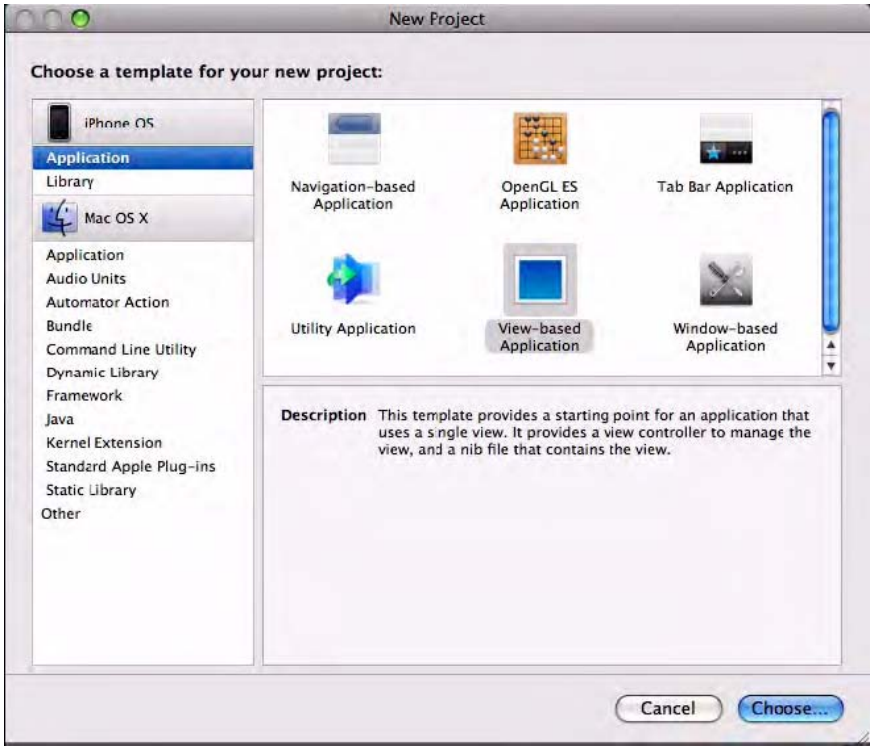
F oto 5–4. Esta es la imagen que voy a utilizar de icono de pantalla… una cara preciosa!

Os recuerdo que los iconos para el iPhone/iPad deben tener un formato recomendado de 320 × 480 pixels Por favor, ten en cuenta en la medida de lo posible, estas dimensiones. Una vez tengas preparadas estas tres imágenes— capa de fondo, capa superior y el icono—guárdalas en tu escritorio.

Xcode – Iniciando un Nuevo Proyecto

Una vez tenidas en cuenta las instrucciones preliminares, empecemos a trabajar en la aplicación:

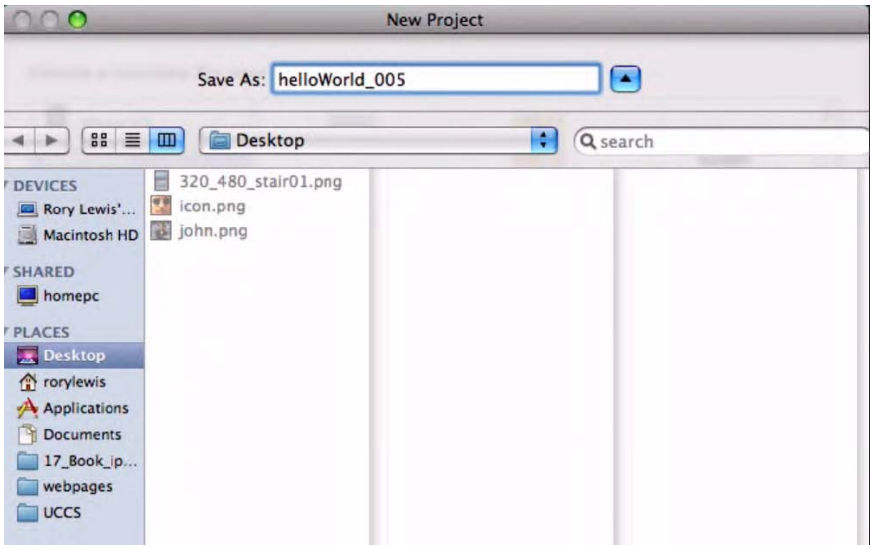
1. Abrimos Xcode y presionamos la combinación de teclas ⌘⇧N. En la ventana de Nuevo Proyecto, seleccionamos el Template View-Based Application, tal y como aparece en la Foto 5–5. El Template view-based application normalmente es utilizado para ayudarnos a diseñar una aplicación con una sola imanten, y que dado que tendremos que hacer uso de una segunda imagen (la imagen de mi mujer sobre las escaleras), puedes pensar que esta segunda imagen puede suponer un problema. En principio el Template navigation-based application puede parecer más adecuado, ya que enlaza datos de modo jerárquico y utiliza múltiples pantallas, pero como comprobareis, en este caso no es así.



F oto 5–5. Presiona ⌘⇧N y selecciona View-based Application en la ventana New Project.

Vamos a tratar únicamente con una perspectiva, sobre la cual vamos a superponer una imagen, no una vista. Si fuésemos a tener partes de código en el panel de navegación y otras partes de código en otros paneles de navegación, si que hubiésemos elegido la aplicación Navigation-Based. Sin embargo, en este proyecto únicamente vamos a manipular una vista sobre la que solaparemos una imagen, y no vamos a navegar de un panel a otro. Por tanto, en esencia vamos a trabajar únicamente con el modo de vista simple.

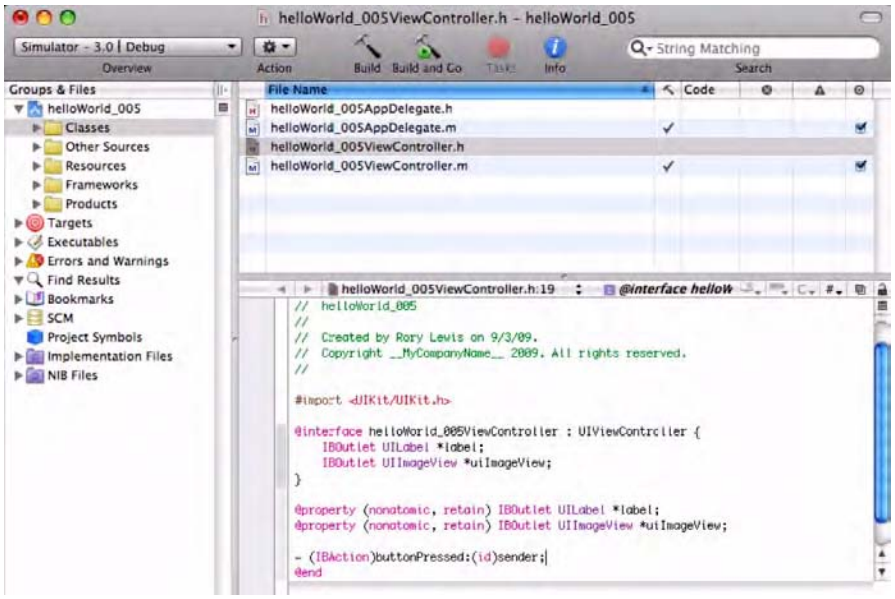
2. Guarda tu View-based Application en el escritorio con el nombre "`helloWorld_005`." Ver Foto 5–6. Esta va a ser nuestra última aplicación "Hello World!". Te sugiero que, una vez terminado este programa, los guardes todos en un directorio llamado `Hello World` dentro de tu directorio `Code`, ya que probablemente en un futuro puedas considerar interesante el volver a estos proyectos y repasar el código de los mismos.



F oto 5–6. Guarda tu aplicación View-based en el escritorio con el nombre helloWorld_005.

Posteriormente en este libro, cuando entremos en detalles relativos al Objective-C and Cocoa, tendremos oportunidad de echarnos la mano a la cabeza y decir, “Maldita sea—todo esto suena bastante complicado, pero en realidad esto lo he hecho con anterioridad. Quiero volver atrás y ver cómo he relacionado archivos en los ejercicios ‘Hello World!’ que hice al principio de este libro.”

Xcode ha inicializado un proyecto llamado `helloWorld_005` (ver Foto 5–7). Clica en el archivo raíz `helloWorld_005` en la sección Overview—Groups & Files, y haz una pequeña pausa. Localizas las carpetas llamadas “Classes,” “Other Resources,” “Resources,” y demás? Como mencionamos con anterioridad, hemos estado utilizando un lenguaje elemental y a partir de ahora vamos a ir utilizando algo más de jerga técnica específica.



F oto 5–7. Después de haber guardado el proyecto Xcode inicia el proyecto llamado helloWorld_005,tal y como reza en la parte superior de la ventana.

En vez de decir, “Abre la carpeta Classes” Debemos de decir “Haz clic sobre el ‘triángulo disclosure’ cercano a la carpeta Classes.” Este triángulo es un pequeño triángulo que está apuntando hacia la derecha, y que una vez clicado sobre él, pasa a apuntar hacia abajo y muestra los archivos que contiene una carpeta. Conocer este tipo de nomenclatura puede ser útil—por ejemplo en una fiesta-cóctel, o para marcarse un tanto con algún colega programador!. “Así que si nunca has oído hablar del “triángulo disclosure”…—hey— poco a poco te estás convirtiendo en un geek!

Entender IBOutlets

Apreciamos dos archivos AppDelegate y otros dos archivos ViewController. En capítulos anteriores hemos hablado acerca de las extensiones.m y .h en detalle. Vamos a hacer lo que la mayoría de los programadores de Cocoa y Objective-C hacen— y es empezar programando los archivos de Cabecera (header files). Técnicamente podríamos decir, "Abre el archivo de Cabecera (Header)." Si alguien te pregunta qué significa esto, puedes explicárselo del siguiente modo, "Haz clic en el triángulo disclosure correspondiente al archivo Classes y abre el archivo con extensión .h, en concreto ViewController.h!"

Ya has programado previamente otros archivos de Cabecera (Headers), por lo que deberías estar acostumbrado ya a obviar esta porción de código. Sin embargo, esta vez vamos a echar el freno y pensar en lo que estamos haciendo. En todos los ejemplos anteriores, únicamente hemos tenido un solo IBOutlet, algo que nos permite interactuar con el usuario. Vamos a intentar se algo más técnicos y específicos, ya que la forma en la que lo hemos dicho es demasiado elemental. Adentrémonos más en lo que es el IBOutlet, de modo que cuando lleguemos a la sección "Volviendo Atrás y Profundizando en El Código", estaremos preparados para entender todo lo que leamos.

Abre tu archivo helloWorld_005ViewController.h y repasa el siguiente código. Vamos a ver si eres capaz de encontrar el camino hacia una comprensión más profunda de estos elementos:

```
#import <UIKit/UIKit.h>

@interface testViewController : UIViewController {

}

@end
```

Fíjate en la primera línea: #import <UIKit/UIKit.h>. Es la que no permite el uso de la palabra clave IBOutlet. Utilizamos #import para importar el UIKit, que es la Marco de Interfaz de Usuario (UI) dentro del gran conjunto de porciones de códigos llamado IPhoneRuntime, la cual es una reducida versión del sistema operativo OS X que podemos encontrar en cualquier Mac. Por supuesto, IPhoneRuntime es más pequeño y liviano para que pueda ser utilizado en un dispositivo iPhone o iPad.

Cuando importemos el UIKit, dotaremos a nuestras herramientas la habilidad de hacer uso de infinitos códigos preprogramados por Apple para nosotros—llamados classes—uno de los classes más populares y extendidos que hay es el IBOutlet, del cual ya hemos hecho uso. El IBOutlet es una directiva especial llamada "instancia variable" que llama Interface Builder para mostrar aquellos elementos que queremos que aparezcan en nuestros iPhone o iPad. A su vez, Interface Builder utiliza estas directivas para comunicarle al compilador que conectarás objetos a nuestros archivos .xib. El Interface Builder no conecta estas salidas a nada, pero le dice al compilador que tú las vas a añadir.

En nuestro ejercicio, usaremos dos IBOutlets—uno relacionado con nuestro texto y otro relacionado con nuestra imagen. Recapitulando: en ejercicios anteriores, únicamente hicimos usos de un IBOutlet, pero en este ejemplo haremos uso de dos. Un IBOutlet será usado para nuestro texto (algo tipo "Hello World, I'm back!"), y otro que será usado para la capa de imagen superior de Kera (Foto 5–3).

Os recuerdo lo que vamos a utilizar: 1) La imagen de fondo de las escaleras, 2) La imagen de superposición de mi mujer, y 3) el texto que vamos a introducir donde ella

afirmará que vuelve a la Universidad.

Ahora sabemos que necesitamos `IBOutlets`. Vamos a centrarnos en los corchetes que aparecen después de `@interface testViewController : UIViewController`. Nuestro código deberá quedar de esta manera.

```
#import <UIKit/UIKit.h>

@interface testViewController : UIViewController {
IBOutlet
IBOutlet
}
@end
```

Como ves, todavía no tenemos código- simplemente una serie de marcadores de posición, pero sí sabemos lo que vamos a utilizar: dos `IBOutlets`. También sabemos que funcionalidad tendrá cada uno; uno reproducirá el texto de Kera y el otro preproducirá una imagen que se superpondrá a la imagen de fondo.

Sabemos que cuando introducimos texto en la pantalla del iPhone o iPad, utilizamos la Clase `UILabel`. Esta clase nos marca múltiples líneas de texto estático. Por tanto continuamos y escribimos `UILabel` a continuación del primer IBOutlet que introdujimos, tal y como puedes ver el la línea de código expuesta abajo. Con respecto al segundo `IBOutlet`, sabemos que lo queremos utilizar para superponer la imagen de Kera tal y como queda en la Foto 5–3.

Una Buena idea es utilizar la Clase `UIImageView` ya que nos facilita un código escrito por Apple que permite mostrar imagines simples o una serie de imágenes en movimiento, por tanto, pasamos a introducir la orden `UIImageView` a continuación del segundo `IBOutlet`.

```
#import <UIKit/UIKit.h>

@interface testViewController : UIViewController {
IBOutlet UILabel IBOutlet UIImageView
}
@end
```

Pointers

Ahora que sabemos la forma de introducir texto y una imagen en la pantalla del iPhone/iPad, necesitamos especificar el texto y la imagen a incluir. Algunas veces utilizamos código predefinido, creado por los amigos de Apple, que hace lo que hace por virtud de referencia o apuntando a nuestros recursos –esto es, nuestro texto e imágenes. Como estás empezando a comprobar, este es el contexto en el que usaremos los Pointers.

En capítulos anteriores, te dije que no prestases atención a la estrella (*). Bien, ahora es el momento de centrarnos en ella. Estas estrellas (*) son Pointers. Al final del capítulo “Profundizando en el Código”, entraremos de lleno en materia, pero llegado a este punto vamos a hacer una pequeña introducción.

Necesitamos una manera indirecta de mostrar nuestro texto e imágenes en la pantalla. Decimos “indirecta” porque no vamos a escribir el código para conseguirlo-usaremos el código de Apple para conseguirlo. Podemos hacer uso de Clases preexistentes, y estas

Clases harán el trabajo de llamar a nuestro texto e imágenes. Por ello decimos que es un modo indirecto de conseguir nuestro objetivo.

Sirva para ello esta pequeña analogía. Supongamos que detienes a un individuo que ha entrado a robar en tu casa. Llamas a la policía y cuando llegan a casa, señalas al criminal y dices "Ese tío es el ladrón!" Entonces, el policía, no tú, detiene al criminal y lo arresta con cargos.

Queremos mostrar texto en nuestro iPhone/iPad. Llamamos a `UILabel`, y cuando "llega" APUNTAS a tus palabras y dices, "Here's the text." Entonces el `UILabel`, no tú, es el que trata con el texto.

Del mismo modo actuaremos cuando sea una imagen la que queremos desplegar en nuestro iPhone/iPad. Llamas a `UIImageView`, y cuando "llega", APUNTAS a tu fotografía o dibujo y dices "Aquí está la imagen". Entonces, el código `UIImageView` code, no tú, es el que trata con la imagen.

Quizás te estés preguntando qué nombre tienen esos Pointers, o deben tener. Lo bueno de todo e que puedes darle el nombre que quieras. Asignemos `*label` al Pointer de `UILabel`, y llamemos `*uiImageView` al Pointer de `UIImageView`:

```
#import <UIKit/UIKit.h>

@interface testViewController : UIViewController {
IBOutlet UILabel *label; IBOutlet UIImageView *uiImageView;
}

@end
```

La gente de Apple describe su razonamiento a la hora de programar IBOutlets como un proceso en el que se le dan indicaciones al Interface Builder sobre lo que debería hacer cuando se le ordena mostrar la interfaz gráfica.

- Un `IBOutlet` ***susurra*** al oído del Interface Builder's que la Clase `UILabel` va a usar el texto indicado por el Pointer `*label`**.**

- El otro `IBOutlet` susurra al oído del Interface Builder's que la Clase `UIImageView` va a usar la imagen referenciada por el Pointer `*uiImageView`**.**

Pero aún no hemos terminado. Después de que le digamos al Interface Builder qué debe esperar, necesitamos decirle a nuestro microprocesador Mac –a través del compilador- que un evento importante está a punto de llegar . Una de las cosas más importantes que necesita nuestro compilador es que le digamos cuando llega un objeto u otro. Esto es debido a que los objetos son masas independientes de números y símbolos que ejercen su peso sobre el microprocesador, haciendo gasto sobre el mismo. Por tanto, el procesador necesita que nosotros, el programador, cuando necesita coger ese objeto y ponerlo en un determinado espacio de memoria.

Los objetos pueden venir de una gran cantidad de maneras—como conceptualmente son diferentes términos como pájaro, gurú, fútbol y casa. Por tanto, para permitir que el microprocesador haga su trabajo cuando llegue el momento, necesitamos informarle de cada uno de los objetos que serán usados en nuestro código, y esto se hará con dos parámetros específicos o características: la "propiedad" y el "tipo".

Pero no alucines! Facilitarle esta información es realmente sencillo y se compone de dos etapas:

El primer paso es el que acabamos de explicar: le damos al compilador información acerca del uso y definición de los objetos mediante dos características: la "propiedad" y el "tipo". El segundo paso: cuando el microprocesador recibe estos datos este utiliza esta información para Sintetizarla más eficientemente..

Resumiendo:

1. Primero, declaramos que nuestro objeto tiene una propiedad con un tipo específico.
2. Segundo, damos instrucciones a la computadora para Implementar –o Sintetizar- esta información.

Informamos al compilador acerca de nuestro objeto, incluyendo parámetros descriptivos específicos. Luego le damos al compilador vía libre para Implementar nuestro objeto, informándole de la Sintetización del mismo.

Y cómo podemos hacer esta declaración y esta Implementación? Usamos herramientas en nuestro código llamadas directivas. Marcamos estas directivas insertando la "@" antes de marcarla. Esto significa que para declarar que propiedad tiene nuestro objeto, debemos preceder la palabra "property" para convertirla en una propiedad directiva: `@property`.

Cuando vemos un `@property` en nuestro código, sabemos que es una propiedad directiva. Por analogía, cuando queremos decirle al compilador que procese y Sintetice, debemos de colocar la "@" delante de nuestra declaración de síntesis: `@synthesize`.

Lo mismo que acabamos de decir, pero en idioma "geek", quedaría del siguiente modo:

1. La directiva `@property` declara que nuestro objeto tiene una propiedad con un tipo específico.
2. La directiva @synthesize Implementa los métodos que hemos declarado en la directiva `@property`.

Fácil, eh? OK, solo dos cosas más y volveremos a nuestro código.

Finalmente quiero hacer una aclaración más concerniente al paso 1 y es que también necesitamos especificar si la propiedad será de sólo-lectura o de lectura-escritura. En otras palabras, necesitamos especificar si siempre será la misma y permanecerá inalterable o si por el contrario puede convertirse en algo Nuevo. En idioma "geek" llamamos a esto "mutabilidad". La mayoría de las veces vamos a usar el código de Apple para manejar la mutabilidad de las propiedades concernientes a nuestros objetos.

Propiedades: Gestión & Control

Con el fin de instruir al código de Apple para manejar la propiedad de la mutabilidad, llamaremos a la propiedad "no atómica". Para entender el significado de este término, busca la diferenciación entre los términos "no atómico" y "atómico". Recuerda que "atómico" significa poderoso, y esto implica la habilidad de llegar al mundo

microscópico y realizar cambios en él. Por tanto "no atómico" deberíamos traducirlo como "no tan poderoso", más superficial y no manipulable.

Si llamamos a una propiedad (como la de la mutabilidad) como "o atómica", estamos diciendo, "Apple, por favor, ayúdanos con el tema de la mutabilidad y demás asunto relacionados con ella-realmente no quiero entrar en detalles. Lo que tu apliques será bueno para mí". Más tarde o temprano querrás tomar un control directo sobre esta propiedad, y entonces pasarás a llamarla "atómica". Por el momento, usaremos una aproximación mucho más laxa a este concepto y permitiremos a Apple que se encargue del mundo microscópico. Por tanto, cuando llegue el momento de escoger entre una u otra designación, decídase por la "no atómica"!

La segunda consideración que quiero hacer concerniente al Paso 1 tiene que ver con la gestión de memoria. Necesitamos abordar el problema de cómo hacer que el iPhone/iPhad conozca, cuando almacenemos un objeto, si el mismo será de lectura o de lectura-escritura. En otras palabras, necesitamos ser capaces de informar al computador de la naturaleza de la memoria asociada con el objeto-en términos de quién quiere cambiarlo, cuando y cómo. En términos comunes, queremos controlar esta información y mantenerla en nuestras manos-es decir, retenerla. A medida en que alcancemos a través de los ejercicios que nos quedan del libro, aprenderemos a mantener el código en nuestras manos, nos reservaremos el derecho a administrar nuestra memoria.

Podemos resumir la adición de estos detalles a las propiedades de las directivas, y la forma en que podremos modificar el código de la siguiente:

- La directiva `@property (nonatomic, retain)` dice lo siguiente:
  - La mutabilidad debe de ser no atómica. Apple, encárgate de ello, por favor!
  - La gestión de memoria es algo que queremos manejar. Mantendremos el control sobre ello.
- La directiva `@synthesize` Implementa los métodos que declaramos en la directiva `@property`.

Tenemos que añadir algo más de complejidad a todo esto. Añadimos estas directivas en dos archivos diferentes. Definimos la directiva `@property` con una declaración en el archivo de Cabecera (Header file), y entonces la Implementamos utilizando la directiva `@synthesize` en nuestro Archivo de Implementación (implementation file.)

- Header File (archivo de cabecera): `helloWorld_005_ViewController.h`
  - `@property (nonatomic, retain)` "nuestras cosas"
  - Implementation File (implementación): `helloWorld_005_ViewController.m`
- El `@synthesize` "nuestras cosas" definido en `@property`.

Necesitamos escribir dos tantas para cada uno de los dos `IBOutlets`: uno para el texto y otro para la imagen. Y dado que estamos con el Header (archivo Cabecera), necesitamos repetir este proceso cuando Sinteticemos en el archivo de Implementación (implementation file).

Bien, procedamos a imputar el código:

```
#import <UIKit/UIKit.h>h

@interface testViewController : UIViewController {
IBOutlet UILabel *label;
IBOutlet UIImageView *uiImageView;
}

@property (nonatomic,
retain) @property
(nonatomic, retain)

@end
```

La verdad, parece que hemos dado muchas explicaciones para simplemente decir: `@property (nonatomic, retain).` Sin embargo recuerda que estamos inmersos en las trincheras … Estamos diciéndole al ordenador que queremos que Apple tome el control de la mutabilidad, pero queremos mantener el control de la memoria para nosotros. Más tarde Sintetizaremos estos comandos en el Archivo de Implementación (implementation file), para ambos `IBOutlets.`

`IBOutlets`? Los recuerdas? Oh sí—volvamos a esa parte del programa. El `IBOutlet` para el texto es `UILabel` con el Pointer `*label`, por tanto, introduce el código para el textos: `IBOutlet UILabel *label;`
El resultado queda así:

```
#import <UIKit/UIKit.h>

@interface testViewController : UIViewController {
      IBOutlet UILabel *label;
      IBOutlet UIImageView *uiImageView;
      }
@property (nonatomic, retain) IBOutlet UILabel *label;
@property (nonatomic, retain)
@end
```

The `IBOutlet` para la imagen es `UIImageView` con el Pointer `*uiImageView`, por tanto, introduce el siguiente código para la imagen:

```
IBOutlet UIImageView *uiImageView;

The result will look like this:#import <UIKit/UIKit.h>

@interface testViewController : UIViewController {
      IBOutlet UILabel *label;
      IBOutlet UIImageView *uiImageView;
      }
@property (nonatomic, retain) IBOutlet UILabel *label;
@property (nonatomic, retain) IBOutlet UIImageView *uiImageView;
@end
```

Hemos terminado con el archivo Header (Cabecera)? Pues todavía no. Necesitamos añadir nuestras `IBActions`. Hemos manejado nuestras dos `IBOutlets`, pero ahora necesitamos utilizar una `IBAction` para algo que nos es realmente necesario. Puedes adivinar qué es?

Añadiendo IBActions

Si, necesitamos un botón! Por tanto vamos a hacer una IBAction para un botón. Podríamos profundizar bastante en esto de Nuevo, en el código para la `IBAction`, pero ya lo hemos hecho bastante en los conceptos anteriores. Por lo que vamos a dejar la parte técnica de estos elementos para el apartado “Profundizando en Código”.

Mientras tanto, simplemente introduce el código que está en negrita. Intente anticipar el funcionamiento de las distintas parte –o parámetros- y veremos qué cerca te has quedado más tarde.

```
#import <UIKit/UIKit.h>

@interface testViewController : UIViewController {
       IBOutlet UILabel *label;
       IBOutlet UIImageView *uiImageView;
}

@property (nonatomic, retain) IBOutlet UILabel *label;
@property (nonatomic, retain) IBOutlet UIImageView *uiImageView;
-(IBAction)buttonPressed:(id)sender;

@end
```

Antes de salir, copia (⌘C) esta línea `IBAction:`

```
 -(IBAction)buttonPressed:(id)sender;
```

y guarda tu trabajo pulsando ⌘S. Por supuesto, ir guardando el trabajo efectuado es una Buena idea!. Acabamos de copiar la línea `IBAction` para un botón. Recuerdas la razón por la que lo hacemos?)

Pues para pegarlo en el Archivo de Implementación y así ahorrar tiempo, algo que siempre se agradece. Bien, hemos terminado con el header! Vamos a tomarnos un descanso.

Codificando el Archivo de Implementación

Ahora que has vuelto de un merecido descanso, continuemos. Es el momento de abrir nuestro Archivo de implementación `helloWorld_OO5ViewController.m`.

1. Usa el comando ⌘V para pegar la línea de código que hemos copiado al final del apartado anterior:

```
-(IBAction)buttonPressed:(id)sender;
```

El resultado será el siguiente:

```
#import "helloWorld_OO5ViewController.h"

@implementation helloWorld_OO5ViewController

-(IBAction)buttonPressed:(id)sender;
```

En el Archivo de Cabecera (Header), le dijimos ala computadora que íbamos a realizar algunas accion(es) cuando se presione un botón. Ahora que hemos pegado este conjunto de comandos en el Archivo de Implementación, reemplazaremos el ";" con un conjunto de corchetes. Dentro de estos corchetes le diremos a la computadora que necesitamos que sea Implementado cuando se pulse el botón.

```
#import "helloWorld_005ViewController.h"

@implementation helloWorld_005ViewController

-(IBAction)buttonPressed:(id) sender{

}
```

2. Antes de profundizar en el código que entrará en acción cada vez que el usuario pulse el botón de nuestra aplicación, necesitamos entrar en profundidad en la segunda parte de lo explicado en la sección anterior. Recuerdas cómo definimos nuestra directiva `@property` con la inclusión que hicimos en el archivo de Cabecera (Header), y cómo dejamos la correspondiente directiva @synthesize para nuestro Archivo de Implementación?. En concreto teníamos una Directiva `@property para el UILabel` y otra Directiva para el `UIImageView`, en concreto, este código:

```
@property (nonatomic, retain) IBOutlet UILabel *label;
@property (nonatomic, retain) IBOutlet UIImageView *uiImageView;
```

Lo que significa que solo necesitamos incluir una directiva `@synthesize` para ambos Pointers, una para la `UILabel` y otra para la `UIImageView`**.** Utilizamos el término "Label" (Etiqueta) para designar al Pointer que trabajaba para `UILabel` named `label`**,** y llamamos **"**`uiImageView`**"** al Pointer de `UIImageView`.

Podemos poner ambos en la misma línea, así que vamos a ello:

```
#import "helloWorld_005ViewController.h"

@implementation helloWorld_005ViewController
@synthesize label, uiImageView;

(IBAction)buttonPressed:(id) sender{

}
```

En este ejemplo hemos utilizado una simple superposición de imagines y texto para que tengas una ligera noción de la tremenda dificultad de un concepto. Hey, si tu cabeza hecha humo, no te amargues! Asimilar los conceptos de Pointer, objetos y síntesis no es algo trivial. Desafortunadamente, no hay otra manera de evitar esta lección de programación. Enseñando estos conceptos en clase, hubo varios estudiantes con conocimientos en C++, C#, y Java que lo pasaron peor que otros estudiantes de otras áreas que simplemente se limitaron a aceptar que:

- La directiva `@property` declara que nuestro objeto tiene una propiedad con un tipo específico..
- La directiva `@synthesize` Implementa los métodos que introdujimos en la directiva `@property`**.**

Algunos de los estudiantes con experiencia en programación que tenían demasiada información asimilada tardaron más en asimilar esos dos conceptos que los estudiantes novatos. Volveremos sobre esos dos puntos una y otra vez, por lo que vamos a seguir adelante. Estas ideas irás asimilándolas poco a poco.

Proporcionando la Síntesis (Synthesize)

Cuando el usuario ejecuta esta aplicación y presiona el botón, la imagen de mi mujer, Kera, aparecerá sobre la imagen de fondo. Ella va a "decir" algo a través del texto incrustado en la aplicación. Algo así como "Hello World, I´m back!"

1. Para conseguirlo, necesitamos asociar la variable Etiqueta (label) con un texto asignado por nosotros, quedando de la siguiente manera:

```
#import "helloWorld_005ViewController.h"

@implementation
helloWorld_005ViewController
@synthesize label, uiImageView;

-(IBAction)buttonPressed:(id) sender{
label.text = @"Hello World, I'm back!";
}
```

2. Una vez completada la tarea de la codificación del texto, necesitamos añadir el código que posibilite que la imagen de Kera aparezca. Para ello, usaremos un método clásico llamado `imageNamed` el cual mostrará la imagen `kera.png`, la imagen de capa superior que preparamos al principio de este proyecto. Introduce la línea en negrita en nuestro código, justo debajo de la línea que hemos empleado para la inserción del texto.

```
#import "helloWorld_005ViewController.h"

@implementation helloWorld_005ViewController
@synthesize label, uiImageView;
-(IBAction)buttonPressed:(id) sender{
label.text = @"Hello World, I'm back!";
UIImage *imageSource = [UIImage imageNamed: @"kera.png"];
}
```

NOTA: En la correspondiente sección de video, en lugar de entrar esta línea de código, uno de nosotros no se dio cuenta e introdujo la siguiente línea:

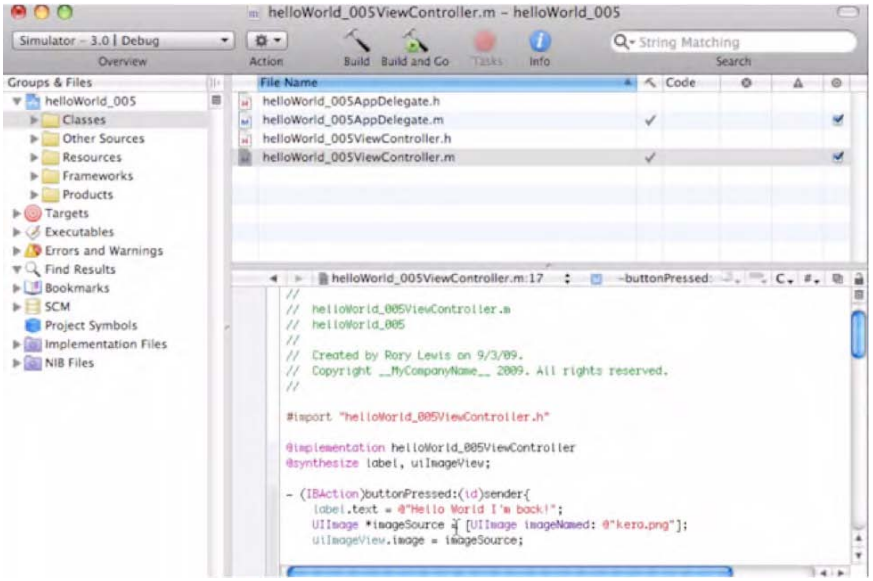
`"ui"Image *imageSource = [UIImage imageNamed: @"kera.png"];`

Evidentemente, esto provocó un error. Al detectarlo, volvimos atrás y lo corregimos convenientemente para mostrar el código adecuado tal y como se muestra. Sin embargo, decidimos mantener este error en el vídeo para ilustrar "cómo" detectar un fallo, vuelve atrás, búscalo y corrígelo.

3. El nombre de nuestro Pointer para la imagen es `uiImageView`, pero ahora el archivo de imagen `kera.png` está en el Pointer asignado para UIImage, el cual se llama `imageSource`. Necesitamos asignar este Pointer a la imagen de `uiImageView` tal y como se muestra en el siguiente código:

```
#import "helloWorld_005ViewController.h"


@implementation helloWorld_005ViewController
@synthesize label, uiImageView;

-(IBAction)buttonPressed:(id) sender{

label.text = @"Hello World, I'm back!";

UIImage *imageSource = [UIImage imageNamed: @"kera.png"];
uiImageView.image = imageSource;
}
```

Si esto no te ha quedado del todo claro, no hay problema. Hay un montón de entradas con “imagen” como parte de nombre, objeto o asociación, y esto es un concepto bastante confuso. Volveremos sobre estos conceptos de un modo más profundo en el Capítulo 7, por lo tanto, que no te quite el sueño!. La Foto 5–8 muestra cómo queda el código llegado a este punto.



F oto 5–8. Así es cómo debería quedar tu código después de que hayas completado la Sintetización y las acciones de Botón.

Guarda tu trabajo presionando ⌘S, y date una palmadita en la espalda. Has trabajado con archivos de cabecera (headers) e Implementación (implementation) de un modo mucho más profundo que en capítulos anteriores . A pesar de haber abordado con anterioridad algunas de estas funciones, les hemos hecho frente de nuevo para adentrarnos en un conocimiento y comprensión más profundos. Además, hemos abordado un conocimiento muy difícil: la síntesis.

Interface Builder: Haciendo las Conexiones

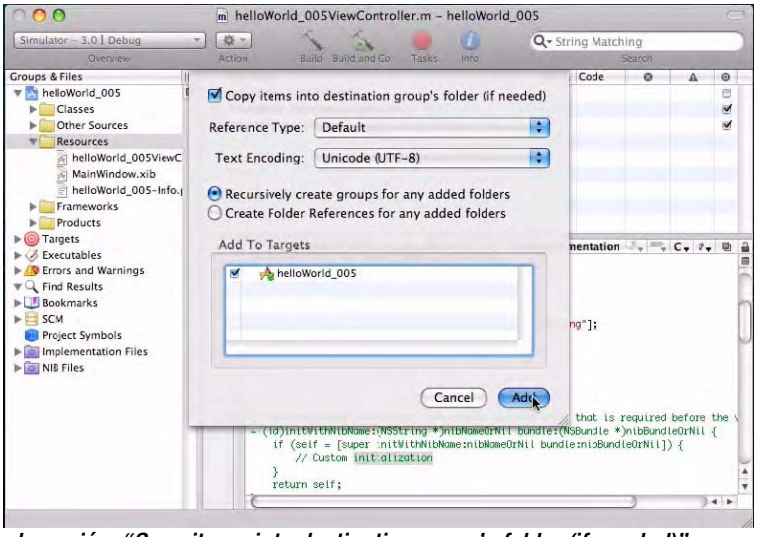
Hemos terminado nuestro código. Ahora solo necesitamos un sitio donde colocar todas nuestras imágenes dentro del directorio `Resources`, asignar el icono, y pasarnos al Interface Builder para conectarlo todo. No ha problema!

1. Para que los Pointer que acabamos de programar tengan algo a lo que referirse, necesitamos colocar los archivos de referencia en el sitio adecuado. Por tanto, arrastre su primera imagen (la imagen de fondo, por ejemplo) a la carpeta `Resources`.

NOTA: Cuando aparece marcada la carpeta, significa que el objeto está seleccionado. Fíjate en dónde está situado el cursor—es el lugar donde la carpeta va a reaccionar. Una vez se subraye, suelta el objeto dejando de pinchar el ratón.

2. Después de soltar la imagen en la carpeta `Resources`, el sistema te preguntará si la imagen estará siempre asociada a su posición en el escritorio o si se integrará con el código y será incluida en el archivo de aplicación, tal y como se muestra en la Foto 5–9.

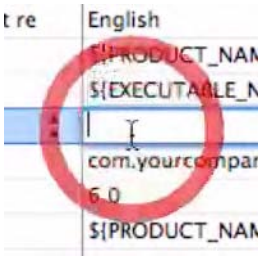
Por supuesto, queremos incorporarlo: pincha en el cuadro "Copy the items into destination's group folder". También selecciona la opción "Recursively create groups for any added folders" . Por ultimo pinchamos en Add (o presionamos Enter).



F oto 5–9. Clica la opción "Copy items into destination group's folder (if needed)" para asegurarte que todas las imagines que necesitamos en nuestro programa serán añadidas al código. También clica sobre la opción "Recursively create groups for any added folders".

3. Dado que estas acciones deben de ser aplicadas para las tres imagines, mueve las otras dos a la carpeta "Resources" tal y como has hecho con la primera.

4. Hemos creado un icono llamado `icon.png`. Este icono aparecerá en nuestro dispositivo iPhone/iPad, en lugar del icono genérico. Para ello, haz doble clic en el archivo info.plist situado en la carpeta, tal y como se muestra en la Foto 5–10, volviendo a hacer doble clic en el archivo Icono Value. En este espacio, entra el nombre de tu icono: `icon.png`. Guarda tu trabajo.

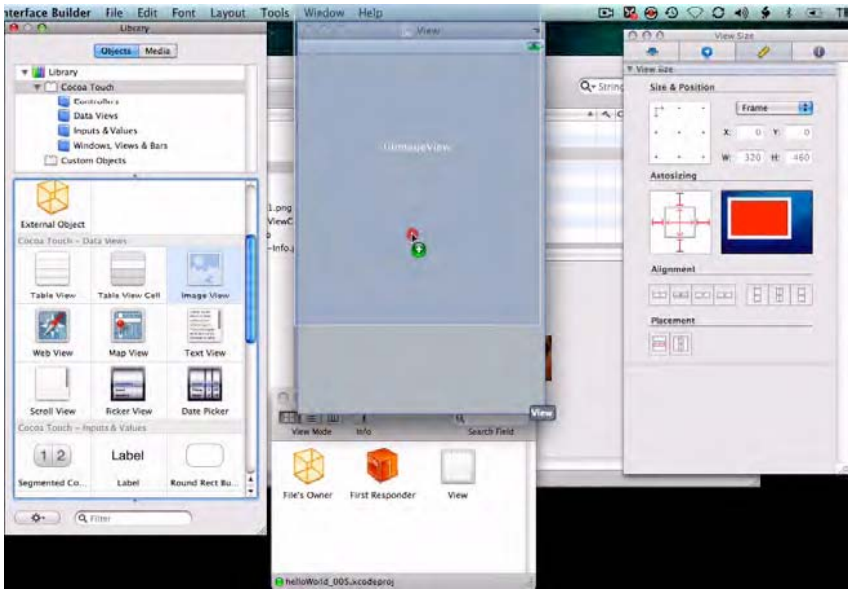


F oto 5–10. Asocia tu icono personalizado con el info.plist.

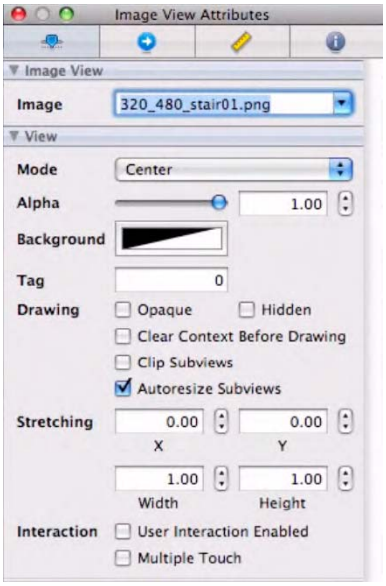
El plist (property list o lista de propiedades), es otra área en la que nos adentraremos más tarde. De momento estos preparados para ir al Interface Builder con objeto de conectar y asociar varias piezas de nuestro puzzle.

5. Recuerda que nuestra imagen de superposición aparecerá dispuesta sobre la imagen de fondo cuando el usuario pulse el botón.. Por tanto, necesitamos manejar la capa de fondo de la misma manera que en el capítulo anterior cuando usamos un wallpaper (imagen de fondo) nuestra. Desplázate hacia abajo en nuestra Library (librería) hasta el elemento carpeta Cocoa Touchem y localiza el icono Image View. Arrástralo a la ventana de Vista (View frame), como en la Foto 5–11.

6. Queremos conectar el archivo `320_480_stair.png` con nuestro Image View de modo que sea visible. Ve a la pestaña Information (Información) de la ventana Image View Attributes, abre la ventana desplegable y selecciona la imagen tal y como se hace en la Foto 5–12.



F oto 5–11. Abre el Interface Builder y arrastra una Image View (Vista de imagen) en la ventana View (Vista).



F oto 5–12. Asocia el archivo 320_480_Stair.png con el campo de imagen en la parte superior de Image View Attributes.

7. Anteriormente decidimos que cuando el usuario presione el botón, Kera Lewis debería aparecer en pantalla y decir, “Hello World, I’m back!” Decidimos que el método a emplear sería una Etiqueta (label) de ejemplo variable- con un texto asignado a “Hello World, I’m back!” tal como:

```
#import "helloWorld_005ViewController.h"

@implementation helloWorld_005ViewController
@synthesize label, uiImageView;

-(IBAction)buttonPressed:(id) sender{
label.text = @"Hello World, I'm back!";
UIImage *imageSource = [UIImage imageNamed: @"kera.png"];
uiImageView.image = imageSource;
}
```

Por tanto, arrastra una Etiqueta (label) que será nuestro ejemplo variable, y le asignaremos el texto “Hello World, I’m back!” en el Base View. Cuando lo introduzcas, repite el procedimiento que hemos seguido en casos anteriores para ajustar el texto. Posteriormente, céntralo y cambia su color a blanco en Properties (Propiedades).

Queremos que la imagen y el texto aparezcan cuando se presione el botón-por lo tanto, evidentemente necesitamos ese botón. Arrastra uno a tu capa base, y en su apartado Título escribe “Guess who’s on campus?” (Adivina quién ha llegado al Campus?), como se muestra en la Foto 5–13.

Cuando los usuarios vean este botón haciéndole la pregunta, se verán invitados a presionarlo. Cuando lo hagan, queremos ver a Kera aparecer y decir, “Hello World, I’m back!”

Foto 5–13. Arrastra un Botón a la capa de fondo.

Es posible que quieras ajustar el tamaño del botón tal y como hemos hecho con anterioridad. Por supuesto, no queremos un botón que parezca un borrón en la parte alta de la imagen, queremos que se vea adecuadamente y muestre parte de la imagen subyacente. Permaneciendo en la ventana Image View Attributes (Atributos de Vista de Imagen), desplázate hacia abajo y ajusta el regulador Alpha sobre los 0.30.

8. De Nuevo estamos usando dos `IBOutlets`, y cada categoría "suspira" por algo en el Interface Builder. Unas dicen que quieren a la Clase UILabel para usar texto al que el Pointer `*label` señala; otras quieren que la Clase UIImageView ponga una imagen situada donde el Pointer `*uiImageView` señala.

Bueno, ¿Qué hemos hecho hasta ahora en el Interface Builder? Hemos colocado la imagen de fondo e insertado un botón que accionará los dos IBOutlets. Recuerde que anteriormente introdujimos:

```
-(IBAction)buttonPressed:(id) sender
```

en el archivo de Implementación (implementation). Esta línea, de hecho, invoca a nuestros dos amigos, la Etiqueta (label) con:

```
label.text = @"Hello World, I'm back!";
```

y la imagen con:

```
UIImage *imageSource = [UIImage imageNamed: @"kera.png"];
```

De hecho, mientras trabajabas en el archivo de Cabecera (Header File), ya hiciste esto afirmando que la Etiqueta iba con:

```
IBOutlet UILabel *label
```

Y diciendo que la imagen va con

```
IBOutlet UIImageView *uiImageView
```

Entonces, lo Sintetizaste correctamente con tus dos Statements "`@property`" para la Etiqueta:

```
@property (nonatomic, retain)
IBOutlet UILabel *label
```

y para la imagen:

```
@property (nonatomic, retain)
IBOutlet UIImageView *uiImageView
```

A la vez, lo Sintetizaste con

```
@synthesize label, uiImageView
```

Lo que significa que estamos listos para la acción. Ya que acabamos de crear un botón que llamará a nuestros dos amiguetes, todo lo que necesitamos es crear la imagen y la Etiqueta (label) y asociarlas con las porciones de código apropiadas.

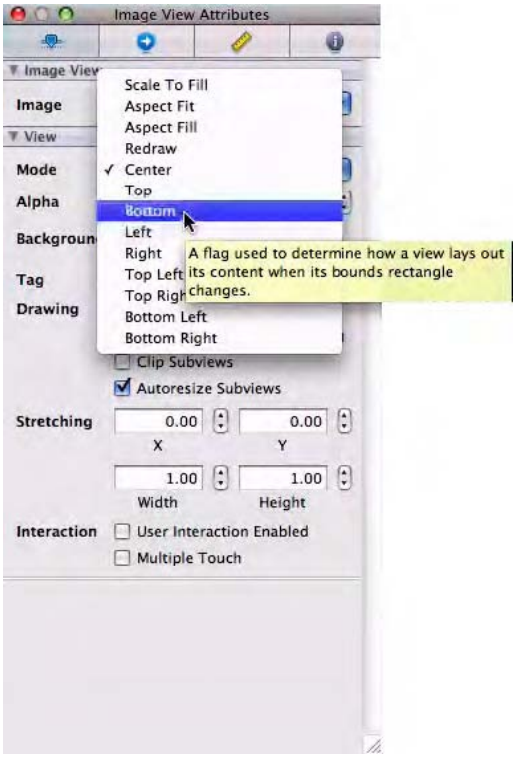
Cuando se presione el botón, la imagen `kera.png` tendrá que hacer acto de presencia. ¿Cómo llegará? Pues será llevada hasta la pantalla por mediación de una Image View. Por tanto, arrastra un Image View a la pantalla, tal y como verás en la Foto 5–14.

F oto 5–14. Desde la Sección Cocoa Touch—Data, selecciona y arrastra una Image View a tu entorno de trabajo.

9. Después de haber arrastrado una Image View a la pantalla, tenemos que enrasarlo con el borde inferior de la pantalla: no queremos la imagen en el centro de la pantalla, y además queremos que parezca que has sido proyectada desde el fondo. Una vez hayas arrastrado la imagen a la pantalla, simplemente déjala. Todavía no tenemos configurado la dimensión del lugar de emplazamiento de la imagen. Ese será nuestro próximo paso!

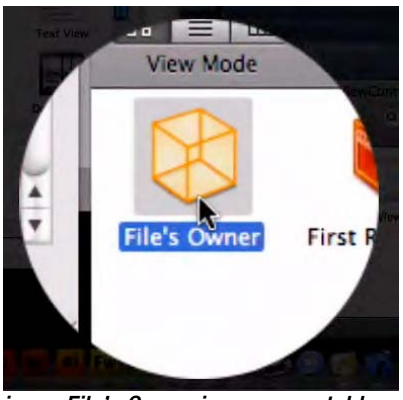
Ve a la Aplicación Image View Application y pincha sobre la pestaña View. Aquí es donde tenemos la opción de alineación. Podrás comprobar que por defecto está configurado el centrado (Center). Vamos a cambiarla por Bottom (Inferior), tal y como vemos en la Foto 5–15.

Antes de pasar al siguiente paso, tómate tu tiempo para alinear la Etiqueta y el Botón uno con otro, y a su vez, busca que ambos queden centrados en la pantalla.



F oto 5–15. Posición de UIImageView en la pantalla View, cambiar a bottom (Inferior).

10. Como aparece en la Foto 5–16, queremos conectar el File's Owner a la Etiqueta (Label). Lo haremos como lo hemos hecho en el pasado-Pulsando _+arrastrar desde nuestro icono File's Owner hasta el botón.



F oto 5–16. Selecciona el icono File's Owner icon para establecer las distintas conexiones.

11. Una vez conectes con la Etiqueta (label), accede a la sección Label del menú (Ver Foto 5–17).

F oto 5–17. Una vez conectes el botón, selecciona la opción "label" del menú.

En ejercicios anteriores, hicimos conexiones "a ciegas", ya que las fuimos hacienda tal y como te pedí. Sin embargo, y como mencionaba antes, en este Capítulo 5 tenemos que profundizar un poco más en el código. Sin profundizar demasiado en detalles de compilación, quiero darte una noción de algunos aspectos del código los cuales permiten selecciones y conexiones en el Interface Builder. Con anterioridad, asociamos la UILabel con un Pointer llamado `*label`**.** También hicimos un Pointer llamado `*uiImageView` que dirigía el `UIImageView:`

```
#import <UIKit/UIKit.h>

@interface testViewController : UIViewController {
IBOutlet UILabel *label;
IBOutlet UIImageView *uiImageView;
}

@end
```

Por tanto, conectamos el File's Owner con el User Interface (Interfaz de Usuario) llamado `label`**.** En realidad hay mucho más aspectos en este tema, pero por le momento únicamente es importante que aprendas cómo asociamos la Etiqueta `label` y la vista de imagen en virtud de conectarlas con las anteriores líneas de código.

12. Después de seleccionar la opción "label", pulsa otra vez ⌘ y arrastra desde el Botón a tu File's Owner como en la Foto 5–18, y selecciona la opción buttonPressed. Una vez más, en el pasado dejamos pasar de largo este concepto, pero veremos si conseguimos adentrarnos un poco más en estas conexiones.

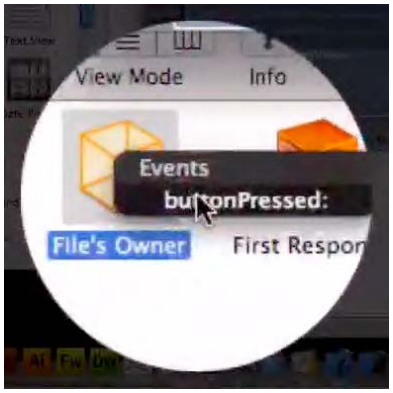


Foto 5–18. Selecciona la opción buttonPressed.

Paso Final: File's Owner & uiImageView

Anteriormente, escribiste el siguiente código:

```
#import <UIKit/UIKit.h>

@interface testViewController : UIViewController {
IBOutlet UILabel *label;
IBOutlet UIImageView *uiImageView;
}
@property (nonatomic, retain) IBOutlet UILabel *label;
@property (nonatomic, retain) IBOutlet UIImageView
*uiImageView;

-(IBAction)buttonPressed:(id)sender;

@end
```

Antes de salir, te sugerí que copiases la línea:

```
 -(IBAction)buttonPressed:(id)sender;
```

Y luego abriste tu Archivo de Implementación (Implementation File) y pegaste esta línea en él. Reemplazaste un punto y coma con corchetes, entre los cuales insertamos todas las cosas que queríamos que hiciese el botón cuando este fuese pulsado por el usuario.

```
#import "helloWorld_005ViewController.h"

@implementation helloWorld_005ViewController
@synthesize label, uiImageView;

-(IBAction)buttonPressed:(id) sender{
label.text = @"Hello World I'm back!";
UIImage *imageSource = [UIImage imageNamed: @"kera.png"];
uiImageView.image = imageSource;
}
```

En sentido amplio, quiero que percibas los dos elementos que son invocados, la Etiqueta (label) y la Imagen, están dentro de buttonPressed, y las cosas que conectamos dentro de los corchetes para el Botón.

Ya hemos conectado la Etiqueta (Label), por tanto, lo que nos queda es conectar el File's Owner con el segundo de los elementos: nuestra imagen: Allá vamos…

1. Tenemos que pulsar ⌘ y arrastrar desde el icono File's Owner hasta el View y conectarlo a la opción uiImageView en el menú desplegable, tal y como se aprecia en la Foto 5–19. Guarda tu trabajo (⌘S) y cierra el Interface Builder (⌘Q).

Foto 5–19. Conecta el icono File's Owner con el uiImageView en el menú desplegable

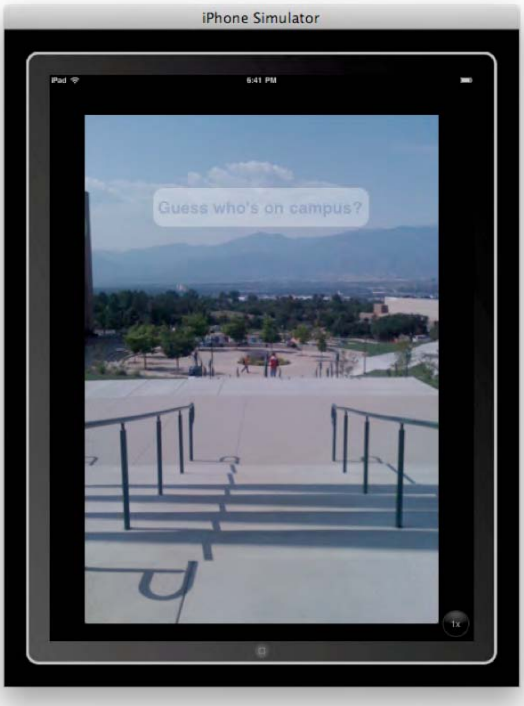
2. Presiona ⌘O para ejecutar el código.
3. Pulsa el botón en el que se lee "Guess who's on campus?" tal y como se muestra en la Foto 5–20, mi amada mujer, Kera, aparece mágicamente y dice, "Hello World, I'm back!"

F oto 5–20. La imagen de superposición y el texto aparece cuando se presiona el botón.

En la Foto 5–21, puedes ver que la vista iPad del iPhone Simulator muestra correctamente nuestra imagen base. El botón está listo para pulsar y mostrar quién ha llegado al campus…

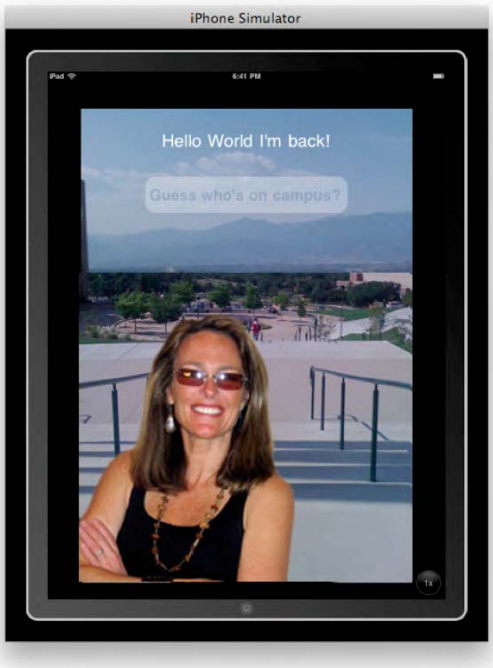
Foto 5–21. El iPad Simulator muestra en tamaño iPhone nuestra creación con nítido detalle.

Como puedes ver en la Foto 5–22, la misma imagen se muestra con zoom “2x” – que de hecho es cuatro veces el área anterior. Aprecia el botón que aparece abajo a la derecha muestra “1x”, para poder volver al tamaño original, ya que el principal propósito de este ejercicio ha sido adaptarse al formato del iPhone (320 x 480 pixels).



F oto 5–22. Presionando el botón “2x” button en la parte inferior derecha, puedes ver la aplicación en modo ampliado. Advierte que una vez pulsado el botón cambia de opción para darnos la posibilidad de volver al formato original: “1x.”

La última imagen de este ejercicio, la Foto 5–23, muestra la versión ampliada con la imagen de superposición y acompañando el texto. Kera anuncia que va a volver a la vida universitaria al pulsar el botón.



F oto 5–23. La imagen iPad del resultado de superponer la imagen: “Hello World, I’m back!”

Felicidades! Con diferencia este ha sido hasta ahora nuestro ejercicio más complicado. Hemos tenido que desenvolvernos en varias áreas técnicas del código, pero hemos sobrevivido. Si te parece bien, podemos profundizar incluso un poco más. Si prefieres no hacerlo, sáltate la sección "Profundizando en el Código" y pasa al siguiente capítulo.

Profundizando en el Código

En esta sección, ampliaremos una serie de aspectos clave que hemos encontrado en este capítulo. Quiero hablar un poco acerca de los IBOutlets y las IBActions – y en particular cómo estas dos acciones incluyen keywords … y hasta quasi-keywords. También tocaremos los Pointers y su relación con las direcciones de código.

IBOutlets e IBActions

Anteriormente trabajamos con keywords IBOutlet e IBAction, y ahora vamos a tratar una serie de conceptos relacionados con los mismos. Estrictamente hablando, están considerados como muchos programadores como "quasi-keywords."

El Appkit de Objective-C ha convertido directivas de preprocesador del lenguaje C original, tales como `#define`, en directivas de preprocesador utilizables. En idioma "geek", podríamos llamarlo "pound-define."

NOTA: Siempre que haga referencia a "iPhone" "iPad," lo estaré hacienda por extensión para cualquier otro dispositivo iPhone o iPad, incluyendo el iPAD Touch. Muchos desarrolladores iPhone/iPad recientemente han empezado a referirse al directiva preprocesador #define simplemente como "define directive.".

La directiva preprocesador `#define` le dice a la computadora que cambie una cosa por otra. ¿Es un concepto sencillo, verdad? Por ejemplo, si tuviese que programar la computadora para sustituir con el término"100" cada vez que apareciese la inserción de tu nombre, nuestro código en C, sería más o menos así::

```
#define yourName 100
```

Esto le diría a la computadora que sustituya por "100" cada vez que proceses `yourName` – una variable que reconoces ejemplos de tu nombre.

Volviendo a Xcode y a nuestro tema, en este contexto los quasi-keywords `IBOutlet` e `IBAction` no son realmente definidos como nada. En otras palabras, no hacen nada sustancial por el compilador, que es el núcleo de la computadora.

Los quasi-keywords son banderas, y sin embargo son importantes para la comunicación con el Builder. Cuando el IB lee los quasi-keywords `IBOutlet` e `IBAction`, este moviliza parte de su código interno y lo prepara para realizar tareas específicas. Consigue prepararse para hacer frente a variables de instancia, y a todos los enganches y conexiones que podemos hacer en el ámbito de la programación.

Más Acerca de los Pointers

Para muchos estudiantes de programación es difícil entender el concepto de los "Pointers"— también conocido con el nombre de "indirection" (direccionamiento indirecto). No es fácil el explicar esta idea, ya que se trata de una de las partes más sofisticadas del lenguaje de programación C.

Al principio de este capítulo, os expuse el ejemplo en el cual pillábamos a un criminal haciendo de las suyas, y luego llamábamos a la policía. Al llegar la policía, le señalábamos al criminal-ya que él, y no tú, es la persona que puede proceder a un arresto. Este ejemplo es bastante útil para muchos estudiantes, pero vamos a profundizar un poco más en el tema.

Si preguntases a un profesor de Ciencias de Computación que es un "Pointer", probablemente te respondería algo parecido a que un "Los Pointers mantienen la dirección de una variable o un método."

Te preguntarás: "¿La dirección?". Bien, toma en consideración este Nuevo ejemplo para seguir con la explicación:.

¿Alguna vez has visto una película en la que un detective o varias personas viajan por muchos lugares buscando pistas, mapas del tesoro, un cuadro perdido o la típica hija secuestrada? A veces , encontrarán una pista en una huella, un recibo o incluso en un sobre con una hoja de papel que contiene un mensaje cifrado- y esa pista es la que lleva al protagonista a conseguir su objetivo- o a encontrar aquellos objetos que estaban buscando.

Pues bien, a esos objetos podemos llamarlos Pointers; ellos indican el próximo lugar a donde ir – o una solución a un determinado problema. No tienen por qué ser necesariamente la última pista determinante que te lleve directamente al objetivo, ya que pueden darte nuevas pistas, direcciones o sitios donde seguir la búsqueda.

Por tanto, lo que el profesor de Ciencias de Computación quiere decir, es que los Pointers no contienen los elementos a los que te dirigen, contienen las ubicaciones en el código-las direcciones- de los objetos deseados, acciones o entradas. Esta importante característica hace de la familia de lenguajes C una herramienta muy poderosa.

Esta idea tan simple consigue convertir de modo muy eficiente tareas complejas en tareas bastante más simples. Los Pointers pueden transformar valores en tipos y argumentos en funciones, representar grandes masas de números y manipular el cómo podemos gestionar la memoria en una computadora. Muchos de vosotros quizá estéis pensando que los Pointers son similares a las variables en el mundo del Álgebra: ¡Correcto!

En nuestro primer ejemplo, un Pointer permitía que un ciudadano desarmado detuviese a un malvado criminal usando la indirección ("indirection")- es decir, llamando a la policía para que se personase y solventase el problema. (Sí, lel término "indirección" – "indirection"- es WIn our first analogy, a Pointer enabled an unarmed citizen to arrest a dangerous criminal by using indirection—that is, by calling the police to come and solve the problem. (Yes, the term "indirection" es una definición extraña, ya que en realidad estamos siendo dirigidos hacia el objetivo.)

Considerando el siguiente ejemplo, usamos un Pointer para dirigirnos a la cantidad de dinero que tenemos en una cuenta bancaria. Para hacer esto, definamos una variable llamada `bankBalance` de esta manera: `int bankBalance = $1,000;`

Ahora, marquemos otra variable intermedia y llamémosla `int_Pointer`. Esta variable nos ayudará a usar la indirección para conectar indirectamente el valor de `bankBalance` por la declaración: `int *int_Pointer;`

La estrella, o asterisco, le dice a la familia del lenguaje C que nuestra variable `int_Pointer` está habilitada para indirectamente acceder al número correspondiente a nuestra cantidad de dinero en nuestra variable (placeholder o marcador de posición): `bankBalance.`

Para finalizar, quiero advertirte de que nuestra profundización en el código en relación a este tema no es una exploración reTo close, I want to remind you, and to acknowledge, that our digging around here is not an exhaustive or rigorous exploration into these topics … just a fun tangent into some related ideas. At this point, there is no reason for you to be bothered if you don't fully understand Pointers. Seeds have been planted and that's what counts for now!

En el Siguiente Capítulo...

En el Capítulo 6, aumentaremos el nivel de complejidad: Ampliaciones visuales cambiantes . Examinaremos como un equipo de caracteres o reglas dentro de tu código trabajarán juntos para dirigir un resultado, una serie de resultados, que den al usuario sensación de flujo continuo.

Aprenderás acerca de delegadores y controladores de cambios visuales, Clases y Subclases y "Lazy Loads". Entraremos en la profundidad de los archivos `.xib` files, examinaremos el concepto de desasignación de memoria y aprenderemos acerca de los comentarios de código incrustados. Esto cada vez está más interesante …

Adelante con el próximo capítulo!.

Capítulo 6

Cambio de Vista con gráficos múltiples

En este capítulo exploraremos uno de los aspectos más notables del iPhone: su habilidad única para cambiar suavemente de una vista a otra. Todos hemos visto anuncios maravillosos del iPhone y el iPad en televisión, en los que los dedos de alguien dirigen un sorprendente flujo de imágenes vívidas, dentro de aplicaciones interactivas, y hacen que una vista se deslice o ruede directamente a otra, dando la impresión de arte interpretativo. El concepto que hay detrás de esto es lo que Apple llama metodología de Cambio de Vista. Como profesor de informática, me he enterado de varias dificultades en la enseñanza –y aprendizaje- de la metodología de Cambio de Vista.

Tengo la intención de beneficiarme de la interesante variedad de resultados que he experimentado cuando presentaba a mis alumnos algunos de los atajos que esa gente inteligente de Apple nos han dado. Mientras presento conceptos y técnicas nuevos, contrastaré varios caminos ciertos y probados hacia el objetivo de crear una aplicación con las capacidades que he descrito. Específicamente, se familiarizará con la plantilla de barra de pestaña de aplicaciones; crea un SDK Xcode que dispone todo con la apariencia y la sensación de una barra de pestaña simple. Para nuestro objetivo de codificar vistas de cambio con gráficos múltiples, apenas tendrá que programar nada – el texto estándar dentro de la plantilla hace la mayor parte del trabajo.

Irónicamente, este “camino más fácil y suave” fue un desastre para algunos de mis alumnos. Ahora me doy cuenta de que cuantos más atajos se toman, sin la debida exploración del código subyacente, algunos estudiantes de programación se encuentran confundidos. Sabía que implementar los cambios de vista en Objetive-C no era de ninguna manera una empresa trivial, y había suministrado a mi clase originalmente una forma de hacer palanca por medio de la plantilla de barra de pestaña de aplicaciones. Cuando esto no dio los resultados positivos que esperaba, volví a la mesa de dibujo.

Di un taller de sábado en el que dividía la lección en tres secciones –cada una de las cuales sería una variación de mi plan original. Cada uno de los tres métodos alternativos usaría las mismas dos imágenes descritas en la Figura 6-1.

NOTA: Algo que puede que haya notado es que este es un capítulo____. Esto es debido lo que acabo de decir –que es en realidad un capítulo tres-en-uno. Puesto que será realmente instructivo mostrarle el camino largo y detallado, el corto y simple y el combinado… para encontrarse con el mismo objetivo básico, he decidido mantenerlo todo en el mismo capítulo. De todos modos, espero que vaya a su ritmo… Por favor, trate el Capítulo 6 como un largo sub-capítulo seguido de un sub-capítulo corto, y seguido de un sub-capítulo mediano.

Como puede ver aquí, hay tres formas en las que podemos conseguir nuestros objetivos de cambio de vista en este capítulo. En cada posibilidad, empezaremos por la primera vista –una foto de mi abuelo cuando era un guapo soltero, y cambiaremos a la segunda vista –de mi abuelo y mi abuela, que me criaron como hijo. Cuando era niño, era una figura mayor que la vida, mi primer héroe, y por tanto he etiquetado este ejercicio con el apodo de otro "tipo listo", por el que aparentemente le llamé en la ocasión: Einstein.

El primer modo en que llevaremos a cabo el cambio de vista (izquierda) tiene el "USTED" allí porque usted, mi querido amigo será quien escriba el código. Note que los botones/pestañas son de un color azulado, que es como los creará.

El segundo modo (centro) está representado con un icono de XCODE que nos lleva de una imagen a la siguiente. Este camino es el que usa la plantilla simple de aplicación de vista de pestaña. Fíjese que las teclas de pestaña no son azuladas, sino negras, ya que es la predeterminada en el código de texto estándar de esta utilidad

La tercera forma de cambiar de vista (derecha) representa una combinación de los otros dos enfoques. Note que usaremos pestañas negras hechas de antemano, pero habrá áreas que construirá y personalizará usted. Este camino es un enfoque intermedio, una mutación entre USTED y Xcode.

Así fue como procedimos en ese sábado por la mañana en ese taller, y funcionó como por encanto. Una idea más sobre este triple método: cada alumno aún expresó una preferencia por un método dado, pero hubo unanimidad en la apreciación de la presentación de la lección, los tres uno junto al otro.

Figura 6–1*. Izquierda, el "camino largo"; centro, el "simple"; derecha, el "combinado"*

Antes de empezar

Hay tres puntos más a tener en cuenta antes de empezar:

Punto 1: No se ponga demasiado cómodo. Algo que noté en las sesiones de sábado fue que los estudiantes pasaban por encima de pasos que pensaban que eran iguales pero que eran ligeramente diferentes. Por tanto, por favor, ponga mucha atención a pasos que puedan resultarle familiares.

Punto 2: Supere la cuestión de usar las mismas imágenes. Cuando acabó el fin de semana, un alumno se quejó, "Dr. Lewis, las imágenes son aburridas ¿No podemos usar nuestro propio material? Mi respuesta fue "Todos estamos usando las mismas imágenes "aburridas" para que nos podamos centrar en la mecánica de cómo funcionan los tres enfoques de creación de Cambio de Vista." Si conseguimos que las dos piezas del puzzle sean consistentes seremos capaces de centrarnos en la tarea.

Punto 3: Las tres versiones y sus capturas de pantalla. The three versions and their screencasts. Podría ser instructivo verme recorrer las tres metodologías en tres películas separadas. Las trataré aquí en detalle, explicando partes que salté a propósito en las películas para ir más rápido. Adelante entonces, vea los tres vídeos. Nos encontraremos aquí luego. Tome nota también de los enlaces "Código correcto" para cada ejercicio propuesto aquí para referencias futuras:

1. Desde el cero: einSwitch_001

Video: `http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/006_einSwitch_001.htm`

Código correcto:
`http://www.rorylewis.com/cCode/006a_einswitch01.zip`

2. Barra de pestañas: einSwitch_002

Video: `http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/007_einSwitch_002.htm`

Código correcto: `http://www.rorylewis.com/xCode006b_enswitch02.zip`

3. Barra de pestañas personalizada: einSwitch_003

Video: `http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/008_einSwitch_003.htm`

Código correcto: `http://www.rorylewis.com/xCode006c_enswitch03.zip`

OK … respire hondo, apague su móvil y cálmese. La siguiente parte del viaje le dará algunos derechos a jactarse en la comunidad de Xcode. Pronto podrá decir: "Si, programé en Objective-C para hacer cambio de vistas. Sé hacerlo de forma completa, de forma abreviada o con plantillas personalizadas. ¡Deje sitio!"

einSwitch_001—una aplicación basada en ventanas

Vuelva un momento si quiere, y recuerde la analogía de la exposición de coches, con seis estilos de cuerpo diferentes. Como ve en este encabezamiento, estamos saltando a uno de los vehículos más versátiles y robustos: la aplicación basada en ventanas. Aunque conduciremos al mismo destino en cada sección de este capítulo de tres partes, saltaremos un poco de un coche a otro. No deje que esto le distraiga; en lugar de eso que esto sea una aventura de prueba de conducción de modelos diferentes.

Preliminares.

Como siempre, empezaremos con un escritorio limpio. Entonces nos prepararemos para el primer ejemplo añadiendo cuatro elementos a nuestro escritorio para que se unan al siempre presente icono Macintosh HD: tres archivos de imágenes .png y un archivo de texto. Como se muestra en la Figura 6-2, descarga el archivo zip desde `http://www.rorylewis.com/xCode/006_Chapter 6h_EinSwitch01.zip`. Una vez hayas descargado el archivo, tendrás que ir a la carpeta Downloads.

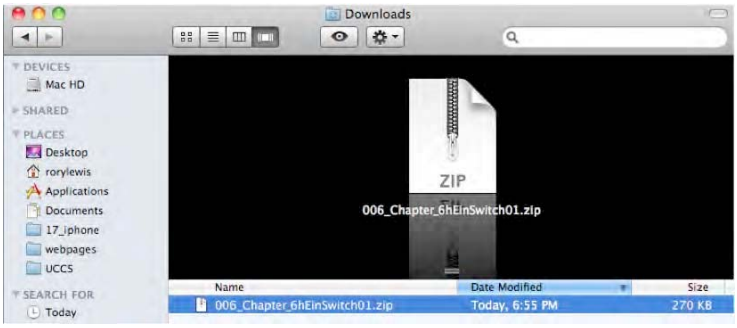


figura 6-2, descárguelo del archivo zip localizado en http://www.rorylewis.com/xCode/006_Chapter 6h_EinSwitch01.zip. Una vez que haya descargado el archivo zip, puede que necesite ir a su carpeta de descargas.

Cuando haya localizado su archivo zip, arrástrelo al escritorio tal y como se muestra en la Figura 6-3. Haga doble clic en el archivo para acceder a la carpeta `006_Chapter_6hEinSwitch01` y extraiga los cuatro elementos a su escritorio, como se muestra en las Figuras 6-3 y 6-4.

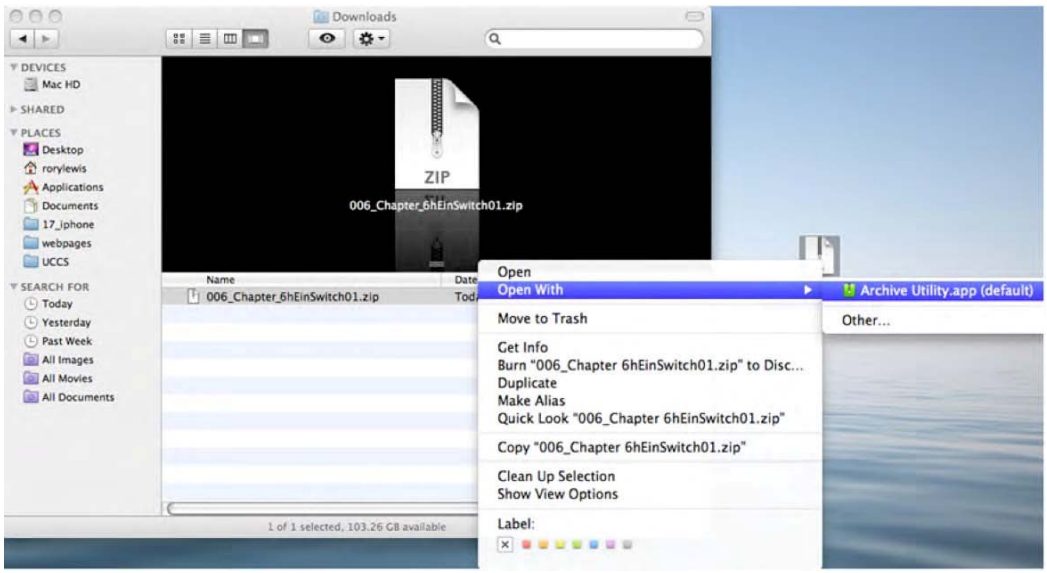


Figura 6–3. Extraiga 006_Chapter_6hEinSwitch01.zip de la carpeta Descargas y muévalo al escritorio.

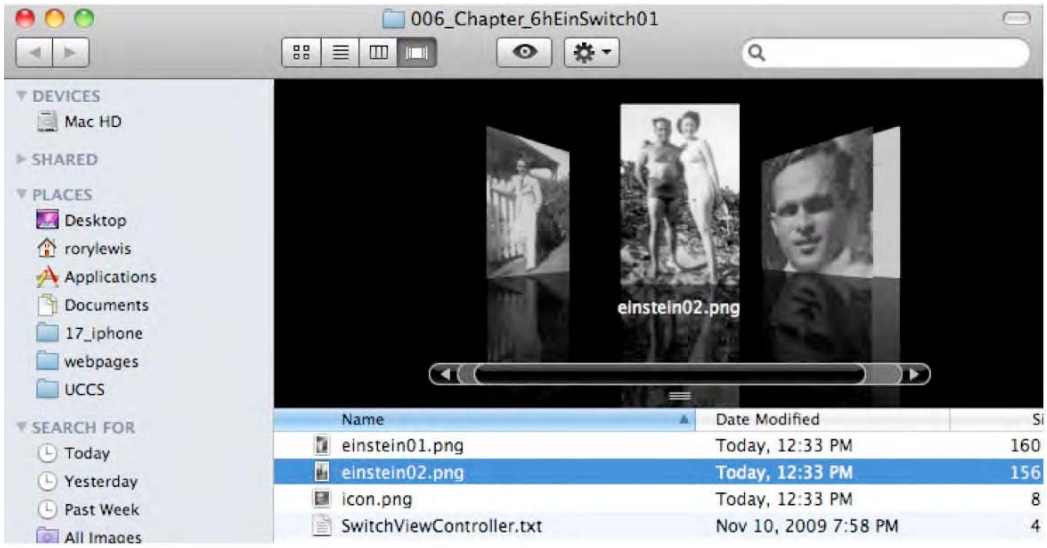


Figura 6–4. Anote los cuatro archivos localizados en la carpeta 06_Chapter_6hEinSwitch01.zip.

NOTA: Cambié la imagen del escritorio en las Figuras 6-3 y 6-4 porque el fondo negro se mezclaba con el fondo negro de la carpeta. No deje que esto le confunda.

Los dos primeros elementos son las fotografías en blanco y negro de mi abuelo, solo y con su novia. Como dije en el Punto 2, de verdad quiero que use esas imágenes que le he suministrado, ya que hará que el resto del capítulo fluya más fácilmente y se distraiga menos. ¡Una variable menos de la que preocuparse.

El tercer elemento en el archivo 006_Chapter_6hEinSwitch01.zip es la imagen del icono de su aplicación, con dimensiones menores, por supuesto. Una vez tenga las tres imágenes: la capa de fondo –mi abuelo de soltero (Figura 6-5), la capa superior –mi abuelo y su novia (Figura 6-6), y el icono (Figura 6-7), guárdelos en su escritorio.

Figura 6–5. Usaremos esta imagen como capa de fondo de nuestra aplicación de cambio de vista..

Figura 6–6. Aquí está la imagen superior, ¡un buen cambio!

Figura 6–7. Esta imagen más pequeña es para el icono de pantalla.

El cuarto elemento en el archivo zip es el documento `SwitchViewController.txt`, que tiene el código en lenguaje estándar que verá una y otra vez. En este capítulo se le mandará copiar y pegar trozos de este texto en su código para manejar ciertas funciones. Volveremos a ello y se lo explicaré con detalle, pero en este punto de su viaje en el código simplemente guárdelo en el escritorio.

En este punto, cuando insertemos este código preparado de antemano, lo llamaremos una "Lazy Load" Puede ver el aspecto que tiene más abajo… Señale esta página, dentro de poco me volveré a referir a este código –cuando haya que copiar y pegar.

```
#import "SwitchViewController.h"
#import "Ein1Controller.h"
#import "Ein2Controller.h"

@implementation SwitchViewController
@synthesize ein1Controller;
@synthesize ein2Controller;

-(void)viewDidLoad
{
Ein1Controller *ein1Controller = [[Ein1Controller alloc]↩
  initWithNibName:@"Einstein1View"
bundle:nil];
        self.ein1Controller = ein1Controller;
        [self.view     insertSubview:ein1Controller.view     atIndex:0];
        [ein1Controller release];
}

-(IBAction)switchViews:(id)sender
{
        // Lazy load - we load the Einstein2View nib the first time the button↩
 is pressed
        if (self.ein2Controller == nil)
        {
                Ein2Controller *ein2Controller =
                [[Ein2Controller alloc] initWithNibName:@"Einstein2View"
                                                bundle:nil];
                self.ein2Controller = ein2Controller;
                [ein2Controller release];
        }

        if (self.ein1Controller.view.superview == nil)
         //This is with no animation
        {
                [ein2Controller.view removeFromSuperview];
                [self.view insertSubview:ein1Controller.view atIndex:0];
        }
        else
        {
                [ein1Controller.view removeFromSuperview];
                [self.view insertSubview:ein2Controller.view atIndex:0];

         }
}
-(id)initWithNibName:(NSString *)nibNameOrNil
                          bundle:(NSBundle *)nibBundleOrNil {
         if (self = [super initWithNibName:nibNameOrNil
                                    bundle:nibBundleOrNil]) {
         }
         return self;
}
-(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)↩
interfaceOrientation {
return (interfaceOrientation == UIInterfaceOrientationPortrait);
}

-(void)didReceiveMemoryWarning {
       [super didReceiveMemoryWarning];
       if (self.ein1Controller.view.superview == nil)
               self.ein1Controller = nil;

        else

               self.ein1Controller = nil;

}
```

```
-(void)dealloc {
      [ein2Controller release];
      [ein1Controller release];
      [super dealloc];
}

@end
```

Llame a su proyecto “einSwitch01”

NOTA: Antes de seguir, tiene que BORRAR la carpeta `O6_Chapter_6hEinSwitchO1`. Ya ha sacado las tres imágenes y el archivo, por lo tanto está vacía. Bórrela. Esto es crítico. Si abre Xcode y guarda uno de sus archivos con el mismo nombre que el de arriba habrá una confusión masiva. Arrastre la carpeta `O6_Chapter_6hEinSwitchO1` a la papelera de reciclaje ahora junto con el zip original, si no lo ha borrado ya.

Abra Xcode e introduzca ⌘⇧N, como se muestra en la Figura 6-8; esto abrirá una ventana de Proyecto Nuevo, en la que usted hará clic en la plantilla de aplicaciones basadas en ventanas (Window-based Application template). Como sabe, puede que no esté predeterminada esta opción, así que asegúrese de posibilitarla. Llame a su nuevo programa “einSwitch01,” como se muestra en la Figura 6-9; entonces guarde su aplicación basada en ventanas en su escritorio seleccionando ⌘S. Voy a ser muy estricto con el protocolo de nombrar archivos aquí porque haremos tres aplicaciones similares que tendrán etiquetas similares. Su código “esperará” que los elementos se llamen como les he llamado.

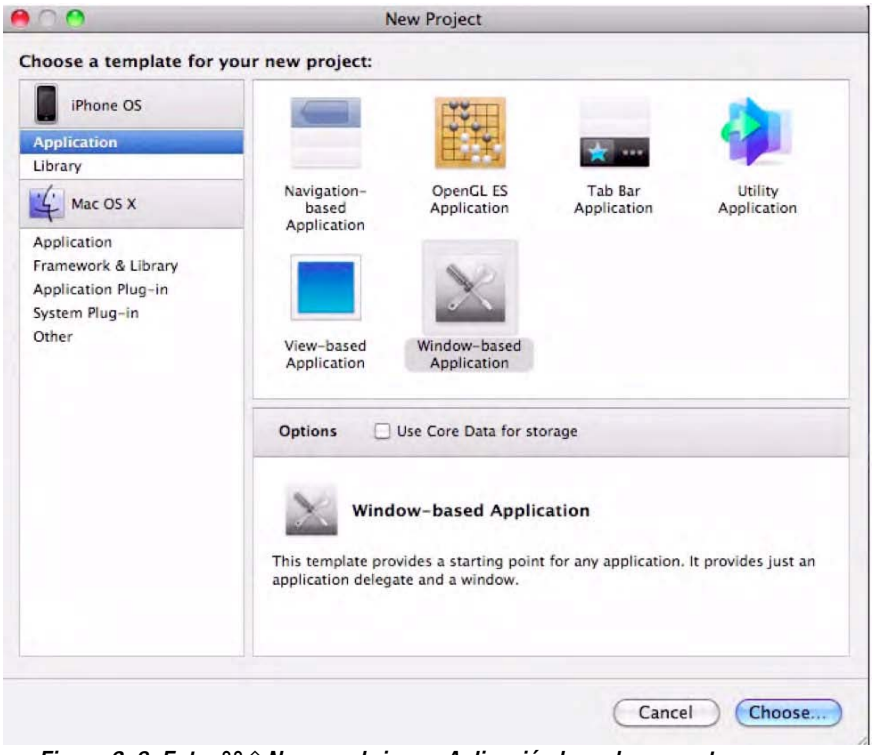


Figura 6–8. Entre ⌘⇧N para abrir una Aplicación basada en ventanas nueva.

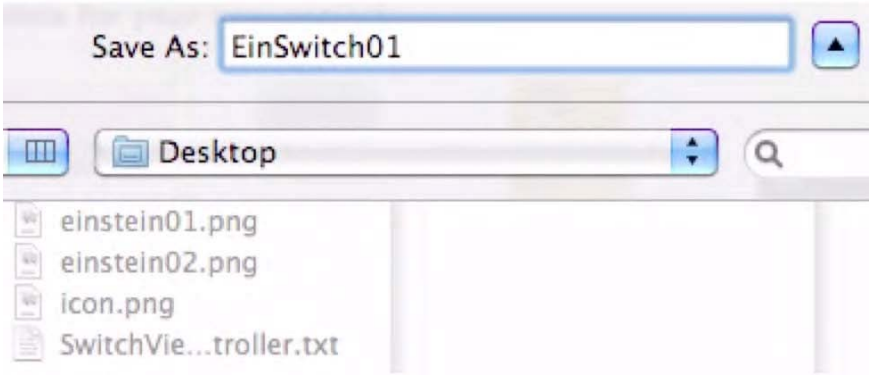


Figura 6–9. Guarde su nueva aplicación basada en ventanas "EinSwitch01" en su escritorio pulsando ⌘S.

PLAN DE ACCIÓN: ¿CÓMO CONTINUAREMOS?

NOTA: Puede saltar hacia delante a la sección may Crear la primera UIViewController Subclass y saltarse esta digresión. Ya que este es el código más complejo que se habrá encontrado hasta ahora, usaremos un plan de acción. La mayoría de los alumnos lo encuentra útil –como todos cuando se trata de proyectos a gran escala.

Sentémonos y diseñemos un plan de acción. Este proyecto permitirá cambiar entre dos vistas, cada una de las cuales contiene una foto de mi abuelo en diferentes etapas de su vida. Mire la Figura 6-10 para ver un diagrama de una analogía que espero le sea útil. El personaje del nivel superior, einSwitchAppDelegate, le dice al SwitchViewController cuando activarse. El personaje del segundo nivel le dice al Ein#Controllers que use su `.xib` tools para sostener sus fotos respectivas. También puede ver como usamos Xcode para programar einSwitchAppDelegate y SwitchViewController y entonces como usamos Interface Builder (constructor de interfaz) para trabajar con las dos Ein#Controllers y los archivos `.xib`.

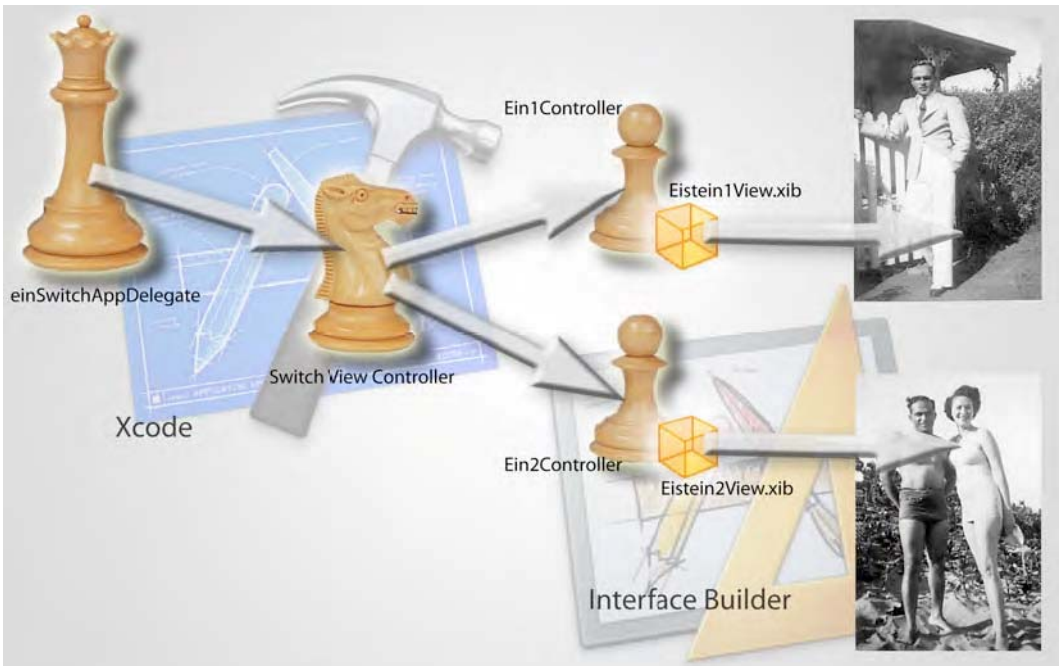


Figura 6–10. Aquí hay un plan de acción de la aplicación einSwitch001.

He aquí un resumen:
`Head Honcho Delegator`: Delega el trabajo en el Switch View Controller.
`Switch View Controller`: dice qué Controller aparecerá en la pantalla
`Einstein 1 Controller`: Lleva la primera foto
`Einstein 2 Controller`: lleva la segunda foto.

Por desgracia no podemos usar la nomenclatura mencionada anteriormente. En Objective-C, como en muchos otros lenguajes informáticos, tenemos que usar un estilo especial de designación: *camelCase*. Refiriéndonos a la pequeña "joroba" en el medio, es el nombre de la forma en que los codificadores tecnoadictos usan para escribir palabras o frases compuestas. La primera letra normalmente no es mayúscula, los elementos están unidos sin espacios, y la inicial de cada elemento es una mayúscula *dentro* del compuesto. Como algunos de ustedes notarán, ya venimos usando esta nomenclatura a lo largo del libro. Es un patrón muy fácil y lo pillarán inmediatamente.

Si por ejemplo, tuviera que escribir código para que le sacara un mapa de un Google parser, podría llamarle `getMapFromGoogleParser.` Esto es algo personal, a su elección… pero intente ser consistente. Yo normalmente uso minúsculas para la primera palabra siempre que no sea un nombre propio (por ejemplo, el nombre de una ciudad o una persona) o una Clase. Usando estas guías básicas, renombremos los jugadores en nuestro código.

En esencia, siga la guía de Apple para dar nombre a las entidades. En general, al poner sus propios nombres ¡queLaPrimeraLetraNoSeaMayúscula! He aquí los nombres que usaremos:

`einSwitchAppDelegate:` Delega el trabaja al Switch View Controller
`SwitchViewController:` Dice qué Assistant Controller aparecerá en la pantalla.
`Ein1Controller:` Contiene la foto de la capa de fondo.
`Ein2Controller:` Contiene la foto de la capa superior.

Ahora que hemos puesto nombre a nuestros personajes, tenemos que definir las herramientas que usan los portadores para llevar esas imágenes. Recogerán y presentarán las fotografías con archivos nib (.xib), a los que hemos llegado a querer. Esto es lo que tenemos:

- `einSwitchAppDelegate`
  - `SwitchViewController`
    - `Ein1Controller`
    - Nib file: `Einstein1ViewwxibbEin2Controller`
      - Nib file: `Einstein2View.xib`

El último paso para identificar todos los archivos que necesitaremos es recordar que cada uno de esos archivos necesita tanto una cabecera como un archivo de *implementación,* como se muestra en la Figura 6-11. Al rediseñar nuestro plan de acción tenemos:

- einSwitchAppDelegate.h and einSwitchAppDelegate.m
  - SwitchViewControllerr.h and SwitchViewController.m
    - Ein1Controller.h and Ein1Controller.m
      - Einstein1View.xib
    - Ein2Controller.h and Ein2Controller.m
      - Einstein2View.xib

Figura 6–11. Cada uno de estos necesita su propia cabecera (.h) y su propio archivo de implementación (.m).

Guapo, ¡eh! Ahora tenemos que crearlos. Específicamente, tenemos que crear tres subclases en de Cocoa Touch llamadas `SwitchViewController`, `Ein1Controller`, y `Ein2Controller`, y también crearemos la cabecera y el archivo de implementación de cada jugador. Entonces tenemos que hacer dos archivos nib: `Einstein1View.xib` and `Einstein2View.xib`.

Creación de la primera subclase UIViewController

Una vez que ha guardado su aplicación basada en ventanas como `einSwitch01` en el escritorio, haga clic en la carpeta de Classes en la barra lateral de Grupos y Archivos. Introduzca ⌘N para abrir la ventana de diálogo de Archivo Nuevo (New File), como se muestra en la Figura 6-12. Seleccione el elemento Cocoa Touch Class en la barra iPhone OS, y haga clic en el icono UIViewController. Marque la casilla Also Createbox "SwitchViewController.h", y haga clic en "Finish" para guardar el archivo como `SwitchViewController.m`. Ver figura 6-13

Figura 6–12. Crear la primer de tres Cocoa Touch UIViewController subclasses.

Figura 6–13. Guardar el archivo como "SwitchViewController.m." Recuerde activa la casilla Also Create "SwitchViewController.h."

En los otros proyectos de este capítulo, seleccionaremos la opción With XIB for User Interface, pero ahora mismo vamos a construir nuestros archivos nib desde el principio. Una vez que lo haya hecho aquí, se sentirá bien sabiendo lo que pasa cuando deja que Apple construya los archivos nib. De este modo, siempre puede cambiar y añadir todos los extra que quiera. Por otra parte, si siempre tiene ya hecho este archivo clave nunca sabrá lo que pasa por dentro.

Según nuestro plan de acción…

- `einSwitchAppDelegate.h` and `einSwitchAppDelegate.m` **Hecho**
    - `SwitchViewController.h` and `SwitchViewController.m` En proceso ahora
        - `Ein1Controller.h` and `Ein1Controller.m`
            - `Einstein1View.xib`
        - `Ein2Controller.h` and `Ein2Controller.m`
            - `Einstein2View.xib`

Creación del Ein1Controller

Aquí repetimos lo que acabamos de hacer para crear el segundo y el tercero de las tres subclases Cocoa Touch UIViewController. Las llamamos `Ein1Controller` and `Ein2Controller.` Recuerde –estos dos personajes que acabamos de nombrar son los subordinados de `SwitchViewController,` y presentarán las fotos para el uso de su aplicación. Si puede hacer esto por su cuenta, adelante, inténtelo. Si no, simplemente siga leyendo y sígame. Verá que sólo estamos repitiendo los pasos de nuevo.

Una vez que ha guardado `SwitchViewController,` introduzca `⌘N` para abrir una ventana de diálogo de archivo nuevo. Seleccione el icono Cocoa Touch en la barra iPhone OS, haga clic en el icono UIViewController subclass, pulse Intro (o seleccione el botón siguiente –next), y guárdelo como `Ein1Controller.` Ver Figura 6-14.

Figura 6–14. Crear la segunda de las tres subclases UIViewController de Cocoa Touch.

Según nuestro plan de acción…

- einSwitchAppDelegate.h and einSwitchAppDelegate.m **Hecho**
    - SwitchViewController.h and SwitchViewController.m **Hecho**
        - Ein1Controller.h and Ein1Controller.m En proceso
            - Einstein1View.xib
        - Ein2Controller.h and Ein2Controller.m
            - Einstein2View.xib

Comprobar la cabecera y los archivos de implementación.

Compruebe rápidamente que se han creado tanto la cabecera como los archivos de implementación para Ein1Controller. Este paso puede parecer redundante, pero a menudo nos olvidamos de marchar las casilla apropiadas, como la necesaria para crear el archivo .h. Si puedo ayudarle a adquirir el hábito de comprobar dos veces este requerimiento aquí, a la larga ahorrará tiempo. Una vez que avance y escriba acciones en un archivo sin el otro, puede ser una lata intentar volver a crear archivos adecuados sobre la marcha.

Mientras estamos en ello, vamos a revisar también la relación entre la cabecera de un elemento y los archivos de implementación. Recuerde, del capítulo 1, que todas las clases tienen dos partes: un archivo de cabecera (.h) y un archivo de implementación (.m). ¿Se acuerda de que le pedí que leyera en voz alta la frase siguiente? “Informamos al ordenador mediante los archivos de cabecera sobre los tipos de comandos que queremos que ejecute en los archivos de implementación.” Este es todavía el caso, pero ahora me gustaría añadir una pequeña frase delante: “Cada clase es el producto de sus archivos de cabecera (.h) e implementación (.m). Informamos al ordenador en nuestros archivos de cabecera de los tipos de comandos que queremos ejecutará en los archivos de implementación.”

¡Primero, informamos; después ejecutamos!

Creación del Ein2Controller

Una vez que ha guardado el Ein1Controller, seleccione ⌘N para abrir una nueva ventana en la carpeta de Clases para abrir la ventana de diálogo Archivo Nuevo. Seleccione el icono Cocoa Touch en la barra iPhone OS, haga clic en el icono UIViewController subclass, pulse intro (o seleccione el botón siguiente), y guárdelo como Ein2Controller. Ver Figura 6-15

Figura 6–15. Crear la tercera de las tres subclases Cocoa Touch UIViewController

Según nuestro plan de acción…

- `einSwitchAppDelegate.h` and `einSwitchAppDelegate,m` **Hecho**
  - `SwitchViewController.h` and `SwitchViewController.m` **Hecho**
    - `Ein1Controller.h` and `Ein1Controller.m` **Hecho**
      - `Einstein1View.xib`
    - `Ein2Controller.h` and `Ein2Controller.m` En proceso
      - `Einstein2View.xib`

Asegúrese de que las imágenes están incrustadas

Creamos los archivos `einSwitchAppDelegate.` y `einSwitchAppDelegate.m` cuando creamos la aplicación basada en ventanas al principio mismo de este ejemplo. Acabamos de terminar de crear las subclases Cocoa Touch UIViewController y las hemos llamado `SwitchViewController`, `Ein1Controller`, y `Ein2Controller`. En este punto, hemos creado todos los personajes de nuestra obra, pero aún necesitamos equipar algunos de ellos con las herramientas que usarán para mostrar las fotos de las capas superior y de fondo –i.e. nuestras imágenes de cambio de vista.

Para esto crearemos dos grupos de herramientas, archivos `.xib`, que usarán cada uno de nuestros ayudantes. En el diálogo de Archivo Nuevo, ir a la capeta de Recursos, hacer clic, introducir `⌘N`, y seleccionar la carpeta User Interface. Entonces seleccionar el icono Vista XIB, como se muestra en la Figura 6-17, e introducir inmediatamente `⌘N` para crear el próximo archivo nib, `Einstein2View.xib.` Guárdelo en dos archivos `.xib, Einstein1View.xib,` como se muestra en la Figura 6-17, e inmediatamente déle al intro para crear su próximo archivo nib, De nuevo, seleccione la carpeta User Interface en el diálogo de Archivo Nuevo y seleccione el icono Ver XIB.

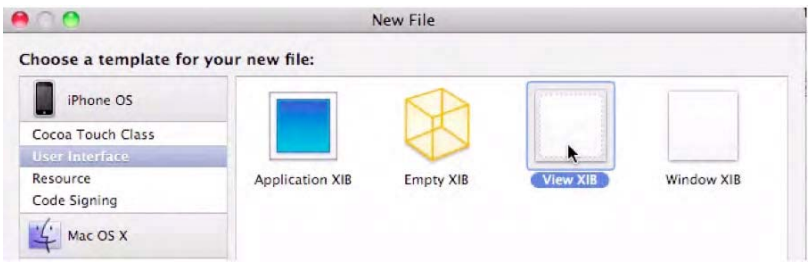


Figura 6–16. Seleccione la opción Vista XIB para crear el primero de sus dos archivos nib

Figura 6–17. Guarde el primero de sus dos archivos nib, "Einstein1View.xib."

Antes de continuar a través del bosque de Objective-C, tomémonos un descanso y veamos dónde estamos:

- `einSwitchAppDelegate.h` and `einSwitchAppDelegate.m` **Hecho**
- `SwitchViewController.h` and `SwitchViewController.m` **Hecho**
- `Ein1Controller.h` y `Ein1Controller.m` **Hecho**
- `Einstein1View.xib` **Hecho**
- `Ein2Controller.h` and `Ein2Controller.m` **Hecho**
- `Einstein2View.xib` *A continuación*

Guarde Einstein2View.xib

Guarde `Einstein2View.xib` como muestra la Figura 6–18, y compruebe que su carpeta de recursos contiene los dos archivos `.xib` que ha creado. Ahora, cuando echamos un vistazo a nuestro plan de acción, vemos que hemos recorrido un largo camino:

- `einSwitchAppDelegate.h` and `einSwitchAppDelegate.m` **Hecho**
- `SwitchViewController.h` and `SwitchViewController.m` **Hecho**
- `Ein1Controller.h` and `Ein1Controller.m` **Hecho**
- `Einstein1View.xib` **Hecho**
- `Ein2Controller.h` and `Ein2Controller.m` **Hecho**
- `Einstein2View.xib` **Hecho**

Por tanto, está mirando la lista y piensa que ya tiene todos los personajes hechos y puede empezar con el código. ¿Cierto?! Hmm… no tan rápido. ¿Ve lo que falta?

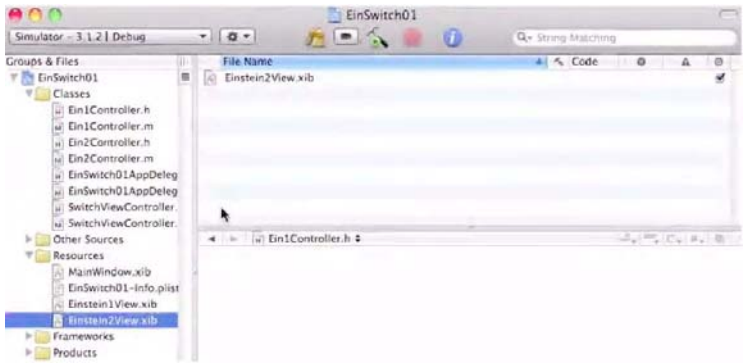


Figura 6–18. Guarde el segundo de sus dos archivos nib, Einstein2View.xib.

Ciertamente tenemos todos nuestros personajes y sus herramientas, pero nos falta algún material esencial. Tenemos que insertar tres elementos en nuestra carpeta de recursos: las fotos de las capas superiores y de fondo así como la imagen del icono.

Arrastre las imágenes dentro de Xcode

Tal y como hizo en el capítulo 5, paso 7, arrastre su primera imagen en la carpeta de Recursos. Asegúrese, como hizo antes, de activar la casilla "Copy ítems into destination's Group forder" (si es necesario) –Copiar elementos a la carpeta del grupo de destino-, así como la de "Recursively create groups for any added folders" –crear grupos recursivamente para cualquier carpeta añadida. Entonces, haga clic en Añadir o pulse intro. De manera similar, arrastre la segunda imagen a la carpeta de recursos, y coja la imagen del icono mientras está en ello.

Como algunos de ustedes puede que sepan, pueden seleccionar más de un elemento haciendo clic, arrastrando y activando casillas en todos los elementos deseados manteniendo pulsada la tecla de control (para elementos no adyacentes), o la tecla de mayúscula (para elementos adyacentes) al mismo tiempo que hace clic en el primer y último elemento (de una lista). Ver Figuras 6-19 y 6-20.

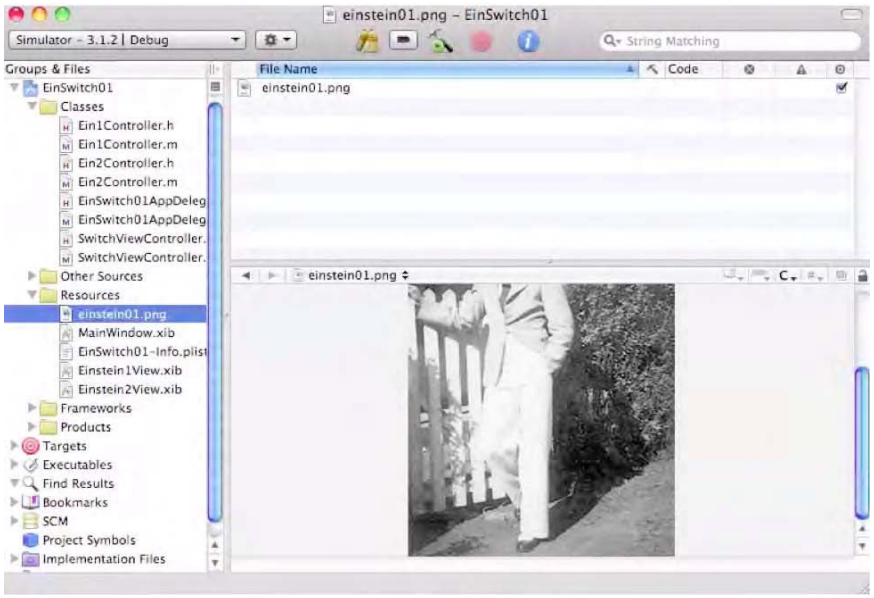


Figura 6–19. Guarde el primero de sus dos archivos de imagen, einstein01.png

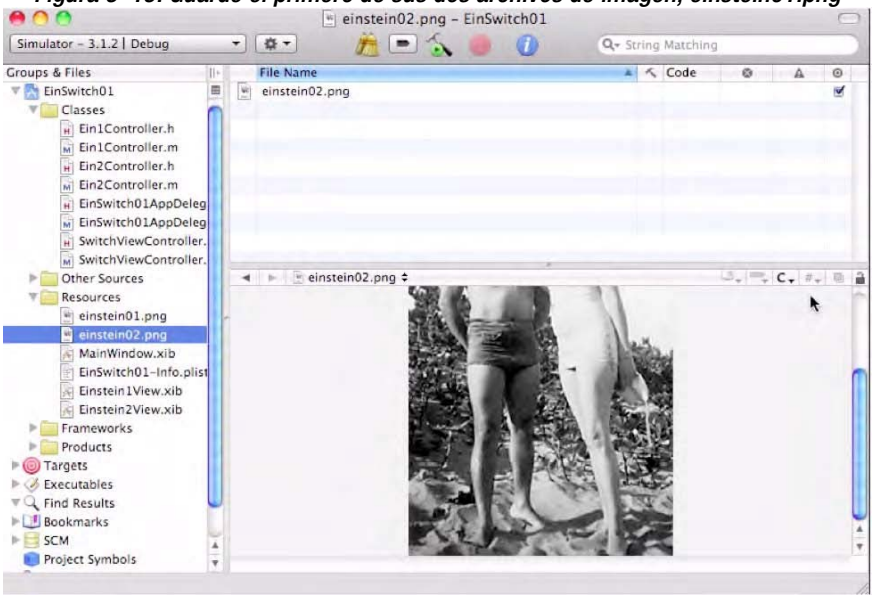


Figura 6–20. Guarde el segundo de sus dos archivos de imagen, einstein02.png.

Asigne el icono en “plist”

Tal y como hizo antes, tiene que asociar la imagen del icono con su proyecto. Abra el archivo `Info.plist` en la carpeta de Recursos y haga doble clic en la celda Value (valor) del archivo de icono. En ese espacio, teclee el nombre del archivo de icono, **icon.png**, como muestra la Figura 6-21. Ahora, guarde su trabajo introduciendo ⌘S.

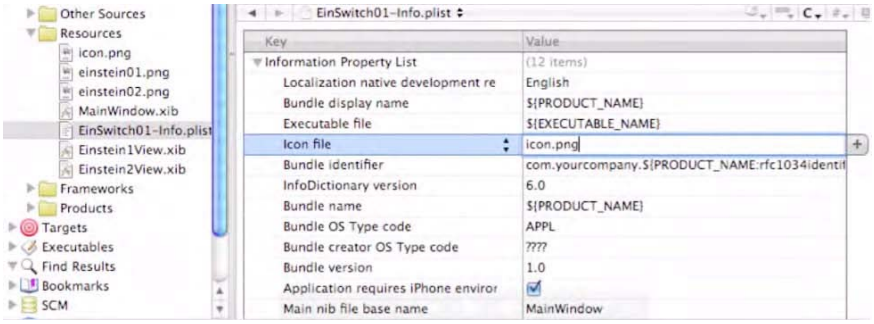


Figura 6–21. Asigne su archivo de icono en la Information Property List: la “plist.”

Codifique el AppViewDelegate

¡De acuerdo! Hemos creado todos los jugadores, hemos asociado todos los archivos y los hemos asociado el archivo de icono con el programa. Esto solo significa una cosa mis queridos infoadictos ¡estamos listos para el programar!

Volvamos a la carpeta de Clases, como se muestra en la Figura 6-22, y hagamos lo de siempre: abrir el archivo de cabecera delegado de la aplicación. Mientras lo hacemos, revisemos lo que sabemos ahora sobre cómo se crean las estructuras de archivo. Cuando el Xcode instancia nuestro proyecto, crea el rol que va a delegar y presidir los otros archivos `EinSwitch01AppDelegate` (ver Figura 6-11). En el proceso de instanciación, vimos como el sistema tanto una cabecera (`.h`) como un archivo de implementación para nuestro (`.m`) `AppDelegate` y pone el nombre del programa justo antes de la frase `AppDelegate`.

En resumen, como llamamos a nuestro programa `EinSwitch01`, Xcode automáticamente etiqueta nuestro archivo AppDelegate `EinSwitch01AppDelegate`. Es más, dentro de este archivo, el sistema generó dos archivos específicos más: `EinSwitch01AppDelegate.h` y `EinSwitch01AppDelegate.m`.

```
#import <UIKit/UIKit.h>

@interface EinSwitch01AppDelegate NSObject <UIApplicationDelegate>{
UIWindow *window;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;

@end
```

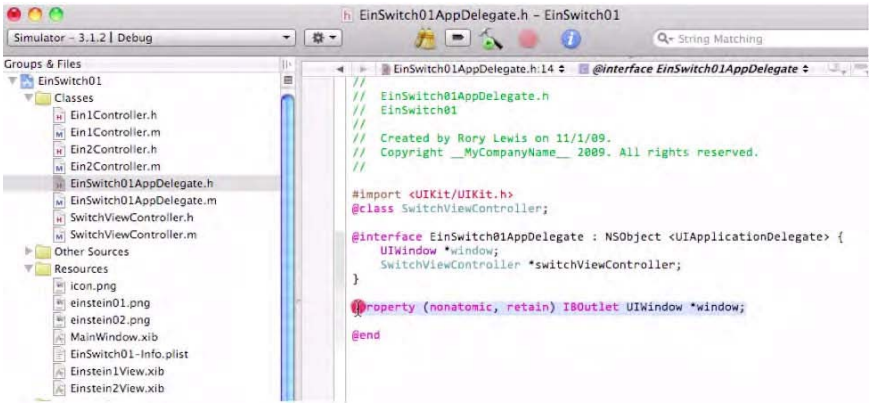


Figura 6–22. Prepárese a insertar código nuevo en el archivo EinSwitch01AppDelegate.h.

En este punto, podríamos hacer fácilmente lo que hicimos en el pasado: entrar las tres líneas de código que queremos añadir… y seguir adelante a oscuras. Pero ahora, vamos a abrir el capó de nuestro coche y examinar el motor. No voy a sobrecargarle con demasiada información, no espero que recuerde todo. Mi objetivo es retarle aquí y ver si puede darle sentido a algún detalle del código.

Volviendo a la Figura 6-10, notará que `EinSwitch01AppDelegate` comunica con y delega trabajo al `SwitchViewController`. El archivo `SwitchViewController.h` necesita tener todos los extras y las cosas que el `SwitchViewController.m` necesita tener para decirle al `SwitchViewController` qué peón mostrará su imagen. Nosotros suministramos esto añadiendo la clase `SwitchViewController` que creamos (ver Figuras 6-12 y 6-13).

Recuerde que cuando añadimos una clase, ponemos el símbolo `@` delante para obtener la atención del ordenador. Insertaremos lo que se llama una directiva de *precompilador de clase@* que anuncia nuestra intención de llamar el `SwitchViewController` en la implementación del archivo. Estas directivas de precompilador de clase@ le dice al compilador que una clase (de cualquier tipo que nos interese crear) estará accesible para nuestro uso. Algunos programadores llaman a esto una declaración adelantada porque le estamos dando al compilador una cabecera antes de que se declare la clase en el archivo de implementación. Por lo tanto, añadimos `@class SwitchViewController` justo después del código `#import` como se ve a continuación:

```
#import <UIKit/UIKit.h>

@class SwitchViewController;

@interface EinSwitch01AppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;

@end
```

Ahora sabemos que cuando codifiquemos el archivo de implementación que corresponde con este archivo de cabecera, para delegar el trabajo al `SwitchViewController`, tendrá la habilidad de hacerlo. ¡Guapo!

A continuación, cambiamos a la porción de código `IBOutlet` para la UIWindow: la ventana de interfaz de usuario y su ventana de puntero asociada. Tenemos que crear un `IBOutlet` para todos los extra del `SwitchViewController`. Por lo tanto, después de la sección `UIWindow *window`, añada `SwitchViewController *switchViewController` al código. Una vez hecho esto, podemos añadir una directiva `@property` y ocuparnos del IBOutlet:

```
#import <UIKit/UIKit.h>

@class SwitchViewController;

@interface EinSwitch01AppDelegate : NSObject <UIApplicationDelegate> {
        UIWindow *window;
        SwitchViewController *switchViewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;

@end
```

Trabajando con SwitchView

Para ponerle la línea de la directiva de propiedad, usaremos un atajo. Seleccione y copie esta línea completa

```
@property (nonatomic, retain) IBOutlet UIWindow *window;
```

y entonces péguela debajo de ella misma. Luego seleccione y copie esta línea:

```
SwitchViewController *switchViewController
```

Ver Figura 6–23.

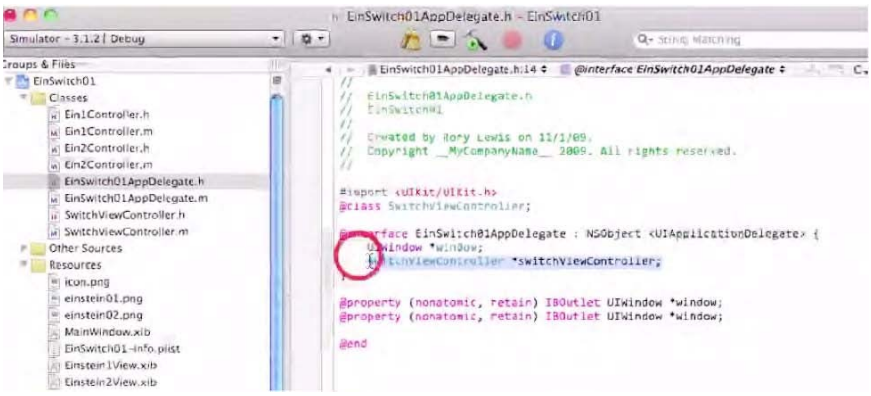


*Figura 6–23. Copie el código SwitchViewController *switchViewController.*

Como se muestra en la Figura 6-24, pegue el código

```
SwitchViewController *switchViewController;
```

introduciendo (⌘V) sobre la porción de código

```
IBOutlet UIWindow *window;
```

Seleccionado que acaba de insertar.

```
#import <UIKit/UIKit.h>

@class SwitchViewController;

@interface EinSwitch01AppDelegate : NSObject <UIApplicationDelegate> {
	UIWindow *window;
	SwitchViewController *switchViewController;

}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet SwitchViewController *switchViewController;

@end
```

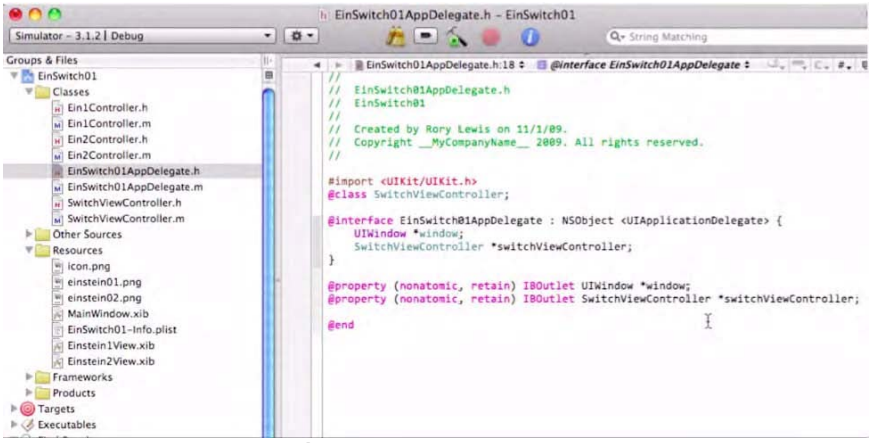


*Figura 6–24. Pegue el código SwitchViewController *switchViewController.*

Recuerde que la directiva, `@property` declara que nuestro objeto tiene una propiedad con un tipo específico, mientras que la directiva `@synthesize`, a la que estamos a punto de dirigirnos, pone en juego métodos que declaramos en la directiva `@property`. Guarde su trabajo en el archivo de cabecera y siga al archivo de implementación, `eEinSwitch01AppDelegate.m`. Ver Figura 6-25

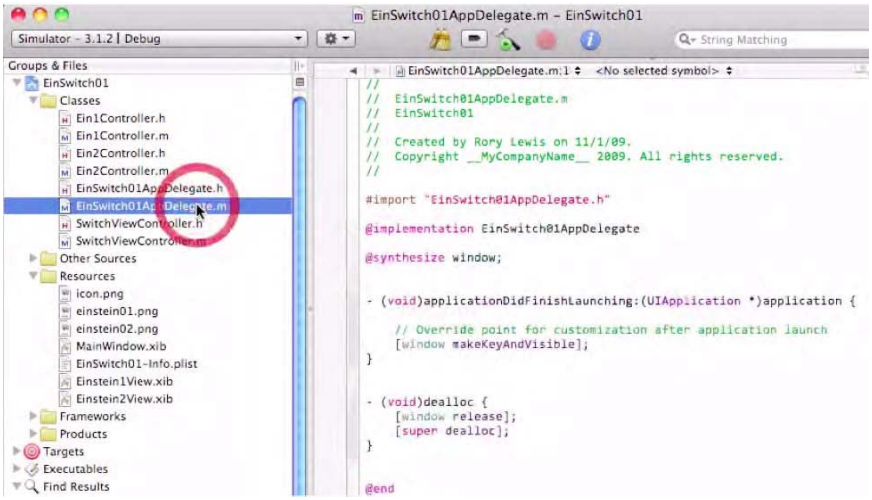


Figura 6–25. Guarde el archivo de cabecera. Ya es hora de continuar al archivo de implementación

SwitchViewController y AppDelegate

En la carpeta de Clases, desplácese hacia abajo al archivo de implementación correspondiente, `EinSwitch01AppDelegate.m`, y ábralo. Como se muestra en la Figura 6-26, verá que tiene el siguiente aspecto.

```
#import "EinSwitch01AppDelegate.h"

@implementation EinSwitch01sAppDelegate

@synthesize window;
-(void)applicationDidFinishLaunching:(UIApplication *)application {
// Override point for customization after application launch
    [windowmakeKeyAndVisible];
}
-(void)dealloc {
    [windowrelease];
    [superdealloc];
}
@end
```

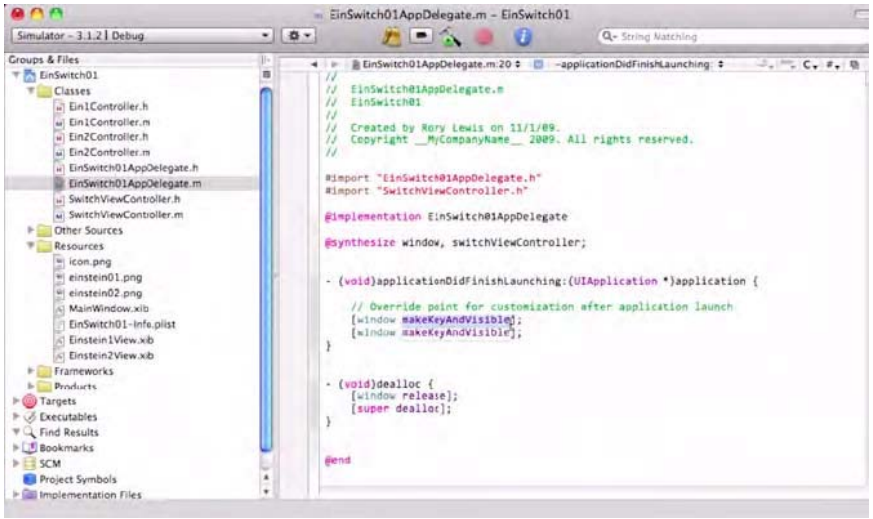


Figura 6–26. Abra el archivo de implementación.

Como puede ver, Apple ha sido lo bastante amable para importar el archivo de cabecera del AppDelegate en que hemos estado trabajando `EinSwitch01AppDelegate.h`. Por lo tanto, todos los avisos que le hicimos para que enviara este archivo de implementación estarán aquí. Hmm… pensemos en eso un momento.

¿Qué otros avisos o anuncios necesitamos aquí? Si vuelve a la Figura 6-10, puede que recuerde que`EinSwitch01AppDelegate` ordena al `SwitchViewController` que mande a sus subordinados hacer acciones específicas –como sostener fotografías de nuestro sujeto. Por tanto, tenemos que importar también todos los avisos del `SwitchViewController.h`, como se ve aquí:

```
#import "EinSwitch01AppDelegate.h"
#import "SwitchViewController.h"

@implementation EinSwitch01AppDelegate

@synthesize window;

-(void)applicationDidFinishLaunching:(UIApplication *)application {

    // Override point for customization after application launch
    [window makeKeyAndVisible];
}
-(void)dealloc {
    [window release];
    [super dealloc]; }

@end
```

Fíjese que la línea siguiente tiene el código que Apple instanció automáticamente por nosotros:

```
@implementation EinSwitch01AppDelegate
```

Esta línea ha sido algo que nos hemos saltado en el pasado, pero vamos a considerarla ahora. Una forma en la que podemos pensar sobre ello es: "En este archivo de implementación, por la presente tomamos nota de esos comandos y directivas que se anunciaron y definieron en el archivo de cabecera `EinSwitch01AppDelegate`.

NOTA: A menudo uso el concepto "darse cuenta" para tratar con el término "instanciar". Cuando el ordenador instancia automáticamente una parte de la estructura del programa, está creando un objeto (haciendo el personaje real).

¿Qué más necesitamos insertar aquí? Recuerde que la directiva `@property`, que siempre está localizada en el archivo de cabecera, declara que nuestro objeto tiene una propiedad con un tipo específico. En contrapartida, la directiva `@synthesize`, localizada en el archivo de implementación, notifica estas directivas al compilador. Recuerde también, que teníamos una directiva `@property`.

```
@property (nonatomic, retain) IBOutlet UIWindow *window
```

para la *ventana IBOutlet*. Entonces añadimos otra directiva `@property` para el SwitchViewController IBOutlet:

```
@property (nonatomic, retain) IBOutlet SwitchViewController
*switchViewController
```

La directiva `@synthesize` para la ventana ya la ha hecho Apple por nosotros, pero tenemos que añadir el elemento que falta, la directiva `@synthesize` para el switchViewController. Veámoslo ilustrado en el código siguiente:

```
#import "EinSwitch01AppDelegate.h"
#import "SwitchViewController.h"

@implementation EinSwitch01AppDelegate

@synthesize window, switchViewController;

-(void)applicationDidFinishLaunching:(UIApplication *)application {

   // Override point for customization after application launch
   [window makeKeyAndVisible];
}
-(void)dealloc {
   [window release];
   [super dealloc];
}

@end
```

¡Guapo, eh? Espero que empiece a ver la lógica que hay detrás de el añadido de estos enunciados. Ahora mire la línea siguiente:

```
(void)applicationDidFinishLaunching:(UIApplication *)application
```

y haga clic con el botón derecho en la parte `applicationDidFinishLaunching`. Desplácese hacia abajo y haga clic en la opción `Find Selected Text Documentation`. Esto le llevará a un lugar al que todavía no nos hemos aventurado en este libro, un mundo fantástico maravilloso (agujero de conejo… por el que se caía Alicia en "Alicia en el País de las Maravillas") que le lleva a la documentación de Xcode y Objective-C de Apple. Cualquier porción de código que no sepa en los meses y años venideros –sin problema, simplemente compruébelo en la documentación de Apple.

Al hacer clic vemos que dice –en parte- lo siguiente:

```
applicationDidFinishLaunching:
Tells the delegate when the application has finished launching.
-(void)applicationDidFinishLaunching:(UIApplication *)application
Parameters
application
The delegating application object.
Discussion
This method is the ideal place for the delegate to perform various initialization
and configuration tasks, especially restoring the application to the previous state
and setting up the initial windows and views of the application…
```

Lo cual es decir, básicamente: "Hey, lo escribí para usted sin que usted tuviera que tratar con ello. Si realmente quiere conocerlo, es donde sus delegados realizan varias tareas de configuración e inicialización."

Justo debajo, dentro de los paréntesis, Apple ha manejado la inicialización y configuración de una tarea esencial: `[window makeKeyAndVisible]`. Este trozo de código hace que `UIWindow` se haga visible y lo convierte en "el primero que responde" a toques del usuario cuando la aplicación está corriendo en un iPhone/iPad.

Sin embargo necesitamos algo más aquí. ¿Sabe qué? También necesitamos asegurarnos de que el personaje al que Delegado del Mandamás está mandando, el `SwitchViewController`, es visible para el usuario. Hacemos esto añadiendo una subsista. El código que lleva a cabo esto es `[window addSubview:switchViewController.view]`..

Como su nombre sugiere, este procedimiento añada una vista a las subsistas del `switchViewController`. Lo que está sucediendo aquí es que, cuando el `switchViewController` les dice a sus subordinados que muestren la fotografía de mi abuelo, queremos añadir una vista a la ventana como subvista. Por tanto, en realidad estamos preguntando al `switchViewController` por la vista que controla, que, cuando la maneja, muestra al usuario la ventana y le capacita para que acepte toques y otras entradas. Inserte esa línea como se muestra aquí:

```
#import "EinSwitchO1AppDelegate.h"
#import "SwitchViewController.h"

@implementation EinSwitchO1AppDelegate

@synthesize window, switchViewController;

-(void)applicationDidFinishLaunching:(UIApplication *)application {

    // Override point for customization after application launch
    [window addSubview:switchViewController.view];
    [window makeKeyAndVisible];
}
-(void)dealloc {
    [window release];
    [super dealloc];
}
@end
```

Nuestro último paso aquí, es desasignar la memoria de todos estos actores en nuestro juego. Apple sabe que tiene que desasignar la memoria de varias ventanas y, por seguridad, también realiza una “super-desasignación”. Simplemente acéptelo por ahora… Ya tenemos la cabeza bastante sobrecargada aquí.

Tenemos que desasignar la memoria que estamos usando para… ¿qué? Hmm. Adivine. ¿El elemento al que nos hemos estado refiriendo constantemente en esta sección? ¿El `switchViewController`?

¡Sí. Hagámoslo!

```
#import "EinSwitchO1AppDelegate.h"
#import "SwitchViewController.h"

@implementation EinSwitchO1AppDelegate

@synthesize window, switchViewController;

-(void)applicationDidFinishLaunching:(UIApplication *)application {

    // Override point for customization after application launch
    [window addSubview:switchViewController.view];
    [window makeKeyAndVisible];
}
```

```
-(void)dealloc {
    [window release];
    [switchViewController release];
        [super dealloc];

}

@end
```

Ahora introduzca ⌘S para guardar su trabajo. ¡El santo archivo de implementación, Batman! ¡Ha terminado!

Se ha ocupado del en términos tanto de cabecera como de sus archivos de implementación. Si nos referimos otra vez a la Figura 6-10, vemos que hemos creado el código para `einSwitchAppDelegate` -¡representado por la reina del ajedrez!

¡Fiuuuu! -¡es hora de descansar!

Archivo de Cabecera SwitchViewController

Continuando con la metáfora del ajedrez, puede decirse que ha acabado de trabajar con el archivo AppDelegate -la reina. Ahora es hora de centrarse en sus inmediatos subordinados, el `SwitchViewController`. Como sabe, esta figura, representada por el caballo, tiene el papel de mandar a cualquiera de sus asistentes que contenga su imagen. En los pasos siguientes, trataremos con el archivo de cabecera (`.h`) y el archivo de implementación (`.m`) del `SwitchViewController`.

Ahora tenemos que codificar lo que dice al `Ein1Controller` que muestre su fotografía a nuestro sujeto; entonces podemos decir al `Ein2Controller` que muestre su fotografía. Por tanto, desplácese hacia abajo en la carpeta de Clases y abra el archivo de cabecera: `SwitchViewController.h`. Antes tenemos que asegurarnos de que el precompilador al menos sabe quien son los `Ein#Controllers`. Ver Figura 6-27. Recuerde que el símbolo `@` capta la atención del ordenador para el establecimiento de una relación específica, y usamos la directiva de precompilador `@class` para hacer esto.

```
#import <UIKit/UIKit.h>
@class Ein1Controller;
@class Ein2Controller;

@interface SwitchViewController : UIViewController {

}

@end
```

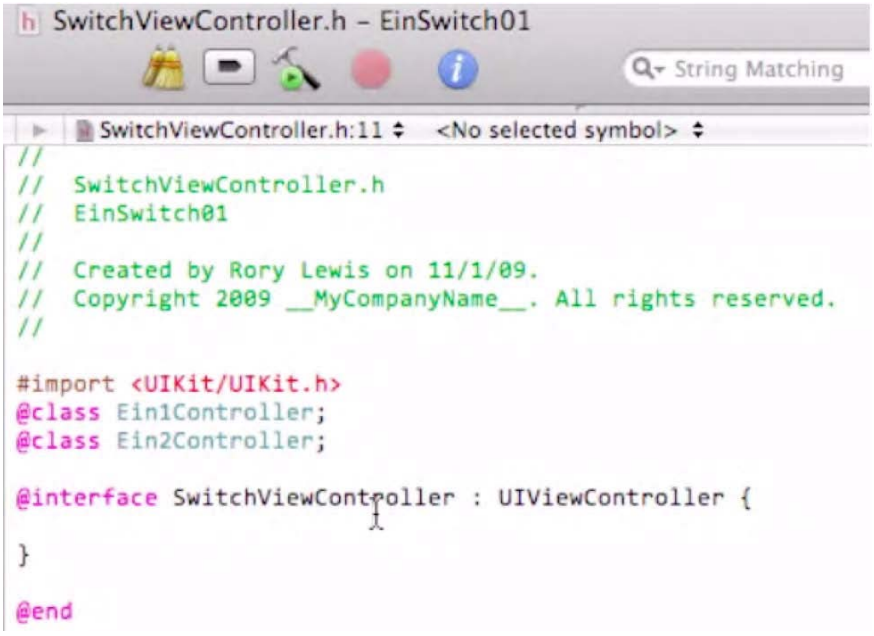


Figura 6–27. Copie la directiva de precompilador @class para el Ein1Controller para crear una para.

A continuación tenemos que asegurarnos de que el archivo de implementación sabrá quienes son los `Ein#Controllers` … que existen siquiera. En términos técnicos, decimos que necesitamos declarar las variables de la instancia que necesitamos usar a través de la clase. Esto se hace dentro de los paréntesis, inmediatamente a continuación de la directiva `@interface SwitchViewController: UIViewController`.

Hacemos esto usando un puntero –si, el asterisco, que, hasta ahora, le he dicho que ignore. Bien, es hora de considerarlo. Tenemos que decirle al archivo de implementación que reserve un sitio en la memoria para `Ein1Controller` y `Ein2Controller`, y lo hacemos usando un puntero… para cada uno de los roles subordinados:

```
#import <UIKit/UIKit.h>
@class Ein1Controller;
@class Ein2Controller;

@interface SwitchViewController : UIViewController {
          Ein1Controller *ein1Controller;
          Ein2Controller *ein2Controller;
          }

@end
```

A continuación, tenemos que usar la directiva `@property` para definir estas variables como propiedades, y hacemos esto con el mismo código que hemos usado muchas veces antes:

```
@property (retain, nonatomic) Ein#Controller *ein#Controller
```

asegurándonos de hacerlo individualmente para cada `Ein#Controller` que presenta el usuario con una fotografía de mi abuelo. Lo hacemos como sigue:

```
#import <UIKit/UIKit.h>
@class Ein1Controller;
@class Ein2Controller;

@interface SwitchViewController : UIViewController {
        Ein1Controller *ein1Controller;
        Ein2Controller *ein2Controller;
}

@property (retain, nonatomic) Ein1Controller *ein1Controller;
@property (retain, nonatomic) Ein2Controller *ein2Controller;

@end
```

El próximo elemento al que nos dirigiremos es que necesitamos una acción de algún tipo para cambiar vistas. A esto lo hemos llamado antes una instancia, y todavía lo llamaremos así. Técnicamente decimos: “Necesitamos un método de instancia (y usamos el signo “menos”) para anunciar al archivo de implementación que vamos a incorporar una `IBAction`.” En otras palabras, “avisará” al archivo de implementación que un método de nuestro código necesita ser activado, o llamado a la acción y que estos comandos serán implementados en Interface Builder.

Necesitamos dar un nombre a esta nueva acción que va a cambiar vistas, por tanto llamémosla… hmm… `switchViews`! ¡Si! Así que introduzcamos este código:

```
-(IBAction)switchViews:
```

Este segmento de código, a cambio, necesita apuntar a una forma o rol específico, y usamos (`id`) para este propósito. Finalmente necesitamos añadir el componente “emisor” que activará el evento.

Así que nuestra última inserción de código es

```
 (IBAction)switchViews:(id)sender
```

Por cierto, recuerde que generalmente ponemos un punto y como a continuación de estas líneas de código, para alertar al ordenador de que hemos acabado con esa línea.

```
#import <UIKit/UIKit.h>
@class Ein1Controller;
@class Ein2Controller;

@interface SwitchViewController : UIViewController {
        Ein2Controller *ein2Controller;
        Ein1Controller *ein1Controller;
}
@property (retain, nonatomic) Ein2Controller *ein2Controller;
@property (retain, nonatomic) Ein1Controller *ein1Controller;

-(IBAction)switchViews:(id)sender;

@end
```

Como se muestra en la Figura 6-28, es hora de entrar ⌘S para guardar nuestro trabajo. Hemos terminado con ese archivo… ¡ Scooby-Dooby-Doo! Ahora continuemos con el archivo de implementación. ¡Bien hecho!

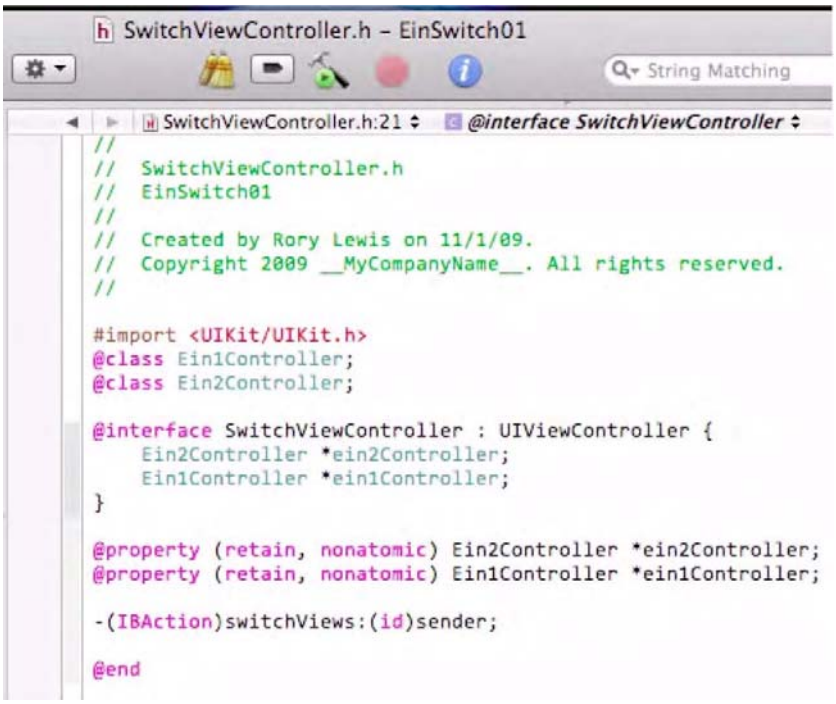


Figura 6–28. Una vez completado SwitchViewController.h, guárdelo, y vaya Lazy Load!

Listo para el archivo de implementación de Lazy Load

En la carpeta de clases, ir a el archivo de implementación del `SwitchViewController`, `SwitchViewController.m`, y haga clic para abrirlo. Cuando lo abra, verá todo el código básico que hemos visto antes, que Apple automáticamente y convenientemente instancia para nosotros. Como se ve en la Figura 6-29, los detalles de este código son invisibles para el compilador porque cada grupo de clases está situado dentro de un comentario. Creo que ahora es buen momento para señalar la naturaleza y función de los comentarios –en el contexto de la programación de aplicaciones.

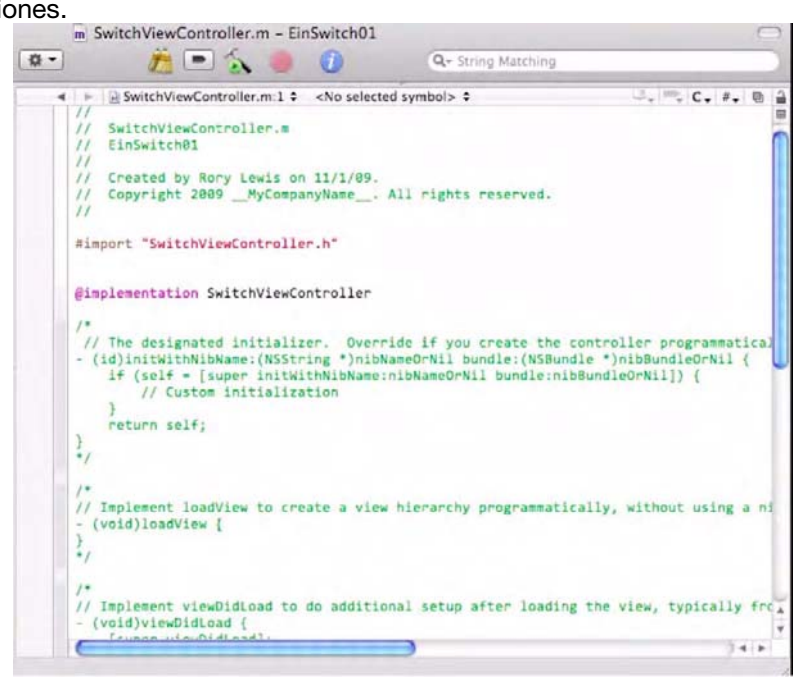


Figura 6–29. El archivo de implementación del SwitchViewController está cargado de código asociado con comentarios

NOTA: Si ya sabe todo sobre los comentarios y lazy loads, sáltese esta sección y vaya al Paso 16. Si no está seguro, quédese con nosotros y siga leyendo

Nota sobre Comentarios y Lazy Loads

Sabemos que el Xcode está basado y usa el lenguaje de programación Objective-C, y que las aplicaciones se ejecutan en virtud del código compilado en ceros y unos que los microprocesadores entienden. En Objective-C, como en otros lenguajes – particularmente el lenguaje C en el que está basado, nuestro preprocesador soporta dos estilos de comentarios. Estos comentarios, en esencia, hacen las cosas invisibles a las tripas de la máquina.

Ya hemos examinado y discutido la señal de la doble barra: //, después de la cual se pueden insertar comentarios- y que prohíbe al compilador ver estos comentarios. Estos se llaman comentarios de estilo BCPL. También está la barra-asterisco: /*, y el asterisco-barra: */, entre los cuales se pueden poner comentarios. Estos se llaman comentarios estilo C. Por ejemplo, podríamos ver algo como esto:

```
/* This is material that will be “invisible“ to the compiler. */
```

Apple sabe que, la mayor parte del tiempo, usaremos al menos una de las clases, por tanto, inserta comentarios para nuestra conveniencia como los siguientes:

```
#import "SwitchViewController.h"

@implementation SwitchViewController

/*
// The designated initializer. Override if you create the controller↩
programmatically and want to perform customization that is not appropriate↩
 for viewDidLoad.
-(id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
   if (self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil]) {
       // Custom initialization } return self;
}
*/

/*
// Implement viewDidLoad to do additional setup after loading the view, typically↩
from a nib.
-(void)viewDidLoad {
    [super viewDidLoad];

}

*/

/*
// Override to allow orientations other than the default portrait orientation.
-(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)↩
interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);

}

*/

-(void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
   [super didReceiveMemoryWarning];
```

```
    // Release any cached data, images, etc that aren't in use.
}
-(void)viewDidUnload {
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
} -(void)dealloc {
   [super dealloc];
}
@end
```

Ir más allá en esta breve digresión sobre los comentarios no nos sería útil en este momento. Sin embargo, quiero que eche un vistazo rápido, para aprecie dónde y cómo se esconden algunos detalles. Quiero que se de cuenta de que hay muchos desarrolladores de iPhone/iPad de éxito que no entienden todo ese bla-bla-bla del ejemplo de arriba. Sin embargo, un buen desarrollador sabe cómo utilizar estos estilos de comentarios cuando surge la necesidad –e incluso como pedir, tomar prestado y robar el trabajo de otros desarrolladores junto con estas líneas.

Al principio de este capítulo, mencioné que podríamos estar tirando –en el archivo de implementación- uno de los trozos de lenguaje de programación más ampliamente usados y genéricos que carga vistas en el iPhone o iPad. Este método se llama Lazy Load. Este proceso se llama así por varias razones. La primera es que somos algo vagos al usarlo. Cierto.

En segundo lugar, es vago en términos de manejo de memoria. Esto está bien para programas pequeños, pero cuando cambiamos a la creación de aplicaciones más grandes, necesitamos algo más sofisticado. Probablemente usaremos Shark u otras herramientas para arreglar “pérdidas de memoria” que las lazy loads causan a menudo. ¡Pero no se preocupe! Aún puede crear aplicaciones excelentes (y lucrativas) que cuentan con un lazy load.

Copia de Contenidos de SwitchViewController.txt

Anteriormente en este capítulo, usted descargó el archivo `006_Chapter_6hEinSwitch01.zip`. Dentro de este archivo zip, encontró y copio el archivo `SwitchViewController.txt` a su escritorio. Es hora de abrir este archivo y seleccionar todos los contenidos introduciendo ⌘A. Una vez seleccionado todo el archivo, copie introduciendo ⌘C. Vea Figura 6-30.

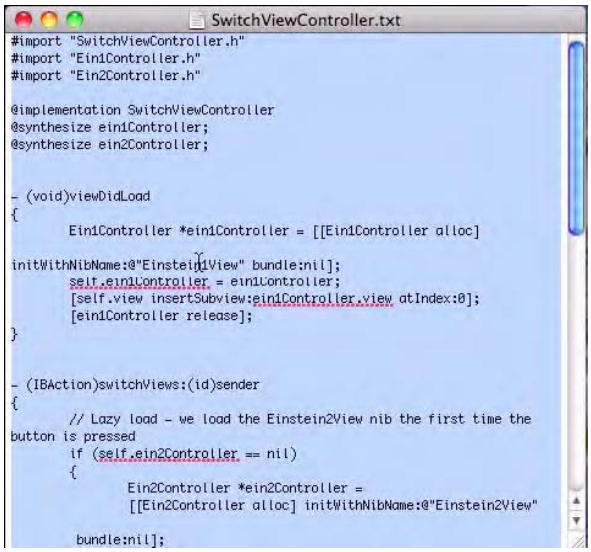


Figura 6–30. Haga clic en el archivo SwitchViewController.txt de su escritorio, haga clic dentro de la ventana, seleccione todo el contenido pulsando ⌘A, y copie el texto seleccionado pulsando ⌘C.

Con su Lazy Load copiado en el portapapeles, abra el archivo de implementación, `SwitchViewController.m`, introduzca ⌘A seleccionando todo su contenido, y entonces, sin hacer nada más, pegue inmediatamente el texto copiado entrando ⌘V, como se muestra en la Figura 6-31. Entonces, entre ⌘S para guardar su "trabajo".

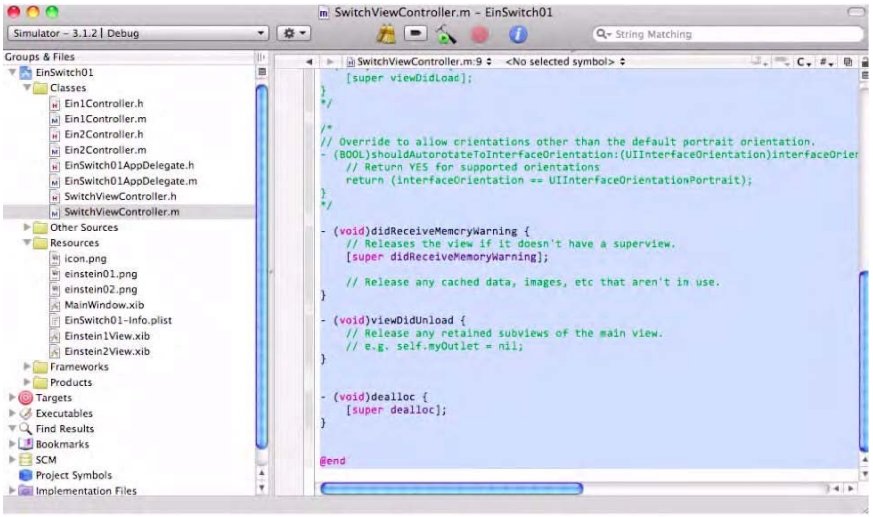


Figura 6–31. Abra el archivo SwitchViewController.m file y pegue su lazy load introduciendo ⌘A e inmediatamente, ⌘V.

¿Sabe qué? ¡Ha acabado con el código para este personaje concreto del juego! Ahora puede ir directamente al paso siguiente, si quiere, y empezar a conectar todo en los archivos nib. Eso sería perfectamente aceptable.

NOTA: Lo que sigue es una breve revisión del lenguaje de código Lazy Load. Si no le interesa en este momento, puede ir directamente a la sección Seleccionar el Propietario del Archivo.

Nota sobre el Lenguaje de Código de Apple del Archivo de implementación.

Usted no tiene que aprendérselo, o entender cada símbolo, pero quiero que salga con, al menos, una comprensión vaga de cómo funciona todo esto. La primera porción de este archivo es código que programé para usted, que incluyo en el Lazy Load. Simplemente importé y sintenticé los dos archivos de cabecera `Ein#Controller.h` como sigue:

```
#import "SwitchViewController.h"
#import "Ein1Controller.h"
#import "Ein2Controller.h"

@implementation SwitchViewController
@synthesize ein1Controller;
@synthesize ein2Controller;

// We've done the above about 7 times already, so I figured you'd be cool↩
with me inserting the import
// and synthesis files for you. The next piece of code in here is the↩
 (void)viewDidLoad.
```

```
-(void)viewDidLoad
{

       Ein1Controller *ein1Controller = [[Ein1Controller alloc]

       initWithNibName:@"Einstein1View" bundle:nil];
       self.ein1Controller = ein1Controller;
       [self.view insertSubview:ein1Controller.view atIndex:0];
       [ein1Controller release];
}
```

El método - (void)viewDidLoad que nos ha dado Apple, es llamado después de que la vista SwitchViewController se cargue en la memoria. Puede que recuerde que este método, o proceso, es llamado (o activado) independientemente de si las vistas se guardaron en los archivos .xib o se crearon programáticamente a través del método loadView. Ahora mismo, déle al código un simple saludo de reconocimiento. Usted se encontrará cortando y pegando esto según vaya avanzando en el trabajo, y quiero ayudarle a asegurarse de que los archivos .xib se cargan apropiadamente. A continuación, el próximo método que vemos es la esencia del Lazy Load:

```
-(IBAction)switchViews:(id)sender
{
        // Lazy load - we load the Einstein2View nib the first time the button↩
is pressed
       if (self.ein2Controller == nil)
       {
              Ein2Controller *ein2Controller =
              [[Ein2Controller alloc] initWithNibName:@"Einstein2View"
bundle:nil];
              self.ein2Controller = ein2Controller;
              [ein2Controller release];
       }
       if (self.ein1Controller.view.superview == nil)
              //This is with no animation
       {
              [ein2Controller.view removeFromSuperview];
              [self.view insertSubview:ein1Controller.view atIndex:0];
       }
       else
       {
              [ein1Controller.view removeFromSuperview];
              [self.view insertSubview:ein2Controller.view atIndex:0];
       }
}
```

¿Puede deducir qué está ocurriendo con esta porción de código? Puede ver que primero mandamos al ordenador que cargue la segunda fotografía y entonces, dependiendo de si la otra se ha cargado o no, las intercambiamos –según el usuario pulsa el botón para cambiar vistas. Apple nos suministra el próximo conjunto de métodos. Estos inicializan los archivos nib, capacitan a la pantalla para usar ciertas herramientas (como “autorotar”, si el usuario rota el iPhone), reciben avisos de memoria (si el usuario empieza a quedarse sin memoria), y desasignar memoria una vez que el usuario ha terminado con la aplicación.

```
// Initialization code
-(id)initWithNibName:(NSString *)nibNameOrNil
                        bundle:(NSBundle *)nibBundleOrNil {
      if (self = [super initWithNibName:nibNameOrNil
                                                    bundle:nibBundleOrNil]) {
      }
      return self;
}
-(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)↩
interfaceOrientation {
       // Return YES for supported orientations
       return (interfaceOrientation == UIInterfaceOrientationPortrait);
 }
-(void)didReceiveMemoryWarning {
      [super didReceiveMemoryWarning]; // Releases the view if it doesn't↩
 have a superview
      // Release anything that's not essential, such as cached data
      if (self.ein1Controller.view.superview == nil)
            self.ein1Controller = nil;
      else
            self.ein1Controller = nil;
}
-(void)dealloc {
      [ein2Controller release];
      [ein1Controller release];
      [super dealloc];
}
@end
```

Esta breve revisión es suficiente dado su nivel de comprensión. Mientras tanto, vale la pena mencionar que, en una conferencia reciente, oí a un equipo de marido y mujer, que están ganando unos $ 50.000 al mes, admitir que ellos apenas entienden el código de implementación del lenguaje de programación de Apple. La clave, como puede deducir, es que son los suficientemente listos para saber dónde pegar los nombres de sus vistas. Por ejemplo, ¿Ve donde inserté `Einstein1View` y lo demás en `(void)viewDidLoad` method?¡Muy bien!

Vayamos al Interface Builder y conectémoslo todo. ¡Casi hemos terminado!

Trabajando con los archivos.xib

Como se indicaba en la Figura 6-32, vaya a la carpeta de Recursos y abra el archivo `MainWindow.xib`. Como siempre, esto abrirá el Interface Builder.

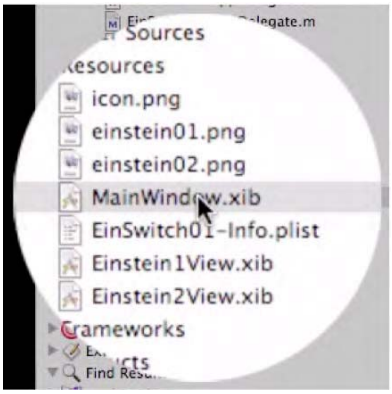


Figura 6–32. Vaya a la carpeta de Recursos y abra el MainWindow.xib.

Una vez que el Interface Builder está abierto, vaya a la ventana de la Librería introduciendo ⌘L. Navegue a la carpeta de Controladores en Cocoa Touch y arrastre un Controlador de Vista a la ventana de Documento, como se muestra en la Figura 6-33.

NOTA: La ventana de Documento es la ventana que contiene sus documentos. NO es la ventana abierta en su escritorio.

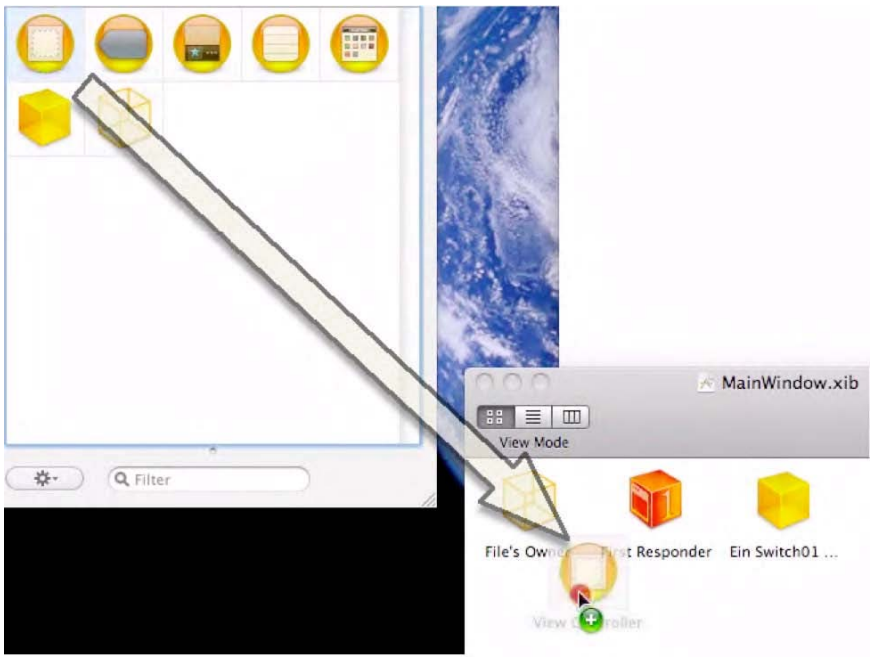


Figura 6–33. Arrastre un Controlador de vista (icono superior izquierdo) a su ventana de Documento

.

Seleccione el propietario del archivo

Con el Controlador de Vista todavía activo (si no es así, haga una clic sobre él), eche un vistazo a la ventana Inspector y tome nota de que, por defecto, está asociado a un `UIViewController` genérico. Pero nosotros no queremos esto. Queremos que nuestro Controlador de Vista interactúe con el código que programamos para nuestra propia creación –un personaje que hemos llamado `SwitchViewController`. Por tanto, haga clic en el menú desplegable y seleccione `SwitchViewController`, que estará asociado con este Controlador de Vista en la Ventana de Documento. Vea Figura 6-34.

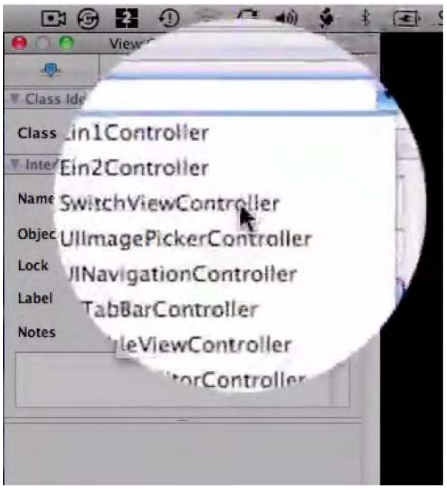


Figura 6–34. Seleccione SwitchViewController para asociarlo al Controlador de Vista en la ventana de Documento.

Arrastre una Vista a la Pantalla

Desplácese hacia abajo a la carpeta Ventanas, Vistas y Barras de su Librería y arrastre una vista a su pantalla como se muestra en la Figura 6-35. Hacemos esto porque necesitamos vistas para el Cambio de Vistas que estamos generando. Podríamos hacer nuestros propios botones para cambiar entre las diferentes vistas, pero esto ya lo hemos hecho. Usted ya sabe cómo hacerlo. Por tanto, pensé que estaría bien usar uno de los botones pre-codificados que hay en la carpeta de Ventanas, Vistas y Barras. Continúe y arrastre una barra de herramientas a su pantalla y sitúela al fondo del espacio de trabajo. Vea la Figura 6-36. Ahora queremos dar un nombre al botón de la barra de herramientas. Haga clic en él y llámelo Swich Views (cambio de vista), como se demuestra en la Figura 6-37.

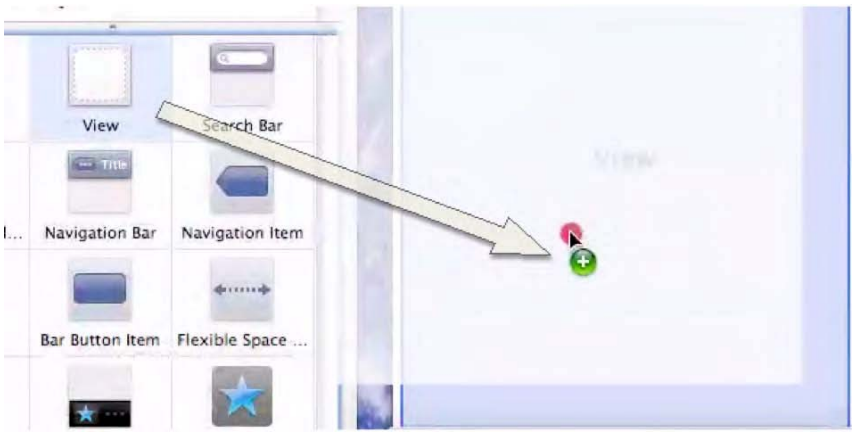


Figura 6–35. Arrastre una vista a su pantalla.

Figura 6–36. Arrastre una barra de herramientas a su pantalla y sitúela en la parte de debajo de su espacio de trabajo

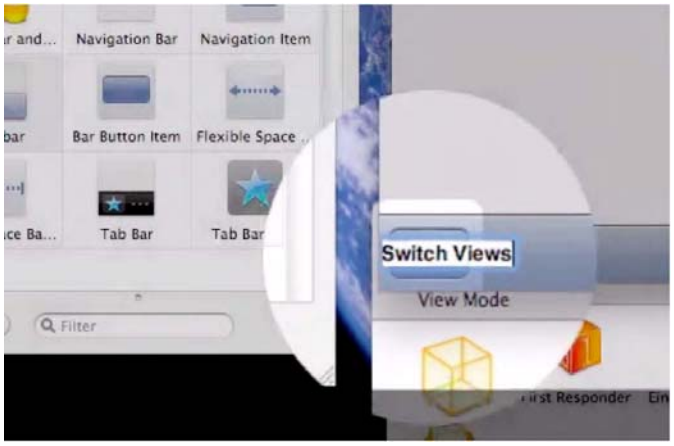


Figura 6–37. Llame el "Switch Views."

Tenemos que conectar el mecanismo que hay dentro del botón Switch Views al código que hemos asociado ahora con el `SwitchViewController`. Presione control y arrastre desde el botón Switch Views hacia abajo al icono de Controlador de Cambio de Vista. En este punto, su pantalla debería tener un aspecto similar a la Figura 6-38.

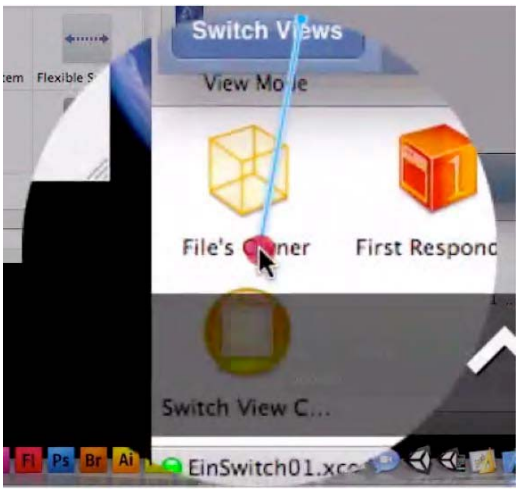


Figura 6–38. Presione control y arrastre desde el botón Switch Views hacia abajo al icono de Controlador de Cambio de Vista.

Ahora queremos conectar el botón Switch Views con la opción apropiada. Mientras arrastra sobre el icono del Controlador de Cambio de Vista, el menú desplegable negro de Acciones Enviadas se abrirá, y podrá seleccionar la opción switchViews, como se muestra en la Figura 6-39.

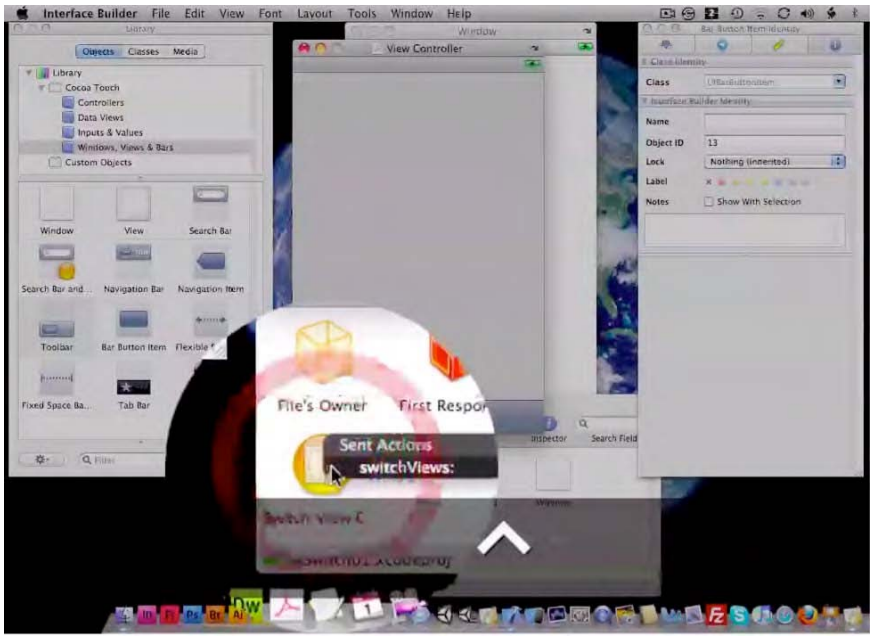


Figura 6–39. Conecte el botón Switch Views a la opción switchViews.

¿Qué está ocurriendo aquí? ¿Recuerda, en el Paso 14, cuando creamos una acción que anunciara que tenemos un cambio de vista? Tenía este aspecto:

```
-(IBAction)switchViews:(id)sender
```

El código que estamos viendo aquí, en el menú desplegable, es el resultado de ese código. Lo he puesto en negrita abajo para refrescarle la memoria.

```
#import <UIKit/UIKit.h>
@class Ein1Controller;
@class Ein2Controller;

@interface SwitchViewController : UIViewController {
        Ein2Controller *ein2Controller;
        Ein1Controller *ein1Controller;
}

@property (retain, nonatomic) Ein2Controller *ein2Controller;
@property (retain, nonatomic) Ein1Controller *ein1Controller;
```

-(IBAction)switchViews:(id)sender;

Ahora hemos conectado el `SwitchViewController` al Controlador de Vista, y por eso vemos el archivo en el menú desplegable. Conectamos el código que hemos introducido previamente al botón de modo que, cuando el usuario pulse el botón Switch Views en la barra de herramientas, invoque y ejecute este código.

Ahora tenemos que conectar el `Ein1Controller` al `SwitchViewController`. Por tanto …empiece por arrastrar presionando control desde el `Ein1Controller` al Controlador de Vista que aloja el `SwitchViewController`.

NOTA: Empezamos con una de las imágenes presentes en la pantalla, y, como decidimos en if statements en el Lazy Load, hacemos al Ein1Controller el subordinado activo a cargo de este despliegue inicial. Ver Figura 6-40.

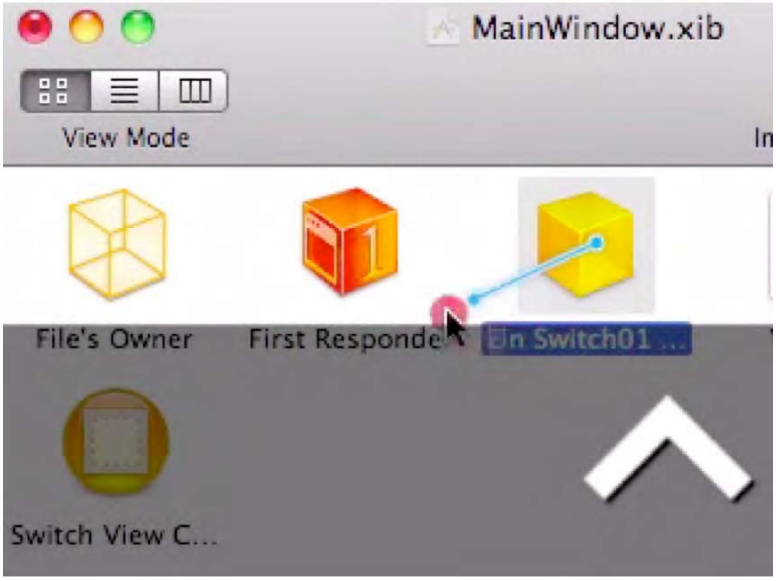


Figura 6–40. Arrastre presionando Control desde el Ein1Controller al icono del Controlador de Cambio de Vista.

De nuevo, mientras arrastra el cursor sobre el Controlador de Vista, verá un menú desplegable negro que aparece con el `SwitchViewController` dentro. Conéctele la línea de pescar y suéltelo, como se muestra en la Figura 6-41. Ahora introduzca ⌘S para guardar su trabajo.

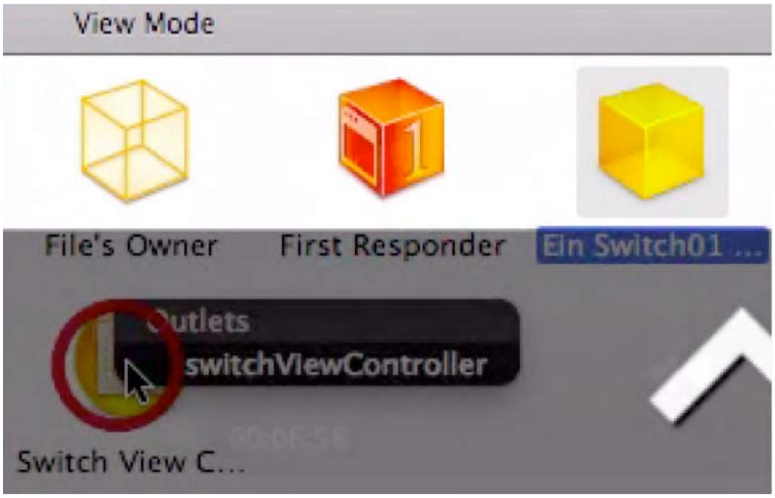


Figura 6–41. En el menú desplegable, seleccione la opción SwitchViewController.

Comience a trabajar en los archivos Einstein#View.xib

¡Casi hemos terminado!
En referencia a la red de personajes descritos en la Figura 6-10, tenemos todo en funcionamiento excepto los dos archivos nib, `Einstein1View.xib` y `Einstein2View.xib`. Nos hemos referido a esos archivos nib como las herramientas que los Ein#Controllers usan para contener y exponer sus respectivas fotografías de mi abuelo.

Vamos a abrir `Einstein1View.xib` como se muestra en la Figura 6-42. Ambos nibs hacen una cosa, y solo una cosa: contener una imagen. Por tanto, tiene sentido que lo primero que hagamos aquí sea arrastrar dentro una Vista de Imagen desde la carpeta de Vistas de Datos (Data Views) de nuestra Librería, como se muestra en la Figura 6-43. Tenemos que asociar un archivo de una imagen específica con la Vista de Imagen que acabamos de arrastrar a la pantalla. Como se ve en la Figura 6-44, vaya al inspector de propiedades y, después de hacer clic en el menú desplegable, seleccione `einstein01.png`, el archivo de imagen que se asociará con la Vista de imagen de el primero de los dos Ein#Controllers.

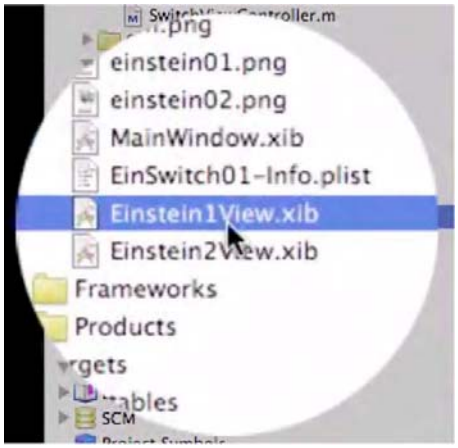


Figura 6–42. Empiece a trabajar en los archivos nib seleccionando el archivo Einstein1View.xib.

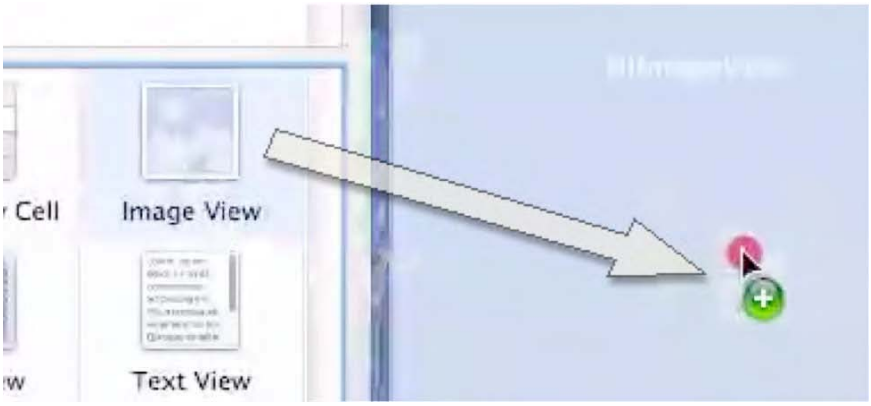


Figura 6–43. Por supuesto, las imágenes necesitan Vistas de Imagen.

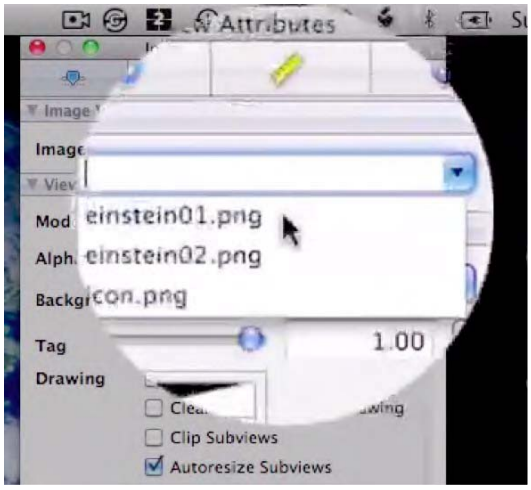


Figura 6–44. Asocie la primera imagen, einstein01.png, con la Vista de Imagen.

Ahora tenemos que dirigirnos al botón de la barra de herramientas y asegurarnos de asignarle la acción correcta. Primero, haga clic en el icono del Propietario del Archivo como se muestra en la Figura 6-45. Tenga cuidado aquí. Por alguna razón, los estudiantes olvidan que, antes de cambiar la etiqueta del icono del Propietario del Archivo, tenemos que hacer clic en él.

Figure 6–45. Click the File's Owner icon.

Una vez seleccionado el icono, vaya a la Casilla de Inspector. Una vez tenga la etiqueta, elija la opción Etiqueta en el menú desplegable negro, como s muestra en la Figura 6-46.

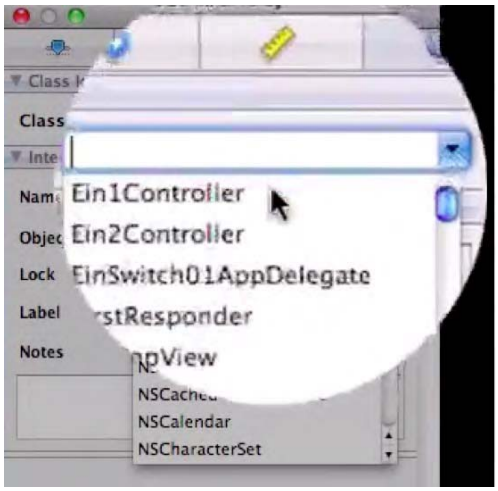


Figura 6–46. Conecte el Ein1Controller al Propietario del Archivo.

¿Recuerda cómo metíamos el código que dirigía los `Ein#Controllers` a usar los archivos nib para mostrar sus imágenes? Bien, tenemos que dejar claro que el tipo que controla aquí todo, el Propietario del Archivo, es en realidad el `Ein1Controller`. Haga clic en el icono del Propietario del Archivo y asócielo, a través del menú desplegable en la caja de propiedades, con el `Ein1Controller`, como muestra la Figura 6-46.

Ahora tenemos que arrastrar, con la tecla de control presionada desde el icono del Propietario del Archivo al icono de Vista, Cuando arrastra sobre el icono de vista, aparece un menú desplegable con Vista como una de las opciones. Apunte a esta opción y suelte. Vea Figuras 6-47 y 6-48. Una vez hecho esto, hemos completado el trabajo con el archivo `Einstein1View.xib`. Introduzca ⌘S para guardar el trabajo, y ⌘Q para salir del Interface Builder y volver a Xcode.

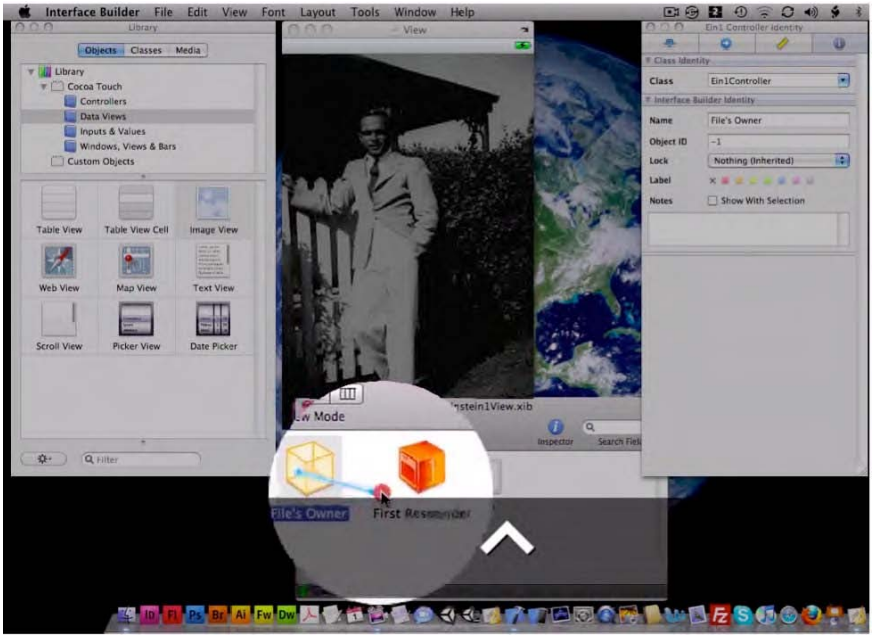


Figura 6–47. Conecte el icono del Propietario del Archivo con la Vista.

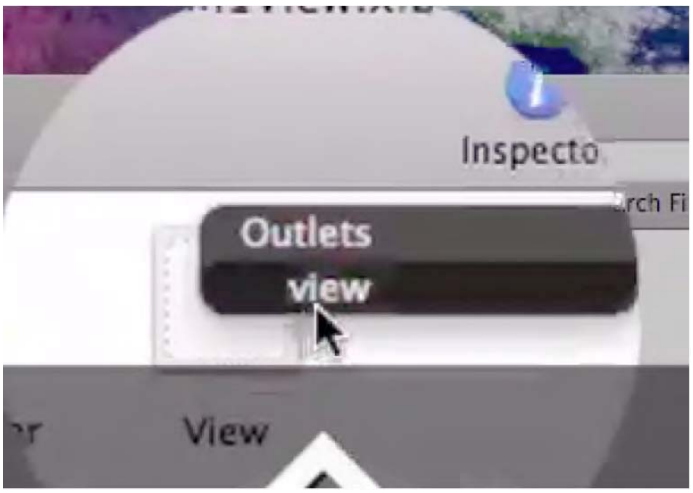


Figura 6–48. Conecte el Propietario del Archivo con la opción Vista dentro de la Vista.

Repita el proceso para la segunda imagen

Tenemos que repetir los mismos pasos mostrados en las Figuras 6-42 a 6-48 para conectar la segunda fotografía, de mis dos abuelos, al `Ein2Controller`. Empiece estas series de acciones finales abriendo el archivo `Einstein2View.xib` y volviendo al Interface Builder, como se muestra en la Figura 6-49.

Igual que para la primera imagen, necesitamos una Vista de Imagen para alojar la segunda fotografía. Por tanto, arrástrela a la pantalla del `Einstein2View.xib`, y, como hicimos antes, haga clic en el icono del Propietario del Archivo, y asócielo con el código en el `Ein2Controller`. Presione control mientras arrastra el icono del Propietario del Archivo de este segundo personaje a su Vista. Recuerde que cuando arrastraba sobre el icono Vista, aparece un menú desplegable con la opción Vista. Apunte a esta opción y suelte. Hecho esto, hemos completado el trabajo con el archivo `Einstein2View.xib`. Introduzca ⌘S para guardar su trabajo y ⌘Q para salir del Interface Builder para volver al Xcode.

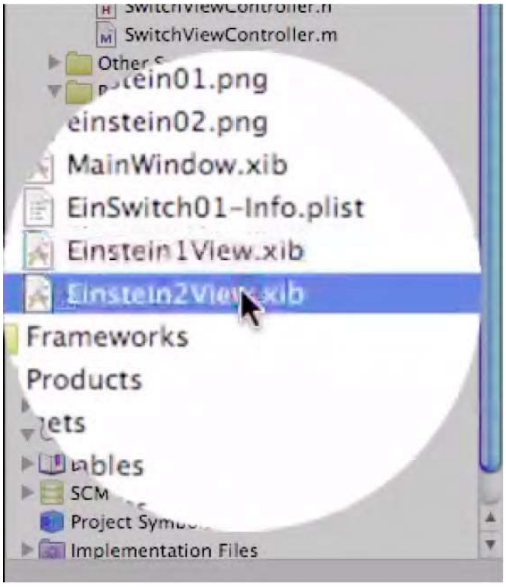


Figura 6–49. Repita el proceso para la segunda imagen.

Ha terminado. ¡Felicidades!

Introduzca ⌘↩ para ejecutar el código. Presionando el botón cambia la imagen… por cambio de vistas… justo como lo programó. ¡Bien hecho! Lo que acaba de conseguir es realmente uno de los puntos de referencia más esenciales e importantes de la programación para iPhone/iPad. En virtud del hecho de que ahora puede programar una aplicación de Cambio de Vista –para mostrar una parte de código en una vista y otra parte en otra vista- está inmediatamente por encima del nivel de programador principiante. Las Figuras 6-50 a 6-54 muestran el fruto de su labor.

Figura 6–50. Presione el botón Cambio de Vista.

Figura 6–51. ¡Hurra, su aplicación funciona!

Figura 6–52. Vemos la imagen inicial, el soltero, en la vista del iPad/iPhone.

Figura 6–53. Esta es la imagen inicial en modo 2x

Figura 6–54. Aquí vemos la imagen cambiada, la pareja feliz, en modo iPad.

Recuerde, sólo hemos hecho el primero de los tres ejemplos de Cambio de Vistas.

1. Desde el cero: einSwitch_001

~~**Video: http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/006_einSwitch_001.htm**~~

~~**Correct Code:**~~
~~http://www.rorylewis.com/xCode/006a_einSwitch01.zip~~

2. Barra de Pestañas: einSwitch_002

Video: http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/007_einSwitch_002.htm

Código Correcto:
http://www.rorylewis.com/xCode/006b_einSwitch02.zip

3. Barra de Pestañas personalizada: einSwitch_003

Video: http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/008_einSwitch_003.htm

Código correcto:
http://www.rorylewis.com/xCode/006c_einSwitch013.zip

¡Aleluya! Debería sentirse bien con este logro.

Hemos dejado atrás lo más difícil de los tres ejercicios. La mayor parte de la gente cambia de vista de este modo, lo hacen de la forma fácil. ¡Lo fácil que le va a resultar el próximo ejemplo!

Así que venga –vamos a por la aplicación de Cambio de Vista fácil. Compruebe lo fácil que es comparada con lo que acaba de hacer.

einSwitch_002—una aplicación de Barra de Pestañas

Como siempre, vamos a empezar con el escritorio limpio. Todo lo que debería tener en el escritorio es las imágenes einstein01.png y einstein02.png del 006_Chapter_6hEinSwitch01.zip que descargó para el ejemplo anterior. Borre, o simplemente guarde en su carpeta de Recursos, el icono de 57x57 píxeles y el archivo de texto de Lazy Load. Guarde las dos imágenes en el escritorio de manera que esté organizado como el mío, como muestra la Figura 6-55.

Figura 6–55. Escritorio limpio al comienzo del ejemplo einSwitch002. Sólo vemos tres iconos: Mac HD y dos archivos de imagen.

1. Abra Xcode e introduzca ⌘⇧N como se muestra en la Figura 6-56. Esto abrirá la ventana de Proyecto Nuevo la que hará clic en la plantilla de Aplicación de Barra de Pestaña. Llame a este proyecto `einSwitch002`, como se muestra en la Figura 6-57.

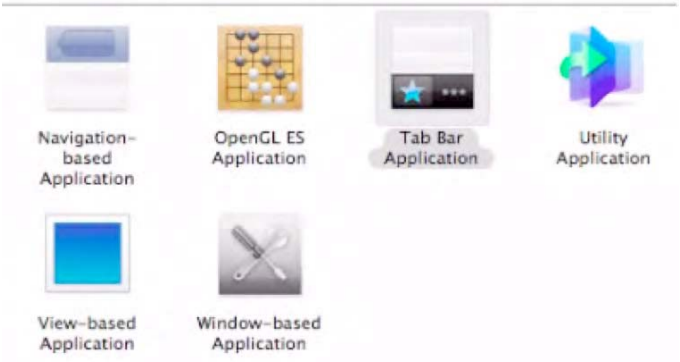


Figura 6–56. Después de introducir ⌘⇧N, cree una Aplicación de Barra de Pestañas

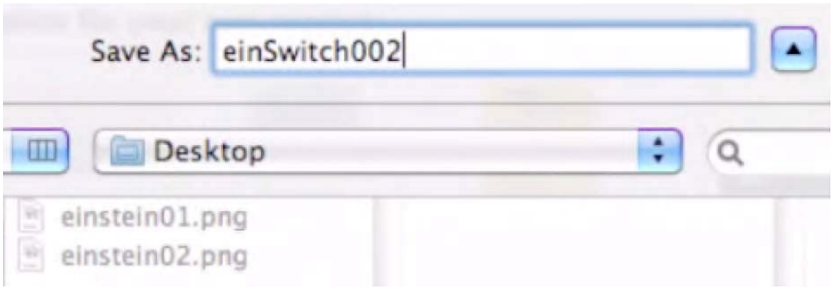


Figura 6–57. Llámele "einSwitch002" u guárdela en el escritorio introduciendo⌘_S.

2. Abra la carpeta de Recursos. Vuelva al escritorio y seleccione ambas imágenes y arrástrelas a la carpeta de Recursos como se muestra en la Figura 6-58. Mientras está en la carpeta de Recursos, abra el archivo `MainWindow.xib` como se muestra en la Figura 6-59.

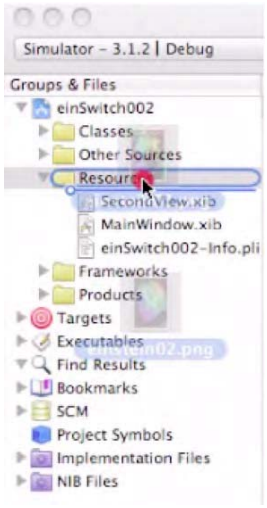


Figura 6–58. Arrastre los dos archivos de imagen a la Carpeta e Recursos.

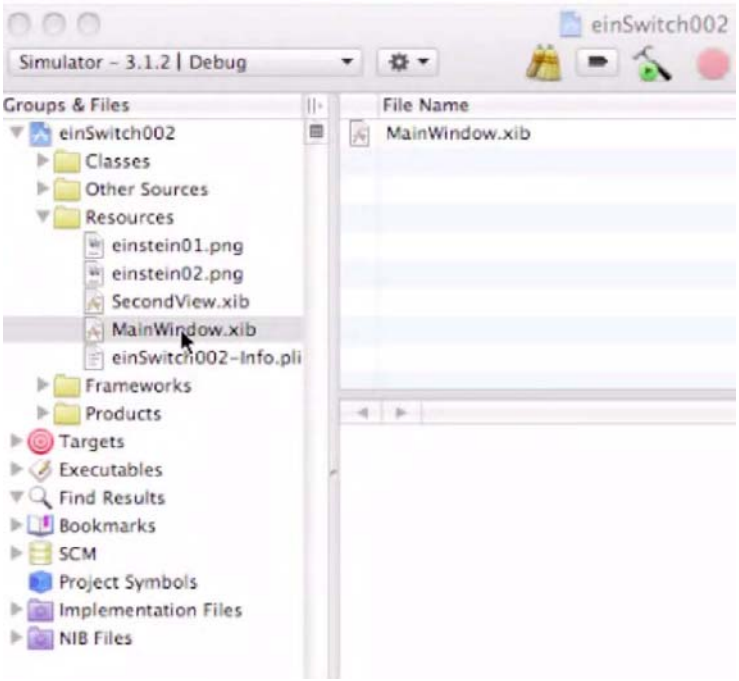


Figura 6–59. Abra el archivo MainWindow.xib.

3. Primero trabajaremos en la Vista Principal y después en la Vista Secundaria. Haga doble clic en el archivo `MainWindow.xib`, que abrirá el Interface Builder automáticamente Una vez abierto, comience por sacar el texto predeterminado del marco del Controlador de la Barra de Pestañas, como se muestra en la Figura 6-60.

Una vez borrado todo el texto predeterminado, introduzca una `UIImageView` de su Librería, como se muestra en la Figura 6-61. Una vez situado este objeto en la pantalla, y mientras todavía está activo, vaya a la ventana de Atributos de Vista de Imagen y seleccione el archivo `einstein01.png` entre las dos alternativas, como se muestra en la Figura 6-62.

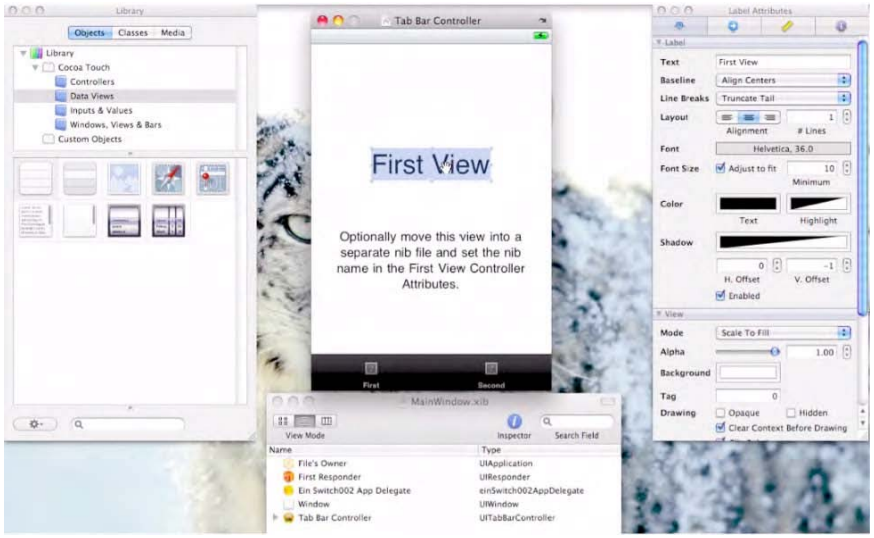


Figura 6–60. Quite todo el texto predeterminado del marco del Controlador de la Barra de Pestañas.

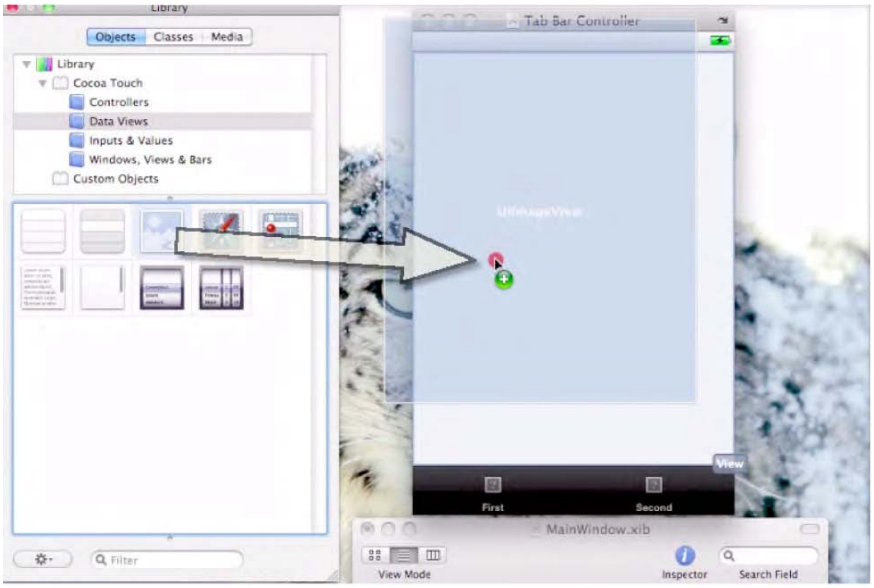


Figura 6–61. Arrastre dentro una UIImageView de la Librería.

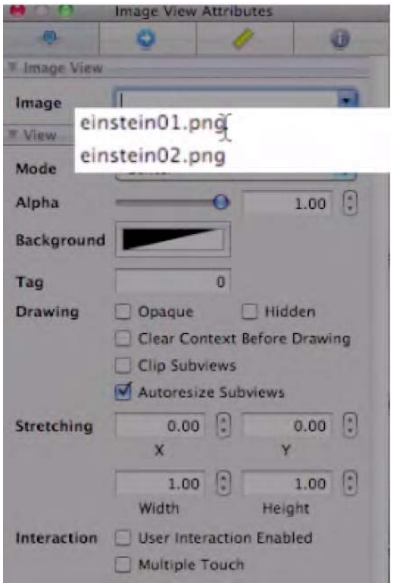


Figura 6–62. Seleccione el archivo einstein01.png file entre las dos alternativas.

4. Haga clic en la pestaña negra inferior izquierda etiquetada “First” (primero), y entonces vaya inmediatamente a la ventana Atributos del Elemento Barra de Pestaña. Haga clic en la celda de Título para borrar el título predeterminado “First”, y entonces introduzca Einstein01, como se muestra en las Figuras 6-63 y 6-64. Puede que tenga que hacer clic dos veces en la celda para situar el cursor adecuadamente.

Ya sabe… probablemente le llevó varias horas llegar a este punto en el primer ejemplo, programando desde el cero. Esto es algo más fácil, ¿verdad?

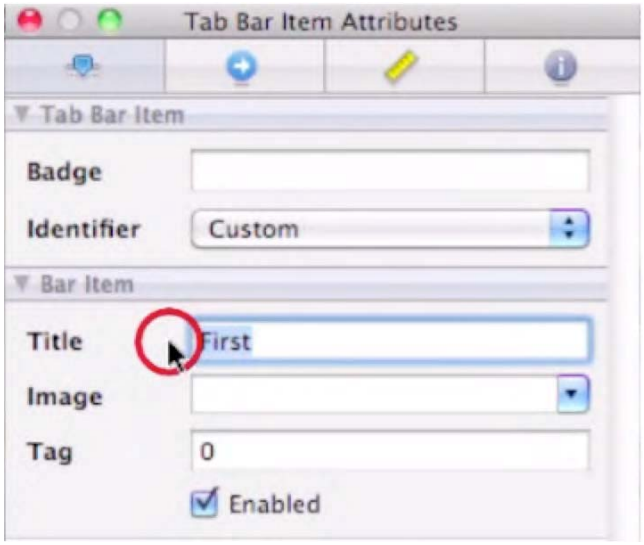


Figura 6–63. Borre el nombre predeterminado de la pestaña de la izquierda.

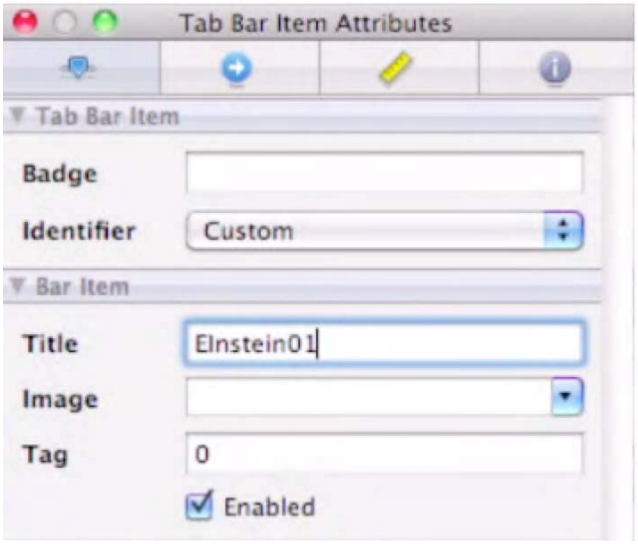


Figura 6–64. Introduzca la nueva etiqueta de la pestaña de la izquierda: Einstein01.

5. Haga clic en la pestaña negra del lado derecho etiquetada “Second” (segunda), y vaya inmediatamente a la ventana de Atributos del Elemento Barra de Pestaña. Haga clic en la celda Título para borrar el título predeterminado “Second”, e introduzca Einstein02, como se muestra en las Figuras 6-65 y 6-66.

Pulse Intro. Guárdelo y vuelva al Xcode.

Figura 6–65. Haga clic en el botón de la Segunda pestaña para repetir los pasos

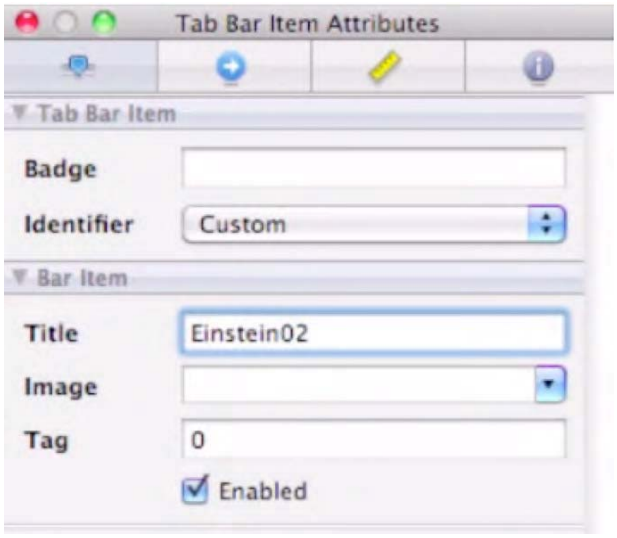


Figura 6–66. Después de borrar el predeterminado, introduzca la nueva etiqueta de la pestaña de la izquierda: Einstein02.

6. Haga doble click en el archivo `SecondView.xib` en la carpeta de Recursos como se muestra en la Figura 6-67. Tal y como hizo antes, borre todo el texto en la Segunda Vista. Vea Figura 6-68.

NOTA: Cuando comienza una aplicación de Barra de Pestaña, la primera Vista se llama Vista Principal, y las que siguen se llaman la segunda, la tercera y así sucesivamente. Acabamos de conectar los puntos para la primera Vista: la Vista Principal. Si queremos que el usuario haga clic en una pestaña que lleve a otra vista, como la segunda foto (los dos abuelos), tenemos que configurar esa Vista aquí.

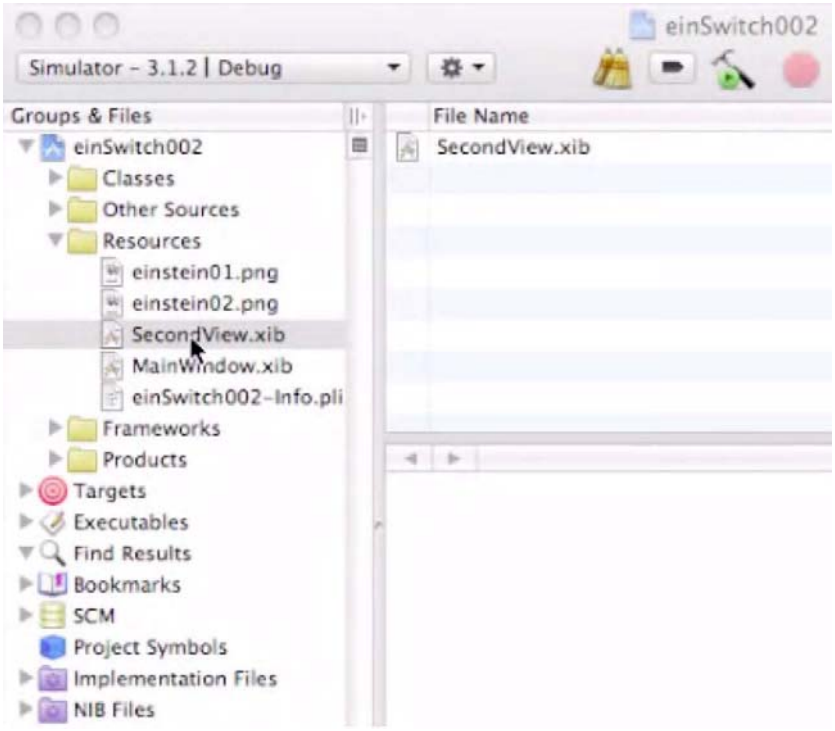


Figura 6–67. Haga doble clic en el archivo `SecondView.xib` de la carpeta de Recursos.

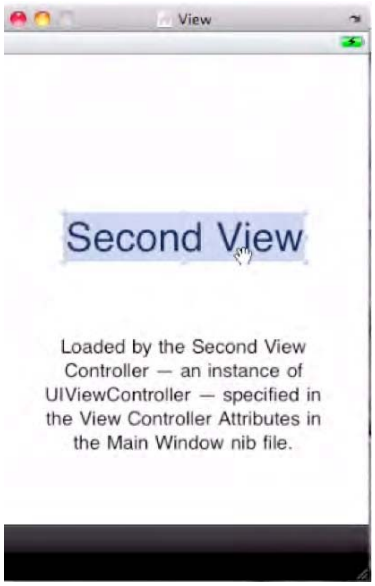


Figura 6–68. Borre todo el texto predeterminado en la Segunda Vista.

Arrastre dentro una UIImageView desde su Librería como se muestra en la Figura 6-69 y sitúela dentro del marco Vista.

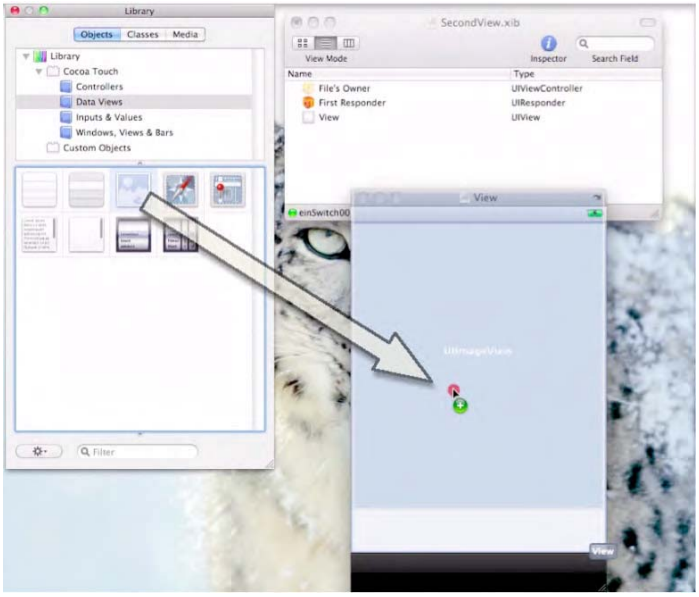


Figura 6–69. Sitúe una UIImageview en la Vista Segunda.

Mientras la pantalla todavía está activa, vaya a la ventana Atributos de Vista de Imagen y seleccione el archivo de imagen `einstein02.png`, como se muestra en las figuras 6-70 y 6-71.

Guarde su trabajo introduciendo `⌘S`, y vuelva al Xcode.

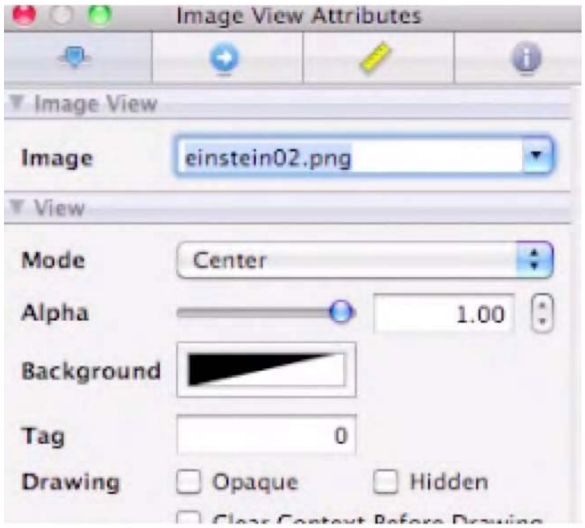


Figura 6–70. Seleccione el archivo einstein02.png entre las dos alternativas.

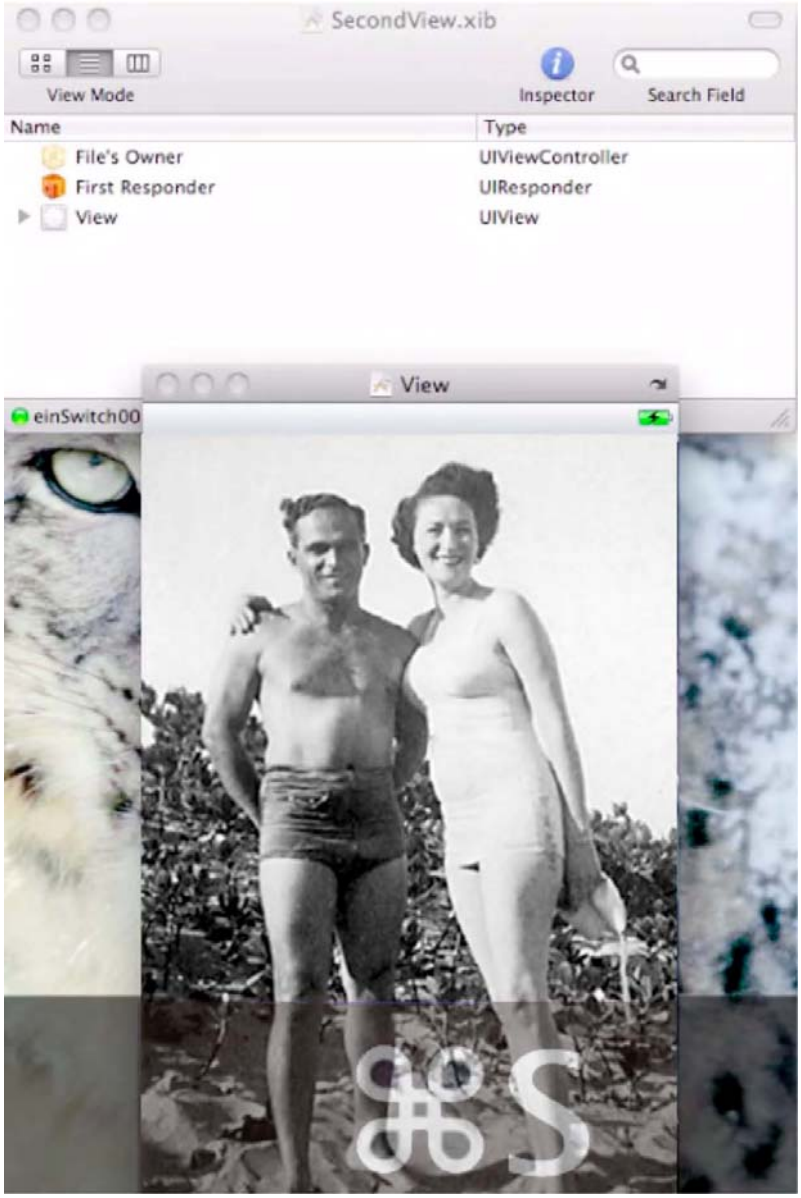


Figura 6–71. Guarde su trabajo introduciendo _S

.

7. Vuelva al Xcode, abra la carpeta de Clases, abra un archivo e introduzca ⌘↩ para poder ejecutarlo como se muestra en la Figura 6-72.

Ya lo se… me odia por obligarle a hacer el primer ejemplo. Si hubiera sabido que este método simple era tan fácil, nunca hubiera accedido a proceder con aquel. Sin embargo hágame caso: los estudiantes que dependían de este camino simple no estaban capacitados para hacer ningún cambio a sus pestañas o para ser efectivamente creativos.

Espera al próximo ejemplo, einSwitch003. Es un híbrido de los dos tipos, ¡y le encantará!

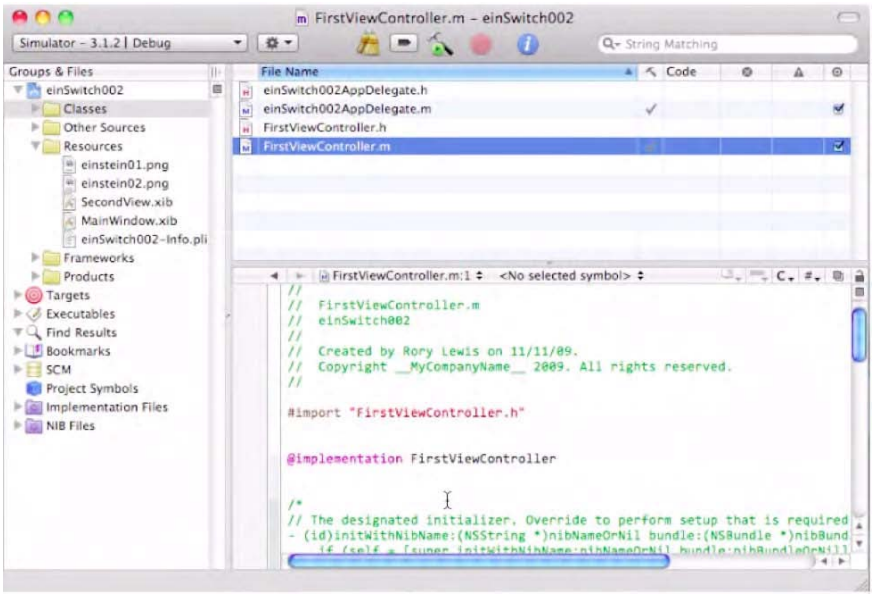


Figura 6–72. Abra Xcode y ejecute el código introduciendo ⌘⮌.

Como puede ver en la Figura 6-73, el método simple consigue resultados muy dignos. La imagen inicial se muestra –y la pestaña Einstein01 queda resaltada

Figura 6–73. Sin una letra de código, nuestra aplicación de Barra de Pestañas aparece con bonitas pestañas negras.

La Figura 6–74 ilustra el cambio de vista mediante el clic en la otra pestaña, Einstein02. La segunda imagen aparece, gracias al código estándar que usted activó

Figura 6–74. Estas bonitas pestañas funcionan estupendamente –y cambian las vistas sin escribir código.

En la Figura 6-75, vemos los mismos resultados: la imagen inicial se muestra y la pestaña Einstein01 se resalta, pero en modo iPad… con la vista iPhone activada.

Figura 6–75. Las pestañas negras tienen una pinta estupenda en la Vista iPad del iPhone.

La Figura 6-76 muestra que el modo iPad no tiene problema con el cambio de vista; la pestaña Einstein02 se resalta y vemos a ambos abuelos.

Figura 6–76. Incluso con las imágenes en blanco y negro, el iPad tiene aspecto sofisticado.

La Figura 6-77 muestra el aumento x2 de la imagen inicial. La resolución La resolución es nítida mientras la pestaña Einstein01 está resaltada.

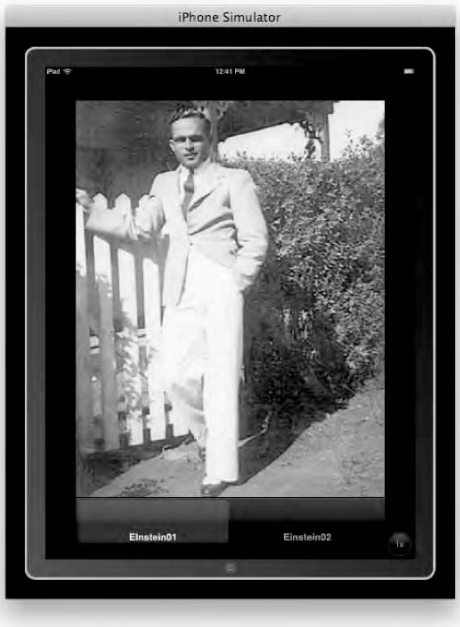


Figura 6–77. La aplicación de Barra de Pestaña Simple funciona estupendamente en vista completa del iPad.

La variación final se describe en la Figura 6-78. El método simple parece que ha hecho su función… y por supuesto, con el mínimo esfuerzo. Dése cuenta, sin embargo, de que habiendo completado el primer ejercicio, su conocimiento se extiende mucho más allá de este simple proyecto de dos pestañas.

¡Las posibilidades son infinitas!

Figura 6–78. Cambio con fluidez en la vista completa del iPad. ¡Sin una sola línea de código!

OK, hemos completado dos de los tres ejemplos de cambio de visa:

1. Desde el cero: einSwitch_001

Video: `http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/006_einSwitch_001.htm`

Código correcto: `http://www.rorylewis.com/xCode/006a_einSwitch01.zip`

2. Barra de pestaña: einSwitch_002

Video: `http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/007_einSwitch_002.htm`

Código correcto: `http://www.rorylewis.com/xCode/006b_einSwitch02.zip`

3. Barra de pestaña personalizada: einSwitch_003

Video: `http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/008_einSwitch_003.htm`

Código correcto: `http://www.rorylewis.com/xCode/006c_einSwitch03.zip`

Ha visto lo fácil que era el método “simple”, y realmente lo aprecia porque el primer enfoque era una verdadera lata. En esta tercera iteración, le enseñaré cómo ajustar las cosas un poco más. Su cerebro creativo unirá los puntos y encontrará relativamente fácil hacer ajustes menores ahora que entiende la lógica interna.

einSwitch_003—una aplicación basada en ventanas.

Las acciones preliminares a realizar para einSwitch_003 son idénticas que para einSwitch_002. Los únicos elementos en nuestro escritorio serán los archivos de imagen `einstein01.png` y `einstein02.png` del `006_Chapter_6hEinSwitch01.zip` que usted descargó para einSwitch_001. Su escritorio debería estar organizado como el mío, como se muestra en la Figura 6-79.

Figura 6–79. Su escritorio al principio: Mac HD y los dos archivos de imagen.

1. Abra Xcode e introduzca ⌘⇧N como se muestra en la Figura 6–80; esto abrirá la ventana del Nuevo Proyecto, en el que usted hará clic en la plantilla de Aplicación Basada en Ventanas. Llame a este proyecto "`einSwitch003`," como se muestra en la Figura 6-81.

Figura 6–80. Después de introducir ⌘⇧N, seleccione la plantilla de Aplicación basada en Ventanas.

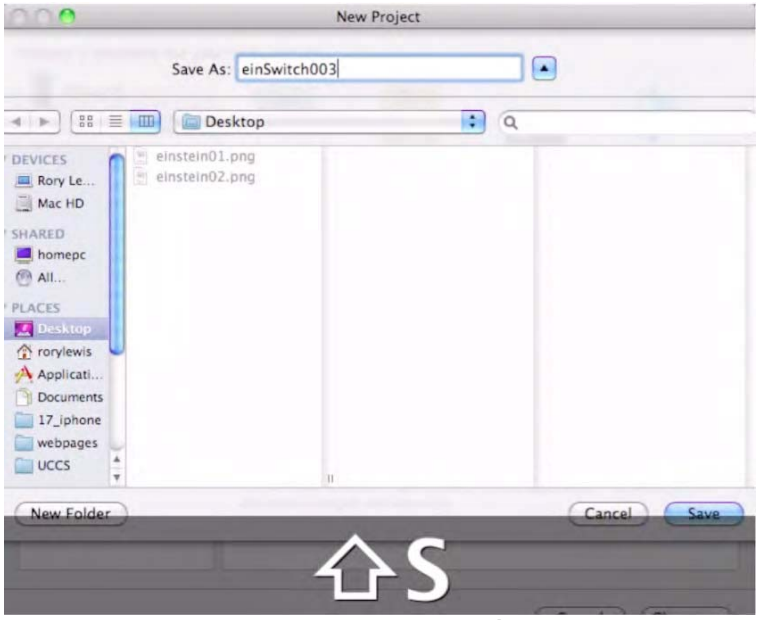


Figura 6–81. Llame a su nuevo proyecto "einSwitch003," y guárdelo en el escritorio introduciendo ⌘S.

2. Una de las cosas que tenemos que hacer es ir a la carpeta de Clases. Haga clic para abrirla y entonces abra el archivo de cabecera haciendo clic en `einSwitch003AppDelegate.h`. Vamos a echar un vistazo a algunos de los sitios donde las modificaciones y personalizaciones del código existente podrían servirnos potencialmente.

Tenemos que meter tanto como podamos de ese material fantástico que hacía que el trabajo de metodología simple funcionara y al mismo tiempo conservara el control de las cosas.

Echemos un vistazo a un poco del código pre-programado que los programadores listos de Apple han escrito, un trozo de código encapsulado en una Clase de Referencia llamada `UITabBarController`. Tiene todo el código pre-escrito que muestra pestañas para seleccionar entre modos diferentes y para mostrar las vistas para ese modo. La clase `UITabBarController` hereda del código que usted programó en el primer ejemplo, específicamente la clase `UIViewController`. Los controladores de barra de Pestañas han hecho su propia vista accesible a través de la propiedad `view`.

Vea en la Figura 6-82 un diagrama que describe cómo se montan las vistas en la interfaz de la barra de pestañas. Podemos cambiar el aspecto y la sensación de la barra de pestañas, y la barra de vistas puede cambiar, pero las vistas que los manejan no cambian.

Figura 6–82. El diagrama del UITabBarController de Apple. Esta disposición parece simple al principio, pero intentar implementarla puede ser bastante complejo…¡ a menos que ya haya completado el ejemplo del primer capítulo!

Por tanto, mejor que pelearnos con esto ahora, haremos tres cosas:

1. Adoptar el `UITabBarControlleDelegate` insertándolo en nuestro `@interface einSwitch003AppDelegate : declaration`.
2. Añadir una nueva declaración de `UITabBarController *tabBarController` después de la declaración `UIWindow *window`, pero antes de la llave `}`.
3. Añadir un `UITabBarController` outlet. Una vez que ha añadido estos tres pasos, como se muestra en el código en negrita de abajo, introduzca `⌘S` para guardar. Esto se muestra en la Figura 6-83.

```
#import <UIKit/UIKit.h>

@interface einSwitch003AppDelegate : NSObject <UIApplicationDelegate,↩
 UITabBarControllerDelegate> {
        UITabBarController *tabBarController;
        UIWindow *window;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet UITabBarController *tabBarController;

@end
```

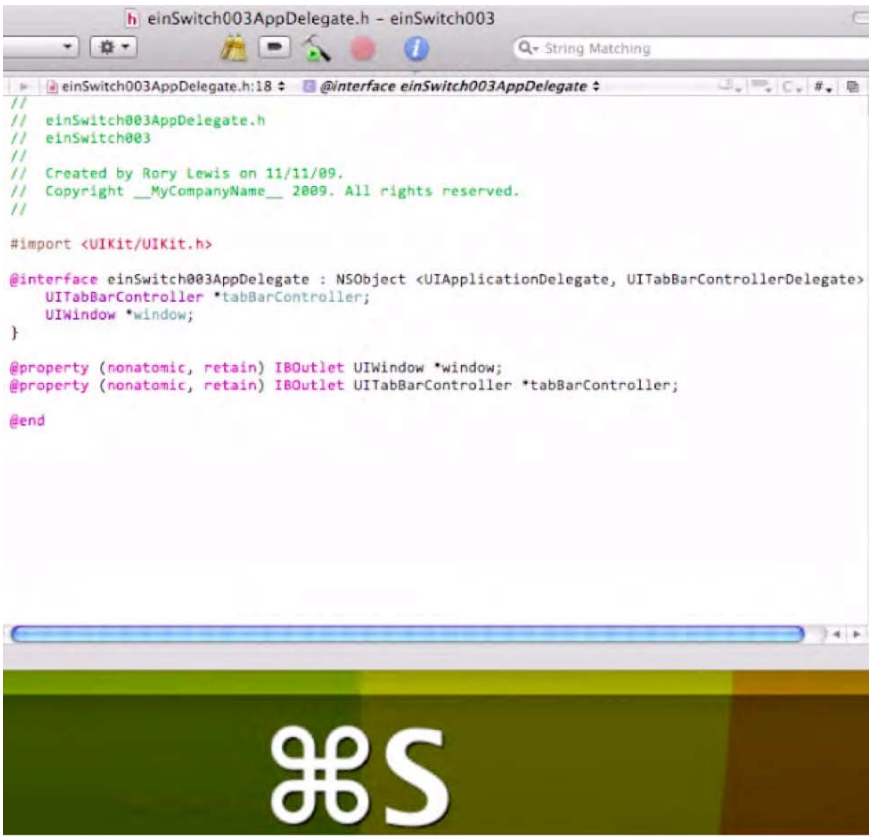


Figura 6–83. Después de añadir una salida UITabBarController, guarde el código introduciendo ⌘S

4. Ahora tenemos que programar el archivo de implementación. Lo primero que haremos aquí es

a. Sintetizar la `tabBarController`
b. Añadir un controlador de Vista subvista
c. Liberar el `tabBarController`

Una vez ha añadido estos tres pasos, como se muestra en el código en negrita siguiente, el resultado debería aparecer como en la Figura 6-84. Una vez hecho todo esto, introduzca ⌘S para guardar su trabajo.

```
#import "einSwitchOO3AppDelegate.h"

@implementation einSwitchOO3AppDelegate

@synthesize window;
@synthesize tabBarController;

-(void)applicationDidFinishLaunching:(UIApplication *)application {

    [window addSubview:tabBarController.view];
    [window makeKeyAndVisible];
}

-(void)dealloc {
    [window release];
    [tabBarController release];
    [super dealloc];
}
```

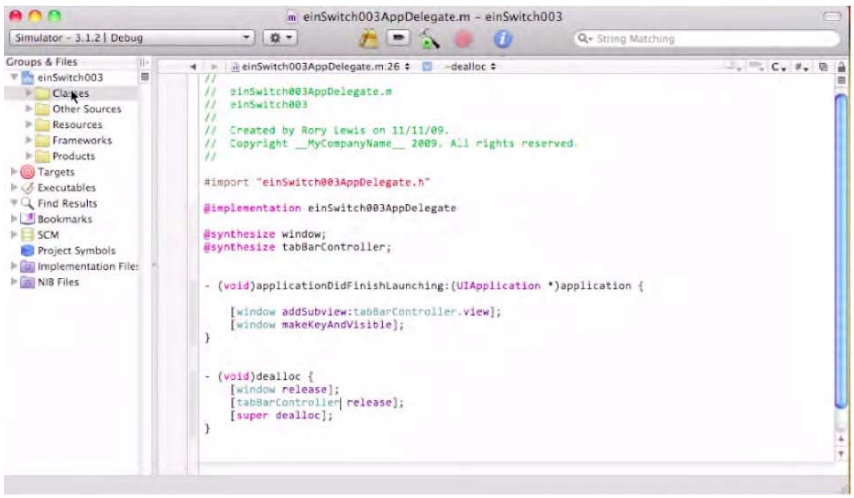


Figura 6–84. Después de que sintetice el tabBarController, añada una subvista y libere el tabBarController, guarde su trabajo introduciendo ⌘S .

5. Cuando usamos el método simple para la Aplicación de Barra de Pestañas en einSwitch_002, añadió todo tipo de extras, archivos que nosotros necesitamos añadir aquí. Pensemos esto por un segundo. El reto es éste:

¿Cómo podemos averiguar qué archivos necesitamos añadir, y que todavía no tenemos?

En el futuro, cuando llegue a este punto crucial, pregúntese: ¿"Cuántas vistas necesita mi aplicación para iPhone/iPad?" Usted podría tener ocho vistas diferentes en su programa. Esto significa que tendría que añadir ocho nuevas vistas. Nosotros tenemos dos fotografías, por tanto solo necesitamos añadir dos vistas. Hacemos esto yendo a la carpeta de Clases, introduciendo ⌘N, y seleccionando la subclase UIViewController, como se muestra en la Figura 6-85. Asegúrese de que selecciona la opción "With XIB for user interface" (Con XIB para interfaz de usuario), como se muestra en la Figura 6-86.

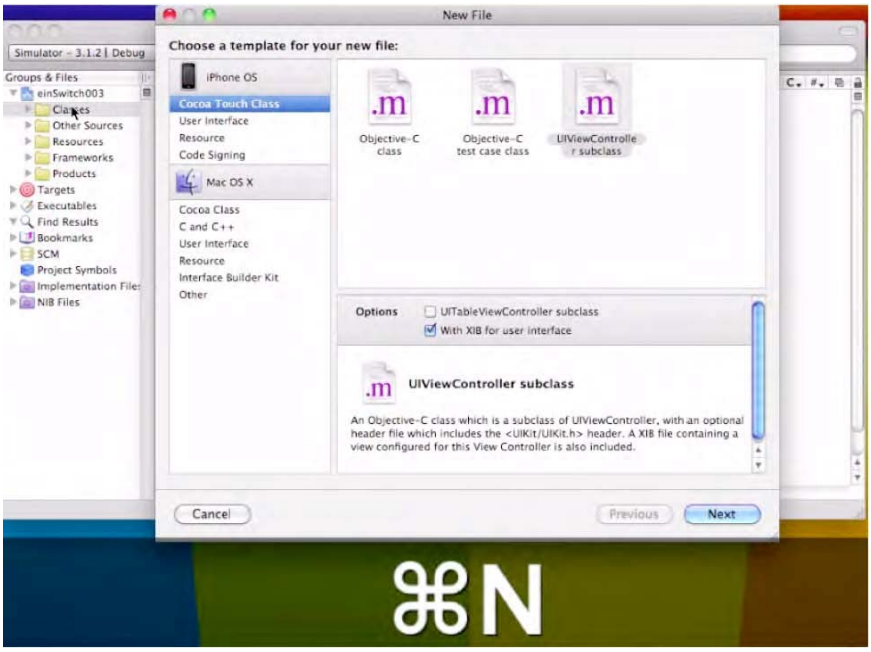


Figura 6–85. En la carpeta de Clases, introduzca ⌘N y seleccione la subclase UIViewController.

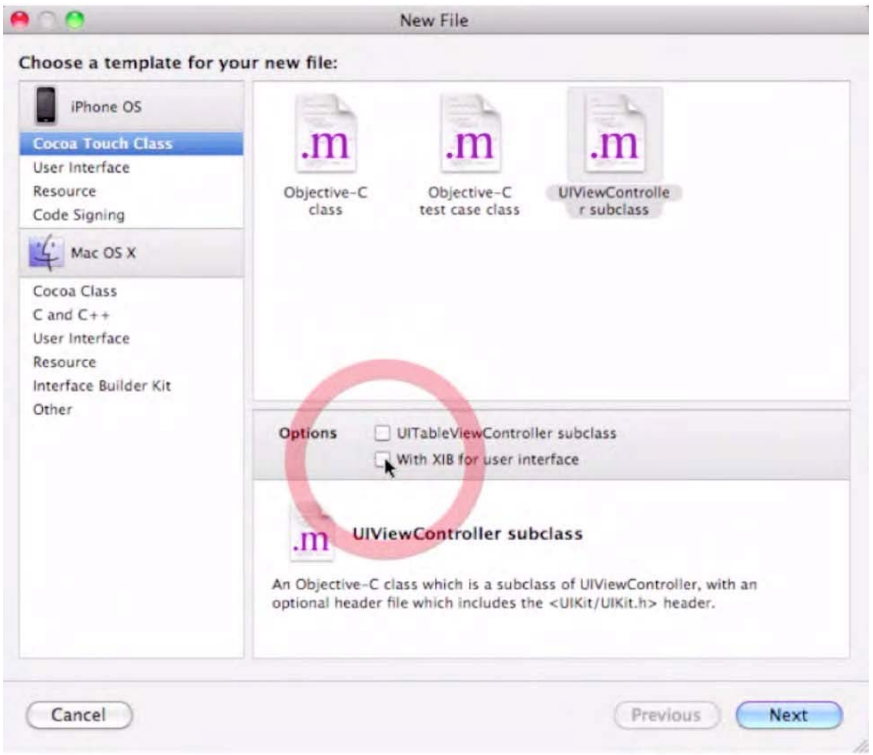


Figura 6–86. Asegúrese de que selecciona la opción de hacer el archivo nib automáticamente

.

Seremos increíblemente creativos y llamaremos el archivo de implementación del primer Controlador de Vista, `FirstViewController.m`, como se muestra en la Figura 6–87.

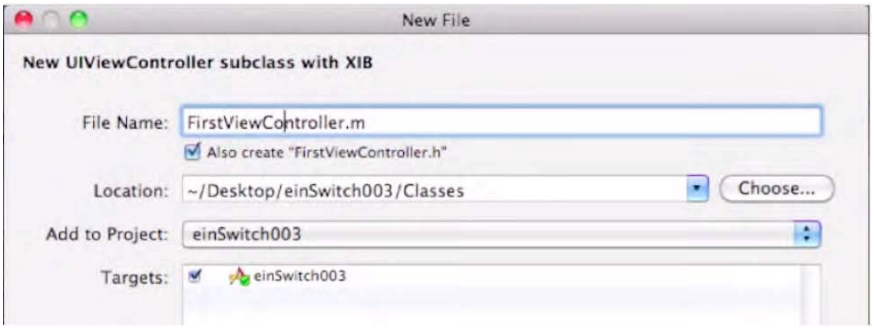


Figura 6–87. Llame la primera subclase de UIViewController—con XIB—FirstViewController.m.

Ahora, tenemos que crear y dar nombre al segundo Controlador de Vista. Así que vuelva a la carpeta de Clases, introduzca ⌘N, y seleccione la subclase UIViewController como hemos visto. Asegúrese otra vez de marcar la opción "With XIB for user interface." Guárdelo como `SecondViewController.m`, como se muestra en la Figura 6-88.

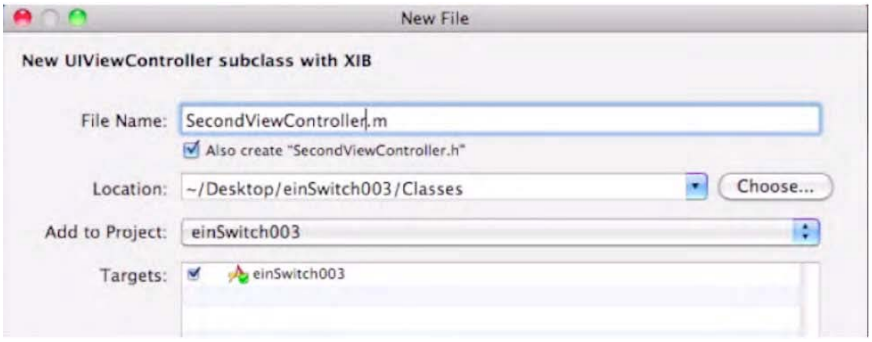


Figura 6–88. Llame la segunda subclase de UIViewController —con XIB— SecondViewController.m.

6. Es un hábito estupendo guardar todos los archivos nib en la carpeta de Recursos. Sin embargo, los dos archivos nib automáticos que acaba de crear están todavía en la carpeta de Clases. Vamos a moverlas a la carpeta de Recursos, como se demuestra en la Figura 6-89.

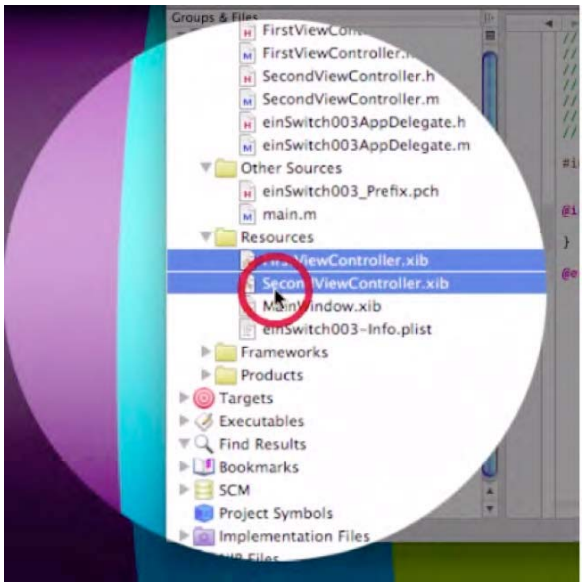


Figura 6–89. Arrastre los dos archivos nib nuevos a la carpeta de Recursos.

El próximo elemento de la agenda es mover nuestro archivo a la carpeta de Recursos. Vaya al escritorio, seleccione las dos fotografías, y arrástrelas dentro de la carpeta de Recursos, como se ilustra en la Figura 6-90. Asegúrese de que, cuando la suelta dentro de la carpeta de Recursos, marca la casilla “Copy items into destination group’s folder (if needed)” (Copiar elementos en la carpeta del grupo de destino –si es necesario-). Ver Figura 6-91.

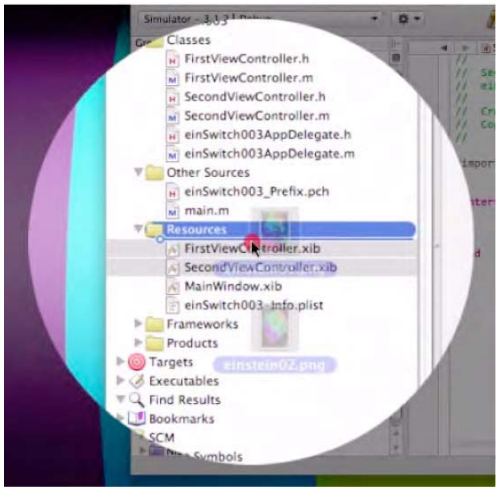


Figura 6–90. Arrastre los dos archivos de imageneinstein01.png y einstein02.png, dentro de la carpeta de Recursos.

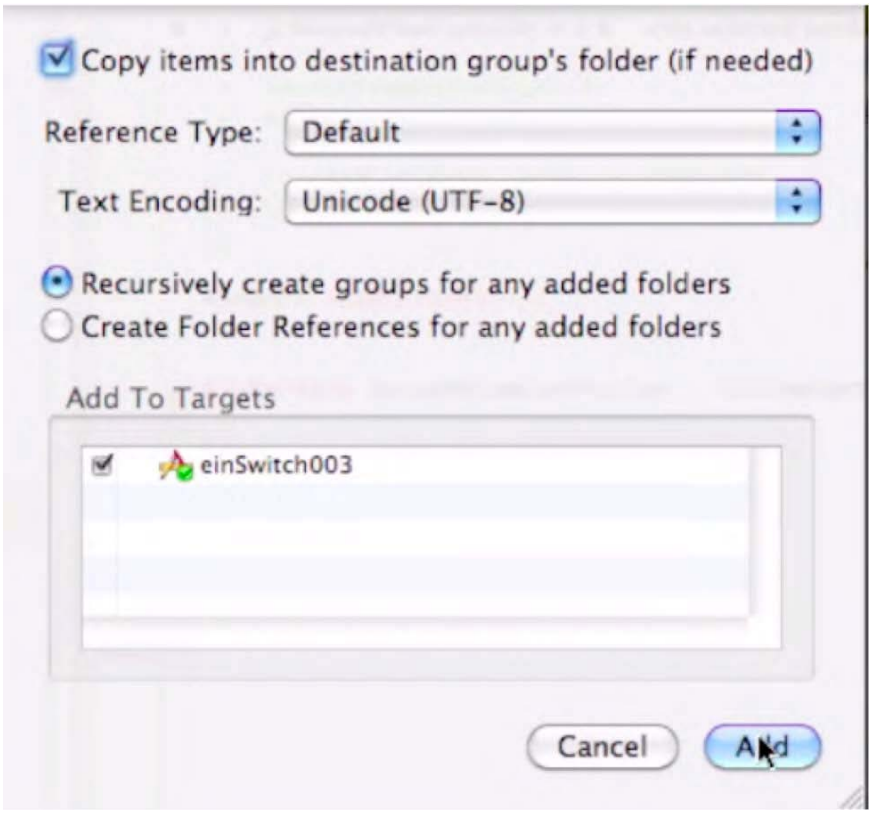


Figura 6–91. No se olvide de marcar "Copy items into destination group's folder (if needed)"!

CONSEJO: Esta puede ser la última vez que te lo recuerdo. De ahora en adelante **¡RECORDARÁS ESTO!**.

7. Es hora de editar el ViewControllers. De la carpeta de Recursos, abra el archivo `FirstViewController.xib`. Cuando se abra el Interface Builder, arrastre una Vista de Imagen a su pantalla, como se muestra en la Figura 6-92. Por supuesto, queremos que la primera imagen se vea en el primer Controlador de Vista. Para conseguirlo, vaya al Inspector de Atributos de Vista de Imagen (Image View Attributes Inspector) y, en el

menú desplegable de Imagen, seleccione `einstein01.png`, como se muestra en la Figura 6-93. Hecho esto, introduzca ⌘S para guardar. Entonces introduzca ⌘Q para salir del Interface Builder y volver a Xcode.

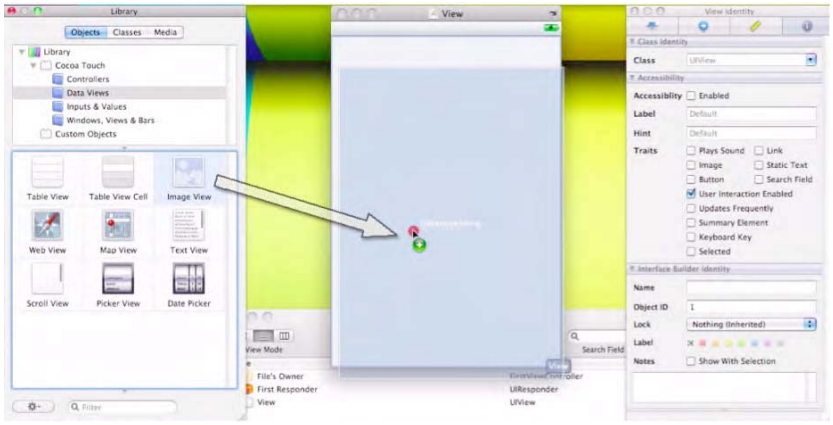


Figura 6–92. Arrastre una Vista de Imagen a su pantalla

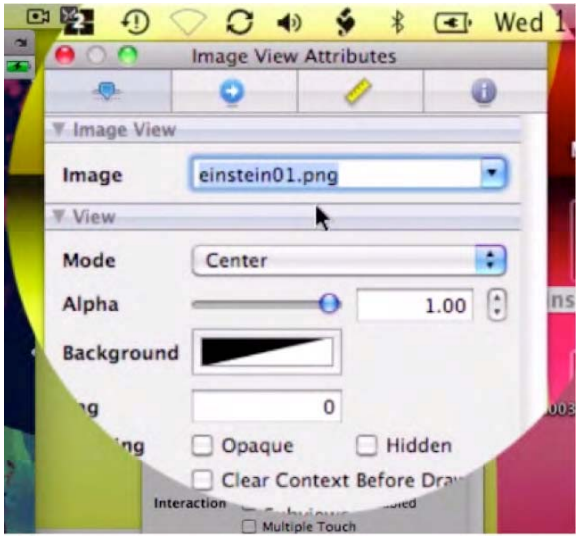


Figura 6–93. Seleccione einstein01.png del menú desplegable.

Por supuesto, tenemos que repetir el proceso completo para el segundo archivo de Controlador de Vista. Por tanto, abra `SecondViewController.xib`, y arrastre una Vista de Imagen a su pantalla. Desde el menú desplegable de Imagen, seleccione `einstein02.png`. Como antes, introduzca ⌘S para guardar, y ⌘Q para salir del Interface Builder.

8. Como se indica en la Figura 6-94, vaya a la carpeta de Recursos y haga clic en clic `MainWindow.xib`. Lo primero que tenemos que hacer es arrastrar un Controlador de Barra de Pestañas de la Librería, dentro del marco MainWindow como se muestra en la Figura 6-95.

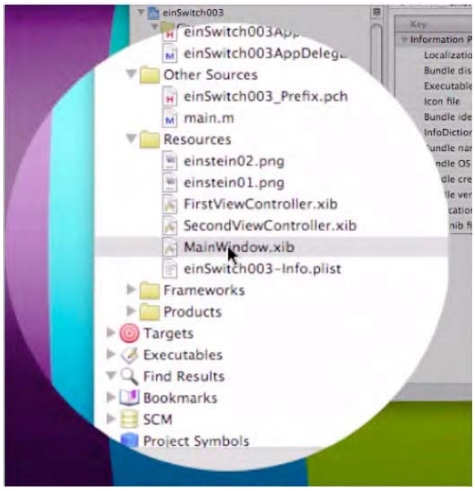


Figura 6–94. De la carpeta de Recursos, abra el MainWindow.xib

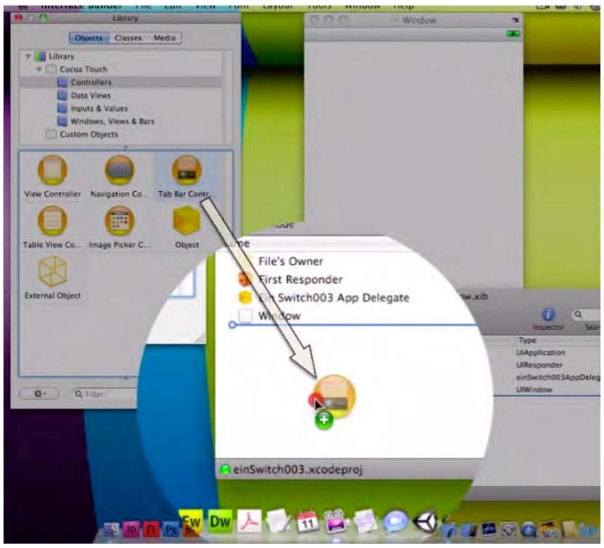


Figura 6–95. Arrastre un Controlador de Barra de Pestañas de la Librería al marco MainWindow.

9. A continuación, tenemos que conectar nuestra barra de pestañas al `EinSwitch_003 App Delegate`. Arrastre, con la tecla de control presionada, del `EinSwitch_003 App Delegate` al Controlador de Barra de Pestañas, como se ve en la Figura 6-96. Mientras lo llevamos al Controlador de Barra de Pestañas, aparece el menú desplegable negro, donde la opción de una de las salidas es `tabBarController`, que es exactamente lo que queremos. Ver Figura 6-97.

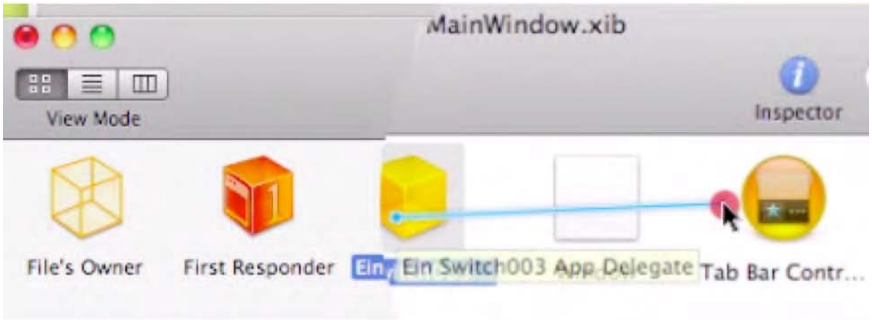


Figura 6–96. Arrastre, con la tecla de control presionada, desde EinSwitch_003 App Delegate al Controlador de Barra de Pestañas.

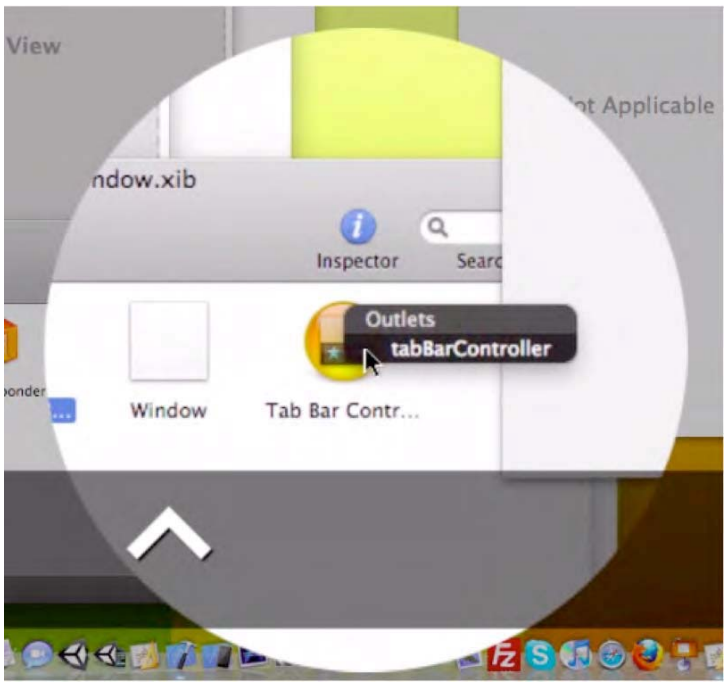


Figura 6–97. Solo hay una opción en el menú Outlets; seleccionar tabBarController

Aquí es donde vemos que tenemos lo mejor de los dos mundos: 1) tenemos el control de lo que sucede por debajo, y 2) el lujo de tener la mayor parte del código pre-escrito por Apple.

10. Si fuera un programa más largo y complejo, podríamos haber conectado con otros controles que, por ejemplo, cambiaran los iconos donde están los usuarios en un juego, o la lengua que prefiere un usuario en una aplicación. Puede haber un millón de razones por las que su programa o juego necesite tener flexibilidad del tipo que da la barra de pestañas. Recuerde que en el enfoque simple teníamos muy pocas opciones. Aquí, sin embargo, siendo todavía todo mucho más simple que en el primer ejemplo, vemos que arrastrar con control presionado de un icono a otro es relativamente fácil e intuitivo

¡Nos acercamos al fin de nuestro viaje! Ahora, necesitamos conectar nuestros Controladores de Vista a los nibs & UIViewControllers correctos.

Primero tenemos que ampliar los contenidos de los iconos. Haga clic en el botón de en medio de Modo de Vista, situado en la parte superior izquierda de `MainWindow.xib` (Figura 6-98).

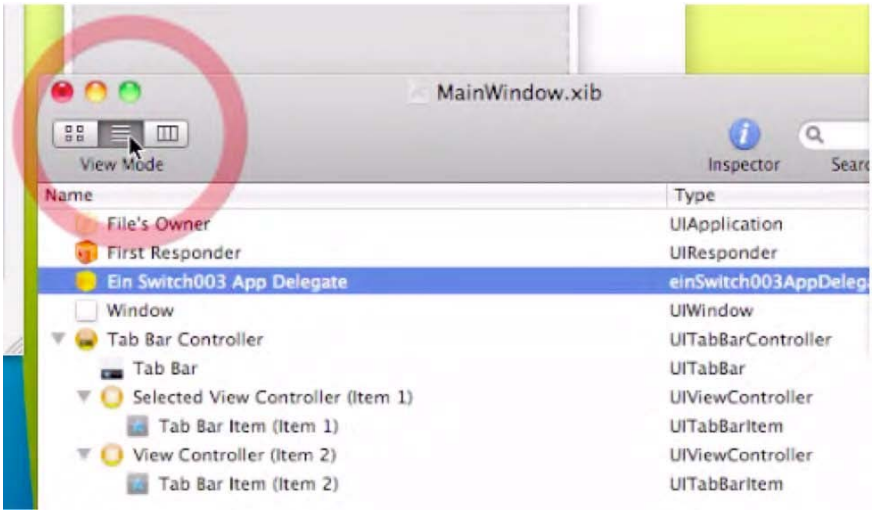


Figura 6–98. Seleccione el botón del medio del Modo de Vista para ampliar los contenidos del Controlador de Barra de Pestañas.

Seleccione el primer Controlador de Vista, “Selected View Controller (Item1),” como se muestra en la Figura 6-99, y haga clic.

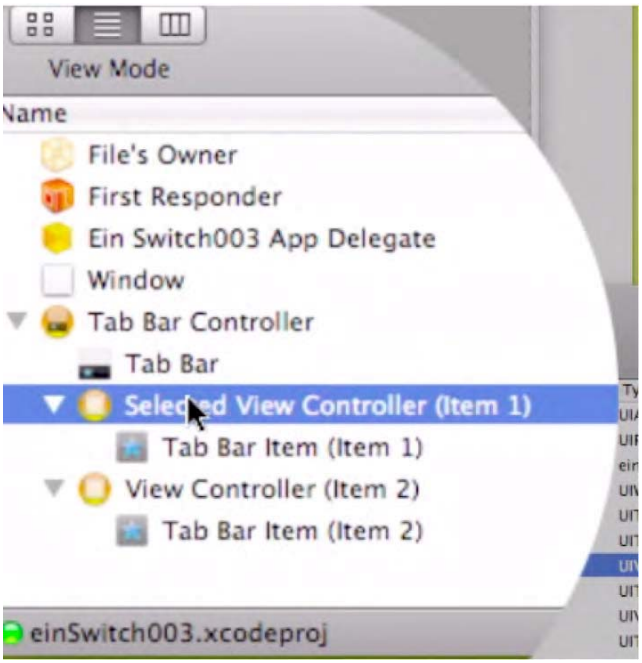


Figura 6–99. Seleccione el primer Controlador de Vista: Selected View Controller (Item1).

Vaya inmediatamente al inspector de atributos del Controlador de Vista, y conéctelo con el archivo nib `FirstViewController`, como se ve en la Figura 6-100

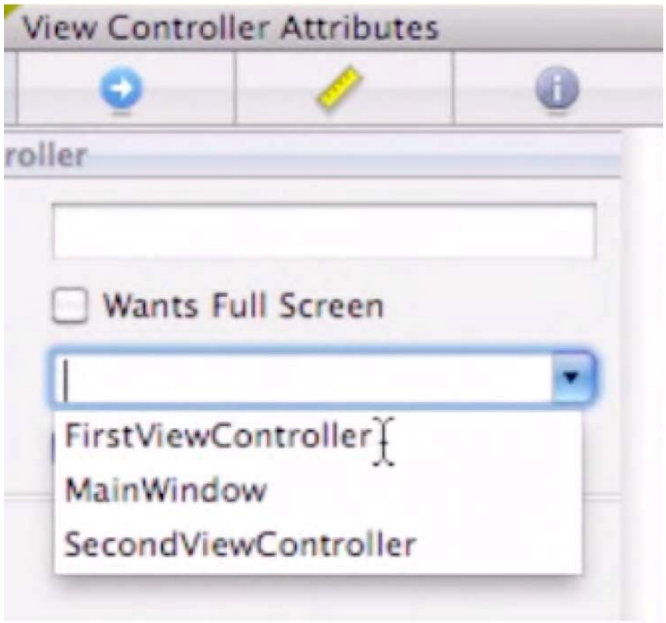


Figura 6–100. Seleccione el FirstViewController nib.

A continuación tenemos que conectar el `UIViewController`. Vaya a la ventana de Atributos del Controlador de Vista y haga clic en la pestaña de Identidad. Entonces, desde el menú desplegable, como se ve en la Figura 6-101, seleccione FirstViewController.

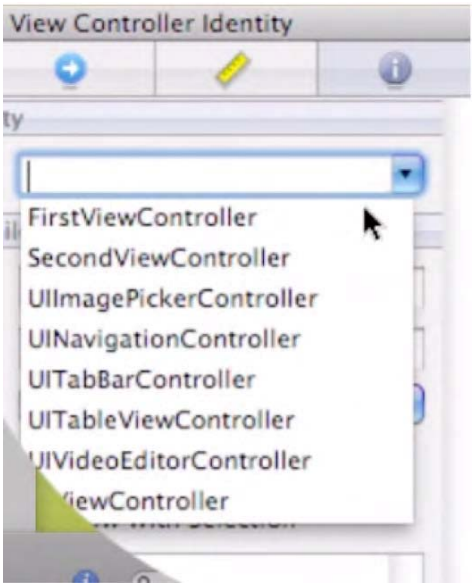


Figura 6–101. Seleccione FirstViewController del menú desplegable

11. De la misma manera conectaremos el segundo Controlador de Vista. Por supuesto, se asegurará de seleccionar el segundo, no el primer Controlador de Vista. Selecciónelo del `MainWindow.xib`, como e ve en la Figura 6-102.

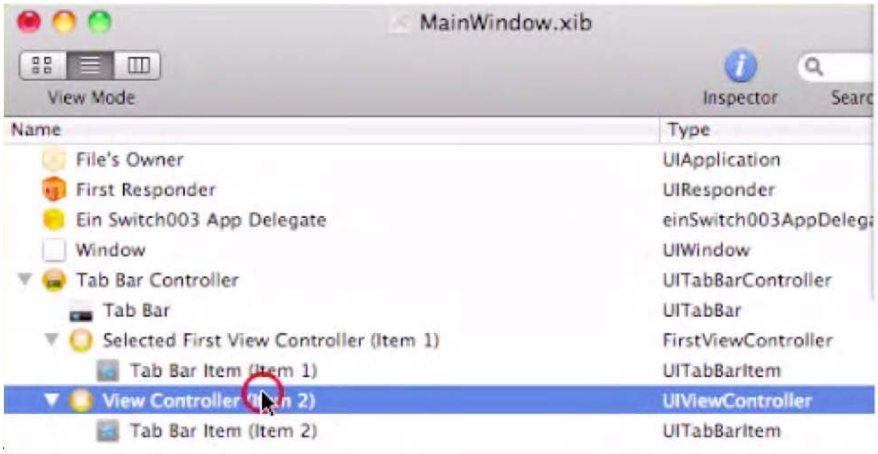


Figura 6–102. Seleccione el segundo Controlador de Vista, se muestra como View Controller (item2).

Con la pestaña de Atributos de Controlador de Vista todavía abierta, seleccione SecondViewController del menú desplegable, como se muestra en la Figura 6-103.

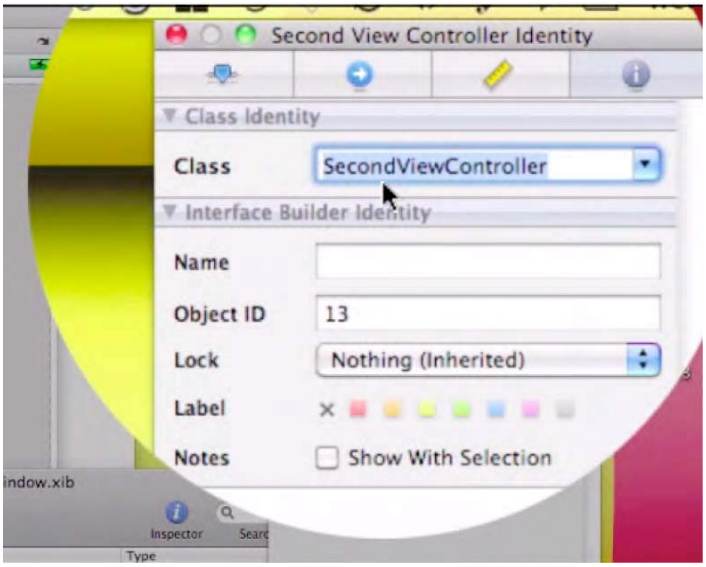


Figura 6–103. Escoja SecondViewController del menú desplegable de Clase.

Nuestra última acción, antes de ejecutar el código, es elegir el nib `FirstViewController` e introducir ⌘S para guardar todo nuestro trabajo, como se ve en la figura 6-104. Entonces introducir ⌘Q y volver al Xcode para compilar nuestro código.

¡Uauuuu! ¡Lo hizo! La Figura 6-105 muestra el resultado.

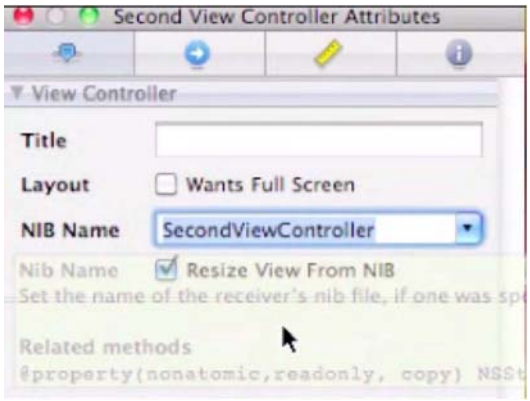


Figura 6–104. Seleccione el SecondViewController.xib, y guarde el trabajo introduciendo ⌘S.

Figura 6–105. Con su comprensión ampliada, y su buena disposición para usar herramientas preparadas, el resultado es ¡ÉXITO! La programación de cambio de vistas está ahora a su alcance.

Profundizando en… su cerebro

Hace años, cuando aprendía a programar en C como ingeniero electrónico luchador en la Universidad de Syracuse, practiqué estos pasos una y otra vez. Me llevó literalmente dos semanas a 6-10 horas diarias dominar con fluidez la programación en C; trabajé con objetos hechos a mano, clases y métodos que ejecutarían un sistema operativo mayor. Fue un proyecto de fin de semestre.

Después, repetí el programa al completo otra vez, refiriéndome a mis notas a menudo. Creé una tarjeta de puntuación y ponía un guión cada vez que miraba mis notas. Cuando volví al programa por tercera vez, creo que tardé unas cinco horas. He olvidado las veces que rehice el código, desde cero, pero al final fui capaz de escribir todo el programa en menos de una hora sin hacer una sola referencia a mis notas. La mayoría de las veces era mi terrible mecanografía lo que me frenaba.

Si, le estoy recomendando que haga lo mismo con esta aplicación de Cambio de Vista. Saque todos los elementos de su escritorio, excepto los cuatro archivos con los que empezamos. Si está dispuesto a hacer esto, apuesto a que –la quinta vez- mirará menos de 10 veces sus notas. También confío en que, después de 10 o 15 intentos prácticos, será capaz de codificar todo, como yo, en menos de cinco minutos -sin mirar notas!

Se que puede parecer excesivo para algunos de ustedes, pero el asunto que planteo es que la práctica realmente lleva a la perfección. Esto es así tanto en la esfera mental como en la física. Se que ha oído esto antes: Una mente es una cosa terrible para desperdiciar.

¡Cuanto más comprometido esté ahora, más seguro será su éxito futuro!

Capítulo 7

Arrastrando, Rotando y Cambiando de Tamaño.

En este capítulo abordaremos juntos nuestro noveno ejercicio. Esta aplicación será la primera en incluir aspectos avanzados en nuestras aplicaciones iPhone e iPad.: la habilidad para arrastrar, rotar y cambiar de tamaño objetos en la pantalla con tus propios dedos. Esta es una de las funcionalidades del iPhone e iPad que más éxito le han dado a estos dispositivos.

En este capítulo nos centraremos es esta característica integral, por medio de la acomodación de estas acciones naturales. Consideraremos esas interacciones desde el mismo interior de la aplicación-del mismo modo en que los mecánicos levantan los coches con plataformas de elevación para tener desde abajo un acceso total al motor y a la transmisión. Después de que tuneemos el coche y lo preparemos en función de nuestro diseño y entreguemos las llaves del coche modificado a usuario final, no nos quedará otra cosa que decir “Buah, este transmisión multi-touch funciona de perlas-fina y suave!”. Nosotros, como los desarrolladores y programadores, debemos aprender a generar esas capacidades- en el código. Por tanto ¿Cómo funcionan estos componentes?

La respuesta sencilla es de tipo matemática; la compleja es trigonométrica. Esta manipulación de objetos e imagines es posible mediante matrices. Ahora que estás a punto de convertirte en un programador iPhone e iPad avanzado y estás metido de lleno en el reino “geek”, pasaremos a efectuar una mirada mucho más profunda dentro de la máquina. Respira y vamos a ello!

Echemos un vistazo a las tripas del SO del iPhone. Nuestra primera tarea es comprender el “multi-touch”. ¿Qué significa esta habilidad?¿Por qué es tan importante?

NOTA: Con afirmaciones como la anterior, en algunos momentos daré a entender que solo me refiero al iPhone. En todo caso, ten siempre en cuenta que cuando haga una afirmación de este tipo, también incluyo al iPad y al iPhod Touch de un modo implícito.

Ya en los primeros días de la computación de dispositivos hand-held, una persona podía usar un palito de plástico con forma de lápiz llamado stylus para interactuar con el dispositivo. Este stylus era algo rebelde y molesto en el uso, se colgaba el sistema con frecuencia y a menudo los perdíamos dado su reducido tamaño.

La innovación de la pantalla multi-touch forzó que este accesorio quedara en poco tiempo obsoleto. El multi-touch dejó al stylus en mantillas debido a la pistas de superficie supersensitivas de los dispositivos, los cuales podían controlarse con los dedos. Lo bueno de que pudiera ser controlado por los dedos es que estos siempre los tenemos a mano y no tienden a perderse, jeje. Hasta aquí este es el origen de la "touch" dentro de la expresión "multi-touch". Con respecto a su prefijo "multi", la evolución fue bastante sencilla: Los humanos, por regla general tenemos diez dedos. Algunos ingenieros creativos se dieron cuenta de que podrían crearse o representarse multitud de de nuevas acciones si incluíamos el uso de alguno de los otros dedos también.

Por tanto, *Multi-Touch* fue un término dado para esta plataforma (ahora patentada), lo que implica que es posible una combinación de imputs simultáneos-y quizá también deseable. Esta nueva característica expandió el abanico de posibilidades, permitiendo el la utilización de gestos y acciones complejas. Fué un gran invento… ¡y continua siéndolo!

La genialidad y flexibilidad de este dispositivo ligero y atractivo- una herramienta que puede ser manejada rápida e intuitivamente con los dedos- tuvo un éxito atronador, incluso antes de que llegase a comercializarse. El Multi-Touch es solo una de las razones por las que el iPhone y el iPad son tan populares, y será la protagonista de este capítulo siete.

Algunos de vosotros os estaréis preguntado, "Bonita lección de historia – pero ¿A dónde quieres llegar?" Básicamente es bueno que estemos entusiasmados con nuestro trabajo y con lo interesante del mismo: El hardware para el que estamos creando nuestras aplicaciones es mágico y radical, y quiero aprovechar vuestra ilusión y asombro para cumplir con mi cometido, que no es otro que el de ayudarte a transformar la creatividad en algo nuevo y asombroso-algo que los usuarios no hayan imaginado todavía.

El primer paso a seguir de este desafío creativo es el de interpretar los imputs del usuario y decodificar sus gestos. Por supuesto, para hacerlo con elegancia, necesitas comprender como funcionan los imputs del Multi-Touch. En este capítulo instruiremos a los procesadores del iPhone e iPad, sobre qué hacer cuando el usuario presione, toque o deslice sus dedos por la pantalla. Sólo una vez comprendidas las convenciones de los distintos imputs, podremos comentar a hablar en el lenguaje basado en el tacto de nuestros usuarios.

ArrastraRotaYAmplía (DragRotateAndScale)—una Aplicación View-Based

Como mencioné en el Capítulo 3, los ejercicios de los capítulos 7-9 están diseñados para continuar desarrollando lo aprendido en Capítulos anteriores. Al igual que en estos Capítulos anteriores, he preparado videos adicionales para estos ejercicios. En todo caso, podrás programar todas las aplicaciones sin tener que hacer uso de estos screencasts.

Sin embargo, y dado que las aplicaciones que vamos a ver en estos tres Capítulos siguientes son considerablemente más complicadas que los ejemplos anteriores, quizás quieras repasar algunos de estos vídeos para conseguir algún enfoque extra. Si decides apoyarte en este material, me versa a mí o a un profesor asistente, entrando en detalles en estos videos tan Buenos. No dudes en darle a la pausa o rebobinar las veces que necesites para sentirte cómodo con el desarrollo del vídeo.

Otra de las cosas a tener en cuenta es que en estos screencasts utilizamos un montón de código reutilizable. Simplemente pegamos componentes ya realizados directamente en aplicaciones en curso. Por el contrario, aquí, en el libro, iremos línea a línea a través del código. Por tanto, repasa los screencast… y luego guárdate gran parte del tiempo a emplear para seguir la elaboración de la aplicación, paso a paso.

Cuando estés preparado, empezamos. Los screencasts podrás encontrarlos en:

```
http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/009_
Drag%20and%20 Rotate.htm
```

Preliminares

Como hiciste en el Capítulo 6, por favor, descarga y extrae las imágenes y código para este capítulo. Si necesitas que recordar como hacer esto, repasa las Fotos 6-2 a la 6-4. Como siempre, empezamos con un escritorio despejado. Abrimos nuestro navegador y cargamos la siguiente dirección:

```
http://rorylewis.com/xCode/009_DragRotateAndScale.zip
```

y descargamos los contenidos en nuestros escritorio. Una vez descargados extraemos los archivos en nuestro escritorio.

Encontrarás cuatro archivos de texto, uno de imagen y una carpeta que contiene el código final de trabajo para DragRotateAndScale (en el caso de que te encuentres con problemas de codificación). El archivo de imagen es el archivo que utilicé con anterioridad de mi perro Shaka. Los archivos de texto consisten en secciones de código reutilizable el cual te pedí que copiases y pegases en otras ocasiones. Puedes utilizar estos archivos en alguno de tus futuros programas que utilicen aspectos de tacto. Encontrarás los archivos Translate.rtf, HelperFunctions.rtf, TranslateRotateScale.rtf, y ViewController.rtf.

Una vez hayas extraído todas las rilas, recuerda borrar DagRotateAndScale.zip y las carpetas 009_DragRotateAndScale. Guarda DragRotateAndScale Xcode en un lugar seguro, para evitar conflictos con el código a utilizar en este ejercicio o problemas de sobreescritura. Los monos van a empezar a escribir obras de Shakespeare, y el mundo se colapsará y desaparecerá … todo por culpa de no haberlo guardado. Después de haber seguido estas instrucciones, tendrás cinco archivos en el Escritorio.

Comenzando con la Aplicación DragRotateAndScale

Para empezar con nuestra aplicación DragRotateAndScale, necesitamos crear un Nuevo proyecto en Xcode tal y como siempre hemos hecho. Abre Xcode y selecciona el Template de Aplicación (tal y como se muestra en la Foto 7–1).

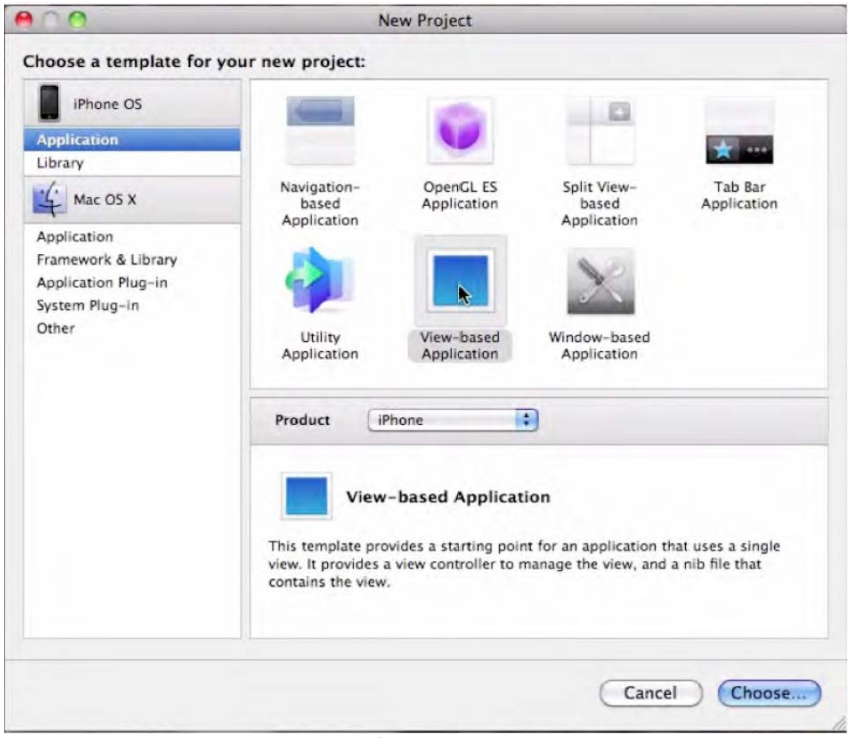


Foto 7–1. Crea un Nuevo Template de Aplicación View-based para comenzar con el proyecto DragRotateAndScale.

Asegúrate que el menú Product está ajustado para iPhone antes de continuar. Nombra el proyecto como DragRotateAndScale y pulas el botón Save, como se muestra en la Foto 7–2.

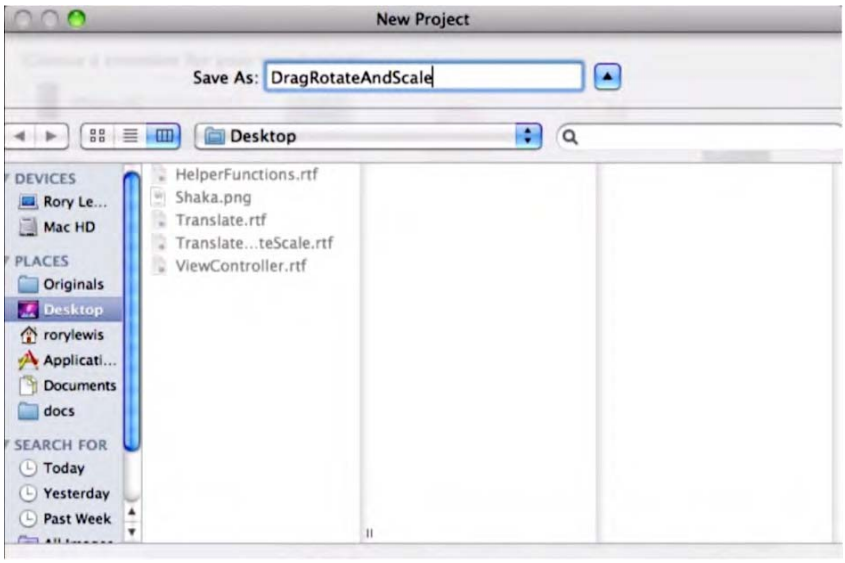


Foto 7–2. Nombra y guarda el proyecto.

Acto seguido, escoge y prepara un archivo de imagen como objeto principal para nuestro ejercicio. Por cierto, hemos escogido una imagen más pequeña que en otros ejercicios (100 x 100 píxeles), con objeto de poder manipularla y cambiar su tamaño..

Copia el archive de imagen en la carpeta de l proyecto, tal y como se aprecia en la Foto 7–3. Clica los cuadros de opción habituales para asegurar la gestión adecuada de este archivo de imagen en nuestro proceso.

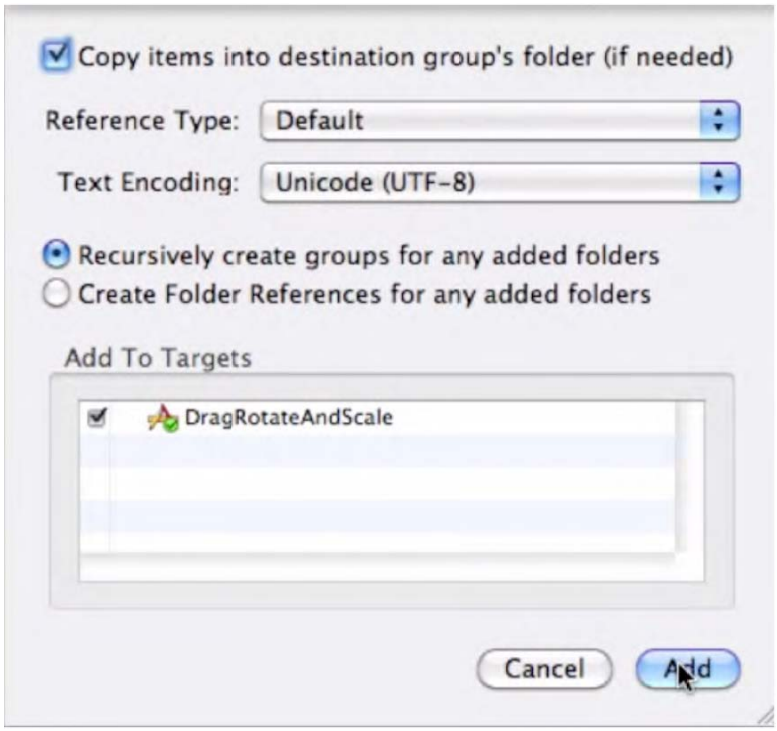


Foto 7–3. Copia la imagen deseada en nuestro proyecto Xcode. La imagen usada en el ejemplo es de 100 × 100 píxeles.

Creando una Subclase UIImageView Personalizada

Vamos a añadir un Nuevo archive a nuestro proyecto: una Subclase UIImageView llamada TransformView. Esta subclase interceptará los posibles contactos en la pantalla y aplicará las consiguientes transformaciones. TransformView es una Subclase de UIImageView de modo que podremos asignarle una imagen a nuestro ejemplo y luego verlo representado en nuestra pantalla. Técnicamente, una Subclase UIView puede hacer lo que queramos, pero para nuestro ejemplo, queremos centrarnos en las transformaciones. No te preocupes acerca de cómo generar el código para una Sublcase de UIView.

Crear una vista personalizada no es estrictamente necesario para esta aplicación, aunque nosotros lo vamos a hacer de todos modos para tonificar nuestra "musculatura de subclases. El DragRotateAndScaleViewController técnicamente podría hace todo lo que necesitamos para que funcione esta aplicación sin necesidad alguna de una Subclase UIView, pero el uso de subclases hace que el código generado sea más simple y más robusto. Por ello, seleccionaremos una nueva clase Objective-C class en la ventana New File (ver Foto 7–4.)

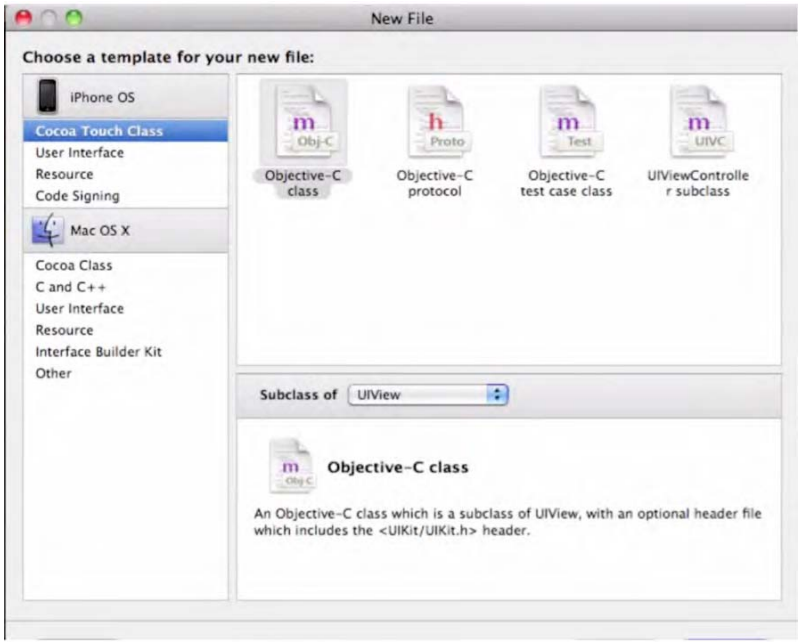


Foto 7–4. Crear un nuevo archivo Clase Objective-C.

En el Archivo de Cabecera (header file), `TransformView.h`, asegúrate que la Superclase –esto es, lo marcado después de los dos puntos (:)- está especificado como `UIImageView`. Lo hacemos así para que se nos permita asignar una imagen de modo rápido y fácil. Tu header file deberá quedar como aparece en la Foto 7–5.

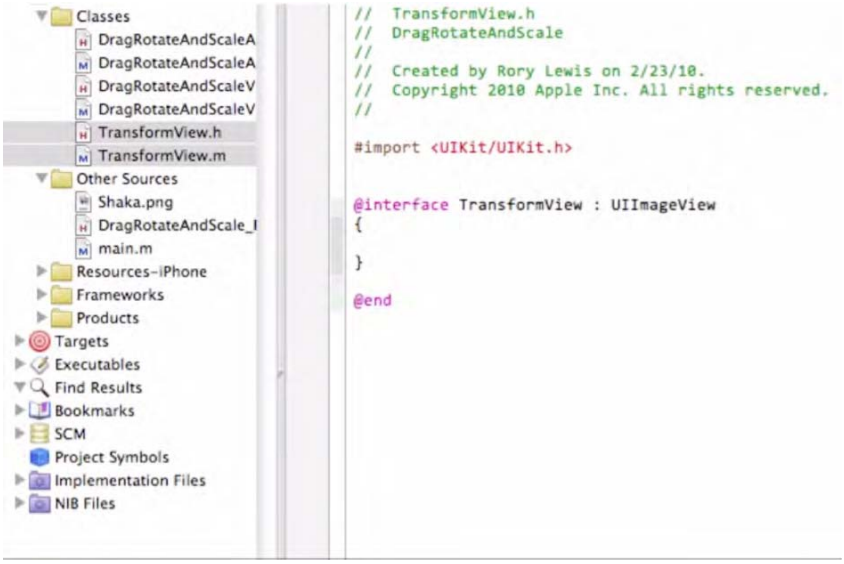


Foto 7–5. La Clase TransformView, una subclase de UIImageView

```
//
// TransformView.h
#import <UIKit/UIKit.h>
@interface TransformView : UIImageView
{
}
@end
```

Reemplazando –initWithImage en TransformView.

Para el Archivo de Implementación (implementation file), `TransformView.m`, queremos controlar la creación de nuestro modo de vista de manera diferente a la normal. ULLmageViews no controla los imputs táctiles por defecto, y por tanto, rechaza cualquier entrada táctil que reciben. Esto no nos sirve! Nosotros queremos controlar tanto la interacción táctil simple como las múltiples.

Para hacer esto, reemplazamos el método

`(id) initWithImage:(UIImage*)image`.

En este reemplazo insertaremos las líneas

`[self setUserInteractionEnabled:YES]`

y

`[self setMultipleTouchEnabled:YES]`.

No hace falta que te recuerde que terminaremos cada línea cerrándola con un punto y coma.

La primera línea permitirá a nuestra vista el responder a las acciones táctiles. La segunda posibilitará a TransformView el recibir múltiples acciones táctiles, que es lo que necesitamos si queremos permitir que el usuario pueda pellizcar y cambiar de tamaño y rotar. El código debería quedar como en la Foto 7–6.

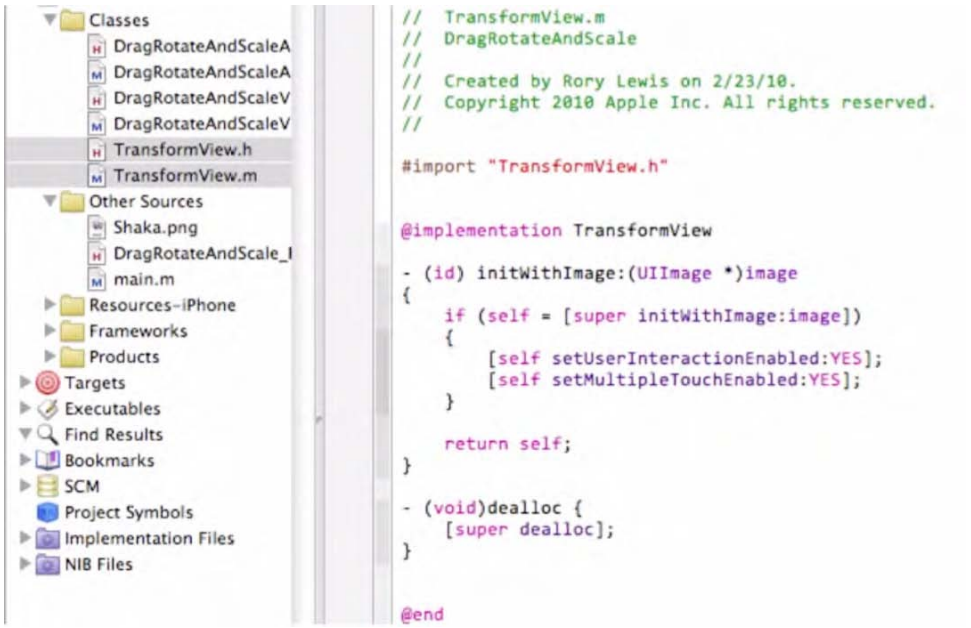


Foto 7–6. Habilita TransformView para poder permitir y procesar múltiples acciones táctiles.

```
//
// TransformView.m

#import "TransformView.h"

@implementation TransformView

-(id) initWithImage:(UIImage *)image
{
       if (self = [super initWithImage:image])
       {
              [self setUserInteractionEnabled:YES];
              [self setMultipleTouchEnabled:YES];
       }
       return self; }
@end
```

Creando Touch-Handling Stubs

Ahora vamos empezar el proceso de creación de porciones de código que definan el cómo afectan a las imagines estas acciones táctiles y otros parámetros de la interfaz. Para hacer esto, anularemos varios métodos cuyos nombres empiezan literalmente con el término “touches.” (toques). Estos métodos de “toques” entran en acción cada vez que el usuario toca con el dedo la pantalla del dispositivo, arrastra un dedo, o deja de tocar la pantalla. Los cuatros métodos que vamos a añadir son los siguientes:

```
-(void) touchesBegan:(NSSet*)touches withEvent:(UIEvent*)event
-(void) touchesMoved:(NSSet*)touches withEvent:(UIEvent*)event
-(void) touchesEnded:(NSSet*)touches withEvent:(UIEvent*)event
-(void) touchesCancelled:(NSSet*)touches withEvent:(UIEvent*)event
```

NOTA: Estos métodos aparecen con el nombre de stubs en la Foto 7-7. Un stub es un esqueleto de un código que puede ser fácilmente adaptado a nuestro proyecto para ahorrar tiempo. En nuestro caso, estos cuatro stubs suponen una programación eficiente cuando se produce contacto con la pantalla, hay movimiento, el movimiento cesa o el mismo es cancelado.

Advierte que `touchesCancelled:withEvent:` llama o hace referencia a `–touchesEnded:withEvent:`. Esto es debido a que normalmente queremos que un toque de *cancelación* se comporte como señal de que el usuario ha terminado una operación táctil. “Cancelled” es lo que llamamos “ninguna entrada del usuario” (en este caso un toque) que es apagado mediante idéntico imput; por ejemplo, un botón que se acciona (posición ON), pero después de pulsado se vuelve a pulsar para pasarlo a apagado (OFF). Un toque “ended” es un imput intencional que lo que busca es el cese, o paro de un toque o interacción en curso, el cual se realiza después de un determinado momento. Esta distinción no siempre será de aplicación, pero es supone una buena aproximación para nuestros propósitos.

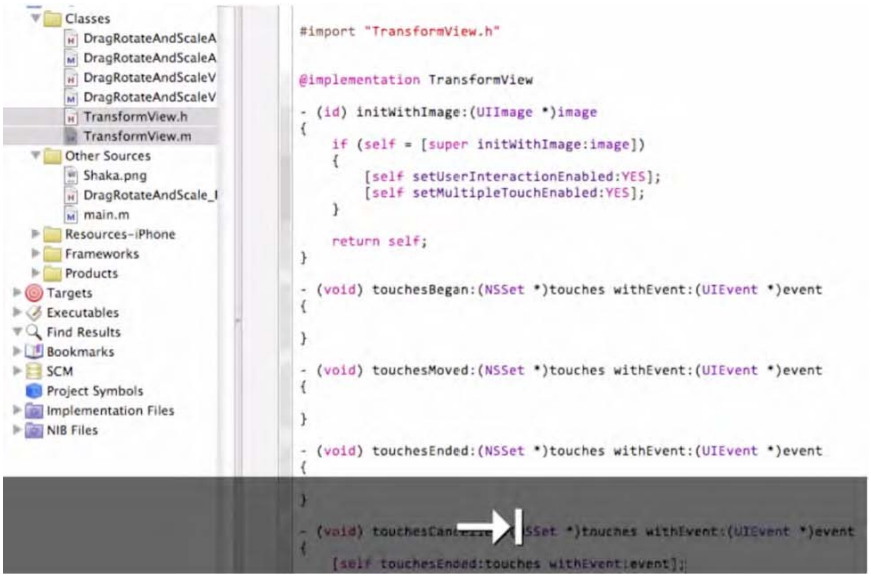


Foto 7–7. Las líneas de código stub insertadas.

No es nuestra intención el que comprendas cada una de las palabras y expresiones el código “touches”. Para nosotros es suficiente que sepas que cada vez que trates con código táctil y estés intentando averiguar qué parte del código por defecto mantener y qué parte eliminar, puedas despreocuparte y usar estas líneas de código preparado y adaptado, conocido como “stubs”. No le des más vueltas al tema: simplemente úsalo y despreocúpate!.

```
//
// TransformView.m

#import "TransformView.h"

@implementation TransformView

-(id) initWithImage:(UIImage *)image

{
        if (self = [super initWithImage:image])
        {
                [self setUserInteractionEnabled:YES];
                [self setMultipleTouchEnabled:YES];
        }

        return self;
}

-(void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
}
-(void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
}
-(void) touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
}
-(void) touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
{
        [self touchesEnded:touches withEvent:event]; }
@end
```

Desplazamientos en *touchesMoved*

Empecemos con la sección de nuestro código que nos permitirá desplazar una imagen- en otras palabras, el código que nos permitirá arrastrar objetos. Esto significa que, por ahora, vamos a centrarnos en una única línea de código: `-touchesMoved:withEvent:`.

Por el momento, los otros métodos táctiles se mantendrán sin cambios. Nuestro touchesMoved debería quedar cono en la Foto 7–8.

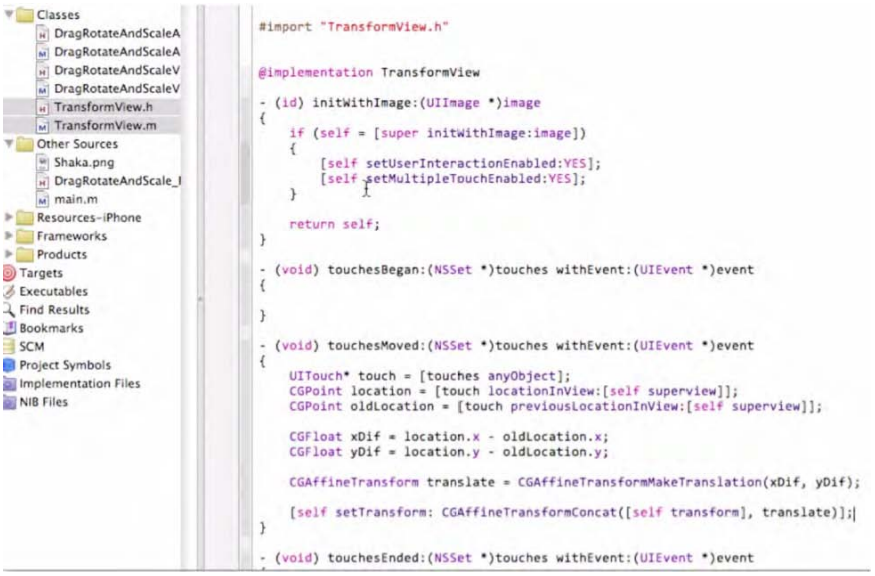


Foto 7–8. Con este código incluido, el usuario puede ahora mover la imagen con un dedo.

```
-(void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
              CGPoint newTouch = [[touches anyObject] locationInView:[self~CCC
superview]];
              CGPoint lastTouch = [[touches anyObject]
previousLocationInView:[self~CCC
superview]];


             float xDif = newTouch.x - lastTouch.x;
             float yDif = newTouch.y - lastTouch.y;

            CGAffineTransform translate = CGAffineTransformMakeTranslation(xDif,~CCC
yDif);
              [self setTransform: CGAffineTransformConcat([self transform],~CCC
 translate)];
}
```

Echemos un vistazo a este código para intentar determinar que está pasando realmente.

Lo primero de todo, qué significa el comando CGPoint? CGPoint es un núcleo de código el cual ha sido programado por la gente de Apple y diseñado para nuestro uso; es la parte del programa en el módulo CoreGraphics que escribe un punto una localización en el espacio 2D, mediante las coordenadas **x** e **y**. Las pulsaciones del usuario se representan en CGPoints cuando su localización es apropiada y solicitada por la naturaleza de la aplicación.

Esta información puede medir la distancia entre dos untos usando matemática elemental. Echemos un vistazo a la llamada `[[touches anyObject] locationInView:[self superview]]`. Este código agarra la pulsación sobre un objeto desde el NSSet y solicita su ubicación en el superview del objeto. En otras palabras, simplemente le estamos preguntando por la ubicación de la pulsación emitida por el usuario en reilación a la superview.

Esto es diferente de lo que uno debería esperar. Por qué nos estamos preguntando acerca de la posición en el superview y no acerca de su propia posición en el TransformView? Pues porque queremos saber dónde mover el TransformView en la superview. Por tanto, tenemos la posición actual y previa de la pulsación en el superview.

NOTA: El IPad y el iPhone realizan un seguimiento sobre qué vista está siendo actualmente mostrada tratándola como una ventana de ejemplo. Estos ejemplos son ordenados de modo piramidal. En el nivel más alto tenemos el content view, (ventana de contenidos) la cual es la raíz de otra serie de vistas, llamadas *subviews*. La ventana superior de la vista que en un momento dado se esté utilizando recibe el nombre de *superview*.

Las dos líneas siguientes trabajan conjuntamente para calcular la diferencia entre la posición antigua y la nueva posición dentro del eje de coordenadas X e Y.

La siguiente línea crea una traducción , guardando esta traslación como una

Transformación temporal *CGAffineTransform*. Realmente es una matriz que guarda los cambios de posición para una vista determinada. Dado que la matriz de "traslación" es relativa, tenemos que añadirla a nuestra transformación actual. Esto lo hacemos en la última línea de código en negrita, concatenando (juntando o combinando) la transformación de vista "actual" y la transformación del "traslado" para conseguir una transformación que recoja nuestra *nueva* posición. Una vez tengamos esa nueva transformación, ajustamos la transformación de vista a esta nueva transformación.

Haciendo Uso de *TransformView*

Nuestra TransformView está preparada para su primer test. Necesitamos hacer un ejemplo añadiendo una subview a algo. Esta transformación vamos a hacerla en DragRotateAndScaleViewController. Para ello, entramos en el Archivo de Cabecera (header file) DragRotateAndScaleViewController. Todo lo que tenemos que hacer es importar nuestro archivo `TransformView.h` como puedes apreciar en la Foto 7–9.

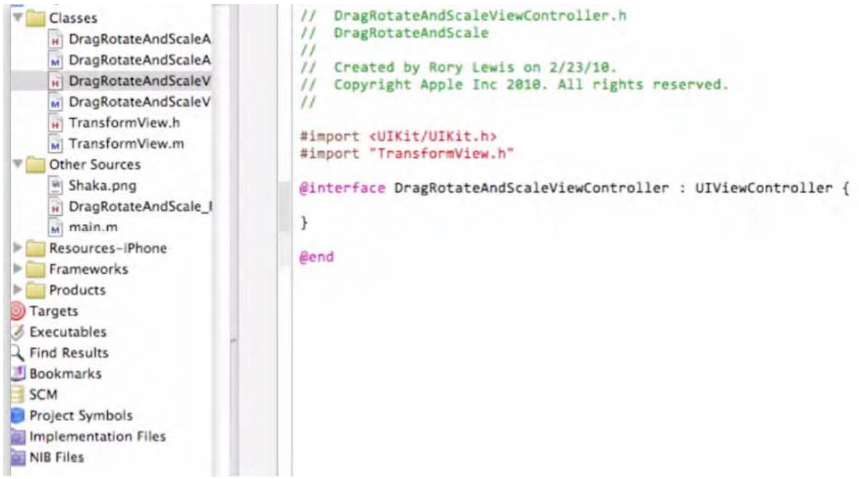


Foto 7–9. Importa el Header TransformView.h.

```
//
// DragRotateAndScaleViewController.h

#import <UIKit/UIKit.h>
#import "TransformView.h"

@interface DragRotateAndScaleViewController : UIViewController {

}

@end
```

Creando una TransformView

En el Archivo de Implementación (implementation) *DragRotateAndScaleViewController*, queremos crear un `TransformView` y que este sea visible para el usuario. Queremos asegurarnos de que la vista está preparada, por lo que realizamos esto en el método de sustitución `viewDidLoad`. El código resultante lo tienes en la Foto 7–10.

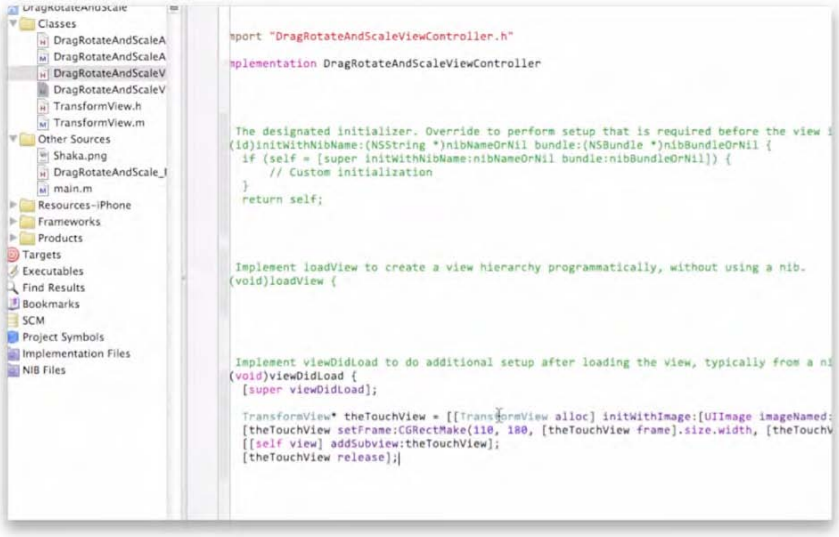


Foto 7–10. Ahora podemos ver y mover nuestro TransformView. Casi hemos terminado!

```
-(void)viewDidLoad
{
    [super viewDidLoad];

        TransformView* theTouchView = [[TransformView alloc] itWithImage:[UIImage~CCC
imageNamed:@"Shaka.png"]];
        [theTouchView setFrame:CGRectMake(110, 180, [theTouchView frame].size.width,~CCC
[theTouchView frame].size.height)];
        [[self view] addSubview:theTouchView];
        [theTouchView release];
}
```

En la primer línea en negrita, creamos un nuevo objeto TransformView object, facilitándole un objeto UIImage con el nombre de la imagen que arrastramos al principio del proceso. Esto llamará al método de sustitución que detallamos con anterioridad, permitiendo que TransformView tome las entradas táctiles.

A continuación, establecemos TransformView a fin de posicionarlo inicialmente dentro de la vista en pantalla. Los números se han obtenido de las dimensiones del iPhone y de las dimensiones de nuestra imagen, 100 × 100 píxeles.

La tercera línea de código añade TransformView como una subvista de la misma vista, de modo que TransformView será mostrado.

La última línea envía una llamada de comunicado—cuya finalidad es la gestión de memoria.

Con esto debería ser suficiente! Llegado a este punto deberíamos poder ejecutar nuestro código y mover por la pantalla TransformView, simplemente tocando y arrastrando. La Foto 7–11 muestra el primer paso en el test de TransformView, el cual es la selección de la plataforma adecuada en la cual simular la aplicación.

Foto 7–11. Asegúrate que el ejecutable está seleccionado para el iPhone Simulator.

Foto 7–12 Muestra el iPhone Simulator funcionando con nuestra imagen cargada. Debemos ser capaces de mover esta imagen, por tanto, pulsa y arrastra tu foto. Sí, es sólo un test, pero Roma no se construyó en un solo día.

Foto 7–12. Pulsa y arrastra tu imagen por la pantalla!

Después de haber controlado la imagen las yemas de los dedos en nuestra simulación, vamos a probar con la otra plataforma. La Foto 7–13 muestra el marco donde cambiar el ejecutable para seleccionar el Simulador iPad. Necesitamos que TransformView funcione igual de bien en el iPad que en el iPhone.

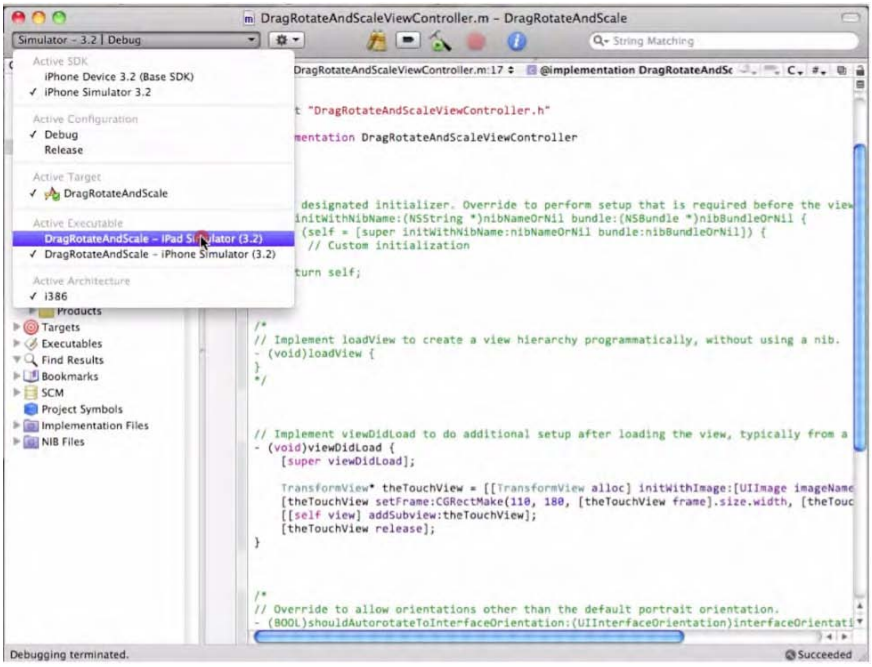


Foto 7–13. Probemos nuestro código en el iPad Simulator.

La Foto 7–14 muestra el iPad Simulator en modo Vista Normal, donde la imagen se expande ocupando una parte proporcional de la pantalla. Tócala y arrástrala para probar la fluidez de la plataforma iPad..

Dado que disponemos de una Vista iPhone dentro del Simulador iPad, vamos a probar para cerciorarnos de que funciona correctamente. La Foto 7–15 muestra esta opción, representando fielmente las dimensiones relativas de la misma imagen en estos dos modos de vista.

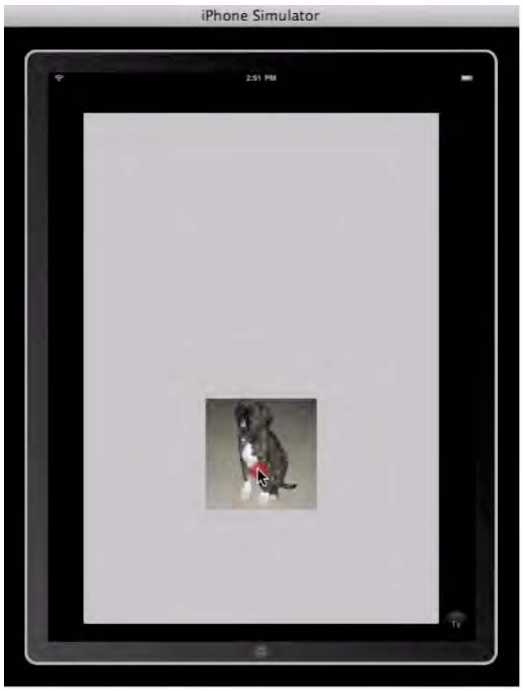


Foto 7–14. Verás tu imagen moviéndose en la Vista Normal del iPad.

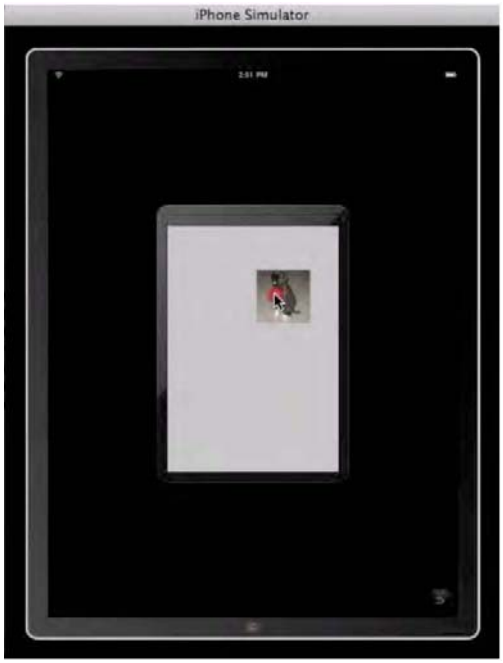


Foto 7–15. En modo Vista iPhone, comprobamos que la imagen aparece en su tamaño normal.

Preparando *TransformView* para Rotación y Escalado

Excelente- estamos en racha! Sigamos con el ejercicio.

Tu aplicación se muestra muy bien hasta ahora, pero se puede mejorar permitiendo al usuario hacer rotación y zoom de la imagen. Esto requiere una computación algo más compleja y monitorización de las interacciones táctiles. Como ya sabes, necesitamos recoger dos pulsaciones simultáneas y determinar sus posiciones. Para lograr esto, necesitamos modificar nuestro Archivo de Cabecera (header file) TransformView.

En el archivo `TransformView.h`, vamos a añadir dos campos UITouch*: `firstTouch` y `secondTouch`**.** Estos dos objetos rastrearán la distancia y ángulo entre los puntos de contacto. Además, añadiremos prototipos de metodología para los métodos de ayuda que usaremos para calcular los cambios de transformación, tal y como puede ver en la Foto 7–16.

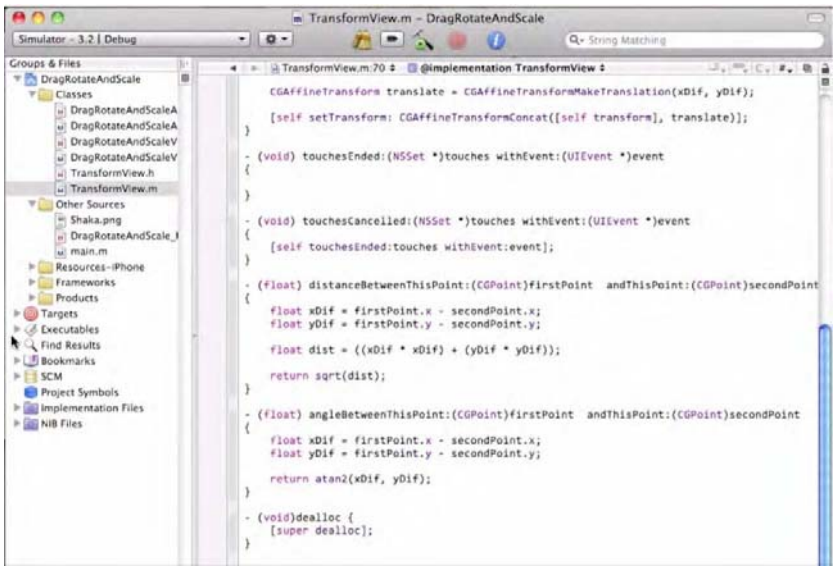


Foto 7–16. Modifica el archivo de implementación TransformView.m para recoger las dos pulsaciones.

```
//
// TransformView.h

#import <UIKit/UIKit.h>

@interface TransformView : UIImageView
{
     UITouch* firstTouch;
     UITouch* secondTouch;
}

-(float) angleBetweenThisPoint:(CGPoint)firstPoint ~CCC
andThisPoint:(CGPoint)secondPoint;
-(float) distanceBetweenThisPoint:(CGPoint)firstPoint andThisPoint:~CCC
(CGPoint)secondPoint;

@end
```

Declaramos las variables de ejemplo para la clase: *dos objetos UITouch* que serán utilizados para recoger los imputs del usuario. Al final, verás los prototipos para los métodos de ayuda que usaremos para el cambio y transformación de la imagen.

Te estarás preguntando, "Por qué molestarnos con pasar CGPoints a las funciones de ayuda? Por qué no usar en su lugar UITouch*" Pues debido a que con posterioridad podemos decidir cambiar el modo en que se manejan estos imputs, desactivándolas, o cambiando las posiciones de contacto desde sus posiciones reales (por cualquier razón que se presente). Para ello necesitaríamos cambiar nuestro código auxiliario, medida que no es la ideal. En cambio, el código auxiliar siempre debe funcionar de las misma manera, mientras que las llamadas al código cambien los imputs, en caso sea necesario.

Métodos de Ayuda

Ahora que tenemos nuestras variables de ejemplo y prototipos de método, ya podemos trabajar sobre el Archivo de Implementación (Implementation File). Estos métodos de ayuda podían haber sido incluidos en interfaz privado, pero esto habría sido demasiado para este tipo de aplicación

Dentro del archivo `TransformView.m`, crea los métodos de ayuda, tal y como se muestra en la Foto 7–17.

Foto 7–17. Trae tus métodos de ayuda "helper methods" desde el archive descargado helperFunctions.rtf.

```
-(float) distanceBetweenThisPoint:(CGPoint)firstPoint andThisPoint:↵
(CGPoint)secondPoint
{
     float xDif = firstPoint.x - secondPoint.x;
     float yDif = firstPoint.y - secondPoint.y;

     float dist = ((xDif * xDif) + (yDif * yDif));

     return sqrt(dist);

}
-(float) angleBetweenThisPoint:(CGPoint)firstPoint↵ andThisPoint:(CGPoint)secondPoint
{
     float xDif = firstPoint.x - secondPoint.x;
     float yDif = firstPoint.y - secondPoint.y;

     return atan2(xDif, yDif);
}
```

Afortunadamente estos "helper methods" o métodos de ayuda son relativamente sencillos. Simplemente calculan la distancia y el ángulo en radianes entre dos interacciones táctiles. Esto será usado por nuestros métodos de pulsado para crear la rotación y el escalado el cual será aplicado al TransformView.

El `-distanceBetweenThisPoint:[*]andThisPoint`: busca las diferencias de posición de los ejes **x** e **y** entre dos puntos y utiliza el Teorema de Pitágoras para calcular la distancia en línea recta entre los dos puntos.

Análogamente, `-angleBetweenThisPoint:[*]andThisPoint`: busca el ángulo desde el primer punto al segundo punto en relación al eje de coordenadas **x**, devolviendo el resultado en radianes.

Añadiendo a "-touchesBegan"

Empezaremos por la parte más sencilla de la siguiente sección de código. Necesitarás añadir código a `touchesBegan:withEvent:` con objeto de registrar las interacciones táctiles. Esto puede parecer algo bastante imponente, pero estamos empezando a trabajar en un nivel bastante avanzado, y para algunos de vosotros, esto será un juego de niños.

Seguimos en el Archivo de Implementación (Implementation File) TransformView.m, y vamos a rescribir nuestros métodos de pulsación para manejar y permitir múltiples pulsaciones y para utilizar funciones auxiliares con objeto de cambiar la transformación.

```
-(void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
     //Single touch
     if ([touches count] == 1)
     {
         if (!firstTouch)
         {
                 firstTouch = [[touches anyObject] retain];
         }
         else if (!secondTouch)
         {
                secondTouch = [[touches anyObject] retain];
         }
     }
     //Multiple touch
     if ([touches count] == 2)
     {
     NSArray* theTouches = [touches allObjects];
     [firstTouch release];
     [secondTouch release];
     firstTouch = nil;
     secondTouch = nil;
     firstTouch = [[theTouches objectAtIndex:0] retain];
     secondTouch = [[theTouches objectAtIndex:1] retain];
    }
}
```

Bien… echa un vistazo y comprueba si sabes lo que estamos hacienda. Primero comprobamos si ha habido un único toque comprobando el número de interacciones táctiles. Si ha habido un único toque, direccionamos el equipo a esto –en memoria- para su uso posterior, en cualquier campo que no esté actualmente en uso.

Si hay dos toques, tomamos nota de todos los toques, se libera el registro de todos los toques anteriores, y se asignan los campos para estos dos primeros toques en la selección de toques. Este enfoque nos da los dos objetos que necesitamos e ignoramos cualquier otro toque extraño. Fácil, verdad?

Copia este código del archivo TranslateRotateScale.rtf descargado al principio del capítulo, el cual guardaste en su escritorio, tal y como se muestra en la Foto 7–18. Una vez copiado, pega esas líneas en el Archivo de Implementación (Implementation File).

Foto 7–18. Coge el nuevo código táctil del archivo TranslateRotateScale.rtf descargado.

Modificando *-touchesMoved*

Toma aire y relájate. El siguiente fragmento de código puede parecer intimidante, pero en realidad es bastante simple. Volveremos a él con posterioridad u verás exactamente qué está pasando y cómo funciona.

Este código modifica `-touchesMoved:[*]withEvent:` method, y también utiliza nuestros métodos auxiliares. Usamos los datos de las interacciones táctiles y los métodos auxiliares para crear transformaciones que están concatenadas con la transformación presente.

```
-(void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
     if ([touches count] == 1)
     {
      CGPoint newTouch = [[touches anyObject] locationInView:[self superview]];
      CGPoint lastTouch = [[touches anyObject] previousLocationInView:
      [self superview]];

      float xDif = newTouch.x - lastTouch.x;
      float yDif = newTouch.y - lastTouch.y;

      CGAffineTransform translate = CGAffineTransformMakeTranslation(xDif, yDif);
      [self setTransform: CGAffineTransformConcat([self transform], translate)];
      }
      else if ([touches count] == 2 && firstTouch && secondTouch)
      {
      //Rotate
      float newAngle = [self angleBetweenThisPoint:[firstTouch locationInView:
      [self superview]]

      andThisPoint:[secondTouch locationInView:[self superview]]];
      float oldAngle = [self angleBetweenThisPoint:
      [firstTouch previousLocationInView:[self superview]]

      andThisPoint:[secondTouch previousLocationInView:[self superview]]];

      CGAffineTransform rotation = CGAffineTransformMakeRotation(oldAngle - newAngle);

      [self setTransform: CGAffineTransformConcat([self transform], rotation)];

      //Scale
      float newDistance = [self distanceBetweenThisPoint:
      [firstTouch locationInView:[self superview]]

      andThisPoint:[secondTouch locationInView:[self superview]]];
      float oldDistance = [self distanceBetweenThisPoint:
      [firstTouch previousLocationInView:[self superview]]
      andThisPoint:[secondTouch previousLocationInView:[self superview]]];

      float ratio = newDistance / oldDistance;

      CGAffineTransform scale = CGAffineTransformMakeScale(ratio, ratio);
      [self setTransform: CGAffineTransformConcat([self transform], scale)];
      }
}
```

En primer lugar, direccionamos el computador para comprobar si solo se movió un toque. En ese caso, llamamos al mismo código que teníamos antes. No hay nada más fácil que esto!

Si hay dos toques, empezaremos a calcular las transformaciones de rotación y escalado. El cálculo de rotación comienza usando nuestro método auxiliar para obtener el ángulo entre los puntos de contacto actuales. Posteriormente se calcula el ángulo entre los dos puntos de contacto iniciales. La rotación se efectúa encontrando la diferencia entre estos dos ángulos. Esto genera una transformación relativa de rotación que, al igual que antes, necesita ser concatenada con la transformación posterior.

A continuación, calculamos el escalado apropiado que se va a aplicar basado en la posición de los contactos. Usamos nuestros métodos auxiliares para calcular la distancia entre la posición de los dedos presente y la inicial, y de este modo se obtiene el ratio entre la distancia inicial y la nueva. Esto nos da un factor de escala con el cual podremos aplicar el zoom a la imagen. Usando este factor, podemos crear este escalado y posteriormente concatenarlo con la siguiente transformación

Lo ves? Te dije que no era tan complicado! Puedes enseñar este código a tus amigos y ver cómo se quedan embobamos mientras tú les muestras tu habitual sonrisa.

Al ejecutar el código, se nos permite arrastrar la imagen con un dedo o rotarla y cambiarla de tamaño si la pellizcamos, tal y como vemos en la Foto 7–19 y Foto 7–20.

Foto 7–19. Arrastra,rota y escala tu imagen

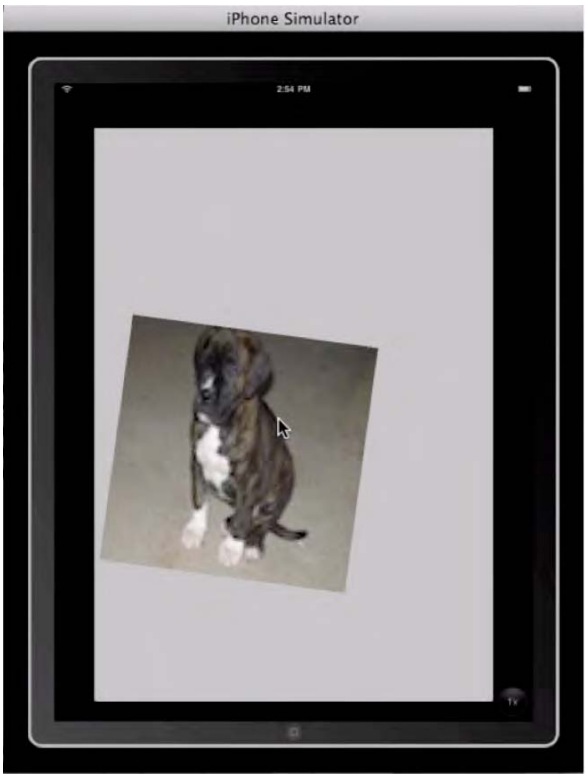


Foto 7–20. Funciona perfectamente en el iPad!

NOTA: Con el código utilizado, un usuario puede hacer una imagen tan pequeña que posteriormente no pueda ser capaz de volver a utilizar los dedos para devolverla a un tamaño más grande. Para evitar esto, debería tomarse en consideración la aplicación de límites o restricciones al escalado. Por la misma razón el usuario podría mover una imagen y sacarla completamente de la pantalla. Esto también puede evitarse aplicando restricciones posicionales de imagen.

Hemos terminado. Comprueba la rotabilidad, arrastrabilidad y capacidad de cambio de tamaña de la subclase!

¿Cuándo nos será de utilidad todo esto? ¿Cuándo debe de ser utilizado y qué cosas hemos de tener en cuenta? Como ya comentamos anteriormente, esta es una interfaz muy intuitiva y amigable, incluso para los usuarios que son nuevos en el mundo iPhone / IPAD. Estos conocimientos deben ser considerados y aplicados en cualquier aplicación que tenga objetos que tengan variedad de detalles y se beneficien de la interacción directa del usuario.

Profundizando en el Código

Vamos a centrarnos en uno de los conceptos que mencionamos con anterioridad sólo de pasada: La Manipulación de Eventos (event-handling). Las cuatro líneas de código que tenemos a continuación tienen que ver con los mencionados eventos y con los métodos por los cuales deseamos que el computador haga frente a esos eventos.

```
-(void) touchesBegan:(NSSet*)touches withEvent:(UIEvent*)event
-(void) touchesMoved:(NSSet*)touches withEvent:(UIEvent*)event
-(void) touchesEnded:(NSSet*)touches withEvent:(UIEvent*)event
-(void) touchesCancelled:(NSSet*)touches withEvent:(UIEvent*)event
```

Con objeto de adentrarnos en este código, primero necesitamos recordar qué métodos y argumentos son. Estos cuatro eventos son llamados “event-handling methods”.

Explicar un método “event-handling” es algo complicado, ya que los mismos son una herramienta bastante abstracta- Es como hacer uso del túnel del tiempo para asesorar a Thomas Jefferson en la elaboración de la Declaración de Independencia a través de una computadora. Así que antes de hablar acerca de estos cuatro métodos “event-handling”, vamos a echar un vistazo a cómo funciona un evento genérico”event-handling”.

Imagina que tu teléfono está sonando. Sabes que alguien te está llamando. Puedes actuar de las siguientes maneras:

A) Descolgar y decir, “¿Hola?”

B) Descolgar y decir “Disculpa, te llamo más tarde.”

o …

C) Dejar que el teléfono suene y salte el contestador

Estas tres opciones son formas de actuar en una determinada situación: son métodos “event-handling”. Tienes distintas opciones a la hora de desenvolverte con un teléfono, y el escoger una u otra depende de varios factores: Quién te está llamando en ese momento, qué estás hacienda en ese momento, lo cansado o no que estás, el hambre que tienes, y múltiples situaciones más.

Manteniendo esta analogía, las líneas que estamos analizando son métodos “event-handling” para gestionar un contacto táctil de un usuario en una aplicación iPhone o iPad. Los programadores de Apple han creado métodos “event-handling” que hacen más sencillo el decidir si vas a contestar al teléfono, responder, y así sucesivamente.

Sin embargo, en nuestro caso, queremos gestionar cuatro diferentes tipos de eventos. Estos eventos de contacto táctil son eventos que se propagan a través de la cadena de respuesta, pero ¿Qué es una cadena de respuesta? Aquí tienes una analogía.

El teléfono suena, pero no tienes ganas de contestar. Le pides a alguien (tu hermano, hermana o madre) que lo coja por ti. El cómo gestione la llamada la persona que se preste a ello es algo que depende exclusivamente de ellos mismos, ya que tú les has

dado la oportunidad de responder. Así es cómo funciona la cadena de respuesta: se presentan situaciones y los objetos pueden manipular estas situaciones (atender tú mismo el teléfono) o dejar pasar esa situación al siguiente en la cadena de respuesta (decirle a alguien que lo atienda por ti).

En esta misma línea podrás encontrarte el termino "primer respondedor" (first responder) en el Interface Builder o en escrito en alguna documentación de Apple. Este es el primer objeto en la cadena de respuesta, y el primer respondedor siempre tiene la primera oportunidad de respuesta con los eventos generados. En el ejemplo del teléfono, tú serías el primer respondedor, al tener la oportunidad de atender el teléfono primero. La mayoría de controles sin objetivos asociados (copo puede ser un botón que no tiene todavía un target asociado) por defecto envían sus acciones al primer respondedor.

Cuando se produce un contacto en el dispositivo, la ventana utiliza el "hit testing" (test de contacto) par determinar qué tipo de contacto se ha producido, pasando la información de evento de contacto a la cadena de respuesta. Si una propiedad `userInteractionEnabled` está establecida como "NO", el evento continuará a través de la cadena de respuesta hasta encontrar una vista que esté habilitada para manejar este evento. Un contacto y su vista asociada están vinculados por la duración del contacto. Esto significa que, incluso si el contacto desplaza la vista generada en el toque inicial, esta vista seguirá asociada a ese contacto: las otras vistas no recibirán información acerca de ese contacto… independientemente del movimiento.

Los eventos generados contienen información de las interacciones táctiles que las han disparado, así como la duración, el tipo de evento y subtipo, siendo posible acceder a toda esta información a través de las correspondientes propiedades.

El Nuevo Sistema Operativo iPhone OS 3.0 tiene eventos de movimientos que se activan de manera similar a los eventos de contacto. Son eventos que dirigidos van dirigidos al por defecto al primer respondedor. El objetivo del evento para un movimiento puede tener su subtipo preparado para UIEventTypeMotionShake, el cual proporciona una manera sencilla de detección de movimientos de agitado. Los tipos y subtipos para un objeto UIEvent proporcionan una gran cantidad de información útil que puede ayudarnos a determinar cómo un evento de entrada puede ser manejado.

Cocoa Touch, un sistema visual del iPhone OS, trabaja de acuerdo con una jerarquía. Las vistas gestionan sus propias representaciones y sus subvistas. Cuando se realiza una llamada del tipo `[[self view] addSubview:aView];` una vista es conformada por una subsista de `esa propia vista`.

Sencillo, verdad? Bien, cada vista tiene una transformación que describe la localización de la misma, rotación, escalado y otros factores en relación con la SuperVista perteneciente a esta vista. Esto es exactamente lo que necesitamos para hacer nuestra vista personalizada de escalado, rotación y movimiento alrededor de esta Supervista cada vez que de detecta e identifica una interacción sobre la pantalla.

Podemos cambiar la transformación de varias maneras, pero para la mayoría de las veces, nos bastará con tratar con distancias y ángulos entre esas interacciones táctiles para hacer todo aquello que necesitemos. La estructura CGAffineTransform es utilizada para almacenar y manipular las transformaciones de vistas. Ahora que estás en el mundo de la programación avanzada, deberías sentirte cómodo usando llamadas C-style para CGAffineTransform. Dedica algo de tiempo para revisar la documentación relativa a CGAffineTransform y echar un vistazo a las guías de vistas de programación para adquirir conocimientos más profundos sobre cómo funcionan todos estos elementos.

Gesture Support and the iPad

El SDK iPad contiene dos comandos nunca vistos con anterioridad (`3Tap.plist` y `LongPress.plist`) los cuales no se encontraban en ningún SDK iPhone hasta la versión 3.1. Que proporcionan los comandos `3Tap` y `LongPress`? Proporcionan exactamente lo que se extrae de sus propios nombres. El iPad reconocerá tres toquecitos rápidos y gestos de larga presión extendidos-diferente del habitualmente usado para copiar y pegar.

Mi predicción es que el primer uso que se le dará al iPad en los próximos años será el de medio para libros de texto; los estudiantes podrán escribir notas en sus ebooks y mostrarlas en sus iPads. Tenga en cuenta que ya se pueden eliminar mensajes y emails simplemente con un toque, con el mismo efecto de pulsar la tecla Borrar. Cuando un estudiante está escribiendo notas en un texto en su iPad, quizás un golpecito será una manera ordenada de eliminar algunos de los comentarios, igual que si usásemos una goma de borrar.

Capítulo 8

Vistas de Tabla, Navegación, y Matrices

Enseñar o no enseñar matrices, esa es la cuestión. La intención original de este libro era la de *no* enseñar las matrices por varias razones:

1. Son bastante difíciles para los expertos de ciencias de computación y más aún para los principiantes
2. Se consideran aburridas.
3. Situados en el contexto de Objective-C se hacen aún más complicadas.

Pero *necesitaba* enseñar matrices. Los estudiantes de mi clase de iPhone / iPad querían hacer el Proyecto de Final de curso con utilización de tablas –las cuales requieren el uso de matrices- y esto hizo darme cuenta de lo importantes que eran.

Para ayudarle a decidir si deberías leer este capítulo, quiero que valores los siguientes puntos:

1. ¿Por qué *no* iba a enseñarte este capítulo?
2. ¿Por qué*, puedes querer*, saltarte este capítulo?
3. ¿Por qué decidí enseñarlo a pesar de todo... y porqué *a mi manera*?

¿Por qué *no* iba a enseñarte este capítulo?

He tenido varias conversaciones con mis colegas sobre lo que ocurre cuando los estudiantes de informática tienen el primer contacto con las matrices. La experiencia nos dice que entender las matrices no va relacionado con ser inteligente. A menudo veo que mis más brillantes estudiantes tropiezan cuando entran en la jungla de las matrices. Por el contrario, veo estudiantes que no tienen nada que hacer andando por las salas del edifico de Ingeniería Informática y comprenden el concepto como si fuera su lengua materna. Uno de los estudiantes que luchó con los capítulos del “Hola mundo”, actualmente domina las matrices.

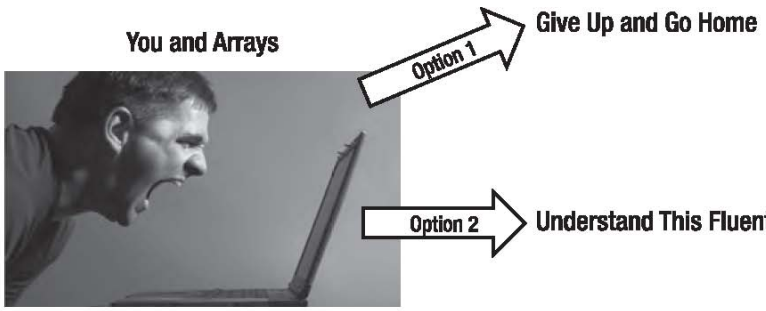
¿Por qué puedes querer puede saltarse este capítulo?
Tiene que tener una cosa clara: no necesita tener conocimientos de matrices para ser un programador de éxito. En una conferencia reciente sobre programación para iPhone e iPad, asistí a una presentación de un excelente equipo de programadores sobre sus propias aplicaciones. Uno de ellos hablaba sobre el ridículo juego de lanzar cosas a un cubo de basura.

La empresa consta de dos programadores cincuentones. Tienen 11 juegos e ingresan más de 20.000$ mensuales. Quédate con esto –dos años antes, estas dos personas eran diseñadoras de interiores y nunca antes habían programado. Cuando se les preguntó acerca de como gestionaban sus matrices, la respuesta que dieron a una sala llena de geeks de alta tecnología fue:

> "*No sabemos nada sobre matrices. Nos limitamos a utilizar esbozos y repetitivas entradas y salidas y luego rezamos*"

¿Por qué decidí enseñarlo a pesar de todo... y porqué *a mi manera*?
Al principio de este libro, dejé al margen este tema. Sin embargo, sabía que en algún momento tendría que decidir si incluirlo, y mis editores también querrían saberlo. *Matrices o no matrices*!

Desde el punto de vista de los estudiantes, esta piedra en el camino fue difícil de superar, pero debido a las exigencias de la academia, se veía en la mayoría de ellos algo parecido a la Figura 8-1. Para la mayoría de mis estudiantes, Opción-1 darse por vencido y abandonar la ingeniería no era una opción en absoluto. Y si llegaron a mi clase, mi objetivo era, por supuesto, conducirlos a la Opción 2- comprometerse lo suficiente para aprender todo el material.




```
- (NSInteger)tableView:(UITableView*)tableView didSelectRowAtIndexPath:(NSIndexPath*)indexPath
{
  NSString *text = [NSDtring stringWithFormat:@"%@%@",[names objectAtIndex:[indexPath row]],@".png"];
  FooVar1Controller* retController = [FooVar1Controller fooVar1ControllerWithImageNamed:text];
  [[self navigationController] pushViewController:retController animated:YES};
```

Foto 8-1. Las opciones tradicionales para enfrentarse a las matrices: 1) Renunciar, abandonar la ingeniería y volver a casa, o 2) comprometerse y hacer todo lo posible para aprender el material de la tortura.

La Figura 8-2. Ilustra mi aproximación al enigma de las "matrices". Podrás ver que consiste en *tres alternativas* - el añadido corresponde a una mezcla de la dicotomía clásica.

1. Evasión: "Matrices-, quién las necesita! Me voy de aquí! Capítulo 9, "Mapkit", "aquí estoy yo."

2. Método Lewis para matrices: Aprenderás dónde y cuando insertar matrices en función del código. Con este enfoque pragmático, tendrás una introducción básica sobre lo que son las matrices y aprenderás algunos trucos útiles. Se han simplificado los aspectos técnicos de forma importante. Terminarás sabiendo mucho de matrices con sólo una muestra de estas. Sin embargo, el código funcionará, y te sentirás realmente inteligente!

3. Aprenda matrices de forma completa: Hay una buena razón por la que Dave Mark y Jeff LaMarche se esperaron a su libro avanzado *More iPhone 3 Development: Tackling iPhone SDK 3* (Apress 2010) para enseñar las matrices. Se refirieron a este aspecto de la programación como "... el diablo..."

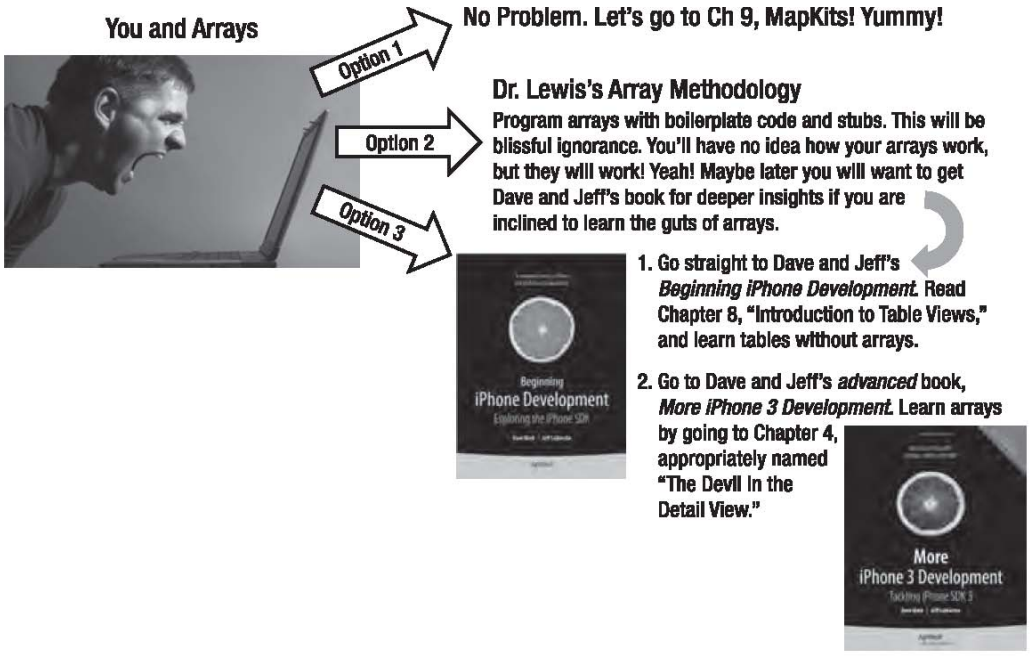


Figure 8–2. *Lewis's take on the issue of arrays: 1) Avoid the issue altogether, 2) Trust in Dr. Lewis's pragmatic approach and learn a few helpful tricks here and there, or 3) Go to Chapter 4 of Dave and Jeff's advanced book and shake hands with the Devil!*

Figura 8–2. Lewis comenta sobre el tema de las matrices: 1) Evitar el tema completamente, 2) Confíe en el enfoque pragmático del doctor Lewis y aprenda algunos trucos útiles aquí y allá, o 3) Ir al capítulo 4 de libro avanzado de Dave y Jeff y estreche la mano con el diablo!

¿Qué hacer?

Si aún estás por aquí y no has saltado al Capítulo 9 o has dejado el libro de lado, estoy asumiendo que estás dispuesto seguir el programa y pasar a la siguiente parte de nuestro viaje. Es importante comunicar dos cosas:

- A veces, pensarás *que no das abasto* con las matrices.
- A veces, pensaré *que te estoy dando demasiados detalles* sobre las matrices.

En otras palabras, muestra la misma flexibilidad y paciencia que has demostrado a lo largo de nuestra reciente "asociación". Confía en mí! ¿De acuerdo?

OK - vayamos al grano entonces.

Vistas de Tabla y Pilas de Navegación (Table Views y Navigation Stacks)

Para entender como se utilizan las matrices en las aplicaciones de iPhone / iPad necesitas entender el papel de los Table Views y las Navigation Stacks, piezas muy potentes y útiles del iOS de iPhone. Hasta ahora, hemos hablado sólo un poco acerca de las tablas. Se utilizan habitualmente para mostrar listas de elementos y permiten al usuario seleccionar y organizar elementos.

Cuando usted quiere hacer la lista de cosas a comprar en la tienda de comestibles ¿qué pasos sigue para prepararla? En primer lugar, busca un papel, y a continuación, escribe todas las cosas que necesita. Un Table View actúa igual que un trozo de papel: organiza los elementos en una lista de modo que pueda encontrar fácilmente cualquier cosa que necesite.

Utilizar Table views para mostrar una lista es sólo una parte de la imagen, sin embargo. Una Navigation Snack nos permite movernos entre Table Views e incluso entre vistas “normales”.

Vamos a mantener el desarrollo de un modo simple el máximo tiempo posible. Ahora, tenga en cuenta cinco aspectos de los Table Views:

1. Un Table View no es más que una lista de cosas, una lista de datos.

2. En el iPhone / iPad, un Table view contiene el código para una vista de objeto – aquello que muestra los datos te tu tabla en la pantalla del iPhone / iPad.

3. El objeto UITableViewCell controla cada fila de un table view. No te preguntes el por qué, simplemente acéptalo.

4. Un table view no almacena los datos de la tabla. Esta sólo almacena código para mostrar las filas que están visibles en la pantalla del iPhone / iPad

5. Los Table Views incluyen al menos dos partes de datos:

> **a.** La información sobre qué tipos de datos son almacenados y cómo estos han de ser configurados por el objeto *UITableViewDelegate*;
>
> **b.** Información acerca de cómo los datos concretos son ordenados y presentados - por el protocolo del objeto *UITableViewDelegat*e.

Del mismo modo, las matrices sólo son listas de cosas y ocurre que los table views son perfectos para mostrar y organizar matrices. Pero antes, debes de tener una vaga idea de cómo funciona una matriz:

> *Una matriz es una colección ordenada de objetos, a partir de cero, y puede almacenar cualquier cantidad de objetos.*

Imagina una máquina expendedora de golosinas con una amplia variedad de productos, numerados desde de A1 a H6 de la parte superior izquierda a la inferior derecha.

Centras tu atención en un snack que parece demasiado tentador como para dejarlo pasar, por lo que pulsas el número y recibes ese snack. El número utilizado para acceder a tu caramelo, para identificar su posición en la secuencia, se llama índice. El método de indexado (o "índices") para encontrar un objeto de una matriz es el mismo que utilizaríamos para obtener un caramelo de la máquina expendedora. Una diferencia importante es que un objeto permanece en una matriz incluso después de encontrarlo con su índice. El caramelo, por suerte, no permanece en la máquina expendedora.

Vamos a imaginar que tu trabajo es el de indicar los tipos de dulces disponibles de la máquina expendedora, pero no puedes mirar ni contar, ya que se ha sustituido el cristal, por una lámina de plástico opaco y todos los números se han eliminado.

El trabajo que acabo de encomendarte es el mismo que desempeñaría una *table view*. Primero, necesitas saber como están dispuestas las golosinas en la máquina. Afortunadamente, las máquinas expendedora tienen métodos para saberlo. Tú lo leerás y lo mostrarás en la pantalla.

Food: Siguiendo el Modelo de la App Store

Para este ejemplo, vamos a utilizar matrices, vistas y tablas para hacer una aplicación muy sencilla. Será construida bajo el paradigma maestro-detalle y utilizara table views para mostrar entradas de deliciosos menús de cenas! El enfoque que utilizaremos será similar al utilizado por la App Store para mostrar las opciones en el iPhone / iPad. Aplicaremos un modelo en el que el usuario empieza en un table view que se le muestra las categorías, pasa a una lista de elementos de la categoría seleccionada, y finalmente abre una página con información detallada de la elección del usuario.

Usaremos código preprogramado que puede usarse para crear matrices, y vamos a utilizarlo en otras aplicaciones como base para crear aplicaciones similares a la App Store descrita anteriormente.

Empezando la aplicación Food

Como de costumbre, abriremos Xcode y crearemos un nuevo proyecto utilizando el acceso directo o seleccionando la opción en el menú desplegable File, como se muestra en la Figura 8-3. Elija la plantilla Navigation-based Application, como se muestra en la Figura 8-4.

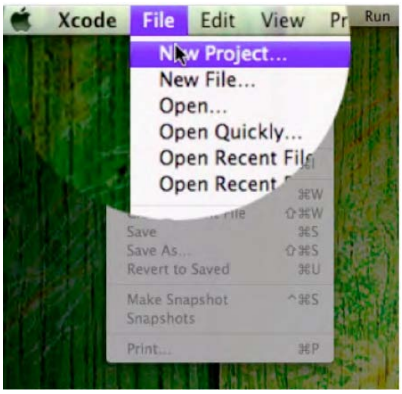


Figura 8–3. Inicie un nuevo proyecto. Tiene razón - No es el atajo utilizado de costumbre... Sólo quería ver si estaba atento!

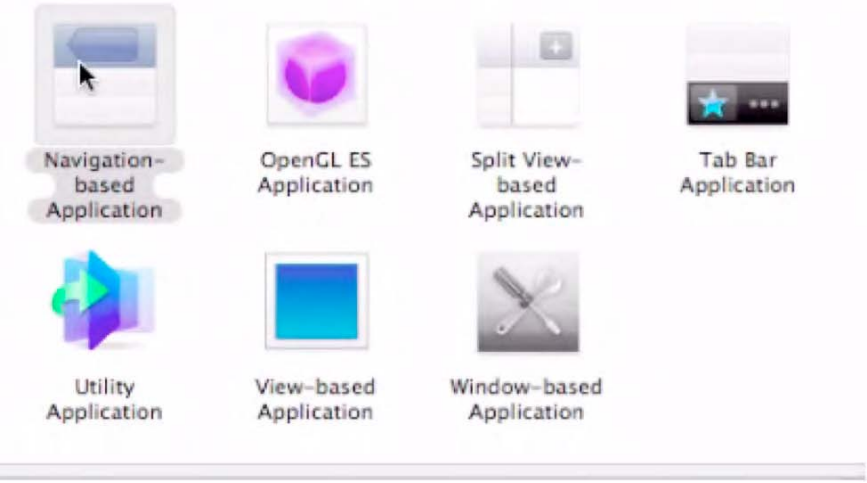


Figura 8–4. Seleccione Navigation-based Application.

Asegúrese de las imágenes que va a utilizar en el proyecto de Xcode, tal y como se aprecia en la Figura 8-5

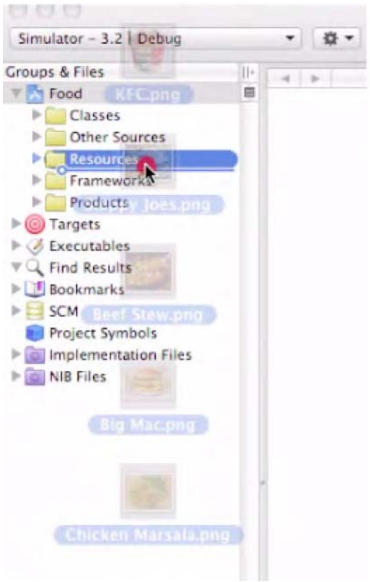


Figura 8–5. Copie las imágenes en el proyecto de Xcode. No se olvide del archivo del icono. Tenga en cuenta que parece las imágenes parece que se han arrastrado a la carpeta Resources, pero en realidad esta en la ruta de la carpeta Other Sources.

.

Las imágenes las puedes encontrar en la web del libro `www.apress.com`, junto con el código fuente para este proyecto. Arrastra las imágenes dentro de la carpeta Other Sources y asegúrate de marcar la casilla Copy Into. Cuando la pantalla muestre algo similar a la Figura 8-6 estarás listo para empezar.

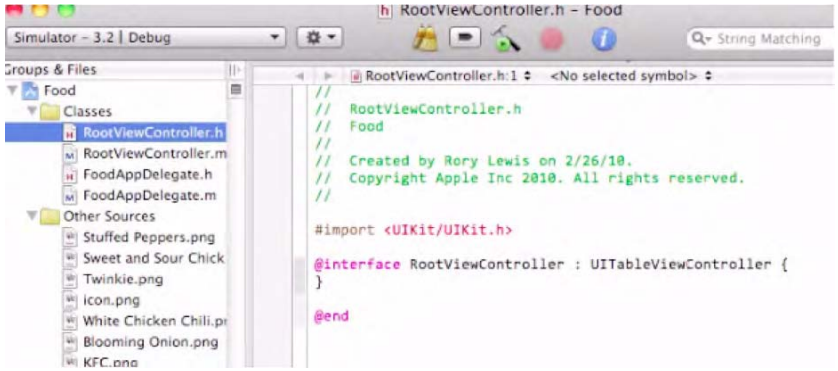


Figura 8–6. Nuestro proyecto esta listo para empezar.

Agregando la matriz Category Names en el RootViewController.h

A esta altura, únicamente vamos a configurar la tabla de la misma forma que crearíamos una lista o gráfico de Excel. Todavía no hemos comenzado códigos preprogramados.

Con el fin de cumplimentar la lista Category Names, tenemos que ser capaces de almacenarla dentro del `RootViewController`. Por lo tanto, tenemos que movernos dentro del archivo `RootViewController.h` y configurar un campo para guardar las categorías. También tenemos que asegurarnos que de que nuestro `RootViewController` pueda rebuscar en la información que el usuario necesita creando un nuevo View Controller. Para ello, añadiremos una línea de importación y un nuevo campo en el archivo de cabecera (header file), como se muestra en la Figura 8-7

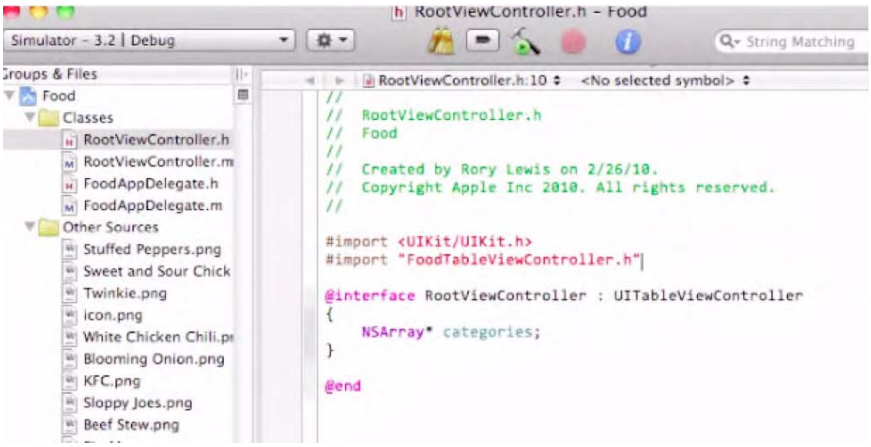


Figure 8–7. Importar el archivo FoodTableViewController.h y crear una matriz de categorías...

Añada la línea `#import "FoodTableViewController.h"` al principio del archivo de cabecera. Esta instrucción importará una clase que crearemos en un paso posterior, pero lo estamos poniendo aquí para que ya esté donde debe estar.

A continuación, agrega un campo en la clase con la línea `NSArray* categories`. Esta matriz contendrá los nombres de las categorías para utilizarlos en el table view.
//

```
// RootViewController.h

#import <UIKit/UIKit.h>
#import "FoodTableViewController.h"

@interface RootViewController : UITableViewController
{
       NSArray* categories;
}
@end
```

Creando la matriz de categorías en -viewDidLoad

Pasamos al archivo de implementación de `RootViewController`. tenemos que configurar como se van a mostrar los nombres de las categorías. En primer lugar, debemos crear los nombres de las categorías y guardarlos para más adelante. Guardaremos nuestra matriz de nombres en el campo Categorías que hemos creado previamente.

En `RootViewController.m`, rescribiremos el método `-viewDidLoad` para configurar lo que necesitamos.. Primero, llamamos a `[super viewDidLoad]`para que la superclase responda a la vista cargada de forma normal. Crearemos una matriz con todos los nombres que queremos y establecemos "categories" como una nueva matriz. Advierte el símbolo @ y el ítem *nil* al final, ya que ambos son importantes!

En la siguiente línea, establecemos el título del View Controller para que cuando el controlador de navegación necesite mostrar el título "Categories", este sea mostrado. Este título se muestra en la parte superior del table view en la barra de navegación. El código es el siguiente:

```
-(void)viewDidLoad
{
   [super viewDidLoad];

        categories = [[NSArray alloc] initWithObjects:@"Chicken", @"Beef", @"Pork",↩
 @"Fish", @"Vegetarian", @"Really, Really Healthy Food", nil];
       [self setTitle:@"Categories"];
}
```

Configurando el Fuente de Datos (Data Source) del Table View

La clase UITableView utiliza delegación y objetos de fuente de datos con el fin de obtener datos para mostrar y manejar las entradas del usuario. Los métodos (methods) que más nos interesan en la fuente de datos, ya vinculados al `RootViewController`, son los siguientes:

```
-(NSInteger)numberOfSectionsInTableView:(UITableView*)tableView
-(NSInteger)tableView:(UITableView*)tableView numberOfRowsInSection:(NSInteger)section
-(UITableViewCell    *)tableView:(UITableView    *)tableView    cellForRowAtIndexPath:↩
(NSIndexPath *)indexPath
```

Estos no son los únicos métodos de fuentes de datos, pero son los métodos utilizados en este ejercicio. Empezaremos con `-numberOfSectionsInTableView`. Este método devuelve el número de secciones en los *argument* del table view. Las secciones separan la tabla en porciones, cada una muestra la información basándose en esa sección. Asegúrate de que en el método (method) aparece "`return 1;`", ya que este indicador es el que avisa al table view de que sólo tiene una sección. Esto se traduce en que todos se mostrarán juntos. El código es el siguiente:

```
-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}
```

A continuación, para `–tableView:numberOfRowsInSection`, quiero recordarte algo para todas tus futuras aventuras con los table views

Devolveremos el número de elementos que queremos mostrar.

En este caso, queremos mostrar *todos* los elementos de la matriz `categories`, y el código para este método es:

```
return [categories count]
```

Este comando pregunta a la matriz cuántos elementos contiene y devuelve ese número al table view para saber el número de elementos que serán mostrados:

```
-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return [categories count];
}
```

Por último, necesitamos rescribir `–tableView:cellForRowAtIndexPath`: para nuestros métodos de fuente de datos.. Este método crea un UITableViewCell, modifica esta celda para mostrar los datos apropiados, y luego devuelve esta misma celda al table view para mostrarla en pantalla. Queremos cambiar el texto de la celda para mostrar el nombre de la categoría que hemos creado anteriormente. Para hacer esto, necesitamos crear una celda con un identificador de reutilización, cambiarlo y devolverlo.

NOTA: ¿Qué es un *identificador de reutilización*? Para aprender más sobre estas pequeñas joyas, remítase al final de este capítulo, en la sección "Profundizando en el Código".

Volvemos al tema de los métodos de fuente de datos! Aquí tienes el código revisado:

```
-(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:↩
 (NSIndexPath *)indexPath
{

    static NSString *CellIdentifier = @"Cell";

       UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:↩
 CellIdentifier];
      if (cell == nil) {
       cell   =   [[[UITableViewCell   alloc]   initWithStyle:UITableViewCellStyleDefault↩
   reuseIdentifier:CellIdentifier] autorelease];
      }

       NSString* text = [categories objectAtIndex:[indexPath row]];
       [[cell textLabel] setText:text];

    return cell;
}
```

Delegación en un Table View

A continuación, necesitamos configurar la *delegación* en el `RootViewController` para nuestro table view. La delegación le indica al table view qué hacer cuando el usuario pulsa sobre un elemento de la tabla. Usaremos:

```
-(void)tableView:(UITableView*)TableViewdidSelectRowAtIndexPath:
(NSIndexPath*)indexPath
```

Como método para lograr esta funcionalidad.

La primera línea de este método simplemente crea un pointer (indicador) hacía una matriz, que inicialmente esta fijado a *nil*. Hacemos esto para poder comprobar con posterioridad el que se producirá un error si el usuario no pulsa sobre la fila sobre la que nosotros estemos dando soporte. Una *instrucción switch (switch statement)* hará el siguiente trabajo; determinará qué fila es seleccionada por el usuario. Si se trata de una de las filas a las que le damos soporte, ya sean la fila0 (la primera fila), fila1 (la segunda fila), ... 5 (la sexta y última fila), redirigiremos el pointer de la matriz que configuramos antes hacia una matriz que contenga los datos que queremos mostrar en el siguiente view controller.

A continuación, compruebe si el pointer de la matriz se ha establecido en algo distinto a nil. En este mundo de la programación, "*nil*" siempre será evaluado como False (Falso), mientras que cualquier otro objeto se evaluará como Trae (Verdadero). Si la matriz es valida, crearemos un nuevo `FoodViewController` con la matriz y desplazaremos el nuevo controlador hacía la navigation stack (*pila de navegación)*. Después de todo esto, tendremos un table view *deseleccionando* la línea que fue seleccionada, solicitando su animación para realizar el proceso de deselección.

Se dará cuenta de que todos los nombres asignados a la matriz que hemos hecho en este método son idénticos a los nombres de las imágenes utilizadas en el proyecto. No es casual. Utilizaremos estos elementos en la matriz para obtener las imágenes más tarde.

El código para de nuestro método de delegación es el siguiente:

```
-(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
 *)indexPath
 {

       NSArray* names = nil;

       switch ([indexPath row])
       {

           case 0:
                  names = [NSArray arrayWithObjects:@"Chicken Marsala",
@"White Chicken Chilli", @"Sweet and Sour Chicken", nil];
              break;

           case 1:
                  names = [NSArray arrayWithObjects:@"Beef Stew",
@"Sloppy Joes", @"Stuffed Peppers", nil];
              break;

           case 5:
                  names = [NSArray arrayWithObjects:@"Big Mac",
@"Twinkie", @"KFC", @"Blooming Onion", nil];
              break;

           default:
                  break;
       }
       if (names)
       {
           FoodTableViewController* ftvc = [FoodTableViewController
foodTableViewControllerWithFoodNames:names];
           [[self navigationController] pushViewController:ftvc animated:YES];
       }
       [tableView deselectRowAtIndexPath:indexPath animated:YES];
}
```

Creando FoodTableViewController

Tenemos que crear el siguiente nivel de detalle de nuestra aplicación. El primer nivel eran las categorías de alimentos, pero ahora queremos más detalles... en este caso, la elección de alimentos dentro de una categoría. Haremos esto mediante la creación de un view controller, nuestro `FoodTableViewController`, para mostrar información más específica (Figura 8-8). Al agregar el nuevo archivo, asegúrese de crear `FoodTableViewController` una subclase de UITableViewController utilizando la casilla de verificación de la sección Options de la plantilla, como se muestra en la Figura 8-9

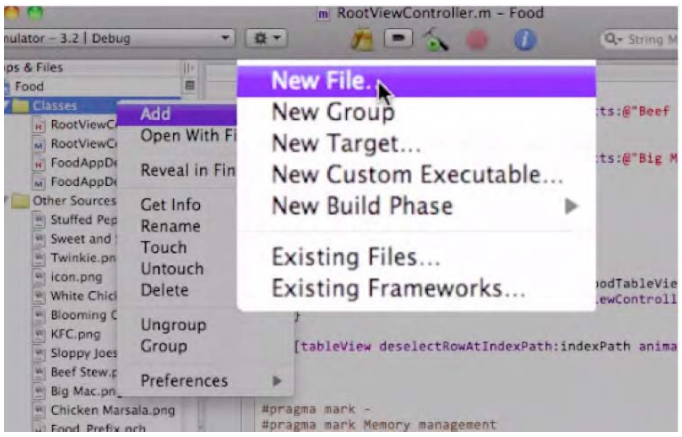


Figure 8–8. Añadiendo un nuevo archivo al proyecto Food..

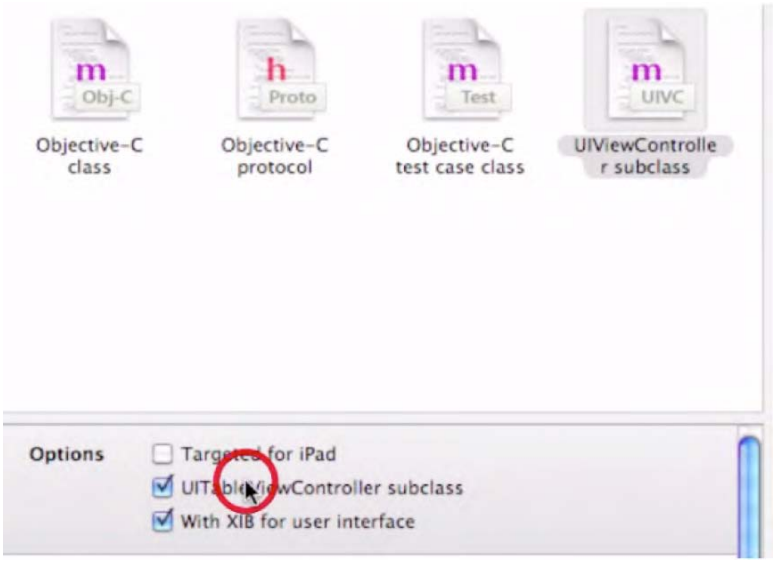


Figura 8–9. Creando el FoodTableViewController como una subclase de UIViewController marcando la casilla UITableViewController Subclass.

Una vez nombrada y almacenada su nueva clase `FoodTableViewController`, como se muestra en la Figura 8–10, abra el archivo `FoodTableViewController.h` . Dentro de este archivo, vamos a añadir código muy similar al del `RootViewController.h` , pero con algunos cambios. En primer lugar, inserte en el archivo la línea `#import" FoodTableViewController.h"`. Esto importará la clase `FoodViewController` una vez que esté funcionando.

A continuación, crearemos un nuevo campo: `NSArray* names`. Esta matriz contendrá los nombres de los alimentos para el controlador del table view. Esta matriz de funciones se parece mucho a la matriz de `categories` del `RootViewController`. Sin embargo, tenemos que hacer visible la matriz de nombres hacía las otras clases para que el `RootViewController` pueda pasarle los datos que el `FoodTableViewController` necesite. Para ello, creamos una propiedad con la línea `@property(nonatomic, retain) NSArray* names`. Esto nos permite realizar una llamada a `setNames`: en una instancia de esta clase.. También crearemos un "constructor de conveniencia" (conveniente constructor llamado `foodTableViewControllerWithFoodNames`.

Qué es un convenience constructor (constructor de conveniencia)?! ¿Por qué nos molestamos en hacerlo? Bueno, un constructor de conveniencia crea un objeto, generalmente inicializado con varios parámetros, y se encarga de gestionar la memoria por nosotros. Esto significa que la llamada al constructor de conveniencia no nos obliga a liberar el objeto cuando se haya terminado de usar. Por ello, no sólo podemos crear un objeto con datos de inicio definidos, sino que no tendremos que preocuparnos en seguir su tiempo de vida, despreocupándonos de posibles pérdidas de memoria. Además, los constructores de conveniencia son muy fáciles de hacer.

NOTA: Véase "Digging the Code" para aprender más acerca de la administración de memoria.

Recordarás que en el capítulo 4 mencionamos que el signo más "+" al principio de un constructor de conveniencia convertía al método (method) en un *método de clase* (class method). Si no te acuerdas, recuerda que simplemente significa que podemos pedir a la clase que nos cree una nueva instancia en lugar de tener que hacer un `alloc` y un `init` sobre el objeto nosotros mismos.

Figura 8–10. Déle un nombre y guarde la nueva clase FoodVTableViewController.

```
//
// FoodTableViewController.h

#import <UIKit/UIKit.h>
#import "FoodViewController.h"

@interface FoodTableViewController : UITableViewController
{
        NSArray* names;
}
+(FoodTableViewController*)
foodTableViewControllerWithFoodNames:(NSArray*)foodNames;

@property (nonatomic, retain) NSArray* names;

@end
```

Creando el constructor de conveniencia para el FoodTableViewController

En `FoodTableViewController.m` comenzaremos creando el cuerpo del constructor de conveniencia que hemos declarado en la cabecera. Pero primero, tenemos que añadir la línea `@synthesize names;` para que la propiedad este disponible para nosotros. A continuación, prepararemos nuestro constructor de conveniencia creando una directiva de compilador en C (una forma realmente fácil de cambiarlo y utilizarlo) que se llamará `FoodTableViewControllerNibName`. Varios espacios más adelante, vamos a definir la directiva que será `@FoodTableViewController`, ya que será el nombre del archivo `.xib` que querremos para el uso del view controller.

¿Por qué estamos haciendo eso? Vamos a utilizar este nombre para nuestro constructor de conveniencia. Al crear esta directiva en el archivo de implementación, esconderemos como funciona nuestro constructor de conveniencia y nos aseguraremos de que el `archivo.xib` se utiliza para construir el view controller. En pocas palabras, hacer la creación de un nuevo `FoodTableViewController` más sencilla al no requerir un nombre nib cuando es llamado desde otro lugar.

Después de escribir nuestra directiva, querremos insertar el código de nuestro constructor de conveniencia. Puede copiar y pegar la declaración de la cabecera en el archivo de implementación, quite el punto y como del final, añada corchetes y estaremos listos para empezar!

Lo primero que haremos en el constructor de conveniencia es crear una nueva instancia de `FoodTableViewController` utilizando el método `-initWithNibName:bundle`, pasando *el nombre de la directiva* en el primer parámetro, y "*nil*" en el segundo. Esto mantendrá la implementación y el nombre del nib ocultos a las miradas indiscretas y hará que el método sea muy sencillo de utilizar.

Antes de que devolvamos la instancia, sin embargo, queremos establecer los nombres del controlador que el usuario pasa al constructor de conveniencia. Llamaremos a la propiedad que hemos establecido en la cabecera para establecer los nombres. Luego llamamos a `return [theController autorelease]` para asegurarnos de que el que llama a la instancia no tiene que preocuparse por la gestión de memoria.

Esto es todo. Ya hemos *terminado* con el constructor de conveniencia!

```
/
// FoodTableViewController.m

#import "FoodTableViewController.h"

#define FoodTableViewControllerNibName
@"FoodTableViewController"

@implementation FoodTableViewController

@synthesize names;

+ (FoodTableViewController*) foodTableViewControllerWithFoodNames:(NSArray*)foodNames
{
          FoodTableViewController* retController = [[FoodTableViewController alloc]↩
 initWithNibName:FoodTableViewControllerNibName bundle:nil];
          [retController setNames:foodNames];
          return [retController autorelease];
}
```

Data Source y Delegación para el FoodTableViewController

La operatividad de nuestra table view no es muy grande hasta ahora, ya que todavía no le hemos dicho qué debe mostrar. Igual que antes, hemos de crear nuestra fuente de datos y nuestros métodos de delegación. Y como antes, queremos `return 1`; para el –`numberOfSectionsInTableView:` y lo mismo para `return [names count];` y para –`tableView:numberOfRowsInSections`. Todo es bastante sencillo y funciona exactamente igual que el código del `RootViewController`. Y como se puede esperar después de todo esto para el –`tableView:cellForRowAtIndexPath:` todo lo que tenemos que hacer es cambiar `categories` a `names` y lo haremos con métodos de fuente de datos para el `FoodTableViewController`:

```
-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}
-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return [names count];
}

-(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:↩
(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:↩
CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault↩
 reuseIdentifier:CellIdentifier] autorelease];
    }
       NSString* text = [names objectAtIndex:[indexPath row]]; [[cell
       textLabel] setText:text];
    return cell;
}
```

En la delegación! Por supuesto, para la delegación vamos a tener que cambiar mucho más de que lo que necesitamos en el `FoodTableViewController`. Dentro del –`tableView:didSelectRowAtIndexPath`: tenemos que crear el siguiente nivel de detalle y pasarle alguna información. Ese es nuestro siguiente nivel del `FoodViewController`, lo crearemos en el siguiente paso. Por ahora, vamos a suponer que está completado y utilizaremos un constructor de conveniencia que crearemos más tarde.

Primero crearemos una cadena que utilizaremos para obtener la imagen del siguiente nivel de detalle. Sin embargo, no podemos utilizar el nombre desde la matriz de nombres; tenemos que agregar la extensión antes de que el UIImage cargue la imagen que estamos buscando.

Por tanto, utilizaremos el método NSString `+stringWithFormat`: pasando @%@%@, para devolver el nombre de la matriz y la extensión que queremos usar, en este caso @“`.png`”. El símbolo %@ para el primer argumento se utiliza para indicar que un objeto va en esa posición. Desde que estamos pasando NSStrings para ambas posiciones, estas tienen efecto de concatenación de cadenas, que es precisamente lo que estamos buscando.

¿QUÉ SIGNIFICAN LOS SÍMBOLOS @%@%@?

La respuesta se sale del alcance de este libro. Si realmente quiere saber el significado de los símbolos %@, ha de tener en cuenta esto. En las cadenas de formato, el carácter @'%' anuncia un marcador de posición para el valor, con los caracteres que siguen para determinar el tipo de valor esperado, y como darle formato al mismo. Por ejemplo, un formato de cadena como "%d `houses`" espera un valor entero para sustituir por la expresión con formato '%d'.`NSString` soporta los caracteres estándar definidos por la función ANSI C print(f), más '@' para cualquier objeto. Si el objeto responde a `description withLocale: message`, `NSString` envía ese mensaje para recuperar la presentación del texto; de lo contrario, envía un mensaje de descripción.

Después, pasamos la cadena que creamos hacía el constructor de conveniencia para el `FoodViewController` que queremos construir. Una vez que hemos llamado al constructor de conveniencia, moveremos el `FoodViewController` hacía la pila de navegación, entonces le pediremos por la animación. Por último, y como antes, deseleccionaremos la fila y marcaremos la animación: YES, con lo que habremos acabado los métodos de delegación para el `FoodTableViewController`.

```
-(NSInteger)tableView:(UITableView *)tableView didSelectRowAtIndexPath:↩
(NSIndexPath *)indexPath
{
     NSString *text = [NSString stringWithFormat:@"%@%@", [names objectAtIndex:↩
[indexPath row]], @".png"];
     FoodViewController* retController = [FoodTableController↩
 foodViewControllerWithImageNamed:text];
      [[self navigationController] pushViewController:retController animated:YES];
}
```

Creando la clase *FoodViewController*

Todavía nos falta algo en el `FoodViewController`. Esta es la última pieza de la aplicación y la clase maneja el más alto detalle en nuestra aplicación de navegación. Cuando añadamos el nuevo archivo al `FoodViewController`, nos aseguraremos de de desactivar el checkbox "`UITableViewController subclass`" en el selector de plantilla, como se muestra en la Figura 8-11. En su lugar, vamos a utilizar una vista sencilla para mostrar los alimentos cuando se cargue el view controller.

Figura 8–11. Cree el FoodViewController y asegúrese que la casilla "UITableViewController subclass" esta desactivada.

El Archivo de Cabecera (Header File) *FoodViewController*

Abra el archivo de cabecera `FoodViewController.h` . Primero agregaremos algunos campos que deberían ser bastante sencillos por ahora. Añada un `UIImageView* field` y un `NSString* field`, y nombre estos campos como `imageView` y `imageName`, respectivamente. El UIImageView será un `IBOutlet` que enlazaremos más tarde. Esta vista de imagen mostrará la imagen de la comida que el usuario ha seleccionado. El campo `NSString` mantendrá el nombre de la imagen deseada hasta que se necesite. Nos centraremos en crear propiedades para estos dos campos. Asegúrese de poner "IBOutlet" delante de UIImageView.

Otra de las tareas que debemos manejar es la declaración del constructor de conveniencia. Este método de clase tiene un único argumento NSString, como se muestra a continuación:

```
//
// FoodViewController.h
#import <UIKit/UIKit.h>

@interface FoodViewController : UIViewController
{
       UIImageView* imageView;
       NSString* imageName;
}
+ (FoodViewController*) foodViewControllerWithImageNamed:(NSString*)name;

@property (nonatomic, assign) IBOutlet UIImageView* imageView;
@property (nonatomic, copy) NSString* imageName;
@end
```

El constructor de conveniencia FoodViewController

Sólo hay dos cosas que tenemos que hacer en el archivo de implementación de `FoodViewController` : crear el constructor de conveniencia, y cargar una imagen para cuando la vista es cargada.
Vamos a empezar con el constructor. Al igual que antes, empezamos por crear una directiva con el nombre del nib llamado `FoodViewControllerNibName`, con el nombre apropiado del nib. A continuación, tenemos que asegurarnos que todas nuestras propiedades estén sintetizadas. Copiar y pegar la firma del método hará que iniciemos nuestro constructor de conveniencia.

La primera línea del constructor de conveniencia crea la instancia del `FoodViewController` que devolveremos, pasando la directiva a `–initWithNibName:bundle`. El nombre de la imagen se ajusta a través de un "property method call", una sub-rutina que ayuda a administrar recursos de forma eficiente. Por último, la nueva instancia se envía en un mensaje de *disparador automático* y el resultado de ese mensaje es devuelto.

```
//
// FoodViewController.m
#import "FoodViewController.h"

#define FoodViewControllerNibName
@"FoodViewController"

@implementation FoodViewController
@synthesize imageView;
@synthesize imageName;

+ (FoodViewController*) foodViewControllerWithImageNamed:(NSString*)name
{
        FoodViewController* retController = [[FoodViewController alloc]↵
  initWithNibName:FoodViewControllerNibName bundle:nil];
        [retController setImageName:name];
        return [retController autorelease];
}
```

Configurando el FoodViewController, -viewDidLoad, y el (.xib)

El último trozo de código que necesitamos es reemplazar –viewDidLoad en el FoodViewController.m. Todo lo que necesitamos hacer es agregar unas pocas líneas de código. La imagen para el outlet UIImageView que se creo en el encabezado necesita una foto para ser mostrada. Para ello, basta con crear un UIImage con el método de clase llamado `+imageNamed`. Esto crea una imagen con los datos del archivo con el nombre facilitado. Al establecer esta imagen en el `UIImageView`, esta se hará visible. No necesitaremos hacer uso de ningún `imageName`, ya que ahora que la imagen se ha cargado, la podremos liberar.

```
-(void) viewDidLoad

{
       [imageView setImage:[UIImage imageNamed:imageName]];
       [imageName release];
}
```

Por último, abra el archivo `FoodViewController.xib`, como se muestra en la Figura 8-12, porque haremos algunos movimientos conocidos para arreglar la vista

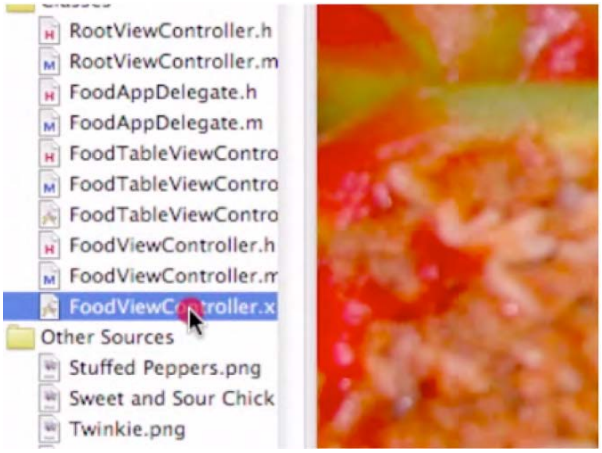


Figura 8–12. Abra el archivo FoodViewController.xib ..

Arrastre un UIImageView a la Vista, como se indica en la figura 8-13 y cambie su tamaño para ocupar todo el espacio. clic-derecho y arrastre desde el icono Files's Owner hacía el nuevo UIImageView, como en la figura 8-14. Entonces, conecte el outlet imageView , como se muestra en la figura 8-15.

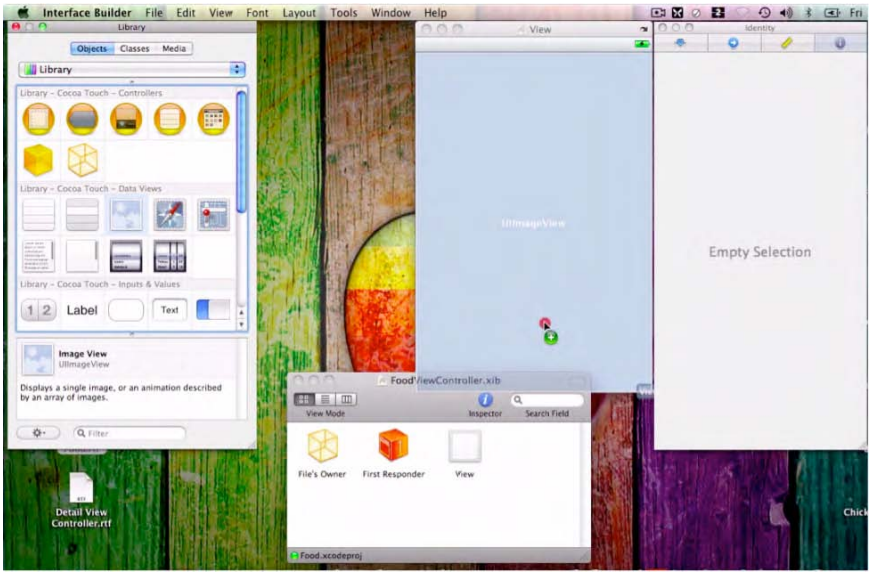


Figura 8–13. Coloque un UIImageView en la vista y asegúrese que tiene el tamaño correcto..

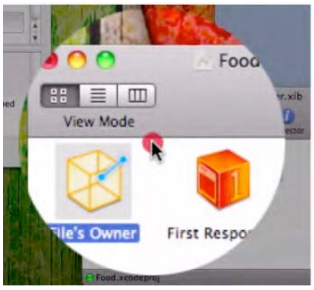


Figura 8–14. clic-derecho y arrástrelo desde el icono File's Owner ….

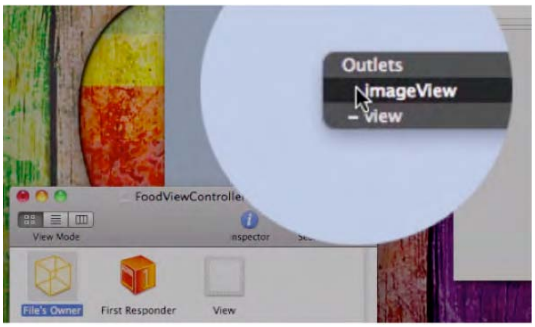


Figura 8–15. … sobre el UIImageView y seleccione la opción imageView en el menú desplegable Outlets.

Archivo de Icono

Nuestra aplicación de búsqueda de alimentos esta acabada! ¿Pero qué aplicación se pude considerar como buena sin un icono decente? El icono es lo que hará que el usuario ejecute su aplicación, que la compre. Será la primera y última impresión de su aplicación. Con esto en mente, vamos a configurar un icono para su aplicación.

El icono, una imagen `.png` de 52 × 52 píxeles, ya se ha añadido a nuestro arsenal de imágenes en la carpeta "Resources". Así, abra el archivo Food `Info.plist` y establezca la entrada Icon File con el nombre que desee utilizar para su icono. Véase la Figura 8-16
Por cierto, en general la extensión `.png`, no hace falta ser incluida en el nombre.

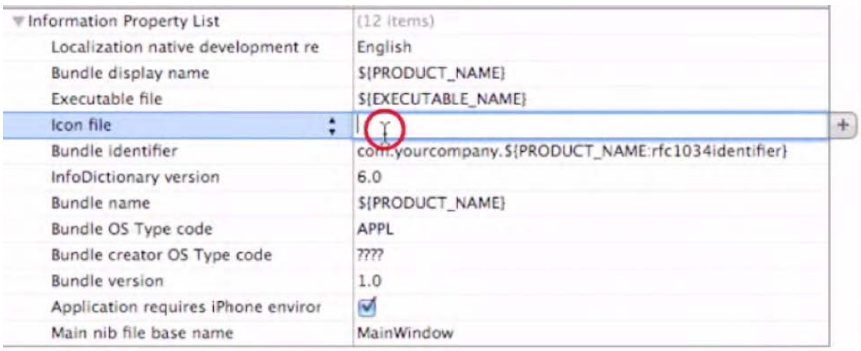


Figura 8–16. En el Info.plist, establezca el nombre del icono.

Probando la Aplicación

Felicidades! Ha terminado con el ejercicio del capítulo 8!

Ahora vamos a probar la aplicación. Ejecutar la aplicación "Food" permitirá al usuario seleccionar las categorías de platos, buscar platos específicos en la categoría, y ver una imagen del plato para que se le haga la boca agua. Las Figuras 8-17 hasta la 8-20 muestran la secuencia desde los menús para ver la entrada seleccionada.

Con el poder del UIKit, específicamente mediante `UINavigationController`, la interfaz de navegación de detalles se nos proporciona casi sin haber introducido ningún tipo de código por nuestra parte. Los chicos de Apple construyeron el UIKit con un montón de clases útiles y trozos de código que reducen significativamente la cantidad de trabajo requerido para producir potentes y pulidas aplicaciones en un corto espacio de tiempo.

Bien hecho!

Figura 8–17. Un bonito menú de opciones representa el nivel de Categorías..

Figura 8–18. Dentro de la categoría Chicken, veremos varias entradas.

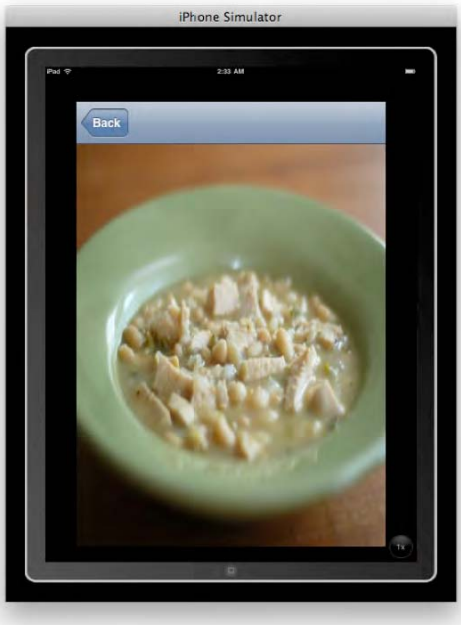


Figura 8–19. Aquí tenemos una imagen detallada de una selección en el Simulador iPad "2x" aumentada. Mmm, pollo!

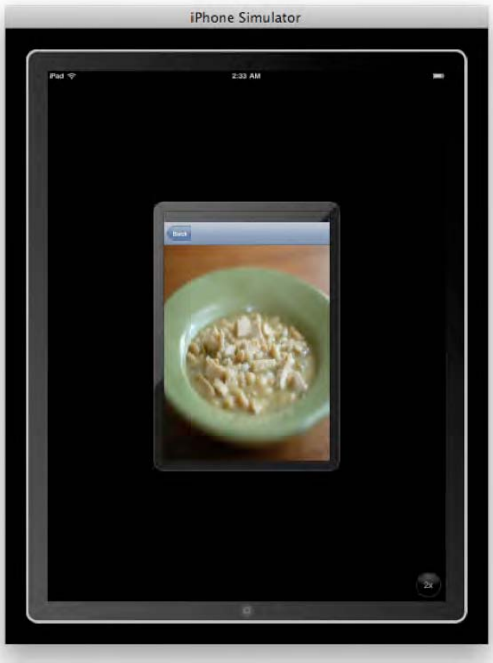


Figura 8–20. Aquí tenemos la imagen después de pulsar sobre el botón "1x" de la parte inferior derecha en el modo simulador.

Profundizando en el Código

Dos conceptos generales me llamaron la atención para este fin de capítulo. El primer tema, la gestión de memoria, es, obviamente, un aspecto primario e importante de las computadoras y la programación, y espero que encuentres este apartado edificante. El segundo tema, los identificadores de reutilización (reuse identifiers), es un concepto que en principio se antoja oscuro y extraño, pero intentaremos arrojar luz sobre él.

Gestión de Memoria

¿Por qué querremos liberar el objeto `imageName` cuando hayamos acabado de usarlo? ¿Cuantos objetos podemos hacer? ¿Que debo hacer si no quiero utilizar más un objeto? La *gestión de memoria* en el iPhone y el iPad es un tema crítico y difícil. El iPhone es un dispositivo muy pequeño en comparación con una computadora de escritorio. Esto hace que la memoria disponible en el iPhone sea comparativamente pequeña, y debemos asegurarnos de que la memoria utilizada sea tan poca como sea posible. El iPad tiene mucha más memoria que el iPhone, pero la gestión de memoria sigue siendo crucial. Hay tres conceptos en el ámbito de la gestión de memoria que quiero que te queden claros: *retener, liberar, y autoliberar*.

Imagina que un trozo de papel en un abarrotado restaurante. Uno de los chefs se acerca a la hoja de papel y anota algo: algo que él necesita saber. Antes de irse, sin embargo, añade esta línea en la parte superior de la nota: “Por favor, no lo tireis! Lo necesito!” Los otros chefs y personal de cocina también escriben cosas en el pedazo de papel y agregan sus peticiones para que nadie pierda esta importante pieza de papel. A lo largo del día y la noche, este personal vuelve y, uno a uno, tachan cada una de sus notas y peticiones. Por último, el último chef viene para tachar sus notas personales. Él se da cuenta que nadie necesita más el trozo de papel y procede a tirarlo.

Así es como *retenemos* (retain) y *liberamos* (release) el trabajo. Retener le dice a un objeto: “Hey, espera un poco, yo aún lo necesito para algo”. Liberar le dice a un objeto: “No te necesito más”. Retener y Liberar crea un contador que crece y decrece, respectivamente, en función de la utilidad de un objeto. Esta cuenta oficial, se llama *cuenta de retención* (retain count) y mantiene un registro de cuantas cosas necesita un objeto. En el ejemplo anterior, el contador sería mantener el número total de firmas del trozo de papel. Cuando la cuenta de retención de un objeto llega a cero significa que nadie necesita el objeto- entonces la asignación se cancela, es decir, se desecha.

Estos son los fundamentos de la gestión de memoria en el iPhone e iPad. ¿Esto es todo? Bueno, no del todo. También tenemos que comprender qué tipo de convenios se han establecido para la gestión de memoria en Objective-C. Afortunadamente, son bastante fáciles y, si los sigues, nunca irás por mal camino. La primera regla es que los métodos que contengan alguno de los siguientes términos “propios” *alloc, init, copy, new*. Los métodos que contengan cualquiera de estas palabras devuelven un objeto con una cuenta de retención de +1. Esto significa que USTED (si, usted!) es responsable de llamar la liberación de este objeto cuando haya acabado con este, para reducir la cuenta de retención a cero. Cualquier método que no contenga uno de esos términos será un objeto autoliberado, por lo que no será necesario mantener una cuenta o preocuparse por ello. De hecho, nunca debe tratar de liberar un objeto autoliberado devuelto desde un objeto a menos que se envíe un mensaje de mantener.

El Autoliberado trabaja liberando objetos en algún momento del futuro. Digamos que acaba de llamar a alloc/init en un objeto, pero no está seguro cuanto tiempo necesita mantener ese objeto. Como acabo de explicar, estos comandos tienen sus propias referencias y, por tanto, es su trabajo señalar cuando se liberan. Debido a que a menudo no estamos seguro de cuanto tiempo se deben conservar, en su lugar enviaremos un mensaje de autoliberación. Esto asegura que la liberación será llamada correctamente en ese objeto, pero que aún se podrá utilizar por un corto espacio de tiempo. Saber cuanto tiempo depende de varias cosas. Si necesita mantener un objeto durante un tiempo prolongado, debería retenerlo en su lugar.

Aún no estamos fuera de peligro. Hay algunas convenciones sobre modificar la retención de objetos que aún no hemos cubierto. La mayoría de contenedores (todos los contenedores de Cocoa, en este caso) retienen sus contenidos. Los métodos con las palabras “add”, “remove” o “st” suelen cambiar la cuenta de retención del objeto que los ejecuto. Por ejemplo, el método `-addObject:` de `NSMutableArray` conserva el objeto pasado como argumento. Esto significa que si no necesitamos el objeto para nada más, querremos enviar al objeto un método de liberación para renunciar a la propiedad del objeto. Si ha estado atento, sabrá que el objeto añadido a la matriz se quedará desde el principio con un recuento de 1, se le añade a la matriz para mantener un recuento de 2, y luego se libera, volviendo a conservar el recuento 1.

¿Lo ve? La gestión de memoria no está mala! Una cuantas reglas simples y será de oro. Recordemos: `alloc/init/copy/new` en los métodos devuelven sus propias referencias y requieren una liberación cuando hayamos acabado con el objeto. Métodos sin `alloc/init/copy/new` devuelven objetos autoliberados, y esto significa que será liberado en el futuro. Si necesita utilizar un objeto autoliberado durante mucho tiempo, asegúrese de mantenerlo (y posteriormente liberarlo). EN este sentido, recuerde mantener objetos autoliberados en el momento de crear sus propios métodos.

Reutilizar Identificadores (Reuse Identifiers)

Un *identificador reutilizado (reuse identifier)* permite la reutilización de table views reutilizando antiguas celdas sin tener que hacer nada más. Es una técnica inteligente utilizada por programadores eficientes que crean table views rápida y fácilmente. Y sí, es el reciclaje digital!

Una vez que tenemos una celda viable, tenemos que cambiar la etiqueta del texto. Llamaremos a `objectAtIndex:` para acceder a nuestra matriz, pasándole `[indexPath row]` para el índice. Este obtendrá el objeto que necesitemos para la fila en la tabla (el table view que se esta mostrando). A continuación, establecemos el texto de la celda del Text Label con el texto que queremos desde la matiz “categories”, y después volvemos a la celda para ser mostrada.

Sí, lo sé, es un torbellino de variables, intercambios, cuentas y liberaciones. Lo entenderá, sin embargo. Recuerde: esto es sólo una rápida visión - un postre después de este sabroso capítulo.

Capítulo 9

MapKit

He esperado a escribir el capítulo sobre MapKit desde el momento que concebí este libro por primera vez. Este es el último capítulo y nuestro viaje juntos casi ha terminado. Estoy seguro de que este tema no te defraudará.

Durante el transcurso de este capítulo verás que algunas de las aplicaciones más interesantes y de éxito se basan en lo que llamamos el framework MapKit. El motivo de dejar Mapkit para el último capítulo es porque precisa la mayor base teórica posible a fin de no agobiar al estudiante.

A pesar de que MapKit nos proporciona los medios para programar aplicaciones potentes y vivas, también nos exige que seamos muy conscientes de cuales son los métodos, clases y frameworks. Originalmente, el alcance de este libro no incluía cubrir todos estos aspectos, pero no había manera de que pudiera dejar de lado el MapKit!

Así que, antes de empezar, tenemos que sentarnos y mirar un par de cosas. MapKit, como conjunto de herramientas, constituye un gran desafío, pero te mostraré algunos conceptos básicos y enseñaré como usarlos para navegar con éxito y creatividad en el ejemplo de este capítulo.

En primer lugar hablaremos de los *frameworks* y las *clases*. A continuación, veremos lo que MapKit es capaz de hacer por nosotros sin tener que programar nada. Luego profundizaremos para ver lo que otros programadores han hecho con MapKit, y recopilar lo que podamos de ellos. Después de perfeccionar la compresión de los métodos, y una vez adquirido un buen conocimiento de los frameworks, clases, y otras chucherías de Apple contenidos en el MapKit, pasaremos a abordar –*con paciencia*- el ejercicio.

En la segunda mitad del capítulo. Serviré un extenso postre en la sección "Profundizando en el código MapKit de mis alumnos". En lugar de acabar con una mezcla ecléctica de referencias técnicas, presentaré tres de los proyectos realizados por mis alumnos relacionados con el MapKit. Espero que cuando vea lo que estos alumnos fueron capaces de lograr poco después de pasar por mi clase, te sientas aún más inspirado.

Mi objetivo, entonces, que lleguéis al punto donde podáis decir: *He programado una aplicación MapKit básica para iPad, y entiendo como avanzar con confianza en el territorio más avanzado*.

Acerca de los Frameworks

Cuando Steve Jobs fue despedido de Apple, formó una empresa llamada NeXT. Su compañía produjo computadoras bonitas, negras y aerodinámicas en los 90, lo que me hizo babear con envidia, ya que unos pocos de mis profesores poseían un ordenador NeXT. El aspecto más profundo de este equipo no fue que ellos utilizaran esas cajas negras y aerodinámicas, sino que utilizaron un lenguaje llamado Objective-C. Jobs consideró que, aunque era difícil de programar en ese lenguaje tan complejo, el código creado fue capaz de "hablar" elegantemente con el microprocesador. ¿Y qué tiene que ver esto con MapKit?!

Lo que NeXT hizo fue crear *frameworks* de código de Objective-C, lo que pueden considerarse como las herramientas que un carpintero tiene en su caja de herramientas. Cuando usamos MapKit, en realidad estamos introduciendo a nuestro código un conjunto de herramientas relacionadas entre sí, igual que un carpintero tiene un conjunto de herramientas para ebanistería y otro conjunto, especialmente hecho para muebles. Estas herramientas serán muy diferentes al tipo de herramientas que un carpintero usa para el tejado.

Con este fin, introduciremos en Xcode dos frameworks que no hemos usado antes. Será casi como si hubieras estado aprendiendo técnicas para carpintería de muebles y suelos en los capítulos, pero hoy vamos a ir a la tienda de hardware a comprar equipamiento para nuestro próximo concierto: instalaciones de audio y video en paredes y techos. Por lo tanto, antes de ir a nuestro próximo programa, vamos a comprar dos nuevas herramientas. Una de ellas, las CoreLocation framework, nos mostrará donde nos encontramos geográficamente. Y la otra herramienta, MapKit, nos permitirá interactuar con los mapas de muchas formas.

Como sabes, la forma de interactuar con el Ipad y el iPhone es completamente diferente a cualquier otro dispositivo visto antes. Antes de la llegada de estos dispositivos, el 99% de todas las interacciones que teníamos instaladas en los equipos se basaban en el ratón y el teclado. Sin embargo, en los ejemplos que has programado, hemos utilizado métodos y clases únicos para saltar entre las pantallas y detectar cuándo un usuario está pellizcando, marcando, o el desplazándose por la pantalla. A este conjunto formidable de herramientas, vamos a añadirle el CoreLocation y el framework MapKit.

La mayor parte de la programación que hemos explorado ha sido relativamente transparente. Este capítulo no lo será tanto. Por ejemplo, tendremos que mantener nuestro enfoque con el fin de realizar un seguimiento de cómo MapKit sabe dónde estamos en un mapa. Analizaremos el modo en que ejecuta las interacciones con el dispositivo que hacemos con los datos y cómo este sabe dónde estamos en las diferentes pantallas y vistas relacionadas con los mapas.

Una de las áreas centrales del desarrollo de aplicaciones para Ipad/iPhone es el *control de eventos (event handling)*. Esta es la parte que más confusión ha creado en mis alumnos cuando llegamos a esta lección, por lo que haré todo lo posible para evitar que te vayas por las ramas. Puedes hacerte una idea del alcance de este tema, considerando que, si bien parte de tu aplicación es el seguimiento de la interacción con el mapa y con un satélite GPS, otra parte de tu código ha de estar siempre mirando cuando el usuario va a dirigir el programa a un nuevo evento.

Cosas que debes saber

Hay tres cosas importantes a saber sobre los fundamentos de las aplicaciones relacionadas con los mapas en el ámbito del iPad e iPhone. Las aplicaciones se basan en tres herramientas importantes: MapKit, CoreLocation, y la referencia de clase `MKAnnotationView`. Como ya he indicado, no vamos a involucrarnos con la forma de trabajar de estos sofisticados instrumentos de trabajo tanto como para practicar el arte de decidir que herramienta utilizar de tu caja de herramientas recientemente ampliada.

Entre otras cosas, estas herramientas nos permiten visualizar mapas de nuestras aplicaciones, para usar anotaciones, para trabajar con lo que se denomina geocodificación (que trabaja con la longitud y latitud), e interactuar con nuestra ubicación (a través de CoreLocation).

Cuando queremos interactuar sin esfuerzo con Google Maps, utilizaremos el framework Apple-provided MapKit. Cuando queremos que nuestre nuestra ubicación o hacer cosas interesantes con tecnología GPS por satélite (con Google Maps), utilizaremos el framework CoreLocation. Por último, cuando queramos poner chinchetas de localización en un mapa, crear referencias, introducir iconos, o insertar una imagen de tu perro que muestre dónde está en un mapa, podremos estas anotaciones y, por tanto, usamos MKAnnotationView.

Aplicaciones MapKit instaladas

Para que puedas aprovechar al máximo las nuevas ideas presentadas en este capítulo, y estar preparado para expandirse a un nuevo nivel de creatividad, primero daremos una pequeña vuelta por las aplicaciones existentes, preinstaladas en el iPad y el iPhone. Es importante que te familiarices con estos conceptos para que puedas añadir más fácilmente las campanitas y avisos a tu propia creación en la parte superior de estas “aplicaciones mapa” ya hechas, como se describe en `Apple.com`.

Buscar ubicaciones

La aplicación buscar ubicaciones (ver Foto 9–1), preinstalada en el iPhone 3GS e iPad, encuentra tu ubicación con rapidez y precisión a través de GPS, Wi-Fi y 3G. Deja una chincheta para marcar tu posición y compartirlo con otros a través de correo electrónico o MMS

Foto 9–1. Buscar ubicaciones, una potente función de zoom en el mapa del iPhone / iPad

Como llegar

Se muestra en la Foto 9-2, la aplicación Como llegar preinstalada te permite ver una lista de direcciones giro a giro o seguir una hoja de ruta y rastrear tu progreso con el GPS. Puedes especificar si deseas caminar o conducir, o ver a qué hora sale el próximo tren o autobús con las instrucciones de tránsito público.

Foto 9–2. Como llegar – se usa en conjunto, o en lugar de, el mapa visual (con la ruta resaltada).

Ver hacia donde estas orientando

En la aplicación “Ver hacia donde estás orientado” preinstalada que se muestra en la Foto 9-3, una brújula digital gira los mapas de manera que siempre coincida con la dirección en que estamos enfrentados. También puede utilizar la brújula por sí sola.

Foto 9–3. Ver hacia donde estás orientado – muestra la orientación con la brújula integrada (en el modelo 3GS)

Ver el tráfico

LA aplicación del iPhone preinstalada “Ver el tráfico”en la Foto 9–4, muestra la información del tráfico en directo, indicando la velocidad del tráfico a lo largo de tu ruta facilitando la lectura, destacándolos en verde, rojo y amarillo

Foto 9–4. Ver el tráfico - es una de las muchas posibilidades cuando se ejecutan 'Mapas' en el iPhone / iPad.

Buscar una ubicación

En el modo “Buscar una ubicación”, mostrada en la Foto 9–5, puedes buscar ubicaciones por dirección o por palabra clave. Por ejemplo, busca por “cafetería” para ver las cafeterías más cercanas. Cuando encuentres lo que buscas, pulsa el número de teléfono para llamar (en el iPhone), toca en la dirección web para abrir la página web en Safari, o añádelo a la Guía para una referencia futura.

Foto 9–5. El modo “Buscar una ubicación” es una característica rápida y potente a su alcance que aporta información específica sobre el destino.

Cambiar la vista

La aplicación preinstalada “Cambiar la vista”, mostrada en la Foto 9–6 te permite cambiar entre vista de mapa, vista satélite, vista combinada, y vista en la calle. Puedes hacer doble toque o pellizcar para ampliar y reducir.

Foto 9–6. El modo “Cambiar la vista”es una característica estándar de “Mapas” en el iPhone/iPad.

Aplicaciones MapKit interesantes para inspirarte

Me di cuenta que realmente ayudaba a mis estudiantes cuando, después de enseñarles las aplicaciones preconstruidas, pasábamos algún tiempo revisando algunas aplicaciones Mapkit de terceras personas… para inspirarlos y realizar un brainstorming. Por tanto, imagínate que estás hacienda con nosotros este pequeño recorrido. Te presento nueve aplicaciones MapKit que llamaron mi atención, algunas de las cuales utilizo muy a menudo.

- **MapMyRide:** Esta es una aplicación MapKit que uso a diario. Hago uso de ella cuando voy en bicicleta. Controla la velocidad, tiempo, kilometraje, e incluso el desnivel que realizo. Tiene en cuenta mi edad, sexo y peso corporal, y me indica cuantas calorías he quemado. [En un buen día, casi llego a quemar dos donuts!] La aplicación calcula todas estas cosas mientras voy como loco dando pedales y resoplando! Cuando llego a casa, puedo comprobar el recorrido que he hecho en mi ordenador. Esta aplicación realiza la mayoría de este trabajo usando y manipulando aplicaciones MapKit preinstaladas.

- **QuikMaps:** Esta es una aplicación de mapas "hágalo usted mismo" que te permite garabatear sobre un mapa. Se integra con una serie de lugares, incluyendo tu sitio web, Google Earth o incluso tu GPS.

- **360 Cities—The World In Virtual Reality:** Muestra panorámicas de 360º de unas 50 ciudades del mundo. Es una tecnología ideal para agentes inmobiliarios, guías turísticos y aventureros.

- **Cool Maps—7 Wonders of the World:** Muestra las siete maravillas del Mundo Antiguo, y las siete maravillas del Mundo Moderno, incluyendo las maravillas naturales, maravillas submarinas, maravillas extrañas y maravillas locales. Me ha impresionado la habilidad que han tenido sus programadores para la sensación y el tacto de la aplicación.

- **Blipstar:** Esta aplicación utiliza direcciones URL de negocios de internet para situarlas en un callejero, presentándolas en un mapa muy interesante.

- **Twitter Spy:** Esta aplicación el permite a la gente ver la localización de las personas con las que están Twitteando.

- **Geo IP Tool:** Esta aplicación muestra información sobre la latitud y longitud de negocios en la web, y muestra el mejor camino para llegar a los mismos.

- **Map Tunneling Tool:** Esta aplicación es divertida e inteligente. Imagina en qué sitio del globo pararías si empezases a hacer un agujero en la tierra. ¿Por qué siempre l*a respuesta es China*?

- **Tall Eye:** Esta aplicación te muestra dónde irías a parar si empezases a andar en línea recta a lo largo del mundo.

MapKit_01: A View-Based Application

Para el ejercicio final, vamos a comenzar con una serie de código preprogramado que cubrirá nuestras necesidades más básicas, y procederemos a modificarlo. Te guiaré a través de las líneas de código que hemos visto a lo largo de este libro y te retaré a que identifiques qué partes de código son parecidas a las que hemos ido viendo en el libro y qué partes son diferentes –dada la naturaleza de la aplicación.

La habilidad de reconocer patrones y estructuras para hacerse con una visión que se sitúe por debajo de la superficie es una aptitud muy interesante que todos tenemos, pero nosotros, los programadores, la desarrollamos de un modo mayor. Por tanto, estate atento a las frases, afirmaciones, sufijos y prefijos –giros gramaticales, por así decirlo- que puedes construir, codificar y/o revisar. Juega a un pequeño juego: intenta anticiparte a algunos de las acciones que voy a llevar a cabo.

Posible Preparación para la Aplicación

Vamos a considerar una amplia variedad de componentes que queremos insertar en nuestra aplicación. Antes de ello, quiero asegurarme de que todos estamos al mismo nivel con la terminología. Nosotros, los programadores, necesitamos tener claros conocimientos básicos de ciencia y geografía para que nuestro código sea lo más efectivo posible.

Cuando le ordenamos a la computadora que muestre una chincheta en pantalla mostrando una serie de anotaciones concernientes a una ubicación específica, dando la latitud y longitud de un emplazamiento, necesitamos conocer qué significan realmente estos conceptos. Las líneas de *latitud* son líneas imaginarias que rodean horizontalmente el globo terráqueo, desplazándose de este a oeste (o de oeste a este, según se mire). Estas líneas invisibles son medidas en grados, minutos y segundos, con respecto al norte o el sur del Ecuador. El Ecuador es la línea elíptica que divide el globo en dos partes exactas con respecto a los polos, que a su vez son puntos físicos reales definidos por la rotación de la Tierra sobre su eje. Muchas veces hacemos referencia a las líneas de Latitud como “Paralelos”. El Polo Norte está a 90º latitud norte; el Polo Sur está a 90 grados latitud sur.

Por otro lado, las líneas de *longitud*, normalmente llamadas “meridianos” son líneas imaginarias verticales (elipses), las cuales cruzan siempre los por los Polos Norte y Sur. También se miden en grados, minutos y segundos, al este u o oeste del Primer Meridiano, el cual se estableció por consenso, siendo el que cruza a través de Greenwich, Inglaterra. A diferencia del Ecuador, el cual es una elipse que da la vuelta al globo, el Primer Meridiano (0 grados longitud) es un semicírculo o semielipse, que se extiende del Polo Norte al Polo Sur, siendo la otra mitad de elipse el arco llamado International Date Line, el cual está definido como longitud 180 grados este o 180 grados oeste.

Para nuestra aplicación del Capítulo 9, el ejemplo que he usado como lugar de localización, es mi oficina en la universidad de Colorado, en Colorado Springs. Tú, por descontado, puedes usar la localización que quieras, como por ejemplo tu dirección, o algún lugar que te guste en especial. Para hacer esto, debes conseguir las coordenadas de latitud y longitud de esa localización, las cuales pueden extraerse de Google Maps o directamente de la lectura de un GPS. También hay muchos sitios en Internet donde puedes conseguir estas coordenadas; La Foto 9–7 muestra uno de ellos, en este caso, `www.batchgeocode.com`.

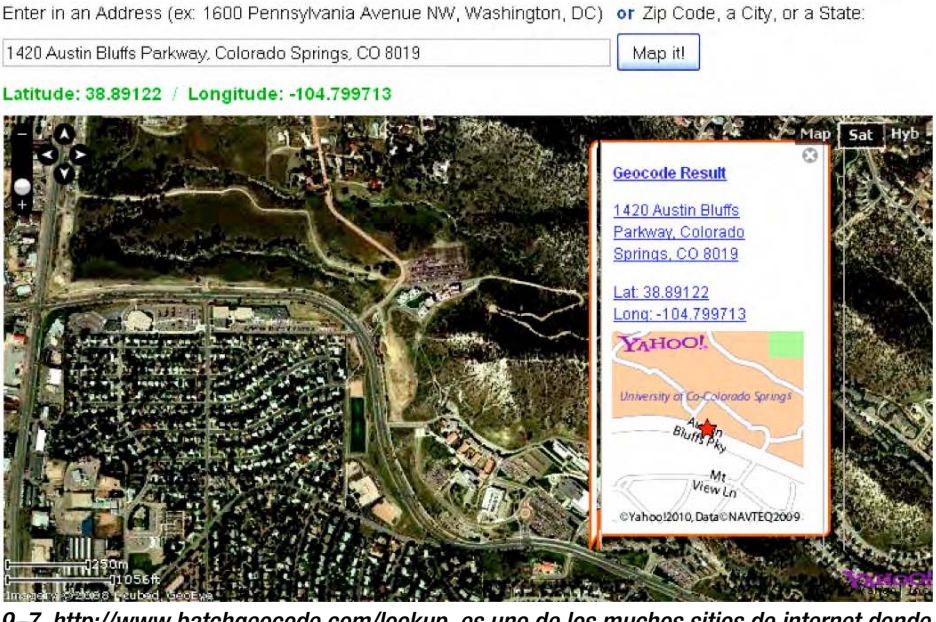


Foto 9–7. http://www.batchgeocode.com/lookup es uno de los muchos sitios de internet donde uno puede introducir una dirección y recibir sus correspondientes coordenadas de latitud y longitud.

Sigue el siguiente planteamiento… Empecemos por lo que sería el final de proceso y vayamos hacia atrás y pensemos durante un minuto. Vaya hacia delante y haga un salto en este capítulo para obtener un vistazo, una idea sobre cómo se verá la aplicación y qué resultados arrojará si todo va bien. En la Foto 9-25, se ve una foto de un mapa hibrido el cual muestra una chincheta roja situada en lo alto de un edificio. Ese edificio es el Edificio de Ingeniería en la Universidad de Colorado en Colorado Springs, y esa chincheta está dispuesta a la derecha, justo encima de mi oficina. La siguiente foto tiene lo que nosotros llamamos una *anotación* (annotation), con el siguiente texto “Dr. Rory Lewis” es el título, y “Universidad de Colorado en Colorado Springs” es el subtítulo.

Más tarde en el tutorial, verás que debemos ser cuidadosos sobre lo que es el título y lo que es el subtítulo, Controlaremos el color la chincheta de posición y decidiremos el estilo de animación, es decir, el cómo aparece este pin en pantalla.

En la Foto 9–27, alejaremos el modo vista y veremos el modo pantalla desde el espacio. Esto nos permite situar la localización en bastas regiones, eso sí, a cambio de perder nivel de detalle. También muestra la diferencia entre la localización del usuario (punto azul) y la localización subrayada (punto rojo). Curiosamente, el simulador iPad asume que el usuario se encuentra en Cupertino, CA, la ubicación de la Sede Central de Apple.

Este es un buen momento para recordarte el título de este libro: *Aplicaciones iPhone e iPhad para Auténticos Principiantes* (iPhone and iPad Apps for Absolute Beginners). ¡Relájate! Incluso si se cumpliesen mis mayores expectativas con respecto a que hayas aprendido la mayor parte de conceptos que te he impartido, no te convertirías en un experto en el área de MapKit. Mi objetivo aquí no es conseguir una gran fluidez, pero sí que consigas una familiaridad razonable con los conceptos y obtener comprensión sobre lo que vamos a ir viendo.

Esto suena bien, vamos a seguir con nuestro ejercicio.

Preliminares

Como en los capítulos anteriores, por favor descarga las imágenes y código para este capítulo. Accede a `http://www.rorylewis.com/xCode/011_MapKit_01.zip` y descarga el contenido. Después, extrae el contenido en tu escritorio.

El archivo zip contiene diferentes carpetas: `MapKit_01.xcodeproj`, `Clases`, y `build`, y algunos archivos individuales: `MapKit_01ViewController.xib`, `MapKit_01-Info.plist`, `MainWindow.xib`, `MapKit_01_Prefix.pch`, y `main.m`.

Una vez extraídos los archivos, recuerda borrar las carpetas 011_MapKit_01.zip y MapKit_01. No queremos que ninguno de tus archivos sean sobreescritos con tu código del ejercicio.

Para ver el screencast del ejercicio del capítulo, vete a:

`http://www.rorylewis.com/docs/02_iPad_iPhone/06_iphone_Movies/011_MapKit.htm`.

Una nueva plantilla View-Based

1 Abre Xcode y teclea ⌘⇧N, como muestra la Foto 9–8, y clica en la plantilla View-based Application. Llamalo `MapKit_01` y guardalo en tu escritorio. Una capeta con ese nombre aparecerá en tu escritorio.

2 Lo primero que debemos hacer son dos frameworks: CoreLocation y MapKit. Botón derecho sobre la carpeta `Frameworks`, pulsa Add y luego Existing Frameworks, luego, en el menú desplegable que aparece, selecciona ambos frameworks CoreLocation y MapKit, como muestra la Foto 9–9.

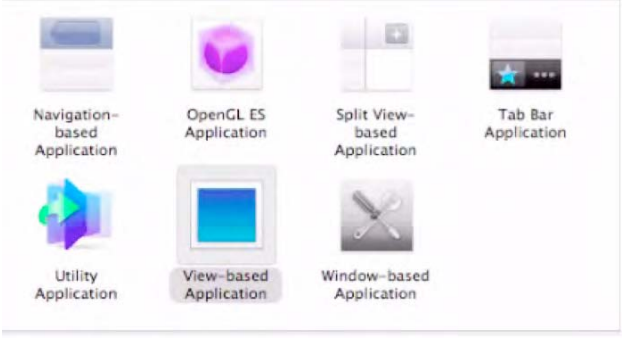


Foto 9–8. Selecciona el icono View-based Application, y guarda el archivo en tu escritorio.

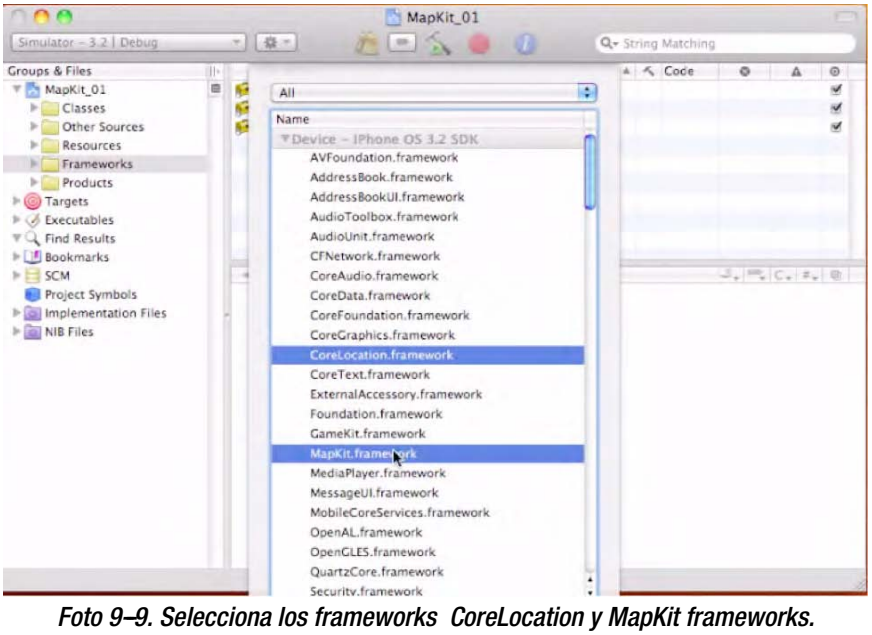


Foto 9–9. Selecciona los frameworks CoreLocation y MapKit frameworks.

Añadiendo el archivo de anotaciones

3. Lo siguiente que necesitamos es una manera de controlar las anotaciones (acuérdate que las anotaciones son las leyendas que aparecen al pulsar sobre una chincheta de posición, dando información sobre la ubicación). Para esto, Crearemos una subclase UIViewController que controlará todas las características que queremos mostrar en esta anotación. Clic en la carpeta `Classes` y pulsa ⌘N como muestra la Foto 9–10. Este controller será encargado de controlar las anotaciones para tu posición, por tanto, un buen nombre para el mismo es "myPos."

Asegúrate de que tus opciones *no están marcadas*. Estas incluyen Targeted for iPad, UITableViewController Subclass, y XIB Interface. Esto creará automáticamente un archivo myPos header y un archivo de implementación en tu carpeta de `Classes` una vez pulsado el botón siguiente o Return.

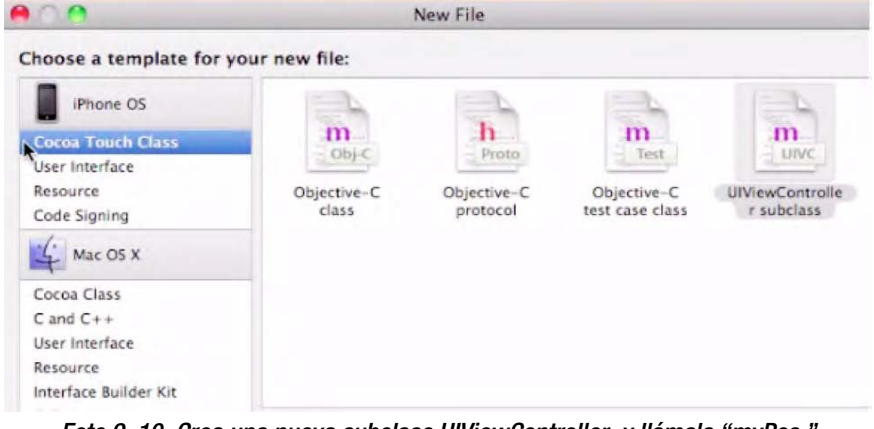


Foto 9–10. Crea una nueva subclase UIViewController, y llámala "myPos."

Ya funciona!

Te lo creas o no, ya tenemos lo suficiente para mostrar un mapa funcionando en nuestro iPad o iPhone. Como dije en la introducción de este capítulo, los programadores de Apple han incluido una enorme cantidad de código Objective-C en los dos frameworks que importamos.

4. Para examinar los detalles, vamos a abrir el archivo nib yendo a la carpeta Resources y abriendo el archivo `MapKit_01ViewController.xib`. Clic en él como muestra la Foto 9–11

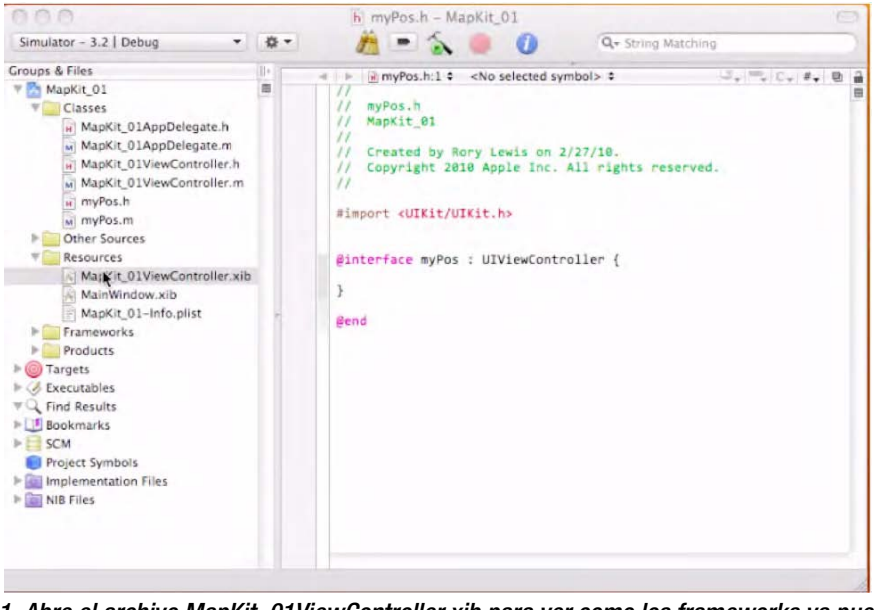


Foto 9–11. Abre el archivo MapKit_01ViewController.xib para ver como los frameworks ya pueden ejecutar un mapa que funcione.

Check It Out—the iPad Simulator

5. Después de abrir Interface Builder, coge un objeto Map View de la Biblioteca y déjalo en View. Luego vete a Inspector y clica en Show User Location. Hecho esto, pulsa ⌘S para guardar, y vuelve a la carpeta Clases en Xcode. Clic en algún archivo y pulsa ⌘Return, y aparecerá el Simulador del iPad, mostrando un mapa del mundo. Seguidamente, una marca azul aparecerá sobre Cupertino, CA, como ejemplo de tu ubicación "actual".

NOTA: Dado que no estás operando en un iPhone o iPad, el simulador cree que tu Localización Geocode es la de las Oficinas centrales de Apple en Supertino, CA, el corazón de Silicon Valley. Una vez tu aplicación – en este estado de desarrollo- se coloque en un dispositivo real, ya sea iPhone o iPad, el sistema GPS del dispositivo usará el framework `Corelocation` que importaste en el Paso 2 para obtener tu ubicación actual.

Tu pantalla inicialmente podría parecer ligeramente diferente porque, por defecto, mostrará la pantalla del iPhone encajada en el Simulador del iPad. Para hacer que la pantalla en el iPad se parezca exactamente a mi pantalla en la Foto 9–12, simplemente clica en el botón Enlarge en la esquina superior derecha de la pantalla del iPad.

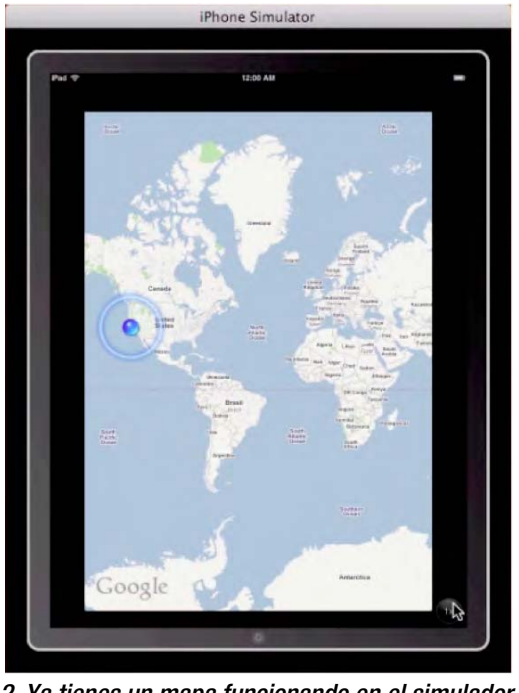


Foto 9–12. Ya tienes un mapa funcionando en el simulador del iPad.

Hazlo que parezca un poquito mejor

Aunque tenemos un mapa en nuestro iPad donde nuestro framework CoreLocation funciona, queremos hacerlo un poquito mejor de lo que es. Especialmente, nos gustaría que muestre la ubicación de un punto que escojamos, y también donde queremos que ponga una chincheta, y que incluya una anotación en la cual el título y el subtítulo explique algo sobre nuestra ubicación. Esta información aparecerá en una pequeña caja negra cuando el usuario haga clic sobre la chincheta, del color que escogimos.

6. Comenzaremos declarando nuestras clases, métodos, y outlets en nuestro archivo `ViewController.h` y luego implementándolos en nuestro archivo `ViewController.m`. En este punto, haremos algunos ajustes a los archivos `myPos` y `AppDelegate`.

7. Abre tu archivo de encabezado haciendo clic sobre `MapKit_01ViewController.h` como muestra la Foto 9–13. Lo primero que debemos hacer es decirle a nuestra aplicación que hemos importado el framework MapKit; lo hacemos insertando `#import <MapKit/MapKit.h>` debajo de la línea `#import <UIKit/UIKit.h>`. En el siguiente paso lo que haremos es decirle al archivo de encabezado que estaremos usando el protocolo `MKMapViewDelegate`. Este protocolo define un conjunto de métodos opcionales que usará nuestra aplicación para recibir registros actualizados.

8. Ahora además debemos añadir un outlet con un puntero a la clase MKMapView. Hacemos esto escribiendo `IBOutlet MKMapView *mapView`, el cual declara un objeto del tipo `MKMapView`. Por último lo que necesitamos es definir la @property, introduciendo

```
@property (nonatomic, retain) IBOutlet MKMapView *mapView;

@end
```

Tu código, en este punto, debería parecerse a lo siguiente:

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>

@class MyPos;

@interface MapKit_01ViewController : UIViewController <MKMapViewDelegate>
{
      IBOutlet MKMapView *mapView;
}

@property (nonatomic, retain) IBOutlet MKMapView *mapView;

@end
```

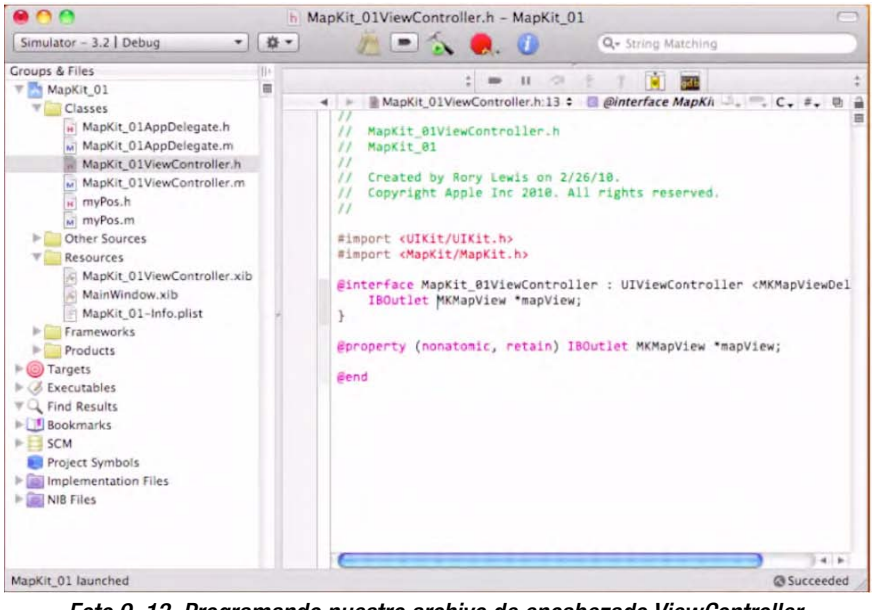


Foto 9–13. Programando nuestro archivo de encabezado ViewController.

Procediendo con la implantación

Como dije en la introducción de este capítulo, controlar y trabajar con los frameworks MapKit y CoreLocation no es algo trivial. Dada su importancia, no pude dejarlos fuera de este libro. Seguimos sobre la base que hemos aprendido hasta ahora para buscar ejemplos familiares, integrar lo que sabemos, y tratar de seguir el camino cuando las cosas se ponen un poquito complicadas!

NOTA: Lo que vamos a hacer en nuestro archivo de implementación es sintetizar nuestra propiedad y lanzarla en el método `dealloc`.

9. Como muestra la Foto 9–14, Ahora copiamos la declaración dealloc de la parte inferior del archivo de implementación, `MapKit_01ViewController.m`. Vamos a pegarlo en la parte superior del archivo justo debajo de la declaración que acabamos de escribir. Luego añade un release para la propiedad mapView, como muestra la Foto 9–15.

La razón por la que te dije que lo pusieras justo en la parte superior es que ahora queremos borrar todo, por debajo del método `viewDidLoad`. Ver Foto 9–16.

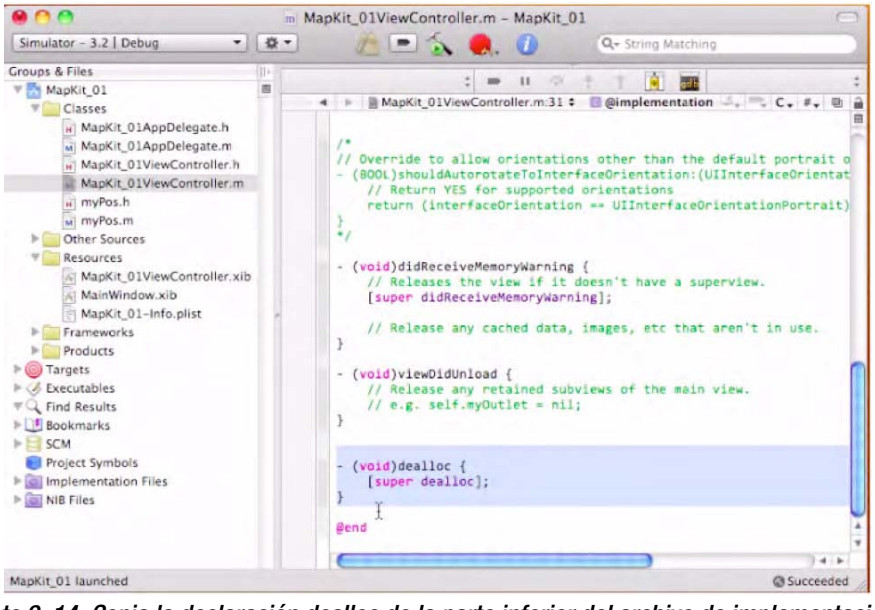


Foto 9–14. Copia la declaración dealloc de la parte inferior del archivo de implementación.

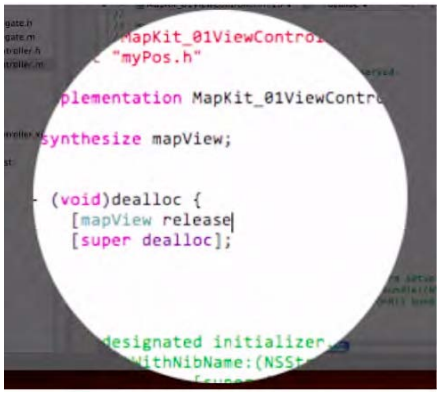


Foto 9–15. Pégalo en la parte superior justo debajo de la declaración mapView. También, añade un release para la propiedad mapView.

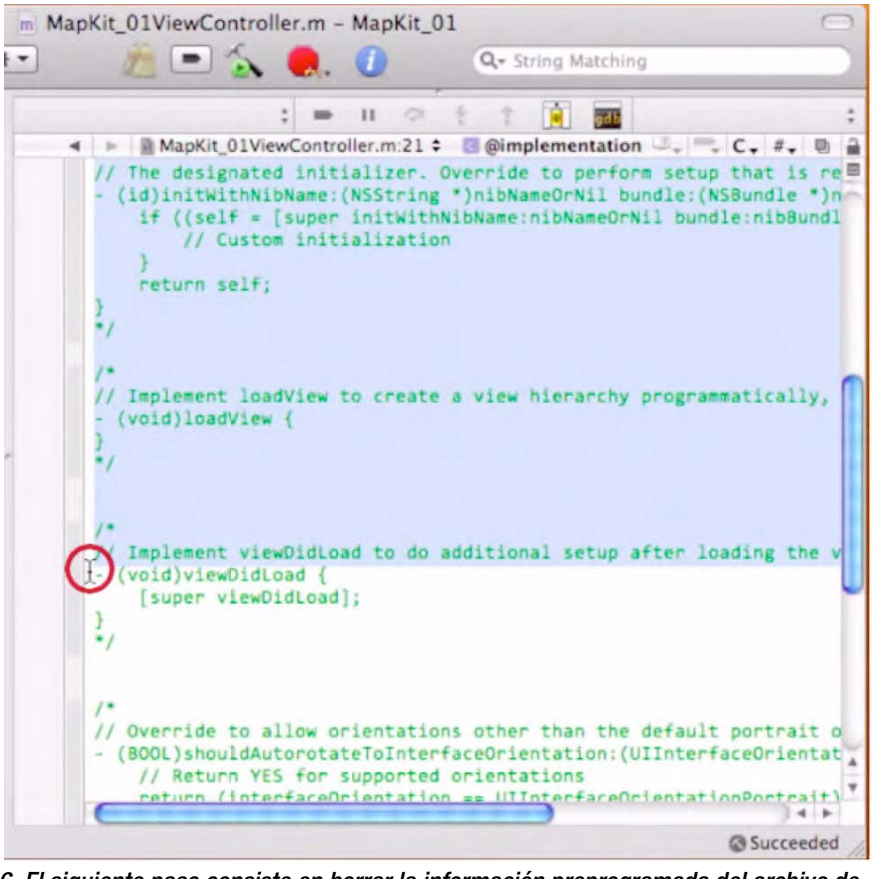


Foto 9–16. El siguiente paso consiste en borrar la información preprogramada del archivo de implementación, hasta la línea (void)viewDidLoad {.

Meditemos sobre esto... Necesitamos añadir el método `viewDidLoad`, con el cual ajustaremos el tipo de mapa y habilitaremos la propiedad de zoom, desplazamiento y algunas otras más. En nuestro caso, ajustaremos el tipo de mapa a Mapa Híbrido. También podrás hacer uso del Mapa Satélite o Mapa Callejero.

Llegado a este punto, vamos a introducir en el la localización que se nos solicita, algún sitio de interés o de significado personal. Si no tienes ninguna preferencia especial, puedes usar mi localización, la cual uso en el ejemplo, tal y como se muestra en el código siguiente.

10. Lo primero que tenemos que hacer es ajustar todas las coordenadas regionales a ceros. Luego introduciremos las coordenadas de nuestro sitio elegido, en mi caso, mi despacho en la Universidad de Colorado en Colorado Springs. Yo he introducido `Region.center.latitud = 38.893432`; (el valor positivo denota que el emplazamiento está al norte del Ecuador) y `region.center.longitude = -104.800161`; (el signo negativo denota que estamos al oeste del Meridiano Principal). En relación a estos parámetros necesitamos ajustar la Latitud y Longitud Delta = 0.01f. Si estás "pegado" en matemáticas o física, recuerda que "delta" hace referencia al cambio o diferencia entre dos valores. Finalmente, para el `viewDidLoad`, he escogido una animación para la chincheta que marque la localización usando el código `[mapView setRegion:region animated:YES].` Ver Foto 9–17.

11. Nuestro siguiente paso es ajustar la clase de vista de controlador como delegada, el papel que se encargará de la interacción entre los frameworks de nuestro mapView. Esto lo conseguimos con `[mapView setDelegate:self].`

En relación a la chincheta que marca la posición y la correspondiente etiqueta que da la información del lugar, necesitamos hacer que el Annotation Object sea el poseedor de la información de nuestras coordenadas. Nuestro Annotation View es el tipo de vista asociado con el Animation Object. Este Annotation Object necesita que se cumplan todas las reglas que estableceremos en nuestro Protocolo MKAnnotation. Con objeto de crear este Annotation Object, debemos de definir una nueva clase, tal y como hicimos cuando creamos la clases `myPos`.

12. Ahora tenemos que ejecutar este objeto myPos y agregarlo a nuestro mapa. Para ello, añadimos la función delegada que mostrará las anotaciones en nuestro mapa. Empezamos haciendo que myPos nombre a un pointer al que llamaremos "ann". Luego ajustaremos el título; en mi caso he escogido mi nombre. Pon tanto, tenemos `ann.title = @"Dr. Rory Lewis".` Nos encargamos del subtítulo del mismo modo: `ann.subtitle = @"University of Colorado at Colorado Springs".` También queremos que la chincheta se disponga en el centro del mapa, esto lo hacemos así: `ann.coordinate = region.center`. Referenciamos todo lo anterior con `[mapView addAnnotation:ann]`.

Ahora vamos a hacer uso de un código preprogramado que la mayoría de los mapas MapKit utilizan.

Por ahora, he incluido este código preprogramado en la carpeta Downloads para esta aplicación. Este código hace dos cosas:

NOTA: Rara vez cambiamos estas porciones de código, y para cuando estés leyendo este libro, el próximo conjunto de códigos pueden pasar a ser parte de una nueva función o una nueva clase. El motivo es que cuado la gente comienza a utilizar la misma pieza de código una y otra vez, se refiere a ella como "código preprogramado", dado que Apple puede decidir el crear una nueva clase o función dentro de ella, o establecerle un nombre específico.

- Crea un *método delegado* que gestiona nuestra anotación (la etiqueta que hace las veces de leyenda) durante el zoom y el desplazamiento, en otras palabras, mantiene la situación de donde estamos – incluso en caso de que el usuario haga un scroll o zoom fuera del área en cuestión.

- Crea un *identificador estático* el cual controla nuestro "queue meaning." Si no se puede quitar de la cola nuestra anotación, asignará la que nosotros escojamos. También he incluido el código que permite cambiar de color la chincheta, en este caso a rojo. También, he permitido vistas de llamada (*callout views*).

Asegura que tu `- (BOOL)shouldAutorotateToInterfaceOrientation` está activado junto con tu `- (void)didReceiveMemoryWarning`. Tu código debería quedar del siguiente modo:

```
#import "MapKit_01ViewController.h"
#import "MyPos.h"

@implementation MapKit_01ViewController
@synthesize mapView;

-(void)dealloc
{
      [mapView release];
   [super dealloc];
}
-(void)viewDidLoad
{
   [super viewDidLoad];

      [mapView setMapType:MKMapTypeStandard];
      [mapView setZoomEnabled:YES];
      [mapView setScrollEnabled:YES];

      MKCoordinateRegion region = { {0.0, 0.0 }, { 0.0, 0.0 } };
      region.center.latitud = 38.893432 ;
      region.center.longitude = -104.800161;
      region.span.longitudeDelta = 0.01f;
```

```
        region.span.latitudDelta = 0.01f;
        [mapView setRegion:region animated:YES];

        [mapView setDelegate:self];

        MyPos *ann = [[MyPos alloc] init];
        ann.title = @"Dr. Rory Lewis";
        ann.subtitle = @"University of Colorado at Colorado Springs";
        ann.coordinate = region.center;
        [mapView addAnnotation:ann];
}
-(MKAnnotationView *)mapView:(MKMapView *)mV viewForAnnotation:↩
(id <MKAnnotation>)annotation
{
        MKPinAnnotationView *pinView = nil;
        if(annotation != mapView.userLocation)
        {

                static NSString *defaultPinID = @"com.invasivecode.pin";
                pinView = (MKPinAnnotationView *)[mapView↩
dequeueReusableAnnotationViewWithIdentifier:defaultPinID];
                if ( pinView == nil )
                        pinView = [[[MKPinAnnotationView alloc]↩
initWithAnnotation:annotation reuseIdentifier:defaultPinID] autorelease];

                pinView.pinColor = MKPinAnnotationColorRed;
                pinView.canShowCallout = YES;
                pinView.animatesDrop = YES;
        }
        else
        {
                [mapView.userLocation setTitle:@"I am here"];
        }
    return pinView;
}
-(BOOL)shouldAutorotateToInterfaceOrientation:↩
(UIInterfaceOrientation)interfaceOrientation
{
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
 -(void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
}
-(void)viewDidUnload
{
}

@end
```

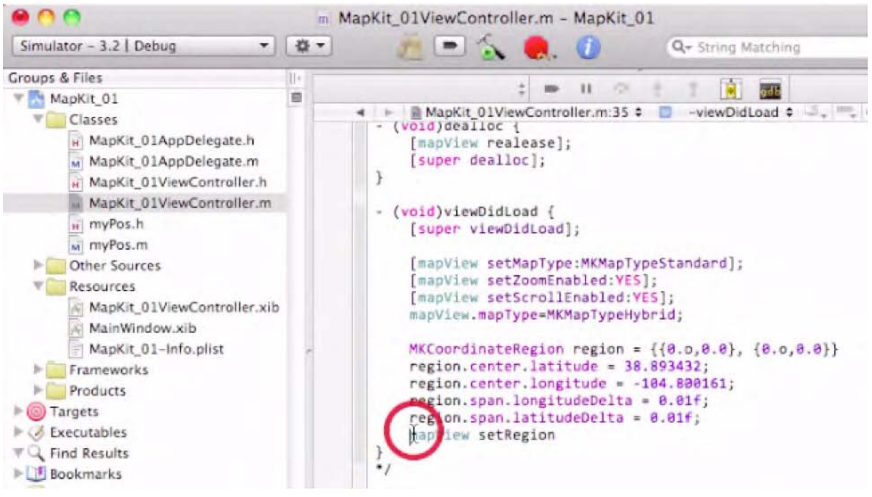


Foto 9–17. Añadiendo coordenadas, parámetros y ajustes de zona.

Codificando el Archivo the myPos.h

13. Lo primero que haremos después de abrir el archivo `myPos.h` es reemplazar la línea de importación `#import <UIKit/UIKit.h>` por `#import <Foundation/Foundation.h>`. Lo siguiente será el añadir el MapKit introduciendo `#import <MapKit/MKAnnotation.h>`.

14. Ajustamos nuestra Clase de Referencia (Class reference) CLLocation para incorporar las coordenadas geográficas y altitud de nuestro dispositivo, como se ve en la Foto 9–18. Esto lo haremos mediante la siguiente línea:

```
CLLocationCoordinate2D coordinate; NSString *title; NSString *subtitle
```

15. Finalmente creamos la declaración “@property” para la localización, título y subtítulo, como apreciarás en el siguiente código. Una vez hayas añadido esto, guarda tu trabajo. Mira la Foto 9–19.

```
#import <Foundation/Foundation.h>
#import <MapKit/MKAnnotation.h>

@interface MyPos : NSObject <MKAnnotation>
{
      CLLocationCoordinate2D coordinate;
      NSString *title;
      NSString *subtitle;
}

@property (nonatomic, assign) CLLocationCoordinate2D coordinate;
@property (nonatomic, copy) NSString *title;
@property (nonatomic, copy) NSString *subtitle;

@end
```

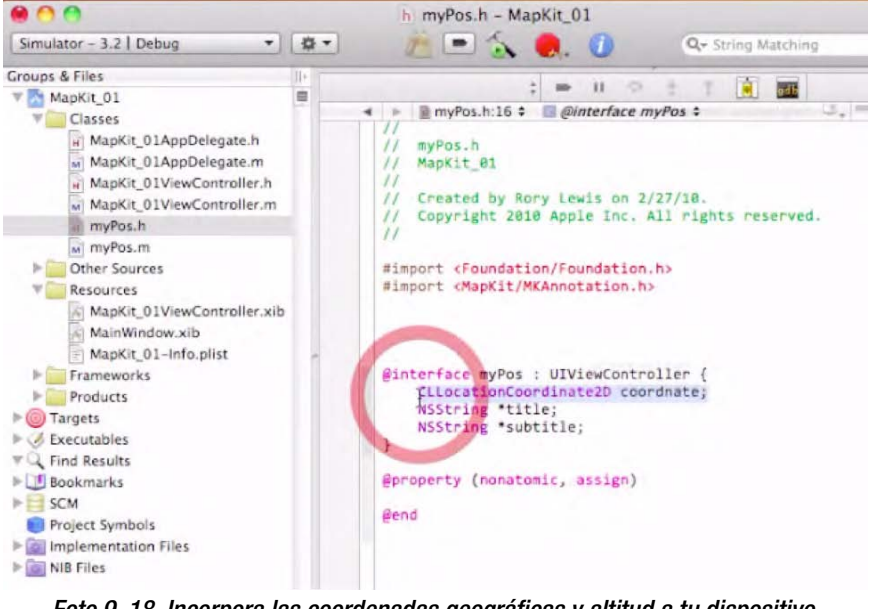


Foto 9–18. Incorpora las coordenadas geográficas y altitud a tu dispositivo.

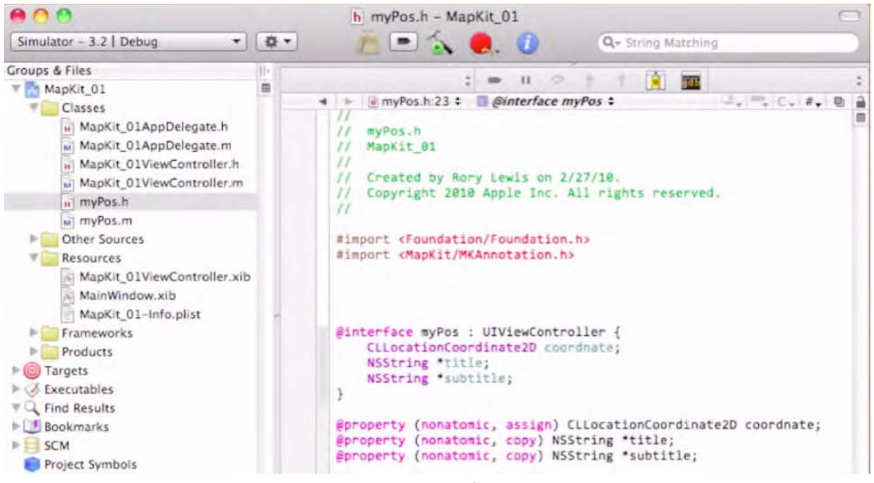


Foto 9–19. Tu archivo de cabecera "myPos" debería de quedar de este modo antes de guardar.

El Archivo myPos.m

16. Aquí simplemente sintetizaremos nuestro título y subtítulo coordinado con una declaración `@synthesize` que incluye `title` y `subtitle` `coordinados`. A continuación, suelta el `título` y el `subtítulo` en nuestra declaración `dealloc` (deallocation). Una vez hecho esto, tu archivo debería quedar parecido a la Foto 9–20. Guarda tu trabajo en este archivo.

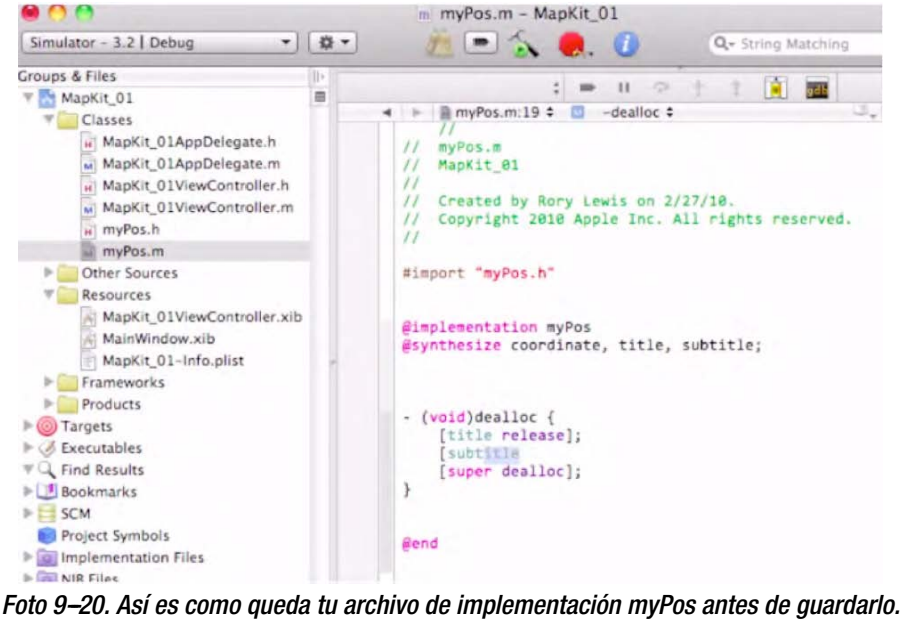


Foto 9–20. Así es como queda tu archivo de implementación myPos antes de guardarlo.

Los Archivos AppDelegate

Analizando la Foto 9–21, advertirás que no tenemos que hacer nada en el archivo de cabecera AppDelegate. Ya contiene todo lo que necesitamos. Necesitaremos, sin embargo, añadir una pequeña cosa en el archivo de implementación AppDelegate.

17. Abre `MapKit_01AppDelegate.m`. Primero, necesitamos sobrescribir la customización antes de ejecute nuestra aplicación. Esto lo hacemos con `[windowaddSubview:viewController.view]` y `[window makeKeyAndVisible]`. Asegúrate de que tu declaración `dealloc` desasigna tanto el `viewController` como la `ventana`.

18. Guarda tu trabajo y accede a la carpeta `Resources`.

NOTA: Por alguna razón, cuando ajusté mi aplicación por defecto en la plataforma iPad, no desasignó como se esperaba. Estoy operando con una versión beta, por lo que esto puede suponer un error menor. Sin embargo, siempre será interesante comprobar este parámetro. Ver Foto 9-22.

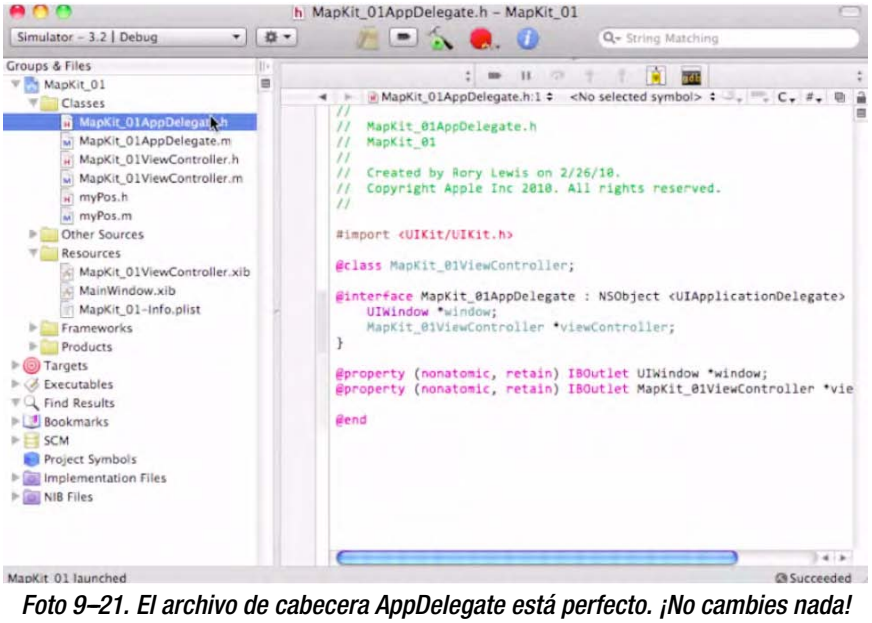


Foto 9–21. El archivo de cabecera AppDelegate está perfecto. ¡No cambies nada!

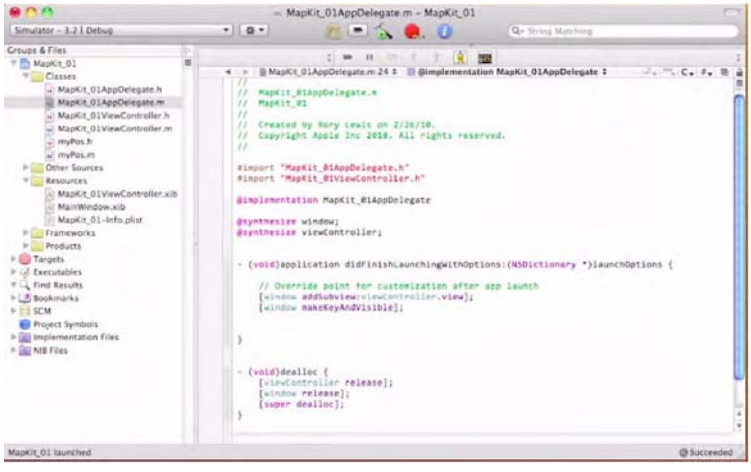


Foto 9–22. El archivo de implementación AppDelegate.

Conecta MapView con MKMapView

19. Como se muestra en la Foto 9–23, ve a la carpeta `Resources` y abre tu archivo nib. En el Interface Builder, conecta el mapView de tu File's Owner con tu MKMapView, tal y como se muestra en la Foto 9–24.

20. Ahora vuelve a Xcode y compila. Aparecerá la simulación iPad, y una chincheta aparecerá en la pantalla cayendo y situándose en lo alto de mi oficina o en la ubicación que tú hayas establecido. La chincheta será roja, y si pinchas sobre su cabeza, se mostrará la anotación que hemos hecho. Mira las Fotos 9–25 a la 9–27.

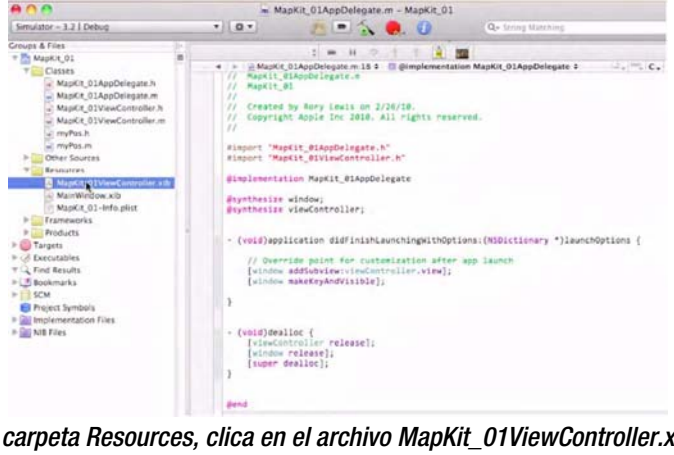


Foto 9–23. En la carpeta Resources, clica en el archivo MapKit_01ViewController.xib para hacer las conexiones y conseguir que la aplicación funcione.

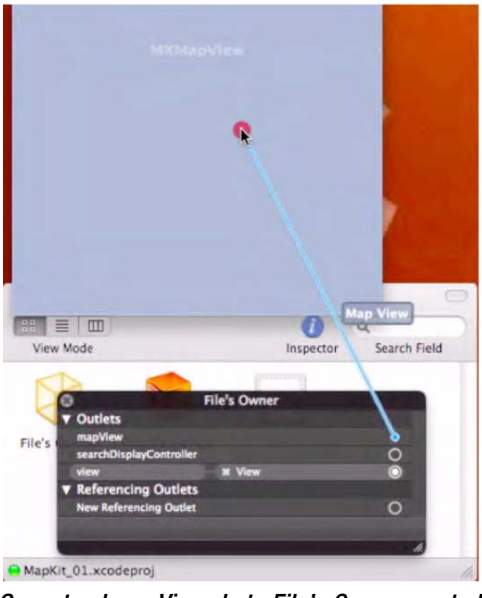


Foto 9–24. Conecta el mapView de tu File's Owner con tu MKMapView

Felicidades! Una vez más has conseguido terminar una aplicación de cierta complejidad, teniendo en cuenta que partimos de un cuerpo de código que hemos ido modificando a nuestro gusto. Cuando compares las imagines obtenidas en tu propio Simulador con las imágenes que te muestro a continuación, disfruta de la sensación de proporciona un trabajo bien hecho.

Ahora, después de un breve descanso, espero que te aventures a seguir investigando y viendo algunos ejemplos de la sección "Profundizando en el Código Mapkit de mis Estudiantes" con objeto de saciar tu "apetito" por el desarrollo personal y el desafío.

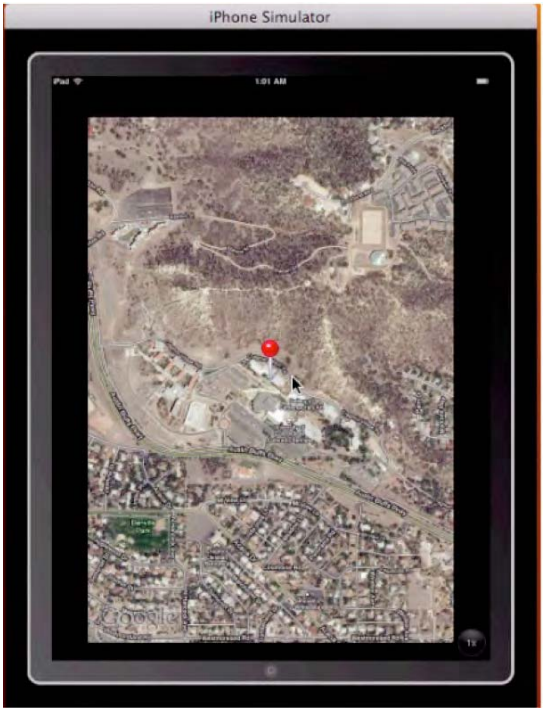


Foto 9–25. La chincheta se sitúa exactamente sobre las coordenadas.

.

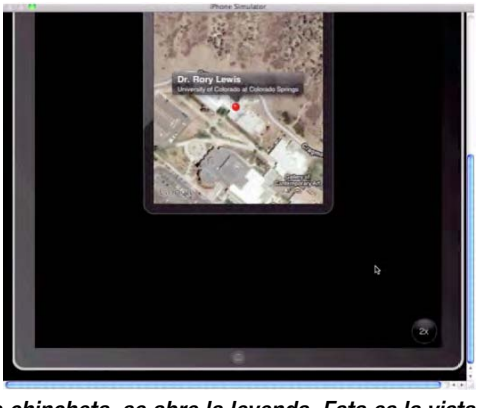


Foto 9–26. Clicando sobre la chincheta, se abre la leyenda. Esta es la vista del iPad Simulator en vista iPhone.

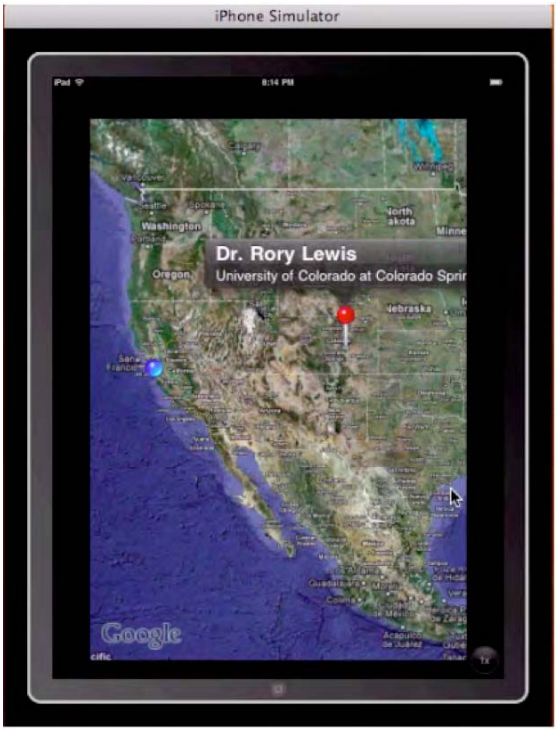


Foto 9–27. Bajando el zoom podemos comprobar visualmente la distancia entre nuestra ubicación actual y las oficinas de Apple en Cupertino, California.

Profundizando en el Código MapKit de Mis Alumnos

Cuando la gente se me acerca y me dice, “Hey, Dr. Lewis, tengo una gran idea para una nueva aplicación ...,” es sorprendente la cantidad de veces que esta idea implica el uso del framework MapKit. Hasta ahora hemos visto lo bueno de esto, pero una vez se profundiza en el código se puede convertir en algo de gran complejidad.

Como en un sabroso buffet de despedida alto en calorías, voy a compartir contigo algunos de los proyectos finales de mis estudiantes. Espero que los sigas y participes activamente de esto, pero también quiero reconocerte el hecho de que hayas llegado hasta aquí. Has conseguido terminar con éxito el ejercicio de este último Capítulo 9. Por tanto, recuerda: esta sección que empezamos es como una de esas “escenas adicionales” que llevan los DVD´s que tanto a los de Hollywood les encanta añadir- sin costo extra. *Relájate y disfruta*!

Parseando para MapKit desde Internet

Un poco de historia: Cuando abordamos por primera vez el MapKit en clase, lo hice de un modo muy parecido al ejemplo visto en este capítulo. Luego fuimos viendo una de las cosas más interesantes que se puede hacer con MapKit—la capacidad de analizar o leer información en vivo. Esta facultad permite a los usuarios "ver" la información sobre el mapa. Explicaré esto detalladamente antes de presentar tres proyectos de final de curso de algunos de mis estudiantes.

Una de las cosas más interesantes que podemos hacer con MapKit es la de conseguir información en tiempo real de internet y configurarla de modo que el Google Maps en el iPhone tome vida, mostrando información como el tiempo, geografía, taxis, mapas y muchas otras cosas más. Por ejemplo, una de las aplicaciones más populares para el área de la Bahía de San Francisco es un programa que se enseña en el libro *iPhone Cool Projects* (Wolfgang Ante, et al., Apress, 2009) (Foto 9–28) llamado "Routesy Bay Area San Francisco Muni and BART," hecho por Steven Peterson.

Peterson parsea (analiza y trata) todos los datos del servidor web BART (Bay Area Rapid Transit, `http://www.bart.gov/`) el cual realiza un seguimiento de todos los trenes: su situación actual y sus velocidades. La aplicación recoge todos estos datos y la convierte en una información útil y relevante para los usuarios de tren de la Bahía de San Francisco. En la Foto 9–29, verás el icono rojo de la aplicación y varios pantallazos del dispositivo iPhone. El de la izquierda muestra todos los lugares donde el usuario puede coger autobuses y trenes. La imagen del medio utiliza el mismo código que hemos usado en nuestro ejemplo para mostrarle al usuario su posición actual con un icono azul, y localizando también en el mapa la parada que se está buscando. La imagen de la derecha informa al usuario de información relevante de la mejor conexión que puede obtener con un tren, dado un determinado contexto, como puede ser, el menor tiempo empleado. La aplicación proporciona datos para los próximos tres trenes que llegarán a la estación más próxima al usuario.

En esencia, el código MapKit en el iPhone es una utilidad de análisis. Consigue información en directo de un servidor sobre el cual la mayoría de usuarios puede desconocer su existencia y proporciona estos datos para un útil propósito.

Dado los resultados inmediatos y prácticos de la aplicación de Peterson y otras por el estilo, pensé que este tema y este ejemplo sería perfecto como colofón para este libro. En primer lugar veremos algunas de mis notas de clase "parseando con MapKits" y luego te enseñaré varios proyectos de final de curso creados por mis estudiantes partiendo sobre esa base.

Con la bendición de mis alumnos, el código para estos proyectos puede ser descargado desde mi sitio web, tal y como se muestra más abajo. Esto te dará la oportunidad de tener el código en tu Mac al mismo tiempo que te enseñaré a modificarlo, te mostraré las principales características del mismo, o s simplemente cargarlo en tu iPad para mostrarles a tus amigos estas fantásticas aplicaciones. *Disfrútalo*!

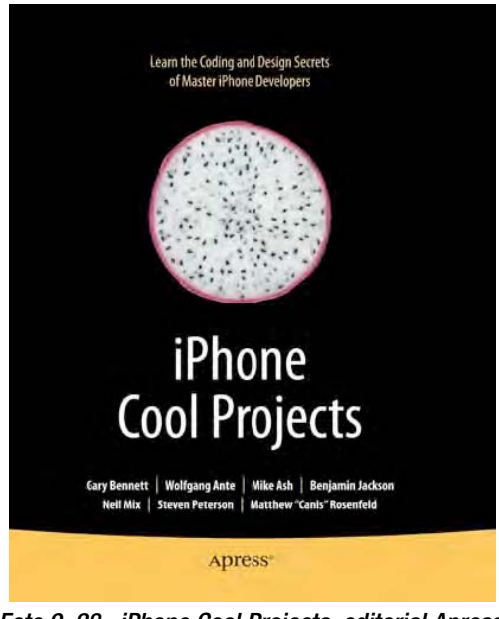


Foto 9–28. iPhone Cool Projects, editorial Apress.

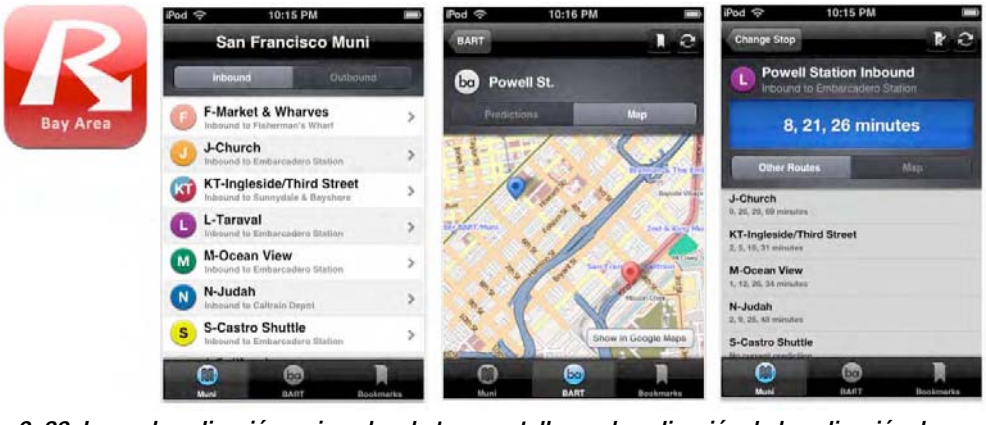


Foto 9–29. Icono de aplicación y ejemplos de tres pantallazos de aplicación de la aplicación de análisis:: "Routesy Bay Area San Francisco Muni and BART."

Podrás descargar el código de los proyectos de final de curso en las siguientes direcciones:

- Stephen A. Moraco (Hijo):
  `http://www.rorylewis.com/xCode/O11b_TrafficCam.zip`
- Stephen M. Moraco (Padre):
  `http://www.rorylewis.com/xCode/O11a_APRSkit.zip`
- Satish Rege:
  `http://www.rorylewis.com/xCode/O11c_MyTraffic.zip`

MapKit Parsing

Recuerde que vamos a sumergirnos en el código a un nivel mucho más profundo que lo hecho anteriormente en este libro. Sin embargo, todas las instrucciones que vienen a continuación pueden ser seguidas también en el código de mis estudiantes, el cual puedes descargar si deseas. De momento, simplemente lee y comprueba si puedes seguir su patrón de análisis, la creación de objetos, y así sucesivamente.

Antes de centrarnos en las aplicaciones, considera un escenario hipotético: Imagina que existe un Servidor web del grupo musical Grateful Dead, el cual difunde información actualizada sobre cada posición de un DeadHead (fans del grupo) - al menos aquellos que quieran dejarse localizar. Esta aplicación hipotética permitiría a un seguidor de Grateful Dead localizar a otros Deadheads de alrededor. Estos fans podrían quedar y compartir bootlegs, pasar tiempo juntos y en general relacionarse con otros seguidores de Grateful Dead.

Punto de Partida: Si fuésemos a crear esta aplicación, podríamos permitir a los usuarios ver donde se encuentran utilizando el Inspector de Atributos (Attribues Inspector) y habilitando el interrupor Permitir la Localización de Usuarios. Podríamos crear un controlador llamado *DeadHeadsView* el cual crearía una instancia del parseador que llamaríamos `Gratefuldead`. Entonces podríamos ajustarlo como el delegado de modo que recibiese el feedback y llamase a un método de datos `getGratefuldead`.

Obtener Datos desde Web: Al mismo tiempo que nuestro programa de análisis obtiene datos a través del XML del Servidor Grateful Dead, también podríamos querer coger ciertos elementos de datos y crear una instancia de cada objeto *Gratefuldead*. Por tanto, para cada instancia que se cree, haremos una llamada hacia nosotros mediante un método `addGratefuldead`. Tendríamos que poner en marcha `Gratefuldead` y métodos *Parser (análisis)* en nuestro `HeadsViewcontroller`. Podríamos pensar en que nuestro `GratefuldeadParser.h` fuera así:

```
+ (id)GratefuldeadParser; // this creates it
```

```
-(void)getGratefuldeadData; // this activates it
```

Añadir Métodos al Controlador de Vista: Antes de añadir métodos de implementación a nuestro controlador `DeadHeadsView`, necesitaríamos implementar el protocolo con la Delegación `GratefuldeadParser` Delegate e importar su archivo de cabecera `#import <GratefuldeadParser.h>`. Llegado a este punto habríamos terminado con la el archivo de cabecera y pasaríamos al de implementación.

Primero copiaríamos los dos métodos de implementación del `GratefuldeadParser.h` y pegaríamos estos dos métodos después de la directiva `@synthesize`:

```
@implementation DeadHeadsViewCOntroller

@synthesize deadView

-(void)getGratefuldeadData:(Gratefuldead *)Gratefuldead;
-(void)parserFinished
```

Probar la información del parseador: Para probar el Servidor Grateful Dead, deberíamos ver si podemos registrar algunos mensajes. Separaríamos los dos métodos, borraríamos los puntos y coma, añadiríamos corchetes y posteriormente añadiríamos el proceso de registro "log" de la siguiente manera:

```
-(void)getGratefuldeadData:(Gratefuldead *)Gratefuldead {
 NSLog(@"Hippie Message");
 }
-(void)parserFinished{
NSLog(@"located a Dead Head at %@", Gratefuldead.place");
}
```

Iniciar el Método del Parser: Habiendo implementado nuestros métodos de delegación, necesitaríamos hacer tres cosas

1 Codificar el método de análisis (parser). Lo pondríamos en un método al que llamaríamos `(void)viewWillAppear`. Esto sería llamado por un controlador de vista cuando su vista fuese a mostrarse. Si fuéramos a hacerlo de esta manera, ten en cuenta que siempre deberíamos llamar a nuestro método `- (void)viewWillAppear`.

2 Crear una instancia de nuestro parser a la cual podríamos llamar *`GratefuldeadParser`*. Con ello, obtendríamos `GratefuldeadParser *parser = [GratefuldeadParser gratefuldeadParser].` Como queremos convertirnos en el delegado, *`GratefuldeadParser parser.delegate = self`*.

3 Dos acciones en este paso: primero, decirle al parser (analizador) que obtuviera los datos:

```
[parser getGratefuldeadData];
```

El Segundo paso sería el gestionar el contenido de la importación de datos:

```
#import "GratefuldeadParser.h"
```

Cuando sea llamado - (void)viewWillAppear , esto crearía una instancia de *GratefuldeadParser*. A medida que se recibiesen las localizaciones de los fans Deadheads, nos mostraría su localización!

¿Recuerdas cómo asegurábamos que el usuario de la aplicación apareciese representado con una chincheta azul? Quiero que imagines esa chincheta azul como una simple anotación. Cuando es añadida al deadView, lo único que hace es preguntarle a su delegado por su propia localización.

NOTA: Cuando no se devuelve el valor nil, nos encontramos con que se mostrará en pantalla no mostrará el pin azul, pasando a mostrar la anotación de vista.

Por tanto, se devuelve el valor nil cuando la anotación no es igual a localización actual del usuario.

```
-(MKAnnotationView *)deadView:(MKDeadView *)deadView
               viewForAnnotation:(id <MKAnnotation>)annotation {
MKAnnotationView *view = nil; return view;
```

Pero aquí está el tema; no queremos devolver nil para nuestras localizaciones Gratefuldead. Queremos hacer algo de utilidad cuando nuestra anotación no sea igual a la propiedad deadView userLocation, la cual en sí misma es una anotación:

```
if(annotation != deadView.userLocation) {

        // THIS IS WHERE WE DO OUR COOL STUFF //
       BECAUSE IT'S A DEADHEAD, NOT THE USER }
```

Por ello vamos a usar el método delegado *dequeueReusableAnnotationViewWithIdentifier*, el cual será habilitado en el momento en que la gente esté localizada fuera de la pantalla. Esta es una manera útil de guardar las anotaciones en estructuras de datos separadas y añadirlas o eliminarlas del mapa cuando las interacciones del usuario lo requieran. Advierte que ReusableAnnotationViewWithIdentifier se utiliza para conseguir las anotaciones reutilizables en el mapa, no teniendo nada que ver con la función de adición o eliminación de anotaciones:

```
GratefuldeadAnnotation *eqAnn = (GratefuldeadAnnotation*)annotation;
view = [self.deadView dequeueReusableAnnotationViewWithIdentifier:@"GratefuldeadLoc"];
```

```
        if(nil == view) {
         view = [[[MKPinAnnotationView alloc] initWithAnnotation:eqAnn
                                                    reuseIdentifier:@"GratefuldeadLoc"]
autorelease];
}
```

La vista de anotación va y busca en la cola de reutilización de anotaciones si hay alguna vista que pueda ser reutilizada si `(nil == view) {` … Si no hay ninguna, devuelve nil – lo que significa que necesitamos crear otra nueva *`view = [[[MKPinAnnotationView alloc] initWithAnnotation:eqAnn.`*

Hay varias maneras creativas para hacer que tus anotaciones aparezcan y lo hagan de un modo animado, ya sea con campanas, logotipos y otra serie de elementos.

En este sentido, a estas alturas de programación de código, el paso más importante es repasar el mismo usando tu depurador NSLog; el cual determina si se está estableciendo conexión al servidor elegido. Una vez terminado con el depurador, llega el momento del análisis del XML. Ahora solo nos queda comprar iCandy (son pack de iconos) para las anotaciones.

Tres Proyectos Finales MapKit: CS-201 Aplicaciones iPhone, Objective-C

A continuación exponemos tres aplicaciones basadas en el tratamiento y análisis de información obtenida en Internet. Las dos primeras son de un padre y un hijo, ambos llamados Stephen Moraco, las cuales me dejaron totalmente confuso en la sala de conferencias, y la tercera es de Satish Rege. Todos ellos tuvieron la deferencia de hacer un breve comentario del por qué se apuntaron a estas clases.

Información Biográfica—Ejemplos 1 & 2

Stephen A. Moraco (Son)

Stephen M. Moraco (Father)

Steve A. (Foto 9–30) estoy en mi ultimo año de escuela secundaria. Me he matriculado en la UCCS. Stephen M. (Foto 9–31), Soy un ingeniero profesional de software y trabajo para

Agilent Technologies, Inc. Ambos poseemos iPhones y tenemos interés en aprender a programar para este dispositivo. El curso de la UCCS nos llamó la atención, ya que podríamos aprender juntos. De hecho, nos divertimos mucho en las clases CS201 del Dr. Lewis, en las cuales aprendimos a utilizar el SDK del iPhone y practicamos creando varias aplicaciones. Los debates en clase y luego entre nosotros cuando volvíamos en coche a casa despertaron en nosotros el interés y la admiración por las cosas que realmente podían hacerse con el iPhone. Nuestro proyecto final partió de esos debates. Dr. Lewis, gracias por impartir este curso. Nos proporcionó, en nuestro caso, unos grandes momentos de aprendizaje compartido. No podríamos haber disfrutado de momentos más agradables.

Foto 9–30. Stephen A. Moraco (Hijo)

Foto 9–31. Stephen M. Moraco (Padre)

Proyecto Final – Ejemplo 1

La aplicación de Stephen M. Moraco es afín a sus gustos. Como radioaficionado amateur, decidió parsear la web Bob Bruning's WB4APR, donde Bob había desarrollado un Automatic Position Reporting System (APRS), o Sistema de Notificación Automática de Posición. Muy parecido al ejemplo impartido en clase, en el cual se localizaban fans de los Deadheads, Stephen padre hizo una aplicación que podía localizar a Radioaficionados que se encontrasen a determinada distancia, especificada por el usuario. No voy a entrar a detallar el código de Stephen, ya que puedes descargarlo y repasarlo a tu gusto. Las partes más interesantes que creo deberías tener en cuenta son las siguientes: Su archivo de cabecera `APRSmapViewController` establece el mapa de ruta con tres 3 IBOutlets, 1 IBAction, y un ViewController:

```
@property (nonatomic, retain) IBOutlet MKMapView *mapView;
@property (nonatomic, retain) APRSwebViewController *webBrowserController;
@property (nonatomic, retain) IBOutlet UISegmentedControl *ctlMapTypeChooser;
@property (nonatomic, retain) IBOutlet UIActivityIndicatorView *aiActivityInd;

-(IBAction)selectMapMode:(id)sender;
```

En el archivo de implementación APRSkit_MoracoDadAppDelegate, usa el siguiente código para darle al usuario la posibilidad de loguearse. El resultado puedes verlo en la Foto 9–32. Lo interesante de esto lo tienes en el método -*(void)applicationDidFinishLaunching*, el cual también ofrece la posibilidad de establecer distancias al usuario para sus búsquedas:

```
-(void)applicationDidFinishLaunching:(UIApplication *)application {

    NSLog(@"MapAPRS_MoracoDadAppDelegate:applicationDidFinishLaunching - ENTRY");
    // Override point for customization after app launch

    [window addSubview:[navigationController view]];
          [window makeKeyAndVisible];

    // preload our applcation defaults
    NSUserDefaults *upSettings = [NSUserDefaults standardUserDefaults];
    NSString *strDefaultCallsign = [upSettings stringForKey:kCallSignKey];
    if(strDefaultCallsign == nil)
    {
        strDefaultCallsign = strEmptyString;
    }
    self.callSign = strDefaultCallsign;
    //[strDefaultCallsign release];

    NSString *strDefaultSitePassword = [upSettings stringForKey:kSitePasswordKey];
    if(strDefaultSitePassword == nil)
    {
        strDefaultSitePassword = strEmptyString;
    }
    self.sitePassword = strDefaultSitePassword;

    NSString *strDefaultDistanceInMiles = [upSettings
stringForKey:kDistanceInMilesKey];
    if(strDefaultDistanceInMiles == nil)
    {
        strDefaultDistanceInMiles = @"30";
    }
    self.distanceInMiles = strDefaultDistanceInMiles;
    //[strDefaultSitePassword release];
    // INCORRECT DECR [upSettings release];
}
```

Foto 9–32. Proyecto Final de la clase CS-201 — Aplicación APRS de Stephen M. Moraco. Apreciamos la ventana de ajustes.

Una de las primeras cosas que Stephen hizo cuando accedió al sitio web fue hacer una lista de todos los atributos en la fuente XML. La siguiente lista muestra lo que se encontró..

Columna-1 era el distintivo de llamada, CALLSIGN

Columna-2 era la URL de tráfico de Mensajes si está disponible.

Columna-3 era la URL de la página de Predicción Metereológica, si está disponible.

Columna-4 era la Latitud

Columna-5 era la Longitud

Columna-6 era la distancia desde el usuario (en millas)

Columna-7 era la hora en DD:HH:MM:SS del ultimo reporte. Par tener en cuenta estos datos, diseño 8 pointers en su archivo `APRSstationParser.m` file. Como ves, diseñó uno extra para posibles columnas desconocidas.

```
NSString *kCallSignCol = @"Callsign"; NSString *kMsgURLCol = @"MsgURL";
NSString *kWxURLCol = @"WxURL";
NSString *kLatitudCol = @"Lat";
NSString *kLongitudeCol = @"Long";
NSString *kDistanceCol = @"Distance";
NSString *kLastReportCol = @"LastReport";
NSString *kUnknownCol = @"???";// re didn't recognize column #
```

En el mismo archivo, introdujo estos casos:

```
case 1:
          m_strColumnName = kCallSignCol;
          break;
     case 2:
          m_strColumnName = kMsgURLCol;
          break;
     case 3:
          m_strColumnName = kWxURLCol;
          break;
     case 4:
          m_strColumnName = kLatitudCol;
          break;
     case 5:
          m_strColumnName = kLongitudeCol;
          break;
     case 6:
          m_strColumnName = kDistanceCol;
          break;
     case 7:
          m_strColumnName = kLastReportCol;
          break;
     default:
          m_strColumnName = kUnknownCol;
          break;
```

También, en el archivo de implementación `APRSkit_MoracoDadAppDelegate`, el método *`(void)recenterMap`* escanea todas las anotaciones para determiner el centro geográfico y, tal y como hicimos en el ejercicio de este capítulo, para calcular la porción de mapa a mostrar en pantalla. Stephen hizo lo mismo después de su tres declaraciones “if”. En la Foto 9-33 puedes ver una foto de la chincheta cayendo en la pantalla.

```
-(void)recenterMap {
     NSLog(@" - APRSpinViewController:recenterMap - ENTRY");
```

```
          NSArray *coordinates = [self.mapView
valueForKeyPath:@"annotations.coordinate"];
     CLLocationCoordinate2D maxCoord = {-90.0f, -180.0f};
     CLLocationCoordinate2D minCoord = {90.0f, 180.0f};
     for(NSValue *value in coordinates) {
          CLLocationCoordinate2D coord = {0.0f, 0.0f};
          [value getValue:&coord];
               if(coord.longitude > maxCoord.longitude) {
                    maxCoord.longitude = coord.longitude;
               }
               if(coord.latitud > maxCoord.latitud) {
                maxCoord.latitud = coord.latitud;
               }
               if(coord.longitude < minCoord.longitude) {
                minCoord.longitude = coord.longitude;
               }
               if(coord.latitud < minCoord.latitude) {
                    minCoord.latitud = coord.latitude;
               }
    }
```

Cabe señalar que en la clase `APRSstation`, Stephen representa los detalles parseados desde el APRS, los cuales establecen la ubicación de las chinchetas de posición.

```
#import <CoreLocation/CoreLocation.h>

@interface APRSstation : NSObject {
    NSString *m_strCallsign;
    NSDate *m_dtLastReport;
    NSNumber *m_nDistanceInMiles;
    NSString *m_strMsgURL;
    NSString *m_strWxURL;
    NSString *m_strTimeSinceLastReport;
    CLLocation *m_locPosition;
    int m_nInstanceNbr;
}

@property(nonatomic, copy) NSString *callSign;
@property(nonatomic, copy) NSNumber *distanceInMiles;
@property(nonatomic, retain) NSDate *lastReport;
@property(nonatomic, copy) NSString *timeSinceLastReport;
@property(nonatomic, copy) NSString *msgURL;
@property(nonatomic, copy) NSString *wxURL;
@property(nonatomic, retain) CLLocation *position;


@end
```

Otra cosa interesante que hizo Stephen fue el distinguir entre las emisoras de radio amateur

que tienen su propios sitios web y aquellas que no lo tienen. Para las que disponían de sitio web, en la anotación se incluía un chevron (escudo heráldico), que cuando es clicado, nos manda al sitio web. Ver Fotos 9–34 y 9–35. Este código puede verse directamente en el archivo de implementación `APRSstationParser.m`.

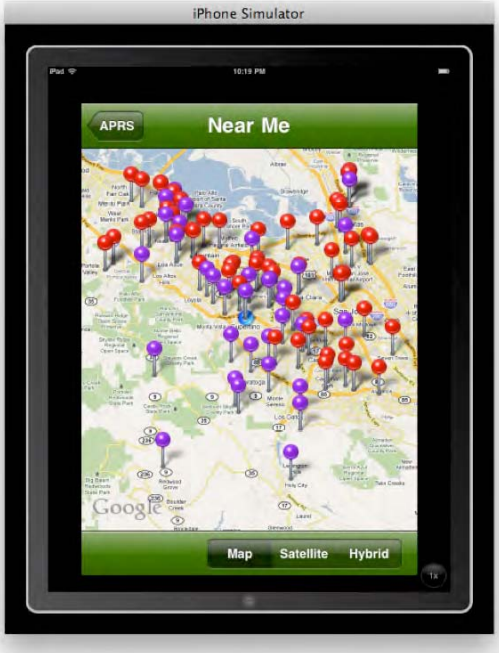


Foto 9–33. Proyecto Final de la clase CS-201— En la aplicación de Stephen M. Moraco podemos apreciar los pines animados cayendo en su ubicación. En vista iPad Simulator, apreciamos estos pines rodeando la sede de Apple.

Foto 9–34. CS-201 Final Project—La aplicación de Stephen M. Moraco muestra leyendas cuando una pulsa sobre uno de los pines y cuando se enlaza con un website del servidor APRS, un chevron azul aparece y el usuario puede clicar sobre el mismo parar ir a al correspondiente sitio web de radio amateur. En este caso sería el sitio de la radio amateur KJ6EXD-7, ya que el mismo está disponible.

Foto 9–35. Proyecto Final de la clase CS-201t—Aplicación de Stephen M. Moraco mostrando el sitio web KJ6EXD-7 en vista iPad.

En el archivo de implementación `APRSmapViewController`le, Stephen incluye, entre otras cosas, una metodología escueta para cambiar entre los distintos tipos de vista, ya sea mapa, híbrida o vista satélite. Un ejemplo de esto podemos verlo cuando mostramos la emisora de radio más cercana al usuario, que en el modo simulador es la Sede de Apple. En la Foto 9-36 podemos apreciar el modo Hibrido.

```
-(IBAction)selectMapMode:(id)sender
{
     UISegmentedControl *scChooser = (UISegmentedControl *)sender;
     int nMapStyleIdx = [scChooser selectedSegmentIndex];
     NSLog(@"APRSmapViewController:selectMapMode - New Style=%d",nMapStyleIdx);

     switch (nMapStyleIdx) {
        case 0:
             self.mapView.mapType = MKMapTypeStandard;
             break;
        case 1:
             self.mapView.mapType = MKMapTypeSatellite;
             break;
```

```
        case 2:
             self.mapView.mapType = MKMapTypeHybrid;
             break;
          default:
             NSLog(@"APRSmapViewController:selectMapMode - Unknown Selection?!");
             break;
    }
}
```

NOTA: Tienes dos opciones: 1) Adquirir el tuyo propio, o 2) Descargar cualquiera de esas tres aplicaciones, las cuales son esencialmente las mismas..

Foto 9–36. Proyecto Final de la clase CS-201—La aplicación de Stephen M. Moraco muestra las emisoras amateur más cercana a la Sede de Apple en vista de mapa Híbrido.

Para terminar, como toque especial de diferenciación el cual siempre animo a mis estudiantes a dar, Stephen incluyó una página tipo “Acerca de” en el `AboutView` nib. Ver Foto 9–37.

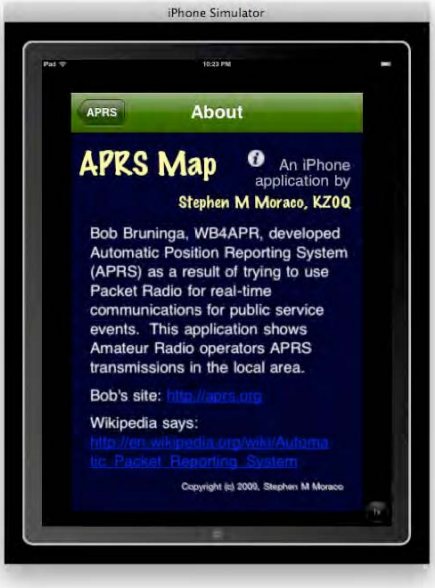


Foto 9–37. Proyecto Final de la clase CS-201—Aplicación de Stephen M. Moraco mostrando su “About Page” -- ¡quedó muy bien!

```
http://itunes.apple.com/us/app/pocketpacket/id336500866?mt=8
http://itunes.apple.com/us/app/ibcnu/id314134969?mt=8
http://itunes.apple.com/us/app/aprs/id341511796?mt=8
```

Proyecto Final – Ejemplo 2

Stephen A. Moraco es un estudiante de talento que asistió a mi clase. Su aplicación parsea el National Weather Cam network (Red Nacional De Cámaras de Meteorología) de `http://www.mhartman-wx.com/wcn/`. Esto puedes verlo en el archivo de implementación TrafficCamParser *`static NSString *strURL =http://www.mhartman-wx.com/wcn/wcn_db.tx`*`t`. Ver Foto 9–38.

Foto 9–38. Proyecto Final de la clase CS-201—La aplicación de Stephen A. Moraco's App comienza con centenares de pines cayendo del cielo y cubriendo un área específica cerca de la posición actual del usuario, en este caso, la Sede Central de Apple.

Descubrió que necesitaba usar un adaptador para filtrar los bad meta tags en las secciones <HEAD></HEAD> . Para ello creó reglas de reemplazo del símbolo "^" con </field><field>, reemplazando los
's con espacios en blanco, reemplazando "(" and ")
" con </field><field>, inicios y finales con <CAM><field> y </field></CAM> y eliminando las etiquetas ... He añadido una numeración para ayudarte a ver el inicio de cada línea, como el ajuste de texto, el cual también me desconcierta incluso a mí!

```
1     NSString *strNoParaQueryResults = [strQueryResults
      stringByReplacingOccurrencesOfString:@"<font size=\"-1\">("
      withString:@"</field><field>"];
2     strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@")<br>" withString:@"</field><field>"];
3     strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@"</font>" withString:@""];
4     strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@" " withString:@""];
5     strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@"></a>" withString:@"></img></a>"];
6     strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@"width=150" withString:@"width=\"150\""];
7     strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@"height=100" withString:@"height=\"100\""];
8     strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@"width=100" withString:@"width=\"100\""];
9     strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@"height=150" withString:@"height=\"150\""];
10    strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@"border=0" withString:@"border=\"0\""];
11    strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@"\"\"" withString:@"\""];
12    strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@".jpg " withString:@".jpg\" "];
13    strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@"&" withString:@"and"];
14    strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@"<font size=\"-1\">" withString:@""];
15    strNoParaQueryResults = [strNoParaQueryResults
      stringByReplacingOccurrencesOfString:@"</b<" withString:@"</b><"];
```

El archivo de cabecera TrafficCamAnnotation.h es directo y sencillo, usando *+ (id)annotationWithCam:(TrafficCam *)Cam*; y - (id)initWithCam:(TrafficCam *)Cam; pointers descritos anteriormente para mi hipotético GratefuldeadParser.h. En este caso, *+ (id)annotationWithCam:(TrafficCam *)Cam*; crea archivo de análisis y - (id)initWithCam:(TrafficCam *)Cam; lo inicializa. El resultado de todo este duro trabajo, teniendo en cuenta el código no útil, puedes verlo en esta anotación limpia. Ver Foto 9–39.

```
#import <MapKit/MapKit.h>
#import <CoreLocation/CoreLocation.h>

@class TrafficCam;

@interface TrafficCamAnnotation : NSObject <MKAnnotation> {
     CLLocationCoordinate2D Coordinate;
     NSString *Title;
     NSString *Subtitle;
     TrafficCam *Cam;
```

```
}

@property(nonatomic, assign) CLLocationCoordinate2D coordinate;
@property(nonatomic, retain) NSString *title;
@property(nonatomic, retain) NSString *subtitle;
@property(nonatomic, retain) TrafficCam *cam;

+ (id)annotationWithCam:(TrafficCam *)Cam;
-(id)initWithCam:(TrafficCam *)Cam;

@end
```

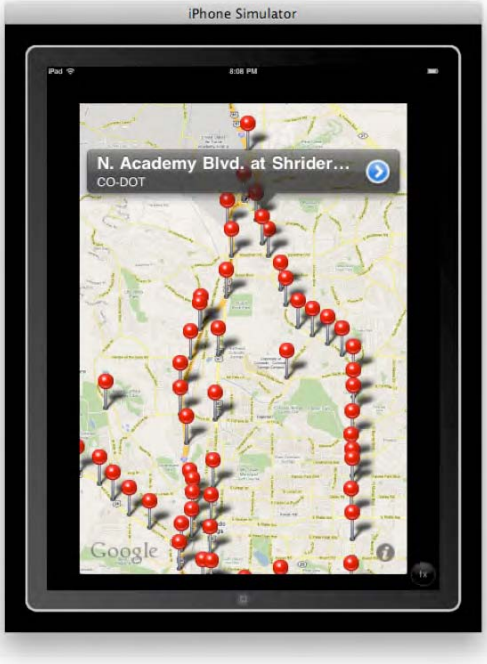


Foto 9–39. Proyecto Final de la clase CS-201—Aplicación de Stephen A. Moraco realizando zoom en el área de Colorado Springs. La leyenda de North Academy at Shrider aparece dado que el usuario ha clicado en esa intersección

Stephen también se encontró con que no podía usar automáticamente las vistas de video. Trabajó bastante en esto, ya que no es precisamente una labor trivial, en el archivo `TrafficCamSettingsViewController.m`. Un ejemplo fue el permitir otras orientaciones a la orientación por defecto:

```
BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
      // Return YES for supported orientations
      return (interfaceOrientation == UIInterfaceOrientationPortrait);

}
```

Tuvo que trabajar este código con objeto de que las imágenes de las videocámaras se adaptasen de un modo correcto a la pantalla, como puede verse en la Foto 9–40.

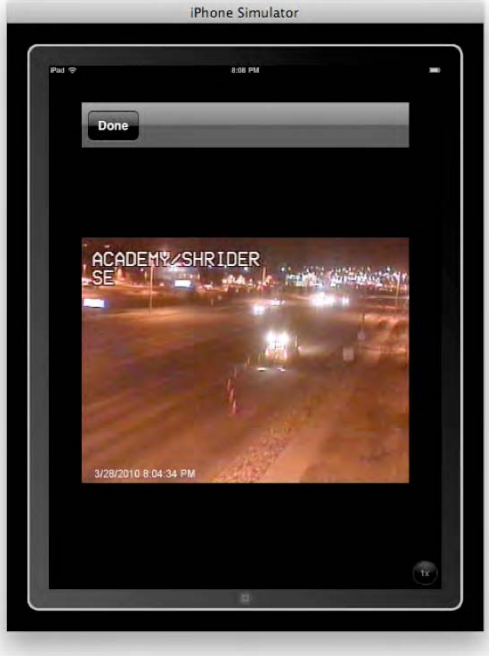


Foto 9–40. Proyecto Final de la clase CS-201—Aplicación de Stephen A. Moraco realizando zoom en el área de Colorado Springs. La leyenda de North Academy at Shrider aparece dado que el usuario ha clicado en esa intersección

Información Biográfica – Ejemplo 3, Satish Rege

Por qué quiero ser un desarrollador iPhone? Muy simple- El iPhone enseña computación, comunicación y la experiencia multimedia de un basto sistema de computación, y todo en la palma de una mano. Facilita interesantes recursos e interfaces de usuario elementales para expresar capacidades creativas de una forma sinérgica. Estas propiedades del iPhone despertaron en mí el interés de conocer las herramientas de desarrollo del iPhone para poder desarrollar mis propias ideas. El curso de Rory fue una excelente introducción que me dic una gran base de desarrollo.

Foto 9–41. Satish Rege

Proyecto Final – Ejemplo 3

Satish (Foto 9–41) siempre utilizó un código simple y elocuente en todos los trabajos que mandaba para casa. Era capaz de escribir en 20 líneas de código lo que otros necesitaban al menos tres veces más de espacio para hacer la misma cosa. Para su proyecto final, Satish ideó una aplicación que permite ver el tráfico que nos vamos a encontrar, analizando un cambio de camino en caso de que hubiera atascos.

Al menos, en teoría, así es como funciona la aplicación. Partiendo de la intersección I-25Northbound, analiza 27 opciones para las 27 cámaras de Colorado Springs. Simple, elegante… bonito.

La Foto 9–42 muestra la lista. Las Fotos 9–43 y 9–44 muestran dos ejemplos de las vistas de tráfico.

```
//Choose the camera depending on your co-ordinate

     switch (cameraCordinate) {
          case 1:
            url = [NSURL
URLWithString:@"http://www.springsgov.com/trafficeng/bImage.ASP?camID=17"]; //Camera -S
Academy/ I-25 North
          break;
          case 2:
            url = [NSURL
URLWithString:@"http://www.springsgov.com/trafficeng/bImage.ASP?camID=18"]; //Camera -HWY
85/87/I-25 N
          break;
>>>>>> >>>>>> >>>>>>
          case 26:
            url = [NSURL
URLWithString:@"http://www.springsgov.com/trafficeng/bImage.ASP?camID=33"]; // Camera -
Monument/ I-25 N
               break;
          case 27:
            url = [NSURL
URLWithString:@"http://www.springsgov.com/trafficeng/bImage.ASP?camID=49"]; //Camera -
CountyLine/ I-25 SE
               break;
```

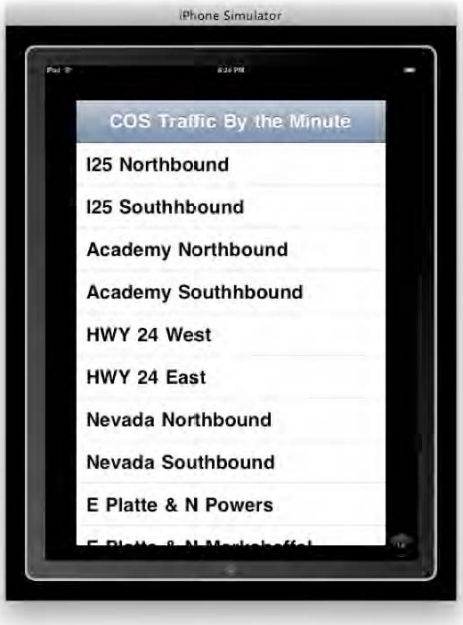


Foto 9–42. Proyecto Final de la clase CS-201—La aplicación de Rege muestra las cámaras más próximas al usuario.

Foto 9–43. Proyecto Final de la clase CS-201—La aplicación de monitorización de Rege mostrando una vista de cámara.

Foto 9–44. Proyecto Final de la clase CS-201—La aplicación de Rege mostrando otra vista de cámara.

Zoom Out … Seeing the Big Picture

Es importante saber de dónde venimos, dónde estamos ahora, y hacia donde vamos. Espero esto no te suene a demasiado metafísico, pero este capítulo es un poco de metáfora para nuestras vidas. Dónde estabas hace 5 años? El año pasado? El día anterior a comprar este libro, donde estarás dentro de 6 meses?

Por eso esta frase es tan popular. A la gente le encanta saber dónde se encuentra! A la gente le encanta saber, le encanta ser mostrada y saber como ir de “aquí” a “allí”.

Conoces el estereotipo por el cual los hombres son reacios a preguntar cómo llegar a un lugar? “Se lo que hago — porque sé a dónde me dirijo”. Cuando el GPS irrumpió en nuestras vidas, me quede impresionado, pero cuando Apple incluyó lo incluyó en el iPhone de la mano de ‘Maps,’, la experiencia fue tremenda. De repente, tenía la capacidad de consultar el oráculo y mantener mi ego masculino al mismo tiempo!

Esto es poder, y esto es autoridad… y es la revolución que has escogido. Ahora que has completado este libro y has completado con éxito los ejercicios que contiene –algunos más sencillos, otros más exigentes- estás en el camino correcto en el mundo de la programación.

Como dije con anterioridad, los objetivos que tenía para ti en este capítulo eran simples. Como en cualquier desafío, la práctica hace la perfección. Si estás cansado, pero todavía excitado acerca de estas ideas y posibilidades, lo tomo como un éxito rotundo – para ti y para mí!

Algunos de vosotros os preguntareis acerca de varios aspectos que no hemos tratado en este libro: el acelerómetro, cámaras/vídeos, protocolos peer-to-peer, RSS, clientes y servidores POP. Si te interesan estos aspectos, es que están en el buen camino. Significa que sabes dónde estás y dónde quieres llegar. La vida es bella!

LIBROS DE PROFESIONALES PARA PROFESIONALES ®

OTROS TITULOS

Los dispositivos iPhone e iPad son increíblemente populares y muy deseados, parte de su popularidad está en el maravilloso gran surtido de aplicaciones disponibles para la iTunes AppStore de Apple. ¿Y por qué no crear nuestras propias aplicaciones para la App Store? Si has soñado con la aplicación iPhone o iPad perfecta pero no sabías como desarrollarla, abre este libro y comienza a programar: *Aplicaciones iPhone e iPad para principiantes*e enseñará a:

- **confgurar tu ordenador para el desarrollo de aplicaciones iPhone/iPad**
- **hacer pequelos cambios en aplicaciones existentes antes de crear tus propias aplicaciones**
- **dar estilo a tus aplicaciones - haciéndolas más divertidas de utilizar y fáciles de navegar**
- **hacer uso de numerosas innovaciones de Apple, como la pantalla multi-touch y el acelerómetro**
- **usar combinaciones de teclas y trucos para crear potentes aplicaciones**

Con un mínimo compromiso y conocimientos de computación, cualquier persona puede crear aplicaciones sencillas para el iPhone y el iPad y este libro te enseñará a hacerlo. El Dr. Rory Lewis le guiará en los pasos necesarios para programar tu primera aplicación y ejecutarla. Su manera de enseñar le llevará a través de la espesa jerga y desinformación que rodea al desarrollo de aplicaciones, facilitando instrucciones simples y paso a paso complemen tadas con un gran número de pantallazos, ilustraciones y ejercicios practicos acompañados de vídeos online que te mermitirán seguir lo enseñado.

El iPhone es el gadget más popular de esta generación y con la aparición del iPad, Apple lo ha conseguido nuevamente: siendo el dispositivo multi-touch multitarea más avanzado, destacando su diseño innovador y gran capacidad de desarrollo. Su manejo más natural y cómodo facilita la creatividad del programador. Este libro es la guía de los fundamentos para el desarrollo de aplicaciones para dispositivos iPhone e iPad, escrito en lenguaje que todo el mundo puede entender y asimilar, sobre todo los soñadores.

INCLUYE eBOOK

Apress®

SOURCE CODE ONLINE

www.apress.com

US $29.99

Pcdcpc l ag_ c l
Am k nsr_ag Ë l n_p_ bgqnmqgrgtmq k Ë tgjcq

Nivel Usuario:
Principiante

ISBN 978-1-4302-2700-7
52999
9 781430 227007

Traducción: CO-2, amacor, mhergon, Euki. Proyecto iPadForos.com