

PROGRAMMEZ

pour iPhone,
iPod touch,
iPad
avec iOS 4



PROGRAMMEZ POUR IPHONE, IPOD TOUCH, IPAD AVEC IOS 4

IPUP

Pearson Education France a apporté le plus grand soin à la réalisation de ce livre afin de vous fournir une information complète et fiable. Cependant, Pearson Education France n'assume de responsabilités, ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

Les exemples ou les programmes présents dans cet ouvrage sont fournis pour illustrer les descriptions théoriques. Ils ne sont en aucun cas destinés à une utilisation commerciale ou professionnelle.

Pearson Education France ne pourra en aucun cas être tenu pour responsable des préjudices ou dommages de quelque nature que ce soit pouvant résulter de l'utilisation de ces exemples ou programmes.

Tous les noms de produits ou marques cités dans ce livre sont des marques déposées par leurs propriétaires respectifs.

Publié par Pearson Education France

47 bis, rue des Vinaigriers

75010 PARIS

Tél. : 01 72 74 90 00

www.pearson.fr

Collaboration éditoriale : Hervé Guyader

Réalisation PAO : Léa B.

ISBN : 978-2-7440-4168-6

Copyright © 2010 Pearson Education France

Tous droits réservés

Aucune représentation ou reproduction, même partielle, autre que celles prévues à l'article L. 122-5 2^e et 3^e a) du code de la propriété intellectuelle ne peut être faite sans l'autorisation expresse de Pearson Education France ou, le cas échéant, sans le respect des modalités prévues à l'article L. 122-10 dudit code.

Table des matières

| | |
|-------------------------------------------------------------|-----|
| Chapitre 1 Configurer son environnement | 1 |
| 1 Pourquoi développer pour cette plate-forme ? | 3 |
| 2 Par où commencer ? | 6 |
| 3 Où trouver les ressources adéquates ? | 13 |
| Chapitre 2 Le Hello iPUP | 19 |
| 4 Introduction à Xcode | 21 |
| 5 Coder quelques éléments simples | 25 |
| 6 Premiers pas dans Interface Builder | 27 |
| 7 Compiler sur simulateur ou sur iPhone ? | 30 |
| Chapitre 3 Appréhender quelques éléments d'interface | 31 |
| 8 Un catalogue des éléments d'interface | 33 |
| 9 Ajouter un fond d'écran à Hello iPUP | 44 |
| 10 Réagir aux événements | 48 |
| 11 Créer un delegate | 60 |
| 12 Présenter des listes de données | 70 |
| 13 Utiliser une scroll view | 81 |
| Chapitre 4 Utiliser les fonctionnalités de l'iPhone | 87 |
| 14 Services autour du GPS | 89 |
| 15 Jouer de la musique en secouant | 105 |
| 16 Multitâche et notifications locales | 115 |
| 17 Communiquer par Bluetooth | 123 |
| 18 Jouer une vidéo | 140 |
| 19 Les notifications Push | 147 |
| 20 Utiliser des photos | 156 |
| 21 L'In App Purchase (achat intégré) | 164 |

| | |
|------------------------------------------------------------------|-----|
| Chapitre 5 Aller plus loin | 187 |
| 22 Comprendre la gestion des vues | 189 |
| 23 Navigation entre vues. | 201 |
| 24 Utiliser un tab bar. | 212 |
| 25 Utilisation d'un timer pour déplacer une vue | 219 |
| 26 Core Animation. | 224 |
| 27 Passer des variables à un serveur (GET/POST) | 241 |
| 28 Utiliser le carnet d'adresses. | 252 |
| 29 Accéder à votre calendrier | 268 |
| 30 Intégrer de la publicité : iAd. | 278 |
| Chapitre 6 Gestion des ressources | 285 |
| 31 Traduire votre application. | 287 |
| 32 La mémoire | 296 |
| 33 Utilisation de SQLite | 310 |
| 34 Parser des fichiers XML et JSON. | 323 |
| 35 Sauvegarder des données. | 340 |
| 36 Utiliser Core Data | 348 |
| 37 Et du côté de l'iPad ? | 374 |
| Lexique | 393 |
| Index | 397 |

Introduction

Vous êtes désireux de programmer sur iPhone ? L'ouvrage que vous tenez entre les mains vous suivra tout au long de votre apprentissage.

Que vous soyez simple débutant ou déjà sensibilisé au développement d'applications pour iPhone, iPod Touch ou iPad, vous découvrirez, au travers des fiches thématiques, des exemples concrets de réalisation d'applications sous forme de tutoriels. Cet ouvrage vous guidera pas à pas pour apprêhender les différentes fonctionnalités nécessaires à l'élaboration d'applications performantes, variées et ergonomiques pour les utilisateurs du monde entier.

Les premières fiches aideront les débutants à trouver les ressources nécessaires avant de se lancer dans l'aventure. Notez que nous ne couvrirons pas ici une formation spécifique en Objective-C, le langage de programmation objet utilisé dans le développement d'applications iPhone, iPod Touch et iPad. Nous avons effectivement choisi de vous guider au travers d'exemples concrets, pendant lesquels vous apprêhenderez peu à peu le langage, que vous pourrez approfondir à travers les ressources rendues disponibles par Apple, notamment. Il est donc primordial de réaliser les fiches dans l'ordre.

Ensuite, dans la seconde partie, vous réaliserez votre première application, un «Hello iPUP», le point de départ de bien des langages. Puis, vous découvrirez dans une troisième partie les éléments d'interfaces mis à votre disposition pour permettre à votre application d'exploiter toutes les possibilités graphiques de la plate-forme.

Dans un quatrième temps, nous apprendrons ensemble à utiliser les fonctionnalités natives des appareils, comme la lecture de la bibliothèque musicale ou l'utilisation du GPS. Arrivé à ce stade, votre soif non étanchée d'approfondir vos connaissances vous amènera à la cinquième partie, qui traitera d'aspects nécessaires mais plus complexes du développement. Vous y apprendrez, entre autres, à gérer les vues, créer des animations, communiquer avec un site web à travers les webservices. Enfin, la dernière partie traitera des aspects non visibles de votre application, comme la gestion des ressources ou la sauvegarde de données. La dernière fiche vous donnera les différences entre l'iPad et l'iPhone, le système d'exploitation étant quasiment le même !

Le SDK iOS est bien trop riche pour être totalement couvert dans un même tenant. Le maître mot est donc *curiosité* !

En supplément à cet ouvrage, vous trouverez un espace dédié à l'échange autour de chaque fiche sur notre forum communautaire : <http://www.ipup.fr/forum>. Cet espace contiendra tous les codes sources créés à partir de chaque projet, et les mises à jour ou corrections éventuelles. De plus, vous pourrez partager avec plus d'un millier de membres, tous développeurs iPhone, et poser toutes vos questions !

Note de lecture : vous trouverez beaucoup d'anglicismes comme *label*, *table view*, *thread*, *push*, *delegate*. C'est un parti pris, en ce sens où les ressources en français sur le Web sont encore trop peu nombreuses pour vous permettre de trouver des réponses en écrivant dans votre moteur de recherche préféré : *ajouter contrôleur de navigation iPhone* (qui vous donnera des résultats concernant le contrôle parental...). Le ton style forum employé tout au long de cet ouvrage est volontaire, par souci de convivialité dans un domaine où le partage et l'entraide sont nécessaires.

Certains passages pourront être déroutants si vous êtes novice en programmation Mac ou iPhone, et donc non familier avec l'Objective-C. C'est pourquoi vous trouverez un lexique expliquant certains termes utilisés dans leur contexte, ainsi qu'un index qui vous permettra d'approfondir certaines notions.

Notez par ailleurs que l'écriture du code suit ces conventions :

- Un code en gras de ce type : `Code`, signale des lignes à modifier.
- Un code grisâtre : `Code`, indique les fichiers à modifier.
- Un code plus clair : `Code`, désigne des méthodes.

iPuP

L'équipe iPuP, ce sont quatre amis et jeunes entrepreneurs, fraîchement diplômés d'école d'ingénieur, Loann Fraillon, Aurélien Hibert, Jérémy Lagrue et Marian Paul. Ces quatre associés ont mis leur passion pour l'univers iPhone au service des autres : en commençant par la création d'un forum d'entraide entièrement dédié à la programmation sur iPhone, pour déboucher ensuite sur la fondation d'une entreprise.

Les auteurs, Marian Paul et Jérémy Lagrue, que vous retrouverez sur le forum sous les pseudos respectifs *ipodishima* (Twitter : @ipodishima) et *Bidou* (Twitter : @mrbidoux), ont rédigé cet ouvrage.

Marian et Jérémy, 23 ans, sont tous deux ingénieurs diplômés de l'École nationale supérieure des Mines de Saint-Étienne, cycle Ingénieur spécialisé en microélectronique, informatique et nouvelles technologies (ISMIN). Leur cursus les a amenés à découvrir la programmation informatique, et plus particulièrement l'univers iPhone, à travers un projet réalisé début 2009 dans le cadre de leurs études, en partenariat avec une PME spécialisée en traitement du signal et interaction vocale (Voxler).

REMERCIEMENTS

MARIAN PAUL

Mes tout premiers remerciements vont à Anaïs, qui a su supporter et endurer ces longues soirées à me voir travailler derrière mon écran. Je tiens également à remercier Jérémy pour ses lectures attentives, ses remarques et conseils sans lesquels cet ouvrage ne serait pas le même. Merci à Aurélien et Loann pour avoir supporté ce projet et aidé à la relecture.

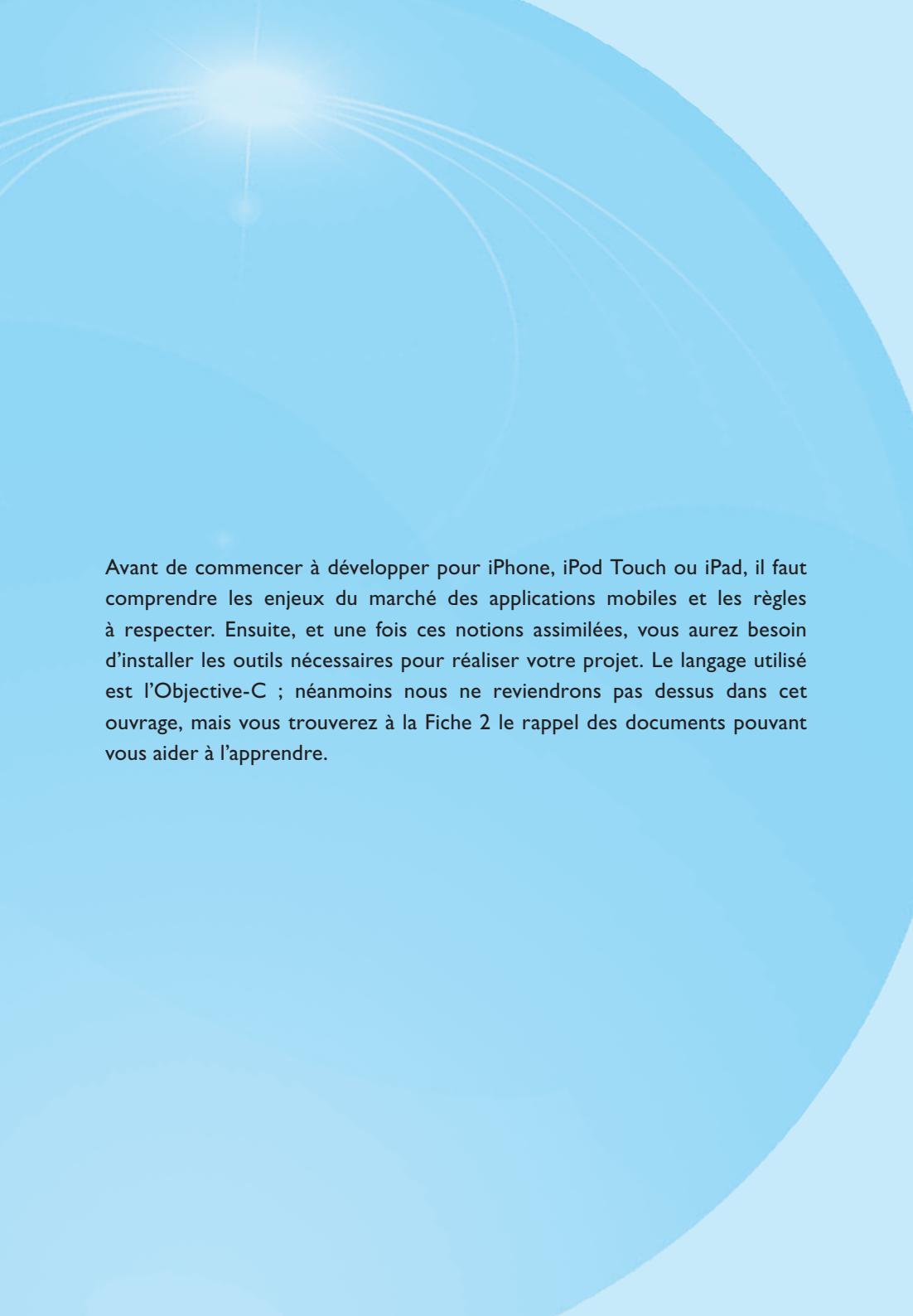
Mes derniers remerciements vont à Patricia Moncorgé des éditions Pearson pour son implication et son expérience apportées tout au long de ce projet, ainsi qu'à Amandine.

JÉRÉMY LAGRUE

Je tiens à remercier Marian pour son implication dans la réalisation de ce livre, ainsi que tous les membres du forum ipup.fr qui, chaque jour, nous apportent notre petite dose de bonne humeur. Merci également à l'équipe des éditions Pearson pour nous avoir suivis patiemment durant le processus de rédaction et correction.

CHAPITRE 1

CONFIGURER SON ENVIRONNEMENT



Avant de commencer à développer pour iPhone, iPod Touch ou iPad, il faut comprendre les enjeux du marché des applications mobiles et les règles à respecter. Ensuite, et une fois ces notions assimilées, vous aurez besoin d'installer les outils nécessaires pour réaliser votre projet. Le langage utilisé est l'Objective-C ; néanmoins nous ne reviendrons pas dessus dans cet ouvrage, mais vous trouverez à la Fiche 2 le rappel des documents pouvant vous aider à l'apprendre.

Ça y est, vous venez de franchir une nouvelle étape dans votre quête du développement d'applications pour les appareils mobiles d'Apple. Besoin de vous motiver encore un peu ?

QUELQUES CHIFFRES

Rien ne vaut quelques chiffres pour se représenter un marché ! L'iPhone a été annoncé pour la première fois le 9 janvier 2007, et commercialisé aux États-Unis à partir du 29 juin de la même année. Nommé iPhone Edge, il suscita un véritable engouement malgré ses quelques limitations techniques. Vendu aux États-Unis à 200 000 exemplaires en trois semaines, le succès fut au rendez-vous et conforta Apple quant à son implication sur le marché du téléphone mobile. Et depuis 2007, Apple sort chaque année un nouvel iPhone. Le 24 juin 2010, la quatrième version de l'iPhone a été présentée.

L'App Store est la plate-forme d'Apple qui distribue les applications. Il existe depuis le lancement de la deuxième génération d'iPhone en 2008. Les chiffres le concernant sont eux aussi éloquents :

- 5 000 000 000 : le nombre de téléchargements d'applications depuis la création de l'App Store.
- 1 000 000 000 dollars : la somme reversée aux développeurs depuis les premières ventes d'applications.
- 100 000 000 : le nombre d'appareils fonctionnant sous iOS vendus dans le monde depuis juin 2007.
- 2 000 000 : le nombre d'iPad vendus sous 59 jours après sa commercialisation (soit un toutes les 3 secondes).
- 225 000 : le nombre d'applications disponibles sur l'App Store.
- 20 000 : le nombre moyen de téléchargements par jour d'une application en première position dans le top 100 gratuit en France.
- 15 000 : le nombre d'applications soumises chaque semaine à Apple.
- 800 : le nombre moyen de téléchargements par jour pour entrer dans le top 10 payant en France.
- 95 % : le pourcentage d'applications acceptées sous une semaine après l'envoi aux services d'Apple.
- 80 : le nombre moyen de téléchargement d'applications par seconde.
- 28 % : la part de marché aux États-Unis pour l'iPhone.
- 10 : le ratio du nombre de téléchargements application gratuite/application payante.

Ces chiffres proviennent majoritairement de la keynote (présentation) d'Apple lors de la Worldwide Developer Conference (WWDC) de juin 2010 et de retours d'expériences de développeurs francophones.

UNE FACILITÉ DE DISTRIBUTION

Lorsque vous aurez acheté la licence (que nous verrons à la Fiche 2), vous pourrez distribuer votre application sur l'App Store. À part l'acquisition de la licence et le pourcentage prélevé par Apple, aucun frais supplémentaire d'hébergement ou de distribution ne vous seront demandés. Cependant, avant d'apparaître sur l'App Store, votre application doit être validée par Apple et se conformer aux règles annoncées par la société. Ces *guidelines* sont disponibles dans la documentation, point que nous aborderons à la Fiche 3.

Une fois votre application disponible sur l'App Store, il ne vous restera plus qu'à suivre son évolution et proposer des mises à jour en fonction des commentaires des utilisateurs, le reste étant géré par Apple.

TROIS CIBLES D'UTILISATEUR

En apprenant à programmer pour iPhone, iPad et iPod Touch, vous serez à même de toucher des publics différents. En effet l'utilisateur de l'iPhone est très nomade et souhaite avoir à sa portée des applications simples et très pratiques. De plus, certains jeux lorsqu'ils sont très bien pensés et réalisés deviennent vite addictifs. L'utilisateur va alors facilement y jouer lors d'un trajet en métro par exemple. Ces jeux, tels Doodle Jump ou Angry Birds, peuvent rester plusieurs mois dans le top 100 des applications vendues dans le monde.

Par contre, l'iPad se veut plus familial et présentera des applications plus évoluées, perfectionnées en terme de design. L'écran de l'iPad (9,7 pouces, 1 024 × 768 pixels contre 3,5 pouces pour l'iPhone) permet de présenter une application en créant des designs élaborés et soignés. Les applications iPad portent généralement le suffixe "HD" (Haute Définition), signe de la qualité de l'interface.

UN MARCHÉ MOBILE

Il faut toujours garder à l'esprit que l'iPhone est un téléphone mobile. Ces appareils présentent des contraintes dont voici une liste non exhaustive :

- **Performances.** Un téléphone mobile n'a pas la puissance, la mémoire, la vitesse de calcul, l'affichage graphique d'un ordinateur. Il faudra donc repenser votre manière de gérer les ressources si vous êtes développeur sur plate-forme fixe. Cette adaptation est un peu déroutante au début, mais les limites de l'appareil s'imposeront si vous n'y prêtez pas attention.
- **Connexion réseau.** La plupart du temps, l'iPhone sera connecté à un réseau mobile qui impose des contraintes de qualité du signal, de débit maximum et de quota de trafic. Si votre application nécessite un accès Internet, il faudra donc vérifier que le réseau est disponible et, le cas échéant, avertir l'utilisateur. L'absence de cet avertissement est une raison de refus de validation de la part d'Apple.
- **Autonomie.** L'autonomie de votre appareil est limitée. Faites donc toujours attention à l'utilisation des ressources pour ainsi minimiser la consommation de la batterie. Par exemple, si vous avez besoin de récupérer la position GPS de l'utilisateur, pensez à désactiver le GPS lorsque vous ne vous en servez plus ! Il en est de même pour l'accéléromètre ou le gyroscope.

- **Qualité d'une application.** Vous ne pourrez pas proposer de version bêta de votre application qui devra nécessairement être un produit fini (en attente éventuellement de mises à jour). Cette contrainte impose donc de travailler de manière correcte et fournir une application exempte de tout bogues. Une application qui crash ou qui utilise des librairies privées du système sont aujourd'hui les principaux motifs de refus de validation de la part d'Apple.

FAIRE LA DIFFÉRENCE

Tout cela est très joli, mais pour vous démarquer de toutes ces applications, il faudra nécessairement avoir le plus qui vous propulsera littéralement vers le succès. Certaines applications très complètes mais mal vendues ne seront jamais visibles et peu téléchargées. D'autres, car moins évoluées et utilisées très peu de temps, seront propulsées dans le top 100 sans aucun effort, car leur objectif sera principalement de cibler un utilisateur désireux de consommer beaucoup sans chercher la qualité. C'est un peu le phénomène du snack !

Le nom de votre application ainsi que les mots-clés sont très importants. L'utilisateur doit directement se faire une idée des fonctionnalités avec ces quelques informations. Les descriptions sont en général très peu lues, placez donc votre message accrocheur dans les premières lignes..

Libre à vous de reprendre un concept existant, mais la règle de base est de proposer encore mieux que ce qui existe.

L'aspect marketing de votre application est très important et ne doit aucunement être négligé si vous souhaitez percevoir une rémunération. Vous apprendrez petit à petit, en expérimentant, car ce qui est déroutant, c'est que chaque cas est unique.

Je vous conseille par ailleurs de vous procurer *Le Guide du Marketing des Applications iPhone* écrit par les responsables du site francophone [applicationiphone.com](http://www.applicationiphone.com)¹. Ce petit guide vous donnera, entre autres, des conseils pour mettre en valeur votre application sur l'App Store, les démarches pour contacter les sites les plus influents, pouvant vous procurer 1 000 téléchargements à l'annonce de votre application, et bien d'autres choses !

1. <http://www.applicationiphone.com/marketing-application-iphone/>.

Il ne suffit pas d'avoir de la bonne volonté pour développer sa première application mobile, il faut aussi le matériel...

Le Mac et l'appareil

Pour développer pour iPhone, iPad ou iPod Touch, un Mac est nécessaire... C'est clair, net et sans détour ! Vous n'aurez pas besoin d'un modèle de course, par contre un grand écran est apprécié pour afficher le simulateur iPad correctement.

Les anciennes versions Mac avec un processeur PowerPC ne sont pas supportés. Il faudra donc un modèle nouvelle génération avec Mac OS10.6.2 ou ultérieur pour faire fonctionner Xcode 3.2.3 qui est la version actuelle à l'heure où ces quelques lignes sont rédigées.

Les tests sur simulateur ne suffiront pas, car le système ne fait qu'émuler iOS sur votre machine. À l'utilisation, vous verrez que très souvent, tout marche correctement sur le simulateur, mais dès lors que vous le compilez sur un appareil, quelques désagréables surprises peuvent arriver. Je vous conseille donc très fortement d'avoir à côté de vous les appareils concernés par le développement de votre application. De plus, le simulateur ne permet pas de simuler l'accéléromètre, le gyroscope...

Astuce

Le temps de lancement d'une application en mode debug¹ est souvent plus long sur appareil que sur simulateur. N'hésitez donc pas à développer sous simulateur et faire des vérifications régulières sur votre appareil si vous n'êtes pas patient !

Le Langage

Le métier de développeur n'est pas inné et, à ce titre, tout débutant en programmation devra travailler pour maîtriser le développement iPhone, iPad ou iPod Touch. Cependant, rien n'est insurmontable si vous êtes patient et curieux.

Le langage de programmation utilisé est l'Objective-C, un langage objet surcouche du langage C. Il faudra donc impérativement maîtriser ce langage. Vous pourrez tout d'abord suivre les tutoriels du site du Zéro <http://www.siteduzero.com/> qui seront une bonne base pour débuter. Ensuite, vous trouverez des ressources sur Developpez.com². Adaptez votre apprentissage en fonction de votre niveau.

Une fois ces bases acquises, vous pourrez commencer à apprendre l'Objective-C, qui est un langage objet.

Info

Si vous avez des connaissances de programmation orientée objet (POO) car venant du monde web par exemple, vous devrez impérativement maîtriser le langage C pour poursuivre l'aventure car certaines notions, comme les pointeurs, vous manqueront..

1. Les termes soulignés de cette façon sont expliqués dans le lexique en fin d'ouvrage.

2. <http://c.developpez.com/cours/?page=lang-c>

Vous trouverez dans la documentation officielle trois documents à lire obligatoirement :

- *Object-Oriented Programming with Objective-C* vous expliquera ce qu'est la programmation orientée objet.
- *Learning Objective-C:A Primer* vous donnera les bases de l'Objective-C en quelques lignes.
- *The Objective-C Programming Language* décrit le langage de manière précise. IND-ISP-ENSABLE !

Info

Si vous avez des connaissances en C++ et/ou Java, vous pourrez lire les documents suivants :

- *De C++ à Objective-C* par Pierre Chatelier, vous permettra de faire la transition si vous venez du monde C++ (<http://chachatelier.fr/programmation/objective-c.php>).
- *De Java à Cocoa* par Sylvain Gamel pour aller de Java à l'Objective-C ! (<http://sylvain-gamel.developpez.com/tutoriel/mac/cocoa/java/>)

TÉLÉCHARGER XCODE

Xcode est l'IDE (Integrated Development Environment) nécessaire pour développer votre application. Xcode permet entre autres, de développer pour iPhone, iPad et iPod Touch. Il est disponible gratuitement sur le Web. Pour cela, créez un compte iTunes ou prenez celui que vous possédez déjà (cela simplifiera les démarches d'acquisition de licence par la suite). Vous trouverez la dernière version de Xcode en téléchargement à l'adresse suivante : <http://developer.apple.com/iphone/index.action>.



Figure 2.1 :
Bienvenue jeune
développeur !

Info

N'hésitez pas à vous rendre à cette adresse, qui vous permettra de gérer votre compte (voir Figure 2.2) <http://developer.apple.com/membercenter/index.action>.

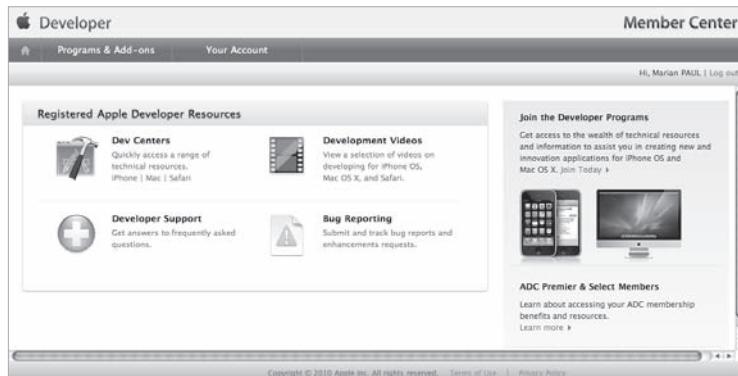


Figure 2.2 :
L'espace membre.

Cette version gratuite, idéale pour débuter, vous permettra de développer et de tester votre application sur simulateur. Si vous souhaitez la distribuer ou la tester sur un appareil, il vous faudra acquérir la licence, ce que nous verrons par la suite.

L'installation créera par défaut un dossier Developer dans Macintosh HD. Xcode sera alors accessible par le chemin Developer > Applications.

ACHETER LA LICENCE

Un jour ou l'autre, si vous poursuivez l'aventure, vous serez obligé d'acheter la licence. Elle vous donnera un accès complet à la documentation et aux forums de développeurs Apple (en anglais) où les ingénieurs de la société pourront vous répondre. De plus, c'est le seul moyen pour tester votre application sur votre appareil, et la distribuer sur l'App Store. Vous aurez également accès aux versions bêta (versions en test interne) lorsqu'il y en a.

La licence gratuite est un bon début pour s'initier au développement. Cependant, je ne peux que vous conseiller de l'acquérir le plus rapidement possible, dès vous êtes sûr de continuer à développer sur ces plates-formes. Je le répète, vous verrez au fil de votre expérience que le simulateur est un piège, car il ne réagit pas exactement comme la plate-forme.

Il existe deux types de licence :

- le programme Standard (iPhone Program Standard) au prix de 99 \$ ou de 79 \$ par an ;
- le programme Entreprise (iPhone Developer Enterprise Program) vendu 299 \$ par an.

Attention

Ne confondez pas. Le programme Entreprise permet simplement de distribuer des applications en interne dans votre entreprise, jusqu'à 500 employés, sans passer par l'App Store. Le programme Standard permet de tester vos applications sur les appareils (dans la limite de 100 par an) et propose la distribution sur l'App Store. Vous pouvez vous inscrire à ce dernier à titre individuel ou au nom d'une entreprise.

Pour adhérer au programme Standard, il faut se rendre à l'adresse suivante : <http://developer.apple.com/programs/start/standard/> puis suivre les indications.

Si vous souhaitez travailler pour une entreprise, le seul moyen pour que son nom d'éditeur apparaisse est qu'elle vous enregistre en tant que développeur sur sa licence.

Info

Si vous développez à titre personnel, vous pouvez adhérer à ce programme en tant qu'individu mais vous serez le seul développeur autorisé à utiliser ce compte, soumettre des applications...

Dans le cas d'une entreprise, vous pourrez créer une équipe de développeurs auxquels vous transferez certains droits.

Après votre adhésion, vous aurez accès au portail iTunes Connect qui vous permettra de gérer vos applications. Pour cela, rendez-vous à cette adresse <https://itunesconnect.apple.com>.

The screenshot shows the iTunes Connect interface. At the top, there's a navigation bar with the Apple logo, the text "iTunes Connect", a user profile "Marian Paul, iPUP", and a "Sign Out" button. Below the header, there are several sections with icons and descriptions:

- Welcome, iPUP**: iTunes Connect provides tools to help manage your content in the App Store. (NEW)
- Sales and Trends**: Preview or download your daily and weekly sales information here.
- Contracts, Tax, & Banking Information**: Request Contracts and manage your contact, banking and tax information. (X New)
- Financial Reports**: View and download your monthly financial reports.
- Manage Your Applications**: Add, view, and manage your applications in the iTunes Store.
- Manage Your In App Purchases**: Create and manage In App Purchases for paid applications.
- iAd Network**: View ad performance and manage the ads that appear in your apps.
- Manage Users**: Create and manage both iTunes Connect and In App Purchase Test User accounts.
- Contact Us**: Having a problem uploading your application? Can't find a Finance Report? Use our Contact Us system to find an answer to your question or to generate a question to an iTunes Rep.

At the bottom of the page, there are links for "Download the Developer Guide.", "FAQs", "Review our answers to common inquiries.", and standard footer links for "Home", "FAQs", "Contact Us", "Sign Out", "Copyright © 2010 Apple Inc. All rights reserved.", "Terms of Service", and "Privacy Policy".

Figure 2.3 :
iTunes Connect.

De plus, sur le portail développeur, vous aurez notamment accès au iPhone Provisioning Portal qui gérera les certificats pour vos applications.

The screenshot shows the iPhone Provisioning Portal interface. At the top, there's a navigation bar with links for Technologies, Resources, Programs, Support, and Member Center. A search bar is also present. Below the navigation, a banner says "Welcome, Marian PAUL". On the left, a sidebar has links for Home, Certificates, Devices, App IDs, Provisioning, and Distribution. The main content area starts with a "Welcome to the iPhone Provisioning Portal" message, followed by a "Visit the Member Center for Team, Account, and Program info" section with a list of actions like sending invitations and requesting support. There's also a "Get your application on an iPhone with the Development Provisioning Assistant" section featuring an icon of a smartphone and a "Launch Assistant" button. Another section, "Reset Your List of Development Devices", allows users to manage up to 100 devices. On the right side, there are "Portal Resources" (Program User Guide, Obtaining your Certificate, Assigning Devices, Creating your App IDs, Creating Provisioning Profiles), "Support Resources" (iTunes Connect Support, Technical Support, Developer Support), and contact information for Team Agent Loans and iTunes Connect.

Figure 2.4 : L'antre des certificats.

CRÉER UN CERTIFICAT

La première étape est de créer un certificat qui sera transmis à Apple et vous permettra d'installer des applications sur vos appareils. Pour cela :

1. Lancez l'application Trousseau d'accès que vous trouverez dans Applications > Utilitaires.
2. Cliquez sur le menu Trousseau d'accès puis Assistant de certification > Demander un certificat à une autorité de certificat...
3. Remplissez ensuite les informations comme à la Figure 2.5, nous allons enregistrer cette demande de certificat sur le disque. Ce fichier aura pour extension .certSigningRequest.
4. Vérifiez que la fenêtre suivante est la même que la Figure 2.6.

Comme je l'ai dit, les certificats se gèrent sur le portail développeur.

5. Rendez-vous à <http://developer.apple.com/iphone/> puis cliquez sur iPhone Provisioning Portal et Certificates.
6. Faites ensuite add Certificates et envoyez le fichier que nous venons de créer.
7. Vos développeurs valident le certificat. Ils doivent cliquer sur Approve dans Team Signing Requests.
8. Téléchargez le certificat et installez-le sur votre ordinateur (en double-cliquant dessus).



Figure 2.5 : Demandez un certificat de développeur.



Figure 2.6 : Sécurisez le tout.

Figure 2.7 : Votre nouveau certificat.

Info

Le certificat est lié à votre machine. Si vous souhaitez développer sur plusieurs machines, il faut vous rendre dans le Trousseau d'accès puis cliquer sur Clés. Ensuite, repérez la clé correspondant à votre certificat de développeur puis cliquez du bouton droit et Export Votre Nom (voir Figure 2.8). Enregistrez le fichier .p12 puis ajoutez un mot de passe.

Cette étape est très importante et je vous conseille de sauvegarder ce fichier même si vous ne possédez qu'une machine. Vous continuerez ainsi à signer des applications depuis un autre ordinateur si le vôtre est en maintenance par exemple.

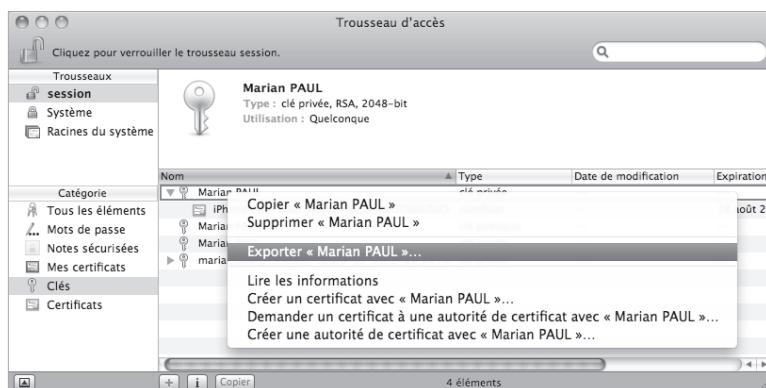


Figure 2.8 :
La clé à sauver.

GÉNÉRER DES PROVISIONINGS

Bien, cela étant fait, vous avez maintenant le droit de signer des applications pour les installer sur des appareils. Encore faut-il autoriser votre appareil pour le développement et créer un [provisioning](#) de développement. Cette tâche, un peu compliquée, a été simplifiée depuis Xcode 3.2.3, alors profitez-en !

Branchez votre appareil à votre ordinateur puis lancez Xcode. Rendez-vous dans Window > Organizer où vous devriez le voir apparaître. Cliquez dessus puis sur Use for Development. Les actions suivantes vont se réaliser :

- envoi de l'identifiant de l'appareil sur le Provisioning Portal (une identification vous sera demandée) ;
- un provisioning profile incluant l'appareil va être téléchargé dans Xcode ;
- les membres de l'équipe pourront développer sur cet appareil une fois que Xcode aura rapatrié le nouveau provisioning.

Vous pouvez néanmoins réaliser ces opérations à la main, depuis le portail developer si vous souhaitez notamment n'autoriser de signer votre application que sur certains appareils.

Le Web regorge de ressources pour répondre à toutes vos questions. En voici une vue d'ensemble.

Avant toute chose, il est bon de rappeler certains réflexes à adopter. Lorsque vous posez une question, commencez par chercher dans la documentation fournie par Apple qui est très bien réalisée et facile d'accès. Elle nécessite cependant un petit temps d'adaptation pour l'exploiter de manière performante. La plupart de vos interrogations y trouveront leurs réponses. Si rien ne correspond, rendez-vous sur le Net.

Info

Beaucoup de ressources sont en anglais, vous n'y couperez pas et devrez maîtriser cette langue. Les ressources en français existent, dont le forum iPUF, mais vous devrez approfondir par *vous-même, dans la langue de Shakespeare*...

LES PREMIERS DOCUMENTS À LIRE

Comme spécifié à la Fiche 2, commencez par lire les guides Apple concernant l'Objective-C :

- *Object-Oriented Programming with Objective-C.*
- *Learning Objective-C: A Primer.*
- *The Objective-C Programming Language.*

Ensuite, vous devrez vous familiariser avec l'environnement iPhone. De plus, votre application devra se conformer à ce document avant d'être validée et vendue sur le Store : *iPhone Human Interface Guidelines* qui décrit le modèle que vous devez calquer en terme d'ergonomie et d'interface utilisateur. Lisez également *iPad Human Interface Guidelines* si vous souhaitez développer sur cette plate-forme.

Attention

N'oubliez pas que vous ne disposez que d'une seule fenêtre. De plus, l'interface graphique et les interactions utilisateurs doivent impérativement être pensées avant d'écrire la moindre ligne de code. Votre application devra être utilisable de manière **intuitive**.

Une fois ces quelques pages lues, vous pouvez vous attaquer à *iPhone Application Programming Guide* qui vous donnera des conseils pour la réalisation de votre application et *iPhone Development Guide* pour apprendre à vous servir des outils de développement.

Je vous conseille également de vous procurer la bible des développeurs Cocoa : *Programmation Cocoa sous Mac OS X* par Aaron Hillegass¹. Certes, ce livre traite de la programmation Mac OS X, mais vous ferez facilement le lien entre Cocoa et Cocoa Touch utilisé par iOS.

I. <http://www.pearson.fr/livre/?GCOI=27440100726260>

LA DOCUMENTATION

Une fois ces quelques documents lus, vous pourrez démarrer votre projet. Vous aurez très souvent besoin d'accéder à la documentation. De plus, faites cette recherche avant de venir poser votre question sur un forum surtout si cette dernière est du type : "Bonjour, j'aimerais savoir comment changer la couleur du texte dans un label ?" La réponse sera immédiate : "Merci de chercher dans la documentation !" Regardons comment faire.

Vous accédez à la documentation de plusieurs manières :

- **En ligne.** Vous la trouverez sur votre portail développeur après identification ou sur ce site : <http://developer.apple.com/iphone/library/navigation/index.html>.
- **Dans Xcode.** Une copie de la version en ligne est disponible directement sous Xcode. Pour cela, se rendre dans Help > Developer Documentation ou faire le raccourci + + .

Astuce

Pour accéder plus rapidement à la documentation, cliquez du bouton droit sur un mot-clé comme `UILabel` ou `shouldAutorotateToInterfaceOrientation` puis choisissez Find Text in Documentation.

Si vous ne souhaitez pas lancer l'Aide, repérez l'élément sur lequel vous souhaitez vous renseigner, appuyez sur la touche puis double-cliquez. Vous aurez ainsi une fenêtre contextuelle comme à la Figure 3.1

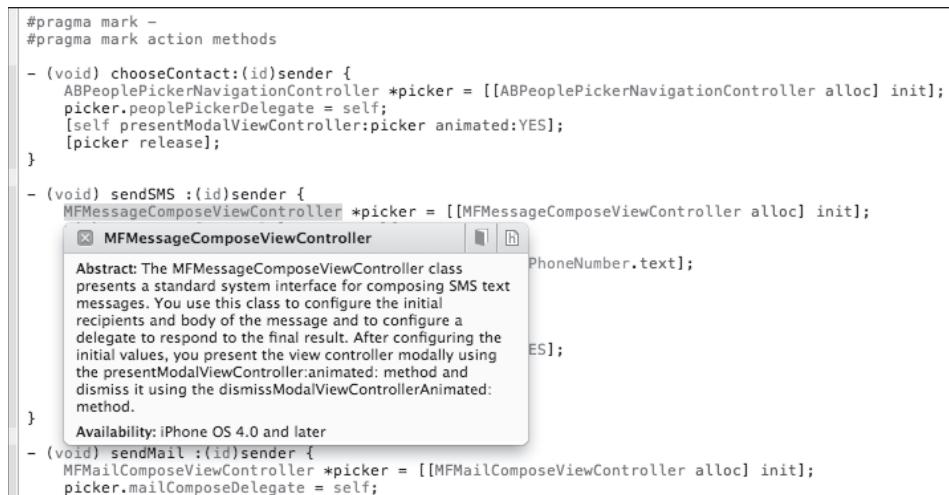


Figure 3.1 : L'aide contextuelle.

Si vous cherchez par exemple à vous renseigner sur `UILabel`, vous devriez obtenir une figure analogue à la Figure 3.2.

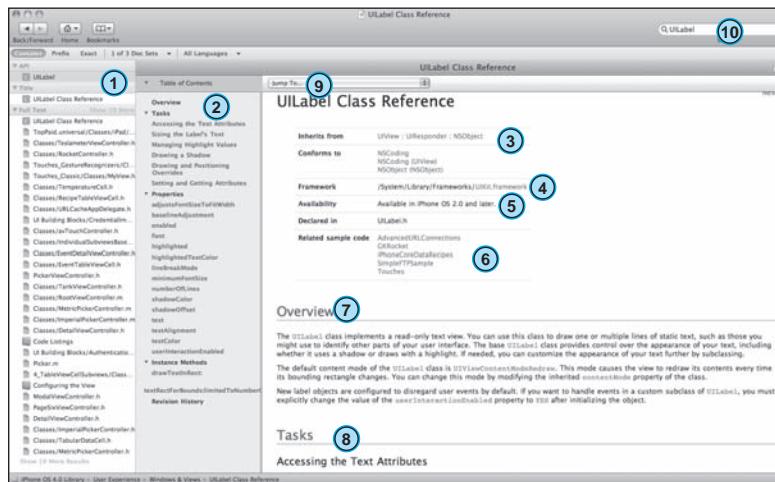


Figure 3.2 : Se familiariser avec l'aide.

En ①, vous trouverez la liste de tous les documents où apparaît `UILabel`. Dans la colonne représentée par le numéro ② sont listées toutes les méthodes, les propriétés et les explications pour exploiter un objet de la classe `UILabel`.

En ③, vous trouverez l'héritage de la classe `UILabel` puis en ④ le framework à importer pour l'utiliser. L'élément représenté en ⑤ est très important car spécifie la version minimum de l'OS pour pouvoir l'utiliser.

En ⑥, vous trouverez une liste de tous les exemples de code employant la classe `UILabel`.

Info

Ces exemples de code (*samples code*) sont très importants car ils vous montrent comment utiliser un objet. N'hésitez donc pas à les télécharger puis les compiler et vous inspirer du code source.

En ⑦, vous trouverez une description de la classe et en ⑧ la liste des méthodes et propriétés détaillée. Enfin, vous pourrez accéder rapidement à un endroit avec la liste déroulante pointée par le numéro ⑨.

Le champ de recherche dans la documentation est représenté par le numéro ⑩.

Attention

La documentation Xcode ne concerne pas uniquement l'iPhone. Pour éviter de chercher dans la documentation Mac des éléments qui n'existent pas sur l'iPhone, ne recherchez que dans la documentation iPhone comme à la Figure 3.3.



Figure 3.3 : Ne donner la parole qu'à l'iPhone

Astuce

Vous ne vous rappelez plus d'une propriété d'un élément ou du nom d'une méthode ? L'autocomplétion n'a pas suffit ? Appuyez sur la touche Esc de votre clavier. Par exemple, Figure 3.4 après avoir déclaré `UILabel *monLabel` puis tapé `mo`, vous aurez un choix parmi lesquels le nom de votre objet. Figure 3.5, après avoir écrit `monLabel.`, vous aurez une liste des propriétés disponibles pour ce label. Enfin, après avoir tapé `[monLabel set`, vous aurez une liste des méthodes commençant par `set` (Figure 3.6)

```
mode_t          pour l
modf
modff
modfl
monLabel        addSubvi
                ALabel;
mo
```

Figure 3.4 : La liste des objets ou variables.

```
// label pour le numéro du destinataire
label.accessibilityFrame = [[UIAccessibilityFrame alloc] init];
label.accessibilityHint = [[UIAccessibilityHint alloc] init];
self.accessibilityLabel = [[UIAccessibilityLabel alloc] init];
label.accessibilityLanguage = [[UIAccessibilityLanguage alloc] init];
label.accessibilityTraits = [[UIAccessibilityTrait alloc] init];
[textF accessibilityValue = [[UIAccessibilityValue alloc] init]];
[textF adjustsFontSizeToFitWidth];
[textF alpha];
label.autoresizesSubviews = YES;
label.autoresizingMask = [UIAutoresizingFlexibleWidth | UIAutoresizingFlexibleHeight];
label.backgroundColor = [UIColor colorWithRed:0.0 green:0.0 blue:0.0 alpha:1.0];
label.baselineAdjustment = [UIBaselineAdjustmentAlignBottom];
label.bounds = [UIRectZero];
UILabel *monLabel = [[UILabel alloc] initWithFrame:CGRectMake(10, 10, 100, 100)];
monLabel.
```

Figure 3.5 : La liste des propriétés de `UILabel`.

```
// label pour le numéro du destinataire
label.setMinimumFontSize: 12.0;
label.setMultipleTouchEnabled: YES;
self.setNeedsDisplay;
label.setNeedsDisplayInRect: CGRectMake(10, 10, 100, 100);
// textF.setNeedsLayout;
textF.setNilValueForKey: @"placeholderText";
[textF setNumberOfLines: 1];
[textF setObservationInfo: nil];
label.setOpaque: NO;
label.setShadowColor: [UIColor blackColor];
label.setShadowOffset: CGSizeMake(0, 1);
label.setTag: 1;
[monLabel setText: @"Hello World"];
UILabel *monLabel = [[UILabel alloc] initWithFrame:CGRectMake(10, 10, 100, 100)];
[monLabel set
```

Figure 3.6 : La liste des méthodes de `UILabel`.

LES RESSOURCES SUR INTERNET

Avant de vous rendre sur le Net pour poser votre question, retournez encore une fois à la documentation pour vérifier que la réponse à votre question n'y est pas. Ensuite, vous pourrez vous rendre sur des forums spécialisés. Vous trouverez en anglais :

- Le forum developer d'Apple accessible depuis votre portail. Ce forum de très bonne qualité vous permettra d'échanger avec les ingénieurs Apple qui s'y rendent très régulièrement.
- Le forum iphonedevsdk.com ou stackoverflow.com sont également deux sources de qualité et très actives. Vous recevrez des réponses dans l'heure, voire moins.

Il existe également des ressources en français dont le site spécialisé [ipup.fr](http://www.ipup.fr) avec son forum <http://www.ipup.fr/forum>. Ce site a été créé suite à une constatation très simple : les ressources en français pour le développement iPhone étaient rares. Les rédacteurs de ce livre, qui sont aussi les fondateurs du site, ainsi que le millier de membres très actifs seront toujours prêts à aider les débutants.

Une rubrique spéciale est mise en place pour échanger autour du livre et télécharger les codes sources de chaque fiche. N'hésitez pas à vous inscrire, la bonne humeur nécessaire pour coder sur iPhone, iPad ou iPod Touch sera là ! De plus, vous trouverez généralement réponse à votre interrogation dans l'heure.

Attention

J'insiste. Les développeurs sont en général toujours prêts à expliquer et partager leurs connaissances. Cependant, les questions de débutants sont souvent posées sur le forum et reviennent régulièrement. Vérifiez toujours avec une recherche avant de poster pour éviter trop de redondance sur un même sujet !

Vous n'obtiendrez jamais une réponse à une question du type : "Pourriez-vous me faire ça ?". Par contre, si vous montrez dans votre post que vous avez cherché, essayé des solutions mais que vous n'y arrivez pas, la réponse sera immédiate, cordiale et expliquée !

Petite astuce : si vous passez par Google et que vous ne trouvez pas de fonction recherche sur le site concerné, vous pouvez essayer de taper dans le champ de recherche la requête suivante : site:l'adresse du site votre interrogation. Par exemple : "site:www.ipup.fr changer la couleur d'une table view".

CHAPITRE 2

LE HELLO iPUP

Ça y est, vous êtes décidé à vous lancer dans l'aventure iPhone ! Mais par où commencer ? Dans ce chapitre, vous découvrirez les outils nécessaires pour démarrer et concevoir une première application très simple.

Pour développer, vous avez besoin d'un IDE (Environnement de Développement Intégré) appelé Xcode. Ce logiciel n'est pas uniquement dédié au développement iPhone, vous pouvez aussi l'employer pour développer des applications en Java ou en C++. Dans notre cas, nous allons écrire nos applications avec le langage Objective-C.

Xcode s'installe à partir du DVD d'installation système reçu avec votre Mac. Après installation, il sera rangé par défaut dans le dossier Macintosh HD/Developer/Applications. Dans notre cas, il a dû s'installer en même temps que le SDK (voir Fiche 2).

En lançant cet outil, vous devriez avoir une fenêtre ressemblant à la Figure 4.1 :



Figure 4.1 : La fenêtre d'accueil de Xcode.

Comme c'est votre première utilisation de Xcode, la fenêtre Recent Projects est vide. Pour vous familiariser avec Xcode, puis avec Interface Builder, vous allez construire votre première application : un Hello iPUP !

Commençons par créer un nouveau projet : ouvrez File > New project ou cliquez, dans la fenêtre d'ouverture, sur Create a new project ($\text{⌘}+\text{N}$). Vous avez le choix entre plusieurs modèles (*templates*). Ce sont en fait des projets avec une architecture toute prête. Ils sont très utiles lorsque vous débutez, car ils vous aident à comprendre et construire correctement une architecture de vue.

En voici le détail :

- **Navigation-based Application.** Ce projet contient une navigation bar (barre de navigation) déjà mise en place. Une barre de navigation (`UINavigationBar` gérée par un `UINavigationController`) est très pratique quand vous affichez des listes de données (`table view`), à l'instar de celle du carnet d'adresses sur votre iPhone (`UITableView`).
- **OpenGL ES Application.** Une vue OpenGL ES déjà configurée, avec un carré rempli d'un dégradé qui tourne sur lui-même.
- **Tab Bar Application.** Un tab bar ou barre d'onglets est une barre située en bas de l'écran vous permettant de choisir parmi plusieurs vues par un simple clic. L'application iPod par exemple en compte un. Un tab bar est géré par un `UITabBarController`.
- **Utility Application.** Vous aurez très certainement besoin de faire des transitions entre vos vues, et cette base de projet vous propose une solution afin de réaliser une animation de type flip (retournement 3D de 180°).

- **View-Based Application.** En commençant par ouvrir un projet de ce type, vous aurez une vue qui s'affichera par-dessus la fenêtre (`UIWindow`). Cette vue sera gérée par un `viewController` (`UIViewController`) chargé depuis un fichier `.xib` (fichier utilisé avec Interface Builder).

Info

Vous trouverez parfois écrit *fichier nib*. C'est un fichier `.xib`, mais avec l'ancienne dénomination pour NextStep Interface Builder.

- **Window-based Application.** Le projet de base, complètement épuré.

Pour notre part, nous allons créer un projet Window-based Application, que nous nommerons HelloiPuP. Vous arrivez sur la fenêtre de la Figure 4.2.

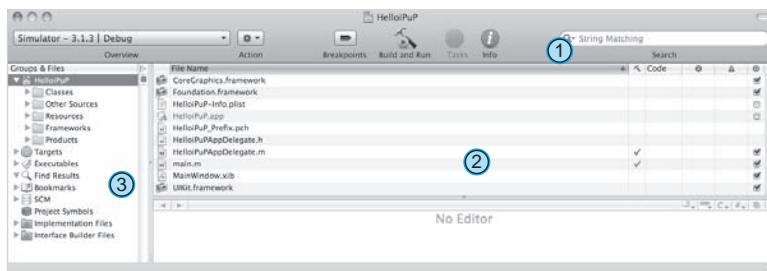


Figure 4.2 :
Fenêtre d'accueil
du projet
Hello iPuP.

Examinons la barre de menu (zone ①) :

- **Overview.** Permet de choisir la cible de compilation, ainsi que les modes de compilation :
 - **debug.** Ce mode utilise les points d'arrêts et exécute le code pas à pas.
 - **release.** L'exécution est plus rapide que le mode debug, mais la différence n'est pas toujours visible.
 - **distribution.** Crée l'exécutable en vue de la distribution sur l'App Store.
- **Action.** Propose quelques actions comme Reveal in Finder (pour ouvrir l'emplacement où est rangé votre fichier), également accessibles en cliquant du bouton droit dans l'éditeur.
- **Breakpoints.** Active le mode debug avec les points d'arrêts.
- **Build and run.** Compile et lance votre application.
- **Tasks.** Arrête le déroulement de l'application.
- **Info.**
- **Champ de recherche.**

Descendez votre regard... mais à quoi correspondent tous ces fichiers (zone ②) ?

- **CoreGraphics.framework, Foundation.framework, UIKit.framework.** Vous vous en doutez peut-être, ce sont les librairies fournies par Apple que vous allez utiliser dans votre application.
- **HelloiPuP.app.** C'est l'application que vous pourrez installer sur l'iPhone, si vous avez payé votre licence...

- **HelloiPuP_Prefix.pch.** C'est un autre fichier `include`. Vous n'avez cependant pas besoin de l'inclure dans vos fichiers, car il est compilé séparément.
- **HelloiPuPAppDelegate.h.** C'est un fichier d'en-tête très proche de ce que vous trouvez en C ou C++. Il contient toutes les déclarations.
- **HelloiPuPAppDelegate.m.** C'est ici que vous allez commencer. Ce fichier va contenir les instructions de votre programme.
- **HelloiPuP-Info.plist.** Ce fichier contient diverses informations à propos de votre programme. Vous n'allez pas vous en servir avant de compiler votre application sur votre iPhone, iPod Touch ou iPad.
- **main.m.** Comme dans beaucoup de langages, ce fichier contient la fonction `main`. C'est là où l'exécution du programme débute. Typiquement, ce `main` lance le programme. Vous ne devriez pas avoir besoin de l'éditer.
- **MainWindow.xib.** Ce fichier contient l'interface graphique de votre fenêtre. En double-cliquant dessus, vous lancerez Interface Builder pour dessiner votre interface.

Regardons ensuite la colonne de gauche (zone ③), nommée Groups & Files. Vous y trouvez notamment :

- **Votre projet “HelloiPuP”.** Dans ce projet, plusieurs dossiers :
 - **Classes.** Vos classes. Il n'y a ici que votre application delegate.
 - **Other sources.** Le “prefix header” et le main.
 - **Ressources.** Les ressources pour votre projet. Il vous faudra glisser ici les ressources dont vous aurez besoin dans votre application (images, vidéos, fichiers xib...).
 - **Frameworks.** Les frameworks à ajouter à votre projet.
 - **Products.** Typiquement, votre .app qui contient l'exécutable.
- **Targets.** La cible lorsque vous compilez.
- **Executable.** L'exécutable de l'application après compilation.
- **Find Results.** L'historique de vos recherches.
- **Bookmarks.** Liste l'ensemble des références ajoutées manuellement à votre code.
- **SCM (Software Control Management).** Sert quand vous souhaitez utiliser un outil de gestion de version (Subversion, Perforce et SCV sont supportés).
- **Project symbols.** Liste tout ce qui est utilisé dans votre projet : éléments d'interface, classes, noms de méthode, instances d'objets...
- **Des dossiers “intelligents”.** Ils regroupent vos fichiers d'implémentation (.m) ou les fichiers interface builder (.xib).

Cliquez sur Classes puis sur HelloIPAppDelegate.m, qui se trouve dans le dossier sélectionné. Du code apparaît dans la fenêtre d'édition comme à la Figure 4.3, et la barre de fonction s'active.

Vous y trouvez, de gauche à droite, des flèches pour naviguer dans l'historique des fichiers consultés (1), l'historique (2), la liste des méthodes (3), la liste des signets (Edit/Add to Bookmarks (⌘+D)) (4), la liste des points d'arrêts (5), les classes et superclasses, ainsi que les catégories (ex. : MacClasse(Animation)) (6), la liste des fichiers utilisant la classe éditée (7), le bouton vous permettant de naviguer aisément entre .h/.m (8).

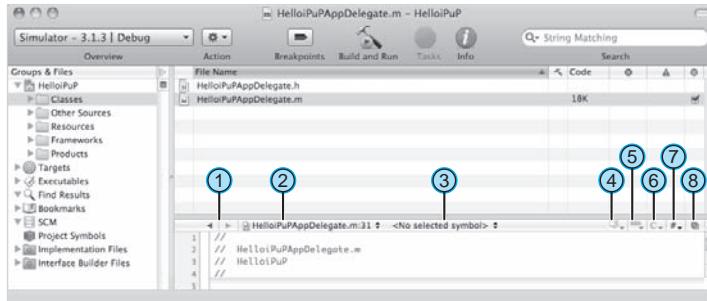


Figure 4.3 : Explication des boutons intégrés.

Maintenant que les bases sont posées, codons un peu !

5

Coder quelques éléments simples

Nous allons ajouter le label `UILabel` à notre vue afin d'afficher le texte "HelloiPad".

Pour cela, il faut modifier le fichier `HelloiPadAppDelegate.h` :

```
#import <UIKit/UIKit.h> ①

@interface HelloiPadAppDelegate : NSObject <UIApplicationDelegate> { ②
    IBOutlet UILabel *helloLabel; ③
    UIWindow *window; ④
}

@property (nonatomic, retain) IBOutlet UIWindow *window; ⑤
@end
```

Attardons-nous un peu sur ce code :

La ligne ① est l'équivalent de `#include`.

La ligne ② déclare la classe `HelloiPadDelegate` héritant de `NSObject` et qui est delegate de `UIApplicationDelegate`. La structure étant la suivante :

```
@interface MaClasse : SuperClass <Delegate1, Delegate2, ...>
```

À la ligne ③, le mot-clé `IBOutlet` lie votre objet à Interface Builder.

La ligne ④ est la déclaration de la première vue dans la pile de vues : votre fenêtre !

La ligne ⑤ facilite l'écriture et clarifie le code grâce à un mécanisme très pratique remplaçant les `getters` et `setters`. Les `nonatomic` et `atomic` sont des paramètres spécifiant le comportement de votre objet en multithreading. Concrètement, utiliser `atomic` sous-entend que l'accès à cet objet sera bloqué par le `thread` appelant (aucun autre ne pourra y accéder si l'objet reste verrouillé). Si deux threads veulent modifier l'objet, ils le feront l'un après l'autre. Avec `nonatomic`, le comportement ne sera pas garanti, mais l'accès sera plus rapide.

Vous pouvez ensuite spécifier le comportement mémoire, en choisissant parmi les paramètres `retain`, `copy` ou `assign`. Nous ne nous y attarderons pas ici, mais je vous invite à vous y pencher le plus rapidement possible. En effet, il n'y a pas de `garbage collector` sur l'iPhone ! Il faut donc gérer votre mémoire très précisément !

Info

Garbage collector est traduit en français par ramasse-miettes. C'est un sous-système qui gère la mémoire : dès lors qu'une ressource allouée est inutilisée, il la supprime de la mémoire.

Passons maintenant au fichier `.m`.

Modifiez-le ainsi :

```
#import "HelloiPadAppDelegate.h"
@implementation HelloiPadAppDelegate ①
@synthesize window; ②
```

```
- (void)applicationDidFinishLaunching:(UIApplication *)application { ③
    [helloLabel setText:@"Hello iPuP"]; ④
    // équivalent : helloLabel.text = @"Hello iPuP";
    [window makeKeyAndVisible]; ⑤
}

- (void)dealloc { ⑥
    [window release]; ⑦
    [super dealloc]; ⑧
}
@end ⑨
```

Les lignes ① à ⑨ permettent d'implémenter les différentes méthodes de la classe.

② est obligatoire lorsque vous écrivez `@property` dans le fichier .h.

③ est une méthode automatiquement lancée lorsque le chargement de l'application est terminé.

④ signifie que l'on met le texte "Hello iPuP" dans le label `helloLabel`.

⑤ rend la fenêtre (`window`) visible.

⑥ à ⑧ méthode `dealloc` : méthode appelée lorsque l'objet de classe est `release`, c'est-à-dire libéré de la mémoire. Par exemple, si vous aviez un objet de la classe Chien alloué par une instance de la classe Ferme avec le mécanisme habituel `Chien *unChien = [[Chien alloc] init];`, un `[unChien release];` appellera automatiquement la méthode `dealloc` de la classe Chien. Dans cette méthode, il faut donc libérer **tous** les objets encore en mémoire.

Ici, `window` est `release` ⑦, puis le `dealloc` de `super` (la classe dont on hérite, `NSObject` ici) est appelé.

Interface Builder est un outil de développement proposé par Apple pour faciliter les créations d'Interface Homme-Machine (IHM). Bien entendu, vous pouvez tout faire en dur, c'est-à-dire par la programmation. Parfois même, vous n'aurez pas le choix, mais vous vous en rendrez compte le moment venu !

Pour l'instant, ouvrez MainWindow.xib dans Resources en double-cliquant dessus.

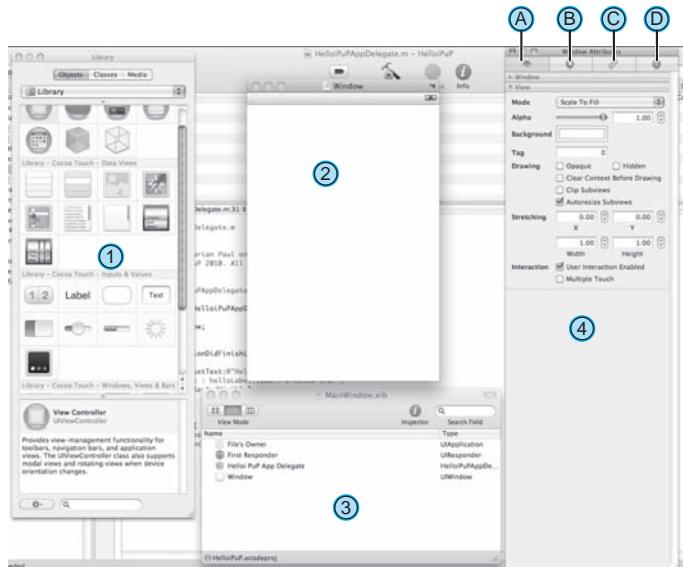


Figure 6.1 : Fenêtres d'Interface Builder.

À gauche ①, vous découvrez la librairie, qui contient tous les éléments du UIKit que vous pourrez utiliser dans votre application.

Au centre ②, votre vue (ici window) telle qu'elle apparaîtra sur votre iPhone.

La fenêtre en dessous ③ contient tout ce dont vous aurez besoin pour faire marcher correctement Interface Builder :

- **File's owner** (propriétaire du fichier). C'est un objet qui établit la communication entre les fichiers xib et les éléments de votre application. L'avantage de la substitution, c'est que les connexions et configurations faites dans le File's owner sont appliquées à l'objet réel quand on le charge.
- **First responder**. Lorsque votre objet est first responder, il sera le premier à intercepter les événements, tels que copier/coller, manipulations de texte, actions que vous avez créées, etc. Le First responder est changé dynamiquement en fonction de la hiérarchie des vues notamment. Si la vue qui est first responder ne répond pas à un événement, alors cet événement traversera la hiérarchie des vues jusqu'à être intercepté.

Enfin, à droite ④, c'est l'Inspecteur, avec 4 onglets : Attributs ⑤, Connections ⑥, Size (taille) ⑦, Identity (identité) ⑧.

Plaçons notre label en faisant un glisser-déposer depuis la librairie.

Puis centrez-le, comme à la Figure 6.2.

Attention

Une vue iPhone fait 320×480 pixels. Avec une barre de statut, elle est réduite à 320×460 pixels. Une vue iPad quant à elle, fait 768×1024 pixels.



Figure 6.2 : Positionnement du label par un glisser-déposer.

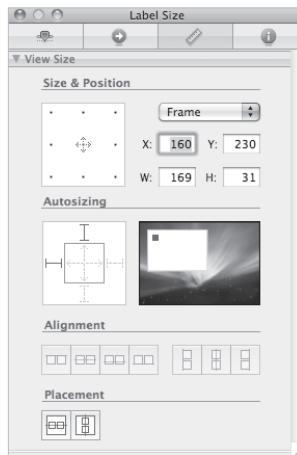


Figure 6.3 : L'Inspecteur de la taille du label.

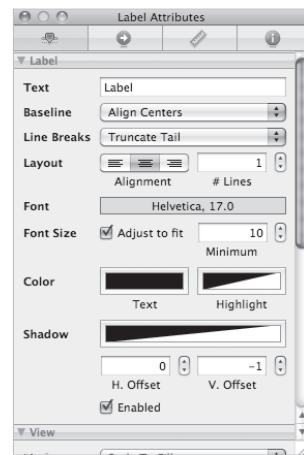


Figure 6.4 : L'Inspecteur des attributs du label.

Centrons le texte en choisissant Align Centers pour l'option Baseline dans l'onglet Attributs de l'Inspecteur (voir Figure 6.4).

Enfin, pour connecter le label qui est sur la vue au helloLabel qui appartient à votre delegate, cliquez sur Hello iPuP App Delegate puis rendez-vous sur l'onglet Connections dans l'Inspecteur. Repérez helloLabel, et cliquez sur le rond à côté. Maintenez et glissez-le jusqu'au label présent dans la vue. Relâchez, sauvegardez et quittez !

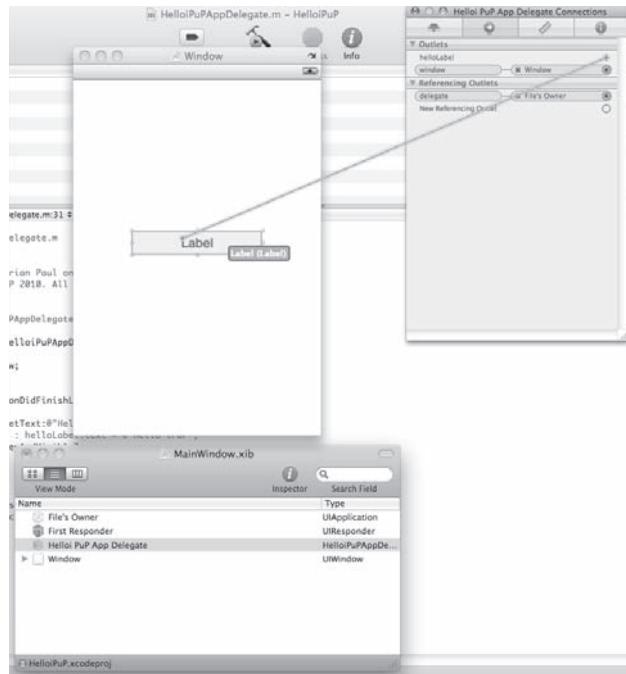


Figure 6.5 : Connexion du label.

De retour sous Xcode, faites un Build/Build and Run ($\text{⌘}+\text{R}$) et...Tadamm !



Figure 6.6 : Votre première application !

Vous pouvez choisir de compiler sur simulateur ou sur **device** (votre iPhone, iPod Touch ou iPad). Pour cela, il faut choisir la cible dans l'Overview (voir Fiche 4). Pour compiler sur device, il faut avoir payé la licence de développement et fait les provisioningnages nécessaires (voir Fiche 2).

Compiler uniquement sur simulateur peut se révéler très dangereux, pour les raisons suivantes :

- Le simulateur tourne sur votre Mac, donc comparé à votre iPhone, ses ressources sont illimitées ! Cela signifie, par exemple, des animations (en apparence) très fluides, un temps de chargement très court, aucune limite dans le nombre d'images chargées en mémoire, etc.
- La gestion mémoire n'est pas la même, notamment en ce qui concerne le mécanisme **d'allocation/désallocation**.
- Vous n'avez pas accès à l'accéléromètre et gyroscope, ni au compas, ni au capteur de proximité.

Attention

Au fur et à mesure que vous avancez dans votre projet, il ne faut pas compiler uniquement sur simulateur, sans tester sur iPhone. Les caractéristiques ne sont absolument pas les mêmes. Si, lors de la phase de développement, vous compiler sur simulateur uniquement, votre application risque fort de ne pas avoir le comportement attendu lorsque vous la testerez sur iPhone !

CHAPITRE 3

APPRÉHENDER QUELQUES ÉLÉMENTS D'INTERFACE

Votre première application iPhone en main, vous voulez aller plus loin... En effet, vous sentez bien qu'un simple label affichant du texte ne suffira pas pour faire de votre application un futur best-seller ! Ce chapitre va donc vous ouvrir de nouveaux horizons sur les éléments par défaut fournis par Apple dans le kit de développement iPhone (UIKit).

Apple fournit par défaut de nombreux éléments d'interface que vous retrouvez dans les applications préinstallées sur votre iPhone, iPod Touch ou iPad. Parcourons-les !

Vous êtes maintenant capable de vous servir d'Interface Builder pour placer vos éléments d'interface. Le principe est le même que pour le label : se servir du mot-clé `IBOutlet` et relier le tout sous Interface Builder !

Parce que Interface Builder est très pratique pour agencer ses vues quand on est débutant, il faut savoir qu'il est possible de tout construire dans le code. Nous allons donc profiter de cette énumération des éléments d'interface pour vous en montrer le mécanisme !

Info

Nous allons donner quelques bases d'utilisation des éléments du UIKit. Le minimum étant de les afficher à l'écran.

N'hésitez pas à vous référer à leur documentation ainsi qu'au projet d'exemple (`sample code : UICatalog`) pour en savoir plus. Pour l'utilisation de la documentation, rendez-vous à la Fiche 3.

Certains éléments explicités ici seront approfondis dans les fiches suivantes.



Figure 8.1 : Liste des éléments du UIKit.

Astuce

N'oubliez pas la touche `[Echap]` de votre clavier. Par exemple, après avoir écrit `unBouton.`, appuyez sur `[Echap]` pour voir la liste des propriétés de votre bouton que vous pouvez modifier. Bien entendu, tout ceci se retrouve dans la documentation !

Info

Dans la suite, nous allons parcourir les éléments d'interfaces à votre disposition. Vous pouvez créer un nouveau projet de type Window-based Application pour tester les différents morceaux de code qui suivent.

WINDOW

La particularité de l'iPhone ou l'iPad est de n'avoir qu'**une fenêtre de disponible**, contrairement aux ordinateurs où le nombre de fenêtres n'est pas imposé. L'objet de la classe UIWindow est donc à la racine de votre future hiérarchie de vues. En effet, faute de pouvoir disposer de plusieurs fenêtres, la navigation entre les différentes vues de votre application se fera par empilements de vues.

L'initialisation de cette fenêtre doit se trouver dans la méthode suivante, dans votre **application delegate** :

```
- (BOOL)application:(UIApplication *)application
  → didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Création de la fenêtre
    // N'oubliez pas UIWindow *window dans le .h
    window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]]; ①
    // ajouts d'une ou plusieurs vue(s)
    [window addSubview:uneVue]; ②
    // affichage à l'écran de la fenêtre et ses sous vues
    [window makeKeyAndVisible]; ③

    return YES;
}
```

INITWITHFRAME

① initWithFrame vous permet d'initialiser une vue en lui spécifiant sa taille. Frame étant le rectangle dans lequel elle va s'inscrire.

Info

Pour créer votre propre rectangle : CGRectMake(xOrigine, yOrigine, largeur, hauteur). Attention au repère de l'iPhone (Figure 8.2). Rappelez-vous que l'écran de l'iPhone fait 320 × 480 pixels...

② La méthode addSubview: permet d'ajouter la vue passée en paramètre à la pile de vues.

③ Cette ligne est indispensable pour afficher la fenêtre ainsi que toutes ses sous-vues. makeKeyAndVisible est propre à UIWindow seulement.

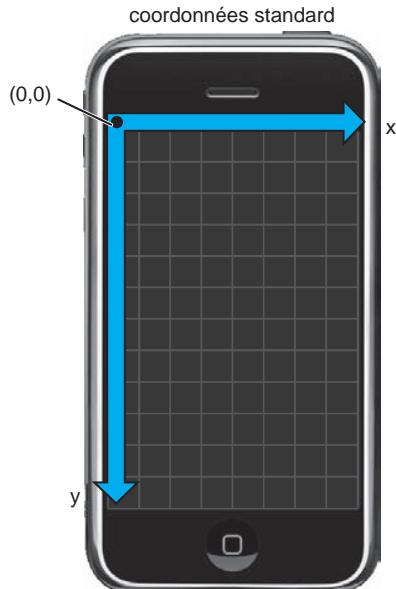


Figure 8.2 : Système de coordonnées de l'iPhone pour le UIKit.

Info

Généralement, on ne crée jamais une fenêtre par le code. D'ailleurs, vous aurez remarqué qu'en créant le projet de type Window-based Application, la fenêtre est déjà instanciée par le fichier window.xib (avec le mot-clé **IBOutlet**).

VIEW

Une fois que votre fenêtre s'affiche, vous pouvez lui ajouter des vues comme à la Figure 8.3 (visualisez-le comme un empilement). Tous les éléments du UIKit sont des vues, vous pouvez donc créer votre propre vue (par exemple votre propre liste déroulante) en partant de la classe **UIView**.

```
- (BOOL)application:(UIApplication *)application  
↳ didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    // initialisation de la vue pour qu'elle remplisse le quart haut/gauche de l'écran  
    UIView *myView = [[UIView alloc] initWithFrame:CGRectMake(0.0, 0.0, 160.0, 240.0)];  
    // on change la couleur du fond de la vue pour qu'il soit bleu  
    [myView setBackgroundColor:[UIColor blueColor]];  
    // on ajoute la vue à la window  
    [window addSubview:myView];  
    [myView release];  
    [window makeKeyAndVisible];  
    return YES;  
}
```

IMAGE VIEW

Cette vue vous permet d'afficher une image à l'écran.

```
- (BOOL)application:(UIApplication *)application  
↳ didFinishLaunchingWithOptions:(NSDictionary *)  
↳ launchOptions {  
    // initialisation de la vue  
    UIImageView *myImageView = [[UIImageView alloc]  
    ↳ initWithFrame:CGRectMake(20.0,  
    ↳ 30.0, 100, 100.0)];  
    // on crée une image à partir de celle mise en  
    ↳ ressource dans le projet  
    UIImage *logoImage = [UIImage  
    ↳ imageNamed:@"logo_ipup_losange.png"]; ①  
    myImageView.image = logoImage; ②  
    // équivalent à [myImageView setImage:logoImage];  
    // on ajoute la vue à la window  
    [window addSubview:myImageView];  
    [myImageView release];  
    [window makeKeyAndVisible];  
    return YES;  
}
```

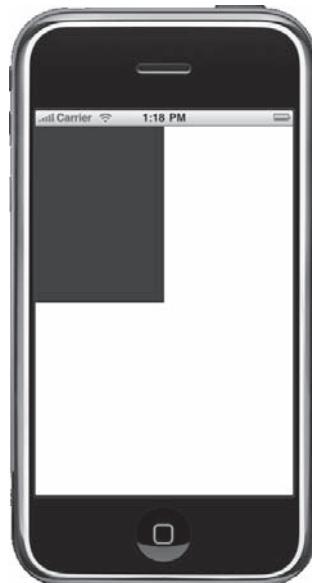


Figure 8.3 : Votre première vue !

① Pour afficher la vue sur l'écran, il faut d'abord créer un objet `UIImageView` à partir d'un fichier, ici `logo_ipup_losange.png`.

② À cet endroit, on spécifie l'image à afficher dans l'image view.

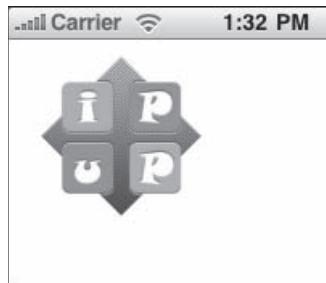


Figure 8.4 : Le logo iPuP sur le simulateur..

Info

Vous pouvez changer le comportement de l'image pour qu'elle s'adapte ou non à votre rectangle défini avec `frame` (voir Figure 8.5).

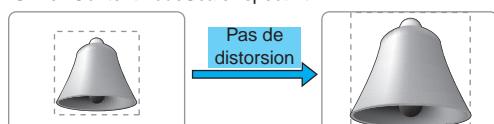
```
myImageView.contentMode = UIViewContentModeScaleAspectFill;
```

Vous pouvez également recourir à une image view pour réaliser de petites animations : dans la documentation Apple : Class `UIImageView` > Tasks > Animating images

UIViewContentModeScaleAspectFill



UIViewContentModeScaleAspectFit



UIEdgeInsets

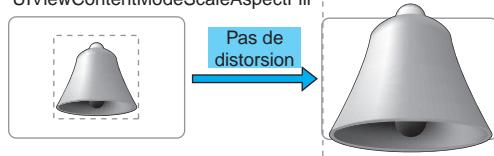


Figure 8.5 : Quelques modes de mise en forme de l'image.

WEB VIEW

Une Web view vous permet de charger du contenu web, depuis Internet ou localement.

```
- (BOOL)application:(UIApplication *)application  
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    // on récupère la taille de l'écran  
    CGRect webFrame = [[UIScreen mainScreen] applicationFrame];  
    UIWebView *myWebView = [[UIWebView alloc] initWithFrame:webFrame];  
    myWebView.backgroundColor = [UIColor whiteColor];  
    myWebView.scalesPageToFit = YES; ①  
    [window addSubview: myWebView];  
  
    [myWebView loadRequest:[NSURLRequest requestWithURL:[NSURL  
        URLWithString:@"http://www.ipup.fr/"]]];  
    [myWebView release];  
    [window makeKeyAndVisible];  
    return YES;  
}
```

① Lorsque scalesPageToFit est à YES, la page chargée est directement zoomée pour être affichée entièrement et le zoom par l'utilisateur est autorisé.

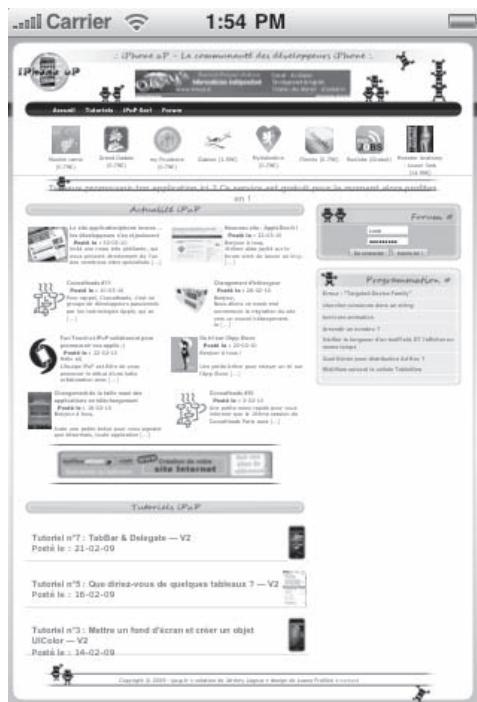


Figure 8.6 : La page iPU sur votre iPhone.

MAP VIEW

Cette vue est très pratique dès lors que vous souhaitez afficher des positions GPS sur l'écran.

Info

N'oubliez pas d'inclure le framework MapKit et faire l'import suivant (voir Fiche 14) :

```
#import <MapKit/MKMapView.h>
```

```
- (BOOL)application:(UIApplication *)application  
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    CGRect mapFrame = [[UIScreen mainScreen] applicationFrame];  
  
    MKMapView *mapView = [[MKMapView alloc] initWithFrame:mapFrame];  
    mapView.showsUserLocation = YES; ①  
    [window addSubview:mapView];  
    [mapView release];  
    [window makeKeyAndVisible];  
    return YES;  
}
```

① showUsersLocation à YES permet de situer directement l'utilisateur quelque part dans le monde et lui montrer sa position sur la carte. Non, je ne suis pas aux États-Unis...



Figure 8.7 : Affichez la position de l'utilisateur sur votre écran.

TEXT VIEW

Cette vue affiche du texte sans limite de taille. Si le texte dépasse la taille de l'écran, des ascenseurs sur le côté indiqueront votre position dans la vue.

```
- (BOOL)application:(UIApplication *)application  
↳ didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    CGRect textFrame = [[UIScreen mainScreen] applicationFrame];  
  
    UITextView *textView = [[UITextView alloc] initWithFrame:textFrame];  
    textView.text = @"Vous pouvez écrire plusieurs lignes de texte. Pour un  
    ↳ retour à la ligne, utilisez simplement le caractère \"\\n\". Par exemple :  
    ↳ \"\\n\\n\" \\n\\n vous donne 2 retours à la ligne";  
    textView.font = [UIFont fontWithName:@"Helvetica" size:17.0];  
    [window addSubview:textView];  
    [textView release];  
    [window makeKeyAndVisible];  
    return YES;  
}
```

Info

N'oubliez pas que le caractère \ permet d'échapper les caractères spéciaux.

Par défaut, une textView est éditable. Vous pouvez l'empêcher en mettant editable à NO.

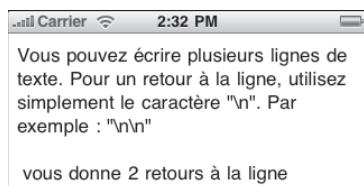


Figure 8.8 : Affichez des lignes de texte.

BOUTON

Est-il nécessaire d'expliquer son utilité ? Voici comment est créé un bouton très facilement :

```
- (BOOL)application:(UIApplication *)application  
↳ didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect]; ①  
    button.frame = CGRectMake(20.0, 30.0, 120.0, 30.0);  
    [button setTitle:@"Touche moi !" forState:UIControlStateNormal]; ②  
    [window addSubview:button];  
    [window makeKeyAndVisible];  
    return YES;  
}
```

① Vous remarquerez que c'est une méthode de classe. En fait, au lieu d'allouer et initialiser un bouton, puis appeler [unBouton uneMethode]; vous utilisez une méthode dite "de classe" de UIButton. Ainsi, l'objet UIButton retourné est en autorelease.

Info

Exemple d'une méthode de classe : (vous remarquerez le +)

```
+ (UIButton*) buttonWithType:(UIButtonType)buttonType;
```

② Un bouton peut avoir plusieurs états : UIControlStateNormal, UIControlStateSelected, UIControlStateHighlighted,... Pour chaque état, on peut définir une image, une couleur, un titre...



Figure 8.9 : Un bouton qui attend qu'on le touche.

TEXTFIELD

Cet objet vous servira pour entrer du texte. Vous pouvez spécifier le type du clavier (entier, à chiffres seulement...), si l'autocorrection est activée ou non (autocorrectionType), si l'entrée du texte est dite sécurisée (les points) : secureTextEntry.

```
- (BOOL)application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    UITextField *textField = [[UITextField alloc]
      initWithFrame:CGRectMake(20.0, 40.0, 160.0, 30.0)];
    textField.textColor = [UIColor blackColor];
    // on ajoute un contour rond autour du textField
    textField.borderStyle = UITextBorderStyleRoundedRect;
    textField.font = [UIFont systemFontOfSize:17.0];
    textField.placeholder = @"<entrez du texte>";
    textField.backgroundColor = [UIColor whiteColor];
    // on utilise le clavier par défaut, qui est entier
    textField.keyboardType = UIKeyboardTypeDefault;
    // on spécifie le type du bouton en bas à droite du clavier
    textField.returnKeyType = UIReturnKeyDone;
    // mettre un bouton 'x' à la droite pour effacer
    textField.clearButtonMode = UITextFieldViewModeWhileEditing;
    [window addSubview:textField];
    [textField release];
    [window makeKeyAndVisible];
    return YES;
}
```

Info

Pour rétracter le clavier, rendez-vous à la Fiche 10 !

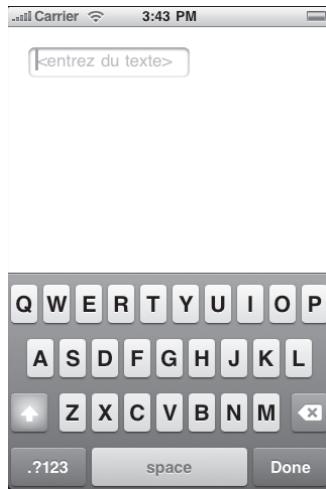


Figure 8.10 : Un champ de texte et son clavier.

SLIDER

Le slider est un élément qui autorise l'utilisateur à choisir une valeur entre un maximum et un minimum en le faisant glisser du doigt.

```
- (BOOL)application:(UIApplication *)application  
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    UISlider *slider = [[UISlider alloc]  
        initWithFrame:CGRectMake(20.0, 40.0, 160.0, 30.0)];  
  
    slider.minimumValue = 0.0;  
    slider.maximumValue = 100.0;  
  
    [window addSubview:slider];  
    [slider release];  
    [window makeKeyAndVisible];  
    return YES;  
}
```

Info

Pour récupérer la valeur du slider : `slider.value`.



Figure 8.11 : Un simple slider.

ACTIVITY INDICATOR VIEW

Cette vue avertit l'utilisateur qu'un traitement est en cours, par exemple quand l'application charge des données. Il est ainsi rassuré et ne pense pas qu'il s'agit d'un bogue.

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // on met la window en gris pour voir l'activity indicator view
    window.backgroundColor = [UIColor grayColor];
    UIActivityIndicatorView *actView = [[UIActivityIndicatorView alloc]
        initWithActivityIndicatorStyle:UIActivityIndicatorViewStyleWhite];
    actView.center = CGPointMake(40.0, 50.0); ①
    // on peut faire un actView.center = window.center;
    [actView startAnimating]; ②
    [actView setHidesWhenStopped:YES]; ③
    [window addSubview:actView];
    [actView release];
    [window makeKeyAndVisible];
    return YES;
}
```

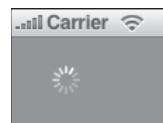


Figure 8.12 : Une roue tournante.

① On définit ainsi le centre de la vue, qui est un CGPoint.

② On lance l'animation de l'Activity indicator view.

③ Lorsque l'on stoppera l'animation, l'Activity indicator view sera immédiatement cachée (setHidden:YES).

Info

Vous aurez très certainement besoin de détacher un thread pour charger des données, et animer l'Activity indicator view sur le thread principal. La notion de thread est assez complexe. Comprenez qu'avec ce mécanisme, vous serez capable de charger des données en tâche de fond.

Rappelez-vous cependant qu'une modification d'éléments du UIKit (les éléments graphiques donc) doit se faire sur le thread principal.

La gestion des threads n'est pas évidente, vous devez avoir une bonne maîtrise de la gestion des ressources. N'hésitez pas à lire les guides proposés par Apple s'y référant.

Voici un exemple d'utilisation des threads, qui vous sera certainement très pratique :

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    window.backgroundColor = [UIColor grayColor];
    // déclarer actView dans le .h
    actView = [[UIActivityIndicatorView alloc]
        initWithActivityIndicatorStyle:UIActivityIndicatorViewStyleWhite];
    actView.center = CGPointMake(40.0, 50.0);
    [self performSelectorInBackground:@selector(backgroundMethod) withObject:nil]; ①
    [actView startAnimating];
    [actView setHidesWhenStopped:YES];
```

```

[window addSubview:actView];

>window makeKeyAndVisible];
return YES;
}

- (void) backgroundMethod {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init]; ②
    // chargement des données ici
    // lorsque les données sont chargées
    [self performSelectorOnMainThread:@selector(backgroundMethodDone)
        ↪ withObject:nil waitUntilDone:NO]; ③
    [pool release];
}

- (void) backgroundMethodDone {
    [actView stopAnimating]; ④
    // et ici continue le déroulement de votre programme
}

```

Attention

Si vous recopiez ce code, rien ne s'affichera car aucune donnée n'est chargée sur le thread détaché.
Si vous souhaitez observer le comportement, écrivez :

```

- (void) backgroundMethod {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    // chargement des données ici
    // lorsque les données sont chargées
    sleep(5);
    [self performSelectorOnMainThread:@selector(backgroundMethodDone)
        ↪ withObject:nil waitUntilDone:NO];
    [pool release];
}

```

- ① On détache ici un nouveau thread, en tâche de fond, en appelant la méthode `backgroundMethod`.
- ② Puisque l'on est dans un autre thread que le principal, il faut créer un **bassin d'autorelease**. La règle est la suivante : un bassin d'autorelease par thread.
- ③ Après avoir fini de charger les données, il faut revenir sur le thread principal pour continuer le déroulement de l'application.
- ④ Rappelez-vous, les éléments du UIKit ne peuvent être modifiés que sur le thread principal.

Info

Vous pouvez également recourir à une progress view qui est une barre de chargement classique. Le principe est le même avec la gestion des threads. Cependant, il faut modifier manuellement la valeur de la Progress view pour la voir bouger à l'écran.

Maintenant que nous avons parcouru la plupart des éléments d'interface, que diriez-vous de passer à la pratique et ajouter un fond d'écran à notre application iPhone HelloiPuP ?

Tout d'abord, il vous faut trouver une image, de préférence avec ces dimensions : 320 × 460 pixels. Ensuite, glissez-la dans Xcode directement, dans le dossier Ressources, et cochez la case Copy items into destination group's folder.

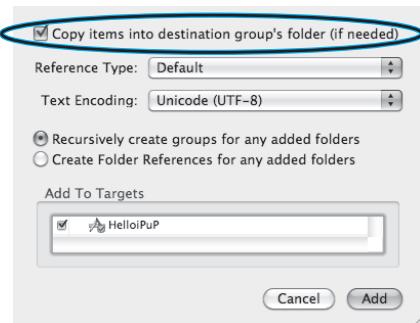


Figure 9.1 : Cochez pour copier l'image dans le projet.

Info

La fenêtre de la Figure 9.1 doit obligatoirement s'afficher. Si ce n'est pas le cas, c'est que vous avez fait une erreur. Ne copiez pas l'image dans le dossier ressources du Finder, mais bien directement dans Xcode.

Pour afficher l'image en fond dans votre application, il faut passer par une UIImageView. Pour vous guider, nous allons examiner les deux méthodes : par le code, ou en utilisant Interface Builder. Faites une copie de votre projet, afin d'avoir un projet par méthode.

AJOUTER L'IMAGE PAR LE CODE

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [helloLabel setText:@"Hello iPUP"];
    // équivalent : helloLabel.text = @"Hello iPUP";
    CGRect imageFrame = [[UIScreen mainScreen] bounds];

    UIImageView *backgroundImageView = [[UIImageView alloc]
        initWithFrame:imageFrame]; ①
    [backgroundImageView setImage:[UIImage imageNamed:@"backgroundImage.png"]];
    [window addSubview:backgroundImageView]; ②
    [backgroundImageView release]; ③

    [window makeKeyAndVisible];
    return YES;
}
```

- ① On alloue l'image view qui fera la taille de l'écran. Son `retain count` vaut 1.
- ② En ajoutant l'image view à la fenêtre (en la mettant sur la pile des vues), vous la placez implicitement dans un tableau (`NSArray`) de vues (accessible avec `[window subviews]`). Ainsi, c'est ce tableau qui va gérer la mémoire de l'objet placé (augmentation du `retain count` de l'objet placé dans le tableau de 1). Ici, à la fin de l'instruction, le `retain count` de `backgroundImageView` vaut donc 2.
- ③ Puisque l'on souhaite toujours ramener les `retain count` des objets à 1, il faut `release backgroundImageView` (décrémenter son `retain count` de 1). `Retain count` de `backgroundImageView` vaut maintenant 1.

Info

Lorsque le `retain count` d'un objet vaut 0, l'emplacement mémoire alloué pour cet objet est libéré.

Lancez ainsi l'application. Vous remarquerez que l'on n'aperçoit plus le label. Ce qui est normal, car rappelez-vous, les vues sont empilées.

Lorsque l'application s'exécute, le fichier `window.xib` va être décodé (c'est un fichier XML). Les éléments d'interface placés à l'aide d'Interface Builder vont être positionnés. Ainsi, le label est placé par-dessus la vue. Ensuite, la méthode `applicationDidFinishLaunching` de l'application delegate va être appelée. Dans cette méthode, nous allouons une `UIImageView` que nous plaçons sur la pile des vues, au-dessus ! Notre image cache donc notre label... Pour y remédier, plusieurs solutions :

```
- (BOOL)application:(UIApplication *)application  
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    [helloLabel setText:@"Hello iPUP"];  
    // équivalent : helloLabel.text = @"Hello iPUP";  
    CGRect imageFrame = [[UIScreen mainScreen] bounds];  
  
    UIImageView *backgroundImageView = [[UIImageView alloc] initWithFrame:imageFrame];  
    [backgroundImageView setImage:[UIImage imageNamed:@"backgroundImage.png"]];  
  
    // Solution 1 : ①  
    [window addSubview:backgroundImageView];  
    [window sendSubviewToBack:backgroundImageView];  
  
    // Solution 2 : ②  
    // [window insertSubview:backgroundImageView atIndex:0];  
  
    [backgroundImageView release];  
  
    [window makeKeyAndVisible];  
    return YES;  
}
```

- ① La première solution consiste à ajouter l'image view à la pile, puis l'envoyer tout en dessous de la pile.

② La deuxième solution vous permet de choisir l'index où ajouter la vue (la position dans la pile de vues, 0 étant le plus bas).

En relançant votre application, votre label apparaît enfin ! Mais il n'est pas très visible en noir... Qu'à cela ne tienne, changeons la couleur du texte en blanc !

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [helloLabel setText:@"Hello iPUP"];
    // équivalent : helloLabel.text = @"Hello iPUP";
    [helloLabel setTextColor:[UIColor whiteColor]]; ①

    CGRect imageFrame = [[UIScreen mainScreen] bounds];

    UIImageView *backgroundImageView = [[UIImageView
        alloc] initWithFrame:imageFrame];
    [backgroundImageView setImage:[UIImage
        imageNamed:@"backgroundImage.png"]];

    [window insertSubview:backgroundImageView atIndex:0];
    [backgroundImageView release];
    [window makeKeyAndVisible];
    return YES;
}
```

① Vous pouvez très facilement choisir parmi des couleurs prédéfinies, comme whiteColor, blackColor, blueColor, grayColor, yellowColor...

Vous pouvez également mettre des couleurs plus précisément :

```
[helloLabel setTextColor:[UIColor colorWithRed:12.0/255.0 green:100.0/255.0
    blue:200.0/255.0 alpha:1.0]];
```

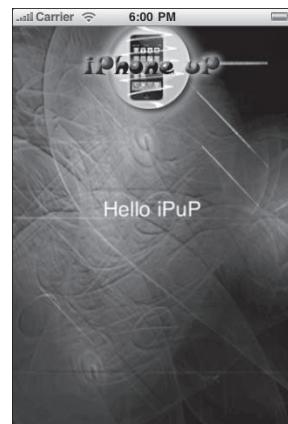


Figure 9.2 : Un label sur un fond d'écran.

AJOUTER L'IMAGE AVEC INTERFACE BUILDER

Pour ajouter l'image de fond avec Interface Builder, faire comme suit :

1. Repartir du projet que vous aviez lorsque vous avez commencé cette fiche.
2. Ouvrir MainWindow.xib et glisser-déposer une UIImageView sur la window. Dans l'onglet Frame de l'image view ainsi déposée, remplir comme à la Figure 9.3.

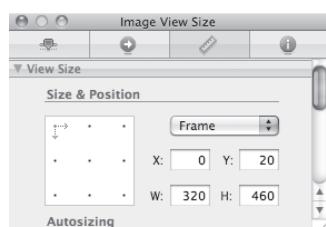


Figure 9.3 : Modification de la frame de l'image view.

3. Cliquer dans l'Inspecteur sur l'onglet Attributes et choisir l'image (voir Figure 9.4).

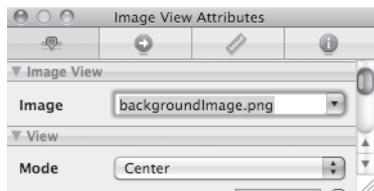


Figure 9.4 : Mettez l'image dans l'image view.

Il faut maintenant repositionner correctement l'image view dans la pile des vues. Pour cela, ranger par liste les éléments de MainWindow.xib (voir ① de la Figure 9.5). Déplacer ensuite l'image view depuis cette fenêtre en la glissant-déposant au-dessus du label, toujours dans la même fenêtre (voir ② de la Figure 9.5).

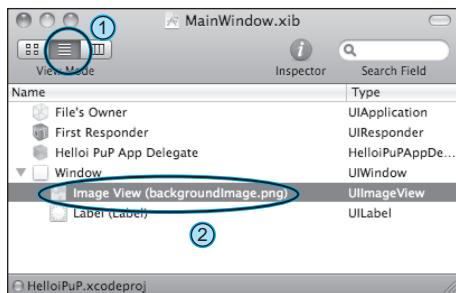


Figure 9.5 : Placez l'image view en fond d'écran.

Une alternative est de passer par le menu Layout > Send Backward ou Layout > Send To Back.

Enfin, pour changer la couleur du texte du label, cliquez dessus. Ensuite, dans l'Inspecteur > onglet Attributes mettre la couleur du texte en blanc (voir ① de la Figure 9.6).

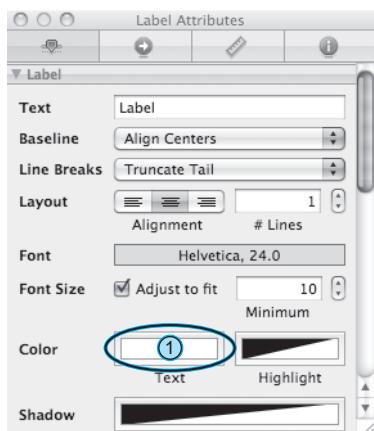


Figure 9.6 : Changez la couleur de la police.

Vous savez placer différents éléments d'interface. Toutefois, si vous placez un bouton, il ne fait pas grand-chose...

Perçons le mystère des boutons. Ici encore, deux solutions : passer par le code, ou utiliser Interface Builder. Nous allons voir les deux méthodes possibles.

UTILISER LE CODE

Recréons un nouveau projet Window Based Application. Nommez-le EventHandling.

Faites ensuite Files > New File. Choisissez iPhone OS > Cocoa Touch Class > UIViewController Subclass et laissez coché With XIB for user interface. Nommez cette classe RootViewController.

Rangez vos fichiers comme à la Figure 10.1. Pour ajouter un dossier, cliquez du bouton droit sur Classes puis Add > New Group.

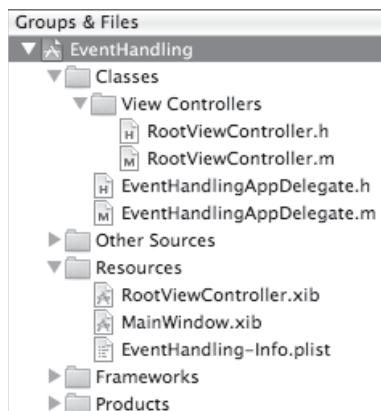


Figure 10.1 : Rangement des fichiers.

Dans le code de votre application delegate, modifiez le fichier ainsi :

```
EventHandlingAppDelegate.h
#import <UIKit/UIKit.h>
#import "RootViewController.h"

@interface EventHandlingAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    RootViewController *rootViewController;
}

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

@end
```

```

EventHandlingAppDelegate.m
#import "EventHandlingAppDelegate.h"

@implementation EventHandlingAppDelegate

@synthesize window;
@synthesize rootViewController;

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    RootViewController *viewController = [[RootViewController alloc]
        initWithNibName:@"RootViewController" bundle:nil]; ①
    self.rootViewController = viewController; ②
    [viewController release]; ③

    rootViewController.view.frame = [[UIScreen mainScreen] bounds];
    [window addSubview:rootViewController.view]; ④

    [window makeKeyAndVisible];
    return YES;
}

- (void)dealloc {
    [rootViewController release];
    [window release];
    [super dealloc];
}
@end

```

① On initialise un objet de la classe RootViewController en lui spécifiant son nom de nib (fichier .xib). Mettre à nil le **bundle** sous-entend qu'il cherchera le .xib dans le **main bundle**. Je vous invite à regarder la documentation afin de comprendre l'architecture du dossier de votre application.

② Grâce au **@property** (setter/getter), l'objet **rootViewController** déclaré dans le .h fait référence à l'objet **viewController** alloué en ① avec un mécanisme implicite de retain count + 1, d'où le **release** de **viewController** en ③.

En ④ on ajoute la vue du **rootViewController** à la fenêtre.

Nous allons ajouter un bouton, comme à la Fiche 8, dans la classe **RootViewController**, dans le fichier d'implémentation :

```

- (void)viewDidLoad {
    UIButton *aButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    aButton.frame = CGRectMake(20.0, 30.0, 120.0, 30.0);
    [aButton setTitle:@"Touche moi !" forState:UIControlStateNormal];
    [aButton setTitle:@"Encore ?" forState:UIControlStateSelected];
    [aButton addTarget:self action:@selector(buttonHandleMethod:)
        forControlEvents:UIControlEventTouchUpInside]; ①
}

```

```

        [self.view addSubview:aButton];
        [super viewDidLoad];
    }
    // Méthode pour gérer les événements du bouton
    - (void) buttonHandleMethod:(id)sender { ②
    }
}

```

① Cette ligne permet de choisir la méthode appelée lorsqu'un certain type d'événement sur le bouton apparaît. Ici, nous avons choisi Touch up inside qui interceptera l'événement suivant : un "après clic" dans le bouton (lorsque le doigt s'en va du bouton).

② La méthode appelée lorsqu'on cliquera sur le bouton. Remarquez le premier paramètre : (id) sender : ce sera le bouton à l'origine de l'événement.

Implémentons cette méthode :

```

// Méthode pour gérer les événements du bouton
- (void) buttonHandleMethod:(id)sender {
    NSLog(@"bouton cliqué !"); ①
    UIButton *theButton = (UIButton*)sender; ②
    [theButton setSelected:!theButton isSelected]]; ③
}

```

① Le NSLog vous permet d'afficher du texte dans la console de debug, très pratique !

② On fait une référence sur le `sender`, que l'on caste en `UIButton`.

③ Au premier abord, cette ligne paraît compliquée. Il n'en est rien : on met le booléen `selected` du bouton à la valeur différente de `selected` (YES devient NO ; NO devient YES).

Compilez et lancez l'application, cliquez sur le bouton, son titre va changer !

RECOMMENCER EN UTILISANT INTERFACE BUILDER

Recommencez un nouveau projet et allez jusqu'à l'étape de création du `RootViewController`. Ensuite, écrivez ce code dans votre application delegate :

```

#import <UIKit/UIKit.h>
#import "RootViewController.h"

@interface EventHandlingAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    RootViewController *rootViewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet RootViewController *rootViewController; ①

@end

```

① Remarquez le mot-clé `IBOutlet`.

Dans le .m :

```
#import "EventHandlingAppDelegate.h"

@implementation EventHandlingAppDelegate

@synthesize window;
@synthesize rootViewController;

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [window addSubview:rootViewController.view];
    [window makeKeyAndVisible];
    return YES;
}

- (void)dealloc {
    [rootViewController release];
    [window release];
    [super dealloc];
}

@end
```

Ouvrez MainWindow.xib et ajoutez un objet viewController comme à la Figure 10.2

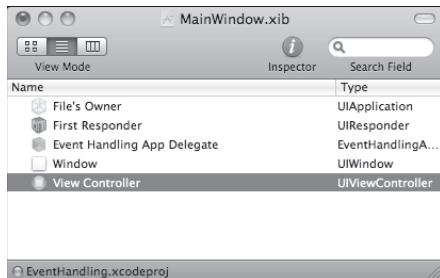


Figure 10.2 : Ajoutez un ViewController.

Cliquez maintenant sur ce view controller puis dans l'onglet Identity de l'Inspecteur. Dans Class, renseignez RootViewController comme à la Figure 10.3.

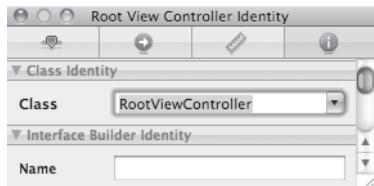


Figure 10.3 : Renseignez le RootViewController.

Ensuite, cliquez sur Event Handling App Delegate et allez dans l'onglet Connections puis relier le rootViewController au rootViewController (voir Figure 10.4)



Figure 10.4 : Reliez le rootViewController.

Sauvez puis allez dans RootViewController.h :

```
#import <UIKit/UIKit.h>
@interface RootViewController : UIViewController {
}
- (IBAction) buttonHandleMethod:(id)sender; ①
@end
```

① Le mot-clé `IBAction` permet de relier une méthode qui répondra à un événement à Interface Builder.

Ouvrez RootViewController.xib et ajoutez un bouton. Cliquez dessus, puis dans l'Inspecteur onglet Attributes, remplissez comme aux Figures 10.5 et 10.6.

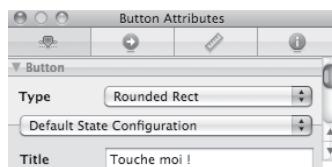


Figure 10.5 : Renseignez le titre pour la configuration par défaut.

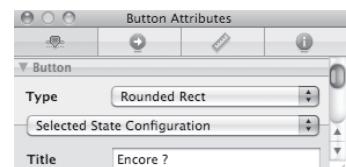


Figure 10.6 : Renseignez le titre pour la configuration sélectionnée.

Enfin, cliquez sur File's owner et allez dans l'onglet Connections de l'Inspecteur. Reliez `ButtonHandleMethod:` dans Received Actions au bouton et choisissez `Touch up inside`. Vous devriez obtenir quelque chose de comparable à la Figure 10.7

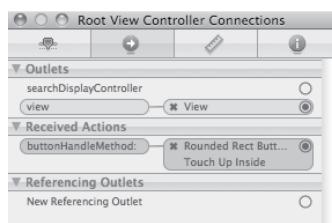


Figure 10.7 : Connexions du RootViewController.

Sauvez et quittez, puis dans RootViewController.m, implémentez comme suit :

```
- (void)viewDidLoad {
    [super viewDidLoad];
}
// Méthode pour gérer les événements du bouton
- (IBAction)buttonHandleMethod:(id)sender {
    NSLog(@"bouton cliqué !");
    UIButton *theButton = (UIButton*)sender;
    [theButton setSelected:[theButton isSelected]];
}
```

Compilez et lancez... Et voilà ! Vous connaissez les deux méthodes. Pour la suite, nous allons préférer la méthode de positionnement par le code. Repartez donc du premier projet créé pour cette fiche.

AJOUTER UN INTERRUPTEUR

Ajoutons maintenant un **switch** (interrupteur) sur la vue de notre rootViewController :

```
RootViewController.h
#import <UIKit/UIKit.h>
@interface RootViewController : UIViewController {
    UIButton *aButton;
    UISwitch *aSwitch;
}
@end
RootViewController.h
- (void)viewDidLoad {
    aButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    aButton.frame = CGRectMake(20.0, 30.0, 120.0, 30.0);
    [aButton setTitle:@"Touche moi !" forState:UIControlStateNormal];
    [aButton setTitle:@"Encore ?" forState:UIControlStateSelected];
    [aButton addTarget:self action:@selector(buttonHandleMethod:)];
    ↪ forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:aButton];

    aSwitch = [[UISwitch alloc] initWithFrame:CGRectMake(20.0, 80.0, 94.0, 27.0)];
    [aSwitch addTarget:self action:@selector(switchHandleMethod:)];
    ↪ forControlEvents:UIControlEventValueChanged]; ①
    [self.view addSubview:aSwitch];

    [super viewDidLoad];
}
// Méthode pour gérer les événements du bouton
- (void)buttonHandleMethod:(id)sender {
    NSLog(@"bouton cliqué !");
    UIButton *theButton = (UIButton*)sender;
```

```

    [theButton setSelected:[theButton isSelected]];
}

// Méthode pour gérer les événements du switch
- (void) switchHandleMethod:(id)sender { ②
    UISwitch *theSwitch = (UISwitch*)sender;
    [aButton setSelected:theSwitch.on];
}
- (void)dealloc {
    [aButton release];
    [aSwitch release];
    [super dealloc];
}
}

```

Vous remarquez que nous avons déclaré dans le .h notre bouton ainsi que le switch, afin d'accéder au bouton dans switchHandleMethod:. En ①, on spécifie que la méthode switchHandleMethod: va être la méthode appelée lorsque le switch changera d'état. En ②, on implémente la méthode qui sera appelée chaque fois que le switch changera d'état. J'ai choisi ici de changer l'état du bouton. La valeur du switch (booléen) est récupérée avec theSwitch.on.

Pour continuer, nous allons jouer avec un slider. Ajoutez donc un slider et un label à votre vue. Le label servira à afficher la valeur du slider.

Dans la méthode viewDidLoad, ajoutez ces quelques lignes :

```

aSlider = [[UISlider alloc] initWithFrame:CGRectMake(20.0, 130.0, 150.0, 20.0)];
aSlider.minimumValue = 0.0;
aSlider.maximumValue = 100.0;
[aSlider addTarget:self action:@selector(sliderHandleMethod:)
    forControlEvents:UIControlEventValueChanged];
[self.view addSubview:aSlider];

aLabel = [[UILabel alloc] initWithFrame:CGRectMake(20.0, 170.0, 150.0, 20.0)];
[self.view addSubview:aLabel];

```

Info

N'oubliez pas les release dans le dealloc et la déclaration dans le .h du slider et du label :

```

UILabel *aLabel;
UISlider *aSlider;

```

Le principe est le même que pour le switch. On observe les changements de valeurs.

Dans la méthode sliderHandleMethod :

```

- (void) sliderHandleMethod:(id)sender {
    UISlider *theSlider = (UISlider*)sender;
    // Notez que l'on pourrait utiliser directement aSlider
    [aLabel setText:[NSString stringWithFormat:@"%.2f", theSlider.value]];
}

```

Compilez et lancez, déplacer votre doigt sur le curseur, le label affiche des valeurs différentes !

Info

Vous aurez sans doute remarqué la syntaxe un peu particulière : “%.2f”. En fait, cela permet de choisir le nombre de chiffres après la virgule à afficher. Ici c'est donc 2 !

AJOUTER UN CHAMP DE TEXTE

Finissons par ajouter un champ de texte (UITextField) à notre vue. Ajoutez dans le .h :

```
UITextField *aTextField;
```

Modifiez la méthode viewDidLoad en y ajoutant :

```
aTextField = [[UITextField alloc] initWithFrame:CGRectMake(20.0, 190.0, 150.0, 30.0)];  
[aTextField setBorderStyle:UITextBorderStyleRoundedRect];  
[aTextField setPlaceholder:@"Entre un chiffre"];  
[aTextField setDelegate:self]; ①  
[aTextField setKeyboardType:UIKeyboardTypeDefault];②  
[aTextField setClearsOnBeginEditing:YES];③  
[self.view addSubview:aTextField];
```

① Voici quelque chose de nouveau ! Le delegate ! Pas de panique, nous allons en parler longuement à la Fiche 11.

② Je vous demande de mettre un clavier de type “par défaut”, alors que nous avons besoin de rentrer des chiffres, car je souhaite vous montrer un mécanisme particulier utile pour la suite !

③ Nous choisissons ici le comportement de votre champ de texte pour que le texte entré soit effacé dès que vous cliquez dedans.

Comme nous avons spécifié que notre classe RootViewController était delegate de UITextField, il faut écrire ceci dans le .h :

```
@interface RootViewController : UIViewController <UITextFieldDelegate> {
```

Ensuite, il faut se tourner vers la documentation pour regarder les méthodes que l'on peut ou doit implémenter lorsque notre classe est delegate de UITextField :

Managing Editing

- textFieldShouldBeginEditing: optional method
- textFieldDidBeginEditing: optional method
- textFieldShouldEndEditing: optional method
- textFieldDidEndEditing: optional method

Editing the Text Field's Text

- textField:shouldChangeCharactersInRange:replacementString: optional method
- textFieldShouldClear: optional method
- textFieldShouldReturn: optional method

Ici, toutes les méthodes sont optionnelles, ce qui signifie que si votre classe RootViewController n'implémente aucune de ces méthodes, le comportement de votre application ne sera pas modifié.

Attention

Si votre classe n'implémente pas une méthode requise, dite `@required`, votre application plantera lors de son appel avec une erreur du type dans la console :

```
*** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '***  
→ [VotreClasse nomDeLaMéthode]: unrecognized selector send to instance adresse mémoire'
```

Info

Lorsqu'une méthode est dite optionnelle, il ne faut pas que l'application plante quand vous demandez au delegate de répondre à l'appel de ladite méthode. Ainsi, on utilise avant l'appel de la méthode :

```
if ([delegate respondsToSelector:@selector(laMethode)]) { [delegate laMethode]; }
```

RÉTRACTER UN CHAMP DE TEXTE

Nous avons besoin de connaître le moment où l'utilisateur clique sur le bouton en bas à droite du clavier (Retour) pour valider son entrée de texte. Ainsi, nous allons pouvoir rétracter le clavier, changer la valeur du slider ainsi que celle du label. Implémentons `textFieldShouldReturn:` dans `RootViewController.m` comme suit :

```
- (BOOL) textFieldShouldReturn:(UITextField *)textField {
    if (textField == aTextField) { ①
        NSString *floatRegEx = @"[0-9]{1,}+[0-9.]{0,}"; ②
        NSPredicate *floatTest = [NSPredicate predicateWithFormat:@"SELF MATCHES
        ↪ %@", floatRegEx];
        BOOL stringMatchingFloat = [floatTest evaluateWithObject:textField.text]; ③

        if (stringMatchingFloat) { ④
            [aSlider setValue:[textField.text floatValue] animated:YES]; ⑤
            NSString *stringToPlace = [NSString stringWithFormat:@"%.2f",
            ↪ aSlider.value];
            [aLabel setText:stringToPlace]; ⑥
            [aTextField setPlaceholder:stringToPlace]; ⑦
            [aTextField resignFirstResponder]; ⑧
        }
        else {
            [aTextField setText:@""]; ⑨
        }
    }
    return YES;
}
```

① Dans le cas où nous aurions plusieurs champs de texte sur la même vue, avec des comportements différents, il est nécessaire de les distinguer. Ainsi, nous testons si le champ de texte en paramètre, celui à l'origine de l'appel de la méthode, correspond bien à `aTextField`.

- ② Pour vérifier que l'utilisateur rentre bien un chiffre uniquement, nous allons faire un `regex` sur le `NSString` entré par l'utilisateur pour savoir s'il contient bien au moins un chiffre entre 0 et 9, puis au moins un point ou chiffre entre 0 et 9 (pour les chiffres après la virgule).
- ③ Le test en question, où nous utilisons `NSPredicate`.
- ④ Si le texte entré correspond bien à un flottant, alors nous passons au ⑤ et mettons la valeur du slider à celle entrée. Notez que nous choisissons d'animer la transition.

Info

Lorsqu'un `string` est par exemple @'12.4', vous pouvez récupérer un flottant grâce à `float aFloat = [@"12.4" floatValue];`. Essayez également `intValue`, `boolValue`, ...

- ⑥ Pour que le tout soit cohérent, il faut changer la valeur du label et mettre la nouvelle valeur du slider. En ⑦, nous mettons le `placeHolder` du champ de texte (le texte en gris affiché avant de saisir le texte) à la valeur du slider également.

La ligne ⑧ est très importante, car elle demande au clavier de se rétracter et donc de disparaître. C'est la raison pour laquelle nous avons choisi un clavier dit "par défaut". En effet, un clavier numérique ne présente pas de bouton Retour.

Enfin, si la chaîne de caractères entrée ne correspond pas à un flottant, le champ de texte est remis à zéro à la ligne ⑨, et le clavier ne se rétracte pas.

Toujours dans l'optique de rester cohérent, nous devons changer le `placeHolder` du champ de texte lorsque nous bougeons le slider. Modifiez ainsi la méthode `sliderHandleMethod` :

```
- (void) sliderHandleMethod:(id)sender {  
    UISlider *theSlider = (UISlider*)sender;  
  
    NSString *stringToPlace = [NSString stringWithFormat:@"%.2f", theSlider.value];  
    [aLabel setText:stringToPlace];  
    [aTextField setPlaceholder:stringToPlace];  
}
```

Encore une fois, compilez et jouez avec votre application !

Pour vous aider, voici à quoi devrait ressembler votre classe `RootViewController` :

```
RootViewController.h  
#import <UIKit/UIKit.h>  
  
@interface RootViewController : UIViewController <UITextFieldDelegate> {  
    UIButton *aButton;  
    UISwitch *aSwitch;  
    UISlider *aSlider;  
    UILabel *aLabel;  
    UITextField *aTextField;  
}  
@end
```

```
RootViewController.m
#import "RootViewController.h"

@implementation RootViewController

- (void)viewDidLoad {
    aButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    aButton.frame = CGRectMake(20.0, 30.0, 120.0, 30.0);
    [aButton setTitle:@"Touche moi !" forState:UIControlStateNormal];
    [aButton setTitle:@"Encore ?" forState:UIControlStateSelected];
    [aButton addTarget:self action:@selector(buttonHandleMethod:)
    ↪ forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:aButton];

    aSwitch = [[UISwitch alloc] initWithFrame:CGRectMake(20.0, 80.0, 94.0, 27.0)];
    [aSwitch addTarget:self action:@selector(switchHandleMethod:)
    ↪ forControlEvents:UIControlEventValueChanged];
    [self.view addSubview:aSwitch];

    aSlider = [[UISlider alloc] initWithFrame:CGRectMake(20.0, 130.0, 150.0, 20.0)];
    aSlider.minimumValue = 0.0;
    aSlider.maximumValue = 100.0;
    [aSlider addTarget:self action:@selector(sliderHandleMethod:)
    ↪ forControlEvents:UIControlEventValueChanged];
    [self.view addSubview:aSlider];

    aLabel = [[UILabel alloc] initWithFrame:CGRectMake(20.0, 170.0, 150.0, 20.0)];
    [self.view addSubview:aLabel];

    aTextField = [[UITextField alloc] initWithFrame:CGRectMake(20.0, 230.0, 150.0, 30.0)];
    [aTextField setBorderStyle:UITextBorderStyleRoundedRect];
    [aTextField setPlaceholder:@"Entre un chiffre"];
    [aTextField setDelegate:self];
    [aTextField setKeyboardType:UIKeyboardTypeDefault];
    [aTextField setClearsOnBeginEditing:YES];
    [self.view addSubview:aTextField];

    [super viewDidLoad];
}

// Méthode pour gérer les événements du bouton
- (void) buttonHandleMethod:(id)sender {
    NSLog(@"bouton cliqué !");
    UIButton *theButton = (UIButton*)sender;
    [theButton setSelected:[theButton isSelected]];
}
```

```

// Méthode pour gérer les événements du switch
- (void) switchHandleMethod:(id)sender {
    UISwitch *theSwitch = (UISwitch*)sender;
    [aButton setSelected:theSwitch.on];
}

// Méthode pour gérer les événements du slider
- (void) sliderHandleMethod:(id)sender {
    UISlider *theSlider = (UISlider*)sender;

    NSString *stringToPlace = [NSString stringWithFormat:@"%.2f", theSlider.value];
    [aLabel setText:stringToPlace];
    [aTextField setPlaceholder:stringToPlace];
}

// Méthode pour gérer les événements du textField
- (BOOL) textFieldShouldReturn:(UITextField *)textField {
    if (textField == aTextField)
    {
        NSString *floatRegEx = @"[0-9]{1,}[0-9.]";
        NSPredicate *floatTest = [NSPredicate predicateWithFormat:@"SELF MATCHES
        ↪%@", floatRegEx];
        BOOL stringMatchingFloat = [floatTest evaluateWithObject:textField.text];

        if (stringMatchingFloat) {
            [aSlider setValue:[textField.text floatValue] animated:YES];

            NSString *stringToPlace = [NSString stringWithFormat:@"%.2f",
            ↪aSlider.value];
            [aLabel setText:stringToPlace];
            [aTextField setPlaceholder:stringToPlace];

            [aTextField resignFirstResponder];
        }
        else {
            [aTextField setText:@""];
        }
    }
    return YES;
}
- (void)dealloc {
    [aButton release];
    [aSwitch release];
    [aSlider release];
    [aLabel release];
    [aTextField release];
    [super dealloc];
}
@end

```

Avant de commencer à approfondir l'utilisation des éléments du UIKit, vous devez comprendre le mécanisme des delegate. Comme toujours, la meilleure manière de comprendre est de faire !

Le delegate, comprenez "gestion des événements", est un mécanisme particulier, qui se révèle très utile lorsque vous avez compris son fonctionnement. Il fait partie des nombreux design patterns (patrons de conception) comme MVC (Modèle Vue Contrôleur), KVO KVC (Key Value Observing, Key Value Coding), les catégories,...

Nous allons créer une classe GestionNotes, qui gérera les notes d'un élève. Cette classe va implémenter un protocole, qui contiendra des méthodes. Ce protocole sera appelé GestionNotesDelegate. Nous aurons donc une vue de saisie de notes (un UIViewController) qui sera delegate de GestionNotesDelegate.

Dans ce protocole, nous trouverons deux méthodes : une dite *requise* qui avertira le delegate lorsqu'une note aura bien été ajoutée dans le tableau et une dite *optionnelle* avertisant le delegate quand la moyenne des notes de l'élève descendra en dessous de 10/20.

DÉFINITION DU PROJET

Créons tout d'abord un nouveau projet de type Window-based Application que nous appellerons ApprentissageDelegate. Ajoutez-y un RootViewController comme à la Fiche 10 (laissez coché With XIB for User Interface). Sur la vue de ce contrôleur, ajoutez un champ de texte (UITextField), un label (UILabel) ainsi qu'un bouton (UIButton) comme à la Figure 11.1.



Figure 11.1 :
Construction du
RootViewController.

Code de l'application delegate :

```
ApprentissageDelegateAppDelegate.h
#import <UIKit/UIKit.h>
#import "RootViewController.h"

@interface ApprentissageDelegateAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    RootViewController *rootViewController;
}

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

ApprentissageDelegateAppDelegate.m
#import "ApprentissageDelegateAppDelegate.h"

@implementation ApprentissageDelegateAppDelegate
@synthesize window;
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    rootViewController = [[RootViewController alloc] init];
    [window addSubview:rootViewController.view];
    [window makeKeyAndVisible];
    return YES;
}
- (void)dealloc {
    [rootViewController release];
    [window release];
    [super dealloc];
}
@end
```

Et le code du RootViewController :

```
RootViewController.h
#import <UIKit/UIKit.h>
@interface RootViewController : UIViewController {
    UILabel *labelNote;
    UIButton *boutonValider;
    UITextField *textFieldNote;
}
@end
```

```
RootViewController.m
#import "RootViewController.h"

@implementation RootViewController
- (void)viewDidLoad {
    // Label affichant la moyenne
    labelNote = [[UILabel alloc] initWithFrame:CGRectMake(20, 30, 200, 30)];
    labelNote.text = @"pas de notes";
    [self.view addSubview:labelNote];

    // bouton pour permettre de valider l'entrée de la note
    boutonValider = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    boutonValider.frame = CGRectMake(20, 70, 80, 30);
    [boutonValider addTarget:self action:@selector(validerSaisieNote:)
        forControlEvents:UIControlEventTouchUpInside];
    [boutonValider setTitle:@"Ajouter" forState:UIControlStateNormal];
    [self.view addSubview:boutonValider];

    // textField permettant de saisir la note
    textFieldNote = [[UITextField alloc] initWithFrame:CGRectMake(20, 110, 200, 30)];
    textFieldNote.keyboardType = UIKeyboardTypeNumberPad;
    textFieldNote.borderStyle = UITextBorderStyleRoundedRect;
    textFieldNote.clearsOnBeginEditing = YES;

    [self.view addSubview:textFieldNote];

    [super viewDidLoad];
}

(void)validerSaisieNote:(id)sender {
    // on rentre le clavier
    [textFieldNote resignFirstResponder];
}

- (void)dealloc {
    [labelNote release];
    [boutonValider release];
    [textFieldNote release];
    [super dealloc];
}

@end
```

CRÉATION D'UNE CLASSE ET D'UN PROTOCOLE DÉLÉGUÉ

Créons la classe GestionNotes : ajoutez un nouveau fichier (Files > New file puis iPhone OS > Cocoa Touch Class > Objective-C classe > Subclass of NSObject).

Implémentez-la comme suit (pour l'instant, nous ne nous préoccupons que du protocole) :

```
GestionNotes.h
#import <Foundation/Foundation.h>

@protocol GestionNotesDelegate;

@interface GestionNotes : NSObject {
    id <GestionNotesDelegate> delegate; ①
}

@property (nonatomic, assign) id <GestionNotesDelegate> delegate; ②

@end

@protocol GestionNotesDelegate ③
@required ④
- (void) gestionNotes:(GestionNotes*)gestionNotes aReçuUneNote:(NSInteger)note
    ↪ nouvelleMoyenne:(float)moyenne; ⑤
@optional ⑥
- (void) gestionNotes:(GestionNotes*)gestionNotes enDessousMoyenne:(float)moyenne; ⑦
@end
```

```
GestionNotes.m
#import "GestionNotes.h"

@implementation GestionNotes
@synthesize delegate; ⑧

@end
```

Tout cela paraît bien compliqué... Pas d'inquiétude ! On retrouve, ligne ①, la déclaration de l'objet delegate (objet à qui GestionNotes va envoyer les événements).

Info

Id est un pointeur vers n'importe quel type d'objet. La puissance de l'Objective-C vient de sa dynamique, et id en est un exemple. Vous pourriez tout coder avec ce type !

On a le mécanisme des getters/setters cachés derrière @property et @synthesize lignes ② et ⑧. Nous spécifions assign, et non pas retain ou copy, pour éviter des références circulaires. Quant à

nonatomic, à la différence d'atomic, la ressource n'est pas bloquée (lock) par le thread appelant, ce qui lui éviterait d'être accédée par plusieurs threads simultanément.

À la ligne ③ commence la déclaration du protocole GestionNotesDelegate. On retrouve les mots-clés requis et optionnel ligne ④ et ⑥. Les méthodes qui sont sous le mot-clé @required devront être implémentées dans le delegate (ici ⑤), contrairement au mot-clé @optional : les méthodes peuvent être implémentées (ici ⑦).

Info

Si vous ne spécifiez rien, ce sera @required par défaut.

UTILISATION DE LA CLASSE ET DU PROTOCOLE

Dans la classe RootViewController, faites les changements suivants en partant du code écrit jusqu'ici :

```
RootViewController.h
#import <UIKit/UIKit.h>
#import "GestionNotes.h" ①

@interface RootViewController : UIViewController <GestionNotesDelegate> { ②
    UILabel *labelNote;
    UIButton *boutonValider;
    UITextField *textFieldNote;

    GestionNotes *gestionnaire; ③
}
@end

RootViewController.m
#import "RootViewController.h"

@implementation RootViewController
- (void)viewDidLoad {
    // construction de la vue comme au dessus ④
    // boutonValider, textFieldNote, et labelNote

    // gestionnaire des notes
    gestionnaire = [[GestionNotes alloc] initWithStudentName:@"Thierry"]; ⑤
    gestionnaire.delegate = self; ⑥

    [super viewDidLoad];
}

- (void)validerSaisieNote:(id)sender {
    NSInteger note = [textFieldNote.text integerValue];
```

```

if (note <= 20) { ⑦
    [gestionnaire ajouterNoteALaMoyenne:note];
    [textFieldNote resignFirstResponder];
}
else {
    textFieldNote.text = @""; ⑧
}
}

#pragma mark - ⑨
#pragma mark GestionNotes delegate methods
- (void)gestionNotes:(GestionNotes*)gestionNotes aReçuUneNote:(NSInteger)note
    ↪ nouvelleMoyenne:(float)moyenne{
    [labelNote setText:[NSString stringWithFormat:@"moyenne : %f", moyenne]]; ⑩
}

- (void)gestionNotes:(GestionNotes*)gestionNotes enDessousMoyenne:(float)moyenne {
    UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"Attention"
                                                       message:[NSString
    ↪ stringWithFormat:@"Votre moyenne de %f n'est pas bonne !", moyenne delegate:self
                                                       cancelButtonTitle:@"Ok"
                                                       otherButtonTitles:nil]; ⑪

    [alertView show];
    [alertView release];
}

- (void)dealloc {
    [labelNote release];
    [boutonValider release];
    [textFieldNote release];

    [gestionnaire release];

    [super dealloc];
}
@end

```

Comme nous avons créé une jolie classe GestionNotes, il faut bien l'utiliser ! On commence donc par un import du .h ligne ①.

À la ligne ②, on spécifie que la classe RootViewController implémente le protocole GestionNotes-Delegate.

On déclare donc un objet de la classe GestionNotes dans le .h ligne ③ que l'on initialisera ligne ⑤. Remarquez que la méthode initWithStudentName:(NSString*)name n'est pas encore implémentée dans GestionNotesDelegate. Nous allons y revenir juste après.

Ligne ⑥, nous écrivons que l'objet de la classe RootViewController, instancié par l'application delegate est delegate de l'objet gestionnaire.

N'oubliez pas qu'à la ligne ④ doit se trouver le code qui place les objets, comme défini plus haut !

Ligne ⑦, nous sommes dans la méthode appelée en cliquant sur le bouton Ajouter et nous testons si la note entrée est bien inférieure à 20 (pour ne pas dépasser 20/20). Si c'est le cas, on donne la note au gestionnaire via la méthode ajouterNoteALaMoyenne:(NSInteger)note qui la placera dans un tableau, puis on rétracte le clavier. À l'inverse, si la note est supérieure, on efface le champ ligne ⑧.

#pragma mark - , que vous retrouvez ligne ⑨ est un mot-clé très utile pour clarifier votre code lorsque vous souhaitez naviguer rapidement.

Info

#pragma mark - introduit une ligne

#pragma mark GestionNotes delegate methods introduit le texte GestionNotes delegate methods
Vous retrouverez le résultat Figure 11.2.

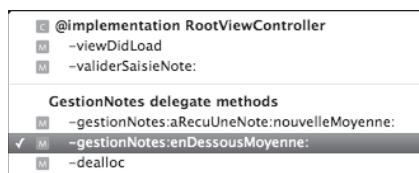


Figure 11.2 : Utilisation de #pragma mark.

La méthode dite *requise* - (void)gestionNotes:(GestionNotes*)gestionNotes aRecuUneNote:(NSInteger)note nouvelleMoyenne:(float)moyenne; doit être implémentée pour mettre le label à jour, dès qu'une note a bien été entrée (voir ligne ⑩). Enfin, si la moyenne descend en dessous de 10/20, on affiche une alert view (sorte de pop up) qui prévient l'utilisateur ligne ⑪.

Il ne vous aura pas échappé qu'il manque des choses dans la classe GestionNotes... Finissons de la construire ensemble !

Rajoutez ces lignes dans le .h :

```

NSString *studentName;
NSMutableArray *arrayDeNotes;

```

Ainsi que :

```
@property (nonatomic, copy) NSString *studentName;
```

et :

```
- (id) initWithStudentName:(NSString*)name;
- (void) ajouterNoteALaMoyenne:(NSInteger)note;
```

Passons au .m maintenant :

```
#import "GestionNotes.h"
```

```
@interface GestionNotes (PrivateMethods) ①
- (float) calculMoyenne; ②
@end
```

```

@implementation GestionNotes
@synthesize delegate;
@synthesize studentName;

- (id) initWithStudentName:(NSString*)name {
    if (self = [super init]) ③
    {
        self.studentName = name; ④
        arrayDeNotes = [[NSMutableArray alloc] init];
    }
    return self;
}

- (void) ajouterNoteALaMoyenne:(NSInteger)note {
    NSNumber *noteNumber = [NSNumber numberWithInteger:note]; ⑤
    [arrayDeNotes addObject:noteNumber];
    float moyenne = [self calculMoyenne]; ⑥

    if (moyenne < 10.0) {
        [delegate gestionNotes:self enDessousMoyenne:moyenne]; ⑦
    }
    if ([delegate respondsToSelector:@selector(
        ↪ gestionNotes:aRecuUneNote:nouvelleMoyenne:)]) { ⑧
        [delegate gestionNotes:self aRecuUneNote:note nouvelleMoyenne:moyenne]; ⑨
    }
}
- (void) dealloc {
    [arrayDeNotes release];
    [studentName release];
    [super dealloc];
}
@end
@implementation GestionNotes (PrivateMethods) ⑩
- (float) calculMoyenne {
    NSInteger cumulNotes = 0;
    for (NSNumber *number in arrayDeNotes) ⑪
    {
        NSInteger integerValue = [number integerValue];
        cumulNotes += integerValue; ⑫
    }
    return (float)cumulNotes/(float)[arrayDeNotes count]; ⑬
}
@end

```

Ouch ! Pas de panique, buvez frais et n'hésitez à refaire/relire cette fiche depuis le début !

Lorsque vous écrivez vos méthodes, il est toujours de bon ton de les rendre privées ou publiques. Nous ne souhaitons pas que la méthode calculMoyenne ligne ② soit accessible, d'où la syntaxe ligne ① que vous risquez de retrouver très souvent, avec d'autres mots-clés que PrivateMethods. C'est une des utilisations possibles du patron de conception nommé Catégories.

Nous avons créé notre propre méthode initWithStudentName: (NSString*)name pour vous montrer la syntaxe que vous devrez respecter pour toutes les méthodes init. Ligne ③, on appelle init de super, qui va retourner un objet alloué de type id. Comme on a self = [super init], self va faire référence à cet objet alloué (comprenez : "Va pointer sur la même case mémoire"). Enfin, il y a le test if. En fait, on pourrait expliciter en écrivant if((self = [super init]) != nil). Ce qui signifie donc que ce test sert à savoir si l'initialisation de super (ici super correspond à NSObject) s'est bien passée, et donc que self est bien alloué en mémoire.

Si tout va bien, on peut donc continuer notre initialisation en settant la propriété studentName ligne ④ et en allouant le tableau qui va nous servir à stocker les notes entrées.

Les NSArray ou NSMutableArray (mutable s'entend pour les objets qui peuvent être modifiés après initialisation) ne peuvent stocker que des objets. Pour ranger nos notes dans ce tableau, qui sont des NSInteger, il faut donc utiliser NSNumber comme à la ligne ⑤.

Info

Si vous souhaitez stocker d'autres types de données, jetez un œil du côté de NSValue. Par exemple, avec une structure que vous avez définie :

```
typedef struct {  
    NSString *nom;  
    NSString *prenom  
    NSInteger age;  
} Personne;  
  
Personne unePersonne;  
unePersonne.nom = @'"Dupont"';  
unePersonne.prenom = @'"Toto"';  
unePersonne.age = 27;
```

Pour encoder en NSValue :

```
NSValue *aValue = [NSValue value:&unePersonne withObjCType:@encode(Personne)];
```

Pour décoder :

```
Personne personneDecodee;  
[aValue getValue:&personneDecodee];
```

Ensuite, on calcule la nouvelle moyenne de l'étudiant ligne ⑥ pour vérifier qu'elle ne descend pas en dessous de 10/20, auquel cas on appelle la méthode gestionNotes:(GestionNotes*) enDessousMoyenne: (float)moyenne tout simplement ligne ⑦. Cela aura pour effet de déclencher l'apparition de l'alertView dans notre rootViewController !

Il faut également avertir le delegate que la note a bien été ajoutée au tableau. Notez que l'on pourrait tester si la note est bien présente dans le tableau, mais ce n'est pas très utile ici. Rappelez-vous, l'implémentation de cette méthode est optionnelle. Ce qui signifie que si le delegate ne l'implémente pas, votre application ne doit pas planter !

C'est pourquoi, ligne ⑧, nous testons si le delegate l'implémente avec `respondsToSelector:`. Si c'est le cas, on avertit le delegate ligne ⑨ que l'objet gestionnaire a bien reçu la note.

Nous avons souhaité rendre la méthode `calculMoyenne` privée. Il faut donc implémenter `GestionNotes (PrivateMethods)` ligne ⑩. Le calcul de la moyenne est très simple, et nous allons découvrir comment utiliser une boucle `For` pour faire une énumération ligne ⑪. En français dans le texte, cela donnerait : "Pour tous les objets de type `NSNumber` dans le tableau `arrayDeNotes`, faire quelque chose."

On calcule donc la moyenne de manière classique, en ajoutant les notes une par une (ligne ⑫) et en retournant ensuite la somme des notes divisée par le nombre total de notes, ligne ⑬.

Il ne vous reste plus qu'à compiler et lancer votre application pour observer son comportement, et bien comprendre ce que nous avons fait ensemble !

Ce mécanisme de delegate est très utile et doit être compris, au même titre que les `eventListener` en Java.

Vous voilà familier avec quelques éléments du UIKit, et roi du delegate ! Il existe des manières élégantes de présenter des listes de données à l'utilisateur, encore faut-il savoir les mettre en place.

Maintenant que le delegate n'a plus de secrets pour vous, nous allons voir comment construire par le code une table view (liste de données) et des picker view (l'élément rotatif lorsque vous choisissez l'heure de votre réveil par exemple).

Commençons par créer une liste (UITableView). Il n'y a rien de plus simple !

UTILISATION D'UNE LISTE DE DONNÉES

Créez un projet Window-based Application que vous nommerez TableView et ajoutez un fichier, de type UIViewController > UITableViewController comme à la Figure 12.1, que vous nommerez DataListViewController.



Figure 12.1 : Créez un table view controller.

Dans votre application delegate, ajoutez à la window la vue du DataListViewController. Petite piqûre de rappel au besoin :

```
TableViewAppDelegate.h
#import <UIKit/UIKit.h>
#import "DataListViewController.h"

@interface TableViewAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    DataListViewController *dataListViewController;
}
```

```

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

TableViewAppDelegate.m
#import "TableViewAppDelegate.h"

@implementation TableViewAppDelegate
@synthesize window;

- (BOOL)application:(UIApplication *)application
  → didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    dataListViewController = [[DataListViewController alloc]
    → initWithStyle:UITableViewStylePlain]; ①
    [window addSubview:dataListViewController.view];
    [window makeKeyAndVisible];
    return YES;
}

- (void)dealloc {
    [dataListViewController release];
    [window release];
    [super dealloc];
}
@end

```

Ligne ①, nous pouvons remarquer que l'on peut choisir le style de la liste de données. Il en existe 2 : *plain* qui affiche les données comme dans le carnet d'adresses, ou *grouped* qui affiche les données par section, comme dans les préférences de votre iPhone/iPod Touch.

Il faut maintenant afficher des données. Dans le .h de DataListViewController, déclarer un NSArray nommé *dataToShow*. Dans le *viewDidLoad* (.m), initialisez-le comme suit :

```

dataToShow = [[NSArray alloc] initWithObjects:@"Element 1", @"Element 2",
    → @"Element 3", @"Element 4", @"Element 5", nil];

```

Ensuite, modifiez ces quelques lignes de code :

```

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    // Return the number of sections.
    return 1; ①
}

- (NSInteger)tableView:(UITableView *)tableView
    → numberOfRowsInSection:(NSInteger)section {
    // Return the number of rows in the section.
    return [dataToShow count]; ②
}

```

Ligne ①, nous spécifions le nombre de sections, comme dans la liste de votre musique dans l'application iPod (rangée par ordre alphabétique, chaque lettre correspondant à une section). Ici, nous ne nous en servons pas, il n'y a donc qu'une section. À la ligne ②, c'est le nombre de lignes que vous définissez. Ici, nous retournons le nombre d'objets de notre tableau.

Enfin, il faut afficher ces données. Pour cela, modifiez la méthode suivante :

```
- (UITableViewCell *)tableView:(UITableView *)tableView  
↳ cellForRowAtIndexPath:(NSIndexPath *)indexPath {  
  
    static NSString *CellIdentifier = @"Cell";  
  
    UITableViewCell *cell = [tableView  
    ↳ dequeueReusableCellWithIdentifier:CellIdentifier]; ①  
    if (cell == nil) {②  
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault  
        ↳ reuseIdentifier:CellIdentifier] autorelease];  
    }  
  
    cell.textLabel.text = [dataToShow objectAtIndex:[indexPath row]]; ③  
  
    return cell;  
}
```

Cette méthode porte bien son nom. Elle demande à retourner la cellule à afficher pour chaque ligne. Pour que le défilement de votre table view soit le plus fluide possible, il faut faire très attention à la gestion mémoire de votre cellule. Par défaut, Apple introduit un mécanisme que vous retrouvez ligne ① où la cellule est mise en cache en fonction de CellIdentifier. Si cette cellule n'est pas allouée, on rentre dans la condition if ligne ② pour l'allouer. On peut choisir le style de la cellule, n'hésitez pas à parcourir la documentation pour les voir tous. Nous apprendrons à en créer une personnalisée à la section "Créer une cellule personnalisée" quelques pages plus loin. Enfin, nous allons afficher le bon élément pour la ligne demandée.

NSIndexPath comprend deux propriétés : section et row (ligne). La cellule de type UITableViewCell-StyleDefault comprend entre autres un label.textLabel. Nous nous en servons pour afficher ligne ③ l'élément correspondant.

Compilez et lancez. Vous devriez obtenir quelque chose de semblable à la Figure 12.2.



Figure 12.2 : Votre première liste de données !

CLIQUEZ SUR UNE CELLULE

Maintenant, nous allons rendre cette liste de données cliquable. Pour cela, il faut tout d'abord utiliser un contrôleur de navigation (`UINavigationController`) qui va permettre de gérer la navigation entre les différentes vues très simplement.

Modifiez votre application delegate pour obtenir ceci :

```
TableViewAppDelegate.h
#import <UIKit/UIKit.h>
#import "DataListViewController.h"

@interface TableViewAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    DataListViewController *dataListViewController;
    UINavigationController *navController;
}
@property (nonatomic, retain) IBOutlet UIWindow *window;
@end

TableViewAppDelegate.m
#import "TableViewAppDelegate.h"

@implementation TableViewAppDelegate
@synthesize window;
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
```

```
dataListViewController = [[DataListViewController alloc]
    ↪ initWithStyle:UITableViewStylePlain];
navController = [[UINavigationController alloc]
    ↪ initWithRootViewController:dataListViewController]; ①
[window addSubview:navController.view]; ②
>window makeKeyAndVisible];

return YES;
}

- (void)dealloc {
    [navController release];
    [dataListViewController release];
    [window release];
    [super dealloc];
}
@end
```

① On alloue le contrôleur de navigation, avec comme rootViewController dataListViewController, puis on affiche la vue du navController ainsi alloué à la place de la vue de dataListViewController ligne ②.

Ensuite, il faut créer un nouveau contrôleur (UIViewController) que vous nommerez DetailListViewController (cochez With XIB for User Interface). Dans ce contrôleur, affichez un label, et déclarez également un NSString en @property.

```
DetailListViewController.h
#import <UIKit/UIKit.h>
@interface DetailListViewController : UIViewController {
    UILabel *label;
    NSString *texteAAfficher;
}
@property (nonatomic, copy) NSString *texteAAfficher;
@end

DetailListViewController.m
#import "DetailListViewController.h"

@implementation DetailListViewController
@synthesize texteAAfficher;

- (void)viewDidLoad {
    label = [[UILabel alloc] initWithFrame:CGRectMake(0, 0, 200, 30)];
    [self.view addSubview:label];
    label.text = texteAAfficher;
    [super viewDidLoad];
}
```

```

- (void)dealloc {
    [label release];
    [texteAAfficher release];
    [super dealloc];
}
@end

```

Revenez à DataListViewController.m et rendez-vous à la méthode - (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath que vous implémenterez comme suit :

```

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)
    →*indexPath {
    DetailListViewController *detailListViewController = [[DetailListViewController
        → alloc] initWithNibName:@"DetailListViewController" bundle:nil];

    detailListViewController.texteAAfficher =
    → [dataToShow objectAtIndex:[indexPath row]];
    // possibilité d'afficher un titre dans la barre de navigation
    detailListViewController.title = [dataToShow objectAtIndex:[indexPath row]];

    [self.navigationController pushViewController:detailListViewController
        → animated:YES];
    [detailListViewController release];
}

```

Maintenant, en cliquant sur une cellule, vous affichez une nouvelle vue. Libre à vous de la remplir comme vous le souhaitez !

Attention

N'oubliez pas le #import "DetailListViewController.h"

CRÉER UNE CELLULE PERSONNALISÉE

Pour finir avec les table view, nous allons voir comment créer une cellule personnalisée.

Ajoutez un nouveau fichier, iPhone > Cocoa Touch Classes > Objective-C class > subclass of UITableViewCell que vous nommerez CustomCell.

Ensuite, modifiez ces nouveaux fichiers pour y ajouter un label :

```

CustomCell.h

#import <UIKit/UIKit.h>
@interface CustomCell : UITableViewCell {
    UILabel *myLabel;
}
@property (nonatomic, retain) UILabel *myLabel;
@end

```

CustomCell.m

```
#import "CustomCell.h"
@implementation CustomCell
@synthesize myLabel;

- (id)initWithStyle:(UITableViewCellStyle)style reuseIdentifier:(NSString *)
{
    if ((self = [super initWithStyle:style reuseIdentifier:reuseIdentifier]) {
        myLabel = [[UILabel alloc] initWithFrame:CGRectMake(150, 5, 120, 30)];
        myLabel.font = [UIFont fontWithName:@"zapfino" size:12.0];
        [self addSubview:myLabel];
    }
    return self;
}
- (void)setSelected:(BOOL)selected animated:(BOOL)animated {
    [super setSelected:selected animated:animated];
}
- (void)dealloc {
    [myLabel release];
    [super dealloc];
}
@end
```

Enfin, dans DataListViewController.m, ajouter l'import de CustomCell.h, puis modifiez la méthode `cellForRowAtIndexPath` comme suit :

```
- (UITableViewCell *)tableView:(UITableView *)tableView
{
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    static NSString *CellIdentifier = @"CustomCell";

    CustomCell *cell = (CustomCell*)[tableView
    dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[CustomCell alloc] initWithStyle:UITableViewCellStyleDefault
        reuseIdentifier:CellIdentifier] autorelease];
    }
    cell.myLabel.text = [dataToShow objectAtIndex:[indexPath row]];
    return cell;
}
```

Compilez et lancez, vous devriez obtenir une liste de données comme à la Figure 12.3.



Figure 12.3 : Une liste de cellules personnalisées.

Une bonne chose de faite non ? Un petit café, ou alors continuez sur votre lancée !

UTILISER UN PICKER VIEW OU LISTE CIRCULAIRE

Créez un nouveau projet Window-based Application, que vous nommerez Picker, pour utiliser les picker view.

Créez ensuite une nouvelle classe de type UIViewController que vous nommerez PickerViewController. Comme d'habitude, ajoutez la vue d'un objet UIPickerView (que vous aurez alloué dans votre application delegate) à la window, .

Ensuite, dans le viewDidLoad de PickerViewController, ajoutez ceci :

```
- (void)viewDidLoad {  
    monPickerView = [[UIPickerView alloc] initWithFrame:CGRectMakeZero]; ①  
    CGSize pickerSize = [monPickerView sizeThatFits:CGSizeZero];  
  
    CGRect screenRect = [[UIScreen mainScreen] applicationFrame];  
    CGRect pickerRect = CGRectMake( 0.0,  
        screenRect.size.height - 84.0 - pickerSize.height,  
        pickerSize.width,  
        pickerSize.height);  
  
    monPickerView.frame = pickerRect; ②  
  
    monPickerView.autoresizingMask = UIViewAutoresizingFlexibleWidth;
```

```

monPickerView.showsSelectionIndicator = YES; ③

monPickerView.delegate = self; ④
monPickerView.dataSource = self; ⑤

[self.view addSubview:monPickerView];

label = [[UILabel alloc] initWithFrame:CGRectMake(10, 30, 200, 20)]; ⑥
[self.view addSubview:label];

pickerViewArray = [[NSArray alloc] initWithObjects:@"Element 1", @"Element 2",
➥ @"Element 3", @"Element 4", @"Element 5", @"Element 6", @"Element 7", nil]; ⑦

[super viewDidLoad];
}

```

La taille d'un picker view est optimisée. Comprenez que vous ne pouvez pas vraiment le dimensionner comme vous le souhaitez. C'est pourquoi de la ligne ① à la ligne ②, vous trouverez un mécanisme qui optimise sa taille, et va le centrer sur votre vue.

Vous pouvez choisir d'afficher une sorte de vitre, vous montrant la ligne sélectionnée. Par défaut, rien ne s'affiche. Ligne ③, nous choisissons donc de montrer à l'utilisateur la sélection actuelle.

Le picker view nécessite d'être rempli par des données, c'est pourquoi nous définissons le view controller actuel comme source de données ligne ⑤, données qui seront dans un tableau, initialisé ligne ⑦. La classe UIPickerView contient un protocole UIPickerViewDelegate. À la ligne ④, on définit donc self comme étant delegate du picker view.

Pour montrer le bon fonctionnement du picker, on ajoute un label à la vue ligne ⑥, qui affichera les éléments sélectionnés.

Comme nous venons de définir la classe PickerViewController comme étant à la fois source de données et delegate de UIPickerView, vous devez écrire ceci dans le .h :

```
@interface PickerViewController : UIViewController <UIPickerViewDelegate,
➥ UIPickerDataSource> {
```

Ensuite, rappelez-vous, un protocole définit des méthodes optionnelles et d'autres requises. Vous les trouverez dans la documentation.

Nous souhaitons afficher deux colonnes dans le picker, une contenant Element 1, Element 2, etc. et une autre contenant le numéro de la ligne.

Ajoutez donc ces méthodes dans PickerViewController.m :

```
#pragma mark -
#pragma mark UIPickerViewDelegate

- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
➥ inComponent:(NSInteger)component
{
    label.text = [NSString stringWithFormat:@"%@ - %d",
                 pickerViewArray objectAtIndex:[pickerView selectedRowInComponent:0]],
                 [monPickerView selectedRowInComponent:1]]; ①
}
```

```

}

#pragma mark -
#pragma mark UIPickerViewDataSource

- (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row
    ↪ forComponent:(NSInteger)component
{
    NSString *returnStr = @"";
    if (component == 0)
    {
        returnStr = [pickerViewArray objectAtIndex:row]; ②
    }
    else
    {
        returnStr = [[NSNumber numberWithInt:row] stringValue]; ③
    }
    return returnStr;
}

- (CGFloat)pickerView:(UIPickerView *)pickerView widthForComponent:(NSInteger) component
{
    CGFloat componentWidth = 0.0;
    if (component == 0)
        componentWidth = 240.0; ④
    else
        componentWidth = 40.0; ⑤

    return componentWidth;
}

- (CGFloat)pickerView:(UIPickerView *)pickerView rowHeightForComponent:(NSInteger)component
{
    return 40.0; ⑥
}

- (NSInteger)pickerView:(UIPickerView *)pickerView
    ↪ numberOfRowsInComponent:(NSInteger)component
{
    return [pickerViewArray count]; ⑦
}

- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView
{
    return 2; ⑧
}

```

Nous choisissons d'afficher deux colonnes dans le picker, grâce à la ligne ⑧.

Lignes ④ et ⑤, on doit définir la taille pour chaque colonne (nommée ici component, celle numérotée 0 étant à gauche).

Ligne ⑥, on définit la hauteur des lignes, et ligne ⑦, on détermine leur nombre qui est défini par le nombre d'éléments du tableau pickerViewArray.

Pour remplir chaque ligne du picker, on implémente la méthode - (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row forComponent:(NSInteger)component. Pour la colonne de gauche (0), le titre de la ligne sera l'élément correspondant dans le tableau, ligne ②. Pour celle de droite, le titre de la ligne est son numéro, ligne ③.

Enfin, ligne ①, on affiche dans le label les éléments sélectionnés dans chaque colonne.

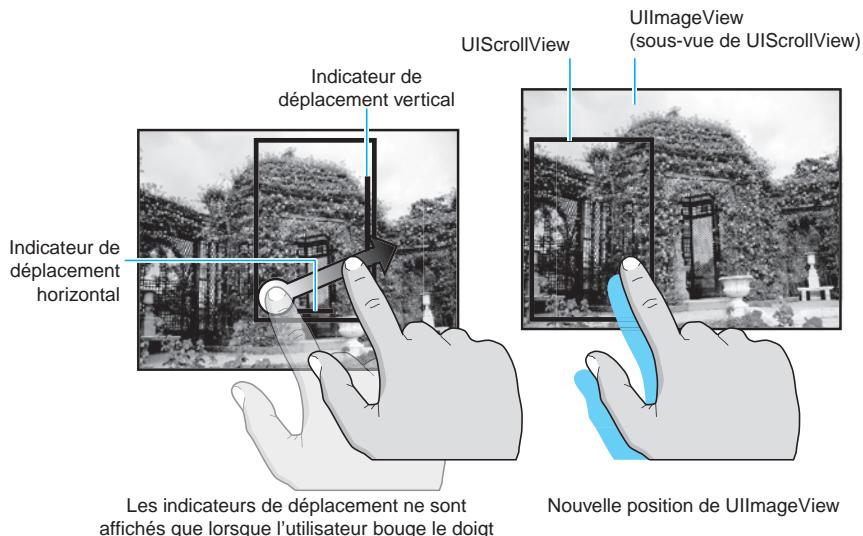
Info

Si vous souhaitez que l'utilisateur choisisse une date, il existe un élément pour ça : le UIDatePicker

Nous venons de voir comment présenter de manière efficace des listes de données à l'utilisateur. Il peut vous arriver d'avoir à afficher un document (image, pdf,...) plus grand que l'écran. Pour cela, il faut utiliser une scroll view.

INTRODUCTION

Pour bien comprendre ce qu'est une scroll view, jetez tout d'abord un œil à la Figure 13.1.



Vous remarquez que la taille d'une scroll view est plus grande que celle de l'écran. Grâce aux doigts, l'utilisateur peut la bouger, zoomer, pour afficher la partie qui l'intéresse sans aucun effort de votre part, ou presque...

Comme vous pouvez le voir à la Figure 13.2, la taille d'une scroll view est définie par les valeurs `contentSize.width` (largeur) et `contentSize.height` (hauteur). Vous pouvez de plus définir des marges avec `contentInset.top` (haut), `.bottom` (bas), `.left` (gauche) et `.right` (droite).

Nous allons donc construire une grande scroll view contenant des boutons !

Pour cela, chaque bouton contiendra le numéro de la colonne et de la ligne sur lequel il est placé. Vous obtiendrez ainsi la même chose que la Figure 13.3.

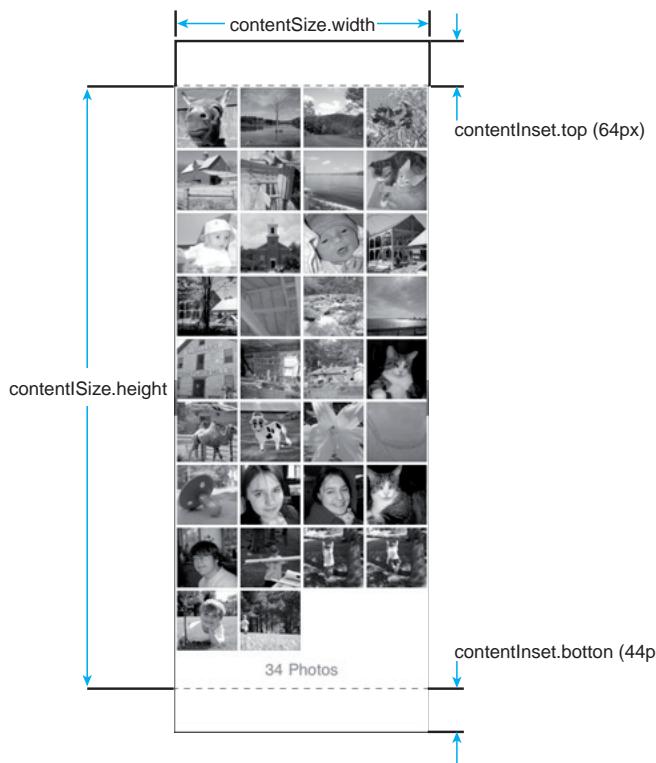


Figure I3.2 : Définition de la taille d'une scroll view.

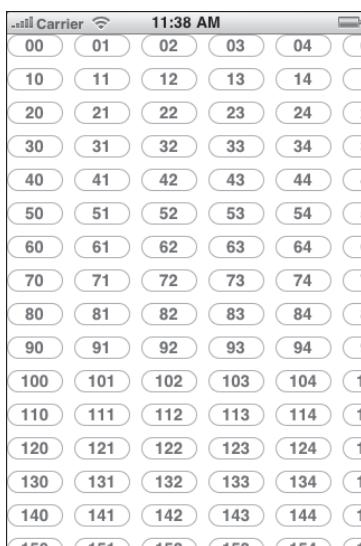


Figure I3.3 : Un nuage de boutons.

CRÉATION DU PROJET

Créez un projet Window-based Application, que vous nommerez ScrollView, puis ajoutez-y un RootViewController comme d'habitude, que vous affichez par-dessus la window.

Ensuite, modifiez le RootViewController comme suit :

```
RootViewController.h
#import <UIKit/UIKit.h>
@interface RootViewController : UIViewController <UIScrollViewDelegate> { ①
}
@end

RootViewController.m
#import "RootViewController.h"
#define NSLogFunction NSLog(@"%@", __FUNCTION__) ②

#define HAUTEUR_SCROLLVIEW 1000 ③
#define LARGEUR_SCROLLVIEW 500
#define MARGE_PLACEMENT_BOUTONS 10
#define HAUTEUR_BOUTONS 20
#define LARGEUR_BOUTONS 50

@implementation RootViewController

- (void)viewDidLoad {
    // définition de la scrollview
    UIScrollView *scrollView = [[UIScrollView alloc] initWithFrame:[[UIScreen
        mainScreen] applicationFrame]]; ④
    scrollView.contentSize = CGSizeMake(LARGEUR_SCROLLVIEW, HAUTEUR_SCROLLVIEW); ⑤
    scrollView.delegate = self; ⑥
    self.view = scrollView; ⑦
    [scrollView release]; ⑧

    // placement des boutons
    // calcul du nombre de boutons à mettre en largeur
    int nbBoutonsLargeur = (int)(LARGEUR_SCROLLVIEW/(LARGEUR_BOUTONS +
        MARGE_PLACEMENT_BOUTONS));
    // calcul du nombre de boutons à mettre en hauteur
    int nbBoutonsHauteur = (int)(HAUTEUR_SCROLLVIEW/(HAUTEUR_BOUTONS +
        MARGE_PLACEMENT_BOUTONS));

    for (int ligne = 0 ; ligne < nbBoutonsHauteur ; ligne++) {
        for (int colonne = 0 ; colonne < nbBoutonsLargeur ; colonne++) {
            UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
            [button setFrame:CGRectMake(colonne*(LARGEUR_BOUTONS +
                MARGE_PLACEMENT_BOUTONS), ligne * (HAUTEUR_BOUTONS +
                MARGE_PLACEMENT_BOUTONS), LARGEUR_BOUTONS, HAUTEUR_BOUTONS)]; ⑨
        }
    }
}
```

```
[button setTitle:[NSString stringWithFormat:@"%d%d", ligne, colonne]
    ↪ forState:UIControlStateNormal]; ⑩
[button addTarget:self action:@selector(handleClic:)
    ↪ forControlEvents:UIControlEventTouchUpInside]; ⑪
button.tag = [button.titleLabel.text intValue]; ⑫
[self.view addSubview:button];
}
[super viewDidLoad];
}

- (void) handleClic:(id)sender {
    UIButton *theSenderButton = (UIButton*)sender;
    NSLog(@"Sender tag : %d", theSenderButton.tag); ⑬
}

#pragma mark -
#pragma mark UIScrollView delegate methods

- (void)scrollViewDidScroll:(UIScrollView *)scrollView { ⑭
    NSLogFunction;
}
- (void)scrollViewWillBeginDragging:(UIScrollView *)scrollView { ⑮
    NSLogFunction;
    for (UIButton *but in [scrollView subviews])
        but.highlighted = YES; ⑯
}
- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView willDecelerate:(BOOL)£
    decelerate { ⑰
        NSLogFunction;
        for (UIButton *but in [scrollView subviews])
            but.highlighted = NO;
    }
- (void)dealloc {
    [super dealloc];
}
@end
```

La classe UIScrollView fournit un protocole delegate : UIScrollViewDelegate. Nous choisissons donc de l'implémenter dans la classe RootViewController, d'où les lignes ① et ⑥.

Ligne ④, nous allouons la scroll view, avec comme dimension la taille de l'écran (minorée de la taille de la status bar si elle est présente).

À la ligne ⑤, on définit la taille du contenu de la scroll view. Notez que nous avons défini ces différentes valeurs à la ligne ③ et suivantes. Cela dans le but de rendre le code modifiable très facilement si vous souhaitez changer en cours de développement une de ces valeurs.

Nous avons par ailleurs défini ligne ② une petite macro, qui va afficher dans la console [ClasseAppelante nomDeLaMethode], lorsque vous écrirez dans votre code NSLogFunction;, [ClasseAppelante nomDeLaMethode];.

Ligne ⑦, nous remplaçons la vue par défaut du RootViewController (`self.view`) par `scrollView`. Notez que l'on pourrait l'ajouter à `self.view`, mais gardez à l'esprit qu'il faut toujours minimiser le nombre de vues empilées. Ensuite, on release `scrollView` ligne ⑧, car `self` la détient (avec un retain).

Pour le besoin de ce tutoriel, nous allons maintenant ajouter des boutons à notre scroll view.

Ligne ⑨, grâce à une double boucle For, et un peu de mathématiques, les boutons seront placés sur la scroll view.

Ligne ⑩, on définit le titre comme étant le numéro de la colonne suivi du numéro de la ligne.

Chaque bouton aura la même méthode qui gérera le clic, ligne ⑪ et on utilisera le tag défini ligne ⑫ pour identifier chaque bouton précisément. Ce tag est en fait le couple colonne/ligne. Dans la méthode qui répond à l'événement `touchUpInside` de chaque bouton, `handleClic:(id)sender`, on affiche dans la console le numéro du tag ligne ⑬. Ainsi, on peut savoir directement quel est le bouton qui a envoyé cet événement.

Regardons maintenant quelques méthodes du delegate.

Ligne ⑭, cette méthode sera appelée périodiquement, tant que l'utilisateur bougera la scroll view.

Ligne ⑮, cette méthode sera appelée dès lors que l'utilisateur, après avoir posé son doigt sur la scroll view, bougera. Pour vous montrer précisément à quel moment cette méthode est appelée, on surligne tous les boutons ligne ⑯, et on les rend non surlignés dans la méthode ligne ⑰ appelée lorsque l'utilisateur relâche son doigt de la scroll view.

Pour gérer le zoom, ajoutez ces quelques lignes : dans le `viewDidLoad`,

```
scrollView.maximumZoomScale = 3.0;  
scrollView.minimumZoomScale = 1.0;
```

puis avant // placement des boutons

```
imageView = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"votreImage.png"]];  
[self.view addSubview:imageView];
```

Enfin, ajoutez cette méthode delegate à la suite des autres :

```
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)scrollView  
{  
    return imageView;  
}
```

Info

Pour zoomer avec le simulateur, appuyez sur la touche Alt puis cliquez !

CHAPITRE 4

UTILISER LES FONCTIONNALITÉS DE L'PHONE

Fervent utilisateur de votre iPhone, il ne vous aura pas échappé qu'il est rempli de capteurs (GPS, accéléromètre, boussole...). Dans ce chapitre, vous allez apprendre à les exploiter, et tirer parti des fonctionnalités comme l'accès à la librairie iTunes, l'utilisation du Bluetooth, le push...

Une des principales fonctionnalités des smartphones est d'intégrer une puce GPS permettant de localiser l'utilisateur. L'iPhone n'y échappe pas ! Voyons ensemble comment s'en servir.

Attention

Vous pourrez compiler sur simulateur pour utiliser le GPS. Toutefois, ce ne sera pas très simple pour tester un déplacement par exemple...

De plus, n'oubliez pas que l'iPod Touch n'intègre pas de puce GPS !

Pour commencer, nous allons faire très simple : utiliser directement l'API intégrée au SDK et fournie par Google. Ainsi, nous allons pouvoir afficher la position de l'utilisateur sur une carte sans trop d'efforts...

Cette carte, un objet de la classe MKMapView, est en réalité plus qu'une simple carte, car elle affiche la position de l'utilisateur, permet d'ajouter des annotations (ex.: les endroits favoris de l'utilisateur), ou une vue superposée (overlay view) pour surligner une route par exemple. Elle possède également un objet de la classe MKUserLocation, si vous souhaitez récupérer des données comme l'altitude, la vitesse, la position en coordonnées GPS, la précision de la localisation...

Comme toujours, créez un nouveau projet que vous appellerez GPS, et ajoutez-y un RootViewController que vous afficherez par-dessus la window. Vous allez finir par le faire les yeux fermés...

AFFICHER LA POSITION DE L'UTILISATEUR SUR UNE CARTE

Une fois tout ceci en place, ajoutez le framework MapKit, et écrivez ce code :

```
RootViewController.h
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h> ①

@interface RootViewController : UIViewController <MKReverseGeocoderDelegate,
MKMapViewDelegate> {
    MKMapView *maMapView; ②
    UILabel *labelReverseGeoCoder;
    UIButton *buttonLaunchReverseGeocode;
    UIActivityIndicatorView *activityReverseGeoCod;
    MKReverseGeocoder *reverseGeocoder; ③
}
@end

RootViewController.m
#import "RootViewController.h"

@implementation RootViewController
```

```
- (void)viewDidLoad {
    // initialisation de la carte
    maMapView = [[MKMapView alloc] initWithFrame:CGRectMake(0, 0, 320, 360)];
    // affichage de la position de l'utilisateur
    maMapView.showsUserLocation = YES; ④
    // on change le type d'affichage pour mettre carte + satellite
    maMapView.mapType = MKMapTypeHybrid; ⑤
    // self recevra les événements de maMapView
    maMapView.delegate = self;

    [self.view addSubview:maMapView];

    // déclaration du label affichant l'adresse actuelle de l'utilisateur
    labelReverseGeoCoder = [[UILabel alloc] initWithFrame:CGRectMake(20, 380, 280, 40)];
    [self.view addSubview:labelReverseGeoCoder];

    // déclaration du bouton qui va servir à lancer le reverse geo coder
    buttonLaunchReverseGeocode = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    buttonLaunchReverseGeocode.frame = CGRectMake(20, 420, 150, 30);
    [buttonLaunchReverseGeocode setTitle:@"Donne mon adresse"
    ↪ forState:UIControlStateNormal];
    [buttonLaunchReverseGeocode addTarget:self
    ↪ action:@selector(reverseGeocodeCurrentLocation:)
    ↪ forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:buttonLaunchReverseGeocode];

    // déclaration de la roue qui indique une activité
    activityReverseGeoCod = [[UIActivityIndicatorView alloc]
    ↪ initWithActivityIndicatorStyle:UIActivityIndicatorViewStyleGray];
    activityReverseGeoCod.hidesWhenStopped = YES;
    activityReverseGeoCod.center = CGPointMake(260, 430);
    [self.view addSubview:activityReverseGeoCod];

    [super viewDidLoad];
}

#pragma mark -
#pragma mark button methods
- (void)reverseGeocodeCurrentLocation : (id) sender ⑥
{
    // test de la précision
    if(maMapView.userLocation.location.horizontalAccuracy > 100.0)
    // distance en mètres ⑦
    {
        //précision insuffisante, on affiche une alert view
```

```

UIalertView *monAlert = [[UIAlertView alloc] initWithTitle:@"Erreur"
message:[NSString stringWithFormat:@"La précision de %.2f m est
➥ insuffisante", maMapView.userLocation.location.horizontalAccuracy]
➥ delegate:self
cancelButtonTitle:@"Ok"
otherButtonTitles:nil];
[monAlert show];
[monAlert release];
}

else
{
    // la précision est suffisante, on demande à faire un reverse
    [activityReverseGeoCod startAnimating];
    // allocation du reverse geocoder avec les coordonnées actuelles de
    // l'utilisateur
    reverseGeocoder = [[MKReverseGeocoder alloc]
    initWithCoordinate:maMapView.userLocation.location.coordinate]; ⑧
    reverseGeocoder.delegate = self;
    [reverseGeocoder start];
}

}

#pragma mark -
#pragma mark MKMapView delegate
- (void)mapView:(MKMapView *)mapView didUpdateUserLocation:(MKUserLocation *)
➥ userLocation
{
    // on zoome
    MKCoordinateSpan span = {0.01, 0.01}; ⑨
    [mapView setRegion:MKCoordinateRegionMake(mapView.userLocation.location
➥ .coordinate, span) animated:YES]; ⑩
}

#pragma mark -
#pragma mark Reverse geocoder delegate
- (void)reverseGeocoder:(MKReverseGeocoder *)geocoder didFailWithError:(NSError *)error
{
    NSLog(@"%@", [error localizedDescription]); ⑪
    [activityReverseGeoCod stopAnimating];
    // release si erreur
    [geocoder release]; ⑫
}

- (void)reverseGeocoder:(MKReverseGeocoder *)geocoder didFindPlacemark:(MKPlacemark
➥ *)placemark
{
}

```

```
labelReverseGeoCoder.text = [NSString stringWithFormat:@"%@ %@ %@ %@",  
    ↪ placemark.country, placemark.postalCode, placemark.locality,  
    ↪ placemark.thoroughfare]; ⑬  
[labelReverseGeoCoder sizeToFit];  
[activityReverseGeoCod stopAnimating];  
// on a fini d'utiliser le reverseGeocoder, que l'on alloue à chaque clic sur le  
↪ bouton, donc on le release ici  
[geocoder release]; ⑭  
}  
(void)dealloc {  
    if(reverseGeocoder)  
        [reverseGeocoder release];  
    [maMapView release];  
    [activityReverseGeoCod release];  
    [labelReverseGeoCoder release];  
    [buttonLaunchReverseGeocode release];  
    [super dealloc];  
}  
@end
```

Comme nous avons ajouté un framework à notre application, le MapKit, il faut faire l'import de la ligne ① !

Ligne ②, un objet de type MKMapView est déclaré, il représente la carte fournie par Google.

Ligne ③, on déclare un objet MKReverseGeocoder qui va nous permettre de retrouver l'adresse (pays, ville, code postal, numéro de rue, nom de la rue) à partir des coordonnées GPS de l'utilisateur !

Passons maintenant à l'implémentation : comme indiqué ligne ④, nous pouvons demander à l'instance de MKMapView de montrer la position de l'utilisateur. Cela signifie que sa position s'affichera à l'aide du point bleu, comme dans l'application Plans de l'iPhone, et que l'affichage de cette position sera automatiquement mis à jour lors du déplacement.

Info

MKMapView gère donc son propre Location Manager (l'objet qui se charge de dialoguer avec le GPS). Ainsi, on peut accéder à l'objet de type CLLocation que vous retrouvez avec maMapView.userLocation.location. Nous allons apprendre à nous en servir plus loin dans cette fiche.

Ligne ⑤, on peut également choisir le type d'affichage de la carte : cartographie, satellite, ou hybride (les deux en même temps). Notez qu'en sélectionnant satellite seulement, vous risquez de ne pas avoir d'images si vous zoomez trop.

On choisit d'appeler une méthode qui va nous permettre de lancer le reverse geocoding avec un bouton. Cette méthode est implémentée ligne ⑥, mais pour obtenir un résultat correct, il faut un minimum de précision sur la mesure. C'est pourquoi nous choisissons de tester la précision horizontale actuelle (horizontalAccuracy pour la précision sur la mesure des coordonnées GPS, et donc la position sur la terre, verticalAccuracy pour la précision sur la mesure de l'altitude). Si cette précision est inférieure à 100 mètres, on choisit d'avertir l'utilisateur et de ne pas lancer le reverse geocoding ligne ⑦.

Ligne ⑧, lorsque la précision est inférieure à 100 mètres, on peut faire appel au reverse geocoder en lui fournissant les coordonnées GPS actuelles de l'utilisateur. On alloue ici l'objet reverseGeocoder, que l'on libère ensuite lorsqu'il a fait son travail ligne ⑫ ou ⑯. Pour éviter tout problème, on libère l'objet geocoder passé en paramètre des méthodes delegate, qui est une référence vers reverseGeocoder.

Info

Vous pouvez donc retrouver n'importe quelle adresse du moment que vous avez les coordonnées GPS ! Il suffit de créer une variable de type CLLocationCoordinate2D, et lui spécifier la latitude ainsi que la longitude.

Passons à la méthode delegate de MKMapView. Ligne ⑩, on choisit de zoomer sur la carte avec animation. La valeur de ce zoom est définie à la ligne ⑨ avec la variable span de type MKCoordinateSpan. Dans la documentation :

```
typedef struct {  
    CLLocationDegrees latitudeDelta;  
    CLLocationDegrees longitudeDelta;  
} MKCoordinateSpan;
```

Ainsi, on choisit le delta en latitude et le delta en longitude à afficher sur la carte. Ces valeurs sont exprimées en degrés.

Rappel

43.4035° correspond à 43°24'12" ($0,4035 * 60 = 24,21 \Rightarrow 24'$; $0,21 * 60 = 12,6 \Rightarrow 12''$)

23°45'04" correspond à 23,7511° ($23 + 45/60 + 4/3600$)

Comme toute application proprement implémentée, les méthodes de gestion des erreurs et exceptions sont primordiales. Et parce qu'il est impossible de prédire le comportement de l'application sur tous les iPhones en circulation, n'oubliez pas d'implémenter (ou créer) aussi souvent que possible de telles méthodes. Ligne ⑪, on se sert d'une méthode très pratique de NSError : localizedDescription pour afficher la description de l'erreur plutôt qu'un vulgaire numéro d'erreur. N'hésitez pas à essayer en coupant votre connexion Internet par exemple !

Enfin, ligne ⑬, on affiche dans le label les différentes informations rendues par le reverse geocoder. Quelques explications : imaginez que vous soyez devant le 4, avenue du Prado, 13006 Marseille. Voici les informations correspondantes :

- administrativeArea : Provence-Alpes-Côte d'Azur
- country : France
- countryCode : FR
- locality : Marseille
- postalCode : 13006
- subAdministrativeArea : Bouches-du-Rhône
- subThoroughfare : 4
- thoroughfare : Avenue du Prado

EXERCICE

Pour vous entraîner, mettez deux boutons sur l'alerte relative à la précision. Cela permettra à l'utilisateur de lancer quand même le reverse geocoding s'il le souhaite.

SOLUTION

Modifier la méthode appelée lorsque vous cliquez sur le bouton :

```
- (void)reverseGeocodeCurrentLocation : (id) sender
{
    // test de la précision
    if(maMapView.userLocation.location.horizontalAccuracy > 100.0) // distance en mètres
    {
        //précision insuffisante, on affiche une alert view
        UIAlertView *monAlert = [[UIAlertView alloc] initWithTitle:@"Erreur"
            message:[NSString stringWithFormat:@"La précision de %.2f m est insuffisante",
            ↪ maMapView.userLocation.location.horizontalAccuracy]
            delegate:self
            cancelButtonTitle:@"Annuler"
            otherButtonTitles:@"Continuer", nil];
        [monAlert show];
        [monAlert release];
    }
}
```

et ajouter cette méthode :

```
- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex
{
    if(buttonIndex == 1)
    {
        // l'utilisateur a cliqué sur continuer
        [activityReverseGeoCod startAnimating];
        // allocation du reverse geocoder avec les coordonnées actuelles de l'utilisateur
        reverseGeocoder = [[MKReverseGeocoder alloc]
            ↪ initWithCoordinate:maMapView.userLocation.location.coordinate];
        reverseGeocoder.delegate = self;
        [reverseGeocoder start];
    }
}
```

AFFICHER DES ANNOTATIONS SUR LA CARTE

Sur la carte, il est possible d'afficher des annotations, qui sont par défaut des petites punaises placées sur la carte. En fait, vous pouvez changer pour mettre votre propre ressource graphique.

L'ajout des annotations personnalisées se fait en créant une classe qui respecte le protocole MKAnnotation. Les façons d'opérer sont nombreuses, nous suivons celle-ci : on peut créer une classe conforme au protocole MKAnnotationView pour dessiner ce que l'on souhaite, mais il faut de toute manière implémenter - (MKAnnotationView *)mapView:(MKMapView *)theMapView viewForAnnotation:(id <MKAnnotation>)annotation.

Commençons par les annotations : créez un fichier Objective-C class subclass of NSObject que vous nommerez MonAnnotation. Implémentez-le comme ceci :

```
MonAnnotation.h
#import <Foundation/Foundation.h>
#import <MapKit/MapKit.h>

@interface MonAnnotation :NSObject <MKAnnotation> { ①
    // pour être conforme au protocole MKAnnotation
    NSString *title;
    NSString *subtitle;
    // requis par le protocole
    CLLocationCoordinate2D coordinate; ②

    // pour ne pas importer le framework CoreLocation
    NSNumber *latitude;
    NSNumber *longitude;
    ③
}

@property (nonatomic, retain) NSString *title;
@property (nonatomic, retain) NSString *subtitle;
@property (nonatomic, retain) NSNumber *latitude;
@property (nonatomic, retain) NSNumber *longitude;
@property (nonatomic, readonly) CLLocationCoordinate2D coordinate;
@end

MonAnnotation.m
#import "MonAnnotation.h"
@implementation MonAnnotation
@synthesize title, subtitle, coordinate, latitude, longitude;
- (CLLocationCoordinate2D)coordinate ④
{
    coordinate.latitude = [self.latitude doubleValue];
    coordinate.longitude = [self.longitude doubleValue];
    return coordinate;
}
@end
```

Lorsque l'on crée cette classe, qui doit respecter le protocole MKAnnotation spécifié à la ligne ①, on doit implémenter certaines propriétés comme la ligne ②. Rien ne vous empêche d'ajouter d'autres variables à partir de la ligne ③, comme une propriété de description.

Enfin, pour ne pas avoir à inclure le framework CoreLocation, on définit une méthode ligne ④ qui construira une variable de type CLLocationCoordinate2D à partir de deux objets NSNumber : latitude et longitude. Ceci pour éviter de solliciter CLLocationCoordinate2DMake nécessitant ledit framework.

Ensuite, ajoutez cette méthode dans le .m de RootViewController, vers les méthodes delegate de MKMapView pour personnaliser notre punaise sur la carte !

```
- (MKAnnotationView *)mapView:(MKMapView *)theMapView
    viewForAnnotation:(id <MKAnnotation>)annotation
{
    // si c'est la position de l'utilisateur, on laisse tel quel
    if ([annotation isKindOfClass:[MKUserLocation class]])
        return nil;

    // On teste nos types d'annotations. Ici c'est inutile car on en a qu'une,
    // mais au moins vous saurez le faire !
    if ([annotation isKindOfClass:[MonAnnotation class]])
    {
        //On essaye de dépiler une annotation view en premier
        static NSString* MonAnnotationIdentifier = @"monAnnotationIdentifier";
        MKPinAnnotationView* pinView = (MKPinAnnotationView *)[maMapView
            // dequeueReusableAnnotationViewWithIdentifier:MonAnnotationIdentifier];
        if (!pinView)
        {
            // Si il n'y en avait aucune de disponible, on en crée une tout simplement !
            MKPinAnnotationView* customPinView = [[[MKPinAnnotationView alloc]
                // initWithAnnotation:annotation
                // reuseIdentifier:MonAnnotationIdentifier] autorelease];
            customPinView.pinColor = MKPinAnnotationColorGreen;
            customPinView.animatesDrop = YES;
            customPinView.canShowCallout = YES;

            // On ajoute une petite flèche sur le côté pour afficher plus
            // d'informations en cliquant dessus
            //
            // Info : on pourrait implémenter directement :
            // - (void)mapView:(MKMapView *)mapView annotationView:(MKAnnotationView *)
            //     view calloutAccessoryControlTapped:(UIControl *)control;
            // L'avantage ici est de choisir ce que l'on veut mettre

            UIButton* rightButton = [UIButton buttonWithType:UIButtonTypeDetailDisclosure];
            [rightButton addTarget:self
                           action:@selector(afficherLesDetails:)
                     forControlEvents:UIControlEventTouchUpInside];
        }
    }
}
```

```

        customPinView.rightCalloutAccessoryView = rightButton;
        // ajout d'une image à gauche
        UIImageView *imageView =
        [[[UIImageView alloc] initWithFrame:CGRectMake(0, 0, 20, 20)];
        imageView.image = [UIImage imageNamed:@"annotationImage.png"];
        customPinView.leftCalloutAccessoryView = imageView;
        [imageView release];

        return customPinView;
    }
    else
    {
        pinView.annotation = annotation;
    }
    return pinView;
}

return nil;
}

```

Mais il faut également les ajouter sur la carte... Dans RootViewController.h, déclarer un NSMutableArray *arrayOfAnnotations;, faites l'import de MonAnnotation.h et rajouter ces quelques lignes dans le viewDidLoad dans le .m :

```

// Ajout des annotations
arrayOfAnnotations = [[NSMutableArray alloc] initWithCapacity:2];

MonAnnotation *uneAnnotation = [[MonAnnotation alloc] init];
uneAnnotation.title = @"Paris";
uneAnnotation.subtitle = @"Capitale de la france";
uneAnnotation.longitude = [NSNumber numberWithDouble:2.0+20.0/60.0];
uneAnnotation.latitude = [NSNumber numberWithDouble:48.0+52.0/60.0];
[arrayOfAnnotations addObject:uneAnnotation];
[uneAnnotation release];

uneAnnotation = [[MonAnnotation alloc] init];
uneAnnotation.title = @"Marseille";
uneAnnotation.subtitle = @"Ville du sud";
uneAnnotation.longitude = [NSNumber numberWithDouble:5.0+22.0/60.0];
uneAnnotation.latitude = [NSNumber numberWithDouble:43.0+17.0/60.0];
[arrayOfAnnotations addObject:uneAnnotation];
[uneAnnotation release];

[maMapView addAnnotations:arrayOfAnnotations];

```

Ajoutez également cette méthode :

```
- (void) afficherLesDetails:(id)sender
{
    NSLog(@"ici, on pourrait présenter un controller qui affichera le détail");
}
```

Voici ce que vous devriez obtenir Figure 14.1.



Figure 14.1 : Une annotation de type pin personnalisée.

Pour finir, nous allons complètement choisir ce que l'on souhaite afficher sur la carte... Dans ce but, créez une nouvelle classe que vous nommerez `MonAnnotationView`, et implémentez-la comme suit :

```
MonAnnotationView.h
#import <MapKit/MapKit.h>

@interface MonAnnotationView : MKAnnotationView
{
}
@end

MonAnnotationView.m
#import "MonAnnotationView.h"

@implementation MonAnnotationView
- (id)initWithAnnotation:(id <MKAnnotation>)annotation
    reuseIdentifier:(NSString *)reuseIdentifier
```

```

{
    self = [super initWithAnnotation:annotation reuseIdentifier:reuseIdentifier];
    if (self != nil)
    {
        UIImageView *view = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0, 40, 40)];
        view.image = [UIImage imageNamed:@"annotationImage.png"];
        [self addSubview:view];
        [view release];
    }
    return self;
}

- (void)setAnnotation:(id <MKAnnotation>)annotation
{
    [super setAnnotation:annotation];
    // pour update si besoin l'affichage lors que l'on utilise reuseIdentifier
    [self setNeedsDisplay];
}

- (void)drawRect:(CGRect)rect
{
    // à implémenter si besoin de dessiner
}
@end

```

Et bien entendu, il faut modifier la méthode du RootViewController.m (ne pas oublier l'import de MonAnnotationView.m) :

```

- (MKAnnotationView *)mapView:(MKMapView *)theMapView viewForAnnotation:(id
➥ <MKAnnotation>)annotation
{
    // si c'est la position de l'utilisateur, on laisse tel quel
    if ([annotation isKindOfClass:[MKUserLocation class]])
        return nil;

    // On teste nos types d'annotations. Ici c'est inutile car on en a qu'une,
    ➥ mais au moins vous saurez le faire !
    if ([annotation isKindOfClass:[MonAnnotation class]])
    {
        //On essaye de dépiler une annotation view en premier
        static NSString* MonAnnotationIdentifier = @"monAnnotationIdentifier";
        MonAnnotationView* pinView = (MonAnnotationView *)[maMapView
        ➥ dequeueReusableCellWithIdentifierWithIdentifier:MonAnnotationIdentifier];
        if (!pinView)
        {

```

```

// S'il n'y en avait aucune de disponible, on en crée une tout simplement !
MonAnnotationView* customView = [[[MonAnnotationView alloc]
initWithAnnotation:annotation reuseIdentifier:MonAnnotationIdentifier]
➥ autorelease];
customView.canShowCallout = YES;
return customView;
}
else
{
    pinView.annotation = annotation;
}
return pinView;
}
return nil;
}

```

Le résultat est illustré Figure 14.2.

Ouf... C'était facile mais long ! Avant l'iOS 3.0, cette carte n'existe pas. Il fallait donc employer un autre framework : CoreLocation. Il se révélera très pratique lorsque vous aurez besoin d'utiliser les coordonnées GPS de l'utilisateur sans pour autant afficher sa position sur une carte.

Pour la petite histoire, une des solutions retenues pour afficher la position de l'utilisateur était de recourir à une WebView affichant le site de Google Maps. Ce n'était pas très pratique ! Cependant, ne négligez pas la suite de cette fiche !



Figure 14.2 : Une annotation complètement personnalisée.

UTILISER LE FRAMEWORK CORELOCATION POUR RÉCUPÉRER LA POSITION

Créez comme toujours un nouveau projet Windows-based Application que nous appellerons GPS2 (très créatif comme nom !) et ajoutez-y une classe RootViewController. Ensuite, ajoutez le framework CoreLocation.

Avec ce framework, nous allons créer un objet de la classe CLLocationManager. Ainsi, nous allons pouvoir utiliser à la fois la puce GPS, et la boussole ! Regardons le code de plus près :

```

RootViewController.h
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>

@interface RootViewController : UIViewController <CLLocationManagerDelegate> {
    CLLocationManager *locationManager;
    BOOL isCompassAvailable;
}

```

```

UITextView *textViewDonneesGPS;
UILabel *labelDonneesHeading;
}
@end
RootViewController.m
#import "RootViewController.h"
@implementation RootViewController

- (void)viewDidLoad {
    // -----
    // Configuration du location Manager
    // -----

    // on alloue un Manager pour la location
    locationManager = [[CLLocationManager alloc] init];
    // self est le delegate
    locationManager.delegate = self;
    // Par défaut, on met la meilleure localisation
    locationManager.desiredAccuracy = kCLLocationAccuracyBest; ①
    // on update autant que possible
    locationManager.distanceFilter = kCLLocationAccuracyNone; ②

    // On vérifie que le device a bien la boussole de dispo
    if ([CLLocationManager headingAvailable] == NO) { ③
        // pour iPhone OS < 4.0 : if (locationManager.headingAvailable == NO)
        // pas de boussole disponible
        isCompassAvailable = NO;
    } else {
        isCompassAvailable = YES;
        // configuration du filtre pour la boussole
        locationManager.headingFilter = kCLHeadingFilterNone;
    }
    // -----
    // Elements graphiques
    //

    textViewDonneesGPS = [[UITextView alloc] initWithFrame:CGRectMake(10, 10, 300, 200)];
    textViewDonneesGPS.editable = NO; ④
    [self.view addSubview:textViewDonneesGPS];

    labelDonneesHeading = [[UILabel alloc] initWithFrame:CGRectMake(0, 220, 320, 30)];
    // multi lignes
    labelDonneesHeading.numberOfLines = 0; ⑤
    [self.view addSubview:labelDonneesHeading];
}

```

```
// -----
// Démarrage localisation + boussole
// -----
// on lance la localisation
[locationManager startUpdatingLocation]; ⑥
// on lance la boussole
[locationManager startUpdatingHeading]; ⑦

[super viewDidLoad];
}

#pragma mark -
#pragma mark location delegate

- (void)locationManager:(CLLocationManager *)manager didUpdateToLocation:(CLLocation
➥ *)newLocation fromLocation:(CLLocation *) oldLocation { ⑧
    NSMutableString *stringToDisplay = [NSMutableString string]; ⑨
    [stringToDisplay appendFormat:@"lat : %f long : %f\n",
     ➤ newLocation.coordinate.latitude, newLocation.coordinate.longitude];
    [stringToDisplay appendFormat:@"altitude : %f\n", newLocation.altitude];
    [stringToDisplay appendFormat:@"vitesse : %f m/s \n", newLocation.speed];
    [stringToDisplay appendFormat:@"précision horiz : %f, vert : %f\n",
     ➤ newLocation.horizontalAccuracy, newLocation.verticalAccuracy];
    textViewDonneesGPS.text = stringToDisplay;
}

#pragma mark -
#pragma mark Delegate de la boussole(heading)
- (void)locationManager:(CLLocationManager *)manager
➥ didUpdateHeading:(CLHeading *)heading { ⑩
    labelDonneesHeading.text = [NSString stringWithFormat:@"%@%f ; x : %f, y : %f,
    ➤ z: %f", heading.trueHeading, heading.x, heading.y, heading.z]; ⑪
    [labelDonneesHeading sizeToFit]; ⑫
}

#pragma mark -
#pragma mark Gestion des erreurs

// Cette méthode est appelée lorsque l'objet actualisant la localisation
➥ rencontre une erreur
- (void)locationManager:(CLLocationManager *)manager didFailWithError:(NSError
➥ *)error { ⑬
    if ([error code] == kCLErrorDenied) { ⑭
```

```

    // Cette erreur indique que l'utilisateur a refusé l'accès au service de
    ↪ localisation
    [locationManager stopUpdatingHeading];
    [locationManager stopUpdatingLocation];
} else if ([error code] == kCLErrorHeadingFailure) { ⑯
    // Cette erreur indique que les données de la boussole ne peuvent être
    ↪ déterminées, probablement à cause d'interférences
}
}

- (void)dealloc {
    [textViewDonneesGPS release];
    [labelDonneesHeading release];

    [locationManager stopUpdatingHeading];
    [locationManager stopUpdatingLocation];
    [locationManager release];
    [super dealloc];
}
@end

```

Cela fait beaucoup en une fois, mais analysons pas à pas :

Ligne **①** : on peut spécifier la précision pour la mesure GPS. Sachez que plus fine sera la précision demandée, plus fréquents seront les appels à la puce GPS, et donc plus votre application consommera de batterie. Tomber en panne alors que l'on est perdu serait le comble ! Prennez donc un temps de réflexion avant de choisir `KCLLocationAccuracyBest`...

Ligne **②**, vous définissez la distance minimale de déplacement en mètre à atteindre, avant d'avertir le delegate. En mettant `kCLDistanceFilterNone`, le plus petit mouvement sera détecté. En mettant `3.0`, le delegate serait averti tous les 3 mètres. (Pour la boussole, le filtre intervient sur les degrés de rotation.)

L'iPhone 3G et l'iPod Touch n'ont pas de boussole disponible. Avant de l'utiliser dans votre application, il faut donc tester si l'appareil qui lance votre application possède ce capteur. C'est ce qui est écrit ligne **③**. Notez qu'avant l'iOS 4.0, on accédait à la propriété `headingAvailable` (`BOOL`) du `locationManager`, maintenant remplacée par une méthode de classe.

Pour afficher toutes ces données, nous allons utiliser une `text view` (pour les données GPS) et un `label` (pour les données du magnétomètre). Tous deux pourront afficher plusieurs lignes.

Ligne **④**, on choisit de rendre la `text view` non éditable. Pour rendre un `label` multiligne, il faut lui spécifier... 0 ligne ! C'est ce qui est fait ligne **⑤**. Ainsi, le `label` va automatiquement afficher les données en créant plusieurs lignes après appel de `sizeToFit` ligne **⑯**. Nos éléments définis et initialisés, il est temps de lancer les mises à jour de position (GPS + boussole) ligne **⑥** et **⑦**.

Il faut un delegate pour se servir correctement de la classe `CLLocationManager`. La méthode la plus importante à implémenter est celle de la ligne ⑧. En effet, elle renseigne le delegate de la position de l'utilisateur selon les données spécifiées lignes ① et ②.

Dans cette méthode, nous allons afficher les différentes ressources mises à votre disposition. Pour cela, on initialise un `NSMutableString` ligne ⑨, puis on ajoute petit à petit au string les valeurs de l'altitude en mètre, la vitesse en mètres par secondes, les coordonnées GPS (latitude longitude) en degrés, les précisions verticale et horizontale en mètres.

Info

Lorsqu'un mutable string (un string modifiable) `unString` vaut @"Hello ", puis que vous faites `[unString appendString:@"iPuP"];`, il vaudra @"Hello iPuP". `appendFormat` est l'équivalent de `appendString:[NSString stringWithFormat:]`

Ligne ⑩, on implémente selon le même principe la méthode delegate pour la boussole, appelée chaque fois que vous effectuez une rotation de l'angle minimum spécifié à l'initialisation. Cette fois-ci, on remplit le label directement ligne ⑪, avec notamment la valeur en degré de votre position par rapport au vrai nord (on peut choisir le nord magnétique).

Pour finir, comme dit précédemment dans cette fiche, la gestion des erreurs est primordiale (lignes ⑬, ⑭ et ⑮).

Info

Avec l'arrivée de l'iOS 4.0, l'utilisateur peut activer ou non les services de localisation dans les préférences de l'iPhone (notamment pour éviter d'avoir des applications exploitant votre position en tâche de fond). Vous devrez donc tester si l'utilisateur a ou non autorisé ces services avec `[CLLocationManager locationServicesEnabled];`. Si la valeur renournée est non et que vous démarrez tout de même la localisation, une alert view s'affichera, demandant à l'utilisateur s'il souhaite réactiver ce service.

Votre iPhone est également un iPod. Apple autorise l'accès à la bibliothèque de l'utilisateur en lecture : impossible de l'écrire (modifier les informations d'un morceau par exemple). En plus de jouer de la musique, nous verrons comment interagir avec l'accéléromètre.

Nous allons accéder à la bibliothèque : créez un nouveau projet Window-based Application que vous nommerez ShakeToPlay, et ajoutez un RootViewController. Ensuite, ajoutez le framework MediaPlayer.

Info

La bibliothèque de l'iPod n'est pas accessible sur simulateur, il vous faudra donc compiler sur iPhone, iPod Touch ou iPad. Il en va de même pour l'accéléromètre !

JOUER LA MUSIQUE DE L'IPod

Dans un premier temps, affichons un bouton Play/Pause ainsi qu'un autre pour sélectionner les morceaux :

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

@interface RootViewController : UIViewController <MPMediaPickerControllerDelegate> {
    MPMusicPlayerController *musicPlayer;    ①
    UIButton *playPauseButton;
}
- (void) registerForMediaPlayerNotifications;
@end

RootViewController.m
#import "RootViewController.h"

@implementation RootViewController

- (void) registerForMediaPlayerNotifications {

    NSNotificationCenter *notificationCenter = [NSNotificationCenter defaultCenter];
    [notificationCenter addObserver: self
        selector: @selector(gerer_LelementJoueEnCoursAChange:)
        name: MPMusicPlayerControllerNowPlayingItemDidChangeNotification
        object: musicPlayer]; ②

    [notificationCenter addObserver: self
        selector: @selector(gerer_LetatPlayPauseAChange:)
        name: MPMusicPlayerControllerPlaybackStateDidChangeNotification
        object: musicPlayer];
}
```

```
[musicPlayer beginGeneratingPlaybackNotifications];
}

- (void)viewDidLoad {
    // bouton pour lancer la sélection d'une chanson
    UIButton *mediaPickerController = [UIButton buttonWithType:UIButtonTypeContactAdd];
    mediaPickerController.center = CGPointMake(20, 20);
    [mediaPickerController addTarget:self action:@selector(selectASong:)
        forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:mediaPickerController];

    // Gestion de la lecture de la musique
    // bouton qui gère le play pause
    playPauseButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [playPauseButton setTitle:@"Play" forState:UIControlStateNormal];
    [playPauseButton setTitle:@"Pause" forState:UIControlStateSelected];
    [playPauseButton addTarget:self action:@selector(playOrPauseMusic:)
        forControlEvents:UIControlEventTouchUpInside];
    [playPauseButton setFrame:CGRectMake(10, 50, 50, 30)];
    // on empêche l'utilisateur de cliquer dessus tant qu'il n'a pas sélectionné
    // de musique
    playPauseButton.enabled = NO; ③
    [self.view addSubview:playPauseButton];

    // on initialise le music player
    musicPlayer = [[MPMusicPlayerController applicationMusicPlayer] retain]; ④

    // on lance l'abonnement aux notifications
    [self registerForMediaPlayerNotifications]; ⑤

    [super viewDidLoad];
}

#pragma mark -
#pragma mark button Methods

- (void) selectASong:(id)sender
{
    // initialisation et affichage du picker pour choisir la musique
    MPMediaPickerController *picker = [[MPMediaPickerController alloc]
        initWithMediaTypes: MPMediaTypeMusic]; ⑥

    picker.delegate = self;
    picker.allowsPickingMultipleItems = YES; ⑦
    picker.prompt = @"Choisir votre liste de lecture";
```

```

[self presentModalViewController: picker animated: YES]; ⑧

[picker release];
}

- (void) playOrPauseMusic:(id)sender {
    MPMusicPlaybackState playbackState = [musicPlayer playbackState];

    if (playbackState == MPMusicPlaybackStateStopped || playbackState ==
        ↪ MPMusicPlaybackStatePaused) {
        [musicPlayer play];
    } else if (playbackState == MPMusicPlaybackStatePlaying) {
        [musicPlayer pause];
    }
}

#pragma mark -
#pragma mark MPMediaPickerControllerDelegate methods
- (void)mediaPicker: (MPMediaPickerController *)mediaPicker
    ↪ didPickMediaItems:(MPMediaItemCollection *)mediaItemCollection
{
    // on vient de choisir une chanson, on autorise le bouton play/pause
    playPauseButton.enabled = YES;

    BOOL wasPlaying = ([musicPlayer playbackState] ==
        ↪ MPMusicPlaybackStatePlaying);

    [musicPlayer setQueueWithItemCollection: mediaItemCollection]; ⑨
    // on joue le morceau si on n'était pas en pause
    if (wasPlaying) {
        [musicPlayer play];
    }

    [mediaPicker dismissModalViewControllerAnimated:YES]; ⑩
}
- (void)mediaPickerDidCancel:(MPMediaPickerController *)mediaPicker
{
    [mediaPicker dismissModalViewControllerAnimated:YES];
}

#pragma mark -
#pragma mark gestionnaires notifications Musique

- (void) gerer_LelementJoueEnCoursAChange: (id) notification { ⑪
}
// Lorsque l'état du player change, on met à jour le bouton play/pause
- (void) gerer_LetatPlayPauseAChange: (id) notification {

```

```
MPMusicPlaybackState playbackState = [musicPlayer playbackState];

playPauseButton.enabled = (playbackState != MPMusicPlaybackStateStopped); ⑫
[playPauseButton setSelected:(playbackState == MPMusicPlaybackStatePlaying)]; ⑬

if (playbackState == MPMusicPlaybackStateStopped) {
    // On désactive le bouton
    [playPauseButton setEnabled:NO];
    // Même si le lecteur était arrêté, on stoppe quand même pour être sûr
    ➔ de vider le buffer
    [musicPlayer stop];
}

- (void)dealloc {
    [musicPlayer stop];
    [musicPlayer release];
    NSNotificationCenter *notificationCenter = [NSNotificationCenter defaultCenter];
    [notificationCenter removeObserver:self];
    [playPauseButton release];
    [super dealloc];
}
@end
```

Lignes ① et ④ : pour jouer de la musique, nous avons besoin de créer et d'initialiser un objet de la classe `MPMusicPlayerController`. Ici, on choisit `applicationMusicPlayer`, qui spécifie que le lecteur sera propre à votre application. Vous pourriez choisir `iPodMusicPlayer`, ce qui aurait pour effet d'utiliser le lecteur iPod de votre iPhone. En quittant votre application, si elle joue de la musique, l'application iPod prendra le relais et continuera la lecture.

Pour lancer la lecture on se sert d'un bouton (`playPauseButton`). Afin d'éviter toute erreur, on le désactive ligne ③. On le réactive par la suite lorsque l'on choisit un ou plusieurs morceaux.

Ligne ⑤, on lance une méthode qui permet de s'enregistrer à des notifications à la réception (en terme de programmation KVO (*Key Value Observing*)). En fait, ligne ②, on spécifie que `self` sera un observateur de l'événement "le morceau en cours de lecture a changé". Chaque fois que ce sera le cas, la méthode ligne ⑪ sera appelée.

Lorsque l'on clique sur le bouton pour ajouter une chanson, un picker (`MPMediaPickerController`) apparaît. C'est une vue que vous pouvez solliciter directement pour accéder aux éléments contenus dans l'iPod.

En une ligne, il s'affiche à l'écran ! (ligne ⑧). C'est le même principe que pour accéder à l'album photo.

Ligne ⑥, on choisit de ne prendre que la musique. Vous pouvez afficher les podcasts, les livres audio ou tout le contenu multimédia de l'appareil. Il est également possible de sélectionner plusieurs morceaux à la fois, comme on le spécifie ligne ⑦.

Se servir du picker implique d'implémenter deux méthodes delegate : une appelée lorsque l'utilisateur valide son choix, et l'autre lorsqu'il annule. Très souvent, il faudra mettre au moins l'appel ligne ⑩ pour enlever de la vue le picker. Pour que le buffer contenant la liste de lecture soit remis à jour, il faut appeler la méthode `play` ligne ⑨. En fait, appeler `play` permet de s'assurer que l'on prend en compte le nouveau buffer.

Passons aux méthodes appelées grâce au système de notifications. Nous allons implémenter plus tard la méthode appelée ligne ⑪ lorsque le morceau change. Pour éviter que le bouton Play ne soit actif quand le lecteur n'a aucun morceau disponible à la lecture, on ne l'active que lorsque le player n'est pas arrêté (aucun morceau joué) ligne ⑫. Ligne ⑬, on change l'état du bouton pour afficher soit le texte Play, soit Pause.

Continuons en affichant les données de la musique en cours et commençons par l'image de l'album. Dans le viewDidLoad, ajoutez ces quelques lignes pour instancier une image view :

```
// Affichage des informations de la musique en cours  
// l'image qui affichera la couverture de l'album  
mediaArtworkImageView = [[UIImageView alloc] initWithFrame:CGRectMake(60, 90,  
➥ 200, 200)];  
[self.view addSubview:mediaArtworkImageView];
```

Ensuite, dans la méthode gerer_LelementJoueEnCoursAChange :

```
// Lorsque le morceau joué change, on update l'affichage de l'image  
- (void) gerer_LelementJoueEnCoursAChange: (id) notification {  
    MPMediaItem *currentItem = [musicPlayer nowPlayingItem];  
    UIImage *artworkImage = nil;  
  
    // On récupère l'image si elle existe  
    MPMediaItemArtwork *artwork = [currentItem valueForProperty:  
        ➤ MPMediaItemPropertyArtwork];  
  
    // On la transforme en image si elle existe  
    if (artwork) {  
        artworkImage = [artwork imageWithSize: CGSizeMake (200, 200)];  
    }  
    mediaArtworkImageView.image = artworkImage;  
}
```

De plus, pour éviter que l'image ne reste quand la liste de lecture est finie d'être lue, ajoutez la ligne en gras dans la méthode gerer_LetatPlayPauseAChange : :

```
if (playbackState == MPMusicPlaybackStateStopped) {  
    // On désactive le bouton  
    [playPauseButton setEnabled:NO];  
    // on enlève l'image  
    [mediaArtworkImageView setImage:nil];  
    // Même si le lecteur était arrêté, on stoppe quand même pour être sûr de  
    ➤ vider le buffer  
    [musicPlayer stop];  
}
```

Ensuite, quelques informations concernant le morceau joué s'afficheront dans un label, comme le titre de la chanson, de l'album, le nom de l'artiste et le classement.

Commencez par ajouter l'initialisation du label dans le viewDidLoad :

```
// label pour afficher les informations de la chanson en cours  
mediaPlayingInformationLabel = [[UILabel alloc] initWithFrame:CGRectMake(10, 300,  
➥ 300, 100)];  
mediaPlayingInformationLabel.numberOfLines = 0;  
mediaPlayingInformationLabel.lineBreakMode = UILineBreakModeClip;  
[self.view addSubview:mediaPlayingInformationLabel];
```

Ensuite, modifiez gerer_elementJoueEnCoursAChange::

```
// On affiche les informations du morceau en cours  
[mediaPlayingInformationLabel setText: [  
    NSString stringWithFormat: @"%@ — %@", rate = %@",  
    [currentItem valueForProperty: MPMediaItemPropertyTitle], ①  
    [currentItem valueForProperty: MPMediaItemPropertyArtist], ②  
    [currentItem valueForProperty: MPMediaItemPropertyAlbumTitle], ③  
    [currentItem valueForProperty: MPMediaItemPropertyRating]]]; ④  
  
if (musicPlayer.playbackState == MPMusicPlaybackStateStopped) {  
    [mediaPlayingInformationLabel setText:@"Choisir une nouvelle liste !"];  
}
```

Ligne ①, on affiche le titre de la chanson, ligne ②, le nom de l'artiste, ligne ③, le nom de l'album et ligne ④ le classement du morceau.

Pour finir, nous allons ajouter quelques contrôles pour changer de morceau. Ajoutez donc deux boutons dans le viewDidLoad :

```
// Gestion de la lecture de la musique  
// bouton previous song  
UIButton *previousSong = [UIButton buttonWithType:UIButtonTypeRoundedRect];  
[previousSong setFrame:CGRectMake(70, 50, 90, 30)];  
[previousSong setTitle:@"En arrière" forState:UIControlStateNormal];  
[previousSong addTarget:self action:@selector(previousSong:)  
➥ forControlEvents:UIControlEventTouchUpInside];  
[self.view addSubview:previousSong];  
// bouton next song  
UIButton *nextSong = [UIButton buttonWithType:UIButtonTypeRoundedRect];  
[nextSong setFrame:CGRectMake(170, 50, 90, 30)];  
[nextSong setTitle:@"En avant" forState:UIControlStateNormal];  
[nextSong addTarget:self action:@selector(nextSong:)  
➥ forControlEvents:UIControlEventTouchUpInside];  
[self.view addSubview:nextSong];
```

Comme vous le voyez, il faut ajouter ces deux méthodes :

```
- (void) nextSong:(id)sender {
    [musicPlayer skipToNextItem];
}
- (void) previousSong:(id)sender {
    [musicPlayer skipToPreviousItem];
}
```

Et n'oubliez pas de mettre à jour le dealloc !

```
- (void)dealloc {
    [musicPlayer stop];
    [musicPlayer release];
    [playPauseButton release];
    NSNotificationCenter *notificationCenter = [NSNotificationCenter defaultCenter];
    [notificationCenter removeObserver:self];

    [mediaArtworkImageView release];
    [mediaPlayingInformationLabel release];
    [super dealloc];
}
```

Voilà, vous avez un miniplayer qui devrait ressembler à la Figure 15.1.



Figure 15.1 : Votre lecteur de musique.

UTILISER L'ACCÉLÉROMÈTRE

Finissons cette fiche en nous intéressant à l'accéléromètre. Les données seront brutes, il faudra les interpréter en faisant un peu de mécanique des solides, puis mettre en forme avec les mathématiques du signal ! Nous allons voir comment détecter le fait que l'utilisateur secoue son iPhone.

Pour commencer, nous allons définir quelques constantes, juste en dessous du #import dans le RootViewController.m

```
#define kAccelerometerFrequency      40    // fréquence (Hz) de mise à jour des
                                            // données de l'accéléromètre
#define kFilteringFactor            0.1   // constante utilisée pour un filtre
                                            // passe haut
#define kMinEraseInterval          0.5   // intervalle minimal en secondes
                                            // entre deux échantillonnages pour
                                            // déclencher la fonction "secouer"
#define kEraseAccelerationThreshold  2.0   // seuil d'accélération pour lequel
                                            // on détecte la fonction "secouer"
```

Pour choisir la valeur de la fréquence, référez-vous à ces intervalles :

- **10-20 Hz.** Adapté pour déterminer le vecteur d'orientation de l'appareil.
- **30-60 Hz.** Adapté pour les jeux et autre applications qui nécessitent l'accéléromètre comme données d'entrées (les jeux de voitures par exemple).
- **70-100 Hz.** Adapté pour les applications qui détectent des mouvements haute fréquence comme la détection des mouvements saccadés rapides.

Pas besoin de créer un objet de la classe UIAccelerometer pour l'utiliser. Écrivez ces lignes dans le viewDidLoad :

```
// on lance l'échantillonnage de l'accéléromètre
[[UIAccelerometer sharedAccelerometer] setUpdateInterval:(1.0 /
➥ kAccelerometerFrequency)]; ①
[[UIAccelerometer sharedAccelerometer] setDelegate:self];
```

Faisons une petite pause théorie : lorsque vous voyez une syntaxe comme à la ligne ①, cela signifie que la classe sollicitée (ici UIAccelerometer) est une classe **Singleton**. En d'autres termes, c'est une classe qui instancie elle-même un objet de sa classe et le partage... Voyons ça avec quelques lignes de code. Voici à quoi devrait ressembler la méthode sharedAccelerometer :

```
+ (UIAccelerometer *)sharedAccelerometer
{
    static UIAccelerometer *sharedAccelerometer;
    @synchronized(self) ①
    {
        if (! sharedAccelerometer)
            sharedAccelerometer = [[UIAccelerometer alloc] init];

        return sharedAccelerometer; } return nil;
    }
}
```

Oups ! qu'est-ce que @synchronized ligne ① ? L'Objective-C supporte le multithreading. Avec le multithreading, deux threads sont susceptibles de modifier le même objet au même moment et cela peut vite devenir un cauchemar... Pour protéger des sections de code et ne leur permettre d'être exécutées que par un seul thread à la fois, il existe le mot-clé @synchronized. Les autres threads sont bloqués tant que l'exécution de la section de code par le thread qui en a le droit n'est pas sorti de la section englobée par @synchronized.

Revenons à notre accéléromètre et implémentez la méthode delegate :

```
#pragma mark -
#pragma mark Accelerometer delegate method
- (void)accelerometer:(UIAccelerometer *)accelerometer
    didAccelerate:(UIAcceleration *)acceleration
{
    UIAccelerationValue length,x,y,z; // ce sont les paramètres de notre
    ↪ accelerometre

    // On utilise un filtre passe-bas pour compenser l'influence de la gravité :
    ↪ on retrouve nos constantes déclarées plus haut dans les #define
    myAccelerometer[0] = acceleration.x * kFilteringFactor +
    ↪ myAccelerometer[0] * (1.0 - kFilteringFactor);
    myAccelerometer ① = acceleration.y * kFilteringFactor +
    ↪ myAccelerometer ① * (1.0 - kFilteringFactor);
    myAccelerometer ② = acceleration.z * kFilteringFactor +
    ↪ myAccelerometer ② * (1.0 - kFilteringFactor); ①

    // On calcule les valeurs pour nos 3 axes de l'accéléromètre (transformation
    ↪ en filtre passe haut simplifié)
    x = acceleration.x - myAccelerometer[0];
    y = acceleration.y - myAccelerometer ①;
    z = acceleration.z - myAccelerometer ②; ②

    // On calcule l'intensité de l'accélération courante
    length = sqrt(x * x + y * y + z * z);

    // Si l'iphone est secoué, on fait quelque chose
    // Pour utiliser l'accéléromètre, on scrute la différence de déplacement
    ↪ ou/et d'accélération entre deux instants donnés
    // La méthode CFAbsoluteTimeGetCurrent() permet d'obtenir la date de
    ↪ l'instant t2, qu'on compare a l'instant t1 stocké dans lastTime
    if((length >= kEraseAccelerationThreshold) && (CFAbsoluteTimeGetCurrent() >
    ↪ lastTime + kMinEraseInterval)) ③
    {
        [self performSelector:@selector(playOrPauseMusic:) withObject:nil];
        // on met à jour le dernier moment où l'on a secoué
        lastTime = CFAbsoluteTimeGetCurrent();
    }
}
```

Ligne ①, on utilise un filtre passe bas simplifié, qui compensera l'influence de la gravité. Ici, on génère une valeur faite de 10 % (0,1 kFilteringFactor) de la valeur actuelle de l'accélération et de 90 % de la précédente valeur.

Ligne ②, on soustrait la valeur obtenue avec le filtre passe bas à la valeur de l'accélération pour obtenir un filtre passe haut simplifié. Le filtre passe haut nous permet d'isoler les mouvements brusques. Enfin, on regarde si la force du shake est bien supérieure au seuil pour lequel on veut réagir, et que nous n'avons pas reçu un autre shake trop récemment, ici 0,5 secondes (pour éviter tout souci avec plusieurs pics rapprochés dans l'échantillonnage) ligne ③.

Notez que l'on peut implémenter des filtres plus évolués à partir des équations obtenues en électrotechnique avec les filtres RC, RL...

Si vous compilez et secouez votre iPhone (avec au moins un morceau dans la liste bien sûr), il se mettra en Pause/Lecture !

LES FILTRES

Un filtre passe bas ne laisse passer que les données en basse fréquence. Par exemple, quand vous écoutez de la musique, un filtre passe bas ne vous restituera que les basses sans les aigus.

Un filtre passe haut, c'est l'inverse. Dans notre exemple du son, l'application de ce filtre ne laisserait passer que les aigus.

Avec l'arrivée de l'OS 4.0, Apple a répondu à une forte demande des utilisateurs : laisser une application tourner en tâche de fond. Cependant, il y a certaines règles à respecter. Enfin, nous utiliserons les notifications locales, qui ne nécessitent pas de serveur comme pour le push !

Premier point important et non des moindres, le multitâche de Apple n'est pas du multitâche classique. Et vous allez devoir travailler pour que votre application fonctionne en tâche de fond. Zut ! me direz-vous... Oui et non ! Certes, ce travail représente du temps, une compréhension du système et de la réflexion, mais il permet aussi d'optimiser le comportement global du système, ainsi que la consommation d'énergie.

En effet, Apple a choisi de garder la maîtrise des applications tournant en tâche de fond. Au lieu d'autoriser un multitâche pour toutes les applications lancées, ce qui risquerait de diminuer de manière drastique l'autonomie de l'appareil, votre application devra se conformer à certaines règles. Pour tourner en tâche de fond, votre application devra proposer au moins :

- un service de localisation (ex. : avertir des radars sur la route à proximité) ;
- de la voip (Voice Over IP) (ex. : rester en ligne sur un réseau de voip) ;
- la lecture de musique (ex. : lire de la musique en streaming).

De plus, vous pouvez lancer une tâche lorsque l'utilisateur appuie sur le bouton Home comme par exemple finir un téléchargement ou réaliser un upload. Par contre, ce temps est limité.

RENDRE VOTRE APPLICATION MULTITÂCHE

Pour que votre application propose au moins un des trois services ci-dessus, **voici quelques règles à respecter :**

IMPLÉMENTATION

Implémentez chacune de ces méthodes si possible afin de réagir à chaque changements d'état :

`application:didFinishLaunchingWithOptions:`: Informe que votre application vient d'être lancée. C'est ici que vous pourrez implémenter le chargement de fichiers xib par exemple.

`applicationDidBecomeActive`: Cette méthode est appelée quand votre application passe du statut inactif à actif. Cela peut se produire par exemple lorsque l'utilisateur ignore un appel ou un SMS. C'est ici que vous pouvez remettre en route votre application si elle était en pause par exemple.

`applicationWillResignActive`: À l'inverse, cette méthode est appelée lorsque vous passez de l'état actif à inactif. C'est donc ici que vous pourrez mettre en pause en stoppant les timers de mise à jour en OpenGL par exemple.

`applicationDidEnterBackground`: Une nouvelle méthode apparue avec l'iOS 4.0. Elle sera appelée à la place de `applicationWillTerminate`: lorsque l'utilisateur appuie sur le bouton Home.

Notez bien que vous devez utiliser cette méthode pour :

- libérer toute ressource inutile (par exemple l'affichage d'images) ;
- arrêter les timers (invalidate) ;
- sauver les données utilisateur ;
- sauver assez d'informations pour être en mesure de restaurer l'application lorsqu'elle se relancera.

Info

Ce mécanisme de sauvegarde de la position dans l'application est devenu très courant. Pas de panique, nous l'abordons à la Fiche 35.

`applicationWillEnterForeground`: Cette méthode sera appelée lorsque votre application passera de l'état de tâche de fond à celui d'actif.

`applicationWillTerminate`: Pour les appareils ne supportant pas le multitâche, cette méthode est classiquement appelée quand l'utilisateur appuie sur le bouton Home. Sinon, elle sera appelée lorsque, par exemple, le système aura besoin de libérer de la mémoire et quittera votre application.

APPLICATION EN TÂCHE DE FOND

Quand votre application est en tâche de fond, suivez ces recommandations :

- Ne continuez pas à mettre à jour votre interface graphique de quelle que manière que ce soit (utilisation d'OpenGL, mises à jour d'éléments du UIKit...).
- N'accédez pas aux ressources partagées comme le carnet d'adresses, les photos.
- Annulez toute communication réseau (protocole Bonjour, stopper les connexions asynchrones).
- Concentrez-vous sur une tâche précise car les ressources allouées sont limitées. Ne faites donc que ce qui est nécessaire !
- Si vous avez besoin de temps (votre application a environ 5 secondes pour réaliser ces tâches), il faudra recourir à un autre mécanisme permettant de demander au système du temps additionnel pour réaliser une tâche. Nous le verrons plus tard dans cette fiche.
- Tester impérativement si l'appareil supporte le multitâche et adapter le comportement en fonction :

```
UIDevice* device = [UIDevice currentDevice];
BOOL backgroundSupported = NO;
if ([device respondsToSelector:@selector(isMultitaskingSupported)])
    backgroundSupported = device.multitaskingSupported;
```

- Déclarer dans le fichier Info.plist ce que votre application va faire parmi les trois services disponibles en ajoutant la clé UIBackgroundModes et en choisissant la bonne valeur.

CONSEILS

- **Applications nécessitant la localisation.** Il est recommandé de ne mettre à jour fréquemment la position que si cela est vraiment nécessaire, comme les applications de guidage. Ainsi, il faut recourir à la méthode de CLLocationManager : - (void)startMonitoringSignificantLocationChanges. Cette méthode ne générera un événement que si l'utilisateur bouge de manière significative.

De plus, l'avantage est que si votre application est arrêtée complètement, elle sera relancée et mise en tâche de fond si la position de l'utilisateur change.

Une autre façon de faire est de changer le .plist. Ainsi, votre application continuera d'exploiter les coordonnées GPS lorsque l'utilisateur appuiera sur le bouton Home. Elle ne sera pas suspendue, en attente, mais continuera d'être exécutée en tâche de fond.

- **Applications utilisant la voip.** Elles doivent configurer les sockets pour la voip et donc communiquer avec les serveurs distants (pour rester en ligne par exemple). Ensuite, elles doivent faire appel à la méthode `setKeepAliveTimeout:handler:` qui déterminera la fréquence à laquelle votre application doit être réveillée pour maintenir le service. Le temps minimal étant de 600 secondes. Ensuite, vous avez 30 secondes pour fournir une réponse, sinon, votre application quittera. Notez également que vous aurez très certainement besoin de l'audio, et donc de spécifier les deux valeurs voip et audio dans le .plist.

Les Blocks

Si vous souhaitez réaliser une tâche au moment où votre application sera suspendue, comme l'upload d'un fichier, il faut tout d'abord apprendre une nouvelle syntaxe un peu particulière : les blocks. Bien que créés par Apple, ils ressemblent fortement aux fonctions en C. En fait, les blocks sont intégrés dans les fonctions et peuvent ainsi exploiter directement les variables locales. On les utilise également en tant que callbacks.

Voici un exemple de block :

```
int adder = 5;

int (^addTheAdder)(int) = ^(int toAdd) {
    return toAdd + adder;
};

NSLog(@"%@", addTheAdder(45));
// retournera 50 !
```

Ici, on déclare donc une variable `addTheAdder` qui retourne un int et prend un int en paramètre. Notez que l'on accède à la variable locale `adder` dans le block. On peut copier un block, le passer en paramètre d'une méthode, le passer à un autre thread...

De plus, il faudra utiliser GCD (*Grand Central Dispatch*) qui est un mécanisme simplifiant l'appel à différents threads pour réaliser des tâches. En clair, plutôt que de laisser le développeur gérer manuellement les threads (allocation, gestion des ressources entre threads, etc.) et risquer de ne rien optimiser, Apple a créé GCD qui est un code open source. On peut utiliser GCD en bas niveau, avec du C, comme cela va être le cas dans le code ci-après, ou en plus haut niveau si vous êtes plus frileux, avec des classes comme `NSOperationQueue`, `NSInvocationOperation`. Pour simplifier, vous allez dire à GCD : "Tiens, voilà un block de code à exécuter quand tu peux, débrouille-toi !" (sous-entendu bien sûr : "Crée un ou plusieurs threads pour le faire, mais je ne veux pas le savoir.").

Voici maintenant la méthode à implémenter pour réaliser une tâche dans un temps additionnel (dans l'application delegate) :

```
- (void)applicationDidEnterBackground:(UIApplication *)application
{
    UIApplication* app = [UIApplication sharedApplication];

    // Demande la permission de fonctionner en tâche de fond.
    // Dans le cas où la tâche dure trop longtemps, on crée un gestionnaire
    // d'expiration
    NSAssert(self->bgTask == UIInvalidBackgroundTask, nil);

    self->bgTask = [app beginBackgroundTaskWithExpirationHandler:^{
        // Dans le cas où la tâche finit quasiment en même temps
        // que l'appel sur le thread principal, on synchronise
        dispatch_async(dispatch_get_main_queue(), ^{
            if (self->bgTask != UIInvalidBackgroundTask)
            {
                [app endBackgroundTask:self->bgTask];
                self->bgTask = UIInvalidBackgroundTask;
            }
        });
    }];

    // On commence la tâche et on la retourne immédiatement
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
        // Faire ici le travail associé à la tâche

        // on synchronise l'appel sur le thread principal dans le cas
        // où le gestionnaire d'expiration se déclenche en même temps
        dispatch_async(dispatch_get_main_queue(), ^{
            if (self->bgTask != UIInvalidBackgroundTask)
            {
                [app endBackgroundTask:self->bgTask];
                self->bgTask = UIInvalidBackgroundTask;
            }
        });
    });
}
```

Ne pas oublier de déclarer dans le .h : UIBackgroundTaskIdentifier bgTask;.

UTILISER LES NOTIFICATIONS LOCALES

Passons maintenant aux notifications locales et créons un nouveau projet Window-based Application nommé Reveil et ajoutez-y un RootViewController. Modifiez-le comme suit :

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

#define kRepeatAlarm @"repeatAlarm" ①
#define kDate @"date"

@interface RootViewController : UIViewController {
    BOOL repeatAlarm;
    NSDate *scheduleDate;
}

-(void)scheduleAlarmForDate:(NSDate*)theDate;

@end

RootViewController.m
#import "RootViewController.h"

@implementation RootViewController

-(void)viewDidLoad {
    // Construction des éléments d'interface
    UIDatePicker *datePicker = [[UIDatePicker alloc] initWithFrame:CGRectMake(0,
    ↪ 245, 0, 0)]; ②
    datePicker.datePickerMode = UIDatePickerModeTime;
    [datePicker addTarget:self action:@selector(hasChangeDate:)
    ↪ forControlEvents:UIControlEventValueChanged];
    [self.view addSubview:datePicker];

    UILabel *labelRepeat = [[UILabel alloc] initWithFrame:CGRectMake(10, 30, 170, 30)];
    labelRepeat.text = @"Tous les jours ?";
    [self.view addSubview:labelRepeat];
    [labelRepeat release];

    UISwitch *switchRepeat = [[UISwitch alloc] initWithFrame:CGRectMake(200, 30,
    ↪ 50, 30)];
    [switchRepeat addTarget:self action:@selector(switchHandle:)
    ↪ forControlEvents:UIControlEventValueChanged];
    [self.view addSubview:switchRepeat];
    [switchRepeat release];
}
```

```
UIButton *buttonScheduleAlarm = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[buttonScheduleAlarm setTitle:@"Ajouter l'alarme" forState:UIControlStateNormal];
[buttonScheduleAlarm setFrame:CGRectMake(10, 60, 300, 30)];
[buttonScheduleAlarm addTarget:self action:@selector(scheduleAlarm:)];
➥ forControlEvents:UIControlEventTouchUpInside];
[self.view addSubview:buttonScheduleAlarm];

// gestion de la sauvegarde ③
repeatAlarm = [[NSUserDefaults standardUserDefaults] objectForKey:kRepeatAlarm];

NSString *dateSavedString =
➥ objectForKey:kDate];
NSDateFormatter* dateFormatter = [[[NSDateFormatter alloc] init] autorelease];
dateFormatter.dateFormat = @"dd MM yyyy HH:mm";
NSDate* date = [dateFormatter dateFromString:dateSavedString];

if (date) {
    [datePicker setDate:date animated:YES];
}
else {
    [datePicker setDate:[NSDate date] animated:YES];
}

[datePicker release];
[super viewDidLoad];
}

- (void) hasChangeDate:(id)sender {
    UIDatePicker *aDatePicker = (UIDatePicker*)sender;
    scheduleDate = [aDatePicker date];
}

- (void) scheduleAlarm:(id)sender {
    NSCalendar *gregorian = [[[NSCalendar alloc]
➥ initWithCalendarIdentifier:NSGregorianCalendar] autorelease];
    NSDateComponents *aComponent = [gregorian components:(NSDayCalendarUnit |
➥ NSMonthCalendarUnit | NSYearCalendarUnit | NSHourCalendarUnit |
➥ NSMinuteCalendarUnit | NSSecondCalendarUnit) fromDate:scheduleDate];
    [aComponent setSecond:0]; ④
    NSDate *theDate = [gregorian dateFromComponents:aComponent];

    [self scheduleAlarmForDate:theDate];
    // sauvegarde de la date du réveil
}
```

```

NSDateFormatter* dateFormatter = [[[NSDateFormatter alloc] init] autorelease];
dateFormatter.dateFormat = @"dd MM yyyy HH:mm";
NSString* dateString = [dateFormatter stringFromDate:theDate];
[[NSUserDefaults standardUserDefaults] setObject:dateString forKey:kDate];
}

(void) switchHandle:(id)sender {
    UISwitch *aSwitch = (UISwitch*)sender;
    repeatAlarm = aSwitch.on;
}

- (void)scheduleAlarmForDate:(NSDate*)theDate
{
    UIApplication* app = [UIApplication sharedApplication];
    NSArray* oldNotifications = [app scheduledLocalNotifications];

    // On enlève toutes les anciennes notifications avant d'en recréer une
    if ([oldNotifications count] > 0)
        [app cancelAllLocalNotifications]; ⑤

    // On crée une nouvelle notification
    UILocalNotification* alarm = [[[UILocalNotification alloc] init] autorelease];
    if (alarm)
    {
        alarm.fireDate = theDate;
        alarm.timeZone = [NSTimeZone defaultTimeZone];
        alarm.repeatInterval = repeatAlarm ? kCFCalendarUnitDay : 0; ⑥
        alarm.soundName = UILocalNotificationDefaultSoundName;
        alarm.alertBody = @"C'est l'heure !";
        alarm.alertAction = @"Se réveiller !";

        [app scheduleLocalNotification:alarm]; ⑦
    }
}

- (void)dealloc {
    [super dealloc];
}
@end

```

Ligne ②, on alloue un UIDatePicker qui permet de choisir l'heure à laquelle la notification se déclenche. Notez que l'on met 0 en largeur et hauteur, car le UIDatePicker va gérer lui-même ses dimensions. De plus, on choisit ici de n'afficher que l'heure et les minutes.

Ensuite, ligne ③, on récupère la date depuis un fichier de préférences. La classe NSDate respectant le protocole NSCoder, nous pourrions l'utiliser pour la sauver dans un fichier. Ici, dans le but purement pédagogique de faire manipuler les dates et strings, on choisit de la sauver sous forme de chaîne de caractères. Les clés nécessaires sont définies ligne ①. Pour que la notification soit déclenchée au changement de la minute, on met 0 pour les secondes ligne ④. Ainsi, si la date était 10 10 2010 13h34m08s, elle deviendrait 10 10 2010 13h34m00s.

On peut prévoir plusieurs notifications en même temps, mais nous choisissons ici d'annuler toutes les précédentes notifications mises en place par l'application ligne ⑤. Les notifications peuvent être répétées ligne ⑥. 0 correspondant à *aucune répétition*. Ici, on choisit de tester repeatAlarm. S'il est à YES, alors on répète tous les jours, sinon on ne répète pas. Avec la propriété soundName, on peut définir un son pour l'alarme à partir d'un fichier dans le main bundle de type caf, wav ou aiff. Par défaut il n'y en a aucun, et on peut également mettre le son par défaut avec UILocalNotificationDefaultSoundName. La propriété alertBody quant à elle, définit le texte de l'alerte de la notification. Le titre de l'alerte étant le nom de l'application, voir Figure 16.1. Lorsque l'écran de votre iPhone est verrouillé, vous pouvez également choisir le texte affiché à la place de déverrouiller via la propriété alertAction (voir Figure 16.2).

Enfin, ligne ⑦, on programme la notification !



Figure 16.1 : Une notification reçue sur l'écran des applications.



Figure 16.2 : Une notification reçue sur l'écran de veille.

L'iPhone intègre une puce bluetooth et à ce titre, vous pouvez l'utiliser pour créer des applications communicantes entre appareils à proximité.

Attention

Nous avons besoin ici du framework GameKit. Pour tester votre application, vous devrez posséder deux appareils. Cependant si vous n'en avez qu'un, rien ne vous empêche de suivre cette fiche pour comprendre le mécanisme !

Mais quel est l'objectif de cette fiche ?

Dans le SDK iOS est intégré le GameKit qui vous permet entre autres de gérer des connexions bluetooth entre appareils. Nous allons ici créer un petit jeu de type pierre feuille ciseau entre deux appareils. Pour cela, il faudra dans un premier temps gérer les connexions entre appareils, puis mettre en place le mécanisme de dialogue (protocole) pour les faire communiquer correctement.

Réaliser une connexion par Bluetooth entre deux appareils est relativement facile. Le plus dur consiste à gérer le format de données qui transitent.

CRÉER UN NOUVEAU PROJET ET INITIALISER LA COMMUNICATION

Pour commencer, créez un nouveau projet de type View-based Application que vous nommerez PierrefeuilleCiseau. Ensuite, ajoutez le framework GameKit en cliquant du bouton droit sur le dossier, dans votre projet, intitulé Frameworks puis add > Existing Frameworks et choisir GameKit.

Dans le fichier PierreFeuilleCiseauViewController.h, faites l'import du framework :

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

La première étape consiste à créer un bouton de connexion/déconnexion pour créer le réseau bluetooth. Pour cela, déclarer le bouton dans le .h :

Info

Par la suite, nous ne travaillerons que dans la classe PierreFeuilleCiseauViewController. .h correspondra donc à PierreFeuilleCiseauViewController.h et .m à PierreFeuilleCiseauViewController.m.

```
// bouton pour la connexion / déconnexion  
UIButton *buttonConnectDisconnect;
```

Pour créer ce réseau, il faudra utiliser un objet de la classe GKPeerPickerController pour afficher l'alerte présente dans le SDK permettant de voir les appareils à proximité. Ensuite, lorsque l'utilisateur aura choisi un appareil, il faudra créer une session. Pour cela, déclarez dans le .h :

```
// déclaration du picker pour se connecter au BT  
GKPeerPickerController *btPicker;  
// déclaration de la session BT
```

```
GKSession *currentSession;
```

Dans le .m, commençons par créer le bouton qui sera unique pour la connexion/déconnexion. Dans le viewDidLoad, ajoutez :

```
// Bouton de connexion / déconnexion  
buttonConnectDisconnect = [[UIButton buttonWithType:UIButtonTypeRoundedRect] retain];  
[buttonConnectDisconnect addTarget:self action:@selector(connectToBluetooth:)  
forControlEvents:UIControlEventTouchUpInside];  
[buttonConnectDisconnect setTitle:@"Connexion" forState:UIControlStateNormal];  
[buttonConnectDisconnect setTitle:@"Déconnexion" forState:UIControlStateSelected];  
[buttonConnectDisconnect setFrame:CGRectMake(30, 30, 260, 30)];  
[self.view addSubview:buttonConnectDisconnect];
```

Comme vous le voyez, il faut implémenter la méthode connectToBluetooth: qui servira à se connecter/déconnecter du réseau :

```
#pragma mark -  
#pragma mark Bluetooth Manager  
  
-(void) connectToBluetooth :(id) sender {  
  
    UIButton *buttonSender = (UIButton*)sender;  
    // on pourrait aussi récupérer à partir de buttonConnectDisconnect  
  
    BOOL wantToDisconnect = buttonSender.selected;  
    if (wantToDisconnect)  
    {  
        // on se déconnecte  
        [self.currentSession disconnectFromAllPeers];  
        // on libère la session  
        [currentSession release];  
    }  
    else  
    {  
        // déclaration du picker  
        btPicker = [[GKPeerPickerController alloc] init];  
        // self est delegate  
        btPicker.delegate = self; ①  
        // GKPeerPickerControllerTypeOnline : une connexion depuis Internet  
        // GKPeerPickerControllerTypeNearby : une connexion locale depuis le bluetooth  
        btPicker.connectionTypesMask = GKPeerPickerControllerTypeNearby; ②  
  
        // on affiche le picker  
        [btPicker show];  
    }  
  
    // on change l'état du bouton  
    buttonSender.selected = !buttonSender.selected;
```

```
}
```

En ligne ②, vous remarquez que vous pouvez choisir entre une connexion “locale” en utilisant le Bluetooth (`GKPeerPickerControllerTypeNearby`) ou une connexion Internet (`GKPeerPickerControllerTypeOnline`). Nous n’aborderons pas ce dernier point ici.

Lorsque le picker s’affichera à l’écran, il faudra répondre à certaines méthodes delegate ligne ①.

Ajoutez le protocole en gras dans le .h :

```
@interface PierreFeuilleCiseauViewController : UIViewController  
↳ <GKPeerPickerControllerDelegate>
```

Puis, regardons la méthode appelée lorsque l’utilisateur a choisi un ami à proximité et a accepté la communication. Il faudra notamment initialiser la session. Ajoutez ces lignes dans le .m :

```
#pragma mark -  
#pragma mark Bluetooth delegate  
  
// la connexion est établie  
- (void)peerPickerController:(GKPeerPickerController *)picker  
↳ didConnectPeer:(NSString *)peerID toSession:(GKSession *) session {  
    // on récupère le peerId de la personne connectée  
    self.gamePeerId = peerID; // copy ①  
    // on récupère la session  
    self.currentSession = session;  
    // self est délégué de la session  
    session.delegate = self; ②  
    // on met self pour recevoir les données entrantes  
    [session setDataReceiveHandler:self withContext:NULL]; ③  
    // on enlève le delegate du picker  
    picker.delegate = nil;  
    // on enlève le picker et on le release  
    [picker dismiss];  
    [picker autorelease];  
}  
}
```

Ligne ①, nous récupérons l’identifiant de l’utilisateur distant qui vient d’accepter de rejoindre le réseau. Il faut tout d’abord le déclarer dans le .h :

```
NSString *gamePeerId;
```

Puis utiliser les getters/setters pour `gamePeerId` et `currentSession`. Dans le .h :

```
@property (nonatomic, copy) NSString *gamePeerId;  
@property (nonatomic, retain) GKSession *currentSession;
```

Dans le .m :

```
@synthesize currentSession, gamePeerId;
```

Info

Pour récupérer le nom du peer id, il suffit de faire :

```
NSLog(@"peer id nom : %@", [currentSession displayNameForPeer:peerID]);
```

Sauvegarder cet identifiant est primordial car il sera nécessaire pour communiquer par la suite !

La classe PierreFeuilleCiseauViewController est déléguée de la session ligne ②. À ce titre, il faut modifier le .h :

```
@interface PierreFeuilleCiseauViewController : UIViewController
    ↪ <GKPeerPickerControllerDelegate, GKSessionDelegate>
```

Dans le .m, il faut implémenter les méthodes déléguées qui correspondent. La première consiste à gérer les changements d'états de la connexion :

```
#pragma mark -
#pragma mark Delegate de la session
//appelée lorsque la session change d'état
- (void)session:(GKSession *)session peer:(NSString *)peerID
    ↪ didChangeState:(GKPeerConnectionState)state {
    switch (state) {
        {
            case GKPeerStateConnected:
                NSLog(@"Connecté");
                break;
            case GKPeerStateDisconnected:
                NSLog(@"Déconnecté");
                // on release la session en cours
                [currentSession release];

                // on remet le bouton dans l'état de connexion
                buttonConnectDisconnect.selected = NO;
                break;
            default:
                break;
        }
    }
}
```

La deuxième consiste à nommer la session pour s'assurer que les appareils qui lancent votre application ne communiquent QUE entre eux et ne peuvent accepter d'autres appareils ne possédant pas l'application :

```
- (GKSession *)peerPickerController:(GKPeerPickerController *)picker
    ↪ sessionForConnectionType:(GKPeerPickerControllerConnectionType)type {
    GKSession *session = [[GKSession alloc] initWithSessionID:kRockPaperScissorSessionID
        ↪ displayName:nil sessionMode:GKSessionModePeer];
    return [session autorelease];
}
```

Il faut définir l'identifiant dans le .m :

```
#define kRockPaperScissorSessionID @"rockPaperScissor" // l'id de la session
```

Enfin, ligne ③, il faut spécifier une classe qui va réceptionner les données provenant des utilisateurs. Ici, c'est la classe PierreFeuilleCiseauViewController qui s'en charge.

Dans le cas où l'utilisateur annulerait la connexion, ajoutez :

```
// la connexion a été annulée
- (void)peerPickerControllerDidCancel:(GKPeerPickerController *)picker
{
    // on release le picker
    picker.delegate = nil;
    [picker autorelease];

    // on revient au bouton pour la connexion
    buttonConnectDisconnect.selected = NO;
}
```

Ça y est, nous venons de connecter deux appareils ! Mais rien n'est gagné encore...

TRANSMETTRE DES DONNÉES

Tout d'abord, revenons aux fondamentaux avant de se compliquer la tâche. Pour transmettre des données, il faut envoyer un objet de type NSData. Ainsi, si vous souhaitez par exemple envoyer un objet de la classe NSString, il suffira de faire :

```
NSString *string = @"Toto ou votre texte ici !";
NSData *data = [string dataUsingEncoding: NSASCIIStringEncoding];
[currentSession sendDataToAllPeers:data withDataMode:GKSendDataReliable error:nil];
```

En fait, il suffit simplement de l'encoder et de le transmettre sous forme de données (octets). Vous pouvez choisir le mode de transmission, ici GKSendDataReliable. Cela signifie que la donnée sera envoyée en boucle tant qu'elle ne sera pas reçue, en fonction d'un time out bien sûr. Le time out est le temps maximal pendant lequel le système va essayer de transmettre les données. Passé ce temps, le système abandonnera pour rendre la main. Utilisez ce mode de transmission lorsque les données à transmettre nécessitent d'être reçues. Si, par exemple, vous envoyez fréquemment des données de mise à jour de la position de l'utilisateur, servez-vous de GKSendDataUnreliable car ce n'est pas très important si un rafraîchissement manque à l'appel.

Pour recevoir le string et l'afficher sur les autres appareils, il faut écrire :

```
- (void)receiveData:(NSData *)data fromPeer:(NSString *)peer
{
    inSession:(GKSession *)session context:(void *)context {
        NSString *string = [[NSString alloc] initWithData:data
            encoding:NSUTF8StringEncoding];
        NSLog(@"%@", string);
        [string release];
}
```

Info

Il est recommandé de ne pas échanger de trop gros paquets de données à la fois, pour éviter de surcharger la connexion. Il faudra donc penser à découper les données à envoyer en paquets si besoin. La taille recommandée est de 1 000 octets soit à peu près 1 Ko (1 Ko = 1024 octets). Ne l'oubliez pas !

Nous allons un peu compliquer l'affaire ! Pour cela, il faut revenir au papier et au crayon puis dessiner. En fait, pour faciliter la compréhension et surtout l'échange de données, il est préférable de définir un serveur et un client. Ainsi, lorsque vous construisez l'architecture du programme et des échanges puis écrivez le code correspondant, raisonner en un même endroit pour à la fois le serveur et le client peut vite devenir un cauchemar. Il faut rester très concentré. Pour éviter tout souci, je vous conseille très fortement de concevoir une machine à état. Je vous propose une architecture (qui n'est pas unique mais que je suivrai par la suite) à la Figure 17.1 (côté serveur) et 17.2 (côté client).

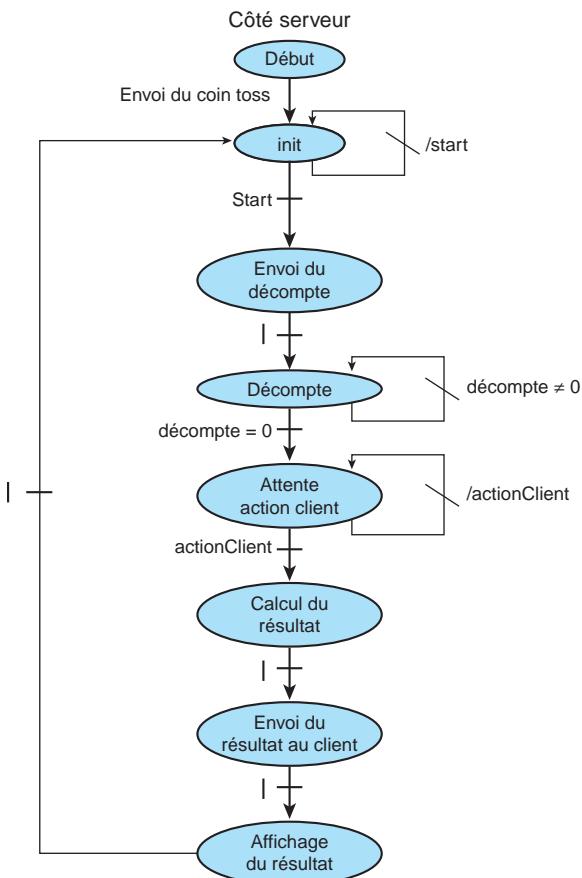


Figure 17.1 : Machine d'état côté serveur.

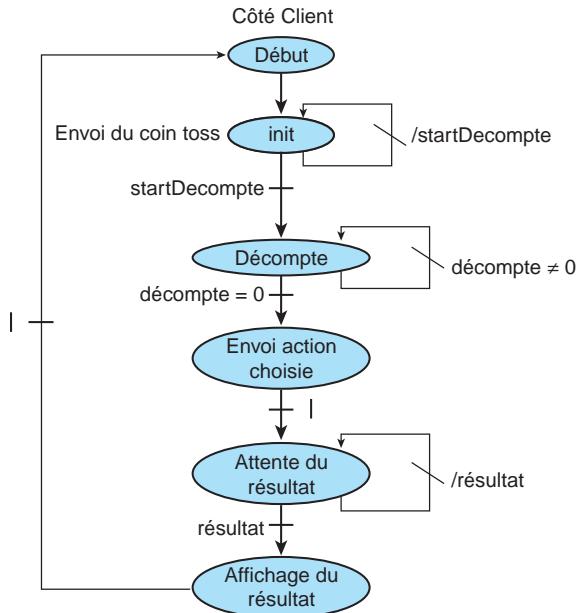


Figure 17.2 : Machine d'état côté client.

Pour commencer, on va se préoccuper de l'interface graphique. Il faudra ajouter trois boutons pour chaque action (Pierre, Feuille, Ciseau), un bouton pour lancer le jeu côté serveur, un label pour afficher si l'on est serveur ou client et un label qui affichera le décompte.

Ajoutez ces lignes dans le `viewDidLoad` :

```

// bouton pour commencer le jeu
buttonStartGame = [[UIButton buttonWithType:UIButtonTypeRoundedRect] retain];
[buttonStartGame addTarget:self action:@selector(startGame:)
  forControlEvents:UIControlEventTouchUpInside];
[buttonStartGame setTitle:@"Commencer" forState:UIControlStateNormal];
[buttonStartGame setFrame:CGRectMake(60, 70, 200, 30)];
// initialement on cache le bouton
buttonStartGame.hidden = YES;
[self.view addSubview:buttonStartGame];

// Label pour spécifier le statut (serveur ou client)
labelServerOrClient = [[UILabel alloc] initWithFrame:CGRectMake(60, 110, 200, 30)];
labelServerOrClient.textAlignment = NSTextAlignmentCenter;
labelServerOrClient.backgroundColor = [UIColor clearColor];
labelServerOrClient.text = @"Connectez-vous";
[self.view addSubview:labelServerOrClient];

```

```
// Bouton feuille
UIButton *buttonPaper = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[buttonPaper addTarget:self action:@selector(gameAction:)]
➥ forControlEvents:UIControlEventTouchUpInside];
[buttonPaper setFrame:CGRectMake(60, 190, 200, 30)];
[buttonPaper setTitle:@"Feuille" forState:UIControlStateNormal];
// on utilise le tag pour savoir à quoi correspond le bouton
[buttonPaper setTag:kPaper];
[self.view addSubview:buttonPaper];

// Bouton pierre
UIButton *buttonRock = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[buttonRock addTarget:self action:@selector(gameAction:)]
➥ forControlEvents:UIControlEventTouchUpInside];
[buttonRock setFrame:CGRectMake(60, 230, 200, 30)];
[buttonRock setTitle:@"Pierre" forState:UIControlStateNormal];
// on utilise le tag pour savoir à quoi correspond le bouton
[buttonRock setTag:kRock];
[self.view addSubview:buttonRock];

// Bouton Ciseau
UIButton *buttonScissor = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[buttonScissor addTarget:self action:@selector(gameAction:)]
➥ forControlEvents:UIControlEventTouchUpInside];
[buttonScissor setFrame:CGRectMake(60, 270, 200, 30)];
[buttonScissor setTitle:@"Ciseau" forState:UIControlStateNormal];
// on utilise le tag pour savoir à quoi correspond le bouton
[buttonScissor setTag:kScissor];
[self.view addSubview:buttonScissor];

// label décompte
labelCountDown = [[UILabel alloc] initWithFrame:CGRectMake(60, 420, 200, 30)];
labelCountDown.textAlignment = NSTextAlignmentCenter;
labelCountDown.backgroundColor = [UIColor clearColor];
labelCountDown.text = @"";
[self.view addSubview:labelCountDown];
```

C'est un peu long, mais il n'y a rien de compliqué si vous avez suivi correctement les fiches précédentes. Vous devrez ajouter dans le .h :

```
// Label pour notifier si l'utilisateur est serveur ou client
UILabel *labelServerOrClient;

// label décompte
UILabel *labelCountDown;
```

Soyez attentifs cependant car nous employons la propriété tag pour identifier les objets. En fait, nous pourrions très bien créer une méthode par action, mais cela peut vite encombrer votre code. Vous l'aurez compris, il faut définir kScissor, kRock et kPaper. Pour cela, ajoutez dans le .m (en dehors de l'implémentation) ce type énuméré :

```
// définition des états possibles du jeu !
typedef enum {
    kRock,
    kPaper,
    kScissor
} gameAction;
```

Ensuite, comme nous sommes dans les types énumérés, définissons les statuts possibles (toujours au même endroit) :

```
// définition de qui est serveur et qui est client
typedef enum {
    kServer,
    kClient
} gameNetwork;
```

Enfin, on définit les différentes étapes dont on va se servir dans la mise en place du protocole (voir les machines d'états aux Figures 17.1 et 17.2).

```
// les différents états nécessaires
typedef enum {
    NETWORK_COINROSS,      // Pour décider qui va être le serveur
    NETWORK_SEND_START_SIGNAL, // On envoie ce paquet pour commencer le compte à rebours
    NETWORK_SEND_ACTION, // On envoie pierre, feuille ou ciseau
    NETWORK_RESULT // on envoie le résultat du jeu
} packetCodes;
```

Ensuite, implémentons la méthode pour envoyer des données. En fait, plutôt que d'envoyer un morceau de texte comme nous l'avons vu juste avant, nous allons construire une trame qui contiendra un en-tête et les données. Dans l'en-tête, vous pouvez définir ce que vous souhaitez. Ici, nous ne nous préoccuperon que d'un identifiant parmi la structure énumérée packetCodes. Cela permet au destinataire d'être en mesure d'interpréter les données qu'il reçoit et pouvoir réaliser l'action définie par le protocole. Notez qu'il est également de bon ton d'inclure dans l'en-tête un comptage du nombre de trames. Ainsi, le destinataire peut les rassembler dans l'ordre. Ici, nous n'échangeons pas de données compliquées, ce n'est donc pas nécessaire.

```
#pragma mark -
#pragma mark Envoi réception des paquets

- (void)sendNetworkPacket:(GKSession *)session packetID:(int)packetID
    withData:(void *)data ofLength:(int)length reliable:(BOOL)howtosend {
    // On va construire une trame pour le paquet
    static unsigned char networkPacket[kMaxPacketSize]; ①
```

```

// on a un entier pour l'entête : il contient l'id du packet (défini dans
→ packetCodes)
const unsigned int packetHeaderSize = sizeof(int); ②
// on vérifie bien que la taille du paquet est la bonne, sinon on ignore
if(length < (kMaxPacketSize - packetHeaderSize)) {
    int *pIntData = (int *)&networkPacket[0];
    // on remplit le header
    pIntData[0] = packetID; ③
    // on copie la data à la suite
    memcpy( &networkPacket[packetHeaderSize], data, length ); ④
    // on convertit le paquet en objet NSData, nécessaire pour envoyer les
    → données
    NSData *packet = [NSData dataWithBytes: networkPacket length:
    → (length+packetHeaderSize)];
    if(howtosend == YES) {
        [session sendData:packet toPeers:[NSArray arrayWithObject:gamePeerId]
        → withDataMode:GKSendDataReliable error:nil]; ⑤
    } else {
        [session sendData:packet toPeers:[NSArray arrayWithObject:gamePeerId]
        → withDataMode:GKSendDataUnreliable error:nil];
    }
}

```

En ①, on réserve un tableau de la taille maximale qu'il faut définir en écrivant :

#define kMaxPacketSize 1024 // on réserve un peu plus. Cela correspond à la
↳ limite conseillée pour la taille d'un paquet

En ②, on définit la taille de l'en-tête. Ici, nous ne réservons de la place que pour un entier sur 32 bits (4 octets). `sizeOf(int)` renvoie donc 4.

En ③, on remplit l'en-tête puis en ④, on copie les données à la suite de l'en-tête. Vous noterez qu'en ⑤, nous n'envoyons pas les données à tous les utilisateurs susceptibles d'être connectés, mais à une liste que nous avons choisie. Ici, seuls deux utilisateurs peuvent communiquer en même temps.

L'envoi étant défini, il faut également réaliser la partie réception. Pour l'instant, ce sera une boîte vide :

```
- (void)receiveData:(NSData *)data fromPeer:(NSString *)peer
  ↪ inSession:(GKSession *)session context:(void *)context {
    unsigned char *incomingPacket = (unsigned char *)[data bytes];
    int *pIntData = (int *)&incomingPacket[0];

    int packetID = pIntData[0];

    switch( packetID ) {
        case NETWORK_COINTOSS:
    }
```

```

        }
        break;
    case NETWORK_SEND_START_SIGNAL:
    {
    }
    break;
    case NETWORK_SEND_ACTION:
    {
    }
    break;
    case NETWORK_RESULT:
    {
    }
    break;
    default:
        // erreur
        break;
    }
}

```

Commençons par le côté serveur (Figure 17.1). La première étape est de demander un identifiant à l'appareil en face pour savoir qui sera serveur...

Pour sécuriser l'identifiant et s'assurer qu'il soit unique, nous exploitons l'UDID (*Unique Device ID*) du téléphone que nous venons ensuite hasher (créer une valeur unique de telle manière qu'il soit impossible de retrouver l'original). Pour cela, écrire dans le viewDidLoad :

```

// Sécurité de l'échange
// on récupère l'UDID de l'appareil
NSString *udid = [[UIDevice currentDevice] uniqueIdentifier];
// on le hash pour protéger la donnée
gameUniqueId = [udid hash];

```

Il faut définir gameUniqueId dans le .h :

```
int gameUniqueId;
```

Les deux appareils vont envoyer la requête avec comme identifiant NETWORK_COINTOSS. Pour cela, nous allons la réaliser immédiatement après la connexion. Ajoutez cette ligne dans peerPickerController :didConnectPeer:toSession :

```

// on envoie le cointoss pour savoir qui sera client / serveur
[self sendNetworkPacket:currentSession packetID:NETWORK_COINTOSS
    withData:&gameUniqueId ofLength:sizeof(int) reliable:YES];

```

Lors de la réception, ce sera l'appareil qui aura le plus grand id qui sera serveur, l'autre sera client. Il faut trancher par moments ! Ajoutez les lignes en gras :

```

case NETWORK_COINTOSS:
{

```

```

// on récupère le coin toss pour savoir qui est serveur client
int coinToss = pIntData ①;
// si le jeton de l'autre est plus grand que le notre, il est serveur...
if(coinToss > gameUniqueID) {
    peerStatus = kClient;
}
else {
    peerStatus = kServer;
}

// on cache si on est client le bouton pour lancer le décompte
buttonStartGame.hidden = peerStatus == kClient;

// on notifie l'utilisateur si il est serveur ou client
labelServerOrClient.text = (peerStatus == kServer) ? @"Serveur" : @"Client";
}

```

Déclarez dans le .h la variable peerStatus :

```
NSInteger peerStatus;
```

On l'initialisera à *client* dans le viewDidLoad :

```
// on se met par défaut en client
peerStatus = kClient;
```

Après avoir défini qui était le serveur/client, le serveur attend que l'utilisateur clique sur le bouton Commencer pour lancer le décompte et informer le client de débuter également le décompte.

Pour cela, on implémente la méthode appelée quand on clique sur le bouton Commencer. Le bouton ne sera visible que par le serveur, selon ce qui a été défini ci-dessus.

```

- (void) startGame:(id)sender {
    NSLog(@"Start game");
    currentCountDown = kStartCounter; ①

    if (peerStatus == kServer) {
        // si on est serveur, on envoie le signal de départ et on commence le
        // décompte
        [self sendNetworkPacket:currentSession packetID:NETWORK_SEND_START_SIGNAL
            ↪ withData:[NSDate date] ofLength:sizeof(NSDate) reliable:YES]; ②
        [self decrementCount:nil]; ③
        [NSTimer scheduledTimerWithTimeInterval:1.0 target:self
            ↪ selector:@selector(decrementCount:) userInfo:nil repeats:YES]; ④
    }
    // sinon, on ne fait rien et on attend le signal de départ dans la méthode
    // de réception des données
}

```

Avant toute chose, pour que la ligne ① soit correcte, il faut déclarer dans le .h :

```
int currentCountDown;
```

et définir le nombre où l'on commence à décompter dans le .m :

```
#define kStartCounter 5 // on a 5 secondes pour choisir
```

Puis, ligne ②, on envoie une trame avec l'identifiant correspondant au lancement du décompte. On choisit de passer la date de l'envoi si l'on souhaitait faire un petit calcul par la suite pour rattraper le retard dû à la latence de la communication.

Après cette ligne, il faut commencer à décompter. Le timer est approprié pour réaliser ceci, mais il faut appeler une première fois la méthode à la main (ligne ③) pour commencer directement à décompter. Le timer est ensuite lancé ligne ④.

Le compte à rebours déclenché, il faut que l'utilisateur clique rapidement sur Papier, Feuille ou Ciseau. Dans la méthode appelée par ces boutons, nous allons mettre de côté la valeur cliquée. Pour cela, écrire :

```
- (void) gameAction:(id)sender {  
    UIButton *buttonSender = (UIButton*)sender;  
    // on récupère l'action depuis le tag du bouton  
    myGameAction = buttonSender.tag;  
}
```

Oui, myGameAction doit être définie dans le .h :

```
int myGameAction;
```

On l'initialise également dans le viewDidLoad :

```
// on met par défaut feuille comme game action  
myGameAction = kPaper;
```

Côté serveur, une fois que le décompte est lancé, il attend de recevoir l'action du client. Une fois qu'elle est reçue, il va chercher qui est le gagnant. Pour cela, ajoutez les lignes en gras :

```
case NETWORK_SEND_ACTION:  
{  
    // on vient de recevoir l'action du client  
    int clientAction = pIntData ①;  
    [self performResultWithClientAction:clientAction];  
}  
break;
```

Ensuite, on définit qui est gagnant du client ou du serveur :

```
- (void) performResultWithClientAction:(int)clientAction {  
    int clientResult;  
  
    switch (myGameAction) {  
        case kPaper:  
            switch (clientAction) {  
                case kPaper:  
                    clientResult = kEquality;  
                    break;  
                case kRock:
```

```
        clientResult = kLose;
        break;
    case kScissor:
        clientResult = kWin;
        break;
    default:
        break;
    }
    break;
case kRock:
    switch (clientAction) {
        case kPaper:
            clientResult = kWin;
            break;
        case kRock:
            clientResult = kEquality;
            break;
        case kScissor:
            clientResult = kLose;
            break;
        default:
            break;
    }
    break;
case kScissor:
    switch (clientAction) {
        case kPaper:
            clientResult = kLose;
            break;
        case kRock:
            clientResult = kWin;
            break;
        case kScissor:
            clientResult = kEquality;
            break;
        default:
            break;
    }
    break;
default:
    break;
}
```

Il y a comme vous le savez trois états possibles : perdre, gagner, ou être à égalité. Ajoutez le type énuméré dans le .m :

```
// cas possibles pour gagner, perdre, ou égalité
typedef enum {
    kWin,
    kLose,
    kEquality
} resultStatus;
```

Côté serveur, la dernière action est de notifier le client du résultat et de l'afficher. Pour ce faire, écrivez à la fin de `performResultWithClientAction:` :

```
// si on est serveur, on avertit le client de sa défaite ou non
if (peerStatus == kServer) {
    [self sendNetworkPacket:currentSession packetID:NETWORK_RESULT
       WithData:&clientResult ofLength:sizeof(int) reliable:YES];
    // on affiche le résultat
    [self displayResultForClientResult:clientResult];
}
```

L'affichage du résultat se fera dans une alert view. Il faudra tout d'abord faire correspondre le message avec le résultat. Pour cela, écrivez :

```
- (void) displayResultForClientResult:(int) clientResult {
    NSString *resultString;

    if (peerStatus == kClient) {
        switch (clientResult) {
            case kWin:
                resultString = @"gagné";
                break;
            case kLose:
                resultString = @"perdu";
                break;
            case kEquality:
                resultString = @"égalité";
                break;
            default:
                break;
        }
    }
    else
    {
        switch (clientResult) {
            case kWin:
                resultString = @"perdu";
                break;
        }
    }
}
```

```

        break;
    case kLose:
        resultString = @"gagné";
        break;
    case kEquality:
        resultString = @"égalité";
        break;
    default:
        break;
    }
}

UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Résultat"
                                              message:[NSString stringWithFormat:@"Vous
                                                ➔ avez %@", resultString]
                                              delegate:nil
                                              cancelButtonTitle:@"Ok"
                                              otherButtonTitles:nil];
[alert show];
[alert release];
}

```

Nous avons fini côté serveur. Passons du côté du client, et regardez la Figure 17.2. Déterminer qui est client/serveur a déjà été réalisé dans la partie serveur. L'étape suivante est d'attendre le top de départ pour commencer à décompter. Pour cela, écrivez ces lignes en gras :

```

case NETWORK_SEND_START_SIGNAL:
{
    currentCountDown = kStartCounter;

    // on a reçu le signal de départ, on commence à décompter
    // on lance une première fois la méthode
    [self decrementCount:nil];
    [NSTimer scheduledTimerWithTimeInterval:1.0 target:self
        ➔ selector:@selector(decrementCount:) userInfo:nil repeats:YES];
}
break;

```

Une fois que le décompte arrive à zéro, il faut notifier le serveur de l'action du client pour qu'il calcule le résultat. Pour cela, écrivez la méthode du timer :

```

#pragma mark -
#pragma mark Timer

- (void) decrementCount:(NSTimer*)theTimer {
    labelCountDown.text = [NSString stringWithFormat:@"%d", currentCountDown];
    // on décrémente le compteur
    currentCountDown--;
}

```

```

if (currentCountDown == 0)
{
    [theTimer invalidate];
    theTimer = nil;
    labelCountDown.text = @"";
    // si on est client, on envoie au serveur mon action
    if (peerStatus == kClient) {
        [self sendNetworkPacket:currentSession packetID:NETWORK_SEND_ACTION
            ↪ withData:&myGameAction ofLength:sizeof(int) reliable:YES];
    }
}
}

```

La dernière étape côté client est la réception du résultat pour l'afficher :

```

case NETWORK_RESULT:
{
    // on vient de recevoir le résultat du serveur
    int clientResult = pIntData;
    [self displayResultForClientResult:clientResult];
}
break;

```

N'oubliez jamais la méthode dealloc :

```

- (void)dealloc {
    [btPicker release];
    [currentSession release];

    [buttonConnectDisconnect release];

    [buttonStartGame release];
    [labelServerOrClient release];
    [labelCountDown release];

    [gamePeerId release];
    [super dealloc];
}

```

Et déclarez dans le .h ces méthodes pour vous débarrasser des derniers warnings :

- (void)sendNetworkPacket:(GKSession *)session packetID:(int)packetID
 ↪ withData:(void *)data ofLength:(int)length reliable:(BOOL)howtosend;
- (void) displayResultForClientResult:(int) clientResult;
- (void) decrementCount:(NSTimer*)theTimer;

Vous pouvez très facilement jouer une vidéo dans votre application, qu'elle soit locale ou disponible depuis le Web.

Il y a plusieurs façons de jouer une vidéo. Nous commencerons par lire une vidéo présente sur le Web, puis une en local, présente dans le dossier de l'application. Ensuite, nous verrons comment ajouter une vue par-dessus, également appelée *overlay view*. Enfin, nous verrons comment jouer une vidéo sans la présenter en plein écran, une nouveauté !

Les formats de vidéo acceptés sont :.mov,.mp4,.3gp et .mpv avec une compression H.264 ou MPEG-4.

Pour le streaming vidéo depuis un site Internet, référez-vous à la fiche *Technical Note TN2224 Best Practices for Creating and Deploying HTTP Live Streaming Media for the iPhone and iPad* que vous trouverez dans la documentation.

CRÉATION DU PROJET

Commencez par créer un nouveau projet de type Window-based Application que vous nommerez MoviePlayer. Ajoutez ensuite un RootViewController (fichier de type UIViewController ; cochez With XIB for User Interface). Puis, ajoutez sa vue dans la fenêtre, dans l'application delegate :

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

@class RootViewController;

@interface MoviePlayerAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    RootViewController *rootView;
}

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

@end

MoviePlayerAppDelegate.m
#import "MoviePlayerAppDelegate.h"
#import "RootViewController.h"

@implementation MoviePlayerAppDelegate
// ...
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    rootView = [[RootViewController alloc] initWithNibName:@"RootViewController"
        bundle:nil];
```

```

[rootView.view setFrame:[[UIScreen mainScreen] applicationFrame]];
[window addSubview:rootView.view];
>window makeKeyAndVisible];

    return YES;
}

// ...
- (void)dealloc {
    [window release];
    [rootView release];
    [super dealloc];
}

```

Pour utiliser la vidéo, il faut ajouter le framework MediaPlayer comme suit :

1. Cliquez du bouton droit sur le dossier frameworks de votre application.
2. Cliquez sur Add > Existing Frameworks...
3. Choisissez MediaPlayer.framework.

Nous allons construire l'interface graphique ici pour nous concentrer ensuite sur la lecture de la vidéo. Dans RootViewController.h, importez le framework MediaPlayer, déclarez un contrôleur de vue de vidéo (MPMoviePlayerViewController), une vue pour l'overlay, un interrupteur pour afficher ou non la vue d'overlay et un contrôleur de vidéo pour faire apparaître la petite vidéo :

```

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

@interface RootViewController : UIViewController {

    MPMoviePlayerViewController *mPlayer;
    UIView *overlayView;
    UISwitch *switchOverlay;

    MPMoviePlayerController *movieControllerSmall;
}

- (void) playVideoAtUrl:(NSURL*)movieURL;
- (void) presentMovieEmbedded;
- (void) createOverlayView;

@end

```

Affichons les deux boutons, le label et l'interrupteur :

```

- (id)initWithNibName:(NSString *)NibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
    if ((self = [super initWithNibName:nibNameOrNilOrNil bundle:nibBundleOrNil])) {
        UIButton *localButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
        [localButton addTarget:self action:@selector(playVideoLocally:)
        ↳ forControlEvents:UIControlEventTouchUpInside];
    }
}

```

```
[localButton setTitle:@"Local" forState:UIControlStateNormal];
[localButton setFrame:CGRectMake(30, 30, 115, 30)];
[self.view addSubview:localButton];

UIButton *webButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[webButton addTarget:self action:@selector(playVideoFromWeb:)
forControlEvents:UIControlEventTouchUpInside];
[webButton setTitle:@"Web" forState:UIControlStateNormal];
[webButton setFrame:CGRectMake(175, 30, 115, 30)];
[self.view addSubview:webButton];

UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(30, 90, 260, 30)];
label.textAlignment = NSTextAlignmentCenter;
label.text = @"Afficher vos propres contrôles ?";
[self.view addSubview:label];
[label release];

UISwitch *switchOverlay = [[UISwitch alloc] init];
switchOverlay.center = CGPointMake(160, 140);
[self.view addSubview:switchOverlay];
}

return self;
}
```

Vous devriez obtenir une interface comme à la Figure 18.1.



Figure 18.1 : L'interface à obtenir.

JOUER UNE VIDÉO DEPUIS LE WEB

Pour jouer la vidéo depuis une URL, implémentez tout d'abord la méthode suivante (nous ne travaillerons que dans la classe RootViewController) :

```
- (void) playVideoFromWeb :(id)sender {
    [self playVideoAtUrl:[NSURL URLWithString:@"http://www.ipup.fr/
    ➔ livre/Lucky_man_pres.mov"]];
}
```

Ensuite, nous allons définir la méthode playVideoAtUrl::

```
- (void) playVideoAtUrl:(NSURL*)movieURL {
    if(!mPlayer)
        mPlayer = [[MPMoviePlayerViewController alloc] init];
    mPlayer.moviePlayer.contentURL = movieURL;
    // on met la vidéo en plein écran
    mPlayer.moviePlayer.fullscreen = YES;
    // on affiche la vidéo
    [self presentMoviePlayerViewControllerAnimated:mPlayer];
    // on joue
    [mPlayer.moviePlayer play];
}
```

On n'alloue le lecteur qu'une fois au premier appel de la méthode, pour optimiser les performances. Utiliser la méthode presentMoviePlayerViewControllerAnimated: permet d'afficher la vidéo avec animation. De plus, lorsque la vidéo est terminée, ou lorsqu'on clique sur le bouton Terminer, la vue est automatiquement enlevée. Cependant, rien ne vous empêche d'afficher la vue de mPlayer comme vous le souhaitez. Il faudra vous abonner aux notifications de fin de lecture de la vidéo pour enlever la vue : MPMoviePlayerPlaybackDidFinishNotification.

Info

Si vous souhaitez optimiser les performances (surtout si la vidéo est longue), téléchargez la vidéo dans un premier temps, puis jouez là une fois que le téléchargement est terminé.

JOUER UNE VIDÉO EN LOCAL

Pour jouer une vidéo en local, rien de plus simple. Il faut tout d'abord copier la vidéo dans votre projet, puis implémenter cette méthode :

```
- (void) playVideoLocally :(id)sender {
    NSString *path = [[NSBundle mainBundle] pathForResource:@"Lucky_man_pres"
    ➔ ofType:@"mov"];
    if (path) {
        [self playVideoAtUrl:[NSURL URLWithString:path]];
    }
}
```

Vous pouvez récupérer la vidéo en ligne depuis votre navigateur en tapant l'adresse employée pour jouer la vidéo depuis le Web.

AJOUTER UNE VUE PAR-DESSUS

Vous pouvez définir une vue à ajouter par-dessus votre vidéo, pour afficher certaines informations, ou pour présenter vos propres contrôles. Nous allons donc ajouter ici une vue si l'interrupteur est sélectionné. Cette vue contiendra un bouton qui permettra de jouer/mettre en pause la vidéo. Sinon, nous laisserons les contrôles par défaut. Ajoutez les lignes en gras :

```
- (void) playVideoAtUrl:(NSURL*)movieURL {
    if(!mPlayer)
        mPlayer = [[MPMoviePlayerViewController alloc] init];
    mPlayer.moviePlayer.contentURL = movieURL;
    // on met la vidéo en plein écran
    mPlayer.moviePlayer.fullscreen = YES;

    if (switchOverlay.on)
    {
        // on ajoute la vue d'overlay
        [self createOverlayView];
        if ([overlayView superview] == nil) {
            [mPlayer.moviePlayer.view addSubview:overlayView];
        }
        // on enlève les controls pour mettre les nôtres
        mPlayer.moviePlayer.controlStyle = MPMovieControlStyleNone;
    }
    else
    {
        // on l'enlève
        [overlayView removeFromSuperview];
        // on mets les contrôles par défaut en mode plein écran
        mPlayer.moviePlayer.controlStyle = MPMovieControlStyleFullscreen;
    }

    // on affiche la vidéo
    [self presentMoviePlayerViewControllerAnimated:mPlayer];
    // on joue
    [mPlayer.moviePlayer play];
}
```

Puis définissez la méthode permettant de créer la vue. Là encore, nous ne créons la vue que si elle n'existe pas :

```
- (void) createOverlayView {
    // si elle existe déjà, on quitte
    if (overlayView) {
        return;
}
```

```

// sinon, on la crée
overlayView = [[UIView alloc] initWithFrame:self.view.frame];
// on peut définir une opacité
overlayView.alpha = 0.8;

UIButton *pauseButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[pauseButton addTarget:self action:@selector(pauseResumeVideo:)
    forControlEvents:UIControlEventTouchUpInside];
[pauseButton setTitle:@"Pause" forState:UIControlStateNormal];
[pauseButton setTitle:@"Resume" forState:UIControlStateSelected];
[pauseButton setFrame:CGRectMake(175, 30, 115, 30)];
[overlayView addSubview:pauseButton];
}

```

Enfin, il faut implémenter la méthode pour jouer/mettre en pause avec le bouton :

```

- (void) pauseResumeVideo :(id)sender {
    UIButton *but = (UIButton*)sender;

    if (but.selected) {
        [mPlayer.moviePlayer play];
    }
    else {
        [mPlayer.moviePlayer pause];
    }
    // on change l'état du bouton
    but.selected = !but.selected;
}

```

Vous pouvez lancer la vidéo, en choisissant d'afficher ou non la vue d'overlay !

JOUER UNE VIDÉO DANS UN COIN DE L'ÉCRAN

Il est possible de jouer une vidéo sans obligatoirement la présenter en plein écran depuis iOS3.2. Pour cela, il faut créer une vue qui servira de conteneur pour l'objet permettant de jouer la vidéo (`MPMoviePlayerController`).

Info

Il n'est pas conseillé d'avoir plusieurs lecteurs vidéo dans une même vue. Nous le faisons ici pour le besoin de cette fiche et ne pas avoir à recréer un nouveau projet.

La vidéo apparaîtra au lancement de l'application, afin de la jouer en boucle. Dans ce but, ajouter à la fin du `initWithNibName:bundle:` :

```
[self presentMovieEmbedded];
```

Ensuite, implémentez-la :

```
- (void) presentMovieEmbedded {
    // la vue qui va contenir notre lecteur
    UIView *myView = [[UIView alloc] initWithFrame:CGRectMake(30, 200, 260, 200)];

    // initialisation du contrôleur
    movieControllerSmall = [[MPMoviePlayerController alloc] initWithContentURL:[NSURL
        URLWithString:@"http://www.ipup.fr/livre/Lucky man pres.mov"]];
    // on définit sa taille
    movieControllerSmall.view.frame = CGRectMake(0, 0, 260, 200);
    // on définit la couleur du fond
    movieControllerSmall.backgroundView.backgroundColor = [UIColor whiteColor];
    // on définit une répétition de la lecture lorsque la vidéo est finie
    movieControllerSmall.repeatMode = MPMovieRepeatModeOne;
    // on ajoute le player
    [myView addSubview:movieControllerSmall.view];
    // on ajoute la vue

    [self.view addSubview:myView];
    // on lance la lecture
    [movieControllerSmall play];

    [myView release];
}
```

La vidéo va s'adapter à la taille de la vue qui sert de conteneur. De plus, lorsque vous cliquerez sur le bouton pour bénéficier du plein écran, la vidéo s'affichera automatiquement dans tout l'écran disponible. Là aussi, vous pouvez vous abonner à certaines notifications pour réagir à l'événement : "Met la vue en plein écran". Ainsi vous pourrez arrêter certaines actions dans la vue comme une animation.

N'oubliez pas le dealloc :

```
- (void) dealloc {
    [movieControllerSmall release];
    [mPlayer release];
    [overlayView release];
    [switchOverlay release];

    [super dealloc];
}
```

Depuis la sortie de iOS3, il est possible d'envoyer des notifications Push à un iPhone, un iPad ou un iPod Touch qui dispose d'une connexion Internet. Pour l'utilisateur, une notification se matérialise la plupart du temps comme un SMS reçu de la part d'une application installée sur son mobile. L'intérêt principal est d'informer l'utilisateur d'événements, même si l'application n'est pas lancée à ce moment-là. Typiquement une notification peut avertir d'une nouvelle importante comme une alerte météo ou un but de l'équipe de France de football.

Nous allons voir étape par étape comment réaliser ce type d'application.

ENREGISTREMENT DANS LE PORTAIL DES PROVISIONINGS

Il y a beaucoup de choses à faire sur le portail des provisioningings d'Apple afin d'exploiter le Push dans votre application. Il faut commencer par créer un nouvel identifiant d'application :

Provisioning portal > app IDs > New App ID.

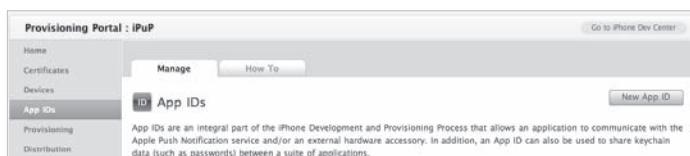


Figure 19.1 : Créez un nouvel App ID.

Il suffit de remplir les champs proposés. Par contre, et c'est important, il faut utiliser un Bundle Identifier explicite c'est-à-dire sans *.

Figure 19.2 : Décrivez l'App ID.

On aboutit à la Figure 19.3.

| | | | | |
|-----------------------------|---------------------------------------------------------------|--------------------------------------------------------------|------------------------------------|------------------------------------|
| 92V5A322JR.fr.ipup.tutopush | <input checked="" type="radio"/> Configurable for Development | <input checked="" type="radio"/> Configurable for Production | <input type="radio"/> Configurable | <input type="checkbox"/> Configure |
|-----------------------------|---------------------------------------------------------------|--------------------------------------------------------------|------------------------------------|------------------------------------|

Figure 19.3 : Votre nouvel App ID !

Passons maintenant à la configuration. Il faut donc cocher la case Enable for Apple Push Notification Service. Ensuite, deux certificats sont à configurer. Le premier est un certificat de développement qui va nous permettre de tester notre application sur un iPhone pendant le développement de notre application. Le second est un certificat de Production qui enverra des notifications à des applications téléchargées à partir de l'App Store.

Nous allons configurer les deux ! Pour le premier :

Info

À ce stade, je vous conseille de créer un dossier tuto_push car nous allons créer plusieurs fichiers.

1. Configurez le certificat de développement en cliquant sur **Configure**.
2. En parallèle, lancez le trousseau d'accès (utilisez le Finder par exemple) et allez dans Trousseau d'accès > Assistant de certification > Demander un certificat à une autorité de certificat.

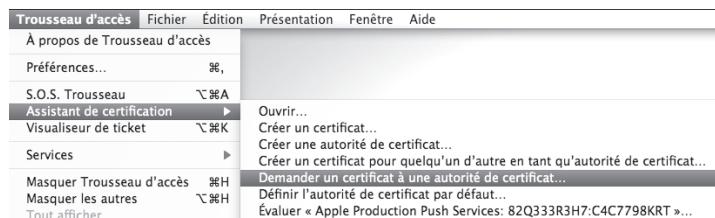


Figure 19.4 :
Le chemin pour demander le certificat.

3. Remplissez les champs demandés (sauf Adresse électronique de l'AC qui n'est pas nécessaire) et choisissez Enregistrer sur le disque.
4. Enregistrez la demande dans le dossier **tuto_push** sous le nom : **CertificateSigningRequestDev.certSigningRequest**.

C'est terminé pour le trousseau d'accès. Retournez sur le portail des provisioningings d'Apple puis :

1. Cliquez sur Continue.
2. Choisissez le fichier que l'on vient de créer.
3. Cliquez sur Generate puis choisissez Continue.
4. Enfin, il reste à télécharger le certificat créé et le mettre dans le dossier **tuto_push** pour ne pas le perdre.
5. Procédez de la même manière avec le Certificat de Production en créant un **CertificateSigningRequestProd.certSigningRequest** grâce au trousseau d'accès.

À ce stade vous devriez avoir quatre fichiers dans le dossier **tuto_push** :

- **aps_developer_identity.cer**
- **aps_production_identity.cer**
- **CertificateSigningRequestDev.certSigningRequest**
- **CertificateSigningRequestProd.certSigningRequest**

Passons maintenant à la création des provisioningings.

1. Dans la rubrique Provisioning, onglet Development, cliquez sur New Profile.
2. Renseignez le champ Profile Name avec Tuto Push et cochez le certificat adéquat.
3. Sélectionnez l'App ID tuto push, puis les appareils sur lesquels vous testerez votre application. Il reste à soumettre le provisioning, le télécharger et le ranger dans le dossier tuto_push.
4. Il faut recommencer l'opération dans l'onglet Distribution pour créer le provisioning vous permettant de compiler votre application pour sa distribution sur l'App Store.

Vous avez maintenant six fichiers dans le dossier tuto_push et toutes les opérations demandées par Apple pour les certificats sont réalisées. Nous allons voir comment exploiter tous ces fichiers.

PRÉPARATION DES CERTIFICATS POUR LE SERVEUR

À ce stade, les possibilités sont nombreuses, et nous n'allons pas toutes les détailler. Par contre, vous apprendrez à envoyer des push à partir d'un serveur quelconque et pour cela, nous allons devoir réaliser des opérations sur les certificats créés précédemment.

Tout d'abord, double-cliquez sur le fichier aps_developer_identity.cer puis retournez dans le trousseau d'accès rubrique Session et dans Certificats. Vous devriez trouver un nouveau certificat installé de type Apple Production Push Service.

En double cliquant dessus, vous devriez retrouver les informations présentées Figure 19.5.



Figure 19.5 : Votre certificat Apple Production Push Service.

Il faut alors sélectionner le Certificat **et** la clé puis cliquer du bouton droit, choisir Exporter 2 éléments et enregistrer dans tuto_push sous le nom CertificatsDev.p12.

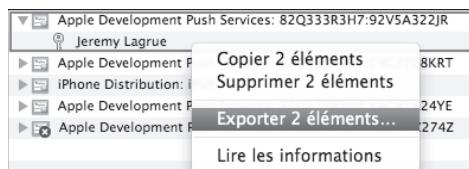


Figure 19.6 : La liste de choix après avoir cliqué du bouton droit.

Il vous est alors demandé un premier mot de passe qui sert à protéger votre clé p12. Puis, on vous demande votre mot de passe de session. Faites bien attention de ne pas vous tromper et souvenez-vous impérativement de ces mots de passe.

Répétez l'opération avec le certificat de Production en nommant l'export CertificatsProd.p12.

Le dossier tuto_push compte maintenant huit fichiers :

- aps_developer_identity.cer
- aps_production_identity.cer
- CertificateSigningRequestDev.certSigningRequest
- CertificateSigningRequestProd.certSigningRequest
- Tuto_Push.mobileprovision
- Tuto_Push_Prod.mobileprovision
- CertificatsDev.p12
- CertificatsProd.p12

Nous devons maintenant convertir les fichiers .p12 en .pem pour les utiliser avec le code PHP que nous allons bientôt faire.

Commencez par créer un petit script Shell qui va simplifier les opérations. Créez un nouveau fichier convert.sh dans le dossier tuto_push avec votre éditeur de texte préféré et ajoutez-y ces lignes de commandes :

```
#traitement du certificat de développement
openssl pkcs12 -clcerts -nokeys -out CertificatDev.pem -in CertificatsDev.p12
openssl      pkcs12 -nocerts -out KeyDev.pem -in CertificatsDev.p12
cat CertificatDev.pem KeyDev.pem > apns-dev.pem

#traitement du certificat de Production
openssl pkcs12 -clcerts -nokeys -out CertificatProd.pem -in CertificatsProd.p12
openssl      pkcs12 -nocerts -out KeyProd.pem -in CertificatsProd.p12
cat CertificatProd.pem KeyProd.pem > apns-prod.pem

#vérification
openssl s_client -connect gateway.sandbox.push.apple.com:2195 -cert apns-dev.pem
openssl s_client -connect gateway.push.apple.com:2195 -cert apns-prod.pem
```

Nous sommes à présent prêts à convertir nos .p12 en .pem. Pour cela nous allons lancer le terminal par le Finder. Le terminal est rangé dans le dossier Utilitaires de vos applications.

Utilisez la commande cd pour vous positionner dans le dossier tuto_push. Si le dossier tuto_push est sur le bureau : cd Desktop/tuto_push.

Info

Les commandes basiques nécessaires pour naviguer dans la hiérarchie de vos fichiers sont rappelées à la Fiche 31.

Il vous reste à lancer le script en écrivant dans la fenêtre du terminal : sh convert.sh

Le script va se lancer et vous poser deux types de question :

- **Enter Import Password.** Il faut donc entrer votre mot de passe d'export de clé p12 puis valider (touche Enter).
- **Enter PEM pass phrase.** On vous demande un mot de passe qui protégera votre certificat pem.

Une fois ces opérations effectuées, votre dossier tuto_push doit contenir quinze fichiers dont les précieux sésames que sont apns-dev.pem et apns-prod.pem.

Beaucoup de ressources sur Internet proposent d'enlever la clé des certificats pem grâce aux commandes suivantes :

```
#traitement du certificat de développement
openssl pkcs12 -clcerts -nokeys -out CertificatDev.pem -in CertificatsDev.p12
openssl pkcs12 -nocerts -out KeyDev.pem -in CertificatsDev.p12
openssl rsa -in KeyDev.pem -out NoKeyDev.pem
cat CertificatDev.pem NoKeyDev.pem > apns-dev.pem

#traitement du certificat de Production
openssl pkcs12 -clcerts -nokeys -out CertificatProd.pem -in CertificatsProd.p12
openssl pkcs12 -nocerts -out KeyProd.pem -in CertificatsProd.p12
openssl rsa -in KeyProd.pem -out NoKeyProd.pem
cat CertificatProd.pem NoKeyProd.pem > apns-prod.pem

#vérification
openssl s_client -connect gateway.sandbox.push.apple.com:2195 -cert apns-dev.pem
openssl s_client -connect gateway.push.apple.com:2195 -cert apns-prod.pem
```

La sécurité est plus faible : il suffirait de dérober le fichier pem pour envoyer des push à vos utilisateurs. Toutefois, omettre cette sécurité simplifie légèrement la programmation serveur. Nous allons donc continuer avec nos certificats pem protégés par notre pass phrase. Nous pouvons maintenant faire un peu de programmation, enfin !

APPLICATION POUR L'IPHONE

Pour commencer, créez un nouveau projet de type Window-based Application et nommez-le Tuto_push.

Dans Tuto_pushAppDelegate.m nous allons enregistrer l'application aux notifications push de la manière suivante :

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    //enregistrement aux notifications Push
    [application registerForRemoteNotificationTypes: UIRemoteNotificationTypeAlert
    | UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeSound];
    [window makeKeyAndVisible];
    return YES;
}
```

De plus, il y a deux méthodes à implémenter qui vont permettre de récupérer le Token ou l'erreur.

```
- (void)application:(UIApplication *)application
    didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken {
    NSLog(@"enregistrement réussi") ;
    NSLog(@"%@", deviceToken);
}

-(void)application:(UIApplication *)application
didFailToRegisterForRemoteNotificationsWithError:(NSError *)error{
    NSLog(@"échec de l'enregistrement") ;
}
```

Il reste maintenant à installer et sélectionner le provisioning de développement pour compiler notre application correctement. Pour ce faire :

1. Dans le dossier tuto_push, double-cliquez sur Tuto_Push.mobileprovision.
2. Puis, dans Xcode, choisissez la configuration Debug sur Device.
3. Ensuite, allez dans Project > Edit Project Settings et remplissez le champ Code Signing Identity avec le provisioning de développement pour le Bundle identifier tutopush (voir Figure 19.7).



Figure 19.7 : Choisissez le bon provisioning pour la compilation.

4. Enfin, n'oubliez pas de remplir correctement le Bundle identifier dans le .plist.

Ces opérations finies, vous pouvez compiler et lancer l'application sur votre iPhone (cela ne fonctionne pas sur simulateur). Dans la console, vous devriez voir apparaître votre token sous la forme :

```
Tuto_push[6914:307] enregistrement réussi
Tuto_push[6914:307] <2b555730 5a83a7cf c216b51a 0df488ec 6bc61168 0ac2078d
    ↳ 5ca8111b 28f428c7>
```

Gardez le message de la console de côté, nous allons bientôt en avoir besoin. L'essentiel de la programmation côté iPhone est maintenant terminé, passons au serveur !

PROGRAMMATION DU SERVEUR

Nous allons écrire du code en PHP afin d'envoyer un push à l'application que nous venons de terminer. Pour cela, votre serveur doit être capable d'utiliser les sockets en PHP (voir votre php.ini) et surtout avoir le port 2195 ouvert (à voir avec votre hébergeur en lisant la documentation mise à disposition ou en le contactant directement).

Comme tout le monde ne dispose pas d'un serveur dédié, et qu'un simple hébergement n'est pas suffisant, nous allons recourir à un serveur local sur Mac. Deux cas de figure : soit vous avez déjà fait du PHP sur Mac et il n'y a pas de problème, soit vous ne maîtrisez pas PHP et je vous conseille d'installer MAMP et de procéder à quelques tests de script PHP.

Nous allons créer un nouveau dossier php_push pour y placer notre code PHP. Pour moi, ce dossier est accessible à l'adresse : http://localhost:8888/php_push/

Dans ce dossier, nous avons besoin de trois fichiers :

- Un fichier avec notre code php (pour moi sample.php).
- Une copie du fichier apns-dev.pem se trouvant dans tuto_push et renommé tutopush-dev.pem.
- Une copie du fichier apns-prod.pem se trouvant dans tuto_push et renommé tutopush-prod.pem.

Passons au code PHP :

Tout d'abord, avec la class push fournie, la seule modification à apporter de votre part est le changement de passphrase par votre mot de passe pem.

```
<?php
class Push
{
    //serveur de développement
    private $apnsHostDev = 'gateway.sandbox.push.apple.com';

    //serveur de Production
    private $apnsHostProd = 'gateway.push.apple.com';

    //port de connection
    private $apnsPort = '2195';

    //mot de passe de de votre pem
    private $passPhrase = 'passphrase';

    //on travaille en local
    private $serviceHost = 'localhost';

    private $apnsConnection;

    public function __construct ($cert, $mode) {
        //connexion chez apple
        $streamContext = stream_context_create();
        //on donne l'adresse du .pem ici cela donne tutopush-dev.pem
        stream_context_set_option($streamContext, 'ssl', 'local_cert',
        ➔ $cert.'-'.$mode.'.pem');
        //on renseigne le mot de passe
        stream_context_set_option($streamContext, 'ssl', 'passphrase',
        ➔ $this->passPhrase);

        // en mode dev on utilise le serveur de développement
        if($mode == 'dev')
            $this->apnsConnection =
                ➔ stream_socket_client('ssl://'.$this->apnsHostDev.':'.$this->apnsPort,
                ➔ $error, $errorString, 60, STREAM_CLIENT_CONNECT, $streamContext);

        // en mod prod on utilise le serveur de production
```

```

if($mode == 'prod')
    $this->apnsConnection =
        ➔ stream_socket_client('ssl://'.$this->apnsHostProd.':'.$this->apnsPort,
        ➔ $error, $errorString, 60, STREAM_CLIENT_CONNECT, $streamContext);
}

public function __destruct () {
    //fermeture de la connexion
    fclose($this->apnsConnection);
}

public function sendMessageToToken ($token, $message)
{
    //envoi du push
    // création du message en JSON à partir du tableau associatif
    $message = json_encode($message);
    //formatage du message
    $apnsMessage = chr(0) . chr(0) . chr(32) . pack('H*', str_replace(' ', '', 
        ➔ $token)) . chr(0) . chr(strlen($message)) . $message;
    //écriture dans la socket
    fwrite($this->apnsConnection, $apnsMessage);
}
}
?>

```

Cette classe se décompose en trois parties principales :

- Ouverture de la connexion socket vers le service de push d'Apple.
- Écriture d'un message à un appareil (on peut envoyer plusieurs messages à travers une seule connexion socket pour plus d'efficacité).
- Fermeture de la connexion socket.

À présent, regardons comment exploiter cette classe à travers un exemple :

```

<?php
// copiez ici le token récupéré dans la console et respectant le format proposé
$token      = '2b555730 5a83a7cf c216b51a 0df488ec 6bc61168 0ac2078d 5ca811b 28f428c7';

//création d'une instance push pour l'application tutopush en développement
$push = new Push('tutopush', 'dev');
//En production on utiliserait
//    $Push = new SendPush('tutopush', 'prod');

//tableau associatif contenant le message Push, le son et le badge
$payload['aps'] = array('alert' => 'Le message à passer', 'sound' => 'default',
    ➔ 'badge'=> 100);

//envoi du message

```

```

$push->sendMessageToToken ($token, $payload);
// c'est ici que l'on pourrait ajouter des destinataires
// $push->sendMessageToToken ($token2, $payload);
// $push->sendMessageToToken ($token2, $payload);

// fermeture de la socket
unset($push);
?>

```

Une fois tout ce code ajouté à votre fichier sample.php, il vous reste à l'exécuter *via* votre navigateur (http://localhost:8888/php_push/sample.php) ou *via* votre terminal et vous devriez recevoir un Push !



Figure 19.8 : La réception de votre premier push !

Même si vous avez reçu votre fameux Push, ce n'est qu'un début. En effet, pour réaliser un service complet, vous devez continuer l'implémentation de la méthode `didRegisterForRemoteNotificationsWithDeviceToken`: afin d'y inclure une requête vers votre serveur (ou votre ordinateur en WiFi par exemple) pour lui envoyer votre token.

Mais, et vous vous en êtes peut-être rendu compte, le vrai défi avec le push est côté serveur et ce n'est pas forcément ni de votre ressort, ni de celui de ce livre de réaliser ce service complet. C'est la raison pour laquelle je vous conseille d'utiliser une des solutions suivantes, qui vous feront gagner du temps :

- Tout d'abord, easyapns (<http://www.easyapns.com>) qui fournit une solution assez complète de gestion de push en PHP.
- Il existe également l'équivalent en java (<http://github.com/notnoop/java-apns>).
- Ou encore, et c'est peut-être le plus simple : appnotify (<http://appnotify.com>).

L'iPhone possède un capteur pour prendre des photos. Dans iOS, il est possible de travailler avec des photos, que ce soit directement à partir de l'appareil photo numérique ou à partir de l'album photo. Cet album est disponible pour l'utilisation dans une application.

Dans cette fiche, nous allons apprendre à manipuler l'appareil photo, choisir une photo, l'afficher dans une application, dessiner par-dessus et sauvegarder le résultat. Pour ajouter un peu de piment, nous effacerons le dessin en secouant l'appareil !

Pas d'inquiétude, nous procéderons par étapes !

Nous obtiendrons le résultat comme à la Figure 20.1. La Figure 20.2 montre l'image telle que sauvegardée ensuite.

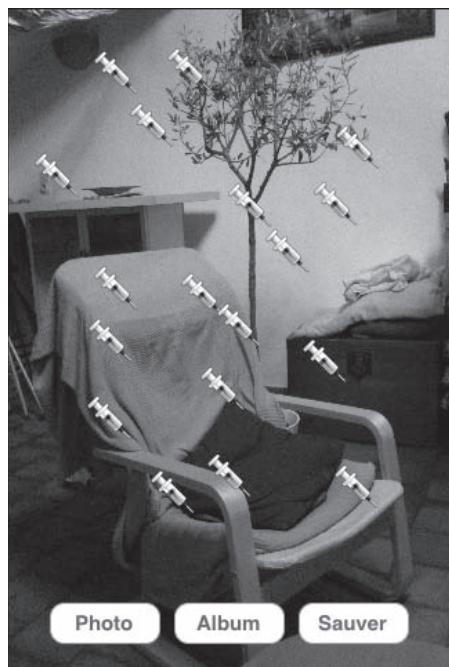


Figure 20.1 : Une capture écran de l'application.

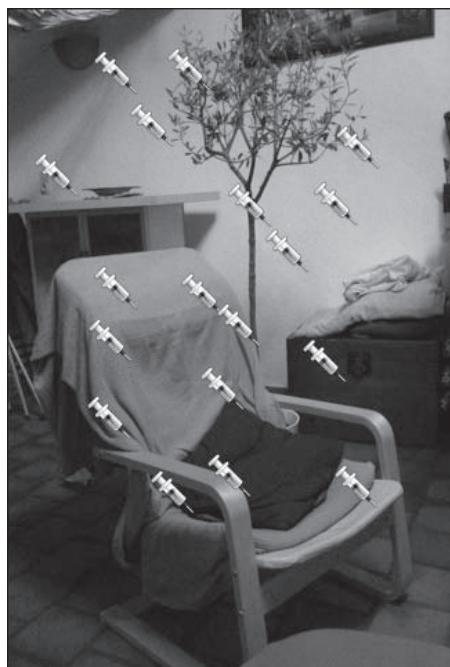


Figure 20.2 : L'image obtenue à la fin.

CRÉER UN PROJET ET AFFICHER LES IMAGES

Pour commencer, créez un nouveau projet de type View-based Application que vous nommerez PlayWithPhoto.

Le but de l'application étant également de faire une capture écran, elle sera plus jolie si nous cachons la barre de statut (où s'affiche le niveau de la batterie, le nom de l'opérateur...). Afin de la cacher, il faut éditer le fichier PlayWithPhoto-Info.plist et ajouter une clé : Status bar is initially hidden puis cocher la case qui apparaît.

On affiche l'appareil photo ou l'album photo grâce à un objet de la classe UIImagePickerController.

Après avoir choisi la photo, il faudra l'afficher. Cet affichage se fera en plein écran, grâce un objet de la classe UIImageView. Déclarez dans PlayWithPhotoViewController.h :

```
UIImageView *imageViewBackgroundPhoto;
```

Puis, dans le viewDidLoad, allouez cette image view :

```
// la vue qui affichera l'image en fond  
imageViewBackgroundPhoto = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0,  
➥ 320, 480)];  
// on ne met pas d'image ici  
[self.view addSubview:imageViewBackgroundPhoto];
```

La vue pour prendre la photo s'affichera en cliquant sur un bouton. Créez donc ce bouton dans le viewDidLoad du PlayWithPhotoViewController.m :

```
// bouton pour prendre la photo  
UIButton *buttonTakePhotoFromCamera =  
➥ [UIButton buttonWithType:UIButtonTypeRoundedRect];  
[buttonTakePhotoFromCamera setFrame:CGRectMake(30, 430, 80, 30)];  
[buttonTakePhotoFromCamera addTarget:self action:@selector(showCameraPicker:)  
➥ forControlEvents:UIControlEventTouchUpInside];  
[buttonTakePhotoFromCamera setTitle:@"Photo" forState:UIControlStateNormal];  
[self.view addSubview:buttonTakePhotoFromCamera];
```

Puis, écrivez la méthode appelée par ce bouton :

```
#pragma mark -  
#pragma mark action methods  
  
- (void) showCameraPicker:(id)sender {  
  
    UIImagePickerController* picker = [[UIImagePickerController alloc] init];  
    picker.sourceType = UIImagePickerControllerSourceTypeCamera; ①  
    picker.delegate = self; ②  
    picker.allowsEditing = NO; ③  
  
    //on affiche le picker  
    [self presentViewController:picker animated:YES];  
}
```

Ligne ①, on sélectionne le type de données à afficher grâce à l'objet picker de la classe UIImagePickerController. Ici, le but étant de permettre à l'utilisateur de prendre une photo, on spécifie UIImagePickerControllerSourceTypeCamera.

Ligne ②, self est délégué. Il faut ajouter dans le .h :

```
@interface PlayWithPhotoViewController : UIViewController  
↳ <UINavigationControllerDelegate, UIImagePickerControllerDelegate>
```

Ligne ③, on peut choisir de laisser l'utilisateur modifier l'image avant de s'en servir dans l'application. Ici, on ne l'autorise pas. Figure 20.3, vous trouverez la vue affichée à l'écran lorsque l'on autorise l'utilisateur à modifier l'image en mettant YES.



Figure 20.3 : Une vue pour l'édition.

Après avoir affichée la vue modale pour prendre la photo, on peut soit annuler, soit choisir l'image. Pour le premier cas, il faut écrire :

```
#pragma mark -  
#pragma mark image picker delegate  
  
- (void) imagePickerControllerDidCancel:(UIImagePickerController *)picker {  
    // on enlève la vue  
    [picker dismissModalViewControllerAnimated:YES];  
    // on release le picker  
    [picker release];  
}
```

Lorsque l'utilisateur choisit l'image, il faudra l'afficher dans l'application :

```
- (void) imagePickerController:(UIImagePickerController *)picker  
didFinishPickingImage:(UIImage *)image editingInfo:(NSDictionary *)editingInfo {  
  
    imageViewBackgroundPhoto.image = image;  
  
    // on enlève la vue  
    [picker dismissModalViewControllerAnimated:YES];  
    // on release le picker  
    [picker release];  
}
```

Info

Dans le cas où vous permettez à l'utilisateur de modifier l'image, le paramètre `image` contiendra l'image modifiée, `editingInfo` contiendra lui l'image originale et le rectangle qui délimite l'image modifiée dans l'image originale.

Attention

Vous devez tester si l'appareil peut prendre des photos, ou afficher l'album. Pour cela, nous choisissons ici d'afficher ou non le bouton correspondant à l'action disponible ou non. Ajoutez dans le `viewDidLoad` :

```
// si on ne peut pas prendre de photo, on cache le bouton  
if (!UIImagePickerController isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera)  
    buttonTakePhotoFromCamera.hidden = YES;
```

Pour afficher l'album photo, créez le bouton dans le `viewDidLoad` :

```
// bouton pour prendre une photo depuis l'album  
UIButton *buttonTakePhotoFromAlbum = [UIButton buttonWithType:UIButtonTypeRoundedRect];  
[buttonTakePhotoFromAlbum setFrame:CGRectMake(120, 430, 80, 30)];  
[buttonTakePhotoFromAlbum addTarget:self action:@selector(showPhotoAlbumPicker:)  
↳ forControlEvents:UIControlEventTouchUpInside];  
[buttonTakePhotoFromAlbum setTitle:@"Album" forState:UIControlStateNormal];  
[self.view addSubview:buttonTakePhotoFromAlbum];  
  
// si on ne peut pas accéder à l'album photo, on cache le bouton  
if (!UIImagePickerController  
isSourceTypeAvailable:UIImagePickerControllerSourceTypeSavedPhotosAlbum])  
    buttonTakePhotoFromAlbum.hidden = YES;
```

La méthode showPhotoAlbumPicker: est la suivante :

```
- (void) showPhotoAlbumPicker:(id)sender {
    UIImagePickerController* picker = [[UIImagePickerController alloc] init];
    picker.sourceType = UIImagePickerControllerSourceTypeSavedPhotosAlbum;
    picker.delegate = self;
    picker.allowsEditing = NO;

    //on affiche le picker
    [self presentModalViewController:picker animated:YES];
}
```

DESSINER PAR-DESSUS L'IMAGE

Dans cette partie, laissons libre court à l'imagination de l'utilisateur en lui permettant de toucher la photo pour y déposer une image. Cette image, nommée ici shot.png a été trouvée sur le site <http://www.iconfinder.com>. Elle fait 32 × 32 pixels, n'oubliez pas d'adapter la taille de funnyImage.

L'image sera chargée une seule fois lors du premier affichage de la vue. Pour cela, déclarez dans le .h :

```
UIImage *funnyImage;
```

Puis, dans le viewDidLoad :

```
// trouvée sur http://www.iconfinder.com Auteur Everaldo Coelho
funnyImage = [[UIImage imageNamed:@"shot.png"] retain];
```

Il est très facile de détecter un appui de l'utilisateur. Il suffit d'implémenter les méthodes :

```
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    // appelé lorsque l'utilisateur pose son doigt
}

- (void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    // appelé tout au long du déplacement du doigt sur l'écran
}

- (void) touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    // appelé lorsque l'utilisateur relève le doigt
}
```

Ces méthodes permettent de détecter le multitouch :

```
int nbOfFingers = [touches count]; // retourne le nombre de doigt sur l'écran
```

et également de compter le nombre de taps consécutifs (lorsqu'ils sont rapides) :

```
touch.tapCount;
```

Nous nous préoccuperons ici de la méthode appelée lorsque l'utilisateur pose son doigt, et afficherons une nouvelle image :

```
#pragma mark -  
#pragma mark Gestion du touch  
  
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {  
    UITouch *touch = [touches anyObject];  
  
    // on récupère les coordonnées dans la vue  
    CGPoint touchCoordinates = [touch locationInView:self.view];  
    // on crée une nouvelle image view  
    UIImageView *shot = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0, 32, 32)];  
    shot.image = funnyImage;  
    // on centre par rapport au doigt  
    shot.center = touchCoordinates;  
    [self.view addSubview:shot];  
    [shot release];  
}
```

SAUVEGARDER L'IMAGE

Après s'être amusé en ajoutant beaucoup de seringues (dans mon exemple) sur la photo, vous pouvez proposer à l'utilisateur de la sauvegarder dans son album. Commencez par créer le bouton qui réalisera l'action en ajoutant dans le viewDidLoad :

```
// bouton pour sauver la photo  
UIButton *buttonSavePhoto = [UIButton buttonWithType:UIButtonTypeRoundedRect];  
[buttonSavePhoto setFrame:CGRectMake(210, 430, 80, 30)];  
[buttonSavePhoto addTarget:self action:@selector(savePhoto:)  
↳ forControlEvents:UIControlEventTouchUpInside];  
[buttonSavePhoto setTitle:@"Sauver" forState:UIControlStateNormal];  
[self.view addSubview:buttonSavePhoto];
```

Ensuite, nous allons nous occuper de la méthode appelée lors de l'appui sur le bouton :

```
- (void) savePhoto:(id)sender {  
}
```

Prendre une capture de l'écran nécessite d'abord de cacher les boutons. Pour cela, on énumère tous les objets de la classe `UIView` dans les sous-vues de la vue du contrôleur (on énumère toutes ses sous-vues) puis, si la vue n'est pas de la classe `UIImageView`, on la cache. Ajoutez :

```
// on va prendre une capture écran  
// on cache toutes les vues sauf les image view  
for (UIView *vi in [self.view subviews])  
    if (![vi isKindOfClass:[UIImageView class]])  
        [vi setHidden:YES];
```

Puis, on peut prendre une capture écran de ce qui s'affiche :

```
// on fait la capture écran  
CGRect screenRect = [[UIScreen mainScreen] bounds];  
UIGraphicsBeginImageContext(screenRect.size);  
  
CGContextRef ctx = UIGraphicsGetCurrentContext();  
[self.view.layer renderInContext:ctx];  
  
UIImage *capturedimage = UIGraphicsGetImageFromCurrentImageContext();  
UIGraphicsEndImageContext();
```

Enfin, on sauvegarde cette image dans l'album photo :

```
UIImageWriteToSavedPhotosAlbum(capturedimage, self,  
➥ @selector(image:didFinishSavingWithError:contextInfo:), nil);
```

À la fin de l'enregistrement dans l'album, la méthode spécifiée (où plutôt le selector) dans le troisième paramètre de la fonction ci-dessus sera appelée. On réaffichera donc les boutons :

```
#pragma mark -  
#pragma mark sauvegarde  
  
- (void)image:(UIImage *)image didFinishSavingWithError:(NSError *) error  
➥ contextInfo:(void *)contextInfo {  
    NSLog(@"Image sauvée");  
  
    // on remet tous les boutons  
    for (UIView *vi in [self.view subviews])  
        vi.hidden = NO;  
}
```

SECOUER POUR EFFACER !

À la Fiche 15, je vous ai présenté comment utiliser l'accéléromètre pour détecter un mouvement de secousse. Il existe une manière plus simple pour détecter ce type d'événements. En fait, il faut choisir le contrôleur qui réceptionnera les événements. Ajoutez ces lignes :

```
#pragma mark -  
#pragma mark gestion de la secousse  
  
- (BOOL)canBecomeFirstResponder {  
    return YES;  
}  
  
- (void)viewDidAppear:(BOOL)animated {  
    [self becomeFirstResponder];  
}
```

Ensuite, la réception d'un événement appellera ces méthodes :

```
- (void)motionBegan:(UIEventSubtype)motion withEvent:(UIEvent *)event
{
    // appelé lorsque l'événement débute
}

- (void)motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)event
{
    // appelé lorsque l'événement est terminé
}

- (void)motionCancelled:(UIEventSubtype)motion withEvent:(UIEvent *)event
{
    // appelé lorsque l'événement est annulé
}
```

Dans notre cas, nous détecterons l'événement UIEventSubtypeMotionShake. Il en existe d'autres qui ont trait aux événements liés à la lecture de l'audio/vidéo. Il vous faudra regarder le type énuméré UIEventSubtype dans la documentation.

Ici, lorsque la secousse sera détectée, on enlèvera toutes les vues de la classe UIImageView sauf celle qui contient l'image originale de fond :

```
- (void)motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)event
{
    if (event.subtype == UIEventSubtypeMotionShake) {
        // on fait une copie des sous vues de la vue du contrôleur
        NSArray *arrayOfSubviews = [self.view subviews];
        // on énumère toutes les image view
        for (UIView *vi in arrayOfSubviews)
        {
            // si ce sont des seringues, on les enlève
            if ([vi isKindOfClass:[UIImageView class]])
                if (((UIImageView*)vi).image == funnyImage)
                    [vi removeFromSuperview];
        }
    }
}
```

N'oubliez pas la méthode dealloc !

```
- (void)dealloc {
    [imageViewBackgroundPhoto release];
    [funnyImage release];
    [super dealloc];
}
```

Où comment intégrer de l'achat de contenu dans votre application.

Lorsque vous distribuez votre application sur l'App Store, qu'elle soit gratuite ou non, vous pouvez demander à l'utilisateur d'y acheter du contenu. Un moyen très pratique dans les cas suivants par exemple :

- Votre application propose le contenu d'un journal. Vous pouvez demander à l'utilisateur de s'abonner au journal moyennant finances pour continuer à bénéficier de votre application.
- Vous avez développé un jeu, et souhaitez proposer du contenu en plus comme l'achat de vies, de niveaux, d'armes.
- Votre application est initialement gratuite et bridée. Vous pouvez demander à l'utilisateur de la débrider en payant via l'In App Purchase.

Vous avez certainement plein d'idées ! Ce système, très pratique, permet de faire évoluer votre application en fonction des désirs de l'utilisateur.

Info

Comme pour la vente d'applications, Apple garde 30 % du prix de vente et vous redistribue les 70 % restants.

L'ajout de l'In App Purchase dans votre application peut se révéler fastidieux car il faut créer un provisioning pour chaque application distribuée, intégrant de l'In App Purchase. Mais ne vous découragez pas !

Nous verrons dans un premier temps comment configurer le provisioning pour disposer de l'In App Purchase dans notre application, puis nous construirons une classe dans un nouveau projet permettant de gérer le store. Enfin, nous ajouterons des objets à acheter, puis nous verrons comment tester l'application sans débourser un seul centime !

Notez que j'entends par store l'élément dans votre application faisant l'interface entre les serveurs Apple et votre application, pour vous distribuer les éléments disponibles en In App Purchase.

Attention

Vous devez avoir acheté la licence pour développer sur iPhone afin de pouvoir réaliser cette fiche.

CRÉATION DU PROVISIONING

Il est primordial de bien suivre l'ordre des étapes, et si possible, de remplir les mêmes données des exemples pour ne pas s'y perdre. Nous allons créer un provisioning pour développer notre application de test In App Purchase.

Rendez-vous sur le site <http://developer.apple.com/iphone/> et loguez-vous. Ensuite, dirigez-vous vers iPhone Provisioning Portal puis sur App IDs. Cliquez sur New App ID comme à la Figure 21.1.

The screenshot shows the iPhone Provisioning Portal interface. At the top, there's a navigation bar with 'Welcome, Marian PAUL' and links for 'Edit Profile' and 'Log out'. Below the navigation bar, the main title is 'iPhone Provisioning Portal' followed by 'Provisioning Portal : iPnP'. On the left, a sidebar lists 'Home', 'Certificates', 'Devices', 'App IDs' (which is selected and highlighted in blue), and 'Distribution'. The main content area has tabs for 'Manage' and 'How To'. Under 'Manage', the 'App IDs' tab is active, showing a sub-section titled 'App IDs'. A sub-sub-section titled 'New App ID' is visible on the right. Below these sections, there's a detailed description of what App IDs are used for.

Figure 21.1 : Créez une nouvelle App ID.

Remplissez les informations comme à la Figure 21.2. Après avoir cliqué sur Submit, vous devriez retrouver une ligne comme à la Figure 21.3 (excepté le numéro avant .com).

This is a 'Create App ID' dialog box. It has a 'Manage' tab at the top. The main area is titled 'Create App ID' and contains several input fields and descriptions:

- Description:** A text input field containing 'Test In App Purchase'. A note below it says: "Enter a common name or description of your App ID using alphanumeric characters. The description you specify will be used throughout the Provisioning Portal to identify this App ID." It also notes: "You cannot use special characters as @, &, *, " in your description."
- Bundle Seed ID (App ID Prefix):** A dropdown menu set to 'Generate New'. A note below it says: "If you are creating a suite of applications that will share the same Keychain access, use the same bundle Seed ID for each of your application's App IDs."
- Bundle Identifier (App ID Suffix):** An input field containing 'com.mondomaine.testInAppPurchase'. A note below it says: "Enter a unique identifier for your App ID. The recommended practice is to use a reverse-domain name style string for the Bundle Identifier portion of the App ID." It also provides an example: "Example: com.domainname.appname".

At the bottom right are 'Cancel' and 'Submit' buttons.

Figure 21.2 : Rentrez les informations de l'App ID.



Figure 21.3 : Votre nouvelle App ID.

Cliquez sur Configure puis activez l'In App Purchase comme à la Figure 21.4. Cliquez sur Done. De retour à la liste des App IDs, vous devriez avoir le bouton vert dans la colonne In App Purchase pour votre App ID.

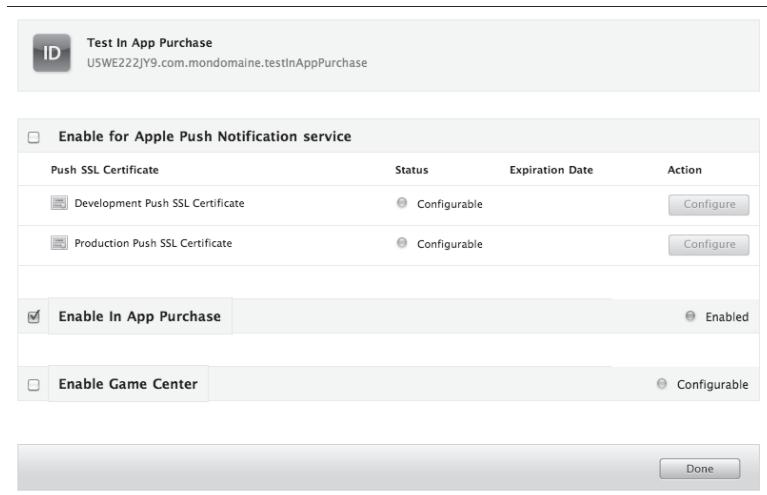


Figure 21.4 : Activez l'In App Purchase.

Maintenant, rendez-vous dans Provisioning (juste en dessous de App IDs) puis cliquez sur New Profile. Rentrez le nom comme à la Figure 21.5, puis choisissez les développeurs concernés par ce provisioning (choix des certificats) puis les appareils concernés et enfin, choisissez l'App ID que nous venons de créer. Vous devriez obtenir quelque chose d'analogique à la Figure 21.5.

| Profile Name | Test InAP Livre |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Certificates | Select All <input type="checkbox"/> Hibert Aurélien <input type="checkbox"/> Loann Fralion <input type="checkbox"/> Marian PAUL |
| App ID | <input type="text" value="Test In App Purchase"/> |
| Devices | Select All <input type="checkbox"/> Julian Pratelli <input type="checkbox"/> Marion Isop <input type="checkbox"/> Camille <input type="checkbox"/> Coline Hervé <input type="checkbox"/> Paul Pratelli <input type="checkbox"/> iPhone de Hibert Aurélien <input type="checkbox"/> iPhone Gofour Pratelli <input type="checkbox"/> iPhone Goo-Go <input type="checkbox"/> iPhone Lagoon <input type="checkbox"/> Paul Isop <input type="checkbox"/> Amélie Pratelli <input type="checkbox"/> Louis Coline <input type="checkbox"/> Marion Hervé-Pratelli <input type="checkbox"/> Amélie amélie optique <input type="checkbox"/> Union applicationnelle <input type="checkbox"/> Wapakoo iPhone <input type="checkbox"/> jérémie lagrue <input checked="" type="checkbox"/> iPhone 3GS de Marian PAUL <input type="checkbox"/> iPhone de Marion <input type="checkbox"/> iPhone Gofour julien <input type="checkbox"/> iPhone gosy <input type="checkbox"/> iPhone Marion <input type="checkbox"/> Paul Isop <input type="checkbox"/> Amélie Pratelli <input type="checkbox"/> Louis Coline <input type="checkbox"/> Marion Hervé <input type="checkbox"/> Amélie amélie optique <input type="checkbox"/> Union applicationnelle <input type="checkbox"/> Wapakoo iPhone |

Figure 21.5 : Remplissez les données pour le provisioning.

Faites Submit et vous devriez maintenant pouvoir télécharger ce nouveau provisioning, comme à la Figure 21.6. Téléchargez-le pour un usage futur dans cette fiche.



Figure 21.6 :
Téléchargez
le nouveau
provisioning.

CRÉATION D'UN STORE MANAGER

Dans cette partie, nous allons créer une classe qui servira d'interface entre l'App Store et l'iPhone. Cette classe permettra de demander à l'App Store les éléments disponibles à l'achat via l'In App Purchase, procédera aux paiements, et sauvegardera le statut des achats. L'avantage, c'est qu'une fois qu'elle sera faite, vous pourrez la réutiliser dans toutes vos applications intégrant l'In App Purchase en ne réalisant que de petites modifications.

Pour ce faire, nous allons créer une classe Singleton. C'est ce que l'on appelle un *design pattern* au même titre que le MVC (*Model View Controller*) ou le delegate. Un design pattern (ou patron de conception en français), c'est un schéma de programmation mis en place pour simplifier la vie des développeurs. Lorsque l'on parle de classe Singleton, on sous-entend une classe A qui contient un objet de la classe A directement instancié. Ainsi, quand on se sert de la classe, on fait toujours appel au même objet qui n'est instancié qu'une seule fois dans la classe.

Info

Il est très important de commencer par lire la Fiche 11 concernant le patron de conception delegate que nous exploitons ici. De plus, vous aurez besoin de la Fiche 35 traitant du protocole NSCoding pour l'archivage de données.

Commencez par créer un nouveau projet de type Window-based Application que vous nommerez TestInAppPurchase. Ajoutez un nouveau fichier, Objective-C class, sous-classe de NSObject que vous nommerez InAppPurchaseStoreManager. Incluez dans votre projet le framework StoreKit et importez-le dans InAppPurchaseStoreManager.h.

Dans ce fichier, nous allons créer un protocole nommé InAppPurchaseStoreManagerDelegate

```
#import <Foundation/Foundation.h>
#import <StoreKit/StoreKit.h>

@protocol InAppPurchaseStoreManagerDelegate;

@interface InAppPurchaseStoreManager : NSObject {
    id<InAppPurchaseStoreManagerDelegate> delegate;
}
```

```
@property (nonatomic, assign) id<InAppPurchaseStoreManagerDelegate> delegate;  
@end  
  
@protocol InAppPurchaseStoreManagerDelegate <NSObject>  
@optional  
// informe le delegate de l'achat d'un produit  
- (void)productPurchased:(NSString *)productId;  
@required  
// demande au delegate d'afficher la liste des produits  
- (void)inAppPurchaseStoreManager:(InAppPurchaseStoreManager*)manager  
    askToDisplayListOfProducts:(NSMutableArray*)list;  
@end
```

Ensuite, nous allons créer une méthode de classe, sharedManager, qui va instancier l'objet de classe.
Ajoutez dans le .m :

```
#import "InAppPurchaseStoreManager.h"  
@implementation InAppPurchaseStoreManager  
@synthesize delegate;  
// notre manager  
static InAppPurchaseStoreManager *sharedManager;  
  
+ (InAppPurchaseStoreManager *)sharedManager  
{  
    @synchronized(self)  
    {  
        if (!sharedManager)  
            sharedManager = [[InAppPurchaseStoreManager alloc] init];  
        return sharedManager;  
    }  
    return sharedManager;  
}  
  
- (void) dealloc {  
    [sharedManager release];  
    [super dealloc];  
}  
  
#pragma mark Singleton Methods  
  
+ (id)allocWithZone:(NSZone *)zone  
{  
    @synchronized(self) {  
        if (sharedManager == nil) {  
            sharedManager = [super allocWithZone:zone];  
            return sharedManager;  
        }  
    }  
}
```

```

        }
        return nil;
    }
- (id)copyWithZone:(NSZone *)zone
{
    return self;
}
- (id)retain
{
    return self;
}
- (unsigned)retainCount
{
    return UINT_MAX; // sous-entend que l'objet ne peut pas être release
}
- (void)release
{
    // on ne fait rien !
}
- (id)autorelease
{
    return self;
}
@end

```

N'oubliez pas de déclarer la méthode dans le .h :

```

@property (nonatomic, assign) id<InAppPurchaseStoreManagerDelegate> delegate;
+ (InAppPurchaseStoreManager *)sharedManager;
@end

```

Info

Pour accéder à notre objet, il faudra donc faire `InAppPurchaseStoreManager *manager = [InAppPurchaseStoreManager sharedManager]`; qui se chargera d'instancier un objet la première fois, puis renverra ensuite une référence vers l'objet `sharedManager` déclaré en static. Si vous êtes débutant, cela peut vous paraître difficile mais cela viendra par la suite.

N'oublions pas que cette classe doit communiquer avec l'App Store et procéder à des paiements. Chaque objet disponible via l'`In App Purchase` aura un identifiant que vous aurez choisi (nous le verrons à la section "Ajouter des produits à acheter"). Ensuite, nous allons créer un tableau qui sauvegardera les objets avec un booléen associé précisant si l'objet a déjà été acheté ou non.

Cela peut vous paraître inutile, car chaque objet a déjà un type (achat pouvant être répété ou unique).

En fait, lorsque vous allez racheter un objet de type unique, une alert view précisera automatiquement que vous avez déjà acheté cet article, et que vous pourrez le télécharger à nouveau gratuitement. Mais dans le cas où vous proposez un abonnement, il est préférable de savoir au lancement de l'application si l'utilisateur y a déjà souscrit. Ainsi, il suffira de lire ce tableau pour le savoir !

La première étape est de créer un objet IAPPProduct respectant le protocole NSCoding pour l'archiver dans un fichier. Ajoutez un nouveau fichier de type Objective-C class sous-classe de NSObject dans votre projet et nommez-le IAPPProduct. Ensuite, écrivez ces quelques lignes :

```
IAPPProduct.h
#import <Foundation/Foundation.h>

#define kIdentifier @"identifier"
#define kPurchased @"purchased"

@interface IAPPProduct : NSObject <NSCoding> {
    NSString *productID;
    BOOL purchased;
}

@property (nonatomic, retain) NSString *productID;
@property (assign) BOOL purchased;
@end

IAPPProduct.m
#import "IAPPProduct.h"

@implementation IAPPProduct
@synthesize productID, purchased;

- (id) initWithCoder:(NSCoder *)coder {
    if (self = [super init]) {
        self.productID = [coder decodeObjectForKey:kIdentifier];
        self.purchased = [coder decodeBoolForKey:kPurchased];
    }
    return self;
}

- (void) encodeWithCoder:(NSCoder *)coder {
    [coder encodeObject:productID forKey:kIdentifier];
    [coder encodeBool:purchased forKey:kPurchased];
}

- (void) dealloc {
    [super dealloc];
    [productID release];
}
@end
```

Déclarez le tableau qui nous servira à stocker tous les objets de la classe IAPPProduct, dans le .h de InAppPurchaseStoreManager :

```
NSMutableArray *listOfProductsPurchasedOrNot;
```

Puis créez la méthode pour lire le tableau depuis les préférences dans le .m :

```
- (void) retrieveListProductsFromSavedFile {
    if (listOfProductsPurchasedOrNot) {
        [listOfProductsPurchasedOrNot release];
    }
    // on récupère depuis le fichier avec NSCoder
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
    ↳ NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *path = [documentsDirectory
    ↳ stringByAppendingPathComponent:@"products.myArchive"];
    NSData *data;
    NSKeyedUnarchiver *unarchiver;
    data = [[NSData alloc] initWithContentsOfFile:path];
    if(data)
    {
        // si le fichier contient des données, on désarchive
        unarchiver = [[NSKeyedUnarchiver alloc]
            initForReadingWithData:data];
        NSMutableArray *arrayArchived = [unarchiver decodeObjectForKey:@"Products"];
        listOfProductsPurchasedOrNot = [arrayArchived retain];
        [unarchiver finishDecoding];
        [unarchiver release];
        [data release];
    }

    if (!listOfProductsPurchasedOrNot) {
        // le fichier n'existe pas, on le crée
        listOfProductsPurchasedOrNot = [[NSMutableArray array] retain];
        // ajout du premier produit
        IAPPProduct *product1 = [[IAPPProduct alloc] init];
        product1.productID = kIdProduit1;
        product1.purchased = NO;
        [listOfProductsPurchasedOrNot addObject:product1];
        [product1 release];

        // ajout du second produit
        IAPPProduct *product2 = [[IAPPProduct alloc] init];
        product2.productID = kIdProduit2;
        product2.purchased = NO;
        [listOfProductsPurchasedOrNot addObject:product2];
        [product2 release];
    }
}
```

```
}
```

N'oubliez pas l'import dans le .h :

```
#import "IAPPProduct.h"
```

Vous voyez ici que nous avons défini des clés. Ajoutez donc ces lignes dans .h

```
// Liste des id
```

```
#define kIdProduit1 @"com.mondomaine.testInAppPurchase.001"
```

```
#define kIdProduit2 @"com.mondomaine.testInAppPurchase.002"
```

Chaque produit disponible via l'In App Purchase aura son propre identifiant qu'il vous conviendra de définir. Un conseil : respectez com(ou fr).votreNomAppStore.nomApplication.numéro, ce sera plus simple pour vous y retrouver !

Info

Pour éviter une mise à jour lorsque vous ajoutez un produit disponible en achat intégré, récupérez cette liste depuis un webservice au lieu de la définir en dur comme nous le faisons ici. Ensuite, bien entendu, il faut que votre application puisse gérer ce nouveau contenu !

Nous ne souhaitons pas que cette méthode soit publique. Ainsi, nous allons créer une catégorie (encore un patron de conception !). Pour cela, rajoutez ces lignes en gras dans le .m :

```
#import "InAppPurchaseStoreManager.h"

@interface InAppPurchaseStoreManager(private)
- (void) retrieveListProductsFromFile;
@end

@implementation InAppPurchaseStoreManager
// ...
```

Nous allons appeler cette dernière méthode lors de linitialisation du manager :

```
+ (InAppPurchaseStoreManager *)sharedManager
{
    @synchronized(self)
    {
        if (!sharedManager)
        {
            sharedManager = [[InAppPurchaseStoreManager alloc] init];
            // on récupère la liste des produits stockés dans un fichier ou les
            // préférences
            [sharedManager retrieveListProductsFromFile];
        }
        return sharedManager;
    }
    return sharedManager;
}
```

La prochaine étape est de demander à l'App Store les produits disponibles à l'achat pour notre application.

Ajoutez dans le .m cette méthode :

```
// demande la liste des produits disponibles auprès de l'app Store
- (void) requestProductData
{
    // crée un tableau ne contenant que les identifiants disponibles
    // remarquez que la clé est le nom de l'objet NSString *productID de IAPPProduct
    NSArray *arrayOfIdentifiers = [listOfProductsPurchasedOrNot
        ↪ valueForKey:@"productID"];
    // création de la requête
    SKProductsRequest *request= [[SKProductsRequest alloc]
        ↪ initWithProductIdentifiers: [NSSet setWithArray:arrayOfIdentifiers]];
    // self est delegate de SKProductsRequest
    request.delegate = self; ①
    // on démarre
    [request start];
    // il n'y a pas de fuites, on release l'objet request plus tard
}
```

Nous souhaitons que cette méthode soit publique. Il faut donc la déclarer dans le .h :

```
- (void) requestProductData;
```

Ligne ①, on met self comme delegate de request. Ainsi, il faut ajouter dans le .m :

```
@interface InAppPurchaseStoreManager : NSObject <SKProductsRequestDelegate>
```

Cela implique également d'implémenter correctement le protocole SKProductsRequestDelegate.

Ajoutez cette méthode dans le .m :

```
#pragma mark -
#pragma mark SKProductsRequestDelegate methods
// appelée lorsque le delegate reçoit la réponse de l'App Store
- (void)productsRequest:(SKProductsRequest *)request
    ↪ didReceiveResponse:(SKProductsResponse *)response
{
    if (!listOfProductsAvailable) {
        listOfProductsAvailable = [[NSMutableArray alloc] init];
    }
    // on enlève tous les objets
    [listOfProductsAvailable removeAllObjects];
    [listOfProductsAvailable addObjectsFromArray:response.products];

    // à vous d'afficher une vue pour présenter les données

    [delegate inAppPurchaseStoreManager:self
        ↪ askToDisplayListOfProducts:listOfProductsAvailable];
```

```
// on release ici la requête instanciée plus haut dans requestProductData
[request autorelease];
}
```

N'oubliez pas la déclaration dans le .h de

```
NSMutableArray *listOfProductsAvailable;
```

Cette méthode sera appelée lorsque le delegate (ici sharedManager) recevra la réponse de l'App Store. Il vous incombe ensuite de présenter les données comme vous le souhaitez ! Nous les afficherons par la suite.

GÉRER LE PAIEMENT

Imaginez que votre application présente la liste des produits disponibles à l'achat. Lorsque le client en sélectionnera un, il faudra qu'il puisse l'acheter. Pour cela, il faut solliciter SKPayment.

Ajoutez le protocole SKPaymentTransactionObserver dans le .h :

```
@interface InAppPurchaseStoreManager : NSObject <SKProductsRequestDelegate,
➥ SKPaymentTransactionObserver>
```

En fait, la gestion du paiement se fait de manière complètement transparente pour vous grâce à un observateur. Ainsi, si l'utilisateur quitte l'application alors qu'une transaction est en cours, elle sera automatiquement terminée au prochain lancement.

La première ligne à écrire consiste en l'ajout d'un observateur du statut des paiements :

```
+ (InAppPurchaseStoreManager *)sharedManager
{
    @synchronized(self)
    {
        if (!sharedManager)
        {
            sharedManager = [[InAppPurchaseStoreManager alloc] init];
            // on récupère la liste des produits stockés dans un fichier
            [sharedManager retrieveListProductsFromSavedFile];

            // on ajoute un observateur du store le plus tôt possible pour gérer
            // les achats
            [[SKPaymentQueue defaultQueue] addTransactionObserver:sharedManager];
        }
        return sharedManager;
    }
    return sharedManager;
}
```

Pour demander le paiement, on écrit une méthode publique (à déclarer dans le .h donc) :

```
#pragma mark -
#pragma mark buy methods
```

```

- (void) wantToPurchaseProduct:(NSString*) productID {
    if ([SKPaymentQueue canMakePayments]) ①
    {
        SKPayment *payment = [SKPayment paymentWithProductIdentifier:productID]; ②
        [[SKPaymentQueue defaultQueue] addPayment:payment];
    }
    else
    {
        // L'utilisateur a désactivé la possibilité d'acheter -> présenter une
        ➔ alert si besoin
    }
}

```

Ligne ①, il faut absolument appeler la méthode `canMakePayments` de la classe `SKPaymentQueue` car l'utilisateur peut avoir désactivé l'achat intégré. Ensuite, ligne ②, on crée un nouveau paiement que l'on ajoutera ensuite dans un buffer qui gérera les paiements.

Info

Si vous proposez à l'utilisateur d'acheter plusieurs objets, il suffit d'écrire `payment.quantity = unNombre;`

Ensuite, il faut implémenter correctement le protocole `SKPaymentTransactionObserver` pour connaître le statut d'un achat :

```

- (void)paymentQueue:(SKPaymentQueue *)queue updatedTransactions:(NSArray *)
    ➔ transactions
{
    for (SKPaymentTransaction *transaction in transactions)
    {
        switch (transaction.transactionState)
        {
            case SKPaymentTransactionStatePurchased:
                [self completeTransaction:transaction];
                break;
            case SKPaymentTransactionStateFailed:
                [self failedTransaction:transaction];
                break;
            case SKPaymentTransactionStateRestored:
                [self restoreTransaction:transaction];
            default:
                break;
        }
    }
}

```

Voici la signification de ces trois états :

- `SKPaymentTransactionStatePurchased` L'App Store a correctement réalisé le paiement. Votre application doit donc montrer à l'utilisateur l'impact de son achat.
- `SKPaymentTransactionStateFailed` La transaction s'est mal déroulée. Vous pouvez regarder la propriété `error` pour en savoir plus.
- `SKPaymentTransactionStateRestored` La transaction a restauré un contenu initialement acheté par l'utilisateur. Vous pouvez lire la propriété `originalTransaction` pour connaître les informations de l'achat original.

Pour finir avec ce manager, nous allons implémenter les méthodes `completeTransaction:`, `failedTransaction:` et `restoreTransaction:` qui seront privées, ainsi que `recordTransaction:` :

```
@interface InAppPurchaseStoreManager(private)
- (void) retrieveListProductsFromSavedFile;
- (void) completeTransaction: (SKPaymentTransaction *)transaction;
- (void) restoreTransaction: (SKPaymentTransaction *)transaction;
- (void) failedTransaction: (SKPaymentTransaction *)transaction;
- (void) recordTransaction:(SKPaymentTransaction*) transaction;
@end
```

Commencez par `completeTransaction:` :

```
- (void) completeTransaction: (SKPaymentTransaction *)transaction
{
    // on enregistre
    [self recordTransaction: transaction];
    // on informe le delegate
    if([delegate respondsToSelector:@selector(productPurchased:)])
    {
        [delegate productPurchased:transaction.payment.productIdentifier];
    }
    // on finit la transaction
    [[SKPaymentQueue defaultQueue] finishTransaction: transaction];
}
```

Il y a trois temps : la sauvegarde dans le fichier, puis on informe le delegate, s'il implémente la méthode `productPurchased:`, et on finit la transaction.

De même,

```
- (void) restoreTransaction: (SKPaymentTransaction *)transaction
{
    // on enregistre
    [self recordTransaction: transaction];
    // on informe le delegate
    if([delegate respondsToSelector:@selector(productPurchased:)])
    {
        [delegate productPurchased:transaction.payment.productIdentifier];
    }
}
```

```

        }
        // on finit la transaction
        [[SKPaymentQueue defaultQueue] finishTransaction: transaction];
    }

- (void) failedTransaction: (SKPaymentTransaction *)transaction
{
    if (transaction.error.code != SKErrorPaymentCancelled)
    {
        // il y a eu une erreur dans l'achat
    }
    // on finit la transaction
    [[SKPaymentQueue defaultQueue] finishTransaction: transaction];
}

```

Et la méthode de sauvegarde (archivage avec NSCoder) :

```

- (void) recordTransaction:(SKPaymentTransaction*) transaction
{
    // on énumère pour mettre à YES le booléen du produit acheté
    for (IAPPProduct *iapProduct in listOfProductsPurchasedOrNot) {
        if ([iapProduct.productID isEqualToString:transaction.payment.productIdentifier]) {
            iapProduct.purchased = YES;
        }
    }
    // on sauvegarde en avec NSCoder
    NSMutableData *data;
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
    ↳ NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *archivePath =
    ↳ [documentsDirectory stringByAppendingPathComponent:@"products.myArchive"];
    NSKeyedArchiver *archiver;

    data = [NSMutableData data];
    archiver = [[NSKeyedArchiver alloc] initForWritingWithMutableData:data];
    [archiver encodeObject:listOfProductsPurchasedOrNot forKey:@"Products"];
    [archiver finishEncoding];
    [data writeToFile:archivePath atomically:YES];
    [archiver release];
}

```

Pour le plaisir, ajoutez une méthode publique qui permettra de savoir si l'on a déjà acheté ou non un produit :

```
// retourne un booléen selon si le produit a déjà été acheté ou non
- (BOOL) havePurchasedProduct:(NSString*) productID {
    for(IAPPProduct *iapProduct in listOfProductsPurchasedOrNot) {
        if ([iapProduct.productID isEqualToString:productID]) {
            return iapProduct.purchased;
        }
    }
}
```

AJOUTER DES PRODUITS À ACHETER

Arrêtons un peu de coder quelques instants, et revenez à votre navigateur préféré pour vous rendre sur l'iTunes Connect : <https://itunesconnect.apple.com/>. Loguez-vous, puis rendez-vous dans Manage Your Applications. Cliquez ensuite sur Add New Application et remplissez toutes les données nécessaires. Lorsque l'on vous le demande, cochez Upload your binary later.

De retour sur la page d'accueil de l'iTunes Connect, cliquez sur Manage Your In App Purchases puis sur Create New. Sélectionnez l'application que vous venez d'ajouter puis choisissez le bon bundle ID comme à la Figure 21.7.

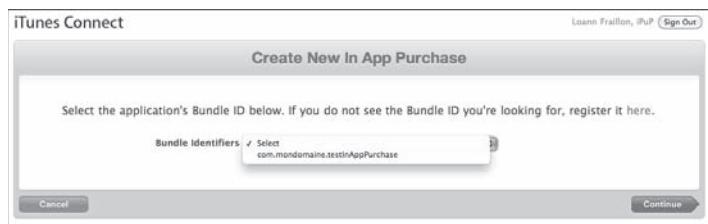


Figure 21.7 : Choisissez le Bundle ID.

Ensuite, créez un nouveau produit en remplaçant les informations comme aux Figures 21.8 et 21.9. N'oubliez pas qu'il faut bien remplir le Product ID comme nous l'avons fait dans l'application (ou l'inverse : modifiez votre application en fonction des Product ID créés).

iTunes Connect

Loann Fraillon, iPuP [Sign Out](#)

Create New In App Purchase

Pricing

Reference Name : Pack de 100 vies [?](#)

Product ID : com.mondomaine.testInAppPurchase.001 [?](#)

Type : Consumable [?](#)

Cleared for Sale :

Price Tier : [Tier 2](#) [?](#) See Pricing Matrix [»](#)

| Price Tier 2 | | | | | | | | | | | | |
|----------------|--------|---------|--------|-------|----------------|---------|---------|---------|-------------|-----------|-------------|-------|
| App Store | US* | Mexico | Canada | UK | European Union | Norway | Sweden | Denmark | Switzerland | Australia | New Zealand | Japan |
| Customer Price | \$1.99 | \$20.00 | \$1.99 | £1.19 | 1,59 € | 11.00kr | 15.00kr | 12.00kr | 2.20Fr | AU \$2.49 | NZ \$2.59 | ¥230 |
| Your Proceeds | \$1.40 | \$1.40 | \$0.72 | | | 0.97 € | | | | AU \$1.58 | | ¥161 |

*The U.S. price applies to all countries where applications are sold in U.S. dollars. The European Union price applies to all countries where applications are sold in Euros. See Details.

Display Detail

This is used by the iTunes Store to display the name of your In App Purchase during purchase.
At least one language is required.

Figure 21.8 : Choix du nom, de l'ID et du prix.

Display Detail

This is used by the iTunes Store to display the name of your In App Purchase during purchase.
At least one language is required.

Language to Add : [Please Select One](#) [?](#)

| |
|-------------------------------------------------------------------------------|
| Language : French |
| Displayed Name : Pack de 100 vies |
| Description : Pour acheter une centaine de vies et jouer pendant des heures ! |

Figure 21.9 : Choix du nom affiché et de la description.

Le type choisi doit être réfléchi : si vous sélectionnez Consumable ou Subscription, le produit pourra être acheté plusieurs fois. Par contre, si vous choisissez Non-Consumable, le produit ne pourra être acheté qu'une unique fois par l'utilisateur et tous ses achats futurs seront gratuits.

Créez un deuxième produit comme à la Figure 21.10.

The screenshot shows the 'Create New In App Purchase' interface in iTunes Connect. The product is named 'Achat d'une voiture de course'. It is a non-consumable item with a price tier of 'Tier 1'. The table below shows the pricing for various countries:

| Price Tier 1 | | | | | | | | | | | | |
|----------------|--------|---------|--------|-------|-----------------|--------|--------|---------|-------------|-----------|-------------|-------|
| App Store | US* | Mexico | Canada | UK | European Union* | Norway | Sweden | Denmark | Switzerland | Australia | New Zealand | Japan |
| Customer Price | \$0.99 | \$10.00 | \$0.99 | E0.59 | 0.79 € | 6.00kr | 7.00kr | 6.00kr | 1.10fr | AU \$1.19 | NZ \$1.25 | ¥115 |
| Your Proceeds | \$0.70 | \$0.70 | \$0.70 | E0.36 | | 0.48 € | | | | AU \$0.76 | | ¥81 |

*The U.S. price applies to all countries where applications are sold in U.S. dollars. The European Union price applies to all countries where applications are sold in Euros. See Details.

Figure 21.10 :
Deuxième produit.

De retour à la liste des produits sur iTunes Connect pour votre application, vous pouvez voir que le statut est Pending Developer Approval. Il faut donc approuver ce que vous venez de faire en cliquant sur Achat d'une voiture de course puis sur Approve comme à la Figure 21.11. Faites de même pour Le pack de 100 vies.

Vous pouvez créer plusieurs descriptions selon les langues des pays dans lesquels vous souhaitez distribuer l'application.

The screenshot shows the product details page for 'Achat d'une voiture de course'. The status is 'Pending Developer Approval'. The 'Display Detail' section contains the following information:

- Language : French
- Displayed Name : Voiture de course
- Description : Achetez une voiture de course pour aller plus vite que tous vos concurrents !

Figure 21.11 :
Proposez l'achat d'une
voiture de course.

CRÉER UN COMPTE DE TEST

Vous n'avez certainement pas envie de payer lorsque vous testez... Ainsi, il faut créer un nouveau compte iTunes. Retournez à l'accueil d'iTunes Connect puis sur Manage Users et In App Purchase Test User. Cliquez sur Add New User et renseignez les champs demandés. L'adresse e-mail devrait être différente de celle de votre compte iTunes "normal".

Il faut maintenant changer le compte principal de votre appareil :

1. Allez dans l'application Réglages puis Store.
2. Déconnectez-vous puis reconnectez-vous avec l'identifiant que vous venez de créer.
3. Remplissez les informations demandées. Ne vous inquiétez pas quand vous devez donner les coordonnées de votre carte bancaire. En effet, vous pouvez vérifier que votre adresse est celle d'Apple ! Tout va bien donc...

DERNIÈRE LIGNE DROITE

Nous allons, pour finir, revenir à notre projet et compiler notre application pour acheter des produits. Commencez par glisser le provisioning créé au début de la fiche sur l'icône d'Xcode. Ensuite, modifiez le .plist de l'application et remplacez la valeur pour la clé Bundle Identifier par com.mondomaine.testInAppPurchase. (Le même que celui spécifié lors de la création du provisioning).

Puis, ajoutez un nouveau fichier dans votre projet, de type UIViewController sous-classe de UITableView-ViewController que vous nommerez StoreDisplayTableViewController. Ce contrôleur servira à afficher les différents produits que l'utilisateur pourra acheter. Créez également un fichier de type UIViewController que vous nommerez RootViewController. Laissez cocher With XIB for User Interface pour ces deux fichiers.

Instanciez ce contrôleur dans l'application delegate :

```
- (BOOL)application:(UIApplication *)application  
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    rootViewController = [[RootViewController alloc]  
        initWithNibName:@"RootViewController" bundle:nil];  
    [rootViewController.view setFrame:[[UIScreen mainScreen] applicationFrame]];  
    [window addSubview:rootViewController.view];  
    [window makeKeyAndVisible];  
  
    return YES;  
}
```

N'oubliez pas que le .h doit être :

```
#import <UIKit/UIKit.h>  
#import "RootViewController.h"  
  
@interface TestInAppPurchaseAppDelegate : NSObject <UIApplicationDelegate> {  
    UIWindow *window;
```

```

    RootViewController *rootViewController;
}
@property (nonatomic, retain) IBOutlet UIWindow *window;
@end

```

Ensuite, ajoutez un bouton à la vue du RootViewController pour afficher le store (dans RootViewController.m) :

```

- (id)initWithNibName:(NSString *)NibNameOrNil
{
    bundle:(NSBundle *)nibBundleOrNilOrNil {
        if ((self = [super initWithNibName:nibNameOrNilOrNil bundle:nibBundleOrNilOrNil])) {
            UIButton *buttonStore = [UIButton buttonWithType:UIButtonTypeRoundedRect];
            [buttonStore addTarget:self action:@selector(askForListOfProducts:)
             forControlEvents:UIControlEventTouchUpInside];
            [buttonStore setTitle:@"Store" forState:UIControlStateNormal];
            [buttonStore setFrame:CGRectMake(30, 30, 100, 30)];
            [self.view addSubview:buttonStore];

            [InAppPurchaseStoreManager sharedManager].delegate = self;
        }
        return self;
    }
}

```

Dans le .h, faire un import du manager puis déclarer RootViewController comme étant delegate de InAppPurchaseStoreManagerDelegate.

```

#import <UIKit/UIKit.h>
#import "InAppPurchaseStoreManager.h"

@interface RootViewController : UIViewController <InAppPurchaseStoreManagerDelegate> {
}
@end

```

Lorsque l'on appuiera sur le bouton Store, on demandera la liste des produits à l'App Store :

```

- (void) askForListOfProducts:(id)sender {
    // demande à l'App Store la liste des produits
    [[InAppPurchaseStoreManager sharedManager] requestProductData];
}

```

Ensuite, il faut implémenter le protocole InAppPurchaseStoreManagerDelegate en commençant par la méthode appelée lorsque le manager a terminé de télécharger la liste des produits. À cet endroit, on affichera le table view controller créé juste avant :

```

#pragma mark -
#pragma mark InAppPurchaseStoreManager delegate
- (void) inAppPurchaseStoreManager:(InAppPurchaseStoreManager *)manager
    askToDisplayListOfProducts:(NSMutableArray *)list {
    // on récupère le tableau
}

```

```

self.arrayToDisplay = list;
// si le navigation controller n'existe pas, on le crée
if(!navController)
{
    StoreDisplayTableViewController *storeDisplayViewController =
        [[[StoreDisplayTableViewController alloc] initWithStyle:UITableViewStylePlain];
    navController = [[UINavigationController alloc]
        initWithRootViewController:storeDisplayViewController];
    storeDisplayViewController.rootViewController = self; ①
    [storeDisplayViewController release];
}
// on affiche la vue
[self presentModalViewController:navController animated:YES];
}

```

Info

On utilise un contrôleur de navigation pour afficher correctement la table view. De plus, le contrôleur de la classe `StoreDisplayTableViewController` aura besoin d'un tableau de données à afficher. Pour cela, il devra récupérer le tableau `arrayToDisplay` de `RootViewController`. Il faut donc que l'instance `storeDisplayViewController` "connaîsse" l'instance actuelle de `RootViewController`, d'où la ligne ①.

Il faut donc modifier le .h :

```

#import <UIKit/UIKit.h>
#import "InAppPurchaseStoreManager.h"

@interface RootViewController : UIViewController <InAppPurchaseStoreManagerDelegate> {
    NSMutableArray *arrayToDisplay;
    UINavigationController *navController;
}
@property (nonatomic, retain) NSMutableArray *arrayToDisplay;
@end

```

Et ajouter dans le .m :

```
@synthesize arrayToDisplay;
```

Ajouter également la méthode appelée lorsque l'utilisateur aura acheté un article :

```
- (void) productPurchased:(NSString *)productId {
    NSLog(@"Vous venez d'acheter %@", productId);
}
```

Ne pas oublier le dealloc :

```
- (void) dealloc {
    [arrayToDisplay release];
    [navController release];
    [super dealloc];
}
```

Ni l'import dans RootViewController.m de StoreDisplayTableViewController.h :

```
#import "StoreDisplayTableViewController.h"
```

Pour afficher les données, nous avons besoin d'une table view. Voici le StoreDisplayTableViewController.h :

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

@class RootViewController;
@interface StoreDisplayTableViewController : UITableViewController {
    RootViewController *rootViewController;
}
@property (nonatomic, retain) RootViewController *rootViewController;
@end
```

N'oubliez pas cette ligne dans le .m :

```
@synthesize rootViewController;
```

Ajoutez un bouton dans la barre de navigation pour le retour (dans le .m) :

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

    UIBarButtonItem *buttonBack = [[UIBarButtonItem alloc]
        initWithBarButtonSystemItem:UIBarButtonSystemItemDone target:self
        action:@selector(backToRoot:)];
    self.navigationItem.leftBarButtonItem = buttonBack;
    [buttonBack release];

    self.title = @"Liste des achats";
}

- (void) backToRoot:(id)sender {
    [rootViewController dismissModalViewControllerAnimated:YES];
}
```

Puis, spécifiez le nombre de sections et de lignes :

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section {
    return [rootViewController.arrayToDisplay count];
}
```

Dans la méthode `cellForRowAtIndexPath`, affichez les données :

```
- (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];
    }

    // on récupère le produit de la ligne
    SKProduct *product = [rootViewController.arrayToDisplay
      ↪ objectAtIndex:indexPath.row];

    // formate le chiffre selon le pays
    NSNumberFormatter *numberFormatter = [[[NSNumberFormatter alloc] init
      ↪ autorelease];
    [numberFormatter setFormatterBehavior:NSNumberFormatterBehavior10_4];
    [numberFormatter setNumberStyle:NSNumberFormatterCurrencyStyle];
    [numberFormatter setLocale:product.priceLocale];
    NSString *formattedPrice = [numberFormatter stringFromNumber:product.price];

    //NSLog(@"Produit : %@\nDescription : %@ \nPrix : %@",[product
    ↪ localizedTitle], [product localizedDescription], formattedPrice,
    ↪ [product productIdentifier]);

    cell.textLabel.text = [NSString stringWithFormat:@"%@ — %@",

      ↪ [product localizedTitle], formattedPrice];

    return cell;
}
```

Info

Notez que toutes les données sont localisées ! Référez-vous à la Fiche 31.

Pour finir, lorsque nous cliquons sur un produit, lancer l'achat !

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
  ↪ *)indexPath {
    // on récupère le produit à acheter
    SKProduct *productToPurchase = [rootViewController.arrayToDisplay
```

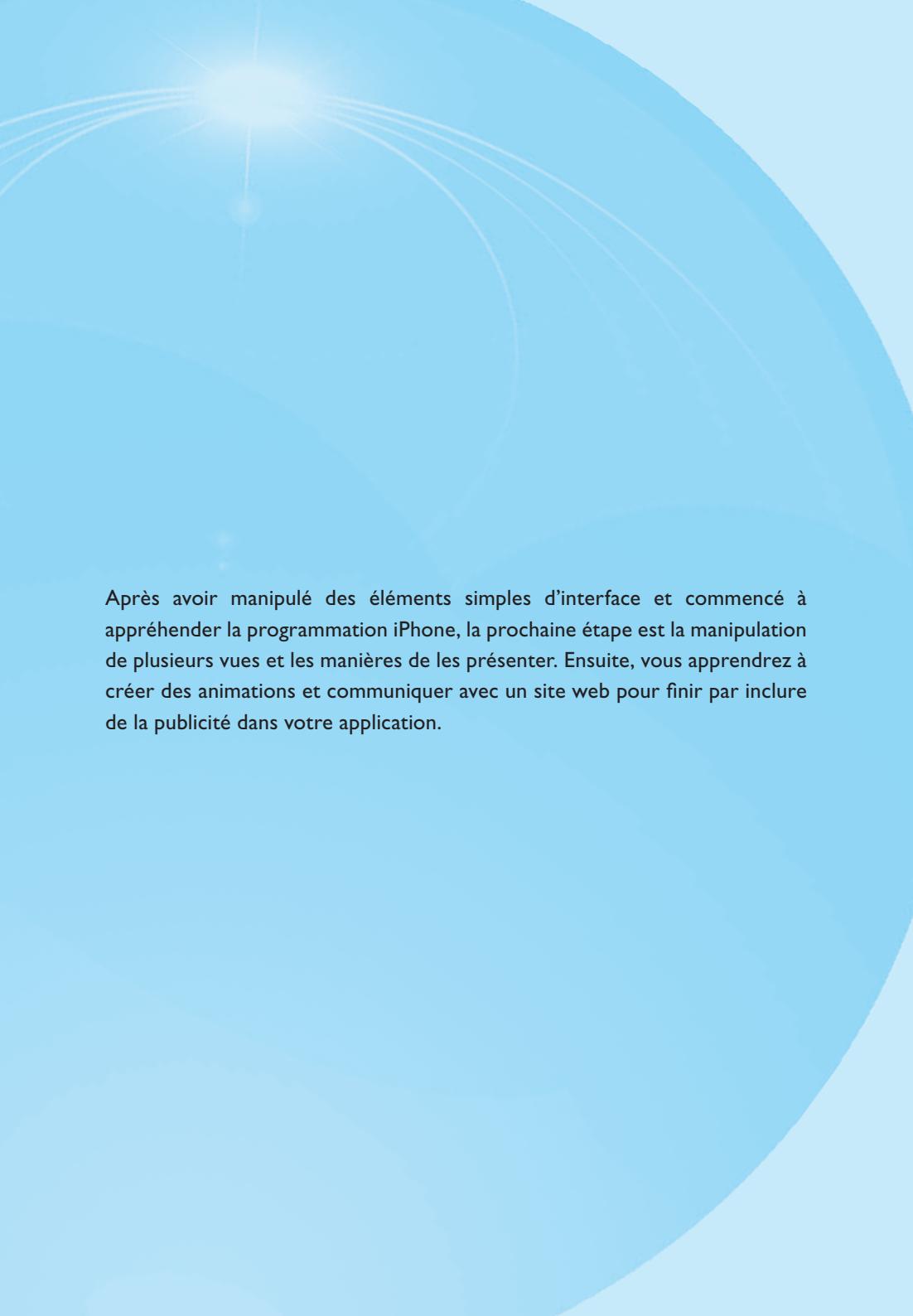
```
    ↪ objectAtIndex:indexPath.row];
    // on demande au manager d'acheter l'objet
    [[InAppPurchaseStoreManager sharedManager]
     ↪ wantToPurchaseProduct:productToPurchase.productIdentifier];
}
```

N'oubliez pas le dealloc :

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

CHAPITRE 5

ALLER PLUS LOIN



Après avoir manipulé des éléments simples d'interface et commencé à apprêhender la programmation iPhone, la prochaine étape est la manipulation de plusieurs vues et les manières de les présenter. Ensuite, vous apprendrez à créer des animations et communiquer avec un site web pour finir par inclure de la publicité dans votre application.

Sur l'iPhone, contrairement aux ordinateurs, vous n'avez qu'une seule fenêtre de disponible pour votre application. Ainsi, l'affichage se fera selon un empilement des vues. Étant débutant, il est assez difficile de le visualiser. Faisons-le ensemble !

Dans cette fiche, nous allons très simplement manipuler deux vues, que l'on enlèvera/affichera grâce à un bouton. Ensuite, nous verrons comment implémenter une gestion plus compliquée, en utilisant un RootViewController.

Tout d'abord créez un nouveau projet de type Window-based Application que vous nommerez GestionVues. Ajoutez ensuite un fichier de type UIViewController que vous nommerez RootViewController. Enfin, ajoutez deux fichiers de type UIView, comme à la Figure 22.1, que vous nommerez respectivement Vue1 et Vue2.

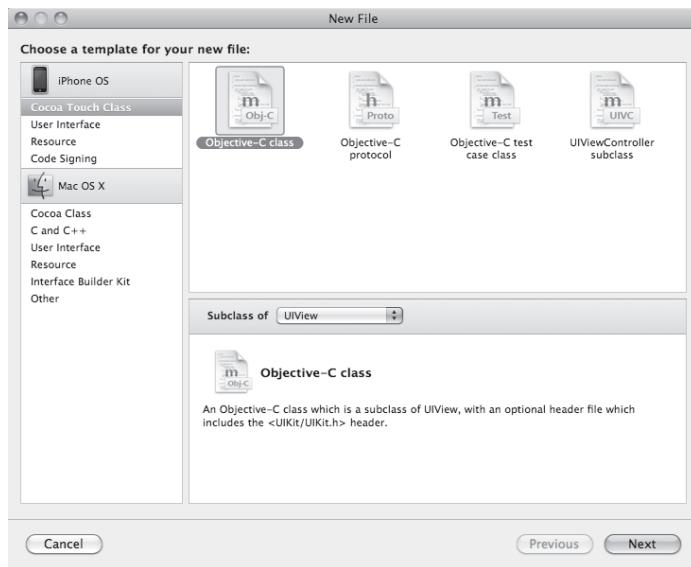


Figure 22.1 : Ajoutez un fichier de type *UIView*.

Dans un premier temps, nous allons ajouter la vue du RootViewController à la fenêtre, dans l'application delegate.

```
GestionVuesAppDelegate.h
#import <UIKit/UIKit.h>
#import "RootViewController.h"

@class RootViewController; ①

@interface GestionVuesAppDelegate : NSObject <UIApplicationDelegate> {
```

```
RootViewController *rootViewController;
UIWindow *window;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) RootViewController *rootViewController; ②

@end

GestionVuesAppDelegate.m
#import "GestionVuesAppDelegate.h"

@implementation GestionVuesAppDelegate

@synthesize window;
@synthesize rootViewController;

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    RootViewController *viewController = [[RootViewController alloc] init];
    self.rootViewController = viewController;
    [viewController release];
    [window addSubview:rootViewController.view]; ③
    [window makeKeyAndVisible];
    return YES;
}
- (void)dealloc {
    [rootViewController release];
    [window release];
    [super dealloc];
}
@end
```

Rien de nouveau dans ce code, si ce n'est une petite différence avec l'objet `rootViewController` instancié. En effet, ligne ②, on se sert du mécanisme `@property` et `@synthesize` qui, rappeler-vous, remplace les getters/setters.

Ligne ①, nous utilisons le mot-clé `@class`. Cela permet de spécifier au compilateur que tout usage de `RootViewController` en tant que nom de classe, dans la déclaration de variables à l'intérieur d'une autre classe (ici `GestionVuesAppDelegate`) est autorisé.

Ligne ③, on ajoute la vue du `RootViewController` à la pile de vues une fois que tout s'affichera. La pile de vues sera donc comme représentée à la Figure 22.2.

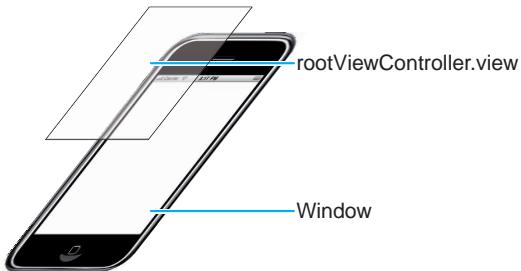


Figure 22.2 : Piles de vues, avec le `rootViewController`.

Ensuite, dans le `RootViewController`, nous allons ajouter la vue1.

```
RootViewController.h
#import <UIKit/UIKit.h>
#import "Vue1.h"

@class Vue1;

@interface RootViewController : UIViewController {
    Vue1 *vue1;
}
@end
```

Modifiez le `loadView`:

```
RootViewController.m
- (void) loadView {
    self.view = [[[UIView alloc] initWithFrame:[[UIScreen mainScreen]
        applicationFrame]] autorelease]; ①
    self.view.backgroundColor = [UIColor whiteColor];
    vue1 = [[Vue1 alloc] initWithFrame:self.view.frame];
    [self.view addSubview:vue1];
}
```

Info

Il est important de noter que lorsque l'on n'utilise pas de fichier .xib, la vue du view controller n'est pas allouée. Il faut donc le faire manuellement ligne ①. Cette méthode sera appelée automatiquement si `view` pointe vers `nil`. Si vous lancez un fichier .xib mais que vous créez tout de même votre interface à la main, il faudra implémenter `initWithNibName: bundle:` voir `viewDidLoad`.

Ensuite, ajoutons dans la vue 1 un bouton qui permettra de changer de vue, ainsi qu'un label pour afficher le nom de la vue.

```
Vue1.h
#import <UIKit/UIKit.h>
#import "GestionVuesAppDelegate.h"
```

```
@interface Vue1 : UIView {
    UILabel *labelStatut;
}
@end

Vue1.m
#import "Vue1.h"
@implementation Vue1

- (id)initWithFrame:(CGRect)frame {
    if ((self = [super initWithFrame:frame])) {
        UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
        [button setFrame:CGRectMake(0, 0, 160, 30)];
        [button setCenter:self.center];
        [button addTarget:self action:@selector(allerALaVue2:)];
        ↪ forControlEvents:UIControlEventTouchUpInside];
        [button setTitle:@"Aller à la vue 2" forState:UIControlStateNormal];
        [self addSubview:button];

        labelStatut = [[UILabel alloc] initWithFrame:CGRectMake(30, 30, 260, 30)];
        labelStatut.text = @"On est sur la vue 1";
        labelStatut.textColor = [UIColor blueColor];
        [self addSubview:labelStatut];
    }
    return self;
}

- (void) allerALaVue2:(id)sender {
    GestionVuesAppDelegate *appDelegate =
    ↪ (GestionVuesAppDelegate*)[[UIApplication sharedApplication] delegate]; ①
    [appDelegate.rootViewController changePourLaVue2]; ②
}

- (void)dealloc {
    [labelStatut release];
    [super dealloc];
}
@end
```

Nous allons nous servir du rootViewController pour échanger nos vues, vue1 et vue2. Pour accéder à l'objet rootViewController instancié par l'application delegate, on fait tout d'abord une référence vers le delegate de l'application ligne ①. Ensuite, on appelle la méthode changePourLaVue2 du rootViewController ligne ②.

Faisons le point sur notre pile de vues : par-dessus la vue du rootViewController, nous affichons la vue1, qui contiendra un bouton et un label. Vous trouverez un schéma Figure 22.3 résumant tout ceci.

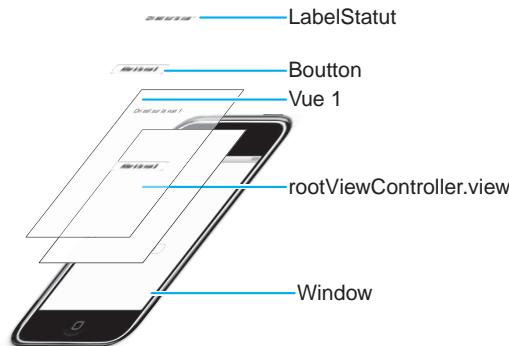


Figure 22.3 :
Piles de vues, avec la vue 1.

Implémentez Vue2.h et Vue2.m de la même manière. Seules ces lignes changeront :

```
[button addTarget:self action:@selector(allerALaVue1:)  
    forControlEvents:UIControlEventTouchUpInside];  
[button setTitle:@"Aller à la vue 1" forState:UIControlStateNormal];  
// ainsi que  
labelStatut.text = @"On est sur la vue 2";  
labelStatut.textColor = [UIColor redColor];  
// et  
- (void) allerALaVue1:(id)sender {  
    GestionVuesAppDelegate *appDelegate =  
        (GestionVuesAppDelegate*)[[UIApplication sharedApplication] delegate];  
    [appDelegate.rootViewController changePourLaVue1];  
}
```

Il ne reste plus qu'à ajouter ou enlever les bonnes vues, et ceci dans RootViewController.m :

```
- (void) changePourLaVue1 {  
    [vue2 removeFromSuperview]; ①  
    [self.view addSubview:vue1]; ②  
}  
- (void) changePourLaVue2 {  
    if(!vue2)  
        vue2 = [[Vue2 alloc] initWithFrame:self.view.frame]; ③  
    [vue1 removeFromSuperview];  
    [self.view addSubview:vue2];  
}
```

C'est l'appel à `removeFromSuperview` (enlever vue2 de la vue sur laquelle elle repose : `rootViewController.view`) qui va enlever vue2 ligne ①. Ensuite, on ajoute ligne ② la vue1 à la vue du `rootViewController` grâce à la méthode `addSubview:`.

C'est ici qu'il peut y avoir une confusion. En effet, en enlevant la vue2, on enlève également le bouton et le label. Pourquoi ? Tout simplement parce que le bouton et le label sont ajoutés à la vue2. Ainsi, il faut préciser que la notion de pile de vues comprend des vues avec des sous-vues potentielles.

Finalement, le label et le bouton appartiennent à la vue2. En fait, pour chaque vue, on peut retrouver ses sous-vues grâce à `NSArray *sousVues = [maVue subviews];`. De fait, un `removeFromSuperview` enlève la vue et toutes ses sous-vues de la pile.

De même, `[maVue superview]` renvoi à la vue sur laquelle est posée `maVue`.

Info

Notez que l'on peut également ajouter une vue dans la pile grâce à `insertSubview: atIndex:`. Ainsi, on peut choisir à quelle position insérer la vue dans la pile. En effet, à chaque vue dans la pile correspond un index. Par exemple, dans `vue1`, le bouton est à l'index 0, le label à l'index 1 dans la sous-pile de `vue1`.

Vous pouvez également déplacer une vue dans la pile, avec les méthodes suivantes :

- `bringSubviewToFront:` Met la vue passée en paramètre au-dessus de la pile de vues.
- `sendSubviewToBack:` Met la vue passée en paramètre en dessous de la pile de vues.
- `insertSubview:aboveSubview:` Insère la vue passée en paramètre 1 au-dessus de la vue passée en paramètre 2.
- `insertSubview:belowSubview:` Insère la vue passée en paramètre 1 en dessous de la vue passée en paramètre 2.
- `exchangeSubviewAtIndex:withSubviewAtIndex:` Échange les vues en fonction de leur index.

Enfin, notez que ligne ③, on teste si la vue2 a déjà été allouée ou non. Si ce n'est pas le cas, on le fait. La raison en est très simple : par défaut, on affiche la vue1. Mais rien ne garantit que l'utilisateur va afficher la vue2. Dans ce cas, pourquoi allouer de la mémoire pour un objet qui ne sera pas utilisé ? C'est une **règle d'or** que vous devrez respecter par la suite : éviter tout chargement mémoire inutile !

RÔLE DE LA FICHE ET SUITE

Cette fiche a un rôle pédagogique. Cependant, vous serez probablement amené par la suite à recourir à un `rootViewController` qui sera un objet de gestion des contrôleurs au-dessus. Pour cela, imaginez ce cas concret : une application avec un menu (`MenuViewController`), une vue de jeu (`JeuViewController`) et une vue de paramètres (`ParametresViewController`). Pour naviguer facilement entre ses vues, une solution consiste à utiliser un `RootViewController`, qui servira à changer les vues entre elles. Prenons ce cas et réalisons-le.

Vous allez donc construire cette interface. Commencez par créer un projet Window-based Application que vous nommerez `UtiliserUnRoot`. Ajoutez ensuite quatre view controllers : `RootViewController`, `JeuViewController`, `MenuViewController` et `ParametresViewController`.

Ajoutez la vue du rootViewController à la window dans l'application delegate, comme au début de cette fiche. Essayez maintenant de construire le schéma suivant :

1. Lorsque le rootViewController apparaît, ajoutez la vue du menu.
2. Ajoutez dans la vue du menu view controller deux boutons qui serviront à se rendre sur la vue du jeu ou la vue des paramètres, ainsi qu'un label affichant le nom de la vue courante.
3. Dans les vues des deux contrôleurs de Jeu et Paramètres, ajoutez un bouton Retour au menu ainsi qu'un label indiquant le nom de la vue courante.

Notez que ce sera le RootViewController qui possédera les trois contrôleurs Jeu, Menu, et Paramètres. C'est un très bon exercice, qui vous permettra de vérifier vos acquis.

Voici ce que vous avez dû construire (je vous laisse construire ParametresViewController sur le modèle de JeuViewController) :

```
RootViewController.h
#import <UIKit/UIKit.h>
#import "MenuViewController.h"
#import "JeuViewController.h"
#import "ParametresViewController.h"

@class MenuViewController;
@class JeuViewController;
@class ParametresViewController;

@interface RootViewController : UIViewController {

    MenuViewController *menuViewController;
    JeuViewController *jeuViewController;
    ParametresViewController *parametresViewController;

}

- (void) changePourJeu;
- (void) changePourParametres;
@end

RootViewController.m
#import "RootViewController.h"
@implementation RootViewController

- (void) loadView {
    self.view = [[[UIView alloc] initWithFrame:[[UIScreen mainScreen]
        applicationFrame]] autorelease];
    self.view.backgroundColor = [UIColor whiteColor];
    menuViewController = [[MenuViewController alloc] init];
    [self.view addSubview:menuViewController.view];
}
```

```
}
```

```
(void) changePourJeu {
```

```
}
```

```
- (void) changePourParametres {
```

```
}
```

```
- (void) dealloc {
```

```
    [menuViewController release];
```

```
    [jeuViewController release];
```

```
    [parametresViewController release];
```

```
    [super dealloc];
```

```
}
```

```
MenuViewController.h
```

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

```
#import "UtiliserUnRootAppDelegate.h"
```

```
@interface MenuViewController : UIViewController {
```

```
    UILabel *labelStatut;
```

```
}
```

```
@end
```

```
MenuViewController.m
```

```
#import "MenuViewController.h"
```

```
@implementation MenuViewController
```

```
- (void) loadView {
```

```
    self.view = [[[UIView alloc] initWithFrame:CGRectMake(0, 0, 320, 460)]
```

```
    ↪ autorelease];
```

```
    self.view.backgroundColor = [UIColor whiteColor];
```

```
    UIButton *boutonJeu = [UIButton buttonWithType:UIButtonTypeRoundedRect];
```

```
    [boutonJeu setFrame:CGRectMake(20, 20, 160, 30)];
```

```
    [boutonJeu addTarget:self action:@selector(changePourJeu:)
```

```
    ↪ forControlEvents:UIControlEventTouchUpInside];
```

```
    [boutonJeu setTitle:@"Aller dans jeu" forState:UIControlStateNormal];
```

```
    [self.view addSubview:boutonJeu];
```

```
    UIButton *boutonParametres = [UIButton buttonWithType:UIButtonTypeRoundedRect];
```

```
    [boutonParametres setFrame:CGRectMake(20, 60, 200, 30)];
```

```
    [boutonParametres addTarget:self action:@selector(changePourParametres:)
```

```
    ↪ forControlEvents:UIControlEventTouchUpInside];
```

```
    [boutonParametres setTitle:@"Aller dans les paramètres"
```

```
    ↪ forState:UIControlStateNormal];
```

```
    [self.view addSubview:boutonParametres];
```

```

labelStatut = [[UILabel alloc] initWithFrame:CGRectMake(30, 30, 260, 30)];
[labelStatut setCenter:self.view.center];
labelStatut.text = @"On est sur le menu";
labelStatut.textColor = [UIColor blueColor];
[self.view addSubview:labelStatut];
}

- (void) changePourJeu : (id) sender {
    UtiliserUnRootAppDelegate *appDelegate =
    ▶ (UtiliserUnRootAppDelegate*)[[UIApplication sharedApplication] delegate];
    [appDelegate.rootViewController changePourJeu];
}

- (void) changePourParametres : (id) sender {
    UtiliserUnRootAppDelegate *appDelegate =
    ▶ (UtiliserUnRootAppDelegate*)[[UIApplication sharedApplication] delegate];
    [appDelegate.rootViewController changePourParametres];
}

- (void) dealloc {
    [labelStatut release];
    [super dealloc];
}
@end
JeuViewController.h
#import <UIKit/UIKit.h>
#import "UtiliserUnRootAppDelegate.h"

@interface JeuViewController : UIViewController {
    UILabel *labelStatut;
}
@end
JeuViewController.m
#import "JeuViewController.h"
@implementation JeuViewController

- (void) loadView {
    // attention à la barre de statuts qui ne fait effet que dans la vue du
    ▶ rootViewController, vu que self.view est une subview de rootViewController
    self.view = [[[UIView alloc] initWithFrame:CGRectMake(0, 0, 320, 460)]
    ▶ autorelease];
    self.view.backgroundColor = [UIColor whiteColor];

    UIButton *boutonMenu = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [boutonMenu setFrame:CGRectMake(20, 20, 160, 30)];
}

```

```
[boutonMenu addTarget:self action:@selector(changePourMenu):
    ↪ forControlEvents:UIControlEventTouchUpInside];
[boutonMenu setTitle:@"Retour au menu" forState:UIControlStateNormal];
[self.view addSubview:boutonMenu];

labelStatut = [[UILabel alloc] initWithFrame:CGRectMake(30, 30, 260, 30)];
[labelStatut setCenter:self.view.center];
labelStatut.text = @"On est sur le jeu";
labelStatut.textColor = [UIColor redColor];
[self.view addSubview:labelStatut];
}

- (void) changePourMenu : (id) sender {
    UtiliserUnRootAppDelegate *appDelegate =
    ↪ (UtiliserUnRootAppDelegate*)[[UIApplication sharedApplication] delegate];
    // on appelle changePourJeu qui va nous permettre de revenir au menu
    [appDelegate.rootViewController changePourJeu];
}

- (void)dealloc {
    [labelStatut release];
    [super dealloc];
}
@end
```

Il ne vous aura pas échappé que nous n'avons écrit aucun code enlevant une vue pour en remettre une autre. Faisons-le pour JeuViewController en implémentant la méthode changePourJeu dans RootViewController.m. Dans cette méthode, nous allons regarder quelle vue du menu ou du jeu s'affiche. Ensuite, nous enlèverons la vue affichée pour mettre l'autre à la place. Vous allez voir, c'est un peu subtil...

```
- (void) changePourJeu {
    if (!jeuViewController) {
        // rappelez-vous, on n'alloue que si l'on a besoin
        jeuViewController = [[JeuViewController alloc] init];
    }

    // on fait référence sur les vues de menu et jeu
    UIView *menuView = menuViewController.view;
    UIView *jeuView = jeuViewController.view;

    if([menuView superview] != nil)
    ↪// si menuView repose sur une vue, alors on l'enlève ①
    {
        [jeuViewController viewWillDisappear:NO]; ②
        [menuViewController viewWillAppear:NO];
        [menuView removeFromSuperview];
        [self.view addSubview:jeuView];
    }
}
```

```

        [jeuViewController viewWillAppear:NO];
        [menuViewController viewDidDisappear:NO];
    }
    else // sinon, c'est jeuView que l'on enlève
    {
        [menuViewController viewDidAppear:NO];
        [jeuViewController viewWillDisappear:NO];
        [jeuView removeFromSuperview];
        [self.view addSubview:menuView];
        [menuViewController viewWillAppear:NO];
        [jeuViewController viewDidDisappear:NO];
    }
}

```

Pour savoir quelle vue enlever, et quelle vue remettre, on teste ligne ① si la superview de menu ne pointe pas vers nil (pointeur nul). En fait, cela revient à tester si la vue du menu est dans la pile de vue.

Ligne ②, on appelle explicitement les méthodes `viewWillAppear:(BOOL)animated`, `viewWillDisappear:(BOOL)animated...`. On spécifie non pour l'animation car la transition se fera sans animations. Compilez et lancez votre application pour tester. Faites de même pour afficher la vue de paramètres.

Pour finir, nous allons ajouter une animation de type *flipside* (retournement de 180°) à l'aide de méthodes de haut niveau de Core Animation. Modifiez la méthode ci-dessus pour obtenir :

```

- (void) changePourJeu {
    if (!jeuViewController) {
        // rappelez-vous, on n'alloue que si l'on a besoin
        jeuViewController = [[JeuViewController alloc] init];
    }

    // on fait référence sur les vues de menu et jeu
    UIView *menuView = menuViewController.view;
    UIView *jeuView = jeuViewController.view;

    [UIView beginAnimations:@"flipAnimation" context:nil]; // réinitialiser le layer
    [UIView setAnimationDuration:0.5]; // durée de l'animation
    if([menuView superview] != nil) // si menuView repose sur une vue, alors on l'enlève
    {
        [jeuViewController viewDidAppear:YES];
        [menuViewController viewWillDisappear:YES];
        [UIView setAnimationTransition:UIViewAnimationTransitionFlipFromLeft
        ↪ forView:self.view cache:YES]; // type de l'animation
        [menuView removeFromSuperview];
        [self.view addSubview:jeuView];
        [jeuViewController viewWillAppear:YES];
        [menuViewController viewDidDisappear:YES];
    }
}

```

```
else // sinon, c'est jeuView que l'on enlève
{
    [menuViewController viewWillAppear:YES];
    [jeuViewController viewWillDisappear:YES];
    [UIView setAnimationTransition:UIViewAnimationTransitionFlipFromRight
    ↪ forView:self.view cache:YES];
    [jeuView removeFromSuperview];
    [self.view addSubview:menuView];
    [menuViewController viewDidAppear:YES];
    [jeuViewController viewDidDisappear:YES];
}
[UIView commitAnimations]; // lancer l'animation
}
```

Info

L'autre animation disponible est `UIViewControllerAnimatedTransitionCurveUp` ou `UIViewControllerAnimatedTransitionCurveDown`. Cette animation permet de simuler un retournement de page.

À la fiche précédente, nous avons vu comment architecturer la navigation entre vues autour d'un `RootViewController`. Il existe d'autres méthodes pour afficher vos vues.

Vous allez commencer par une méthode qui permet très simplement de naviguer entre deux vues, ou plusieurs lorsque la hiérarchie reste simple. Cette méthode repose sur la notion de protocole et delegate, il faut donc préalablement lire la Fiche 11. Dans un second temps, nous apprendrons à nous servir d'un contrôleur de navigation (`UINavigationController`) qui permet de mettre en place très rapidement une hiérarchie de vues compliquée.

Typiquement, le choix d'une méthode pour réaliser votre navigation dépend :

- De la complexité de la hiérarchie des vues et des relations entre vues elles-mêmes. En effet, si la hiérarchie se développe progressivement, comme dans l'application Réglages, un contrôleur de navigation est très adapté. À l'inverse, si les transitions entre vues se résument à échanger la vue principale avec une vue de réglages, alors la première méthode de cette fiche conviendra amplement. Pour finir, si votre application présente des relations entre vues (typiquement un jeu vidéo avec la vue du jeu, du menu, des préférences...) un `RootViewController` sera bien adapté.
- Du contrôle sur les animations des vues. Nous l'avons vu, avec un `RootViewController`, vous pouvez facilement personnaliser vos animations pour échanger les vues (les méthodes d'`UIView` ou celles plus bas niveau de Core Animation avec les layers que nous verrons à la Fiche 26). Les deux autres méthodes quant à elles proposent des animations prédéfinies.

UTILISER UN DELEGATE POUR NAVIGUER SIMPLEMENT

Pour commencer, nous allons réaliser une navigation entre deux vues et on choisira parmi quatre types d'animations prédéfinies. La Figure 23.1 reproduit le schéma de ce que l'on souhaite réaliser.

Créez un nouveau projet de type Window-based Application que vous nommerez Navigation. Ajoutez ensuite deux fichiers de type `UIViewController` et cochez With XIB for user interface pour éviter de créer la vue de chaque contrôleur manuellement. Vous nommerez ces fichiers `MainViewController` et `SecondViewController`.

Qu'allons-nous faire ? Dans un premier temps, nous allons modifier notre application delegate pour afficher la vue du `MainViewController`. Ensuite, nous afficherons la vue du `SecondViewController` avec la méthode dont la signature est `presentModalViewController:animated:`. Enfin, nous créerons un protocole delegate dans `SecondViewController` pour informer le `MainViewController` que la vue de `SecondViewController` souhaite se retirer.

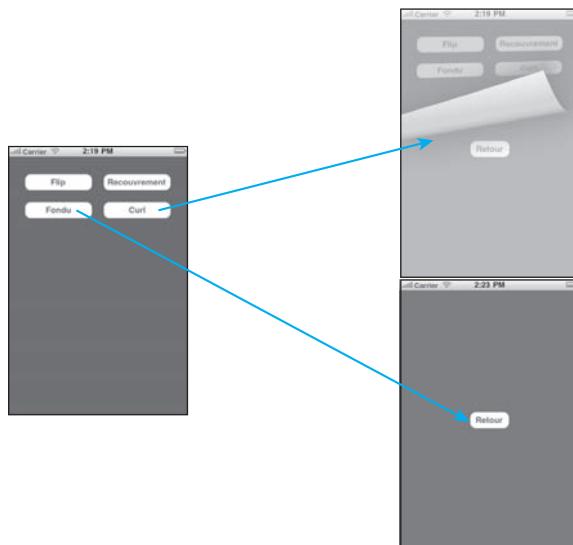


Figure 23.1 : Le but de cette première partie.

Commençons donc par notre application delegate comme à notre habitude :

```
NavigationAppDelegate.h
#import <UIKit/UIKit.h>
@class MainViewController;

@interface NavigationAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    MainViewController *mainViewController;
}
@property (nonatomic, retain) IBOutlet UIWindow *window;
@end

NavigationAppDelegate.m
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    mainViewController = [[MainViewController alloc]
        initWithNibName:@"MainViewController" bundle:nil];
    mainViewController.view.frame = [[UIScreen mainScreen] applicationFrame];
    [window addSubview:mainViewController.view];
    [window makeKeyAndVisible];

    return YES;
}
```

N'oubliez pas le dealloc !

Occupons-nous du protocole delegate de SecondViewController, très simplement lui aussi, si vous avez bien lu la Fiche 11 :

```
SecondViewController.h
#import <UIKit/UIKit.h>
@protocol SecondViewControllerDelegate;

@interface SecondViewController : UIViewController {
    id<SecondViewControllerDelegate> delegate;
}

@property (nonatomic, assign) id<SecondViewControllerDelegate> delegate;
@end

@protocol SecondViewControllerDelegate
- (void)secondViewControllerDidFinish:(SecondViewController *)controller;
@end

SecondViewController.m
@synthesize delegate;

- (void)viewDidLoad {
    // on met une couleur pour bien voir les animations
    [self.view setBackgroundColor:[UIColor redColor]];

    // construction du bouton retour
    UIButton *theButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [theButton addTarget:self action:@selector(retourToMain:)
    ↪ forControlEvents:UIControlEventTouchUpInside];
    [theButton setFrame:CGRectMake(0, 0, 70, 30)];
    [theButton setCenter:self.view.center];
    [theButton setTitle:@"Retour" forState:UIControlStateNormal];
    [self.view addSubview:theButton];
    [super viewDidLoad];
}

- (void)retourToMain:(id)sender {
    // informe le delegate que l'on souhaite se retirer
    [self.delegate secondViewControllerDidFinish:self];
}
```

Pour finir, nous allons nous occuper de MainViewController. Il faut tout d'abord qu'il soit conforme au protocole SecondViewControllerDelegate. Pour cela, dans le .h :

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

@interface MainViewController : UIViewController <SecondViewControllerDelegate> {
}
@end
```

Et ajoutez dans le .m :

```
// Méthode delegate de SecondViewController  
- (void)secondViewControllerDidFinish:(SecondViewController *)controller {  
}
```

Modifiez le viewDidLoad pour afficher une couleur bleue :

```
// on met une couleur pour bien voir les animations  
[self.view setBackgroundColor:[UIColor blueColor]];
```

Pour vous montrer les possibilités des animations, nous allons utiliser les quatre. Il faudra donc créer quatre boutons ! Les animations sont les suivantes :

- **UIModalTransitionStyleFlipHorizontal** Cette animation permet une rotation de 180° suivant un axe vertical. Elle est typiquement lancée dans le lecteur de musique de votre appareil quand vous passez d'une pochette d'album aux titres de l'album.
- **UIModalTransitionStyleCoverVertical** Anime l'arrivée de la deuxième vue en la faisant glisser par-dessus la première en arrivant par le bas.
- **UIModalTransitionStyleCrossDissolve** Cette animation crée un fondu.
- **UIModalTransitionStylePartialCurl** Toute nouvelle, cette animation était auparavant disponible mais non documentée. Elle permet de présenter une vue comme dans l'application Plans lorsque l'on choisit la vue satellite, carte ou mixte.

Modifiez le viewDidLoad pour ajouter les quatre boutons :

```
// flip  
UIButton *flipButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];  
[flipButton addTarget:self action:@selector(flipToSecondView:)  
    forControlEvents:UIControlEventTouchUpInside];  
[flipButton setFrame:CGRectMake(30, 30, 120, 30)];  
[flipButton setTitle:@"Flip" forState:UIControlStateNormal];  
[self.view addSubview:flipButton];  
  
// Recouvrement vertical  
UIButton *coverButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];  
[coverButton addTarget:self action:@selector(coverToSecondView:)  
    forControlEvents:UIControlEventTouchUpInside];  
[coverButton setFrame:CGRectMake(170, 30, 120, 30)];  
[coverButton setTitle:@"Recouvrement" forState:UIControlStateNormal];  
[self.view addSubview:coverButton];  
  
// Fondu  
UIButton *dissolveButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];  
[dissolveButton addTarget:self action:@selector(dissolveToSecondView:)  
    forControlEvents:UIControlEventTouchUpInside];  
[dissolveButton setFrame:CGRectMake(30, 80, 120, 30)];  
[dissolveButton setTitle:@"Fondu" forState:UIControlStateNormal];  
[self.view addSubview:dissolveButton];
```

```
// Recouvrement partiel
UIButton *curlButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[curlButton addTarget:self action:@selector(partialCurlToSecondView:)
    forControlEvents:UIControlEventTouchUpInside];
[curlButton setFrame:CGRectMake(170, 80, 120, 30)];
[curlButton setTitle:@"Curl" forState:UIControlStateNormal];
[self.view addSubview:curlButton];
```

Puis ajoutez ces méthodes :

```
- (void) flipToSecondView:(id)sender {
    [self goToSecondView:UIModalTransitionStyleFlipHorizontal];
}
- (void) coverToSecondView:(id)sender {
    [self goToSecondView:UIModalTransitionStyleCoverVertical];
}
- (void) dissolveToSecondView:(id)sender {
    [self goToSecondView:UIModalTransitionStyleCrossDissolve];
}
- (void) partialCurlToSecondView:(id)sender {
    [self goToSecondView:UIModalTransitionStylePartialCurl];
}
```

Pour afficher la vue, nous allons recourir à la méthode de la classe UIViewController :`presentModalViewControllerAnimated:`. Pour enlever ce contrôleur, il faudra faire appel à `dismissViewControllerAnimated:`. Ajoutez cette méthode (la déclarer également dans le .h) :

```
- (void)goToSecondView:(UIModalTransitionStyle)transitionStyle {
    // allocation d'une instance de SecondViewController
    SecondViewController *controller = [[SecondViewController
        alloc] initWithNibName:@"SecondViewController" bundle:nil];
    // self est le delegate
    controller.delegate = self;
    // on choisit l'animation selon le paramètre
    controller.modalTransitionStyle = transitionStyle;
    // on affiche controller
    [self presentModalViewController:controller animated:YES];
    [controller release];
}
```

Et modifiez la méthode delegate dans MainViewController :

```
// Méthode delegate de SecondViewController
(void)secondViewControllerDidFinish:(SecondViewController *)controller
{
    [self dismissModalViewControllerAnimated:YES];
}
```

Et voilà ! Compilez et lancez pour observer les différentes animations !

NAVIGUER TOUT EN ÉTANT CONTRÔLÉ...

Passons maintenant à l'utilisation d'un contrôleur de navigation. Voici le schéma de navigation que nous souhaitons atteindre Figure 23.2. Notez que l'on pourrait très facilement mettre en place une navigation du type de la Figure 23.3.

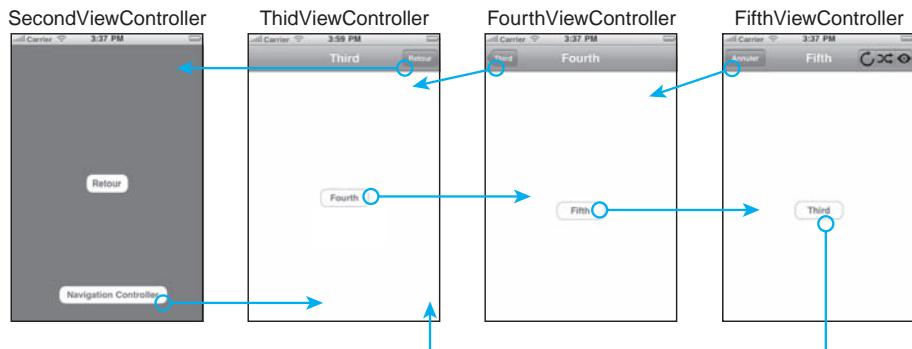


Figure 23.2 : La navigation souhaitée.

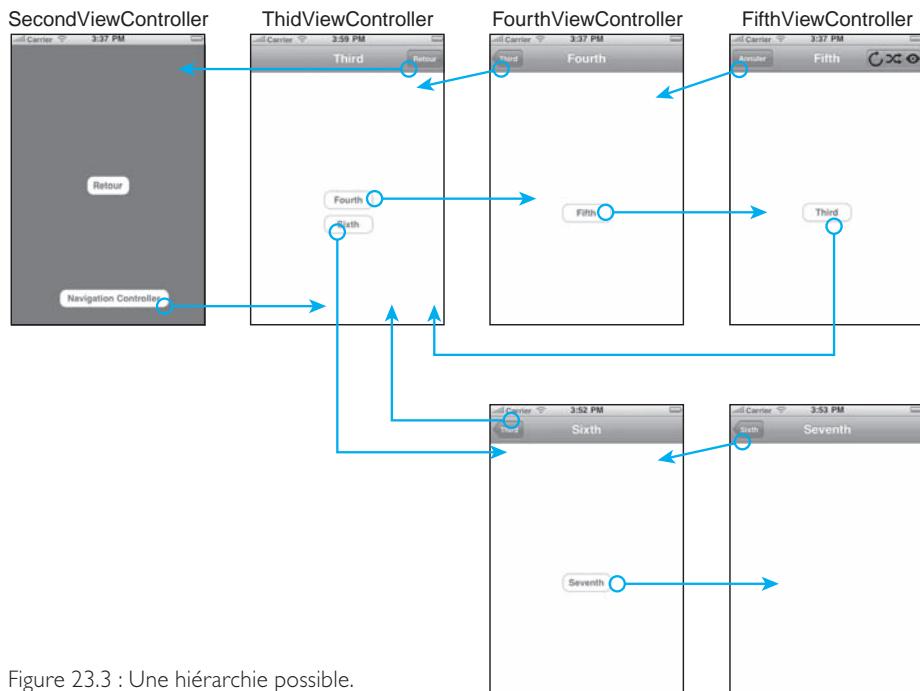


Figure 23.3 : Une hiérarchie possible.

Pour commencer, créez trois nouveaux fichiers de type `UIViewController` que vous nommerez `ThirdViewController`, `FourthViewController` et `FifthViewController`.

Dans un premier temps, nous allons créer un bouton dans `SecondViewController` pour afficher le `ThirdViewController`. Ajoutez ces lignes dans le `viewDidLoad`:

```
// construction du bouton pour aller dans le navigation controller  
UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];  
[button addTarget:self action:@selector(goToThirdView:) forControlEvents:UIControlEventTouchUpInside];  
[button setFrame:CGRectMake(80, 400, 180, 30)];  
[button setTitle:@"Navigation Controller" forState:UIControlStateNormal];  
[self.view addSubview:button];
```

Ensuite, nous allons implémenter `goToThirdView:` grâce à `presentModalViewControllerAnimated:`. Cependant, nous n'allons pas afficher directement la vue de `SecondViewController`, mais nous allons créer un contrôleur de navigation que nous initialiserons avec une instance de `SecondViewController`, puis nous afficherons sa vue. Notez que par défaut, l'animation est *Cover vertical*.

Déclarez ceci dans `SecondViewController.h`:

```
UINavigationController *navigationController;  
ThirdViewController *thirdViewController;
```

Et n'oubliez pas d'ajouter

```
@class ThirdViewController;
```

Ensuite, implémentez cette méthode dans le `.m`:

```
- (void) goToThirdView:(id)sender {  
    if (!thirdViewController) ①  
        thirdViewController = [[ThirdViewController alloc]  
            initWithNibName:@"ThirdViewController" bundle:nil];  
  
    if (!navigationController) ②  
        navigationController = [[UINavigationController alloc]  
            initWithRootViewController:thirdViewController];  
  
    [self presentModalViewControllerAnimated:navigationController animated:YES];  
}
```

Lignes ① et ②, nous testons si les objets ont déjà été alloués pour éviter à la fois des fuites, et des allocations de variables locales inutiles si nous ne les avions pas déclarées dans le `.h`.

Maintenant que notre `SecondViewController` s'affiche (vous pouvez le vérifier en compilant) nous aimeraisons mettre un titre à notre barre de navigation. Pour cela, rien de plus simple : ajoutez cette ligne dans le `viewDidLoad` de `ThirdViewController`.

```
self.title = @"Third";
```

Une navigation bar compte trois sections différentes comme nous le voyons Figure 23.4.

- La section de gauche affiche par défaut le bouton retour au précédent contrôleur dans la pile de navigation. Si on choisit d'afficher une vue à la place, il faut employer `leftBarButtonItem`.

backBarButtonItem
affiché par défaut avec le titre du précédent
contrôleur, sinon c'est leftBarButtonItem

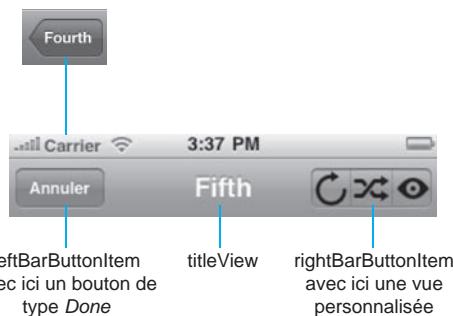


Figure 23.4 : Structure d'une navigation bar.

- La section du centre affiche par défaut le titre du contrôleur actuellement affiché.
- La section de droite n'affiche rien par défaut. Il faut donc se servir de rightBarButtonItem.

Affichons donc un bouton Retour. En effet, ThirdViewController étant le premier view controller de la pile, il faut pouvoir revenir au SecondViewController qui lui n'a strictement rien à voir avec notre structure de contrôleur de navigation. Pour cela, ajoutez dans le viewDidLoad :

```
// bouton done
UIBarButtonItem *doneButton = [[UIBarButtonItem alloc]
                                initWithTitle:@"Retour"
                                style:UIBarButtonItemStyleDone
                                target:self
                                action:@selector(returnToSecond:)];
self.navigationItem.rightBarButtonItem = doneButton;
[doneButton release];
```

Puis ajoutez cette méthode pour revenir au SecondViewController :

```
- (void) returnToSecond:(id)sender {
    [self dismissModalViewControllerAnimated:YES];
}
```

Enfin, pour vous rendre dans la vue de FourthViewController, ajoutez un bouton dans le viewDidLoad :

```
// construction du bouton pour aller dans la quatrième vue
UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[button addTarget:self action:@selector(goToFourthView:)
forControlEvents:UIControlEventTouchUpInside];
[button setFrame:CGRectMake(0, 0, 80, 30)];
[button setCenter:self.view.center];
[button setTitle:@"Fourth" forState:UIControlStateNormal];
[self.view addSubview:button];
```

Pour nous rendre dans cette vue, nous passons par une méthode de UINavigationController : pushViewController:animated:. Ajoutez cette méthode :

```

- (void) goToFourthView:(id)sender {
    FourthViewController *viewController = [[FourthViewController alloc]
    ↪ initWithNibName:@"FourthViewController" bundle:nil];
    [self.navigationController pushViewController:viewController animated:YES];
    [viewController release];
}

```

N'oubliez pas l'import de FourthViewController dans le .m !

Nous allons construire quasiment à l'identique FourthViewController (nous n'ajouterons pas de bouton Retour mais le laisserons par défaut, le backBarButtonItem).

```

- (void)viewDidLoad {
    self.title = @"Fourth";

    // construction du bouton pour aller dans la cinquième vue
    UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [button addTarget:self action:@selector(goToFifthView:)
    ↪ forControlEvents:UIControlEventTouchUpInside];
    [button setFrame:CGRectMake(0, 0, 80, 30)];
    [button setCenter:self.view.center];
    [button setTitle:@"Fifth" forState:UIControlStateNormal];
    [self.view addSubview:button];

    [super viewDidLoad];
}

- (void) goToFifthView:(id)sender {
    FifthViewController *viewController = [[FifthViewController alloc]
    ↪ initWithNibName:@"FifthViewController" bundle:nil];
    [self.navigationController pushViewController:viewController animated:YES];
    [viewController release];
}

```

N'oubliez pas l'import de FifthViewController dans le .m !

Pour finir, construisons la navigation bar de FifthViewController avec un bouton de droite personnalisé avec une barre à segments (UISegmentedControl).

Ajoutez ceci dans le viewDidLoad :

```

- (void)viewDidLoad {
    self.title = @"Fifth";

    // bouton annuler
    UIBarButtonItem *cancelButton = [[UIBarButtonItem alloc]
        initWithTitle:@"Annuler"
        style:UIBarButtonItemStyleDone
        target:self
        action:@selector(returnToFourth:)];
}

```

```

self.navigationItem.leftBarButtonItem = cancelButton;
[cancelButton release];

// Construction d'une vue custom pour la droite de la barre

UISegmentedControl *segmentedControl = [[UISegmentedControl alloc]
➥ initWithItems:[NSArray arrayWithObjects:
[UIImage imageNamed:@"01-refresh.png"],
[UIImage imageNamed:@"05-shuffle.png"],
[UIImage imageNamed:@"12-eye.png"],
nil]];

[segmentedControl addTarget:self action:@selector(segmentAction:)
➥ forControlEvents:UIControlEventValueChanged];
segmentedControl.frame = CGRectMake(0, 0, 90, 30);
segmentedControl.segmentedControlStyle = UISegmentedControlStyleBar;
segmentedControl.momentary = YES;

// création du bouton de droite à partir du segmented control
UIBarButtonItem *segmentBarItem = [[UIBarButtonItem alloc]
➥ initWithCustomView:segmentedControl];
[segmentedControl release];
self.navigationItem.rightBarButtonItem = segmentBarItem;
[segmentBarItem release];

[super viewDidLoad];
}

```

Il ne vous aura pas échappé que nous n'employons pas le bouton de retour par défaut ! C'est en effet pour vous montrer la méthode `popViewControllerAnimated:` pour revenir au précédent contrôleur.

Ajoutez cette méthode :

```

- (void) returnToFourth:(id)sender {
    [self.navigationController popViewControllerAnimated:YES];
}

```

Pour gérer les segments, ajoutez cette méthode :

```

- (void) segmentAction:(id)sender {
    UISegmentedControl *control = (UISegmentedControl*)sender;
    NSLog(@"a appuyé sur %d", control.selectedSegmentIndex);
    // Typiquement, faire un switch case sur le selectedSegmentIndex
}

```

Enfin, pour voir les méthodes `popToViewController:animated:` et `popToRootViewControllerAnimated:` nous allons ajouter un bouton qui va nous permettre de revenir au `ThirdViewController`. Dans le `viewDidLoad`, ajoutez :

```

// construction du bouton pour aller dans la troisième vue.
UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];

```

```
[button addTarget:self action:@selector(goToThirdView:)
    ↵ forControlEvents:UIControlEventTouchUpInside];
[button setFrame:CGRectMake(0, 0, 80, 30)];
[button setCenter:self.view.center];
[button setTitle:@"Third" forState:UIControlStateNormal];
[self.view addSubview:button];
```

Arrêtons-nous un instant sur notre pile de contrôleurs : lorsque `FifthViewController` apparaît, voici ce que l'on obtient avec :

```
NSArray *allControllers = [self.navigationController viewControllers];
Navigation[1566:207] (
    "<ThirdViewController: 0x5a35740>",
    "<FourthViewController: 0x5a38640>",
    "<FifthViewController: 0x5a38e50>"
)
```

Ce qui est somme toute assez logique. Le premier objet de ce tableau est le plus ancien contrôleur de la pile : `ThirdViewController`. Si nous souhaitons revenir à ce premier contrôleur directement depuis `FifthViewController`, plusieurs solutions :

- **Solution 1.** Utiliser l'index du contrôleur à afficher (s'applique du moment que l'on connaît l'index de l'objet dans la pile que l'on souhaite atteindre) :

```
- (void)goToThirdView:(id)sender {
    NSArray *allControllers = [self.navigationController viewControllers];
    [self.navigationController popToViewController:[allControllers
        ↵ objectAtIndex:0] animated:YES];
}
```

- **Solution 2.** Ici `ThirdRootViewController` est le `RootViewController` de notre pile de contrôleurs :

```
- (void)goToThirdView:(id)sender {
    [self.navigationController popToRootViewControllerAnimated:YES];
}
```

- **Solution 3.** La plus générale pour accéder à un contrôleur dont on connaît le nom de la classe :

```
- (void)goToThirdView:(id)sender {
    NSArray *allControllers = [self.navigationController viewControllers];
    UIViewController *theController = nil;
    for(UIViewController *vc in allControllers)
        if ([vc isKindOfClass:[NSClassFromString(@"ThirdViewController") class]]) {
            theController = vc;
            [self.navigationController
                ↵ popToViewController:theController animated:YES];
        }
}
```

Finalement, c'est assez simple ! `Push` pour ajouter un contrôleur dans la pile, `pop` pour se rendre à un contrôleur dans la pile !

Le tab bar est un élément d'interface incontournable lorsque vous souhaitez présenter des données différentes dans une même application. Vos vues seront alors accessibles par une liste d'onglets !

On trouve le tab bar (`UITabBar`) dans l'application iPod, par exemple quand vous choisissez d'afficher votre musique en triant par noms d'artiste, morceaux, albums... La Figure 24.1 montre ce que nous allons réaliser.

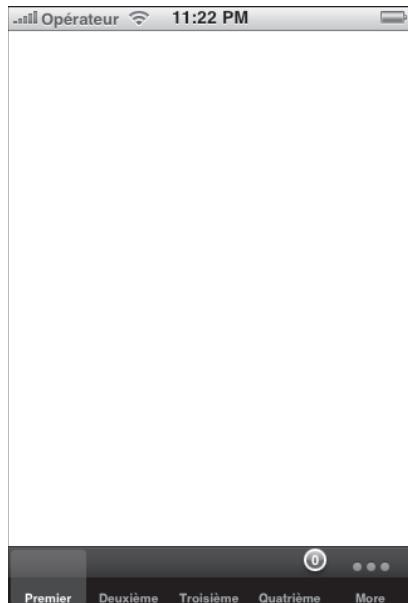


Figure 24.1 : Exemple d'un tab bar.

AFFICHER UN TAB BAR

Commençons par créer un nouveau projet, de type Window-based Application que vous nommerez TabBar. Nous allons tout d'abord afficher trois contrôleurs de vues. Pour cela, ajoutez au projet trois fichiers de type `UIViewController` que vous nommerez respectivement `FirstViewController`, `SecondViewController` et `ThirdViewController` (cochez `With XIB for user interface`).

Pour gérer et afficher le tab bar, nous avons recours à un *tab bar controller* de la classe `UITabBarController`. Pour faire apparaître les différents onglets, il faut ajouter des fichiers au tableau `viewControllers` qui est une propriété d'instance de `UITabBarController`. Le contrôleur à l'index 0 correspond à celui affiché le plus à gauche.

Ajoutez ces lignes dans l'application delegate (.h) :

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

@interface TabBarAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    UITabBarController *tabBarController;
}
@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) UITabBarController *tabBarController;
@end
```

Ensuite, ajoutez cette ligne dans le .m :

```
@synthesize tabBarController;
```

Dans la méthode application:didFinishLaunchingWithOptions;, initialisons notre tab bar controller :

```
self.tabBarController = [[[UITabBarController alloc] init] autorelease];
```

Puis, nous allons initialiser trois contrôleurs de vue. Vous noterez que pour le deuxième contrôleur, nous lui ajouterons un contrôleur de navigation, simplement pour montrer qu'il est possible de les combiner.

Commencez par importer les trois contrôleurs :

```
#import "FirstViewController.h"
#import "SecondViewController.h"
#import "ThirdViewController.h"
```

Ajoutez ces quelques lignes :

```
FirstViewController *firstViewController = [[FirstViewController alloc]
➥ initWithNibName:@"FirstViewController" bundle:nil];
firstViewController.title = @"Premier";

SecondViewController *secondViewController = [[SecondViewController alloc]
➥ initWithNibName:@"SecondViewController" bundle:nil];
UINavigationController *navigationController = [[UINavigationController alloc]
➥ initWithRootViewController:secondViewController];
navigationController.title = @"Deuxième";
[secondViewController release];
[navigationController setNavigationBarHidden:NO];

ThirdViewController *thirdViewController = [[ThirdViewController alloc]
➥ initWithNibName:@"ThirdViewController" bundle:nil];
thirdViewController.title = @"Troisième";
```

Puis, il faut ajouter ces contrôleurs au tab bar :

```
tabBarController.viewControllers = [NSArray arrayWithObjects:firstViewController,
➥ navigationController, thirdViewController, nil];
```

Lorsque les contrôleurs ont été ajoutés, il faut libérer la mémoire puis ajouter la vue du tab bar à la fenêtre :

```
[firstViewController release];
[navigationController release];
[thirdViewController release];

>window addSubview:tabBarController.view];
>window makeKeyAndVisible];
return YES;
```

Info

Par défaut, le titre de chaque onglet sera construit à l'aide de la propriété title de chaque contrôleur. Nous verrons ensuite comment personnaliser cet affichage.

Compilez votre application pour voir apparaître vos trois onglets ! C'est assez facile non ?

SÉLECTIONNER UN ONGLET PAR LE CODE

Parfois, il est important de pouvoir sélectionner un onglet par le code. En effet, si votre application possède un onglet dédié à l'Aide, l'utilisateur peut y accéder seul à tout moment, mais vous pouvez également souhaiter l'y renvoyer si besoin. Pour cela, il faut passer par la propriété selectedViewController ou selectedIndex de votre tab bar contrôleur.

Dans le troisième contrôleur de vue (*ThirdViewController*), nous allons ajouter deux boutons :

```
- (void)viewDidLoad {
    UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [button setFrame:CGRectMake(30, 30, 280, 30)];
    [button setTitle:@"Premier contrôleur de la liste" forState:UIControlStateNormal];
    [button addTarget:self action:@selector(selectFirstViewControllerOfListe:)
        forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:button];

    UIButton *button2 = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [button2 setFrame:CGRectMake(30, 70, 280, 30)];
    [button2 setTitle:@"Premier contrôleur" forState:UIControlStateNormal];
    [button2 addTarget:self action:@selector(selectFirstViewController:)
        forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:button2];
    [super viewDidLoad];
}
```

Commençons par sélectionner un onglet par son index. En fait, on sélectionne l'onglet selon son numéro d'affichage (0 représentant celui de gauche).

```
- (void) selectFirstViewControllerOfListe:(id)sender {
    // référence vers l'application delegate pour récupérer le tab bar controller
```

```

TabBarAppDelegate *appDelegate = (TabBarAppDelegate*)[[UIApplication
    sharedApplication] delegate];
UITabBarController *tabController = appDelegate.tabBarController;
// sélection de l'onglet le plus à gauche
tabController.selectedIndex = 0;
}

```

N'oubliez pas l'import du fichier TabBarAppDelegate.h !

Il faut savoir que vous pouvez changer l'ordre d'affichage de vos onglets, ce que nous verrons par la suite. Sélectionner un onglet par son numéro présente donc des limites, ou vous impose de connaître son index dans le tab bar. Ainsi, pour choisir un contrôleur de vue non plus par l'index de l'onglet mais par le nom du contrôleur, il faut récupérer le contrôleur à partir du nom de sa classe. Aussi, nous allons dans le deuxième bouton sélectionner l'onglet qui est attaché au contrôleur de vue FirstViewController :

```

- (void) selectFirstViewController:(id)sender {
    // référence vers l'application delegate pour récupérer le tab bar controller
    TabBarAppDelegate *appDelegate = (TabBarAppDelegate*)[[UIApplication
        sharedApplication] delegate];
    UITabBarController *tabController = appDelegate.tabBarController;

    // retrouver le contrôleur de la classe FirstViewController
    // on parcourt le tableau des contrôleurs
    UIViewController *controllerToSelect = nil;
    for (UIViewController *controller in tabController.viewControllers)
        if([controller isKindOfClass:[NSClassFromString(@"FirstViewController")]])
    {
        controllerToSelect = controller; break; // on sort de la boucle
    }
    tabController.selectedViewController = controllerToSelect;
}

```

Pour l'instant, le comportement est le même. Nous allons donc ajouter des contrôleurs de vues pour que vous en compreniez l'enjeu !

AFFICHER PLUS DE CINQ CONTRÔLEURS DE VUES

Un tab bar ne peut pas afficher plus de cinq onglets. Zut, je souhaite en afficher plus... Il faut donc faire un choix ? Pas du tout ! De manière automatique, lorsque vous ajouterez plus de cinq contrôleurs de vues dans le tableau viewControllers, le cinquième onglet sera remplacé par More si votre application est en anglais. En cliquant sur cet onglet, vous aurez donc accès à une liste des contrôleurs de vues qui ne peuvent s'afficher. Créez trois nouveaux contrôleurs que vous nommerez FourthViewController, FifthViewController et SixthViewController.

Ajoutez ces lignes dans l'application delegate et modifiez les parties en gras :

```
FourthViewController *fourthViewController = [[FourthViewController alloc]
➥ initWithNibName:@"FourthViewController" bundle:nil];
fourthViewController.title = @"Quatrième";

FifthViewController *fifthViewController = [[FifthViewController alloc]
➥ initWithNibName:@"FifthViewController" bundle:nil];
fifthViewController.title = @"Cinquième";

SixthViewController *sixthViewController = [[SixthViewController alloc]
➥ initWithNibName:@"SixthViewController" bundle:nil];
sixthViewController.title = @"Sixième";

tabBarController.viewControllers = [NSArray arrayWithObjects:firstViewController,
➥ navigationController, thirdViewController, fourthViewController,
➥ fifthViewController, sixthViewController, nil];

[firstViewController release];
[navigationController release];
[thirdViewController release];
[fourthViewController release];
[fifthViewController release];
[sixthViewController release];
```

N'oubliez pas les imports de fichiers, même si vous devez maintenant le faire seul.

Compilez, puis lancez votre application. Vous pourrez cliquer sur More pour accéder aux contrôleurs Cinquième et Sixième. Vous trouverez également un bouton Edit qui modifie l'ordre des vos onglets. Modifiez cet ordre et rendez-vous sur l'onglet nommé Troisième pour bien comprendre la sélection d'un onglet que nous avons vue précédemment.

Info

Par défaut, tous les onglets sont susceptibles d'être modifiés. Vous pouvez choisir ceux modifiables en les ajoutant dans le tableau `customizableViewControllers` (propriété d'instance de la classe `UITabBarController`) qui par défaut reçoit une copie de la propriété `viewControllers`. Pour empêcher toute modification, vous pouvez faire pointer ce tableau vers `nil`.

Et le delegate ?

Vous pouvez également implémenter les méthodes de `UITabBarDelegate` qui vous avertira du début de l'édition, de la fin de l'édition, de la sélection d'un contrôleur... Vous retrouverez ces méthodes dans la documentation de `UITabBarDelegate`.

COMMENT MODIFIER L'ORDRE D'AFFICHAGE DES ONGLETS LORSQU'IL Y EN A CINQ OU MOINS ?

Lorsque vous avez cinq onglets, ou moins, et pas d'onglet More vous permettant d'accéder au bouton d'édition, vous pouvez quand même changer l'ordre d'affichage. Pour cela, vous avez besoin des méthodes d'instance de UITabBar (vous pouvez accéder au tabBar d'un contrôleur avec tabBarController.tabBar) :

- **beginCustomizingItems:(NSArray*)items** Va afficher une vue modale permettant de réarranger les onglets spécifiés en paramètres. Typiquement, vous appellerez cette méthode en appuyant sur un bouton Éditer par exemple.
- **endCustomizingAnimated:(BOOL)animated** Va permettre d'enlever la vue permettant de modifier l'ordre d'affichage. Typiquement, vous n'emploierez pas cette méthode, car le bouton Done ou Terminer de la vue modale le fera pour vous automatiquement.
- **isCustomizing** Renvoie un booléen indiquant si l'utilisateur est en train d'éditer ou non le tab bar.

PERSONNALISER LES ONGLETS

Les onglets sont personnalisables. En effet, par défaut, le titre de l'onglet est celui du contrôleur qu'il représente. Mais vous pouvez personnaliser vos onglets en ajoutant une image. Attention cependant, cette image devra être en format png, faire 30×30 pixels, et être transparente. Lorsque l'onglet est sélectionné, vous remarquerez que l'image s'affiche en bleu. Ce comportement est géré en interne et vous ne pouvez le modifier.

Ajoutons au cinquième contrôleur de vue un onglet avec un type prédéfini. Pour cela, ajoutez les lignes en gras :

```
FifthViewController *fifthViewController = [[FifthViewController alloc]
➥ initWithNibName:@"FifthViewController" bundle:nil];
UITabBarItem *tabBarItemFifth = [[UITabBarItem alloc]
➥ initWithTitle:UITabBarSystemItemTopRated tag:5];
fifthViewController.tabBarItem = tabBarItemFifth;
[tabBarItemFifth release];
fifthViewController.title = @"Cinquième"; // à supprimer
```

Vous avez le choix entre plusieurs types : UITabBarSystemItemTopRated affiche une étoile, UITabBarSystemItemRecents affiche une horloge... Les titres sont également définis par défaut, et vous trouverez les autres types dans la documentation de la classe UITabBarItem.

Il est également possible de faire apparaître votre propre image ! Pour cela, allons sur le site glyphish.com, qui propose un large choix d'icônes gratuites à placer dans vos barres de navigation.

```
SixthViewController *sixthViewController = [[SixthViewController alloc] initWithNibName:@"SixthViewController" bundle:nil];
UITabBarItem *tabBarItemSixth = [[UITabBarItem alloc] initWithTitle:@"Sixième"
➥ image:[UIImage imageNamed:@"17-bar-chart.png"] tag:6];
sixthViewController.tabBarItem = tabBarItemSixth;
[tabBarItemSixth release];
sixthViewController.title = @"Sixième";
```

Pour finir, nous allons voir comment ajouter un badge sur un onglet. Ce badge permet, par exemple, d'afficher le nombre d'articles non lus dans le cadre d'un lecteur de flux RSS. Pour cela, nous allons ajouter un bouton qui incrémentera la valeur du badge dans le contrôleur de vues FourthViewController :

```
- (void)viewDidLoad {
    // on met le badge à la valeur 0
    self.tabBarItem.badgeValue = @"0";

    UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [button setFrame:CGRectMake(30, 30, 100, 30)];
    [button setTitle:@"++" forState:UIControlStateNormal];
    [button addTarget:self action:@selector(addBadgeValue:)
    ↪ forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:button];

    [super viewDidLoad];
}

- (void) addBadgeValue:(id)sender {
    // on récupère la valeur du badge
    NSInteger theInteger = [self.tabBarItem.badgeValue integerValue];
    // on incrémente la valeur
    theInteger++;
    // on affiche cette nouvelle valeur
    self.tabBarItem.badgeValue = [NSString stringWithFormat:@"%d", theInteger];
}
```

Dans cette fiche, nous allons voir comment utiliser les timers (`NSTimer`) pour déplacer une vue représentant une balle sur l'écran. Nous gérerons également les collisions de la balle sur un objet.

Info

Cette fiche est un préambule pour réaliser ce que l'on appelle des *game loop*. C'est en fait une méthode appelée de manière régulière par un timer et qui met à jour des données. Typiquement, dans un jeu vidéo, cette méthode mettra à jour le dessin des personnages, des éléments de décor...

Core Animation est nécessaire dans le cas où vous souhaiteriez simplement déplacer une vue pour ajouter une animation ponctuelle. C'est une question de performance (voir Fiche 26).

Comme à notre habitude, créez un projet de type Window-based Application que vous nommerez TimerBall, et ajoutez-y un `RootViewController`.

Nous allons construire une interface comme à la Figure 25.1. Lancez-vous, et au moindre doute référez-vous à ce code :

```
- (void)viewDidLoad {
    ballImageView = [[UIImageView alloc]
        initWithFrame:CGRectMake(0, 0, LARGEUR_
        HAUTEUR_BALLE, LARGEUR_HAUTEUR_BALLE)];
    ballImageView.image = [UIImage
        imageNamed:@"balle.png"];
    [self.view addSubview:ballImageView];

    wallImageView = [[UIImageView alloc]
        initWithFrame:CGRectMake(130, 110, 70, 110)];
    wallImageView.image = [UIImage
        imageNamed:@"mur_de_briques.jpg"];
    [self.view addSubview:wallImageView];
```

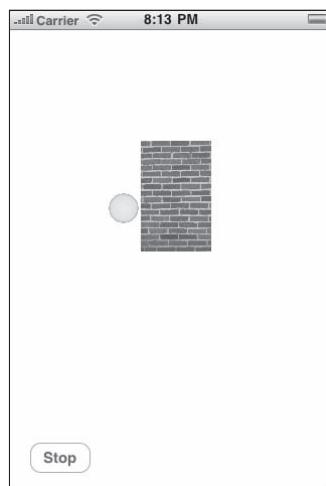


Figure 25.1 : La vue à obtenir.

```
UIButton *startStopButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[startStopButton setTitle:@"Start" forState:UIControlStateNormal];
[startStopButton setTitle:@"Stop" forState:UIControlStateSelected];
[startStopButton addTarget:self action:@selector(startStop:)
    forControlEvents:UIControlEventTouchUpInside];
[startStopButton setFrame:CGRectMake(20, 410, 60, 30)];
[self.view addSubview:startStopButton];

ballVelocity = CGPointMake(7, 3); ①
topSide = wallImageView.frame.origin.y; ②
```

```

bottomSide = wallImageView.frame.origin.y + wallImageView.frame.size.height;
leftSide = wallImageView.frame.origin.x;
rightSide = wallImageView.frame.origin.x + wallImageView.frame.size.width;

[super viewDidLoad];
}

```

Tout ceci a déjà été vu dans les anciennes fiches. Précisons cependant qu'il faut ajouter ces lignes pour définir la taille de la balle ainsi que la fréquence de rafraîchissement :

```

#define LARGEUR_HAUTEUR_BALLE 30.0
#define FREQUENCE_RAFRAICHISSEMENT 30.0 // Hz

```

De plus, on définit ligne ① ce que l'on appelle ballVelocity, qui correspond au déplacement de la balle à chaque appel de la méthode de rafraîchissement par le timer. Ici, la balle se déplacera donc de 7 pixels en x et 3 en y.

Ensuite, pour gérer la collision sur le mur placé sur l'écran, on calcule les points correspondants à chaque coin du mur à partir de la ligne ②.

Le bouton Start/Stop va nous permettre de lancer ou mettre en pause le déplacement de la balle. Pour cela, nous allons le positionner dans l'état sélectionné pour animer la balle, puis désélectionné pour arrêter. Voici à quoi la méthode doit ressembler :

```

- (void) startStop:(id)sender {
    UIButton *but = (UIButton*)sender;
    if (but.selected) { ①
        [refreshTimer invalidate];
        refreshTimer = nil; ②
    }
    else { ③
        refreshTimer = [NSTimer scheduledTimerWithTimeInterval:1/FREQUENCE
            ↪_RAFRAICHISSEMENT target:self selector:@selector(refresh:)
            ↪ userInfo:nil repeats:YES];
    }
    [but setSelected:!but isSelected];
}

```

Pour rappel, voici le .h :

```

#import <UIKit/UIKit.h>

@interface RootViewController : UIViewController {
    UIImageView *ballImageView, *wallImageView;
    CGPoint ballVelocity;
    NSTimer *refreshTimer;

    int topSide, bottomSide, leftSide, rightSide;
}
@end

```

Ligne ①, si le bouton est sélectionné, alors le texte affiché est Stop. Cela signifie donc qu'il faut mettre en pause le déplacement, et donc arrêter le timer avec invalidate.

Info

Faire pointer vers nil le timer ligne ② est un automatisme à prendre lorsque vous libérez des objets. En effet, imaginez un objet unObjet déclaré en variable globale et utilisé à plusieurs endroits dans le code.

Lorsque vous faites un release sur unObjet, unObjet ne pointera pas vers nil (qui est un pointeur nul) mais sur une adresse d'un objet n'existant plus en mémoire. Donc, si vous faites la chose suivante :

```
[unObjet release];
...
[unObjet uneMethode];
```

Vous aurez une erreur et votre application plantera. Par contre, étant donné que les messages vers nil sont ignorés, vous n'aurez aucun bogue en écrivant :

```
[unObjet release];
unObjet = nil;
...
[unObjet uneMethode]; // ne plantera pas
```

Ligne ③, lorsque le bouton affiche Start, on lance le timer qui appellera la méthode de RootViewController spécifié par target:self : - (void)refresh:(NSTimer*)theTimer de manière répétée avec repeats:YES. Cette méthode sera appelée toutes les 1/30 secondes, soit toutes les 0,033 secondes environ.

Info

Vous souhaitez passer un paramètre à la méthode appelée par le timer ? Par exemple, imaginons une méthode de rafraîchissement des graphismes, appelée par un timer et commune à plusieurs vues, trois boutons. Comment spécifier quelle est la vue concernée par le timer ? Il faut utiliser userInfo :

```
refreshTimer = [NSTimer scheduledTimerWithTimeInterval:1/FREQUENCE_RAFRAICHEMENT
target:self selector:@selector(refresh:) userInfo:button1 repeats:YES];
```

avec

```
- (void)refresh:(NSTimer*)theTimer {
    UIButton *boutonAAnimer = (UIButton*)[theTimer userInfo];
    // faire quelque chose avec le bouton
}
```

Astuce

Et pour passer plusieurs paramètres ? Rien de plus simple, utilisez un tableau (NSArray) ou un dictionnaire (NSDictionary) !

Enfin, implémentons la méthode appelée à intervalles réguliers. Nous allons faire deux tests principaux : tester si la balle touche les bords de l'écran puis tester si elle touche le mur. Le but de ces tests est de changer le signe du déplacement en x ou en y. En fait, lorsque la balle touche le bord de l'écran droit ou gauche, il faut la faire rebondir dans son déplacement en x (horizontal). On change donc le signe de ballVelocity.x. De même pour les bords hauts et bas avec ballVelocity.y.

```
- (void) refresh : (NSTimer*)timer {  
    if(ballImageView.center.x - LARGEUR_HAUTEUR_BALLE/2.0 < 0.0 ①  
        ||  
        ballImageView.center.x + LARGEUR_HAUTEUR_BALLE/2.0 > 320.0)  
        ballVelocity.x = -ballVelocity.x;  
    if(ballImageView.center.y - LARGEUR_HAUTEUR_BALLE/2.0 < 0.0  
        ||  
        ballImageView.center.y + LARGEUR_HAUTEUR_BALLE/2.0 > 480.0)  
        ballVelocity.y = -ballVelocity.y;  
  
    //On a besoin de savoir si la nouvelle position touche le mur, mais il faut  
    ➔ garder l'ancienne position pour faire des calculs  
    CGRect newFrame = ballImageView.frame;  
  
    // on calcule la nouvelle position  
    newFrame.origin.x += ballVelocity.x;  
    newFrame.origin.y += ballVelocity.y;  
  
    // Attention, la balle n'a pas encore bougée. On regarde juste si elle  
    ➔ toucherait le mur  
    if (CGRectIntersectsRect(newFrame, wallImageView.frame)) { ②  
        // haut ou bas du mur  
        if (ballImageView.center.y < topSide || ballImageView.center.y > bottomSide)  
            ballVelocity.y = -ballVelocity.y;  
  
        // gauche ou droit du mur  
        if (ballImageView.center.x < leftSide || ballImageView.center.x > rightSide)  
            ballVelocity.x = -ballVelocity.x;  
  
    } else {  
        // On ne bouge la balle que s'il n'y a pas de collision  
        ballImageView.frame = newFrame;  
    }  
}
```

Vous distinguez deux méthodes :

La première, plus naïve à partir de la ligne **①** qui détecte si la balle a touché l'écran de l'iPhone.

La seconde, ligne **②**, quant à elle utilise une méthode de CoreGraphics :CRRectIntersectsRect(CGRect firstRect, CGRect secondRect) qui retourne true si les deux rectangles passés en paramètre se touchent.

Finissons par un peu de théorie sur les timers et leur place dans le système. Les timers fonctionnent avec les objets de la classe NSRunLoop. Ces objets contrôlent les boucles qui attendent une entrée comme un événement sur la souris ou le clavier sur un ordinateur. Les objets de NSRunLoop se servent donc des timers pour déterminer le temps pendant lequel ils doivent attendre. Lorsque ce temps est écoulé, la *run loop* lance le timer et regarde les nouvelles entrées.

Pour résumer, une *run loop* est lancée pour planifier des tâches et coordonner la réception d'événements entrants. En fait, cela permet de garder le thread occupé quand il y a des tâches à réaliser, et de le mettre en veille lorsqu'il n'y a rien à faire.

Info

Il existe une alternative aux timers, par exemple :

```
[self performSelector:@selector(aMethod:) withObject:anObjet afterDelay:delay];
```

Cela permet d'appeler la méthode `aMethod:` après un temps donné par `delay`.

Vous l'aurez compris, il ne faut plus utiliser `sleep(unTemps)` !

Et ne négligez jamais l'implémentation de `dealloc` :

```
- (void)dealloc {  
    [refreshTimer invalidate];  
    refreshTimer = nil;  
    [wallImageView release];  
    [ballImageView release];  
    [super dealloc];  
}
```

Astuce

Ici, le mur est statique. Si vous souhaitez détecter la collision avec un objet en mouvement, il faut inclure dans la méthode du timer le calcul des quatre variables `topSide`, `bottomSide`, `leftSide` et `rightSide`.

Vous pouvez très facilement mettre en place des animations, plus ou moins compliquées, pour faire vivre votre application. En effet, faire arriver ses boutons en glissant, réaliser des fondus, des déplacements de vues peut rendre votre application plus chaleureuse avec seulement quelques lignes de code !

Dans un premier temps, nous allons nous intéresser aux méthodes de haut niveau de Core Animation (ces méthodes sont incluses dans la classe `UIView`). Ensuite, nous réaliserons une sorte d'introduction au bas niveau de ce framework.

Tout d'abord, soyez attentifs au fait que Core Animation n'est pas de l'OpenGL. Même si vous trouvez des animations de déplacement en 3D, ce n'est pas un moteur de rendu 3D. Le framework de cette fiche vous sera très utile pour créer des animations simples ou non. Par contre, recourir à cette technologie pour réaliser un jeu complet pourrait se révéler être un challenge !

Core Animation est un sujet très vaste, que nous ne pourrons pas développer entièrement dans cet ouvrage. Considérez donc cette fiche comme une initiation et reportez-vous à la documentation pour approfondir.

UTILISER DES ANIMATIONS DE HAUT NIVEAU

CRÉATION DU PROJET

Créez un nouveau projet de type Window-based Application que vous nommerez `Animations`. Ajoutez ensuite deux fichiers de type `UIViewController` que vous nommerez `RootViewController` et `UIViewAnimations`. Comme à notre habitude, affichez la vue du `RootViewController` par-dessus la window dans notre application delegate. Ensuite, dans `RootViewController`, nous allons ajouter un bouton pour afficher notre vue de `UIViewAnimations`. Perdu ?

```
RootViewController.h
#import <UIKit/UIKit.h>
@class UIViewAnimations;
@interface RootViewController : UIViewController {
    UIViewAnimations *viewAnimations;
}
@end

RootViewController.m
#import "RootViewController.h"
#import "UIViewAnimations.h"

@implementation RootViewController

- (void)viewDidLoad {
    UIButton *buttonForViewAnimations =
        [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [buttonForViewAnimations setFrame:CGRectMake(30, 30, 130, 30)];
    [buttonForViewAnimations setTitle:@"View animations"]
}
```

```

    ↪ forState:UIControlStateNormal];
[buttonForViewAnimations addTarget:self action:@selector(displayViewAnimations:)
    ↪ forControlEvents:UIControlEventTouchUpInside];
[self.view addSubview:buttonForViewAnimations];

[super viewDidLoad];
}

- (void)displayViewAnimations:(id)sender {
    if(!viewAnimations)
        viewAnimations = [[UIAnimations alloc]
            ↪ initWithNibName:@"UIAnimations" bundle:nil];
    [self.view addSubview:viewAnimations.view];
}

- (void)dealloc {
    [viewAnimations release];
    [super dealloc];
}

```

Info

Pas de panique, on ne gère pas ici le retour de la vue viewAnimations, ce n'est pas le propos !

Dans UIAnimations, nous allons gérer plusieurs animations :

- une animation d'une vue qui va changer de couleur de manière aléatoire et en douceur ;
- une animation d'une vue pour changer sa transparence ;
- une animation d'un bouton qui tremble ;
- une animation de zoom d'une image ;
- une animation de déplacement d'une vue.

Commencez par déclarer les variables nécessaires dans UIAnimations.h :

```

#import <UIKit/UIKit.h>
@interface UIAnimations : UIViewController {
    UIView *viewForColorAnimation, *viewForAlphaAnimation;
    UIButton *buttonForWiggleAnimation;
    UIImageView *imageViewForZoomAnimation;
    UIView *viewForMultiplesAnimations;

    NSTimer *timerForColorAnimation;

    BOOL canAnimate;
}
@end

```

CHANGER LES COULEURS

Pour cette animation, nous allons utiliser un timer qui appellera toutes les 4 secondes une méthode qui animera les transitions de couleurs. Commençons par déclarer un bouton de Start/Stop pour commencer/arrêter les animations et la vue qui changera de couleur. Dans le viewDidLoad, ajoutez :

```
// vue pour l'animation de la couleur
viewForColorAnimation = [[UIView alloc] initWithFrame:CGRectMake(30, 30, 60, 60)];
[self.view addSubview:viewForColorAnimation];

UIButton *buttonToLaunchStopAnimations =
    [UIButton buttonWithType:UIButtonTypeRoundedRect];
[buttonToLaunchStopAnimations setFrame:CGRectMake(100, 440, 70, 30)];
[buttonToLaunchStopAnimations addTarget:self action:@selector(launchStopAnim:)
    forControlEvents:UIControlEventTouchUpInside];
[buttonToLaunchStopAnimations setTitle:@"Start" forState:UIControlStateNormal];
[buttonToLaunchStopAnimations setTitle:@"Stop" forState:UIControlStateSelected];
[self.view addSubview:buttonToLaunchStopAnimations];
```

Ensuite, définissons la méthode start/stop avec l'appel au timer :

```
- (void) launchStopAnim:(id)sender {
    UIButton *button = (UIButton*)sender;

    if (![button isSelected]) {
        canAnimate = YES;

        // on lance les anims
        // Couleur
        // On lance tout de suite une animation
        [self performSelector:@selector(changeColor:) withObject:nil];
        // on lance le timer
        timerForColorAnimation = [NSTimer scheduledTimerWithTimeInterval:4.0
            target:self selector:@selector(changeColor:) userInfo:nil repeats:YES];
    }
    else {
        canAnimate = NO;

        // on stoppe l'anim de la couleur
        [timerForColorAnimation invalidate];
        timerForColorAnimation = nil;
    }
    // on change l'état du bouton
    [button setSelected:[button isSelected]];
}
```

Toutes les 4 secondes, nous appellerons donc la méthode changeColor:. Explicitons-la :

```
- (void)changeColor:(NSTimer*)theTimer {
    [UIView beginAnimations: @"Color animation" context: NULL]; ①
    [UIView setAnimationDuration: 3.5]; ②

    CGFloat redLevel     = rand() / (float) RAND_MAX; ③
    CGFloat greenLevel   = rand() / (float) RAND_MAX;
    CGFloat blueLevel    = rand() / (float) RAND_MAX;

    viewForColorAnimation.backgroundColor = [UIColor colorWithRed: redLevel
    ↪ green: greenLevel blue: blueLevel alpha: 1.0]; ④
    [UIView commitAnimations]; ⑤
}
```

Ligne ①, on commence les animations, avec comme nom d'animation Color animation. Ce nom n'a pas d'importance, il permet de s'y retrouver dans votre code et/ou de gérer plusieurs animations en même temps (référez-vous à l'animation multiple plus loin).

Ligne ②, on choisit simplement la durée de l'animation en seconde.

Ligne ③ et suivantes, on définit une nouvelle couleur au hasard. Ligne ④, on change la couleur de notre vue avec la nouvelle. Enfin, ligne ⑤, on lance l'animation !

Info

Ce qu'il faut comprendre : avant d'entrer dans le bloc d'animation, la vue viewForColorAnimation a une certaine couleur. Ensuite, ligne ④, on définit une nouvelle couleur aléatoirement. Cette nouvelle couleur ne sera pas appliquée instantanément mais avec animation ! La transition se fera donc en douceur en 3,5 secondes.

Compilez et lancez pour observer le comportement de votre vue !

CHANGER LA TRANSPARENCE

Même combat pour cette animation. Une grosse différence cependant : nous n'utiliserons pas de timer. Oui, mais alors comment faire répéter cette animation ? Avec la propriété animationRepeatCount !

Ajoutez dans le .h :

```
- (void) startAlphaAnimation;
- (void) startWiggleAnimation;
- (void) startZoomAnimation;
- (void) startMultipleAnimations;
```

Puis dans le .m, définissez :

```
- (void) startAlphaAnimation {
#define alphaStart 0.9
#define alphaStop 0.1
    // point de départ
    viewForAlphaAnimation.alpha = alphaStart;
```

```
[UIView beginAnimations: @"Alpha animation" context: NULL];
[UIView setAnimationDuration: 2.0];
[UIView setAnimationRepeatAutoreverses:YES]; ①
[UIView setAnimationRepeatCount:1E100]; ②
viewForAlphaAnimation.alpha = alphaStop; // s'arrête ici et recommence
[UIView commitAnimations];
}
```

Ligne ②, on définit le comportement de notre animation : elle va se répéter quasi indéfiniment ($1E100 = 1*10^{100}$).

Ligne ①, on spécifie que notre animation va aller et venir : l'alpha de départ étant de 0.9, l'animation va l'amener à 0.1. Puis, l'animation va remettre la valeur de transparence d'origine de la vue en **l'animateur** puis va recommencer son cycle. Si on enlève la ligne ①, l'animation de notre vue va se comporter comme suit : l'alpha va passer de 0.9 à 0.1 puis brusquement de 0.1 à 0.9, sans animation. Essayez pour voir (impératif) !

L'animation ne se lance pas ? Évidemment, il faut rajouter la ligne en gras dans launchStopAnim :

```
// on lance le timer
timerForColorAnimation = [NSTimer scheduledTimerWithTimeInterval:4.0 target:self
➥ selector:@selector(changeColor:) userInfo:nil repeats:YES];

[self startAlphaAnimation];
```

Et initialiser la vue dans le viewDidLoad (initialisons-les toutes pour que ce soit fait !) :

```
// vue pour l'animation de la transparence
viewForAlphaAnimation = [[UIView alloc] initWithFrame:CGRectMake(100, 30, 60, 60)];
[viewForAlphaAnimation setBackgroundColor:[UIColor redColor]];
[self.view addSubview:viewForAlphaAnimation];

// bouton qui va trembler
buttonForWiggleAnimation = [[UIButton buttonWithType:UIButtonTypeRoundedRect]
➥ retain];
[buttonForWiggleAnimation setFrame:CGRectMake(30, 100, 70, 30)];
[buttonForWiggleAnimation setTitle:@"-°-" forState:UIControlStateNormal];
[self.view addSubview:buttonForWiggleAnimation];

// vue pour le zoom
imageViewForZoomAnimation = [[UIImageView alloc] initWithFrame:CGRectMake(120,
➥ 100, 100, 100)];
imageViewForZoomAnimation.image = [UIImage imageNamed:@"logo_ipup_losange.png"];
[self.view addSubview:imageViewForZoomAnimation];

//vue pour l'animation multiple
viewForMultiplesAnimations = [[UIView alloc] initWithFrame:CGRectMake(0, 160, 40, 40)];
viewForMultiplesAnimations.backgroundColor = [UIColor blueColor];
[self.view addSubview:viewForMultiplesAnimations];
```

Enfin, comment arrêter les animations ? En jouant sur le CALayer de chaque vue. Cette entité vous permet de réaliser les animations que ce soit en haut niveau, ou en plus bas niveau.

Importez le framework Quartz Core et ajoutez cette ligne dans UIViewAnimations.h :

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

Puis ajoutez les lignes en gras :

```
else {  
    canAnimate = NO;  
  
    // on stoppe l'anim de la couleur  
    [timerForColorAnimation invalidate];  
    timerForColorAnimation = nil;  
    // on arrête toutes les animations  
    for (UIView *view in [self.view subviews]) {  
        [view.layer removeAllAnimations];  
    }  
}
```

Info

Il est primordial, indispensable, d'**arrêter vos animations** dès lors que vous ne vous en servez plus, notamment lorsque vous quittez la vue !

FAIRE TREMBLER UN BOUTON

Ajoutez cette méthode pour faire trembler le bouton comme sur l'écran d'accueil de votre iPhone :

```
- (void) startWiggleAnimation {  
#define RADIANS(degrees) ((degrees * M_PI) / 180.0)  
  
    CGAffineTransform initialPosition =  
    ↪ CGAffineTransformRotate(CGAffineTransformIdentity, RADIANS(-7.0)); ①  
    CGAffineTransform finalPosition =  
    ↪ CGAffineTransformRotate(CGAffineTransformIdentity, RADIANS(7.0));  
    // point de départ :  
    buttonForWiggleAnimation.transform = initialPosition; ②  
  
    [UIView beginAnimations:@"Wiggle animation" context:NULL];  
    [UIView setAnimationRepeatAutoreverses:YES];  
    [UIView setAnimationRepeatCount:1E100];  
    [UIView setAnimationDuration:0.1];  
  
    buttonForWiggleAnimation.transform = finalPosition; // finit ici et recommence  
  
    [UIView commitAnimations];  
}
```

Cette fois-ci, nous agissons sur la propriété transform de notre vue (ligne ②). Cette propriété se révèle d'ailleurs très pratique pour transformer une vue (la déplacer, appliquer un zoom, la faire tourner...). Cette transformation se définit ligne ① et suivante, et attend en paramètre un angle en radian. Pour rappel, $180^\circ = \pi$ radian.

N'oubliez pas l'ajout de la ligne en gras pour lancer l'animation :

```
[self startAlphaAnimation];
[self startWiggleAnimation];
```

RÉALISER UN ZOOM

Cette fois-ci, plus rien ne devrait vous surprendre :

```
- (void) startZoomAnimation {
    CGAffineTransform initialZoom = CGAffineTransformMakeScale(1.0, 1.0);
    CGAffineTransform finalZoom = CGAffineTransformMakeScale(0.6, 0.6);
    // point de départ :
    imageViewForZoomAnimation.transform = initialZoom;

    [UIView beginAnimations:@"Zoom animation" context:NULL];
    [UIView setAnimationRepeatAutoreverses:YES];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseIn]; ①
    [UIView setAnimationRepeatCount:1E100];
    [UIView setAnimationDuration:0.6];

    imageViewForZoomAnimation.transform = finalZoom; // finit ici et recommence
    [UIView commitAnimations];
}
```

La seule différence se situe ligne ①. On y définit la courbe de vitesse de l'animation. Voici les quatre types possibles :

- UIViewAnimationCurveEaseInOut L'animation commence doucement, accélère jusque moitié, puis décélère jusque sa fin.
- UIViewAnimationCurveEaseIn L'animation commence doucement, puis accélère tout au long de la progression.
- UIViewAnimationCurveEaseOut L'animation commence rapidement puis ralentit vers la fin.
- UIViewAnimationCurveLinear L'animation est linéaire.

Vous n'avez pas oublié l'ajout de la ligne en gras ?

```
[self startWiggleAnimation];
[self startZoomAnimation];
```

RÉALISER PLUSIEURS ANIMATIONS ENCHAÎNÉES

Pour finir avec ces méthodes, nous allons réaliser un chemin d'animation : nous commencerons par déplacer une vue, la faire tourner de 180 degrés, la faire se redéplacer en changeant sa transparence puis la ramener à son point de départ.

Pour cela, nous allons utiliser le paramètre context de l'animation pour passer le numéro de l'animation.

Info

Nous pourrions tout à fait nous servir ici des noms des animations pour réaliser ce chemin.

Définissez cette méthode :

```
#pragma mark -  
#pragma mark multiple animations  
  
- (void) startMultipleAnimations {  
    CGPoint finalPosition = CGPointMake(300, 200);  
    CGPoint initialPosition = CGPointMake(0, 160);  
  
    viewForMultipleAnimations.center = initialPosition;  
  
    [UIView beginAnimations:@"First" context:[NSNumber numberWithInt:1]];  
    [UIView setAnimationDuration:3.0];  
    [UIView setAnimationDelegate:self]; ①  
    [UIView setAnimationDidStopSelector:@selector  
     ↪ (multipleAnimationEnded:finished:context:)]; ②  
  
    viewForMultipleAnimations.center = finalPosition;  
    [UIView commitAnimations];  
}
```

Et n'oubliez pas son appel :

```
[self startZoomAnimation];  
[self startMultipleAnimations];
```

Comme vous le voyez ligne ①, l'animation a un délégué. En fait, cette fois-ci, nous allons recourir à une méthode déléguée appelée lorsque l'animation sera terminée ligne ②. Dans cette méthode, nous ferons un `switch case` sur le numéro de l'animation et aiguillerons la prochaine animation. Pour les connaisseurs, c'est une machine à état !

```
// deuxième animation : retournement 180 degrès  
- (void) startSecondAnimation {  
    CGAffineTransform initialPosition = CGAffineTransformMakeRotation(RADIANS(0));  
    CGAffineTransform finalPosition = CGAffineTransformMakeRotation(RADIANS(180));  
  
    viewForMultipleAnimations.transform = initialPosition;  
  
    [UIView beginAnimations:@"Second" context:[NSNumber numberWithInt:2]];  
    [UIView setAnimationDuration:1.5];  
    [UIView setAnimationDelegate:self];
```

```
[UIView setAnimationDidStopSelector:@selector
➥ (multipleAnimationEnded:finished:context:)];
```

```
viewForMultiplesAnimations.transform = finalPosition;
[UIView commitAnimations];
}
```

```
// troisième animation : déplacement + changement de transparence
- (void) startThirdAnimation {
    CGPoint finalPosition = CGPointMake(30, 400);

    [UIView beginAnimations:@"Third" context:[NSNumber numberWithInt:3]];
    [UIView setAnimationDuration:3.0];
    [UIView setAnimationDelegate:self];
    [UIView setAnimationDidStopSelector:@selector
➥ (multipleAnimationEnded:finished:context:)];
```

```
viewForMultiplesAnimations.center = finalPosition;
[UIView commitAnimations];

[UIView beginAnimations:@"ThirdBis" context:NULL];
[UIView setAnimationDuration:3.0];
viewForMultiplesAnimations.alpha = 0.2;
[UIView commitAnimations];
}
```

```
// dernière animation : retour au début
- (void) returnToStartAnimation {
    CGPoint finalPosition = CGPointMake(0, 160);

    [UIView beginAnimations:@"Fourth" context:[NSNumber numberWithInt:4]];
    [UIView setAnimationDuration:3.0];
    [UIView setAnimationDelegate:self];
    [UIView setAnimationDidStopSelector:@selector(
➥ multipleAnimationEnded:finished:context:)];
```

```
viewForMultiplesAnimations.center = finalPosition;
[UIView commitAnimations];

[UIView beginAnimations:@"FourthBis" context:NULL];
[UIView setAnimationDuration:3.0];
viewForMultiplesAnimations.alpha = 1.0;
[UIView commitAnimations];
}
```

```

// méthode appelée à la fin de chaque animation
- (void) multipleAnimationEnded:(NSString *)animationID
    ↗ finished:(NSNumber *)finished context:(void *)context
{
    NSNumber *numberId = (NSNumber*)context;
    // récupération du numéro de l'animation qui a appelé cette méthode
    int idOfAnimation = [numberId intValue];

    if(canAnimate)
    {
        switch (idOfAnimation) {
            case 1:
                [self startSecondAnimation];
                break;
            case 2:
                [self startThirdAnimation];
                break;
            case 3:
                [self returnToStartAnimation];
                break;
            case 4:
            {
                [viewForMultiplesAnimations setTransform:CGAffineTransformIdentity];
                // on recommence
                [self startMultipleAnimations];
            }
            break;
        default:
            break;
    }
}
}

```

On remarque que l'on peut lancer plusieurs animations en même temps (déplacement et changement de transparence) en déclarant simplement autant de blocs d'animations que d'animations désirées !

RÉALISER DES ANIMATIONS EN CREUSANT

Pour cette partie, nous allons solliciter Core Animation dans une forme moins tape à l'œil. En effet, vous savez que plus on descend en terme de "niveau de programmation", plus le contrôle sur ce que l'on fait est précis. Ainsi, nous allons réaliser ici une animation qui va découper une image en plusieurs petits carrés et les faire se déplacer en changeant leur transparence... Ce code s'inspire du travail de Bill Dudney (<http://bill.dudney.net/roller/objc/>).

Notre objectif est de réaliser quelque chose de comparable à la Figure 26.1.



Pour ce faire :

1. Créez un nouveau fichier de type UIViewController et nommez-le Layers.
2. Dans le RootViewController, créez un nouveau bouton, comme pour afficher UIViewAnimations et ajoutez une action pour l'afficher.
3. Vérifiez en compilant qu'une vue vierge s'affiche bien. Si c'est le cas, passez à la suite, sinon revenez à la section "Création du projet" en l'adaptant.

Pour faire ces animations, nous allons nous appuyer sur les CALayer. Un CALayer, c'est un peu l'UIView de Core Animation. C'est le point de départ pour toute animation.

Dans Layers.h, ajoutez ces lignes :

```
#import <UIKit/UIKit.h>
@interface Layers : UIViewController {
    // layer pour l'animation
    CALayer *imageLayer;

    CGImageRef imageRef;
}
@property (nonatomic, retain) CALayer *imageLayer;
- (CGImageRef)scaleAndCropImage:(UIImage *)fullImage;
@end
```

N'oubliez pas dans le .m :

```
@synthesize imageLayer;
```

Ensuite, nous allons charger notre vue dans la méthode loadView pour changer (elle est appelée avant le viewDidLoad) :

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

    // Construction du layer qui va recevoir l'animation
```

```

    self.imageLayer = [CALayer layer];
    self.imageLayer.frame = CGRectMake(XPosition, YPosition, LargeurMax, HauteurMax);
    self.imageLayer.contentsGravity = kCAGravityResizeAspectFill;
    self.imageLayer.masksToBounds = YES;
    [self.view.layer addSublayer:self.imageLayer];
}

```

Puis, nous allons charger l'image et l'afficher dans notre CALayer. Toujours dans le loadView, ajoutez :

```

// image que l'on va transformer
UIImage *image = [UIImage imageNamed:@"logo_ipup_losange.png"];
// on redimensionne l'image et renvoie un CGImageRef
imageRef = [self scaleAndCropImage:image];
// on place l'image dans le layer
imageLayer.contents = (id)imageRef;

```

Ensuite, créons un bouton qui nous permettra de lancer l'animation. Ajoutez toujours au même endroit :

```

// bouton pour lancer l'animation
UIButton *buttonToPop = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[buttonToPop setFrame:CGRectMake(130, 400, 70, 30)];
[buttonToPop setTitle:@"Pop" forState:UIControlStateNormal];
[buttonToPop addTarget:self action:@selector(pop:)];
    ↗ forControlEvents:UIControlEventTouchUpInside];
[self.view addSubview:buttonToPop];

```

Ah tiens, il y a des valeurs qu'il ne connaît pas ! Qu'à cela ne tienne... Ajoutez ces lignes en dessous de l'import :

```

#define LargeurMax 300.0f
#define HauteurMax 300.0f
#define XPosition 10.0f
#define YPosition 20.0f
#define nombreDeCarreX 10.0f
#define nombreDeCarreY 10.0f

```

Vous l'avez remarqué également, il faut définir la méthode dont la signature est scaleAndCropImage:. C'est un peu compliqué car elle fait appel à Core Graphics dont le langage est un peu particulier (se rapprochant de Carbon).

```

// méthode pour redimensionner l'image et la convertir en CGImageRef
- (CGImageRef)scaleAndCropImage:(UIImage *)fullImage {
    CGSize imageSize = fullImage.size;
    CGFloat scale = 1.0f;
    CGImageRef subimage = NULL;

    if(imageSize.width > imageSize.height) {
        // La hauteur est plus petite que la largeur

```

```
scale = HauteurMax / imageSize.height;
CGFloat offsetX = ((scale * imageSize.width - LargeurMax) / 2.0f) / scale;
CGRect subRect = CGRectMake(offsetX, 0.0f, imageSize.width - (2.0f *
➥ offsetX), imageSize.height);
subimage = CGImageCreateWithImageInRect([fullImage CGImage], subRect);
} else {
    // La largeur est plus petite que la hauteur
    scale = LargeurMax / imageSize.width;
    CGFloat offsetY = ((scale * imageSize.height - HauteurMax) / 2.0f) / scale;
    CGRect subRect = CGRectMake(0.0f, offsetY, imageSize.width, imageSize.height
➥ - (2.0f * offsetY));
    subimage = CGImageCreateWithImageInRect([fullImage CGImage], subRect);
}

// on zoom l'image selon LargeurMax et HauteurMax définis au dessus
CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
CGContextRef context = CGBitmapContextCreate(NULL, LargeurMax, HauteurMax,
➥ 8, 0, colorSpace, kCGImageAlphaPremultipliedFirst);
CGContextSetInterpolationQuality(context, kCGInterpolationHigh);
CGRect rect = CGRectMake(0.0f, 0.0f, LargeurMax, HauteurMax);
CGContextDrawImage(context, rect, subimage);
CGContextFlush(context);
// on obtient la nouvelle image
CGImageRef scaledImage = CGBitmapContextCreateImage(context);
CGContextRelease (context);
CGImageRelease(subimage);
subimage = NULL;
subimage = scaledImage;
return subimage;
}
```

Maintenant, définissons l'action à réaliser lorsque nous appuyons sur le bouton. Premièrement, nous allons créer un tableau qui contiendra tous les petits carrés de notre image que nous allons découper : (jusqu'à ce que je vous dise le contraire, tous les morceaux de code iront dans cette méthode, dans le if) :

```
// Là où tout commence...
- (void)pop:(id)sender {
    if(nil != imageLayer.contents) {

        //on récupère la taille de l'image à découper
        CGSize imageSize = CGSizeMake(CGImageGetWidth(imageRef),
➥ CGImageGetHeight(imageRef));
        NSMutableArray *layers = [NSMutableArray array];
    }
}
```

Ensuite, nous allons découper notre image en carrés avec deux boucles imbriquées. Ajoutez :

```
// on la découpe !
for(int x = 0;x < nombreDeCarreX;x++) {
    for(int y = 0;y < nombreDeCarreY;y++) {
        // création d'une nouvelle Frame (la taille du petit carré)
        CGRect frame = CGRectMake((imageSize.width / nombreDeCarreX) * x,
        ↪ (imageSize.height / nombreDeCarreY) * y, imageSize.width /
        ↪ nombreDeCarreX, imageSize.height / nombreDeCarreY);

        //on crée un layer pour ce petit carré
        CALayer *layer = [CALayer layer];
        layer.frame = frame;
        // définition de l'action (l'animation)
        layer.actions = [NSDictionary dictionaryWithObject:[self
        ↪ animationForX:x Y:y imageSize:imageSize] forKey:@"opacity"];
        // on crée une mini image pour ce petit carré
        CGImageRef subimage = CGImageCreateWithImageInRect(imageRef, frame);
        layer.contents = (id)subimage;
        CFRelease(subimage);
        [layers addObject:layer];
    }
}
```

Enfin, on ajoute tous ces petits carrés en tant que sous-layers de notre `imageLayer` :

```
// on ajoute les petits layers créés à notre imageLayer
for(CALayer *layer in layers) {
    [imageLayer addSublayer:layer];
    layer.opacity = 0.0f;
}
```

Cela a pour effet de reconstituer notre image. Rappelez-vous, chaque sous-layer a comme contenu la mini-image (`subimage`) découpée spécialement. Il faut donc enlever l'image du layer principal sinon l'image sera en double et l'on ne verra pas l'animation !

```
// maintenant que l'on a reconstitué l'image avec nos petits carrés, on supprime
// le contenu (l'image originale) de imageLayer
imageLayer.contents = nil;
```

Si vous avez été attentif, une nouvelle méthode a fait son apparition : `animationForX:Y:imageSize:`. Elle va nous servir pour créer l'animation. En fait, chaque petit carré découpé aura sa propre animation étant donné que chaque carré est un `CALayer`. C'est là toute la puissance !

L'animation du carré en compte en fait deux : une qui va changer la transparence de chaque petit carré, et une autre qui va le déplacer. On crée donc un groupe d'animations (CAAnimationGroup) :

```
// Crée une animation pour chaque petit carré
- (CAAnimation *)animationForX:(NSInteger)x Y:(NSInteger)y imageSize:(CGSize)size {
    // On retourne un group d'animation qui comporte l'animation de l'opacité
    // (transparence) et l'animation de la position (déplacement)
    CAAnimationGroup *group = [CAAnimationGroup animation];
    group.delegate = self;
    group.duration = 2.0f;

    // animation pour la transparence
    CABasicAnimation *opacity = [CABasicAnimation animationWithKeyPath:@"opacity"];
    opacity.fromValue = [NSNumber numberWithDouble:1.0f]; // transparence de départ
    opacity.toValue = [NSNumber numberWithDouble:0.0f]; // transparence de fin

    //animation du déplacement
    CABasicAnimation *position =
    // [CABasicAnimation animationWithKeyPath:@"position"];
    position.timingFunction = [CAMediaTimingFunction
    // functionWithName:kCAMediaTimingFunctionEaseIn];
    CGPoint dest = [self randomDestinationX:x Y:y imageSize:size];
    // on choisit aléatoirement une position
    position.toValue = [NSValue valueWithCGPoint:dest];

    group.animations = [NSArray arrayWithObjects:opacity, position, nil];
    // On crée un group d'animation
    return group;
}
```

Un petit schéma est nécessaire pour résumer la situation (voir Figure 26.2) :

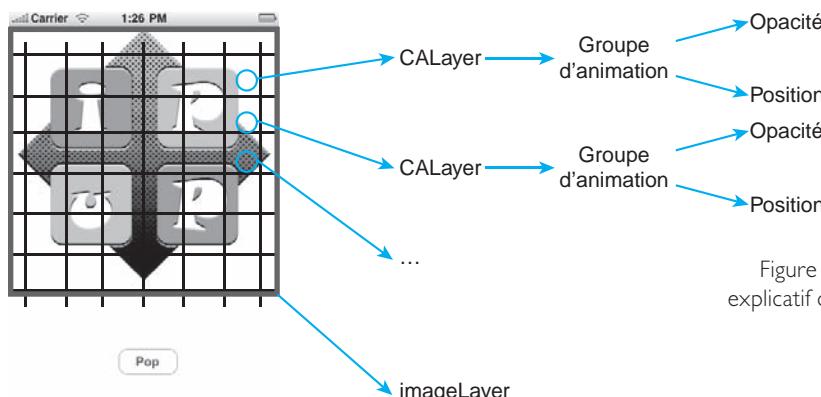


Figure 26.2 : Schéma explicatif du découpage.

Encore une fois, une nouvelle méthode est apparue : randomDestinationX:Y:imageSize:... Elle va renvoyer un point (CGPoint) qui sera la destination aléatoire de notre carré :

```
// Choisir une destination aléatoire pour le carré

- (CGPoint)randomDestinationX:(CGFloat)x Y:(CGFloat)y imageSize:(CGSize)size {
    CGPoint destination;

    // on tire au hasard une valeur (soit 0 soit 1)
    int sensX = random()%2;
    // Si c'est 0, alors le sens sera -1 sinon ce sera 1
    sensX = sensX == 0 ? -1 : 1;

    int sensY = random()%2;
    sensY = sensY == 0 ? -1 : 1;

    // on tire au sort une destination
    destination.x = (CGFloat)sensX * 50.0f * ((CGFloat)(random() % 10000)) / 2000.0f;
    destination.y = (CGFloat)sensY * 50.0f * ((CGFloat)(random() % 10000)) / 2000.0f;

    return destination;
}
```

Voilà, c'est maintenant fini ! Vous n'avez plus qu'à compiler et lancer votre application pour apprécier le résultat. Notez que si vous avez des [warnings de compilation](#), vous devez vous souvenir des règles et ordres de lecture des fichiers lors de la compilation (nous ne déclarons pas toutes les méthodes dans le header).

Voici donc l'architecture globale de votre .m à obtenir :

```
#import "Layers.h"
#import <QuartzCore/QuartzCore.h>
//Définition des variables

#define LargeurMax 300.0f
#define HauteurMax 300.0f
#define XPosition 10.0f
#define YPosition 20.0f
#define nombreDeCarreX 10.0f
#define nombreDeCarreY 10.0f

@implementation Layers
@synthesize imageLayer;
- (void)loadView {
    ...
}
// Appelé lorsque l'animation se termine
```

```
- (void)animationDidStop:(CAAnimation *)theAnimation finished:(BOOL)flag {
    ...
}

// Choisir une destination aléatoire pour le carré

- (CGPoint)randomDestinationX:(CGFloat)x Y:(CGFloat)y imageSize:(CGSize)size {
    ...
}

// Crée une animation pour chaque petit carré
- (CAAnimation *)animationForX:(NSInteger)x Y:(NSInteger)y
    ...
}

// Là où tout commence...
- (void)pop:(id)sender {
    ...
}

// méthode pour redimensionner l'image et la convertir en CGImageRef
(CGImageRef)scaleAndCropImage:(UIImage *)fullImage {
    ...
}

- (void)didReceiveMemoryWarning {
    ...
}

- (void)dealloc {
    [imageLayer release];
    CGImageRelease(imageRef);
    [super dealloc];
}
@end
```

Les applications iPhone font très souvent appel à ce que l'on nomme des webservices, c'est-à-dire des applications situées sur des serveurs distants (voir Fiche 34 pour l'utilisation du JSON). Nous allons aborder cette problématique par trois notions importantes :

- les méthodes de classe ;
- la notion de connexion Synchrone et Asynchrone ;
- les requêtes HTTP utilisant les méthodes POST ou GET.

Commencez par créer un nouveau projet avec la vue associée : File > New Project > View-based Application. Vous pouvez nommer votre projet WebService pour respecter le même nommage que celui de l'ouvrage.

Nous allons, pour ce projet, nous contenter d'une application compatible iPhone.

MÉTHODE DE CLASSE

Les méthodes de classe créent des petites fonctionnalités facilement réutilisables et constituent donc des outils précieux au quotidien pour un développeur. Analysons tout cela avec un exemple concret.

Commencez par créer un dossier (Group) iPnP (Ctrl+Clic sur votre projet > Add > New Group). Ensuite, ajoutez une classe CiPnP (.h et .m) dans ce dossier (Ctrl+Clic > Add > New File > Objective-C class subclass of NSObject).

Puis ordonnez vos fichiers pour aboutir à la Figure 27.1.

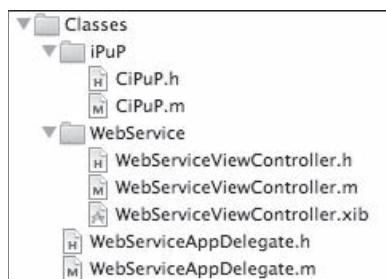


Figure 27.1 : Organisation initiale du Projet.

Implémentons maintenant une méthode de classe dans CiPnP, fichier qui d'ailleurs ne comprendra que des méthodes de classe. Cette méthode permettra de faire apparaître une UIAlertView présentant un message.

```
CiPnP.h
#import <Foundation/Foundation.h>
@interface CiPnP : NSObject

+(void) afficherAlerte:(NSString *)string;
@end
```

```
CiPuP.h
#import "CiPuP.h"
@implementation CiPuP

//La méthode de classe se distingue pas le + qui la précède (en lieu et place du -)
+ (void) afficherAlerte:(NSString *)string{
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Alerte"
→ message:string delegate:self cancelButtonTitle:@"ok" otherButtonTitles:nil];
    [alert show];
    [alert release];
}
@end
```

Voyons maintenant comment utiliser cette méthode :

Dans WebServiceViewController.m, commençons par ajouter l'import :

```
#import "CiPuP.h"
@implementation WebServiceViewController
```

Puis dans le viewDidLoad,

```
- (void)viewDidLoad {
    [super viewDidLoad];
    [CiPuP afficherAlerte:@"Ma super méthode"];
}
```

À ce stade, en compilant votre projet et lançant l'application, vous devriez voir apparaître une alerte avec votre message.

VÉRIFIER LA DISPONIBILITÉ DU RÉSEAU

Vous allez maintenant comprendre pourquoi nous abordons à ce stade du livre les méthodes de classe.

En effet, s'il y a une chose à retenir pour qu'une application ne soit pas refusée par Apple au moment de la validation, c'est de toujours vérifier que la connexion Internet est disponible avant d'effectuer une requête synchrone sur un serveur distant.

Implémentons ainsi une méthode de classe vérifiant cette disponibilité et dont l'appel sera simplifié.

Modifions le CiPuP.h ainsi :

```
#import <Foundation/Foundation.h>
#import <netinet/in.h>
#import <SystemConfiguration/SystemConfiguration.h>

@interface CiPuP : NSObject

+(BOOL) reseauDisponible;
+(void) afficherAlerte:(NSString *)string;

@end
```

Info

Il ne faut pas oublier d'ajouter le framework *SystemConfiguration* à votre projet.

```
+ (BOOL)reseauDisponible {
    struct sockaddr_in zeroAddress;
    bzero(&zeroAddress, sizeof(zeroAddress));
    zeroAddress.sin_len = sizeof(zeroAddress);
    zeroAddress.sin_family = AF_INET;

    SCNetworkReachabilityRef defaultRouteReachability =
    ↪ SCNetworkReachabilityCreateWithAddress(NULL, (struct sockaddr *)&zeroAddress);
    SCNetworkReachabilityFlags flags;

    BOOL didRetrieveFlags =
    ↪ SCNetworkReachabilityGetFlags(defaultRouteReachability, &flags);
    CFRelease(defaultRouteReachability);

    if (!didRetrieveFlags)
        return NO;

    BOOL isReachable = flags & kSCNetworkFlagsReachable;
    BOOL needsConnection = flags & kSCNetworkFlagsConnectionRequired;
    return (isReachable && !needsConnection) ? YES : NO;
}
```

Info

Le code de la méthode *reseauDisponible* est extrait de l'exemple *Reachability* proposé par Apple. Vous êtes invité à vous y référer pour approfondir les notions abordées dans cette fiche.

Nous aurons donc désormais la possibilité de tester très simplement si l'iPhone (ou l'iPod) dispose d'une connexion Internet :

```
if (![CiPuP reseauDisponible]){
    [CiPuP showAlert:@"pas de réseau disponible"];
}
else {
    //utilisation du webservice
}
```

RÉALISATION DE L'INTERFACE GRAPHIQUE

L'interface graphique se réalise grâce à Interface Builder. Dans la classe WebServiceViewController, ajoutez deux UISegmentedControl, un UILabel et un UIButton.

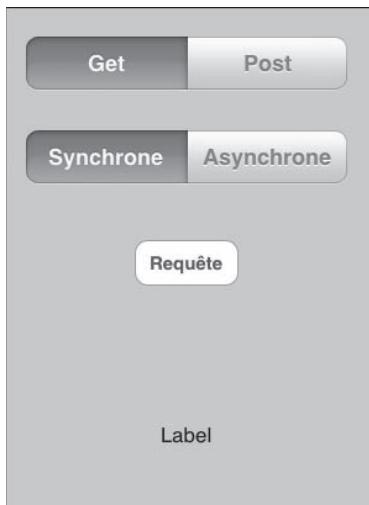


Figure 27.2 : Interface graphique à réaliser.

Passons à l'implémentation du contrôleur de vue.

```
#import <UIKit/UIKit.h>
@interface WebServiceViewController : UIViewController {
    IBOutlet UISegmentedControl *getPost;
    IBOutlet UISegmentedControl *synchAsynch;
    IBOutlet UILabel *label;
}
-(IBAction)boutonRequete;
@end
```

Dans le .m :

```
-(IBAction)boutonRequete{
    //Cas 1 : Segment 1 sur GET et Segment 2 sur Synchrone
    if(getPost.selectedSegmentIndex == 0 && synchAsynch.selectedSegmentIndex == 0){
        label.text = @"cas 1" ;
    }
    //Cas 2 : Segment 1 sur POST et Segment 2 sur Synchrone
    else if(getPost.selectedSegmentIndex == 1 && synchAsynch.selectedSegmentIndex == 0){
        label.text = @"cas 2" ;
    }
    //Cas 3 : Segment 1 sur GET et Segment 2 sur Asynchrone
```

```

        else if(getPost.selectedIndex == 0 && synchAsynch.selectedIndex == 1){
            label.text = @"cas 3" ;
        }
        //Cas 4 : Segment 1 sur POST et Segment 2 sur Asynchrone
        else {
            label.text = @"cas 3"
        }
    }
}

```

Les deux UISegmentedControl permettent de choisir une des quatre méthodes d'interrogation du webservice. Le bouton, quant à lui, vous permettra de lancer la requête.

Faites bien attention de relier correctement chacun des UISegmentedControl, le UILabel sans oublier de connecter l'IBAction au UIButton.

Vérifiez ensuite que cela fonctionne correctement avant de continuer.

LE WEBSERVICE À VOTRE DISPOSITION

Pour ce tutoriel, je vous propose un webservice réalisé par mes soins qui est volontairement très simple. En fait il va simplement vous renvoyer le contenu de la variable parametre que vous allez lui envoyer. Il fonctionne aussi bien en GET qu'en POST, et il est disponible à cette adresse :<http://www.ipup.fr/livre/getPost>

Réalisé en PHP, ce service aurait tout aussi bien pu l'être en Java, en Ruby ou avec tout autre langage compatible avec le protocole http.

Ce service est vraiment simpliste :

```

<?php
echo $_POST['parametre'];
echo $_GET['parametre'];
?>

```

L'idée est juste de récupérer la variable parametre transmise en GET ou en POST et de la renvoyer.

Tout l'intérêt est maintenant de réussir à transmettre à notre serveur cette variable avec des informations à l'intérieur puis d'en récupérer la réponse.

SYNCHRONE/ASYNCROHNE

Il n'est pas toujours simple pour les débutants de faire la distinction entre une méthode Synchrone et une méthode Asynchrone.

Commençons par regarder comment se présente en code chacune des deux méthodes :

```

//Synchrone :
-(void)maMethode{
    [self actionUn] ;
    [self actionDeux] ;
    [self actionTrois] ;
}

```

Ici on appelle actionUn qui bloque le thread principal, jusqu'à la fin de son exécution, puis actionDeux qui, à son tour, bloque le thread principal et ainsi de suite.

```
//ASynchrone :  
-(void)maMethode{  
    [self actionUnWithDelegate:self] ;  
}  
-(void)actionUnTerminee{  
    [self actionDeuxWithDelegate:self] ;  
}  
-(void)actionDeuxTerminee{  
    [self actionTroisWithDelegate:self] ;  
}  
-(void)actionTroisTerminee{  
    //c'est fini  
}
```

La grande différence en asynchrone est que l'on ne bloque pas le thread principal, une méthode déléguée est appelée quand la méthode d'action est terminée.

La technique de programmation est différente et peut paraître un peu plus lourde, mais elle est finalement très pratique. En effet, dans notre cas, nos méthodes réalisent une requête vers un serveur dont on ne peut pas maîtriser la vitesse d'exécution qui dépend de la qualité du signal ainsi que de la performance du serveur.

À l'usage, pour réaliser des connexions en synchrone, il serait nécessaire de détacher un thread pour réaliser la requête afin d'éviter le blocage de l'application pendant l'exécution de la méthode. Le problème est qu'à ce stade de votre apprentissage, la gestion des threads est encore assez compliquée à mettre en place.

Les méthodes asynchrones évitent deux choses : de paralyser le thread principal et d'avoir à réaliser une gestion difficile (manuelle) des threads.

Astuce

Je vous conseille le plus possible de ne pas gérer les threads vous-même car cela conduit très rapidement à des applications instables, ou à une réalisation très lourde.

GET/POST

Sans entrer trop dans les détails (si cela vous intéresse il faut vous référer aux normes du protocole http), on peut dire que la distinction principale entre POST et GET est le moyen de passer l'information au serveur.

Pour passer une variable en GET c'est très simple puisqu'on modifie simplement l'URL. Dans notre cas si on veut passer la variable maVariable à <http://monsite.com> il suffit de faire une requête à l'URL : <http://monsite.com?maVariable=maValeur>

Pour passer plusieurs variables :

<http://monsite.com?maVariable1=maValeur1&maVariable2=maValeur2>

Le principal problème est que l'on ne peut pas passer tous les caractères dans une URL (il faut respecter la norme RFC 2396). Il est possible d'utiliser la méthode stringByAddingPercentEscapesUsingEncoding: de la classe NSString pour remplacer les espaces par %20 par exemple, mais cela devient très vite lourd.

Pour passer une variable en POST, il est nécessaire d'écrire dans le corps de la requête. Opération un peu plus compliquée, mais qui présente l'avantage de nous laisser beaucoup plus libre quant aux informations que l'on souhaite passer.

Ces deux mises au point faites, nous allons mettre tout cela en œuvre en modifiant le code de notre contrôleur.

```
WebServiceViewController.h
#import <UIKit/UIKit.h>
@interface WebServiceViewController : UIViewController {
    IBOutlet UISegmentedControl *getPost;
    IBOutlet UISegmentedControl *synchAsynch;
    IBOutlet UILabel *label;

    NSURLConnection      *getConnection;
   NSMutableData      *getData;

    NSURLConnection      *postConnection;
    NSMutableData      *postData;
}

-(IBAction)boutonRequete;
-(void)actualiserLabel:(NSData*)data;

-(void)getSynch;
-(void)postSynch;
-(void)getAsynch;
-(void)postAsynch;

@end

WebServiceViewController.m
-(IBAction)boutonRequete{
    //Cas 1 : Segment 1 sur GET ET Segment 2 sur Synchrone
    if(getPost.selectedSegmentIndex == 0 && synchAsynch.selectedSegmentIndex == 0){
        [self getSynch];
    }
    //Cas 2 : Segment 1 sur POST ET Segment 2 sur Synchrone
    else if(getPost.selectedSegmentIndex == 1 && synchAsynch.selectedSegmentIndex == 0){
        [self postSynch];
    }
    //Cas 3 : Segment 1 sur GET ET Segment 2 sur Asynchrone
    else if(getPost.selectedSegmentIndex == 0 && synchAsynch.selectedSegmentIndex == 1){
        [self getAsynch];
    }
}
```

```

    }
    //Cas 4 : Segment 1 sur POST ET Segment 2 sur Asynchrone
    else {
        [self postAsynch];
    }
}

//méthode pour afficher une NSData dans notre label
-(void)actualiserLabel:(NSData*)data{
    NSString *message = [[NSString alloc] initWithData:data
                                                encoding:NSUTF8StringEncoding];
    label.text = message;
    [message release];
}

```

Passons maintenant aux quatre méthodes permettant de faire une requête vers un serveur web que nous allons détailler.

GET SYNCHRONE

Méthode pour envoyer une variable GET en synchrone :

```

-(void)getSynch{
    //Vérification de la disponibilité du réseau
    if(![CiPuP reseauDisponible]){
        [CiPuP showAlert:@"pas de réseau disponible"];
    }
    else {
        //On passe le paramètre dans l'url => méthode GET
        NSURL *url = [[NSURL alloc] initWithString:@"http://www.ipup.fr/livre/
                                                ↪ getPost?parametre=get_synch"];
        NSURLRequest *request = [[NSURLRequest alloc] initWithURL:url];
        [url release];
        NSURLResponse *reponse = nil;
        NSError *error = nil;
        NSData *data = [NSURLConnection sendSynchronousRequest:request
                                                ↪ returningResponse:&reponse error:&error];
        //l'application va bloquer ici le temps de faire la requête.
        [request release];
        [self actualiserLabel:data];
        if(error){
            //On affiche la description de l'erreur
            [CiPuP showAlert:[error localizedDescription]];
        }
    }
}

```

POST SYNCHRONE

```
-(void)postSynch{
    //Vérification de la disponibilité du réseau
    if (![CiPuP reseauDisponible]){
        [CiPuP showAlert:@"pas de réseau disponible"];
    }
    else {
        NSURL *url = [NSURL URLWithString:@"http://www.ipup.fr/livre/getPost"];
        NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];
        //On passe le paramètre dans le corps de la requête => méthode POST
        [request setHTTPMethod:@"POST"];
        [request setHTTPBody:[@{"parametre":post_synch
        ↪ dataUsingEncoding:NSUTF8StringEncoding}]];
        NSURLResponse *reponse = nil;
        NSError *error = nil;
        NSData *data = [NSURLConnection sendSynchronousRequest:request
        ↪ returningResponse:&reponse error:&error];
        //l'application va bloquer ici le temps de faire la requête.
        [self actualiserLabel:data];
        if(error){
            //On affiche la description de l'erreur
            [CiPuP showAlert:[error localizedDescription]];
        }
    }
}
```

GET ASYNCHRONE

```
-(void)getAsynch{
    //On passe le paramètre dans l'url => méthode GET
    NSURL *url = [NSURL URLWithString:@"http://www.ipup.fr/livre/
    ↪ getPost?parametre=get_a_synch"];
    NSURLRequest* request = [NSURLRequest requestWithURL:url
    ↪ cachePolicy:NSURLRequestUseProtocolCachePolicy timeoutInterval:40.0];
    getConnection = [[NSURLConnection alloc] initWithRequest:request delegate:self];
    getData = [[NSMutableData data] retain];
    [getConnection release];
}
```

POST ASYNCHRONE

```
-(void)postAsynch{
    NSURL *url = [NSURL URLWithString:@"http://www.ipup.fr/livre/getPost"];
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];
    //On passe le paramètre dans le corps de la requête => méthode POST
    [request setHTTPMethod:@"POST"];
    [request setHTTPBody:[@"parametre=post_a_synch"
        → dataUsingEncoding:NSUTF8StringEncoding]];
    postConnection = [[NSURLConnection alloc] initWithRequest:request delegate:self];
    postData = [[NSMutableData data] retain];
    [postConnection release];
}
```

DÉLÉGUÉE DE MÉTHODE ASYNCHRONE

```
#pragma mark -
#pragma mark NSURLConnection delegate

//méthode appelée de manière asynchrone lorsque la connexion reçoit des données.
- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data {
    if(connection == getConnection)
    {
        [getData appendData:data];
    }
    else if(connection == postConnection)
    {
        [postData appendData:data];
    }
}

//Méthode appelée en cas d'erreur
- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error{
    if(connection == getConnection)
    {
        [getData release];
    }
    else if(connection == postConnection)
    {
        [postData release];
    }
    //On affiche la description de l'erreur
    [CiPuP showAlert:[error localizedDescription]];
}
```

```
//méthode appelée lorsque le téléchargement est terminé
- (void) connectionDidFinishLoading:(NSURLConnection*)connection {
    if(connection == getConnection){
        [self actualiserLabel:getData];
        [getData release];
    }
    else if(connection == postConnection){
        [self actualiserLabel:postData];
        [postData release];
    }
}
```

CONCLUSION

On peut maintenant dire que vous maîtrisez les fondamentaux des requêtes sur serveur, et c'est à vous de vous adapter à chaque situation.

Vous l'aurez probablement compris, la méthode à privilégier, car à mon sens elle est la plus performante, est la méthode POST asynchrone.

Par ailleurs, dans certains cas, il peut être très intéressant de sous-classer NSURLConnection afin de l'adapter à votre situation, en y ajoutant par exemple un tag ou une gestion du cache.

Enfin, et parce qu'il faut aussi profiter des bonnes ressources du Net, je vous recommande le framework ASIHTTPRequest disponible ici : <http://allseeing-i.com/ASIHTTPRequest/> qui est une surcouche du SDK très pratique à utiliser.

Votre iPhone possède un carnet d'adresses. Vous pouvez y accéder depuis votre application, et nous allons apprendre à l'utiliser pour envoyer des SMS et des mails.

Nous allons réaliser une application qui va ressembler à la Figure 28.1. Créez un nouveau projet de type Window-based Application, que vous nommerez Composer. Ajoutez comme à notre habitude un nouveau fichier de type UIViewController que vous nommerez RootViewController (cochez With XIB for user interface). Je vous conseille vivement de créer vous-même l'interface graphique d'après la Figure 28.1. Si vous n'êtes pas encore prêt, pas de souci, tout sera expliqué par la suite ! Mais n'oubliez pas qu'il faudra vous jeter à l'eau un jour.

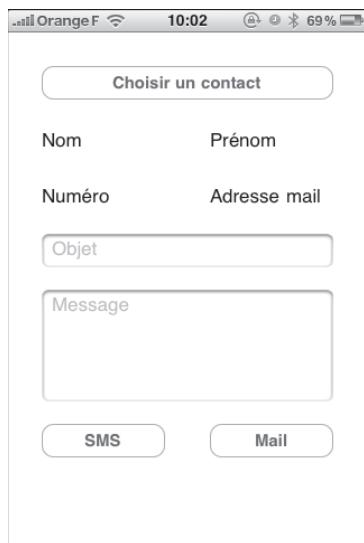


Figure 28.1 : Un aperçu de la fin.

AFFICHER LA LISTE DU CARNET D'ADRESSES

Vous pouvez accéder au carnet d'adresses de deux manières :

- en utilisant un contrôleur créé par Apple, ABPeoplePickerNavigationController qui vous présentera directement la liste de vos contacts ;
- en écrivant des lignes de codes pour accéder à la base de données du téléphone.

Faites très attention à cette dernière solution, car il peut sembler facile de récolter des données personnelles sans que l'utilisateur ne le sache. N'oubliez pas qu'une telle pratique est strictement illégale. Si vous souhaitez stocker des données personnelles sur un serveur par exemple, il faudra le faire de manière transparente, et le déclarer à la Cnil (Commission nationale de l'informatique et des libertés) en demandant une autorisation. Par contre, vous verrez qu'accéder à la base de données du carnet d'adresses permet de proposer de l'autocomplétion très facilement.

Commençons par créer notre bouton pour afficher la vue du carnet d'adresses. Ajoutez ces lignes dans le initWithNibName:bundle: du RootViewController :

```
// Ajout d'un bouton pour afficher le répertoire
UIButton *buttonAddressBook = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[buttonAddressBook setTitle:@"Choisir un contact" forState:UIControlStateNormal];
[buttonAddressBook addTarget:self action:@selector(chooseContact:)
    forControlEvents:UIControlEventTouchUpInside];
[buttonAddressBook setFrame:CGRectMake(30, 30, 260, 30)];
[self.view addSubview:buttonAddressBook];
```

Avant d'implémenter la méthode chooseContact:, il faut ajouter les frameworks pour bénéficier du carnet d'adresses. Ajoutez donc AddressBook et AddressBookUI à votre projet. Importez-les ensuite dans RootViewController.h :

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

Revenez au .m, et écrivez la méthode chooseContact::

```
- (void) chooseContact:(id)sender {
    ABPeoplePickerNavigationController *picker = [[ABPeoplePickerNavigationController
        alloc] init];
    picker.peoplePickerDelegate = self;
    [self presentModalViewController:picker animated:YES];
    [picker release];
}
```

Cette dernière méthode ne présente rien d'exceptionnel, elle va juste afficher le contrôleur. Par contre, on voit que self est delegate du picker, il faut donc écrire dans le .h :

```
@interface RootViewController : UIViewController
    <ABPeoplePickerNavigationControllerDelegate>
```

et implémenter les méthodes suivantes dans le .m :

```
#pragma mark -
#pragma mark ABPeoplePickerNavigationControllerDelegate
// appelée lorsque l'utilisateur annule
- (void)peoplePickerNavigationControllerDidCancel:(ABPeoplePickerNavigationController
    *)peoplePicker ①
{
    // on enlève le picker affiché
    [self dismissModalViewControllerAnimated:YES];
}

// appelée lorsque le picker choisit un contact et demande si il peut continuer
```

```
(BOOL)peoplePickerNavigationController:(ABPeoplePickerController
➥*)peoplePicker shouldContinueAfterSelectingPerson:(ABRecordRef)person ②
{
    return YES;
}

// appelé lorsque l'utilisateur choisit une propriété d'un contact
- (BOOL)peoplePickerNavigationController:(ABPeoplePickerController
➥*)peoplePicker shouldContinueAfterSelectingPerson:(ABRecordRef)person
➥ property:(ABPropertyID)property identifier:(ABMultiValueIdentifier)identifier ③
{
    return YES;
}
```

Attardons-nous quelques instants sur ces méthodes qui sont requises toutes les trois.

La méthode ligne ① est appelée lorsque l'utilisateur appuie sur le bouton Annuler. Le comportement basique est d'enlever le contrôleur de l'écran.

La méthode à la ligne ②, est appelée quand l'utilisateur choisit un contact. Cette dernière demande si le picker peut continuer, c'est-à-dire si elle peut afficher dans une nouvelle vue les informations du contact (Nom, prénom, adresse, numéros de téléphones...).

Enfin, la méthode ligne ③ demande si le picker peut continuer après que l'utilisateur a appuyé sur une des propriétés du contact. En retournant YES comme au-dessus, le comportement sera le suivant : l'utilisateur sélectionne un contact, puis peut choisir une propriété. S'il appuie sur un numéro de téléphone, l'iPhone appellera ce numéro, s'il appuie sur l'adresse mail, cela lancera l'application mail.

Maintenant, si vous souhaitez utiliser le carnet d'adresses dans votre application pour demander par exemple à l'utilisateur de sélectionner un numéro de téléphone, il faudra laisser ① et ② comme ci-dessus et écrire dans la méthode ③ :

```
// appelé lorsque l'utilisateur choisit une propriété d'un contact
- (BOOL)peoplePickerNavigationController:(ABPeoplePickerController
➥*)peoplePicker shouldContinueAfterSelectingPerson:(ABRecordRef)person
➥ property:(ABPropertyID)property identifier:(ABMultiValueIdentifier)identifier
{
    if(property == kABPersonPhoneProperty)
    {
        // on récupère le ou les numéro(s) de téléphone
        ABMultiValueRef phoneNumbers =
        ➤ ABRecordCopyValue(person, kABPersonPhoneProperty);
        // on les place dans un tableau
        NSArray *theArray =
        ➤ [(id)ABMultiValueCopyArrayOfAllValues(phoneNumbers) autorelease];
        // on affiche le numéro de téléphone choisi en fonction de l'identifiant
```

```

    NSLog(@"%@", [theArray
        objectAtIndex:ABMultiValueGetIndexForIdentifier(phoneNumbers, identifier)]);
    [self dismissModalViewControllerAnimated:YES];
}
return NO;
}

```

Dans notre cas, nous allons demander à l'utilisateur de choisir un contact pour récupérer :

- son nom ;
- son prénom ;
- un numéro de téléphone ;
- une adresse e-mail.

RÉCUPÉRER LES PROPRIÉTÉS D'UN CONTACT

Créons tout d'abord nos quatre labels pour afficher ces informations. Pour cela, écrivez dans .h :

```
UILabel *labelName, *labelFirstName, *labelPhoneNumber, *labelMailAddress;
```

Puis, dans le `initWithNibName:bundle:`, écrivez :

```

// ajout du label pour le nom
labelName = [[UILabel alloc] initWithFrame:CGRectMake(30, 80, 110, 30)];
[labelName setText:@"Nom"];
labelName.adjustsFontSizeToFitWidth = YES;
[self.view addSubview:labelName];

// label pour le prénom
labelFirstName = [[UILabel alloc] initWithFrame:CGRectMake(180, 80, 110, 30)];
[labelFirstName setText:@"Prénom"];
labelFirstName.adjustsFontSizeToFitWidth = YES;
[self.view addSubview:labelFirstName];

// label pour le numéro du destinataire
labelPhoneNumber = [[UILabel alloc] initWithFrame:CGRectMake(30, 130, 110, 30)];
[labelPhoneNumber setText:@"Numéro"];
labelPhoneNumber.adjustsFontSizeToFitWidth = YES;
[self.view addSubview:labelPhoneNumber];

// label pour le numéro du destinataire
labelMailAddress = [[UILabel alloc] initWithFrame:CGRectMake(180, 130, 110, 30)];
[labelMailAddress setText:@"Adresse mail"];
labelMailAddress.adjustsFontSizeToFitWidth = YES;
[self.view addSubview:labelMailAddress];

```

Notez le recours à `adjustsFontSizeToFitWidth` qui va permettre d'ajuster la taille de la police pour afficher entièrement les données (il y a tout de même une taille de police minimale).

Ensuite, nous allons implémenter les méthodes `delegate` de `ABPeoplePickerControllerDelegate`. Nous ne permettrons pas l'affichage de la vue détaillée d'un contact, car nous souhaitons sélectionner quatre de ses propriétés en même temps :

```
#pragma mark -
#pragma mark ABPeoplePickerControllerDelegate
// appelée lorsque l'utilisateur annule
- (void)
peoplePickerControllerDidCancel:(ABPeoplePickerController
➥*)peoplePicker
{
    // on enlève le picker affiché
    [self dismissModalViewControllerAnimated:YES];
}

// appelée lorsque le picker choisit un contact et demande si il peut continuer
(BOOL)peoplePickerController:(ABPeoplePickerController
➥*)peoplePicker shouldContinueAfterSelectingPerson:(ABRecordRef)person
{
    [self dismissModalViewControllerAnimated:YES];
    return NO;
}

// appelé lorsque l'utilisateur choisit une propriété d'un contact
- (BOOL)peoplePickerController:(ABPeoplePickerController
➥*)peoplePicker shouldContinueAfterSelectingPerson:(ABRecordRef)person
property:(ABPropertyID)property identifier:(ABMultiValueIdentifier)identifier
{
    [self dismissModalViewControllerAnimated:YES];
    return NO;
}
```

Affichons maintenant le nom ainsi que le prénom du contact choisi :

```
// appelée lorsque le picker choisit un contact et demande si il peut continuer
(BOOL)peoplePickerController:(ABPeoplePickerController
➥*)peoplePicker shouldContinueAfterSelectingPerson:(ABRecordRef)person
{
    // on récupère le prénom
    NSString* name = (NSString *)ABRecordCopyValue(person,
➥ kABPersonFirstNameProperty); ①
    labelFirstName.text = name;
    [name release];
```

```

// on récupère le nom
name = (NSString *)ABRecordCopyValue(person, kABPersonLastNameProperty);
labelName.text = name;
[name release];
[self dismissModalViewControllerAnimated:YES];
return NO;
}

```

Jusque-là, rien de bien sorcier. Un contact ne pouvant avoir qu'un nom et qu'un prénom, nous copions la propriété prénom (kABPersonFirstNameProperty), par exemple, dans un objet NSString à partir du contact sélectionné person ligne ①.

Par contre, nous souhaitons maintenant permettre de choisir une adresse mail. Or, un contact peut en avoir plusieurs... Toujours dans la même méthode, à la suite de la récupération du nom, ajoutez ces deux lignes :

```

// on récupère le ou les e-mail(s)
ABMultiValueRef emails = ABRecordCopyValue(person, kABPersonEmailProperty);
// on les place dans un tableau
NSArray *theArray = [(id)ABMultiValueCopyArrayOfAllValues(emails) autorelease];

```

Voilà, on a un tableau contenant toutes les adresses mails de notre contact. Pour demander à l'utilisateur de choisir, nous avons besoin d'une UIActionSheet dont chaque bouton présentera une adresse mail, dans le cas où le contact sélectionné a au moins deux adresses mails.

Ajoutez à la suite :

```

UIActionSheet *actionSheet = nil;

if ([theArray count] == 0) {
    // pas d'adresse mail
    labelMailAddress.text = @"Pas de mail";
}
else {
    if ([theArray count] == 1) {
        // une seule adresse
        labelMailAddress.text = [theArray objectAtIndex:0];
    }
    else {
        // on demande à l'utilisateur de choisir
        actionSheet = [[UIActionSheet alloc] initWithTitle:@"Choisir un email"
            → delegate:self cancelButtonTitle:nil destructiveButtonTitle:nil
            → otherButtonTitles:nil];
        // ajouter un bouton pour chaque e-mail
        for (NSString *email in theArray)
            [actionSheet addButtonWithTitle:email];
        [actionSheet showInView:self.view];
    }
}

```

```

}
// on libère
[(id)emails release];
// on release l'action sheet
if (actionSheet) {
    [actionSheet release];
}

```

Comme vous le voyez, self est delegate de l'action sheet, et doit être conforme au protocole **UIActionSheetDelegate** :

```

@interface RootViewController : UIViewController
    ↪ <ABPeoplePickerNavigationControllerDelegate, UIActionSheetDelegate>

```

Ensuite, il faut implémenter la méthode appelée lorsque l'utilisateur sélectionne un bouton de cette liste, puis afficher l'adresse mail sélectionnée :

```

#pragma mark -
#pragma mark actionSheet delegate
- (void)actionSheet:(UIActionSheet *)actionSheet
    ↪ willDismissWithButtonIndex:(NSInteger)buttonIndex
{
    [labelMailAddress setText:[actionSheet buttonTitleAtIndex:buttonIndex]];
}

```

Facile non ? Ajoutons maintenant le choix du numéro de téléphone à la suite. Pour cela, il va falloir jouer un petit peu des coudes... En effet, dans le cas le plus défavorable, où le contact sélectionné à plusieurs adresses mails ainsi que plusieurs numéros de téléphone, il faudra présenter deux objets **UIActionSheet** à la suite. Or, cette vue de sélection s'affiche de manière modale dans la vue, c'est donc la première action sheet affichée qui reçoit les événements. Dans le cas où vous ajoutez immédiatement une autre action sheet par-dessus, elle s'affichera par-dessus la première, mais ce sera celle du mail qui répondra à la sélection... Ça ne marche donc pas... Si le contact a plusieurs adresses mails et plusieurs numéros, il faut sauvegarder le tableau de numéros de téléphone pour l'afficher ensuite. Pour cela, on écrit dans le .h :

```
ABMultiValueRef multiValueToDisplay;
```

et dans le .m (en gras),

```

// appelée lorsque le picker choisit un contact et demande si il peut continuer
- (BOOL)peoplePickerNavigationController:(ABPeoplePickerNavigationController
    ↪ *)peoplePicker shouldContinueAfterSelectingPerson:(ABRecordRef)person {

    // on récupère le prénom
    NSString* name = (NSString *)ABRecordCopyValue(person, kABPersonFirstNameProperty);
    labelFirstName.text = name;
    [name release];

    // on récupère le nom
    name = (NSString *)ABRecordCopyValue(person, kABPersonLastNameProperty);
}

```

```

labelName.text = name;
[name release];

// on récupère le ou les e-mail(s)
aBMultiValueRef emails = ABRecordCopyValue(person, kABPersonEmailProperty);
// on les place dans un tableau
NSArray *theArray = [(id)ABMultiValueCopyArrayOfAllValues(emails) autorelease];

UIActionSheet *actionSheet = nil;

if ([theArray count] == 0) {
    // pas d'adresse mail
    labelMailAddress.text = @"Pas de mail";
}
else {
    if ([theArray count] == 1) {
        // une seule adresse
        labelMailAddress.text = [theArray objectAtIndex:0];
    }
    else {
        // on demande à l'utilisateur de choisir
        actionSheet = [[UIActionSheet alloc] initWithTitle:@"Choisir un email"
            → delegate:self cancelButtonTitle:nil destructiveButtonTitle:nil
            → otherButtonTitles:nil];
        // ajouter un bouton pour chaque e-mail
        for (NSString *email in theArray)
            [actionSheet addButtonWithTitle:email];
        [actionSheet showInView:self.view];
    }
}
// on libère
[(id)emails release];

// on récupère le ou les numéro(s) de téléphone
aBMultiValueRef phoneNumbers =
    → ABRecordCopyValue(person, kABPersonPhoneProperty);
// on les place dans un tableau
theArray = [(id)ABMultiValueCopyArrayOfAllValues(phoneNumbers) autorelease];

if ([theArray count] == 0) {
    // pas de téléphone
    labelPhoneNumber.text = @"Pas de téléphone";
}
else {
    if ([theArray count] == 1) {

```

```

    // un seul numéro
    labelPhoneNumber.text = [theArray objectAtIndex:0];
}

else {
    if(actionSheet)
    {
        // on a déjà une action sheet d'affichée donc on attend avant
        // d'afficher la suivante
        multiValueToDisplay = [(id)phoneNumbers retain];
    }
    else
    {
        // il n'y avait que 0 ou 1 mail
        // on demande à l'utilisateur de choisir
        actionSheet = [[UIActionSheet alloc] initWithTitle:@"Choisir un
        // téléphone" delegate:self cancelButtonTitle:nil
        // destructiveButtonTitle:nil otherButtonTitles:nil];
        // ajouter un bouton pour chaque numéro
        for (NSString *numbers in theArray)
            [actionSheet addButtonWithTitle:numbers];
        [actionSheet showInView:self.view];
    }
}
}

[(id)phoneNumbers release];
// on release l'action sheet
if (actionSheet) {
    [actionSheet release];
}
[self dismissModalViewControllerAnimated:YES];
return NO;
}
}

```

Puis, dans la méthode delegate de UIActionSheetDelegate, faites ces modifications :

```

#pragma mark -
#pragma mark actionSheet delegate
- (void)actionSheet:(UIActionSheet *)actionSheet
    willDismissWithButtonIndex:(NSInteger)buttonIndex
{
    if ([actionSheet.title isEqualToString:@"Choisir un email"]) ①
    {
        [labelMailAddress setText:[actionSheet buttonTitleAtIndex:buttonIndex]];
        if (multiValueToDisplay) {②

```

```

    // on va maintenant demander de choisir parmi les numéros de
    ↪ téléphones disponibles
    NSArray *theArray =
    ↪ [(id)ABMultiValueCopyArrayOfAllValues(multiValueToDisplay) autorelease];

    UIActionSheet *actionSheet = [[UIActionSheet alloc]
    ↪ initWithTitle:@"Choisir un téléphone" delegate:self
    ↪ cancelButtonTitle:nil destructiveButtonTitle:nil
    ↪ otherButtonTitles:nil];
    // ajouter un bouton pour chaque numéro
    for (NSString *numbers in theArray)
        [actionSheet addButtonWithTitle:numbers];
    [actionSheet showInView:self.view];

    [actionSheet release];
    [(id)multiValueToDisplay release];
}
}
else {③
    [labelPhoneNumber setText:[actionSheet buttonTitleAtIndex:buttonIndex]];
}
}
}

```

Ici, on regarde le titre de l'action sheet concernée (nous pourrions également utiliser la propriété tag). Si l'action sheet présente comme titre Choisir un email ligne ①, alors on affiche l'adresse mail choisie. Ensuite on regarde s'il y a plusieurs numéros de téléphone à afficher en testant la valeur de multiValueToDisplay qui ne doit pas pointer vers nil ligne ②. Cela revient à tester l'existence en mémoire de multiValueToDisplay. Dans le cas où l'action sheet présente les numéros (ligne ③), on affiche le numéro choisi.

AJOUTER LES ÉLÉMENTS D'INTERFACE POUR ÉCRIRE UN MESSAGE

Avant d'envoyer un mail ou un SMS, nous allons créer deux champs de texte pour saisir l'objet et le corps du message et deux boutons pour envoyer le SMS ou le mail.

Info

Lorsque vous envoyez un SMS, vous ne pouvez spécifier d'objet. Ce champ de texte concerne donc uniquement l'envoi d'e-mail.

Avant de permettre à l'utilisateur de lancer une fonctionnalité, il faut toujours vérifier si elle est disponible sur l'appareil. On a recours aux méthodes canSendText pour l'envoi de SMS et canSendMail pour l'envoi de mails.

Ajoutez ces lignes dans le `initWithNibName:bundle::`

```
// textField pour l'objet (dans le cas d'un mail)
textFieldSubject = [[UITextField alloc] initWithFrame:CGRectMake(30, 180, 260, 30)];
[textFieldSubject setBorderStyle:UITextBorderStyleRoundedRect];
➥ setPlaceholder:@"Objet"];
textFieldSubject.delegate = self;
[self.view addSubview:textFieldSubject];

// textField pour le contenu
textFieldBody = [[UITextField alloc] initWithFrame:CGRectMake(30, 230, 260, 100)];
[textFieldBody setBorderStyle:UITextBorderStyleRoundedRect];
[textFieldBody setPlaceholder:@"Message"];
textFieldBody.delegate = self;
[self.view addSubview:textFieldBody];

// si l'appareil peut envoyer des sms, on ajoute le bouton
if ([MFMessageComposeViewController canSendText]) {
    // bouton envoi SMS
    UIButton *buttonSendSMS = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [buttonSendSMS setTitle:@"SMS" forState:UIControlStateNormal];
    [buttonSendSMS addTarget:self action:@selector(sendSMS:)
➥ forControlEvents:UIControlEventTouchUpInside];
    [buttonSendSMS setFrame:CGRectMake(30, 350, 110, 30)];
    [self.view addSubview:buttonSendSMS];
}

// si l'appareil peut envoyer des mails, on ajoute le bouton
if ([MFMailComposeViewController canSendMail]) {
    // bouton envoi Mail
    UIButton *buttonSendMail = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [buttonSendMail setTitle:@"Mail" forState:UIControlStateNormal];
    [buttonSendMail addTarget:self action:@selector(sendMail:)
➥ forControlEvents:UIControlEventTouchUpInside];
    [buttonSendMail setFrame:CGRectMake(180, 350, 110, 30)];
    [self.view addSubview:buttonSendMail];
}
```

Il vous faudra donc ajouter le framework `MessageUI` et l'importer dans le .h :

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

Il faut également déclarer les deux champs de texte :

```
UITextField *textFieldBody, *textFieldSubject;
```

De plus, nous souhaitons que la classe RootViewController se conforme au protocole UITextFieldDelegate pour cacher le clavier lorsque l'utilisateur appuie sur le bouton Retour.

```
@interface RootViewController :  
    UIViewController <ABPeoplePickerControllerDelegate,  
    UIActionSheetDelegate, UITextFieldDelegate>
```

Écrire dans le .m :

```
#pragma mark -  
  
#pragma mark text field delegate methods  
  
- (BOOL)textFieldShouldReturn:(UITextField *)textField {  
    // cacher le clavier  
    [textField resignFirstResponder];  
    return YES;  
}
```

Si vous lancez votre application, vous remarquerez qu'en cliquant sur le champ de texte où rentrer le corps de votre message, le clavier vient s'y superposer pour le cacher... Nous allons donc déplacer la vue (la remonter) en même temps que l'apparition du clavier :

```
// appelée lorsque l'utilisateur va éditer  
- (void)textFieldDidBeginEditing:(UITextField *)textField {  
    // on anime le retour de la vue  
    [UIView beginAnimations:nil context:NULL];  
    [UIView setAnimationDuration:0.5];  
    // on déplace la vue de 120 pixels vers le haut  
    self.view.transform = CGAffineTransformMakeTranslation(0.0, -120.0);  
    [UIView commitAnimations];  
}  
  
// appelée lorsque l'utilisateur arrête l'édition  
- (void)textFieldDidEndEditing:(UITextField *)textField {  
    // on déplace la vue en même temps que le clavier pour qu'il ne cache pas le  
    // text field  
    [UIView beginAnimations:nil context:NULL];  
    [UIView setAnimationDuration:0.5];  
    // on remet la vue à son état initial  
    self.view.transform = CGAffineTransformIdentity;  
    [UIView commitAnimations];  
}
```

ENVOYER UN SMS

L'envoi de SMS dans une application est une nouveauté de iOS4. Auparavant, l'envoi de SMS était possible en faisant quitter votre application, et vous ne pouviez passer de corps de message :

```
[[UIApplication sharedApplication] openURL:[NSURL URLWithString:@"sms:0612345678"]];
```

Attention

L'envoi de SMS par votre application utilise le forfait de l'utilisateur. Il devra donc payer s'il ne dispose pas d'un forfait illimité.

Vous ne pouvez pas envoyer de SMS en le cachant à l'utilisateur et sans lui demander confirmation à travers le contrôleur présenté à la Figure 28.2.

Votre application ne pourra recevoir de SMS qui seront, par défaut, affichés dans l'application SMS natives de votre téléphone.

Pour afficher un SMS, écrivez ceci dans le .m, tout simplement :

```
- (void) sendSMS :(id)sender {
    MFMessageComposeViewController *picker =
    ➔ [[MFMessageComposeViewController alloc] init];
    picker.messageComposeDelegate = self;
    // on met le numéro de téléphone
    picker.recipients = [NSArray
    ➔ arrayWithObject:labelPhoneNumber.text];
    // on met le contenu du SMS
    picker.body = textFieldBody.text;

    // on affiche la vue
    [self presentModalViewController:picker
    ➔ animated:YES];
    [picker release];
}
```

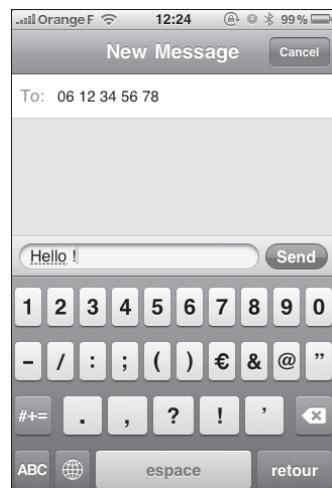


Figure 28.2 : Fenêtre de confirmation de l'envoi d'un SMS.

Cela aura pour effet de placer le numéro dans le champ recipients (destinataires), ainsi que le texte écrit dans le champ body (corps) et d'afficher la vue de confirmation de l'envoi.

Il ne vous aura pas échappé que RootViewController doit se conformer au protocole MFMessageComposeViewControllerDelegate :

```
@interface RootViewController : UIViewController
➔ <ABPeoplePickerNavigationControllerDelegate, UIActionSheetDelegate,
➔ UITextFieldDelegate, MFMessageComposeViewControllerDelegate>
```

Il faut donc implémenter la méthode suivante, appelée lorsque l'utilisateur annule l'envoi ou appuie sur le bouton Envoyer, et procéder au retrait de la vue de l'écran. Cette méthode retourne également le statut de l'envoi. À vous d'avertir l'utilisateur !

```
#pragma mark -
#pragma mark Message SMS delegate
- (void) messageComposeViewController:(MFMessageComposeViewController *)controller
didFinishWithResult:(MessageComposeResult)result {
    switch (result) {
        case MessageComposeResultSent:
            NSLog(@"SMS envoyé");
            break;
        case MessageComposeResultCancelled:
            NSLog(@"SMS annulé");
            break;
        case MessageComposeResultFailed:
            NSLog(@"SMS échec de l'envoi");
            break;
        default:
            break;
    }
    [self dismissModalViewControllerAnimated:YES];
}
```

ENVOYER UN E-MAIL

Envoyer un e-mail ressemble fortement à l'envoi d'un SMS et est régi par les mêmes règles en ce qui concerne la validation par l'utilisateur. L'adresse e-mail d'envoi est celle spécifiée dans les réglages comme étant l'adresse e-mail d'envoi par défaut.

```
- (void) sendMail :(id)sender {
    MFMailComposeViewController *picker = [[MFMailComposeViewController alloc] init];
    picker.mailComposeDelegate = self;
    // on met le destinataire du mail
    [picker setToRecipients:[NSArray arrayWithObject:labelMailAddress.text]];
    /* on peut aussi spécifier
     // les gens en copie
     [picker setCcRecipients:];
     // les gens en copie cachée
     [picker setBccRecipients:];
     */
    // on met le sujet du mail
    [picker setSubject:textFieldSubject.text];
    // on met le contenu du mail. On peut choisir de mettre du contenu html
    [picker setMessageBody:textFieldBody.text isHTML:NO];
    // on affiche la vue
```

```
[self presentModalViewController:picker animated:YES];
[picker release];
}
```

Vous pouvez également envoyer un mail avec un contenu HTML :

```
[picker setMessageBody:messageHTML isHTML:YES];
```

Pour finir, implémentez le protocole MFMailComposeViewControllerDelegate :

```
@interface RootViewController : UIViewController
    ↵ <ABPeoplePickerNavigationControllerDelegate, UIActionSheetDelegate,
    ↵ UITextFieldDelegate, MFMessageComposeViewControllerDelegate,
    ↵ MFMailComposeViewControllerDelegate>
```

et :

```
#pragma mark -
#pragma mark Mail compose delegate
- (void) mailComposeController:(MFMailComposeViewController *)controller
    ↵ didFinishWithResult:(MFMailComposeResult)result error:(NSError *)error {
    switch (result) {
        case MFMailComposeResultSent:
            NSLog(@"Mail envoyé");
            break;
        case MFMailComposeResultSaved:
            NSLog(@"Mail sauvegardé");
            break;
        case MFMailComposeResultFailed:
            NSLog(@"Mail échec de l'envoi");
            break;
        case MFMailComposeResultCancelled:
            NSLog(@"Mail annulé");
            break;
        default:
            break;
    }
    [self dismissModalViewControllerAnimated:YES];
}
```

Cette fiche est maintenant presque terminée, n'oubliez pas le dealloc !

```
- (void)dealloc {
    [labelPhoneNumber release];
    [labelName release];
    [labelFirstName release];
    [labelMailAddress release];
    [textFieldBody release];
    [textFieldSubject release];
    [super dealloc];
}
```

DERNIÈRES REMARQUES

Nous avons vu comment lire le carnet d'adresses de votre appareil, mais vous pouvez également y ajouter des contacts, modifier ceux existants, en créer une copie... Pour cela, référez-vous à la documentation ! En effet, couvrir tous ces aspects mériteraient un ouvrage dédié. Par contre, si vous souhaitez récupérer tout le carnet d'adresses par le code dans un tableau :

```
NSMutableArray *peopleArray =  
    ▶ (NSMutableArray *)ABAddressBookCopyArrayOfAllPeople(ABAddressBookCreate());  
    // utilisation du tableau  
    [peopleArray release];
```

Info

Vous aurez certainement remarqué qu'il y a plus simple pour envoyer un SMS ou un mail, car les contrôleurs dédiés intègrent directement le choix des destinataires... Mais cela ne présentait aucun intérêt dans la rédaction de cette fiche...

Avec iOS4, Apple autorise l'accès au calendrier de votre téléphone pour le consulter et le modifier. Regardons tout ceci de plus près !

Pour l'instant, vous ne pouvez pas créer votre propre calendrier. Les calendriers disponibles sont ceux de l'application Calendrier, stockés dans sa base de données. Par contre, il vous incombe de construire l'interface graphique pour présenter les données à l'utilisateur. Ici, nous allons le faire très simplement avec une **table view**.

Info

Aucune interface graphique prête à l'emploi n'existe pour présenter le calendrier. Cependant, les contrôleurs nécessaires pour créer un nouvel événement ou en modifier un sont, eux, disponibles.

Commencez par créer un nouveau projet de type Navigation-based Application que vous nommerez Event-FromCalendar. Ensuite, ajoutez les deux frameworks EventKit et EventKitUI comme à la Figure 29.1

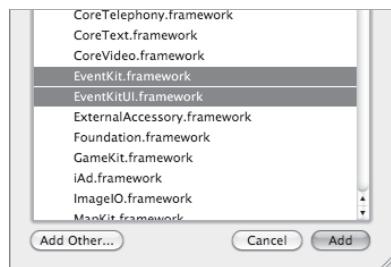


Figure 29.1 : L'ajout des frameworks.

RÉCUPÉRER LES ÉVÉNEMENTS DU CALENDRIER

Dans cette première partie, nous allons afficher les événements des calendriers en les divisant par nom de calendriers. Chaque événement à une propriété `calendar` (`EKCalendar`) qui lui-même a une propriété `title` (`NSString`). L'affichage se fera par section, comme présenté à la Figure 29.2.

Pour commencer, importer les frameworks dans le `RootViewController.h` :

```
#import <EventKit/EventKit.h>
#import <EventKitUI/EventKitUI.h>
```



Figure 29.2 : L'affichage final.

Puis, nous allons déclarer un objet de la classe EKEventStore. Cet objet permet de faire le lien avec la base de données du calendrier sur votre appareil. Déclarez également un tableau qui contiendra les éléments à afficher.

```
@interface RootViewController : UITableViewController {  
    EKEventStore *eventStore;  
    NSMutableArray *eventsToDisplay;  
}  
@property (nonatomic, retain) NSMutableArray *eventsToDisplay;  
@end
```

Dans le .m, ajoutez la ligne en gras

```
@implementation RootViewController  
@synthesize eventsToDisplay;
```

Commençons par allouer notre objet eventStore dans le viewDidLoad :

```
- (void)viewDidLoad {  
    [super viewDidLoad];  
    // titre du contrôleur  
    self.title = @"Calendrier";  
  
    // initialisation de l'event store  
    eventStore = [[EKEventStore alloc] init];  
}
```

Pour afficher les données, nous allons remplir un tableau, déclaré dans le .h. Ajoutez cette ligne, toujours dans le viewDidLoad :

```
// initialisation du tableau avec les événements deux mois avant et un mois  
➥ après la date actuelle  
self.eventsToDisplay = [self getEventsToDisplay];
```

Il faut maintenant définir cette nouvelle méthode getEventsToDisplay. Pour cela, ajoutez cette ligne dans le .h :

```
(NSMutableArray*) getEventsToDisplay;
```

Implémentons cette méthode, dans le .m en commençant par définir un intervalle de temps dans lequel nous allons récupérer tous les événements. Cet intervalle de temps commencera 2 mois avant la date actuelle et finira 1 mois après.

```
- (NSMutableArray*) getEventsToDisplay {  
    // on crée les dates limites pour le tri  
    CFGregorianDate gregorianStartDate, gregorianEndDate;  
    // deux mois avant aujourd'hui  
    CFGregorianUnits startUnits = {0, -2, 0, 0, 0, 0}; // années, mois, jours,  
    ➥ heures, minutes, secondes
```

```
// un mois après aujourd'hui
CFGregorianUnits endUnits = {0, 0, 30, 0, 0, 0};
// récupérer l'heure actuelle dans la zone où vous êtes placés
CFTimeZoneRef timeZone = CFTimeZoneCopySystem();

gregorianStartDate = CFAbsoluteTimeGetGregorianDate(CFAbsoluteTimeAddGrego
➥ rianUnits(CFAbsoluteTimeGetCurrent(), timeZone, startUnits), timeZone);

gregorianEndDate = CFAbsoluteTimeGetGregorianDate(CFAbsoluteTimeAddGrego
➥ rianUnits(CFAbsoluteTimeGetCurrent(), timeZone, endUnits), timeZone);

NSDate* startDate = [NSDate dateWithTimeIntervalSinceReferenceDate:CFGreg
➥ orianDateGetAbsoluteTime(gregorianStartDate, timeZone)];
NSDate* endDate = [NSDate dateWithTimeIntervalSinceReferenceDate:CFGreg
➥ orianDateGetAbsoluteTime(gregorianEndDate, timeZone)];

CFRelease(timeZone);

// L'ajout des méthodes se fera toujours avant la ligne suivante, ici
return nil; // pour l'instant on retourne nil
}
```

Ensuite, il faut récupérer dans la base de données les événements qui correspondent. Regardez dans la documentation la classe EKEvent pour voir tous ses attributs et vous les approprier.

```
// On crée le prédicat
NSPredicate *predicate = [eventStore
➥ predicateForEventsWithStartDate:startDate endDate:endDate calendars:nil];

// On récupère tous les événements qui correspondent
NSArray *events = [eventStore eventsMatchingPredicate:predicate];
```

Nous pourrions nous arrêter ici et afficher directement les événements. Pour corser le tout, je vais vous les faire afficher en utilisant une section par calendrier. Pour cela, nous allons remplir un tableau comme à la Figure 29.3.

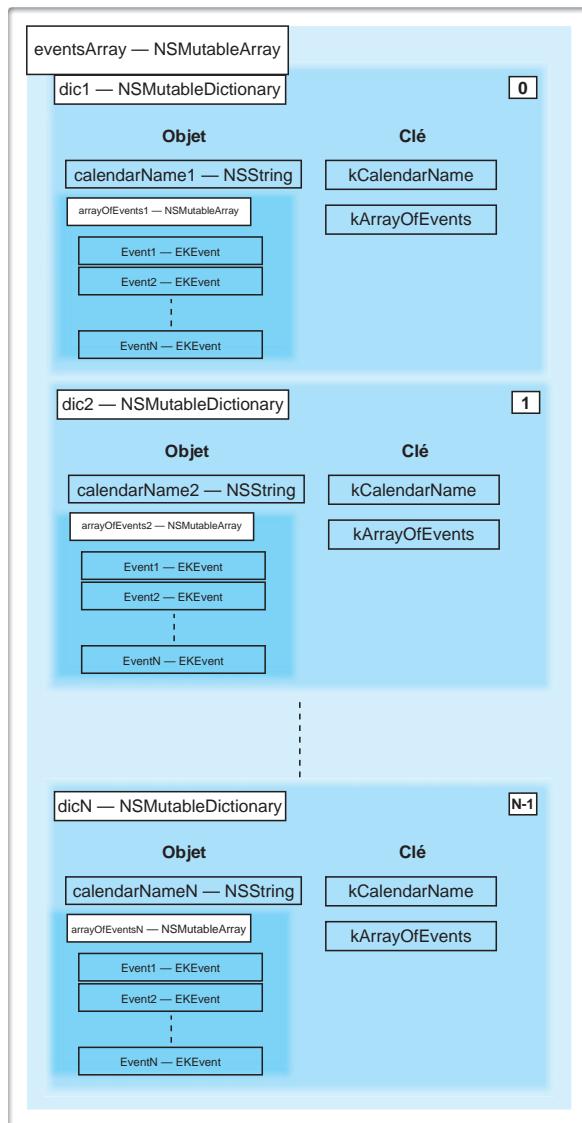


Figure 29.3 : La structure du tableau.

La première chose à faire est de récupérer les noms des calendriers pour tous les événements que l'on vient de prendre depuis la base de données. Pour cela, on crée un nouveau tableau en fonction de `calendar.title`. En fait, `events` est un tableau de `EKEvent`, donc pour chaque objet `event` de cette classe, on peut récupérer le titre du calendrier auquel il appartient par `event.calendrier.title`.

```
// on récupère les différents type de calendrier
NSArray *arrayOfCalendarsWithPotentialDuplication =
    [events valueForKeyPath:@"calendar.title"];
```

Il y a un risque potentiel de doublons, car `valueForKeyPath:` crée un tableau de la même taille que celui spécifié en paramètre. On les élimine facilement, grâce aux propriétés de la classe `NSSet` (qui représente un ensemble d'objets **distincts** et non organisés).

```
// comme il y a un risque de duplication des noms de calendriers, on utilise
// NSSet qui va automatiquement enlever les doublons
NSSet *setCalendars = [NSSet setWithArray:arrayOfCalendarsWithPotentialDuplication];
// on récupère le bon tableau sans doublons
NSArray *arrayOfCalendars = [setCalendars allObjects];
```

Ensuite, on construit le nouveau tableau en énumérant nos différents noms de calendriers :

```
// on construit un tableau de dictionnaires contenant le nom du calendrier et un
// tableau d'événements correspondants à ce calendrier
NSMutableArray *arrayToReturn = [[NSMutableArray alloc] init];
// on parcourt tous les calendriers
for (NSString *calendar in arrayOfCalendars)
{
    // on cherche les événements qui ont pour titre le calendrier en cours
    NSPredicate *predicateCalendar =
        [NSPredicate predicateWithFormat:@"calendar.title == %@", calendar];
    // on crée un nouveau tableau en utilisant ce filtre
    NSMutableArray *arrayOfEvents =
        [[events filteredArrayUsingPredicate:predicateCalendar] mutableCopy];
    // on crée le dictionnaire associé
    NSMutableDictionary *dic = [NSMutableDictionary
        dictionaryWithObjects:[NSArray arrayWithObjects:calendar, arrayOfEvents, nil]
        forKeys:[NSArray arrayWithObjects:kCalendarName, kArrayOfEvents, nil]];
    [arrayOfEvents release]; // release à cause de mutableCopy
    // on l'ajoute dans le tableau
    [arrayToReturn addObject:dic];
}
```

Et on retourne le tableau construit, remplacez cette ligne :

```
return nil; // pour l'instant on retourne nil
```

par :

```
// on retourne le tableau en autorelease cf Fiche mémoire
return [arrayToReturn autorelease];
}
```

N'oubliez pas de déclarer :

```
#define kCalendarName @"Calendar name"
#define kArrayOfEvents @"Array of events"
```

AFFICHER LES ÉVÉNEMENTS DU CALENDRIER

Pour afficher les événements dans notre table view, il faut tout d'abord définir le nombre de sections qui correspond au nombre de calendriers :

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return [eventsToDisplay count];
}
```

Puis, il faut spécifier le nombre de lignes pour chaque section :

```
- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section {
    return [[[eventsToDisplay objectAtIndex:section] objectForKey:kArrayOfEvents] count];
}
```

Affichons le titre de chaque section en ajoutant cette méthode :

```
- (NSString*)tableView:(UITableView *)tableView
    titleForHeaderInSection:(NSInteger)section {
    return [[eventsToDisplay objectAtIndex:section] objectForKey:kCalendarName];
}
```

Puis, affichez chaque événement dans une cellule en modifiant le type de cellule en UITableViewCellStyleSubtitle pour faire apparaître les éléments sur deux lignes :

```
- (UITableViewCell *)tableView:(UITableView *)
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle
            reuseIdentifier:CellIdentifier] autorelease];
    }
    return cell;
}
```

Récupérer l'événement à afficher en ajoutant au-dessus de return cell ; :

```
EKEvent *event = (EKEvent*)[[[eventsToDisplay objectAtIndex:indexPath.section]
    objectForKey:kArrayOfEvents] objectAtIndex:indexPath.row];
```

Ensuite, formatons les dates sur le format Jour Mois Année/Heure:Minute :

```
NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
dateFormatter.dateFormat = @"dd MMMM YY / HH:mm";
NSString *startDate = [dateFormatter stringFromDate:event.startDate];

NSString *endDate = [dateFormatter stringFromDate:event.endDate];
[dateFormatter release];
```

Enfin, affichons dans les labels :

```
cell.textLabel.text = event.title;
cell.detailTextLabel.text = [NSString stringWithFormat:@"%@ -> %@", startDate,
➥ endDate];
```

Voilà, vous devriez avoir un affichage correct !

AJOUTER UN NOUVEL ÉVÉNEMENT

Vous pouvez ajouter un nouvel événement dans un calendrier de deux manières :

- soit en le créant à la main ;
- soit en utilisant un contrôleur tout fait : `EKEVENTEDITVIEWCONTROLLER`. Vous devez passer à ce dernier l'objet de la classe `EKEVENTSTORE` que vous manipulez. Ensuite, si vous laissez sa propriété `event` à `nil`, il créera automatiquement un nouvel événement, sinon, il modifiera celui passé en paramètre.

Ajoutons un bouton + dans la barre de navigation. Dans le `viewDidLoad`, ajoutez :

```
// ajout du bouton ajouter un nouvel événement
UIBarButtonItem *plusButton = [[UIBarButtonItem alloc]
➥ initWithBarButtonSystemItem:UIBarButtonSystemItemAdd target:self
➥ action:@selector(addEvent:)];
self.navigationItem.leftBarButtonItem = plusButton;
[plusButton release];
```

Puis implémentez la méthode `addEvent` :

```
- (void)addEvent:(id)sender {
    EKEVENTEDITVIEWCONTROLLER* controller = [[EKEVENTEDITVIEWCONTROLLER alloc] init];
    controller.eventStore = eventStore;
    controller.editViewDelegate = self;
    [self presentModalViewController: controller animated:YES];
    [controller release];
}
```

Comme vous le voyez, `RootViewController` est delegate de `EKEVENTEDITVIEWDELEGATE`, rajoutez les termes en gras dans le .h :

```
@interface RootViewController : UITableViewController <EKEVENTEDITVIEWDELEGATE> {
```

Et implémentez la méthode appelée lorsque le contrôleur a fini (dans le .m) :

```
- (void)eventEditViewController:(EKEVENTEDITVIEWCONTROLLER *)controller
➥ didCompleteWithAction:(EKEVENTEDITVIEWACTION)action
{
    [self dismissModalViewControllerAnimated:YES];
}
```

Info

Notez que cette méthode passe en paramètre une variable `action` de type `EKEventEditViewAction`. Cette variable peut prendre les valeurs suivantes :

- `EKEventEditViewActionCanceled` Lorsque l'utilisateur annule l'ajout d'un nouvel événement.
- `EKEventEditViewActionSaved` Lorsque l'utilisateur sauvegarde ses modifications.
- `EKEventEditViewActionDeleted` Lorsque l'utilisateur supprime l'événement.

Vous pouvez récupérer l'événement modifié, ajouté, ou supprimé avec `controller.event`.

Regardons une méthode requise demandant le calendrier nécessaire à l'ajout de l'événement. Nous retournerons ici le calendrier par défaut, sachant également que l'utilisateur peut le modifier par la suite dans la vue du contrôleur.

```
- (EKCalendar
    ➔ *)eventEditViewControllerDefaultCalendarForNewEvents:(EKEventEditViewController
*)
    ➔ controller
{
    return eventStore.defaultCalendarForNewEvents;
}
```

L'ajout de nouveaux événements est opérationnel. Problème, ils n'apparaissent pas... En fait, le contrôleur met à jour la base de données des calendriers. Avec le multitâche, une application peut modifier cette base de données, et donc rendre votre affichage obsolète. Nous allons donc ajouter un observateur qui va appeler une méthode chaque fois que la base de données changera, pour la mettre à jour dans votre application. Ajoutez cette ligne dans le `viewDidLoad` :

```
// ajout de l'observation du changement de la base de données
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(storeChanged:)
    ➔ name:EKEEventStoreChangedNotification object:eventStore];
```

Puis, implémentez `storeChanged:` où nous récupérerons la nouvelle base de données pour ensuite relancer l'affichage :

```
- (void)storeChanged:(EKEEventStore*)theEventStore
{
    self.eventsToDisplay = [self getEventsTodisplay];
    [self.tableView reloadData];
}
```

Info

Pour être dans les règles, il ne faudrait exécuter le contenu de la méthode `storeChanged:` que lorsque l'application est visible (le désactiver quand l'application est en fond de tâche).

SUPPRIMER UN ÉVÉNEMENT

Implémentons la possibilité de supprimer un événement de la base de données. Pour cela, ajoutez un bouton d'édition dans la barre de navigation. Écrivez dans le viewDidLoad :

```
// ajout du bouton modifier
self.navigationItem.rightBarButtonItem = self.editButtonItem;
```

Puis, il faut supprimer l'événement lorsque l'utilisateur appuie sur le bouton Supprimer :

```
- (void)tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
    forRowAtIndexPath:(NSIndexPath *)indexPath {
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        // on récupère l'événement à supprimer
        EKEvent *eventToRemove = [[[eventsToDisplay
            objectForKey:kArrayOfEvents]
            objectAtIndex:indexPath.section] objectAtIndex:indexPath.row];
        // on l'enlève de la base de données
        [eventStore removeEvent:eventToRemove span:EKSpanThisEvent error:nil]; ①
        // -> storeChanged sera automatiquement appelé

        // on l'enlève du tableau (pour éviter que ça ne plante à la ligne suivante)
        [[[eventsToDisplay objectAtIndex:indexPath.section] objectForKey:kArrayOfEvents]
            removeObjectAtIndex:indexPath.row];
        // on supprime la vue avec animation
        [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
            withRowAnimation:UITableViewRowAnimationFade];
    }
}
```

Info

Lorsque la ligne ① est exécutée, le contenu de la base de données change, appelant automatiquement la méthode `storeChanged:` qui réaffichera les nouvelles données.

Faites attention au paramètre `EKSpanThisEvent` de cette même ligne qui signifie : "Ne supprimer que cet événement". Il existe également `EKSpanFutureEvents` qui supprimera cet élément et tous les éléments futurs liés (s'il y a récurrence de l'événement). Le mieux est de demander à l'utilisateur ce qu'il souhaite faire !

MODIFIER UN ÉVÉNEMENT

Nous allons solliciter un contrôleur de la classe EKEventViewController pour modifier ou visualiser un événement. L'édition ne sera autorisée qu'en mode Édition. Pour cela, il faut activer la possibilité de sélectionner une ligne en mode Édition en rajoutant dans le viewDidLoad :

```
// autoriser la selection en mode édition  
self.tableView.allowsSelectionDuringEditing = YES;
```

Ensuite, implémentez cette méthode :

```
- (void)tableView:(UITableView *)tableView  
didSelectRowAtIndexPath:(NSIndexPath *)indexPath {  
    EKEventViewController *eventViewController = [[EKEventViewController alloc] init];  
    // on passe l'événement à visualiser ou éditer  
    eventViewController.event = [[eventsToDisplay  
        objectAtIndex:indexPath.section] objectForKey:kArrayOfEvents]  
    // objectAtIndex:indexPath.section] objectForKey:kArrayOfEvents]  
    // objectAtIndex:indexPath.row];  
    // on n'autorise l'édition seulement lorsque la table view est en mode d'édition  
    eventViewController.allowsEditing = self.tableView.editing;  
    // on présente le contrôleur  
    [self.navigationController pushViewController:eventViewController animated:YES];  
    [eventViewController release];  
}
```

Voilà, vous savez accéder aux calendriers présents sur votre appareil.

Pour aller plus loin, sachez que vous pouvez créer des événements plus complexes avec des règles de récurrences poussées. Lorsque c'est possible, utilisez les contrôleurs que nous avons vus avant de créer votre propre interface pour créer/modifier des événements.

Info

N'oubliez pas le dealloc !

```
- (void)dealloc {  
    [eventsToDisplay release];  
    [eventStore release];  
    [[NSNotificationCenter defaultCenter] removeObserver:self];  
    [super dealloc];  
}
```

Lorsque vous souhaitez distribuer votre application, vous avez plusieurs choix parmi un prix fixe à l'achat, l'ajout de publicités incorporées ou le don sous forme d'In App Purchase. Apple a introduit avec iOS4 un service de publicité nommé iAd.

Une application gratuite est en moyenne téléchargée dix fois plus qu'une payante. Ainsi, si vous souhaitez distribuer votre application en plus gros volume, ce choix peut être judicieux. Cependant, vous pouvez espérer obtenir une rémunération pour votre travail. Cela peut se faire sous forme de dons, grâce à l'In App Purchase, ou en intégrant de la publicité à votre application qui vous rémunérera en nombre de clics et de vues. Le choix du type de distribution est primordial dans la gestion de votre projet et n'est pas universel. Une application payante à 1,99 ou plus par exemple, reste perçue comme un gage de qualité, alors qu'une application gratuite peut être considérée comme une version bridée ou une application ne présentant pas de grand intérêt. Rassurez-vous, cette tendance change, mais le public visé ne reste pas le même. Il suffit de comparer les commentaires sur une application gratuite avec une payante pour s'en rendre compte. Gratuit, les gens acquièrent votre application de manière impulsive sans forcément de besoin et vous pouvez ne pas les toucher, donc récolter un mauvais commentaire.

La publicité intégrée se présente sous forme de bannières d'une cinquantaine de pixels de haut. Là encore, quand vous concevez le design de votre application, cette contrainte est à prévoir ! Jusqu'à maintenant, les publicités intégrées provenaient majoritairement de Admob mais Apple vient d'introduire iAd, un service de publicité directement intégré dans le SDK visant à mieux rémunérer les développeurs.

À l'heure où ces lignes sont écrites, je ne connais pas de manière sûre le taux de rémunération (rémunération par vue et par clic) mais Apple conservera 40 % des gains et vous en reversera 60 %. *A priori*, ce sera 10 \$ pour 1 000 impressions (CPM) et 2 \$ par clic, ce taux étant un des plus hauts du marché. La force d'iAd réside dans le HTML5 qui propose une interactivité avec l'utilisateur très poussée. Les publicités deviendront des mini-applications, et seront attractives.

Info

Peut-être vous posez-vous la question suivante : "Et si je vendais mon application, tout en intégrant de la publicité pour doubler mes revenus ?" C'est à déconseiller : un logiciel payant doit être agréable à manipuler, et l'ajout de publicités peut vite agacer. Cette pratique, rare, est très mal perçue par l'acheteur qui ne manquera pas de vous le faire remarquer !

Réfléchissez bien à la position de votre bannière publicitaire, pour que l'utilisateur ait plus de chances de cliquer dessus malencontreusement, sans le forcer non plus. Cette astuce peut vous choquer, mais elle se révèle plutôt efficace et des études ont montré que la rémunération était plus importante lorsque la bannière publicitaire était placée en bas de l'écran plutôt qu'en haut.

INTÉGRER LE PLUS SIMPLEMENT DU MONDE iAd

Commencez par créer un nouveau projet Window-based Application que vous nommerez iAd. Ajoutez un nouveau fichier de type UIViewController que vous nommerez RootViewController (laissez coché Use XIB for user interface). Initialisez-le dans l'application delegate, puis ajoutez sa vue à window. Ajoutez dans votre projet le framework iAd. N'oubliez pas de spécifier la taille de la vue du RootViewController pour qu'elle prenne en compte la présence ou non de la barre de statut :

```
- (BOOL)application:(UIApplication *)application  
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    rootViewController = [[RootViewController alloc]  
        initWithNibName:@"RootViewController" bundle:nil];  
    rootViewController.view.frame = [[UIScreen mainScreen] applicationFrame];  
    [window addSubview:rootViewController.view];  
    [window makeKeyAndVisible];  
    return YES;  
}
```

Dans RootViewController.h, importez ce framework et déclarez un objet de la classe ADBannerView :

```
#import <UIKit/UIKit.h>  
#import <iAd/iAd.h>  
  
@interface RootViewController : UIViewController {  
    ADBannerView *adView;  
}  
@end
```

Puis, dans RootViewController.m, nous allons ajouter cette bannière en haut de l'écran :

```
- (id)initWithNibName:(NSString *)NibNameOrNil bundle:(NSBundle *)bundleOrNil {  
    if ((self = [super initWithNibName:nibNameOrNilOrNil bundle:nibBundleOrNil])) {  
        adView = [[ADBannerView alloc] initWithFrame:CGRectMakeZero];  
        adView.currentContentSizeIdentifier = ADBannerContentSizeIdentifier320x50;  
        // on choisit d'afficher la bannière selon la taille prédéfinie en mode portrait  
  
        [self.view addSubview:adView];  
    }  
    return self;  
}
```

Info

Encore une fois, n'oubliez pas vos dealloc ! Faites un release de rootViewController dans iAdAppDelegate.m et de adView dans RootViewController.m.

Lancez votre application. Vous devriez obtenir quelque chose d'analogique à la Figure 30.1. Lorsque vous cliquez sur la bannière, une vue remplit l'écran présentant la publicité comme à la Figure 30.2.



Figure 30.1 : La bannière de publicité.

Info

Les dimensions de votre bannière sont donc de 320 × 50 pixels en mode portrait et de 480 × 32 pixels en mode paysage.

L'utilisation de notre bannière n'est pas optimale, il faut implémenter le protocole delegate : `ADBannerViewDelegate`.

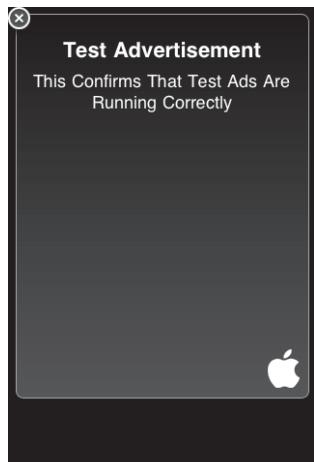


Figure 30.2 : La vue de la publicité.

RÉPONDRE AU PROTOCOLE `ADBANNERVIEWDELEGATE`

Le protocole vous propose ces méthodes :

- `bannerViewDidLoadAd`: Appelée lorsque la publicité a bien été chargée.
- `bannerViewActionShouldBegin:willLeaveApplication`: Appelée lorsque l'utilisateur clique sur la bannière de publicité. En réalisant cette action, la publicité peut amener à faire quitter l'application, vous en serez avertis. Cependant, vous n'avez rien à faire pour sauvegarder l'état actuel de votre application, cela sera réalisé automatiquement. Typiquement, il faut annuler ici toutes les méthodes qui pourraient demander à l'utilisateur une action (ne pas afficher d'alert view, mettre votre jeu en pause...). Cette méthode retourne un booléen. Si vous retournez YES, une page de publicité couvrira votre écran. À l'inverse, retourner NO empêchera l'affichage d'une nouvelle vue.

Info

N'oubliez pas de considérer que le multitâche n'est pas disponible sur tous les appareils !

- `bannerViewActionDidFinish`: Lorsque l'utilisateur revient de la page de publicité, cette méthode sera appelée.
- `bannerView:didFailToReceiveAdWithError`: Votre application doit répondre aux erreurs, il est donc primordial d'implémenter cette méthode. Pour tester, il suffit d'activer le mode avion sur votre appareil.

Nous allons voir comment afficher la publicité si elle est déjà chargée. Cela évite de faire apparaître un bandeau vide disgracieux. Pour cela, modifiez les lignes en gras dans RootViewController.h :

```
@interface RootViewController : UIViewController <ADBannerViewDelegate> {
    ADBannerView *adView;
    BOOL adBannerIsVisible;
}
@end
```

Puis dans le .m :

```
- (id)initWithNibName:(NSString *)NibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
    if ((self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil])) {
        adView = [[ADBannerView alloc] initWithFrame:CGRectMakeZero];
        adView.delegate = self;
        adView.currentContentSizeIdentifier = ADBannerContentSizeIdentifier320x50;
        adView.alpha = 0.0; // met la publicité en transparent

        [self.view addSubview:adView];
    }
    return self;
}
```

Nous allons présenter notre publicité en réalisant un fondu. Vous pouvez également choisir de la déplacer pour la cacher hors de l'écran par exemple.

Ajoutez ces lignes pour afficher la publicité lorsqu'elle est chargée :

```
- (void)bannerViewDidLoadAd:(ADBannerView *)banner
{
    if(!adBannerIsVisible) // si la pub n'est pas déjà affichée
    {
        [UIView beginAnimations:nil context:NULL];
        [UIView setAnimationDuration:1.0];
        adView.alpha = 1.0;
        [UIView commitAnimations];
        adBannerIsVisible = YES;
    }
}
```

Ensuite, on choisit de l'enlever si elle n'a pas pu être chargée :

```
- (void)bannerView:(ADBannerView *)banner didFailToReceiveAdWithError:(NSError *)error
{
    // Appelée lorsque la pub n'a pas réussi à être téléchargée
    if(adBannerIsVisible) // si la pub est visible
    {
        [UIView beginAnimations:nil context:NULL];
        [UIView setAnimationDuration:1.0];
```

```
adView.alpha = 0.0;
    [UIView commitAnimations];
adBannerisVisible = NO;
}
}
```

AFFICHER LA PUBLICITÉ SELON L'ORIENTATION DE L'ÉCRAN

Jusqu'ici, nous avons affiché la bannière de publicité en mode portrait seulement. Pour l'afficher également en mode paysage, il faut tout d'abord spécifier la taille de la bannière pour les différents modes :

```
adView.currentContentSizeIdentifier = ADBannerContentSizeIdentifier320x50;
adView.requiredContentSizeIdentifiers = [NSSet setWithObjects:
➥ ADBannerContentSizeIdentifier320x50, ADBannerContentSizeIdentifier480x32, nil];
➥// à ajouter
adView.alpha = 0.0; // met la publicité en transparent
```

Ensuite, autoriser la rotation de la vue (ici, on autorise toutes les positions) :

```
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
➥ interfaceOrientation {
    return YES;
}
```

Puis changer la taille de la bannière selon l'orientation qui va être prise en compte :

```
- (void)willRotateToInterfaceOrientation:(UIInterfaceOrientation)
➥ toInterfaceOrientation duration:(NSTimeInterval)duration
{
    if (UIInterfaceOrientationIsLandscape(toInterfaceOrientation))
        adView.currentContentSizeIdentifier = ADBannerContentSizeIdentifier480x32;
    else
        adView.currentContentSizeIdentifier = ADBannerContentSizeIdentifier320x50;
}
```

ACTIVER iAD POUR VOTRE APPLICATION

Tout d'abord, vous devez accepter le contrat pour bénéficier de iAd dans vos applications comme à la Figure 30.3. Ensuite, à chaque soumission d'application, il vous sera demandé si vous utilisez iAd comme à la Figure 30.4. Notez qu'une fois que vous avez accepté iAd pour votre application, vous ne pourrez plus l'enlever sans soumettre un nouveau binaire.

CRÉER VOTRE PUBLICITÉ

Vous pouvez également créer votre publicité et devenir annonceur. Pour cela, il faudra utiliser iAd JS qui est une librairie JavaScript disponible dans votre espace développeur.

iTunes Connect Manage Your Contracts

Request New Contracts

Select the agreements which you would like to enter into.

| Request Contract | Contract Region | Contract Type |
|-------------------------------------|-----------------|---------------|
| <input checked="" type="checkbox"/> | World | iAd Network |

Figure 30.3 : Activez le contrat iAd.

iTunes Connect

Enable iAd Advertising Network

Enable Advertising for This Application

The iAd Network gives you an opportunity to earn advertising revenue through ads in your application. [Learn more.](#) ⓘ

- Once your application has been submitted, iAd cannot be disabled. To remove ads from an application, you will need to submit a new binary with ad functionality removed.
- Ads will not begin appearing in your application until you agree to the iAd Network Contract.

My primary target audience is users under 17 years of age. Yes No

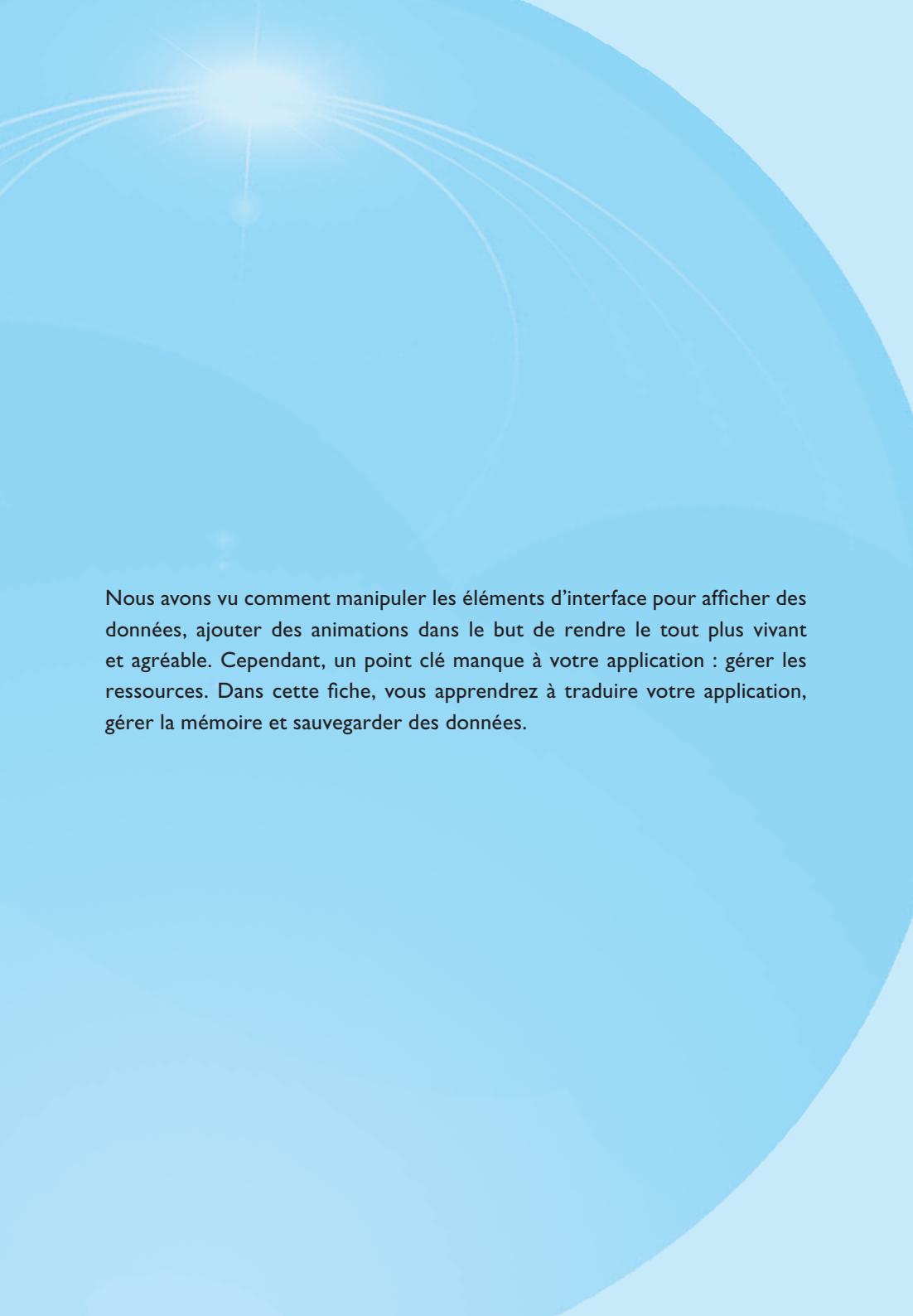
Enable Ads

Go Back **Continue**

Figure 30.4 : Activez ou non iAd lors de la soumission de votre application.

CHAPITRE 6

GESTION DES RESSOURCES



Nous avons vu comment manipuler les éléments d'interface pour afficher des données, ajouter des animations dans le but de rendre le tout plus vivant et agréable. Cependant, un point clé manque à votre application : gérer les ressources. Dans cette fiche, vous apprendrez à traduire votre application, gérer la mémoire et sauvegarder des données.

Le marché français ne représente qu'une petite portion du nombre de ventes d'applications. Pour obtenir gloire et succès, il est indispensable de présenter l'application dans la langue maternelle des plus gros marchés : anglais, français et allemand.

Dans cette fiche, nous allons apprendre quelques rudiments pour localiser votre application et présenter les données en s'adaptant à l'utilisateur. Les données nécessaires qui doivent être traduites sont les suivantes :

- texte affiché à l'écran ;
- les images contenant du texte ou des éléments à traduire ;
- les sons joués à l'utilisateur s'ils sont vocaux ;
- la présentation des données (les dates, les chiffres...).

Votre application doit au minimum être traduite en anglais pour espérer toucher des clients non francophones. Deux opérations sont nécessaires : localiser du texte dans l'application (lorsque le changement de texte est dynamique) et changer les .xib en fonction de la langue (quand vous ne modifiez pas ces éléments dans votre application).

Vous pouvez vous-même opérer cette traduction ou confier les textes à des spécialistes.

Info

Le changement de langue de votre application dépendra des réglages de l'iPhone (en fonction de la langue choisie).

Commencez par créer un nouveau projet que vous nommerez LocalizedApp. Ajoutez ensuite un fichier de la classe UIViewController, avec un .xib, que vous nommerez RootViewController. Affichez sa vue, comme à notre habitude, par-dessus la fenêtre dans l'application delegate.

LOCALISER DU TEXTE

Par défaut, vous pouvez changer la langue en modifiant le .plist comme à la Figure 31.1 (propriété *Localization native development region*). En fait, si votre application ne possède pas la traduction d'une langue, elle la prendra depuis la langue spécifiée. Préférez donc English !

1. Dans le fichier RootViewController.h, ajoutez ces deux lignes dans l'@interface :

```
IBOutlet UILabel *label;  
IBOutlet UIButton *button;
```
2. Ensuite, ouvrez le fichier RootViewController.xib et ajoutez ces deux éléments dans la vue comme à la Figure 31.2.
3. Puis, reliez ces éléments en cliquant tout d'abord sur File's owner.
4. Dans l'onglet Connections de l'Inspecteur, reliez les éléments dans la liste à ceux de votre vue (voir Figure 31.3).

| Key | Value |
|---------------------------------------------|-----------------|
| Information Property List (12 items) | |
| Localization native development region | France |
| Bundle display name | Canada (French) |
| Executable file | China |
| Icon file | France |
| Bundle identifier | Germany |
| InfoDictionary version | Italy |
| Bundle name | Japan |
| Bundle OS Type code | Korea |
| Bundle creator OS Type code | Taiwan |
| Bundle version | United Kingdom |
| Application requires iPhone environment | United States |
| Main nib file base name | MainWindow |

Figure 31.1 : Changez la langue par défaut.



Figure 31.2 : Ajout d'un bouton et d'un label.

| Name | Type |
|-----------------|--------------------|
| File's Owner | RootViewController |
| First Responder | UIResponder |
| View | UIView |

Outlets

- button → Rounded Rect Button
- label → Label (Label)
- searchDisplayController
- view → View

Referencing Outlets

Figure 31.3 : Reliez les éléments.

Nous allons créer un fichier que nous rendrons localisable. En fait, ce fichier sera disponible pour plusieurs langues et contiendra du texte pour notre application. Créez un nouveau fichier de type *strings*, comme à la Figure 31.4. Nommez-le Localizable.

Info

Le nom du fichier est très important, car il est reconnu par le système au même titre que le fichier d'icône *Icon.png* ou l'image de lancement : *Default.png*.



Figure 31.4 :
Création d'un fichier
de type *strings*.

Ensuite, nous allons localiser ce fichier. Pour cela, repérez-le dans Xcode, dans votre projet, puis cliquez du bouton droit dessus et sur Get Info. Rendez-vous dans l'onglet General puis cliquez sur Make File Localizable. Ajoutez ensuite une langue (fr) en cliquant sur Add Localization comme à la Figure 31.5.

Info

Si vous souhaitez ajouter l'italien, ce sera *it*, l'espagnol *es*...

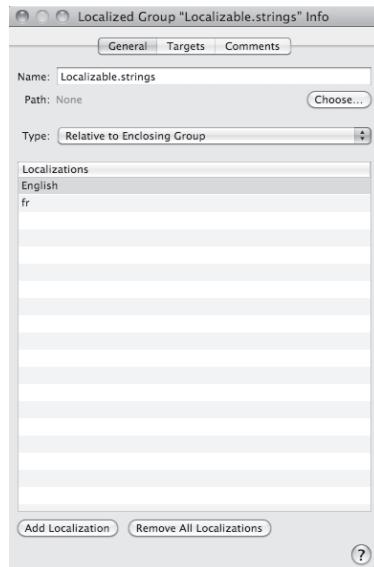


Figure 31.5 : Ajoutez une langue.

Vous devriez obtenir quelque chose de semblable à la Figure 31.6.



Figure 31.6 : Un fichier avec plusieurs langues.

Revenez au RootViewController.m et ajoutez les lignes en gras :

```
- (void)viewDidLoad {
    NSLog(@"La langue actuelle est : %@", [[NSLocale currentLocale]
    ↪ localeIdentifier]);
}

label.text = NSLocalizedString(@"TextLabel", @"Un commentaire");
[button setTitle:NSLocalizedString(@"TextButton", "")]
↪ forState:UIControlStateNormal];
[super viewDidLoad];
}
```

Pour localiser un contenu, il faut donc utiliser NSLocalizedString (clé, commentaire) tout simplement ! Oui mais là, il va traduire tout seul ? Pas du tout ! Il suffit d'associer les traductions aux clés. Pour cela, se rendre dans Localizable.strings et cliquer sur English. Ajouter ensuite ces lignes :

```
"TextLabel" = "Hi there";
```

```
"TextButton" = "Start";
```

On va maintenant traduire en français : cliquez sur fr et ajoutez :

```
"TextLabel" = "Bonjour !";  
"TextButton" = "Commencer";
```

Vous pouvez lancer votre application. Quittez puis changez la langue en vous rendant dans Réglages > Général > International. Vous verrez que vos textes seront traduits selon les différentes langues !

Info

Pour traduire, n'attendez pas le dernier moment pour placer vos `NSLocalizedString`. Rien ne vous empêche d'ajouter autant de fichiers de type strings que de langues disponibles dans votre application à la fin de votre développement. Par contre, remplacer tous les éléments de textes à la fin peut se révéler fastidieux...

Astuce

Si vous souhaitez par exemple afficher Compteur : 1 en français et Counter : 1 en anglais, vous pouvez très bien faire :

```
label.text = [NSString stringWithFormat:@"%@ : %d",  
    ↪ NSLocalizedString(@"CompteurTrad", @""), counterVariable];
```

Si vous souhaitez changer les éléments de place (les constructions de phrases ne sont pas les mêmes suivant les langues) vous pouvez écrire :

Français :

```
"TexteDifferent" = "Prénom : %1$@ ; Nom : %2$@";
```

Anglais :

```
"TexteDifferent" = "Last name : %2$@ ; First name : %1$@";
```

Que vous utiliserez comme ceci :

```
label.text = [NSString stringWithFormat:NSLocalizedString(@"TexteDifferent",  
    ↪@"Prenom nom"), prenom, nom];
```

LOCALISER UN .XIB

Comme pour le fichier strings, vous pouvez localiser vos .xib en cliquant du bouton droit sur le fichier .xib puis Get Info, et Make File Localizable. Cependant, faites ce travail à la fin, car toute modification du fichier .xib devra être reportée dans les fichiers correspondants à chaque langue. De plus, ce travail peut se révéler fastidieux car il faut ouvrir Interface Builder pour chaque fichier, puis modifier un à un les éléments pour les traduire. Si le terminal vous est familier, utilisez-le pour créer des fichiers .strings automatiquement en fonction des éléments dans votre .xib, ce que nous allons voir ici.

Ouvrez le fichier RootViewController.xib et ajoutez un bouton ainsi qu'un label comme à la Figure 31.7.



Label



First Name



Figure 31.7 : Ajoutez un bouton et un label.

Vérifiez ensuite que l'architecture dans le Finder est la même que celle de la Figure 31.8. Si ce n'est pas le cas, modifiez les chemins dans les lignes suivantes commençant par `ibtool`, sinon, déplacez les fichiers pour obtenir la même architecture. N'oubliez pas que vous devrez les réimporter sous Xcode (sans les copier) !

Vous remarquez l'existence des dossiers `.iproj` qui contiennent les éléments de ressources en fonction de la langue.

Ouvrez un terminal :Applications > Utilitaires > Terminal. Voici quelques commandes de base pour les débutants :

- `ls` Permet de lister l'ensemble des fichiers/dossiers à l'endroit où vous êtes.
- `cd UnRépertoire` Permet de rentrer dans UnRépertoire.
- `cd ..` Permet de revenir en arrière dans la hiérarchie.

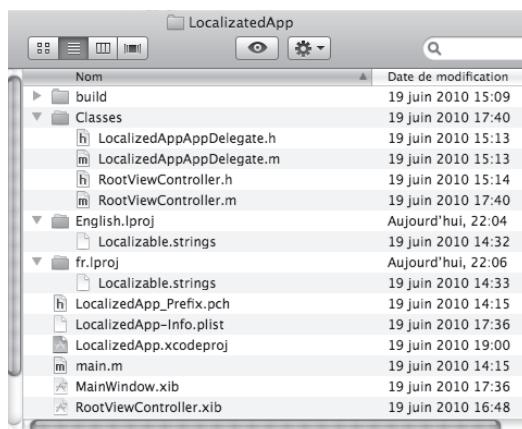


Figure 31.8 : Dans le Finder...

Info

Si votre répertoire s'appelle Un repertoire, (avec un espace dans le nom du répertoire), faites cd Un repertoire pour vous y rendre !

Vous devriez être placé dans le dossier Home de votre ordinateur. Faites ls pour voir les fichiers/dossier à cet emplacement, puis naviguez jusqu'au dossier de votre application avec cd.

Ensuite, exécutez la commande suivante pour créer un fichier .strings contenant les éléments de texte du .xib sélectionné.

```
ibtool --generate-strings-file NomFichier.strings NomFichier.xib
```

Dans notre cas (assurez-vous d'être dans le répertoire qui contient le fichier RootViewController.xib), faites :

```
ibtool --generate-strings-file RootViewController.strings RootViewController.xib
```

Retournez sous Xcode et rendez le fichier RootViewController.strings localisable, comme dans la première partie de cette fiche. Ajoutez la langue fr.

Modifiez les valeurs avec par exemple pour le fichier English :

```
/* Class = "IBUILabel"; text = "Label"; ObjectID = "4"; */
"4.text" = "Label";

/* Class = "IBUILabel"; text = "First Name"; ObjectID = "9"; */
"9.text" = "First Name";

/* Class = "IBUIButton"; normalTitle = "Touch"; ObjectID = "10"; */
"10.normalTitle" = "Touch";
```

et pour la partie française :

```
/* Class = "IBUILabel"; text = "Label"; ObjectID = "4"; */
"4.text" = "Label";

/* Class = "IBUILabel"; text = "First Name"; ObjectID = "9"; */
"9.text" = "Prénom";

/* Class = "IBUIButton"; normalTitle = "Touch"; ObjectID = "10"; */
"10.normalTitle" = "Touche !";
```

Info

Il se peut que les valeurs des clés ne soient pas les mêmes. Pas d'affolement, c'est la valeur qui compte (rappel : "Clé" = "Valeur";)

Il ne vous aura pas échappé que les fichiers RootViewController.strings et Localizable.strings sont tous deux dans les dossiers .lproj correspondants.

Pour créer les .xib pour chaque langue à partir du fichier RootViewController.strings, tapez dans le terminal :

```
ibtool --strings-file Langue.lproj/NomFichier.strings  
↳ --write Langue.lproj/NomFichier.xib NomFichier.xib
```

Ce qui donne dans notre cas :

- Génération du RootViewController.xib pour l'anglais :

```
ibtool --strings-file English.lproj/RootViewController.strings  
↳ --write English.lproj/RootViewController.xib RootViewController.xib
```

- Génération du RootViewController.xib pour le français :

```
ibtool --strings-file fr.lproj/RootViewController.strings  
↳ --write fr.lproj/RootViewController.xib RootViewController.xib
```

Info

Cette procédure présente l'énorme avantage de donner aux traducteurs un fichier texte (le .strings) qui contient les éléments à traduire, sans dévoiler votre interface graphique. De plus, cette méthode est bien plus facile à utiliser que traduire un à un les fichiers .xib.

LOCALISER D'AUTRES RESSOURCES

En plus du texte, votre application peut afficher des données susceptibles d'être différentes selon les pays. Par exemple, si vous présentez des unités, vous pourrez vous servir de [NSLocale currentLocale] pour faire apparaître des mètres ou des miles, des degrés Celsius ou Fahrenheit.

Exemple :

```
if ([[NSLocale currentLocale] objectForKey:NSLocaleUsesMetricSystem] boolValue]  
↳ == YES) {  
    // on utilise les mètres  
}  
else  
    // on utilise les miles  
  
// retourne l'élément (, ou ,) utilisé en tant que virgule  
NSString *separation = [[NSLocale currentLocale]  
↳ objectForKey:NSLocaleDecimalSeparator];
```

Vous trouverez toutes les valeurs dans la documentation de la classe NSLocale.

Un autre exemple frappant est l'affichage de la date. En France, on écrit la date de cette manière 21 Juin 2010 qui correspond en anglais à June 21 2010. Ainsi, pour formater la date en fonction du pays, on utilise NSDateFormatter :

```
NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];  
// choix du type (ici jour mois année)  
[dateFormatter setDateStyle:NSDateFormatterMediumStyle];
```

```
// choix de la langue  
[dateFormatter setLocale:[NSLocale currentLocale]];  
// création d'un objet NSString avec la date actuelle  
NSString *date = [dateFormatter stringFromDate:[NSDate date]];  
NSLog(@"%@", date);  
[dateFormatter release];
```

De même, vous pouvez utiliser NSNumberFormatter pour formater correctement vos chiffres en fonction de la langue du téléphone.

La gestion de la mémoire est primordiale dans le développement de votre application pour la rendre stable (pas de crashes intempestifs). Il faut savoir qu'une application qui plante, notamment à cause de la mémoire mal gérée, est le principal motif de refus de validation de la part d'Apple.

Tout ceci n'est pas banal et peut faire peur à un développeur iPhone débutant. Je vous conseille donc de vous référer à cette fiche aussi souvent que possible, dès que vous avez le moindre doute dans l'écriture de votre code. J'en profite pour vous donner un petit conseil : n'attendez pas le dernier moment (quelques heures avant la soumission) pour vous préoccuper de cette gestion de la mémoire... Essayez de mettre en place un mécanisme d'écriture du code où vous voyez exactement ce qu'il va se passer dans la RAM de l'appareil. Beaucoup d'applications crashent à cause d'une mauvaise compréhension globale de la manipulation des variables.

Info

La gestion mémoire est devenue encore plus importante depuis le passage à iOS4 et les applications qui quittent très rarement (suspendues les trois quarts du temps). En effet, lorsque vous redémarrez votre application, vous repartez de zéro. Par contre, les fuites s'accumulent lorsque vous suspendez/réactivez votre application.

LA MÉMOIRE... DE QUOI PARLE-T-ON ?

Si vous êtes programmeur en C, vous êtes sûrement familier avec le `malloc()` pour allouer de la mémoire et le `free()` pour la libérer. Le système d'exploitation iPhone, contrairement à Mac OS n'intègre pas de *garbage collector* (ou ramasse-miettes). Lorsque vous demandez d'allouer de la mémoire à un objet, vous réservez un emplacement spécifique. Cet emplacement, dans le cas de l'iPhone, ne sera pas libéré automatiquement contrairement à un système avec un ramasse-miettes, qui serait capable de déterminer si l'emplacement mémoire réservé est utile ou non. Par contre, quand votre application quittera, la mémoire sera libérée et n'affectera pas le système.

La mémoire de votre appareil, que ce soit l'iPhone, l'iPod ou l'iPad est limitée. Il existe donc deux cas de figures :

- Vous allouez de la mémoire à des objets, mais vous ne la libérez jamais. Il y a donc ce que l'on appelle des fuites (*leaks*) qui vont perturber le comportement de votre application jusqu'à la faire planter.
- Vous allouez de la mémoire à beaucoup d'objets, sans libérer les anciens dont vous n'avez plus nécessairement l'utilité. Votre application va donc saturer en mémoire et planter.

Dans ce dernier cas, vous devrez implémenter la méthode `didReceiveMemoryWarning` qui sera appelée lorsque le système de l'iPhone n'aura plus de mémoire disponible pour votre application. Dans cette méthode, il faudra donc libérer tous les objets inutiles. Si vous n'implémentez pas cette méthode, vous prenez le risque de voir votre application cracher un jour ou l'autre.

QUELQUES NOTIONS

Commençons par expliquer les termes que vous rencontrerez tout au long de votre aventure. Chaque objet a ce que l'on appelle un compteur de référence ou en anglais *retainCount*. C'est un nombre, positif ou nul, qui informe du nombre d'allocations d'un objet.

- **Retain.** Augmente de 1 le compteur de référence d'un objet.
- **Release.** Diminue de 1 le compteur de référence d'un objet.
- **Autorelease.** Diminue de 1 le compteur de référence d'un objet à un certain moment dans le futur.
- **Alloc.** Alloue de la mémoire pour un objet, et le retourne avec un compteur à 1.
- **Copy.** Fais une copie d'un objet et le retourne avec un compteur à 1.

La vie de votre objet compte trois étapes : la création, la gestion, puis la destruction.

La **création** d'un objet passe par l'appel à la méthode `alloc` de `NSObject` puis par son initialisation, par défaut, l'appel à la méthode `init` (vous pouvez créer vos propres méthodes).

Par exemple,

```
MonObjet *objet = [[MonObjet alloc] init];
```

Après cette ligne, le compteur de référence de objet sera égal à 1.

VOTRE MÉTHODE INIT

Vous pouvez créer votre propre méthode d'initialisation. Par exemple, si vous souhaitez initialiser un objet d'une classe `Personne` en lui passant un nom et un prénom :

```
Personne *unePersonne = [[Personne alloc] initWithLastName:@"Dupont" andFirstName:@"Thierry"];
```

Avec votre méthode d'initialisation :

```
- (void) initWithLastName:(NSString*)lastName andFirstName:(NSString*)firstName {
    if(self = [super init]) ①
    {
        self.lastName = lastName;
        self.firstName = firstName;
    }
    return self;
}
```

La ligne ① mérite que l'on s'y arrête un instant. Que fait-on ? Tout d'abord, il y a l'appel à la méthode `init` de `super` (`super` désigne la classe dont on hérite, ici `NSObject` (`Personne` est une sous-classe de `NSObject`)) avec `[super init]`. On affecte donc à `self` la valeur renournée (l'objet initialisé par `[super init]`). Puis, on teste la valeur de `self`. `if(self = [super init])` revient à écrire `if((self = [super init]) != nil)` ce qui revient à tester si `self` ne pointe pas vers `nil`, donc existe.

Ensuite, vient la phase de vie de votre objet. C'est là où il sera copié, modifié, sauvegardé dans un tableau. Prenons par exemple :

```
MonObjet *objet = [[MonObjet alloc] init];
// retain count à 1
[objet retain];
// retain count à 2
[objcet release];
// retain count à 1
[objcet release];
// retain count à 0 -> l'objet est détruit
```

À la dernière ligne, avec release, on détruit l'objet en mémoire car son compteur de référence atteint 0. Ainsi, la méthode dealloc de la classe MonObjet est appelée. La méthode dealloc doit donc être implémentée. Typiquement, elle doit contenir entre autres :

- la libération de tous les objets déclarés dans le .h et utilisés dans votre fichier d'implémentation (le .m) ;
- l'arrêt des observations (KVO) avec par exemple [[NSNotificationCenter defaultCenter] removeObserver:self];
- l'arrêt des timers (NSTimer).

Ici, le dealloc serait :

```
- (void) dealloc {
    [lastName release];
    [firstName release];
    [super dealloc];
}
```

ATTENTION

Dans le cas où Interface Builder gère un élément d'interface, avec le mot-clé IBOutlet, vous ne devez pas release l'objet (sauf cas suivant) car ce n'est pas vous qui le possédez. Cependant, lorsque vous avez un objet géré par Interface Builder, et que vous utilisez les accesseurs (@property et @synthesize), vous devez libérer l'objet dans le dealloc.

UTILISER LES ACCESSEURS

Comme dans tout langage objet, vous pouvez affecter des valeurs à un objet d'une instance de classe. Par exemple avec la classe Personne, vous souhaitez changer le nom (name) d'un objet présentant de cette classe.

En utilisant retain, qui va incrémenter le compteur de référence :

```
- (void) setName:(NSString*)newName {
    if(newName != name) // on ne met à jour que si le nouveau est différent
```

```

    {
        [name release];
        name = [newName retain]; // on incrémenté de 1 le retainCount de
        ➔ newName, il vaut donc 2
    }
}

```

Vous utilisez de cette manière cette méthode :

```

Personne *president = [[Personne alloc] init];
NSString *presidentName = [[NSString alloc] initWithString:@"Dupont"]; ①
[president setName:presidentName]; ②
[presidentName release]; ③
// plus tard
[president release];

```

Info

Faites très attention à la petite subtilité suivante : `if(newName != name)` ne vient pas tester si les deux objets `NSString` ont la même valeur, mais si ce ne sont pas les mêmes en terme de valeur d'adresse. Par exemple, si l'adresse de `newName` est `0x45674` et celle de `name` est `0x25234`, alors les adresses sont différentes donc la condition est vraie. De plus, si la valeur de `newName` est @"toto" et celle de `name` est également @"toto", la condition sera toujours vraie.

Pour tester si deux objets `NSString` ont des valeurs différentes, il faut écrire `if([newName isEqualToString:name] == NO)`

Il est nécessaire de bien comprendre ce qu'il se passe ici :

Ligne ①, on alloue et initialise un nouveau nom de président. `presidentName` a donc son compteur de référence à 1.

Ligne ②, on vient changer la propriété `name` de `president`. Dans `setName:`, on `release` l'ancien `name`, puis on incrémenté le compteur de référence de `newName`, soit `presidentName`, de 1 qui vaut donc 2 maintenant. Ensuite, on fait pointer `name` vers `newName` (on fait une référence).

Puis, ligne ③, on `release` `presidentName`, son compteur de référence est décrémenté et vaut donc 1. Ainsi, l'objet `name` de `president` pointe vers le `NSString` alloué avec Dupont.

On peut également utiliser `copy` :

```

- (void) setName:(NSString*)newName {
    if(newName != name) // on ne met à jour que si le nouveau est différent
    {
        [name release];
        name = [newName copy]; // on copie newName. name pointe alors vers cette copie
    }
}

```

En principe, vous n'aurez pas à écrire les accesseurs explicitement, car l'Objective-C introduit `@property` et `@synthesize` pour le faire à votre place et alléger votre code. Ainsi, écrire `@property(retain) NSString *name` dans le `.h` et `@synthesize name` dans le `.m` revient exactement au même que d'implémenter la méthode `setName:` avec comme comportement `retain` (vous n'avez donc pas à écrire cette méthode car elle est implicite).

De fait, vous pourrez changer le nom avec l'une de ces deux lignes :

```
[president setName:newName];
// ou
president.name = newName;
```

Info

Vous trouverez très souvent des accesseurs écrit `@property(nonatomic, copy) NSString *name` par exemple. Mais que signifie le mot-clé en gras ? Si vous utilisez `nonatomic`, cela sous-entend que plusieurs threads pourront accéder en même temps à l'objet `name`. Ainsi, il peut potentiellement y avoir quelques soucis car la ressource ne sera pas bloquée. Par contre, si vous recourez à `atomic`, qui est le comportement par défaut si vous ne spécifiez rien, votre objet ne sera détenu que par un seul thread à la fois.

Question performances, l'accès à un objet avec `nonatomic` est plus rapide qu'avec un objet `atomic`.

L'AUTORELEASE

Peut-être vous posez-vous la question en ce moment même : comment faire pour ne pas avoir de fuites dans ce cas-là ?

```
- (NSString*) returnUnStringEnMajuscule:(NSString*)aString {
    NSString *stringARetourner;
    stringARetourner = [[NSString alloc] initWithFormat:@"%@", aString,
    ↪ [aString uppercaseString]];
    // release n'est jamais appelé ! -> fuite
    return stringARetourner;
}
```

Dans cet exemple de code, on alloue un objet, `stringARetourner` que l'on retourne mais cet objet n'est pas libéré de la mémoire ! En effet, parce que sa déclaration est locale, dès lors que l'on sort de la fonction, les objets et variables déclarées localement ne sont plus accessibles. Cela ressemble à une mission impossible, car il n'est pas possible de faire :

```
- (NSString*) returnUnStringEnMajuscule:(NSString*)aString {
    NSString *stringARetourner;
    stringARetourner = [[NSString alloc] initWithFormat:@"%@", aString,
    ↪ [aString uppercaseString]];
    return stringARetourner;
    [stringARetourner release]; // trop tard
}
```

En effet, le return quitte la méthode, la ligne où l'on release stringARetourner ne sera jamais appelée ! Il n'est pas possible non plus de faire :

```
- (NSString*) returnUnStringEnMajuscule:(NSString*)aString {
    NSString *stringARetourner;
    stringARetourner = [[NSString alloc] initWithFormat:@"%@", aString,
    ↪ [aString uppercaseString]];
    [stringARetourner release]; // trop tôt !
    return stringARetourner;
}
```

Ici, le release apparaît bien trop tôt ! L'objet retourné n'existera plus en mémoire... Cela ne convient pas. Dans ce cas de figure, on utilise l'autorelease, on parle également de bassin d'autorelease (NSAutoreleasePool). Concrètement, au lieu d'appeler release sur un objet, et donc libérer la mémoire immédiatement, l'appel d'autorelease place l'objet dans une sorte de boucle. Cette boucle contiendra donc un nombre d'objets en attente d'être libérés. À un instant T, lorsque le processus courant aura fini sa tâche, tous les objets dans la boucle seront libérés.

La solution est donc la suivante :

```
- (NSString*) returnUnStringEnMajuscule:(NSString*)aString {
    NSString *stringARetourner;
    stringARetourner = [[NSString alloc] initWithFormat:@"%@", aString,
    ↪ [aString uppercaseString]];
    [stringARetourner autorelease];
    return stringARetourner;
}
```

ALLER PLUS LOIN AVEC L'AUTORELEASE

Par défaut, les objets s'enregistrent dans le bassin d'autorelease le plus proche, très souvent celui créé dans main.m :

```
#import <UIKit/UIKit.h>
int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

Cependant, vous pouvez (devez) créer vous-même votre bassin d'autorelease lorsque :

- vous détachez un nouveau thread : la création d'un bassin d'autorelease pour le thread est obligatoire ;
- vous allouez beaucoup d'objets en même temps.

Prenons ce morceau de code :

```
NSString *unString = @"";
for (int i = 0; i < 1000000; i++)
{
    unString = [unString stringByAppendingString:[NSString stringWithFormat:@"%d", i]];
}
return unString;
```

(Le résultat va donner 01234567891011121314....999998999999).

Ici, on remarque que l'on crée beaucoup d'objets dans une boucle qui a 1 000 000 d'itérations. D'ailleurs, les méthodes `stringByAppendingString:` et `stringWithFormat:` sont toutes deux des méthodes de classe retournant un objet (`NSString`) en autorelease. Ainsi, à chaque itération, sont placés deux nouveaux objets dans le bassin d'autorelease.

L'effet sera le suivant : lorsque le bassin d'autorelease libérera les objets présents, votre application risquera de geler, car cette libération ne sera pas instantanée.

Une solution est bien entendu d'utiliser `release` directement :

```
NSString *unString = @"";
for (int i = 0; i < 100000; i++)
{
    NSString *unAutreString = [[NSString alloc] initWithFormat:@"%@%d", unString, i];
    [unString release]; // on release l'ancien unString
    unString = unAutreString; // on pointe vers unAutreString
}
return unString;
```

Mais si vous souhaitez passer par des méthodes qui retournent des objets en autorelease, alors :

```
NSString *unString = @"";
for (int i = 0; i < 1000000; i++)
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init]; ①
    NSString *unAutreString = [unString stringByAppendingString:[NSString
        stringWithFormat:@"%d", i]]; ②
    [unString release]; ③
    [unAutreString retain]; ④
    unString = unAutreString; ⑤
    [pool release]; ⑥
}
return unString;
```

Que se passe-t-il exactement ici ?

Ligne ①, on crée un nouveau bassin d'autorelease. Ainsi, ligne ②, les objets créés en autorelease sont placés dans le bassin le plus près, c'est-à-dire celui de la ligne ①.

Ligne ③, on release l'ancien unString car nous n'en avons plus besoin.

Ligne ④, on applique retain à unAutreString pour éviter que le bassin d'autorelease alloué ligne ① ne le détruise !

Enfin, ligne ⑤ on fait pointer unString vers unAutreString puis on release le bassin ligne ⑥ ce qui détruira tous les objets présents dans ce bassin. Notez que l'on pourrait être un peu plus malin en ne créant/détruisant le bassin d'autorelease que toutes les 1 000 itérations par exemple.

Info

Il faut faire très attention avec l'utilisation de l'autorelease. En effet, lorsque vous débutez, cela semble magique car vous n'avez pas à vous préoccuper explicitement du cycle mémoire de votre application. Grave erreur ! Je vous conseille très fortement de ne recourir à l'autorelease que dans les cas particuliers comme ceux présentés ci-dessus. Certes, cela nécessitera un temps d'adaptation et de compréhension, mais les gains seront immenses. Un des problèmes majeurs de l'autorelease est que vous ne pouvez prédire le moment exact où les objets seront détruits.

CONSEILS ET ERREURS

QUELQUES CONSEILS SIMPLES

- **Utiliser les accesseurs.** Lorsque vous écrivez par exemple :

```
NSString *nouveauString = [[NSString alloc] initWithString:@"toto"];
[unString release];
unString = [nouveauString copy];
[nouveauString release];
```

Il est préférable d'écrire :

```
NSString *nouveauString = [[NSString alloc] initWithString:@"toto"];
self.unString = nouveauString;
[nouveauString release];
```

avec dans le .h :

```
@property (nonatomic, copy) NSString *unString;
```

et dans le .m :

```
@implementation VotreClasse
@synthesize unString;
```

- **Ne pas allouer/détruire des objets inutilement.** Si vous avez un objet dont une propriété change, préférez modifier la propriété de l'objet plutôt qu'en recréer un nouveau. Il faut donc réaliser ce travail de réflexion en amont lors de la conception de vos classes.
- **Le simulateur est un piège.** Je ne vous le rappellerai jamais assez : le simulateur consomme les ressources de votre ordinateur; le comportement mémoire est donc totalement faussé, de même que l'analyse des fuites avec les instruments.

DES ERREURS CLASSIQUES

Ces erreurs sont issues de la réalité et reviennent fréquemment sur le forum. Lisez attentivement cette section pour en comprendre les subtilités !

Un tableau gère lui-même la mémoire des objets qu'il contient (de même qu'un dictionnaire (NSDictionary, ...)). Par exemple, pour remplir un tableau de 10 nombres, vous avez 2 solutions :

```
NSMutableArray *array = [[NSMutableArray alloc] init];
int i;
// ...
for (i = 0; i < 10; i++)
{
    NSNumber *n = [NSNumber numberWithInt:i];
    [array addObject:n];
}
```

ou :

```
NSMutableArray *array = [[NSMutableArray alloc] init];
int i;
// ...
for (i = 0; i < 10; i++)
{
    NSNumber *n = [[NSNumber alloc] initWithInt: i];
    [array addObject: n];
    [n release];
}
```

Dans le dernier cas, il n'y a aucune raison de ne pas faire le `release` sur `n`, car `addObject:` fais automatiquement un `retain` sur l'objet passé en paramètre.

Veuillez à ne pas `release` plus que nécessaire un objet. En effet :

```
rootViewController = [[RootViewController alloc] init];
[rootViewController release];
[rootViewController release];
```

Générera ce message d'erreur :

```
malloc: *** error for object 0x5e276f0: pointer being freed was not allocated
*** set a breakpoint in malloc_error_break to debug
```

Ne pas allouer plus que nécessaire non plus ! Par exemple, ceci est incorrect :

```
NSString *unAutreString = [[NSString alloc] initWithString:@"toto"];
// ....
NSString *unString = [[NSString alloc] init]; ①
unString = unAutreString; ②
```

Ligne ①, on alloue de la mémoire pour `unString`. Ensuite, ligne ②, on fait pointer `unString` vers `unAutreString`. L'emplacement mémoire réservé pour `unString` initialement est donc perdu à jamais, il y a une fuite !

L'OUTIL D'ANALYSE STATIQUE

Un outil d'analyse statique est inclus depuis Xcode 3.2. Cet outil va parcourir votre code et appliquer certaines méthodes pour déterminer les erreurs. Il est utile par exemple pour regarder les erreurs de gestion mémoire. Créez un nouveau projet Window-based Application que vous nommerez LeaksDetection, et écrivez ceci dans l'application delegate :

```
- (BOOL)application:(UIApplication *)application  
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(20, 20, 80, 30)];  
    [window addSubview:label];  
  
    NSNumber *number = [[NSNumber alloc] initWithFloat:45.67f];  
  
    label.text = [NSString stringWithFormat:@"%@", [number floatValue]];  
  
    [window makeKeyAndVisible];  
    return YES;  
}
```

Ici, on crée donc deux fuites : l'une concerne l'objet label qui n'est pas libéré et l'autre number qui, lui aussi, n'est pas détruit. Vous remarquerez cependant que votre projet compile sans warnings, et que le label est bien affiché à l'écran.

Lançons l'outil d'analyseur statique : faites Build > Build and Analyze. L'outil trouve deux erreurs comme à la Figure 32.1.

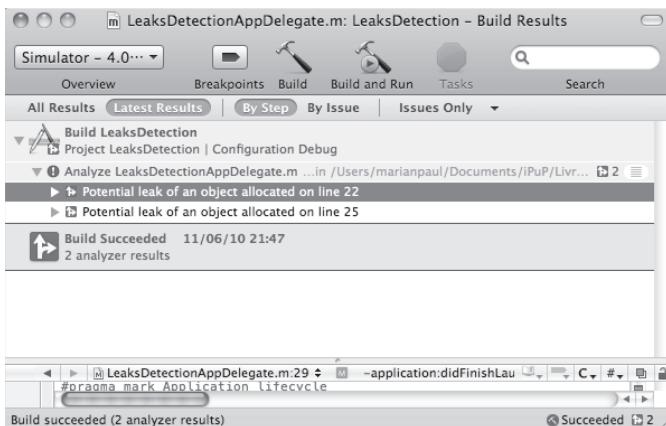


Figure 32.1 :
Les résultats de
l'analyse statique.

Comme vous le constatez, les deux erreurs potentielles sont des fuites mémoire. Cliquez sur un des résultats pour voir le détail comme à la Figure 32.2. Vous apercevez que l'outil vous explique votre erreur ! À l'annotation numéro 1, vous allouez un label, avec un retain count de 1, ce qui est normal. Par contre, à l'annotation numéro 2, l'outil vous dit qu'à partir de ce point, il ne trouve plus aucune référence à l'objet label, qui n'est toujours pas release. Il y a donc une fuite potentielle.

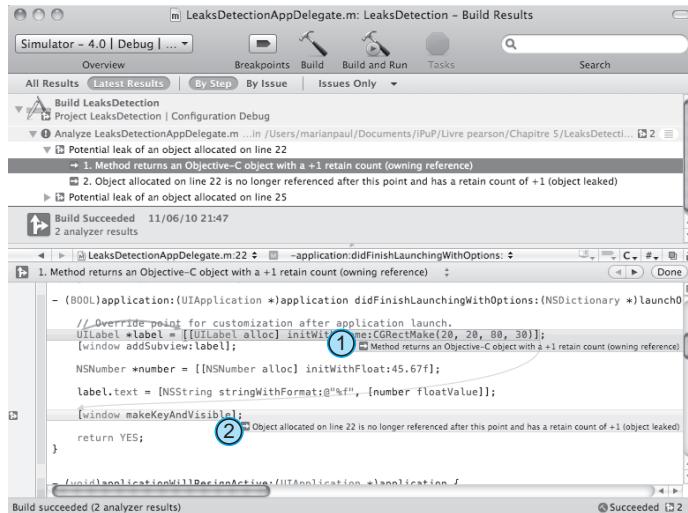


Figure 32.2 :
Les explications
de l'outil.

Info

Il faut tout de même faire attention à cet outil et réfléchir avant de corriger. Je vous conseille même de prendre un snapshot de votre projet avant de commencer à corriger ces erreurs en faisant File > Make Snapshot. Cela aura pour effet de prendre votre projet en 'photo', et pouvoir revenir à cette version si vous rencontrez des erreurs par la suite. N'oubliez jamais de tester le comportement de votre application après toute correction de fuite.

Ici, il peut être tentant de corriger la fuite comme ceci, en ajoutant la ligne en gras :

```
UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(20, 20, 80, 30)];
[window addSubview:label];
[label release];
```

C'est une erreur ! En effet, deux lignes après, vous changez le texte du label ce qui fera planter l'application car label n'existe plus. Il faut donc placer le release après le changement de la propriété text. Cet outil vous explique vos erreurs, mais ne vous les résout pas !

L'analyseur statique permet aussi de vous prévenir d'erreurs autres que de potentielles fuites mémoire. Par exemple, en ajoutant ces lignes de code :

```
int i;
if (i == 3)
    NSLog(@"i = %d", i);
```

L'outil vous signalera : "Variable 'i' declared without initial value" signifiant que la variable 'i' n'a pas été initialisée.

Si vous souhaitez compiler chaque fois à l'aide de l'analyseur statique, rendez-vous dans Project > Edit Project Settings, puis dans l'onglet Configuration ① comme à la Figure 32.3.

Ensuite, nous allons dupliquer le mode de compilation debug. Pour cela :

1. Sélectionnez-le, puis cliquez sur duplicate ② et renommez-le en Analyzer.
2. Reportez-vous maintenant à la capture écran de la Figure 32.4. Rendez-vous dans l'onglet Build et choisissez la configuration Analyzer ①.
3. Cochez Run Static Analyzer pour cette configuration dans Build options ②.
4. Fermez la fenêtre et vous pouvez maintenant choisir de compiler en mode Analyzer comme à la Figure 32.5.



Figure 32.3 : La configuration de votre projet.

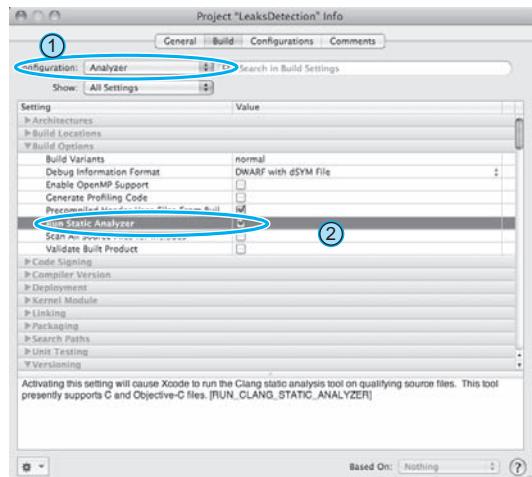


Figure 32.4 : Changez les options de compilation.



Figure 32.5 : Choisissez le mode de compilation Analyzer.

UTILISER UN INSTRUMENT... LEAKS !

Pour vérifier les fuites dans votre code, vous pouvez également lancer un instruments qui va analyser le déroulement de votre application.

Pour lancer l'outil Leaks, cliquez sur Run > Run with Performance Tool > Leaks. Patientez quelques instants avant que l'instrument ne mesure les fuites. Vous devriez obtenir quelque chose d'analogique à la Figure 32.6.

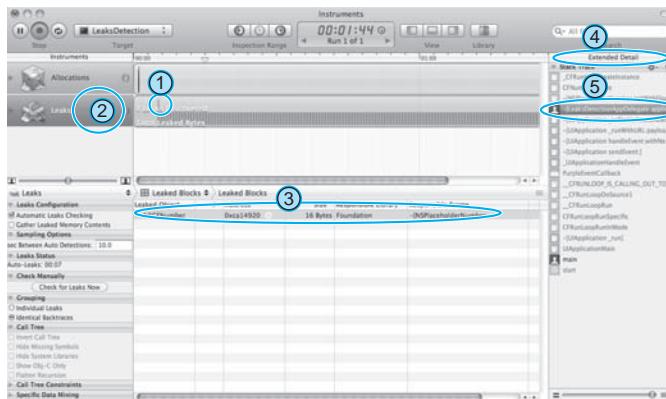
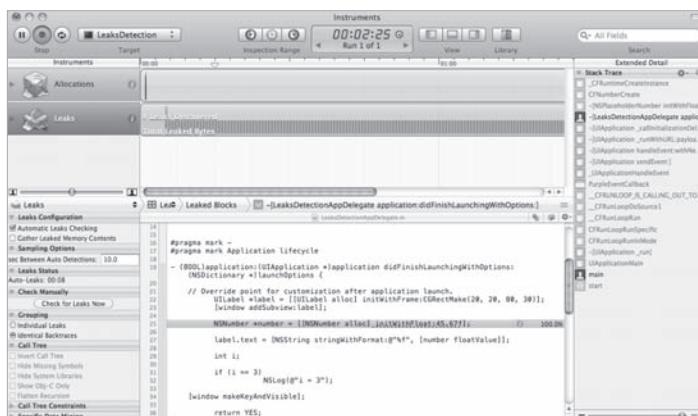


Figure 32.6 : La fenêtre d'accueil de Leaks.

En ①, en orange, est représenté l'événement une fuite a été détectée. Si vous cliquez sur la ligne représentant les fuites ②, vous obtiendrez une liste des fuites détectées en ③. Ces informations ne permettent pas pour l'instant de trouver l'origine de la fuite. Nous allons activer la vue étendue ④ en cliquant sur View > Extended Detail. Cette nouvelle vue permet de représenter la pile des instructions menant à la fuite. Dans cette pile sont représentées toutes les méthodes, même celles qui ne vous appartiennent pas. Les vôtres sont précédées de l'icône avec le haut d'un personnage. Nous remarquons donc en ⑤ que la fuite provient de notre application delegate. Double-cliquez sur cette ligne pour afficher la portion de code incriminée comme à la Figure 32.7. Il ne reste plus qu'à corriger la fuite !



```
#pragma mark -
// Override point for customization after application launch.
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    UIWindow *window = [[UIWindow alloc] initWithFrame:CGRectMake(28, 28, 381)];
    [window addSubview:label];
    NSNumber *numberValue = [NSNumber alloc];
    numberValue.value = 123.456789;
    label.text = [NSString stringWithFormat:@"%.2f", numberValue];
    int i;
    if (i == 3)
        NSLog(@"%@", label);
    [window makeKeyAndVisible];
    return YES;
}
```

Figure 32.7 : La ligne qui fuit.

ACTIVER LES ZOMBIES

Lorsque vous déboguez votre application, il se peut que vous libérez trop un objet. Dans ce cas, les erreurs dans la console peuvent vite devenir incompréhensibles. Pour y remédier, il faut activer ce que l'on appelle les zombies. En réalisant ceci, les variables qui seront détruites seront changées en _NSZombie et la zone mémoire ne sera jamais marquée comme libre. Ainsi, toute erreur de gestion mémoire amenant normalement à un plantage en règle de votre application sera plus explicite dans la console.

Pour activer ce mode, double-cliquez sur l'exécutable comme à la Figure 32.8. Ensuite, rendez-vous dans l'onglet Arguments puis ajoutez une variable dans Variables to be set in the environment que vous nommerez NSZombieEnabled avec comme valeur YES comme à la Figure 32.9.

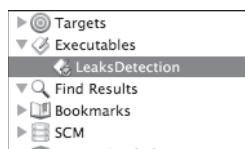


Figure 32.8 : Trouvez l'exécutable.

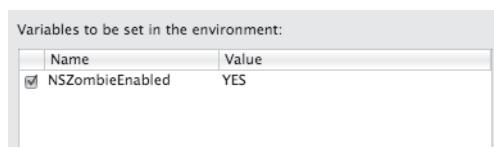


Figure 32.9 : Activez les zombies.

Le résultat est le suivant, avec un double release de number :

Avant :

```
LeaksDetection(5032,0xa00e14e0) malloc: *** error for object 0x5d0a950: double free
*** set a breakpoint in malloc_error_break to debug
```

Après :

```
LeaksDetection[5038:207] *** -[CFNumber release]: message sent to deallocated
instance 0x591edb0
```

Attention cependant, l'utilisation de ce mode doit rester ponctuelle ! Ne laissez pas ce mode dans la distribution de votre application, les utilisateurs ne voudraient pas voir arriver des zombies partout...

SQLite est un système de gestion de base de données libre et multiplate-forme, particulièrement adapté aux mobiles. Il est, par exemple, aussi bien disponible sur iPhone que sur Android. C'est un avantage très intéressant en cas de portage à réaliser. De plus, ce système a été retenu par Apple dans l'implémentation de Core Data, l'autre solution de stockage sur iPhone.

Dans cette fiche, nous allons nous servir d'une base de données SQLite pour réaliser un système de gestion de scores.

CRÉATION DE LA BASE DE DONNÉES

Nous allons créer un fichier database.sql, qui va contenir la base de données de notre projet. Dans notre exemple, nous créerons le fichier, puis nous y insérerons des données initiales, afin de voir qu'il est tout à fait possible de partir d'un fichier contenant une base complète (extraite d'un site Internet par exemple).

C'est à l'aide du Terminal (Spotlight > Terminal) que nous allons créer cette base. Positionnez-vous ensuite dans un dossier (pour moi c'est le bureau), (cd Desktop) et lançons les commandes nécessaires :

```
#création du fichier
sqlite3 database.sql

-- création de la Base de Données (BDD)
CREATE TABLE score ( id INTEGER PRIMARY KEY, pseudo VARCHAR(50), resultat INT);
INSERT INTO score (pseudo, resultat) VALUES ('iPodishima', 5);
INSERT INTO score (pseudo, resultat) VALUES ('Heyfeel', 4);
INSERT INTO score (pseudo, resultat) VALUES ('Aurel', 3);
INSERT INTO score (pseudo, resultat) VALUES ('Bidou', 2);

-- on quitte sqlite3
.quit
```

Il restera à importer le fichier database.sql au projet que nous allons créer.

CRÉATION DU PROJET iPHONE ET DE L'INTERFACE GRAPHIQUE

Nous allons partir d'une Window-based Application que vous pouvez nommer SQLite.

Vous pouvez ajouter tout de suite le fichier database.sql à votre projet dans le dossier ressources, copiez-le, et nous allons commencer l'interface graphique :

- Ajoutez tout d'abord une classe RootViewController héritant de UIViewController à notre projet.
- Ajoutez également une classe ListeViewController héritant elle, de UITableViewController.

Nous allons maintenant implémenter une première partie de l'interface graphique, en affichant dans notre ListeViewController une liste d'éléments. Je ne rentre pas dans les détails, mais j'utilise un

RootViewController pour modifier plus facilement la hiérarchie des vues en cas de nécessité. Cela ne devrait plus vous poser de problèmes arrivé ici.

```
SQLiteAppDelegate.h
#import <UIKit/UIKit.h>
#import "RootViewController.h"

@interface SQLiteAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    RootViewController *rootViewController;
}

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

@end

SQLiteAppDelegate.m
#import "SQLiteAppDelegate.h"
@implementation SQLiteAppDelegate
@synthesize window;

#pragma mark -
#pragma mark Application lifecycle

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    rootViewController = [[RootViewController alloc] init];
    [window addSubview:rootViewController.view];
    [window makeKeyAndVisible];
    return YES;
}

- (void)dealloc {
    [window release];
    [rootViewController release];
    [super dealloc];
}

RootViewController.h
#import <UIKit/UIKit.h>
@interface RootViewController : UIViewController {
    UINavigationController *navController;
}
@end
```

```
RootViewController.m
#import "RootViewController.h"
#import "ListeViewController.h"

@implementation RootViewController

- (void)loadView {
    ListeViewController *listeViewController =
    ▶ [[ListeViewController alloc] initWithStyle:UITableViewStylePlain];
    navController = [[UINavigationController alloc]
    ▶ initWithRootViewController:listViewController];
    [listeViewController release];
    navController.navigationBar.barStyle = UIBarStyleBlackTranslucent;
    self.view = navController.view;
}

- (void)dealloc {
    [navController release];
    [super dealloc];
}
@end
```

Vérifions que tout fonctionne avant d'aller plus loin. Pour cela, dans `ListeTableViewController`, procédez aux changements suivants :

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section {
    return 5 ;
}

- (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.text = @"élément";
    return cell;
}
```

Normalement, en compilant et en lançant votre application, vous devriez obtenir une UITableView affichant cinq éléments. Si ce n'est pas le cas, référez-vous à la Fiche 12 sur les UITableView.

IMPLÉMENTATION DU SQLMANAGER

Nous allons maintenant implémenter une classe Score héritant de NSObject qui va gérer les scores des joueurs :

```
Score.h
#import <Foundation/Foundation.h>

@interface Score : NSObject {
    NSInteger primaryKey;
    NSString *pseudo;
    NSInteger resultat;
}

@property (assign, nonatomic, readonly) NSInteger primaryKey;
@property (nonatomic, retain) NSString *pseudo;
@property (assign, nonatomic, readonly) NSInteger resultat;

-(id)initWithKey:(NSInteger)k pseudo:(NSString *)n resultat:(NSInteger )s;

@end

Score.m
#import "Score.h"
@implementation Score
@synthesize pseudo, resultat, primaryKey;

-(id)initWithKey:(NSInteger)k pseudo:(NSString *)p resultat:(NSInteger )r{
    primaryKey = k;
    self.pseudo = p;
    resultat = r;
    return self;
}

@end
```

Passons à l'implémentation de la classe SQLManager héritant également de NSObject. Cette classe sera le cœur de cette fiche et vous sera utile dans vos futurs développements si vous décidez d'opter pour SQLite. Dans un premier temps, nous allons simplement implémenter l'initialisation de la base de données et la lecture des scores. Nous verrons plus tard comment réaliser une insertion et une suppression dans notre base.

Tout d'abord, il faut ajouter la librairie `libssqlite3.0.dylib` à votre projet afin d'exploiter les commandes SQLite dans votre projet.

```
SQLManager.h
#import <Foundation/Foundation.h>
#import <sqlite3.h>      // Import du framework sqlite
#import "Score.h"        // Import de la classe Score

@interface SQLManager : NSObject {
    // Variables de la Base de Données
    NSString *databaseName;
    NSString *databasePath;
    // Tableau de Scores
    NSMutableArray *scores;
}

@property (nonatomic, retain) NSMutableArray *scores;
-(id) initDatabase;
-(void) readScoresFromDatabase;
@end

SQLManager.m
#import "SQLManager.h"

@implementation SQLManager
@synthesize scores;

- (id) initDatabase {
    if(self = [super init]) { ①
        //On définit le nom de la base de données
        databaseName = @"database.sql";

        // On récupère le chemin
        NSArray *documentPaths =
        ➔ NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
        ➔ NSUserDomainMask, YES);
        NSString *documentsDir = [documentPaths objectAtIndex:0];
        databasePath = [documentsDir
        ➔ stringByAppendingPathComponent:databaseName];

        // On vérifie si la BDD a déjà été sauvegardée dans l'iPhone de
        ➔ l'utilisateur
        BOOL success;

        // Crée un objet FileManagerCreate qui va servir à vérifier le statut
        // de la base de données et de la copier si nécessaire
    }
}
```

```

NSFileManager *fileManager = [NSFileManager defaultManager];

// Vérifie si la BDD a déjà été créée dans les fichiers system de l'utilisateur
success = [fileManager fileExistsAtPath:databasePath];

// Si la BDD existe déjà on ne fait pas la suite
if(!success){
    // Si ce n'est pas le cas alors on copie la BDD de l'application
    // vers les fichiers système de l'utilisateur

    // On récupère le chemin vers la BDD dans l'application
    NSString *databasePathFromApp = [[[NSBundle mainBundle]
    // resourcePath] stringByAppendingPathComponent:databaseName];

    // On copie la BDD de l'application vers le fichier système de
    // l'application
    [fileManager copyItemAtPath:databasePathFromApp toPath:databasePath
    // error:nil];

    [fileManager release];
}

[databasePath retain];

}

return self;
}

- (void) dealloc {
    [databasePath release];
    [databaseName release];
    [scores release];
    [super dealloc];
}
@end

```

- ① La méthode initDatabase permet de récupérer la base de données à partir du fichier que nous avons créé précédemment. Cet appel initialise également databasePath qui est le chemin de la base qui va nous permettre d'utiliser la base lorsque l'on en aura besoin.

```

-(void) readScoresFromDatabase {
    // Déclaration de l'objet database
    sqlite3 *database;

    // Initialisation du tableau de score ②
    if(scores == nil){
        scores = [[NSMutableArray alloc] init];
    }
}

```

```
else {
    [scores removeAllObjects];
}

// On ouvre la BDD à partir des fichiers système
if(sqlite3_open([databasePath UTF8String], &database) == SQLITE_OK) (3){
    // Préparation de la requête SQL qui va permettre de récupérer les
    ➔ objets score de la BDD
    //en triant les scores dans l'ordre décroissant
    const char *sqlStatement = "select * from score ORDER BY resultat DESC";

    //création d'un objet permettant de connaître le status de l'exécution de
    ➔ la requête
    sqlite3_stmt *compiledStatement;

    if(sqlite3_prepare_v2(database, sqlStatement, -1, &compiledStatement,
    ➔ NULL) == SQLITE_OK) (4){
        // On boucle tant que l'on trouve des objets dans la BDD
        while(sqlite3_step(compiledStatement) == SQLITE_ROW) (5){
            // On lit les données stockées dans le fichier sql
            // Dans la première colonne on trouve du texte que l'on place
            ➔ dans un NSString
            NSInteger key = sqlite3_column_int(compiledStatement, 0);
            NSString *pseudo = [NSString stringWithFormat:(char
            ➔ *)sqlite3_column_text(compiledStatement, 1)];
            // Dans la deuxième colonne on récupère le score dans un NSInteger
            NSInteger resultat = sqlite3_column_int(compiledStatement, 2);

            // On crée un objet Score avec les paramètres récupérés dans la BDD
            Score *score = [[Score alloc] initWithKey:key pseudo:pseudo
            ➔ resultat:resultat]; (6)

            // On ajoute le score au tableau
            if(![scores containsObject:score])
                [scores addObject:score];

            [score release];
        }
    }
    // On libère le compiledStamentent de la mémoire
    sqlite3_finalize(compiledStatement);
}

//On ferme la bdd
sqlite3_close(database);
}
```

La méthode `readScoreFromDatabase` permet la lecture de notre base de données SQLite et crée un `NSMutableArray` contenant la liste des Scores. Ceci se passe en trois étapes :

- ② Initialisation du tableau scores ou réinitialisation s'il existe déjà.
 - ③ On ouvre la base de données à partir de `databasePath`, ④ on crée la requête SQL et ⑤ on parcourt les résultats.
 - ⑥ Pour chaque résultat, on crée un objet score que l'on ajoute au tableau scores s'il n'y est pas déjà.
- Modifions maintenant `ListeTableViewController` pour afficher les scores présents dans le tableau `scores` (les modifications à apporter sont en gras).

```
ListeTableViewController.h
#import <UIKit/UIKit.h>
#import "SQLManager.h"

@interface ListeViewController : UITableViewController {
    SQLManager *manager;
}

@end

ListeTableViewController.m
- (id)initWithStyle:(UITableViewStyle)style {
    if ((self = [super initWithStyle:style])) {
        self.title = @"Liste des Scores";
        manager = [[SQLManager alloc] initDatabase];
        [manager readScoresFromDatabase];
    }
    return self;
}

- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section {
    return [manager.scores count];
}

- (UITableViewCell *)tableView:(UITableView *
    *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell =
        [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue1
            reuseIdentifier:CellIdentifier] autorelease]; ①
    }
}
```

```

Score *score = [manager.scores objectAtIndex:indexPath.row];
cell.textLabel.text = score.pseudo;
cell.detailTextLabel.text = [NSString stringWithFormat:@"%d",score.resultat]; ②
cell.selectionStyle = UITableViewCellStyleNone;
return cell;
}
- (void)dealloc {
[manager release];
[super dealloc];
}
}

```

① J'ai recours ici à un modèle particulier de cellule fourni dans le SDK qui permet l'utilisation d'un detailTextLabel ② afin de disposer le pseudo à gauche et le score en aligné à droite.

Si tout va bien à ce niveau, vous devriez voir s'afficher la liste initiale des scores de votre base de données. (Figure 33.1)

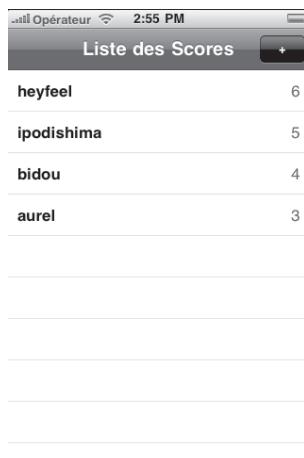


Figure 33.1 : La liste des scores.

AJOUT D'UN SCORE DANS LA DATABASE.SQL

Nous allons modifier le manager et notre interface graphique afin d'ajouter un score à notre base de données. Pour cela, il faut implémenter une méthode permettant de faire une insertion dans la base SQLite

```

SQLManager.m
- (void)insertIntoDatabase:(Score*)newScore {
    // Déclaration de l'objet database
    sqlite3 *database;
    // On ouvre la BDD à partir des fichiers système
    if(sqlite3_open([databasePath UTF8String], &database) == SQLITE_OK) {

```

```

// Préparation de la requête SQL qui va permettre d'ajouter un score à la BDD
NSString *sqlStat = [NSString stringWithFormat:@"INSERT INTO score (pseudo,
➥ resultat) VALUES ('%@', %d);",newScore.pseudo, newScore.resultat];
//conversion en char *
const char *sqlStatement = [sqlStat UTF8String];
//On utilise sqlite3_exec qui permet très simplement d'exécuter une
➥ requête sur la BDD
sqlite3_exec(database, sqlStatement,NULL,NULL,NULL); ①
[self readScoresFromDatabase]; ②
}
sqlite3_close(database);
}

```

① On utilise sqlite_exec pour exécuter directement une requête dans notre base.

② Après avoir fait l'insertion, on appelle readScoreFromDatabase afin de remettre à jour le tableau score.

N'oubliez pas d'écrire dans le .h :

```
- (void)insertIntoDatabase:(Score*)newScore;
```

On pourrait également ajouter directement newScore au tableau scores, mais il faudrait ensuite trier le tableau avant de rafraîchir la UITableView.

Passons aux modifications de l'interface graphique en créant tout d'abord une nouvelle classe AjoutViewController héritant de UIViewController.

```

AjoutViewController.h
#import <UIKit/UIKit.h>
#import "SQLManager.h"

@interface AjoutViewController : UIViewController {
    UITextField *pseudo;
    UITextField *score;
    SQLManager *manager;
}
@property(nonatomic, retain) SQLManager *manager;
@end

AjoutViewController.m
#import "AjoutViewController.h"
#import "Score.h"

@implementation AjoutViewController

@synthesize manager;

- (void)loadView {
    self.view = [[[UIView alloc] init] autorelease]; ①
}

```

```
//Création d'un UITextField pour entrer le pseudo
pseudo = [[UITextField alloc] initWithFrame:CGRectMake(100, 60, 120, 20)];
pseudo.backgroundColor = [UIColor colorWithRed:0.9 green:0.9 blue:0.9 alpha:1.0];
pseudo.textAlignment = NSTextAlignmentCenter;
//on fait apparaître tout de suite le clavier
[pseudo becomeFirstResponder];
pseudo.placeholder = @"Pseudo";
[self.view addSubview:pseudo];
[pseudo release];

//Création d'un UITextField pour le score
score = [[UITextField alloc] initWithFrame:CGRectMake(100, 100, 120, 20)];
score.backgroundColor = [UIColor colorWithRed:0.9 green:0.9 blue:0.9 alpha:1.0];
score.textAlignment = NSTextAlignmentCenter;
//On utilise un clavier numérique
score.keyboardType = UIKeyboardTypeNumberPad;
score.placeholder = @"Résultat";
[self.view addSubview:score];
[score release];

//Création d'un bouton pour réaliser l'ajout
UIButton *bouton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[bouton setTitle:@"ajouter" forState:UIControlStateNormal];
[bouton addTarget:self action:@selector(valider)
    forControlEvents:UIControlEventTouchUpInside];
bouton.frame = CGRectMake(100, 140, 120, 30);
[self.view addSubview:bouton];
}

//méthode de validation
-(void)valider{
    //Création d'un object Score à partir des informations entrées
    Score *newScore = [[Score alloc] initWithKey:-1 pseudo:pseudo.text
        resultat:[score.text intValue]];
    //insertion du score
    [manager insertIntoDatabase:newScore];
    [newScore release];
    [[self.navigationController.viewControllers objectAtIndex:0] viewWillAppear:YES]; ②
    [self.navigationController popViewControllerAnimated:YES];
}

- (void)dealloc {
    [manager release];
    [super dealloc];
}
@end
```

- ① Il faut bien penser à initialiser une vue à notre contrôleur car ici, il n'y a pas de XIB associé.
- ② Je préviens l'instance de `ListeViewController` que sa vue va bientôt apparaître, afin que la méthode `viewWillAppear` soit appelée.

Il faut maintenant modifier `ListeTableViewController` de la manière suivante :

```
#import "ListeViewController.h"
#import "AjoutViewController.h"

-(id)initWithStyle:(UITableViewStyle)style {
    if ((self = [super initWithStyle:style])) {
        self.title = @"Liste des Scores";
        //création d'un bouton pour l'ajout
        UIBarButtonItem *add = [[UIBarButtonItem alloc]
        ↪ initWithTitle:@"Ajouter" style:UIBarButtonItemStylePlain target:self
        ↪ action:@selector(ajoutScore)];
        self.navigationItem.rightBarButtonItem = add;
        [add release];

        manager = [[SQLManager alloc] init];
        [manager readScoresFromDatabase];
    }
    return self;
}

-(void)ajoutScore{
    ajoutViewController *ajoutViewController = [[AjoutViewController alloc] init];
    ajoutViewController.manager = manager;
    [self.navigationController pushViewController:ajoutViewController animated:YES];
    [ajoutViewController release];
}

-(void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    //on recharge le Liste au retour après ajout
    [self.tableView reloadData];
}
```

SUPPRESSION D'UN SCORE

Ajoutons une méthode de suppression à notre manager. C'est le même principe que pour l'ajout, à l'exception de la requête SQL.

```
- (void)deleteFromDatabase:(Score*)oldScore {

    // Déclaration de l'objet database
    sqlite3 *database;
```

```
// On ouvre la BDD à partir des fichiers système
if(sqlite3_open([databasePath UTF8String], &database) == SQLITE_OK) {
    // Préparation de la requête SQL qui va permettre de supprimer un score
    // à la BDD
    NSString *sqlStat = [NSString stringWithFormat:@"DELETE FROM score WHERE
    // id = %d;",oldScore.primaryKey];
    //conversion en char *
    const char *sqlStatement = [sqlStat UTF8String];
    //On utilise sqlite3_exec qui permet très simplement d'exécuter une
    // requête sur la BDD
    sqlite3_exec(database, sqlStatement,NULL,NULL,NULL);
    [self readScoresFromDatabase];
}
sqlite3_close(database);
}
```

N'oubliez pas de déclarer dans le .h :

```
- (void)deleteFromDatabase:(Score*)oldScore;
```

Comme pour la méthode d'ajout, il faut penser à appeler readScoreFromDatabase pour mettre à jour le tableau.

Passons à l'interface graphique, dans ListViewController.m :

```
- (void)tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
    forRowAtIndexPath:(NSIndexPath *)indexPath {
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        [manager deleteFromDatabase:[manager.scores objectAtIndex:indexPath.row]]; ①
        [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
        // withRowAnimation:YES]; ②
    }
}
```

① ② : Il est important de réaliser ces deux opérations dans cet ordre précis, sinon votre application quittera inopinément.

Lorsque vous utiliserez des webservices, vous aurez très certainement besoin de **parser** des fichiers, qu'ils soient **XML ou JSON**. Cette fiche vous apprend à le faire.

UTILISER XML

Parsons un fichier XML contenant des pays. Créez un nouveau projet Window-based Application que vous nommerez XML. Écrivez ces quelques lignes dans un fichier .xml (capitales.xml) et ajoutez-le au projet (rappel : glisser-déposer le fichier directement dans Xcode).

```
<?xml version="1.0" encoding="UTF-8"?>
<Monde>
    <pays>
        <nom>France</nom>
        <capitale>Paris</capitale>
        <habitants>60M</habitants>
    </pays>
    <pays>
        <nom>Suisse</nom>
        <capitale>Berne</capitale>
        <habitants>7,5M</habitants>
    </pays>
    <pays>
        <nom>Allemagne</nom>
        <capitale>Berlin</capitale>
        <habitants>82M</habitants>
    </pays>
</Monde>
```

Il existe un parser XML intégré directement dans le framework Foundation : NSXMLParser. Nous allons donc nous en servir. Pour cela, créez une nouvelle classe, subclass of NSObject que vous nommerez XMLParser.

Implémentons cette classe en commençant par la méthode qui lancera le parsage :

```
- (void)parseXMLFileAtPath:(NSString *)path {
    countries = [[NSMutableArray alloc] init]; ①

    NSURL *xmlURL = [NSURL fileURLWithPath:path];

    textParser = [[NSXMLParser alloc] initWithContentsOfURL:xmlURL];
    [textParser setDelegate:self]; ②

    [textParser parse]; ③
}
```

Chaque pays présent dans le fichier XML va être ajouté dans un tableau : countries, que l'on initialise ligne ①.

Ligne ②, la classe NSXMLParser propose un protocole delegate, et on choisit de l'implémenter. C'est obligatoire pour que le parser soit opérationnel. Enfin, on lance le parseage ligne ③.

Regardons maintenant le .h où l'on déclare tous les objets nécessaires pour construire le tableau des pays.

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

#define kPays @"pays"
#define kCapitale @"capitale"
#define kNom @"nom"
#define kPopulation @"habitants"

@class XMLTableViewController;

@interface XMLParser : NSObject <NSXMLParserDelegate> {

    NSXMLParser *textParser;
    NSMutableArray *countries;

    // un objet temporaire (ici un pays).
    // il est ajouté au tableau "countries" puis effacé pour le nouveau pays
    NSMutableDictionary *item;

    // On va parser le document de haut en bas.
    // On va donc collecter chaque sous-éléments, les sauver dans item, puis
    // ajouter dans le tableau

    NSString *currentElement;
    NSMutableString *currentCountry;
    NSMutableString *currentPopulation;
    NSMutableString *currentCapital;

    XMLTableViewController *tableViewController;
}

- (void)parseXMLFileAtPath:(NSString *)path;

@property (nonatomic, retain) NSMutableArray *countries;
@property (nonatomic, retain) XMLTableViewController *tableViewController;
@end
```

N'oubliez pas ces lignes dans le .m :

```
#import "XMLParser"
#import "XMLTableViewController"

@implementation XMLParser
@synthesize countries;
@synthesize tableViewController;
```

Ensuite, implémentons les méthodes du delegate. Commençons par celle appelée lorsque le parser rencontre un nouvel élément :

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
    ↪ namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
    ↪ attributes:(NSDictionary *)attributeDict{
    if(currentElement)
    {
        [currentElement release];
        currentElement = nil;
    }
    currentElement = [elementName copy]; ①

    if ([elementName isEqualToString:kPays]) { ②
        //On efface notre cache qui est item
        if(item)
        {
            [item release];
            item = nil;
        }
        if(currentCountry)
        {
            [currentCountry release];
            currentCountry = nil;
        }
        if(currentCapital)
        {
            [currentCapital release];
            currentCapital = nil;
        }
        if(currentPopulation)
        {
            [currentPopulation release];
            currentPopulation = nil;
        }
        item = [[NSMutableDictionary alloc] init];
        currentPopulation = [[NSMutableString alloc] init];
        currentCapital = [[NSMutableString alloc] init];
        currentCountry = [[NSMutableString alloc] init];
    }
}
```

Dans cette méthode, on met à jour l'élément courant rencontré ligne **①**. Ensuite, si cet élément (la balise rencontrée) est égale à Pays ligne **②**, on sait que l'on rencontre un nouveau pays. On remet donc à zéro tous les objets qui servent à construire item (un dictionnaire contenant les données du pays).

Après avoir détecté la balise Pays, il faut “mettre en cache” les valeurs des balises Nom, Capitale et Population. Pour cela, implémentez la méthode suivante :

```
- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string{
    // on sauve les éléments du pays pour l'item en cours
    if ([currentElement isEqualToString:kCapitale])
        [currentCapital appendString:string];
    else if ([currentElement isEqualToString:kNom])
        [currentCountry appendString:string];
    else if ([currentElement isEqualToString:kPopulation])
        [currentPopulation appendString:string];
}
```

Ensuite, définissons la méthode appelée lorsque l'on rencontre une balise de fermeture (exemple : </balise>). Dans cette méthode, lorsque l'on ferme pays, on sauve les valeurs des balises nom, capitale et population pour le pays en question, puis on ajoute ce pays dans le tableau des pays.

```
- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName{
    if ([elementName isEqualToString:kPays]) {
        // on sauve les valeurs dans item, qui est ensuite ajouté dans le
        // tableau countries
        [item setObject:currentCountry forKey:kNom];
        [item setObject:currentCapital forKey:kCapitale];
        [item setObject:currentPopulation forKey:kPopulation];

        [countries addObject:item];
    }
}
```

Enfin, lorsque le parser a fini de parcourir le document, on choisit de recharger la table view. En effet, nous allons présenter les données parsées grâce à une table view, comme nous l'avons vu à la Fiche 13. Pour cela, on appelle la méthode reloadData de la table view du table view controller tableViewController, propriété du parser. Cette méthode est appelée sur le thread principal, car nous allons par la suite parser dans un thread détaché pour l'occasion.

```
- (void)parserDidEndDocument:(NSXMLParser *)parser {
    NSLog(@"C'est fini !");
    NSLog(@"countries a %d pays", [countries count]);

    [tableViewController.tableView performSelectorOnMainThread:@selector(reloadData)
        // withObject:nil waitUntilDone:NO];
}
```

Et le dealloc ?

```
- (void) dealloc {  
    [countries release];  
    [textParser release];  
    [item release];  
    [currentCapital release];  
    [currentCountry release];  
    [currentPopulation release];  
    [super dealloc];  
}
```

Créons maintenant le table view controller qui gérera l'affichage des données du fichier XML. Pour cela, ajoutez un nouveau fichier comme à la Figure 34.1 que vous nommerez XMLTableViewController. Dans l'application delegate, ajoutez ces lignes en gras :

```
- (BOOL) application:(UIApplication *)application  
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    xmlTableViewController = [[XMLTableViewController alloc]  
    initWithStyle:UITableViewStylePlain];  
    [window addSubview:xmlTableViewController.view];  
    [window makeKeyAndVisible];  
  
    return YES;  
}
```

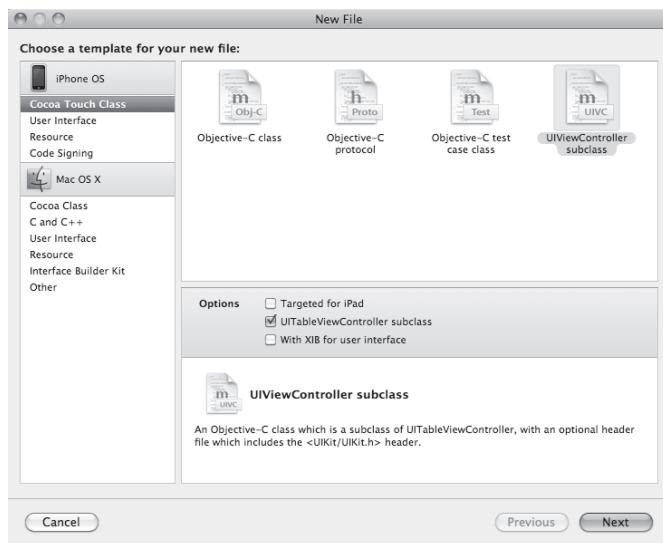


Figure 34.1 : Ajoutez un table view controller.

Ensuite, modifiez ces méthodes (lignes en gras) dans le table view controller fraîchement créé :

```
- (void)viewDidLoad {
    [super viewDidLoad];
    self.clearsSelectionOnViewWillAppear = NO;
    NSString *xmlFilePath = [[NSBundle mainBundle] pathForResource:@"capitales"
    ↪ ofType:@"xml"];
    [self performSelectorInBackground:@selector(parseXMLFile:) withObject:xmlFilePath]; ①
}

- (void)parseXMLFile:(NSString*)thePath {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init]; ②
    parser = [[XMLParser alloc] init];
    parser.tableViewController = self; ③
    if ([parser.countries count] == 0)
    {
        [parser parseXMLFileAtPath:thePath];
    }
    [pool release];
}
```

Ligne ①, on choisit donc de parser le document dans un thread séparé. Ainsi, le thread principal n'est pas dédié au parсage, et reste libre pour intercepter et réagir aux événements utilisateurs, pour afficher des éléments à l'écran. En détachant un thread, on doit créer un bassin d'autorelease, ce que l'on fait ligne ②. En effet, le bassin d'autorelease créé dans le fichier main.m n'est disponible que pour le thread principal (voir Fiche 32). Pour permettre au parser d'appeler le rafraîchissement de l'affichage des données, on set la propriété tableViewController ligne ③.

Enfin, pour afficher les données :

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    // Retourne le nombre de sections
    return 1;
}
- (NSInteger)tableView:(UITableView *)tableView
    ↪ numberOfRowsInSection:(NSInteger)section {
    // Retourne le nombre de lignes dans la section
    return [parser.countries 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];
}
NSDictionary *pays = [parser.countries objectAtIndex:[indexPath row]];
cell.textLabel.text = [NSString stringWithFormat:@"%@ — %@ — %@", 
    [pays objectForKey:kNom], [pays objectForKey:kCapitale],
    [pays objectForKey:kPopulation]];
return cell;
}

N'oubliez pas le dealloc !
- (void)dealloc {
    [parser release];
    [super dealloc];
}

```

Pour information, voici le .h :

```

#import <UIKit/UIKit.h>
#import "XMLParser.h"

@interface XMLTableViewController : 
    UITableViewController {
    XMLParser *parser;
}
@end

```

Compilez et lancez votre application, vous devriez obtenir une liste comme à la Figure 34.2.



Figure 34.2 : La liste des pays.

Info

Pour parser un fichier XML directement depuis une source Internet, il suffit de créer un objet NSURL du fichier XML et le passer directement en paramètre de la méthode :

```
textParser = [[NSXMLParser alloc] initWithContentsOfURL:xmlURL];
```

UTILISER LE JSON

L'iPhone et l'iPad sont des appareils mobiles et connectés qui permettent un accès facile à Internet. Bien que l'on puisse utiliser le navigateur intégré, grâce à une UIWebView, il est bien souvent plus confortable pour l'utilisateur de lancer une application spécifique à l'iPhone, dite optimisée, pour accéder aux informations qui l'intéressent, comme sa page Facebook ou son fil Twitter.

Cet accès est possible grâce aux webservices qui sont en quelque sorte des sites web spécifiques, sans mise en page, qui nous permettent d'obtenir les informations nécessaires. Beaucoup de sites proposent des API qui utilisent le JSON comme système de fichier en plus du traditionnel XML.

Voici quelques exemples :

- **Twitter.** <http://apiwiki.twitter.com/Twitter-Search-API-Method%3A-trends-current>
- **Facebook.** <http://wiki.developers.facebook.com/index.php/Comments.get>
- **Météo.** <http://www.geonames.org/export/JSON-webservices.html>

POURQUOI LE JSON ?

La question n'est pas anodine. Le JSON a séduit les développeurs par sa simplicité et sa rapidité. Même si ce n'est pas une technologie mise en avant par Apple, qui utilise du XML pour les plist par exemple, le JSON a l'avantage d'être un format plus compact et donc plus rapide lors de son téléchargement sur un webservice.

Comparaison sur un tableau de score par exemple :

XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<Resultats>
    <Joueur>
        <Nom>Jean Martin</Nom>
        <Score>10000</Score>
    </Joueur>
    <Joueur>
        <Nom>Pierre Dupond</Nom>
        <Score>9000</Score>
    </Joueur>
    <Joueur>
        <Nom>Alice Bateau</Nom>
        <Score>8500</Score>
    </Joueur>
</Resultats>
```

Longueur : 271 caractères.

```
JSON :  
  
{  
    "resultats":{  
        "joueurs":[  
            {  
                "nom":"Jean Martin",  
                "score":10000  
            },  
            {  
                "nom":"Pierre Dupond",  
                "score":9000  
            },  
            {  
                "nom":"Alice Bateau",  
                "score":8500  
            }  
        ]  
    }  
}
```

Longueur : 194 caractères.

Que peut-on conclure de cette comparaison ? Le XML, de par sa taille, va être environ 50 % plus lent au téléchargement pour la même information, ce qui est loin d'être négligeable sur un objet mobile.

UTILISATION SUR IPHONE

La méthode présentée n'est pas LA méthode mais elle a le mérite de fonctionner parfaitement. Pour corser le tout, nous allons faire une application compatible **iPhone et iPad (Universelle)**.

Lancer Xcode et faire : File > New Project > Window-based Application et choisir Universal dans Product. Appelons ce projet JSON_Tuto.

Info

Nous allons devoir utiliser un framework qui ne fait pas parti du SDK officiel afin de traiter les fichiers en JSON.

Rappelez-vous, je vous ai dit qu'Apple fonctionne avec XML pour le moment !

Il existe plusieurs frameworks pour traiter le JSON sur iPhone, notre préférence va à celui de disponible à cette adresse : <http://code.google.com/p/json-framework/>. Il a l'avantage d'avoir fait ses preuves dans beaucoup d'applications.

Téléchargez la dernière version, vous devriez arriver sur la fenêtre représentée à la Figure 34.3.

Pour vous en servir, le plus simple est de glisser directement le dossier JSON dans votre projet Xcode (et le copier).

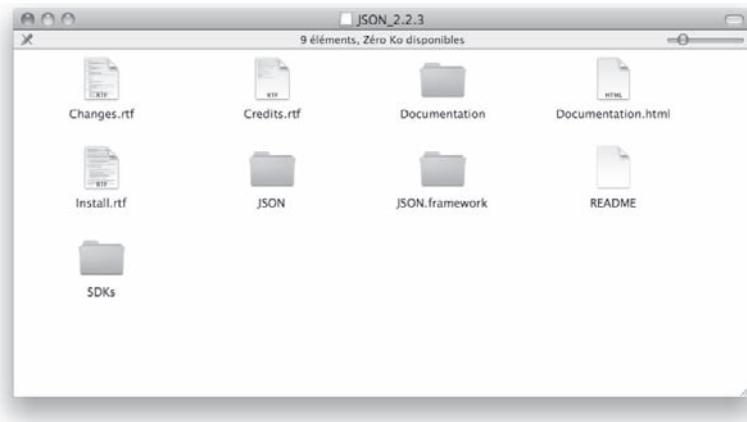


Figure 34.3 :
Le framework
JSON dans
le Finder.

Vous devriez obtenir la hiérarchie présentée à la Figure 34.4.



Figure 34.4 : Le framework JSON
dans votre application.

Nous allons maintenant intégrer un fichier en JSON dans notre projet afin de pouvoir l'exploiter.

1. Sur JSON_Tuto faire **Ctrl**+Clic > Add > New File, pour ajouter un nouveau fichier où nous écrirons le JSON.
2. Choisir ensuite other dans la colonne de gauche puis Empty File.
3. Vous pouvez l'appeler JSON.txt par exemple et ensuite y intégrer les informations que nous avons utilisées dans notre comparaison entre le JSON et le XML.

Vous devriez obtenir quelque chose de comparable à la Figure 34.5.

Nous allons maintenant afficher ces données dans une UITableView. Pour ce faire : **Ctrl**+Clic > Add > New File. Choisir ensuite Cocoa Touch Class > UIViewController et ne cocher que UITableViewController subclass comme à la Figure 34.6.

Info

Il ne faut pas cocher la case de création du fichier XIB, car cela nous obligera à en faire deux différents : un pour l'iPad et un pour l'iPhone.

The screenshot shows the Xcode interface. On the left is the project navigator with 'JSON_Tuto' selected, showing files like 'JSON.txt', 'AppDelegate_Pad.', 'AppDelegate_Phor.', 'MainWindow_Pad.', 'AppDelegate_Phor.', 'MainWindow_Phor.', 'Shared', 'Other Sources', 'Frameworks', 'Products', 'Targets', and 'Executables'. The main editor window displays the contents of 'JSON.txt' which is a JSON array:

```
{
    "resultats": [
        {
            "joueur": [
                {
                    "nom": "Jean Martin",
                    "score": 10000
                },
                {
                    "nom": "Pierre Dupond",
                    "score": 9000
                },
                {
                    "nom": "Alice Bateau",
                    "score": 8500
                }
            ]
        }
    ]
}
```

Figure 34.5 : Votre premier fichier JSON.



Figure 34.6 : Création d'un Table View Controller.

Nommez ce fichier TableViewController.

Il reste à ajouter cette vue aux deux Appdelegate (un pour l'iPad et l'autre pour l'iPhone) comme vous savez le faire maintenant :

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

@class TableViewController;
@interface AppDelegate_iPhone : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    TableViewController *tableViewController;
}
```

```
@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) UITableViewController *tableViewController;
@end

AppDelegate_iPhone.m
#import "AppDelegate_iPhone.h"
#import "TableViewController.h";

@implementation AppDelegate_iPhone
@synthesize window;
@synthesize tableViewController;

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    TableViewController *viewController = [[TableViewController alloc]
    initWithStyle:UITableViewStylePlain];
    self.tableViewController = viewController;
    [viewController release];
    [window addSubview:tableViewController.view];
    [window makeKeyAndVisible];
    return YES;
}

- (void)dealloc {
    [window release];
    [tableViewController release];
    [super dealloc];
}
@end
```

C'est exactement la même chose dans AppDelegate_iPad.h et AppDelegate_iPad.m :

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

@class TableViewController;
@interface AppDelegate_iPad : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    TableViewController *tableViewController;
}

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

AppDelegate_iPad.m
#import "AppDelegate_iPad.h"
#import "TableViewController.h"
```

```

@implementation AppDelegate_iPad
@synthesize window;
@synthesize tableViewController;

- (BOOL)application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    UITableViewController *viewController = [[UITableViewController alloc]
      initWithStyle:UITableViewStylePlain];
    self.tableViewController = viewController;
    [viewController release];
    [window addSubview:tableViewController.view];
    [window makeKeyAndVisible];
    return YES;
}

- (void)dealloc {
    [window release];
    [tableViewController release];
    [super dealloc];
}
@end

```

Avant d'aller plus loin, vous allez compiler votre projet pour le tester. Dans `TableViewController.m` faites simplement :

```

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    // Retourne le nombre de sections
    return 1;
}
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    // Retourne le nombre de lignes dans la section
    return 1;
}

```

et :

```

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
  interfaceOrientation {
    // Retourner YES pour les orientations désirées
    return YES;
}

```

Cette dernière méthode permet à votre application de fonctionner dans toutes les orientations. Il est vivement recommandé de l'implémenter dans les applications iPad, d'après le guide d'Apple concernant l'Interface Homme Machine.

Vous devriez maintenant obtenir une liste de données vide et qui fonctionne dans toutes les orientations.

REmplir la liste depuis le fichier JSON

Afin de faciliter le travail et de bénéficier de toute la puissance d'un langage objet, nous allons créer un objet Joueur. Pour cela, faites **[Ctrl]+Clic > Add > New File**. Créez un fichier Objective-C class héritant de `NSObject` comme à la Figure 34.7.



Figure 34.7 : Ajout d'un nouveau fichier héritant de `NSObject`.

Appelez cette classe Joueur tout simplement !

Passons maintenant à l'implémentation. Notre classe Joueur va respecter la structure du JSON et va ainsi posséder un nom et un score :

```
Joueur.h
#import <Foundation/Foundation.h>
@interface Joueur : NSObject {
    NSString *nom;
    NSInteger score;
}
@property (nonatomic, retain) NSString *nom;
@property NSInteger score;
@end

Joueur.m
#import "Joueur.h"

@implementation Joueur
@synthesize nom;
@synthesize score;
-(void)dealloc {
    [nom release];
    [super dealloc];
}
@end
```

LE TRAITEMENT DU JSON

C'est la section la plus intéressante. Un UITableViewController nécessite un tableau contenant les données à afficher (voir Fiche 12).

Dans notre cas, nous travaillerons avec une liste d'objets Joueur.

```
TableViewController.h
#import <UIKit/UIKit.h>
@interface TableViewController : UITableViewController {
    NSMutableArray *liste;
}
@end
```

Info

Dans le format JSON, la chose importante est de distinguer les éléments des tableaux.

Voici deux règles qui permettent de s'en sortir facilement :

Règle 1 : le format normal est :

```
{ "clé1" : "élément1", "clé2" : "élément2" }
```

Règle 2 : un élément peut être un tableau d'éléments et un tableau s'écrit :

```
[{ "sous_clé1" : "sous_élément1"} , {"sous_clé2" : "sous_élément2"}]
```

Le réflexe est donc [] donne un tableau.

Dans notre cas, dans la clé résultat, on a un élément avec la clé Joueur qui contient un tableau de joueurs !

Passons au code :

```
#import "Joueur.h"
#import "JSON.h"

@implementation TableViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    liste = [[NSMutableArray alloc] init];

    //récupération du chemin vers le fichier contenant le JSON
    NSString *filePath = [[NSBundle mainBundle] pathForResource:@"JSON" ofType:@"txt"];

    //création d'un string avec le contenu du JSON
    NSString *myJSON = [[NSString alloc] initWithContentsOfFile:filePath
    ↳ encoding:NSUTF8StringEncoding error:NULL];
```

```
//Parsage du JSON à l'aide du framework importé
NSDictionary *json          = [myJSON JSONValue];

//récupération des résultats
NSDictionary *resultats     = [json objectForKey:@"resultats"];

//récupération du tableau de Joueurs
NSArray *listeJoueur       = [resultats objectForKey:@"joueurs"];

//On parcourt la liste de joueurs
for (NSDictionary *dic in listeJoueur) {
    //création d'un objet Joueur
    Joueur *joueur = [[Joueur alloc] init];

    //renseignement du nom
    joueur.nom = [dic objectForKey:@"nom"];

    //renseignement du score
    joueur.score = [[dic objectForKey:@"score"] intValue];

    //ajout à la liste
    [liste addObject:joueur];

    //libération de la mémoire
    [joueur release];
}

//à ne pas oublier après l'allocation effectuée au début
[myJSON release];
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section {
    return [liste 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];
}
Joueur *joueur = [liste objectAtIndex:indexPath.row];

//on affiche le nom et le score
cell.textLabel.text = [NSString stringWithFormat:@"%@ - %d",joueur.nom, joueur.score];

return cell;
}
```

Sans oublier le dealloc bien sûr !

Une application qui ne sauvegarde pas de données n'est pas vraiment une application... Cette fiche aborde la sauvegarde de données dans un fichier plist et l'utilisation du protocole NSCoder.

Imaginez une application contenant des sliders, des switchs, ainsi qu'un champ de texte. Nous souhaitons sauvegarder les valeurs de chaque élément lorsque l'utilisateur appuiera sur le bouton Home. Pour cela, nous allons créer un objet de la classe NSDictionary qui contiendra les couples clés/valeurs de chaque élément affiché à l'écran. Cet objet sera détenu par notre application delegate, et sera créé à partir du fichier plist de sauvegarde.

Mais tout d'abord, qu'est-ce qu'un fichier plist ? C'est un type de fichier couramment utilisé pour sauvegarder des données dans l'univers Mac. Regardez par exemple sur votre ordinateur, dans Bibliothèques > Préférences ! Plist signifie Property Lists. La structure d'un plist est la suivante (vous reconnaîtrez bien entendu le XML) :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
➥ "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Nom</key>
    <string>Albert Dupont</string>
    <key>Telephones</key>
    <array>
        <string>0354231287</string>
        <string>0964289302</string>
    </array>
</dict>
</plist>
```

Comme on le constate, le plist contient ici un dictionnaire (balise dict) composé des couples clés/valeurs.

Voici le code pour lire des données :

```
- (BOOL) readDataFromFile {
    NSString *errorDesc = nil;
    NSPropertyListFormat format;
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
➥ NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];

    NSString *plistPath = [documentsDirectory
➥ stringByAppendingPathComponent:@"preferences.plist"];
    NSData *plistXML = [[NSFileManager defaultManager] contentsAtPath:plistPath];
```

```

NSDictionary *dictionary = (NSDictionary *)[NSPropertyListSerialization
➥ propertyListFromData:plistXML
➥ mutabilityOption:NSPropertyListMutableContainersAndLeaves format:&format
➥ errorDescription:&errorDesc];
if (!dictionary) {
    // Il y a eu une erreur, errorDesc est alloué
    NSLog(@"erreur : %@",errorDesc);
    [errorDesc release];
    // retourner NO éventuellement selon l'implémentation désirée
}

self.prefDictionary = [NSMutableDictionary
➥ dictionaryWithDictionary:[dictionary objectForKey:@"prefDictionary"]];
return YES;
}

```

Et le code pour écrire dans un fichier :

```

- (BOOL)writeDataToFile {
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
➥ NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *plistPath = [documentsDirectory
➥ stringByAppendingPathComponent:@"preferences.plist"];

    NSString *errorDesc;
    NSMutableDictionary *prefDict;
    // on remplace par les valeurs que l'on a modifiées
    prefDict = [[NSMutableDictionary alloc] initWithObjects:[NSArray
➥ arrayWithObject:prefDictionary]
➥ forKeys:[NSArray arrayWithObject:@"prefDictionary"]];

    NSData *plistData = [NSPropertyListSerialization
➥ dataFromPropertyList:prefDict format:NSPropertyListXMLFormat_v1_0
➥ errorDescription:&errorDesc];
    if (plistData) {
        BOOL returned = [plistData writeToFile:plistPath atomically:YES];
        return returned;
    }
    else {
        [errorDesc release];
        return NO;
    }
    return NO;
}

```

Info

N'espérez pas récupérer/lire le fichier directement dans Xcode. En effet, il faut bien comprendre que les fichiers de votre projet seront compilés (les .h, .m, .cpp...) pour créer l'exécutable de votre application ou copiés (les images, fichiers texte...) dans le dossier de l'application. Par contre, ils ne seront pas mis à jour si votre application les modifie. Cela vous semble facile ? Tant mieux ! Je me permets ce rappel car la question a déjà été posée sur notre forum plusieurs fois...

Pour accéder aux fichiers depuis votre iPhone, rendez-vous dans Xcode : Window > Organizer > votre appareil > la liste des applications, puis la petite flèche sur la droite. Ainsi, vous rapatrierez les données de l'application sur votre ordinateur pour les interpréter. Pour récupérer le dossier du simulateur, il faut se placer dans le dossier racine de votre compte. Ensuite, Bibliothèque > Application Support > iPhone Simulator > la version de l'OS sur lequel l'application est compilée > l'identifiant de votre application > Documents.

Dans un nouveau projet de type View-based Application nommé PropertyList, placez un slider, un switch et un text field ainsi qu'un bouton Valider dans la vue. Dans la méthode appelée par le bouton, nous allons sauver les états courants de chaque élément :

```
- (void)freezeButton : (id) sender {  
    PropertyListAppDelegate *appDelegate =  
        (PropertyListAppDelegate*)[[UIApplication sharedApplication] delegate];  
    // On fait une référence vers le dictionnaire qui contient les données  
    NSMutableDictionary *prefDict =  
        (NSMutableDictionary*)appDelegate.prefDictionary;  
  
    NSNumber *sliderValue = [NSNumber numberWithFloat:aSlider.value];  
    [prefDict setObject:sliderValue forKey:kSlider];  
    [prefDict setObject:[NSNumber numberWithBool:aSwitch.on] forKey:kSwitch];  
    [prefDict setObject:aTextField.text forKey:kTextField];  
}
```

Attention

J'attire votre attention sur cette ligne :

```
PropertyListAppDelegate *appDelegate = (PropertyListAppDelegate*)[[UIApplication  
    sharedApplication] delegate];
```

(N'oubliez pas d'importer `PropertyListAppDelegate.h`).

Cette ligne fait pointer `appDelegate` vers l'objet de la classe `PropertyListAppDelegate` instancié par votre application.

N'oubliez pas ces quelques lignes pour définir les clés utilisées :

```
#define kSlider @"slider"
#define kSwitch @"switch"
#define kTextField @"textField"
```

En récupérant le fichier preferences.plist comme précisé au début de la section, nous obtenons ceci par exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
➥ "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>prefDictionary</key>
  <dict>
    <key>slider</key>
    <real>0.42418771982192993</real>
    <key>switch</key>
    <true/>
    <key>textField</key>
    <string>iPuP</string>
  </dict>
</dict>
</plist>
```

On retrouve bien le dictionnaire avec la clé prefDictionary puis les valeurs de nos trois éléments.

Astuce

Si vous avez besoin de sauver seulement quelques valeurs, comme un pseudo ou une taille de police, vous pouvez utiliser plus simplement NSUserDefaults. Voici comment écrire `[[NSUserDefaults standardUserDefaults] objectForKey:@"aKey"]`; et lire `[[NSUserDefaults standardUserDefaults] setObject:anObject forKey:@"aKey"]`. Notez qu'il existe `boolForKey:`, `integerForKey:`, ...

Imaginons maintenant que vous souhaitez sauver des objets plus compliqués, très facilement... Cela fait rêver, non ? Eh bien c'est possible. Grâce au protocole NSCoder, nous allons archiver les données puis les enregistrer dans un fichier. En fait, vous allez très vite vous rendre compte de la puissance de NSCoder pour archiver des structures de données reliées entre elles. Typiquement, les fichiers nib utilisent ce protocole pour archiver votre hiérarchie de vue, et la dépiler au lancement de l'application. Attention, le fichier nib ne sera pas modifié par votre application, mais seulement désarchivé depuis la valeur par défaut lorsque vous avez compilé le binaire.

Finalement, l'objet désarchivé sera une copie conforme (propriétés, dépendances...) de l'objet archivé.

Un exemple concret serait une liste de matières. Créons donc une classe Matière implémentant le protocole NSCoding :

```
#import <Foundation/Foundation.h>
#define kNumero @"number"
#define kMatiereName @"Matiere"
#define kDateDebut @"DateDebut"
#define kCouleur @"CouleurCell"
#define kProfesseur @"Professeur"
#define kSalle @"Salle"

@interface Matiere : NSObject <NSCoding> {
    int number;
    NSString *matiere;
    NSString *dateDebut;
    UIColor *couleur; // cet objet ne sert à rien mais vous montre simplement que l'on
    ➔ peut archiver n'importe quel objet implémentant le protocole NSCoding
    NSString *professeur;
    NSString *salle;
}

- (id)initWithNumber:(int)myNumber color:(UIColor*)myColor date:(NSString*)myDate
    ➔ teacher:(NSString*)myTeacher subject:(NSString*)mySubject room:(NSString*)myRoom;
@property int number;
@property (nonatomic, retain) NSString *subject;
@property (nonatomic, retain) NSString *date;
@property (nonatomic, retain) UIColor *color;
@property (nonatomic, retain) NSString *teacher;
@property (nonatomic, retain) NSString *room;
@end
```

Attention

Les objets que vous souhaitez archiver doivent implémenter le protocole NSCoding sinon cela ne marchera pas. Par exemple, la classe UIImage n'implémente pas ce protocole. Par contre, la classe NSData, si. Il faut donc archiver les données de votre image sous forme de NSData.

Le protocole NSCoding requiert l'implémentation de deux méthodes : une servant à archiver et l'autre à désarchiver. Voici ce que vous devez ajouter :

```
#pragma mark -
#pragma mark NSCoding

- (void)encodeWithCoder:(NSCoder *)coder {
    [coder encodeInt:self.number forKey:kNumero];
    [coder encodeObject:self.color forKey:kCouleur];
```

```

    [coder encodeObject:self.date forKey:kDateDebut];
    [coder encodeObject:self.teacher forKey:kProfesseur];
    [coder encodeObject:self.room forKey:kSalle];
    [coder encodeObject:self.subject forKey:kMatiereName];

}

- (id)initWithCoder:(NSCoder *)coder {
    if (self = [super init]) {
        self.number = [coder decodeIntForKey:kNumero];
        self.color = [coder decodeObjectForKey:kCouleur];
        self.date = [coder decodeObjectForKey:kDateDebut];
        self.teacher = [coder decodeObjectForKey:kProfesseur];
        self.subject = [coder decodeObjectForKey:kMatiereName];
        self.room = [coder decodeObjectForKey:kSalle];
    }
    return self;
}

```

N'oubliez pas ces deux méthodes (et les @synthesize) :

```

- (id)initWithNumber:(int)myNumber color:(UIColor*)myColor date:(NSString*)myDate
➥ teacher:(NSString*)myTeacher subject:(NSString*)mySubject room:(NSString*)myRoom {

    if (self = [super init]) {
        self.number = myNumber;
        self.color = myColor;
        self.date = myDate;
        self.teacher = myTeacher;
        self.subject = mySubject;
        self.room = myRoom;
    }
    return self;
}

-(void)dealloc{
    [color release];
    [date release];
    [teacher release];
    [room release];
    [subject release];

    [super dealloc];
}

```

Enfin, pour lire depuis un fichier :

```
-(BOOL) unarchiveSubject {
    BOOL toReturn = NO;
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
    ↪ NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *path =
    ↪ [documentsDirectory stringByAppendingPathComponent:@"Matieres.myArchive"];
    if(path)
    {
        NSData *data;
        NSKeyedUnarchiver *unarchiver;

        data = [[NSData alloc] initWithContentsOfFile:path];
        if(data)
        {
            unarchiver = [[NSKeyedUnarchiver alloc] initForReadingWithData:data];

            NSMutableArray *arrayArchived =
            ↪ [unarchiver decodeObjectForKey:@"Matieres"];
            self.list = arrayArchived;
            [unarchiver finishDecoding];
            [unarchiver release];
            toReturn = YES;
        }
        [data release];
        return toReturn;
    }
}
```

Et archiver :

```
-(BOOL) archiveSubject {
    NSMutableData *data;
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
    ↪ NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *archivePath = [documentsDirectory
    ↪ stringByAppendingPathComponent:@"Matieres.myArchive"];

    NSKeyedArchiver *archiver;
    BOOL result;

    data = [NSMutableData data];
    archiver = [[NSKeyedArchiver alloc] initForWritingWithMutableData:data];
```

```
[archiver encodeObject:list forKey:@"Matieres"];
[archiver finishEncoding];
result = [data writeToFile:archivePath atomically:YES];
[archiver release];

return BOOL;
}
```

info

Vous pouvez créer ou utiliser n'importe quelle extension pour votre fichier de sauvegarde.

Où comment se simplifier la vie lorsque l'on souhaite stocker des données.

Core Data est un framework créé par Apple qui stocke des données dans la méthode de conception MVC (*Model View Controller* ou Modèle Vue Contrôleur). Vous me direz : "Et pourquoi j'utiliserais ça ? Je m'en sors tout seul !" Ce à quoi je vous répondrais que Core Data permet dans un premier temps de réduire significativement la taille de votre code utilisé pour stocker vos données (de l'ordre de 50 %) et dans un second temps, qu'il est entièrement optimisé et éprouvé.

Avant de commencer, il est donc nécessaire de comprendre les enjeux du MVC. MVC se compose de trois entités distinctes :

- **Le Modèle.** Il contient l'ensemble des données qui seront manipulées. Il propose donc un accès à ces données en lecture, et bien entendu des méthodes pour les modifier.
- **La Vue.** Elle va présenter ces données à l'écran et permet l'interaction avec l'utilisateur.
- **Le Contrôleur.** Il fait l'interface entre la vue et le modèle. Il reçoit les événements de la vue, demande à modifier les données en conséquence, et avertit la vue que les données ont changé.

Tout ceci est résumé par un schéma Figure 36.1.

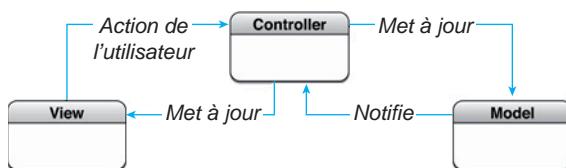


Figure 36.1 : Le modèle Model View Controller.

Avec Core Data nous allons réaliser une petite application en deux temps. Le but de cette application sera de créer un profil d'employé (Nom, Prénom, Âge, Sexe).

Dans un premier temps, nous apprendrons à utiliser Core Data en ajoutant des données grâce à une vue créée pour l'occasion. De plus, ces données seront triées, et nous pourrons les supprimer. Puis, nous verrons comment gérer l'édition.

APPRENDRE À UTILISER CORE DATA

Les prérequis sont la Fiche 12 sur les table view ainsi que les premières fiches pour appréhender l'univers de développement Cocoa Touch.

Créez un nouveau projet Window-based Application que vous nommerez CoreData. N'oubliez pas de cocher Use Core Data for storage. Vous remarquerez que, par rapport à d'habitude, vous avez un nouveau fichier : CoreData.xcdatamodel. C'est ce fichier qui va vous permettre de créer le graph des données de votre modèle. De plus, l'application delegate contient des nouvelles méthodes, qui sont utilisées par Core Data pour gérer les données.

Tout d'abord créez votre graph de données. Pour cela, double-cliquez sur CoreData.xcdatamodel. Vous devriez obtenir la fenêtre de la Figure 36.2.

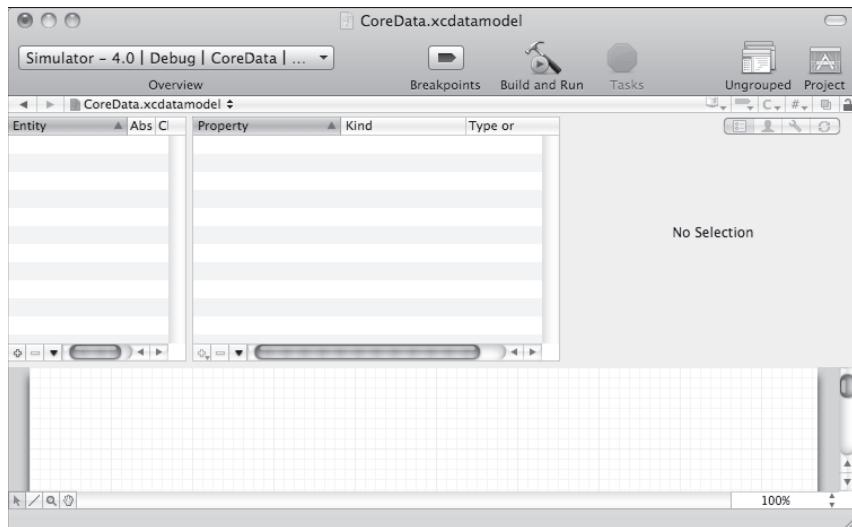


Figure 36.2 : La fenêtre d'accueil pour créer le graph de données.

Puis ajoutez une entité. Une entité correspond à une classe. Faites :

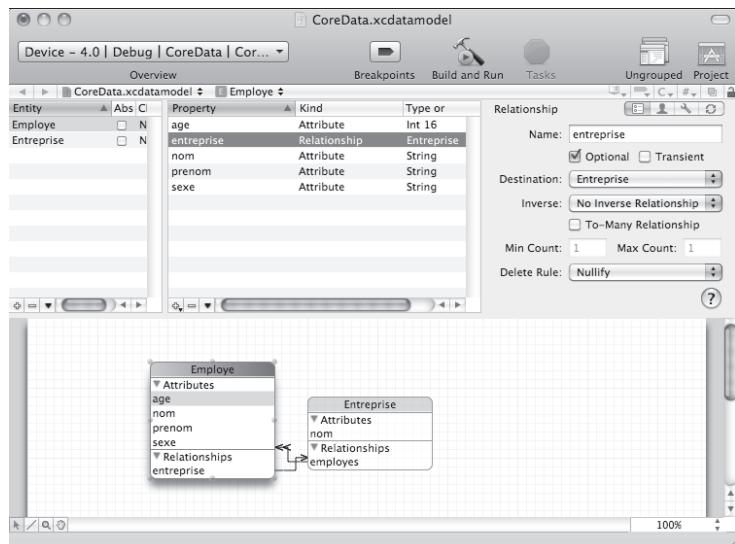
1. Design > Data Model > Add Entity. Changez son nom pour Entreprise.
2. Ajoutez-lui un attribut, qui correspond à une variable d'instance. Pour l'entreprise, ce sera simplement son nom.
3. Après avoir sélectionné l'entité Entreprise, faites Design > Data Model > Add Attributes.
4. Changez le nom pour nom et choisissez comme type : String.

Vous remarquerez que la liste des données disponibles est finie. Pensez-y !

Faites de même en créant une entité Employe pour lequel vous ajouterez trois attributs de type string : nom, prenom, sexe et un attribut de type integer 16 : age. Une entreprise a plusieurs employés, mais un employé ici n'aura qu'une entreprise. En effet, avec Core Data, nous pouvons facilement ajouter des relations entre les entités et c'est ce qui fait sa puissance, entre autres. Pour ce faire :

1. Cliquez sur l'entité Entreprise, et faites Design > Data Model > Add Relationship.
2. Choisissez comme nom employes puis dans Destinations l'entité Employe.
3. Cochez To-Many Relationship, car une entreprise a plusieurs employés.
4. Cliquez ensuite sur l'entité Employe et ajoutez une relation avec Entreprise nommée entreprise. Un employé n'appartenant qu'à une unique entreprise à la fois, décochez To-Many Relationship.

Vous devriez obtenir quelque chose de semblable à la Figure 36.3.



Pour rendre les choses plus propres, cliquez sur l'entité Employe puis sur entreprise et choisissez employes dans Inverse. Cette fois-ci, la flèche est devenue unique et bidirectionnelle comme à la Figure 36.4.

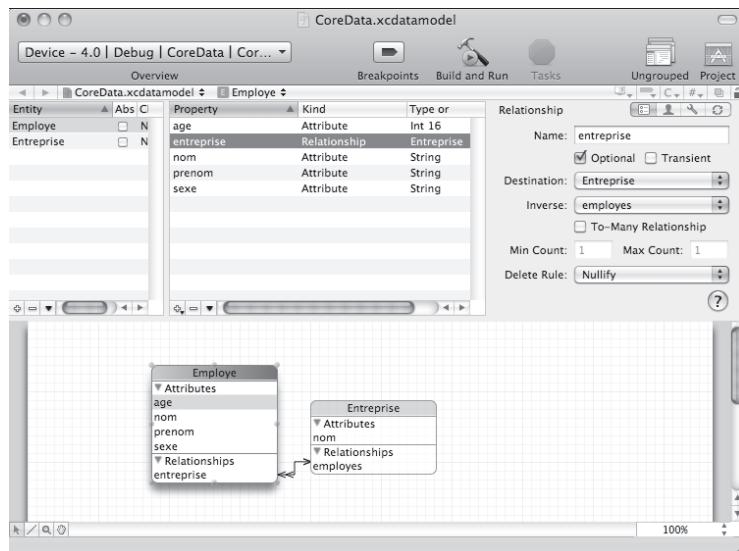


Figure 36.3 :
Les relations
entre Employe
et Entreprise.

Figure 36.4 : Une
flèche plus lisible.

Avant de revenir au code, il faut générer les classes depuis les objets créés avec l'interface graphique. Pour cela :

1. Cliquez sur l'entité Entreprise et faites Files > New File.
2. Choisissez ensuite iPhone OS > Cocoa Touch Class > Managed Object Class.
3. Appuyez sur Next. Laissez le chemin proposé par défaut et cliquez encore sur Next.
4. Sélectionnez Entreprise et Employe puis cochez seulement Generate accessors et Generate Obj-C 2.0 properties. Vous devriez obtenir quatre fichiers : Entreprise.h, Entreprise.m, Employe.h et Employe.m.

Arrêtons-nous quelques instants sur ces fichiers. Vous remarquerez qu'il n'y a pas de méthode dealloc. C'est tout à fait normal, Core Data se chargeant de gérer les objets en mémoire, cela se fait en interne.

De plus, en regardant Employe.h, il ne vous aura pas échappé que la variable age était en fait un objet NSNumber. En effet, Core Data gère des objets et non des types comme int ou float.

Revenons maintenant à notre application. Vous allez afficher dans une première table view la liste des entreprises. Pour cela, il faut créer une table view. Ajoutez un nouveau fichier UIViewController > UITableViewController que vous nommerez TableRootViewController. Vous allez l'afficher par-dessus la window en modifiant les fichiers :

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

@interface CoreDataAppDelegate : NSObject <UIApplicationDelegate> {

    UIWindow *window;
    UINavigationController *navigationController;

    @private
        NSManagedObjectContext *managedObjectContext_;
        NSManagedObjectModel *managedObjectModel_;
        NSPersistentStoreCoordinator *persistentStoreCoordinator_;
}

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

@property (nonatomic, retain, readonly) NSManagedObjectContext *managedObjectContext;
@property (nonatomic, retain, readonly) NSManagedObjectModel *managedObjectModel;
@property (nonatomic, retain, readonly) NSPersistentStoreCoordinator *persistentStoreCoordinator;
@property (nonatomic, retain) UINavigationController *navigationController;

- (NSString *)applicationDocumentsDirectory;
```

```
@end
CoreDataAppDelegate.m
#import "CoreDataAppDelegate.h"
#import "TableRootViewController.h"

@implementation CoreDataAppDelegate
@synthesize window;
@synthesize navigationController;
#pragma mark -
#pragma mark Application lifecycle
-(BOOL)application:(UIApplication
    *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Configure le table view controller
    TableRootViewController *rootViewController = [[TableRootViewController
        alloc] initWithStyle:UITableViewStylePlain];

    NSManagedObjectContext *context = [self managedObjectContext];
    if (!context) {
        // Gestion de l'erreur
    }
    // On passe le managed object context au controller
    rootViewController.managedObjectContext = context;
    UINavigationController *aNavController = [[UINavigationController
        alloc] initWithRootViewController:rootViewController];
    self.navigationController = aNavController;
    [window addSubview:[navigationController view]];
    [window makeKeyAndVisible];
    [rootViewController release];
    [aNavController release];
    return YES;
}
// ne pas modifier la suite
// mais dans le dealloc, release le navigationController
```

Il faut également modifier votre TableRootViewController. Il va posséder une instance managedObjectContext qui servira de passerelle jusqu'à Core Data ainsi qu'un bouton pour ajouter une entreprise. Modifier le .h :

```
#import <UIKit/UIKit.h>
@interface TableRootViewController : UITableViewController {
    NSManagedObjectContext *managedObjectContext;
    UIBarButtonItem *addButton;
}
@property (nonatomic, retain) NSManagedObjectContext *managedObjectContext;
@end
```

Ajoutez dans le .m :

```
@synthesize managedObjectContext;
```

Ensuite, dans le viewDidLoad, ajoutez un titre :

```
// Mettre le titre  
self.title = @"Entreprises";
```

Puis un bouton pour modifier la liste des entreprises :

```
// Mettre les boutons self.navigationItem.leftBarButtonItem = self.editButtonItem;
```

Et un autre pour ajouter une nouvelle entreprise :

```
addButton = [[UIBarButtonItem alloc]  
initWithBarButtonSystemItem:UIBarButtonSystemItemAdd  
target:self  
action:@selector(ajouterEntreprise)];  
self.navigationItem.rightBarButtonItem = addButton;
```

À ce stade, votre application devrait ressembler à la Figure 36.5.

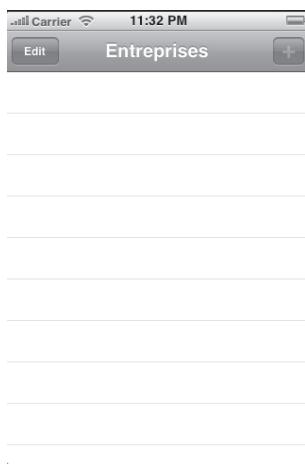


Figure 36.5 : Le début d'une grande aventure.

Pour donner à notre entreprise, nous allons recourir à une UIAlertView qui contient un champ de texte. Pour cela, nous allons concevoir une subclass de UIAlertView que nous nommerons AlertViewTextField. Créez une nouvelle classe NSObject nommée AlertViewTextField.

```
AlertViewTextField.h  
#import <Foundation/Foundation.h>  
@interface AlertViewTextField : UIAlertView {  
    UITextField *theTextField;  
}  
@property (nonatomic, retain) UITextField *theTextField;
```

```
@property (readonly) NSString *textEntered;

-(id)initWithTitle:(NSString *)title message:(NSString *)message
➥ delegate:(id)delegate cancelButtonTitle:(NSString *)cancelButtonTitle
➥ okButtonTitle:(NSString *)okButtonTitle;

@end

AlertViewTextField.m
#import "AlertViewTextField.h"
@implementation AlertViewTextField
@synthesize theTextField;
@synthesize textEntered;

-(id)initWithTitle:(NSString *)title message:(NSString *)message delegate:(id)delegate
➥ cancelButtonTitle:(NSString *)cancelButtonTitle okButtonTitle:(NSString *)okButtonTitle
{
    if (self = [super initWithTitle:title message:message delegate:delegate
➥ cancelButtonTitle:cancelButtonTitle otherButtonTitles:okButtonTitle,
➥ nil])
    {
        UITextField *aTextField = [[UITextField alloc]
initWithFrame:CGRectMake(12.0, 45.0, 260.0, 25.0)];
        [aTextField setBorderStyle:UITextBorderStyleRoundedRect];
        [aTextField setBackgroundColor:[UIColor clearColor]];
        [self addSubview:aTextField];
        self.theTextField = aTextField;
        [aTextField release];
    }
    return self;
}
-(void)show {
    // on rend le champ de texte first responder, pour permettre automatiquement
    // d'ajouter du texte
    [theTextField becomeFirstResponder];
    [super show];
}
// retourne le texte entré
-(NSString *)textEntered {
    return theTextField.text;
}
-(void)dealloc
{
    [theTextField release];
    [super dealloc];
}
@end
```

Ensuite, de retour dans le TableRootViewController, dans la méthode ajouterEntreprise, vous allez afficher une alert view personnalisée. N'oubliez pas de faire l'import de UIAlertViewTextField.h.

```
- (void) ajouterEntreprise {
    UIAlertViewTextField *alert = [[UIAlertViewTextField alloc] initWithTitle:@"Entreprise"
message:@"Ajouter une nouvelle entreprise"
delegate:self
cancelButtonTitle:@"Annuler"
okButtonTitle:@"Ajouter"];
    [alert show];
    [alert release];
}
```

Lorsque l'utilisateur annulera ou validera sa saisie, la méthode -(void)alertView:(UIAlertView *)alertView willDismissWithButtonIndex:(NSInteger)buttonIndex sera appelée. À cet endroit, si l'utilisateur valide son choix, il faut ajouter une nouvelle entreprise.

Commencez par récupérer le texte entré :

```
- (void)alertView:(UIAlertView *)alertView willDismissWithButtonIndex:(NSInteger)
buttonIndex {
    if (buttonIndex != [alertView cancelButtonIndex])
    {
        // on récupère le texte entré
        NSString *entered = [(UIAlertViewTextField *)alertView textEntered];
    }
}
```

Ensuite, créez une nouvelle entreprise en ajoutant cette ligne, toujours dans la condition if :

```
// créer une nouvelle entreprise
Entreprise *entreprise = (Entreprise *)[NSEntityDescription
➥ insertNewObjectForEntityForName:@"Entreprise"
➥ inManagedObjectContext:managedObjectContext];
```

Info

Attention à ne pas oublier vos imports de .h ! Entreprise.h et UIAlertViewTextField.h

La ligne ci-dessus demande à Core Data d'insérer un nouvel objet et nous attribuons le bon nom à cette nouvelle entreprise par la suite :

```
// lui mettre le nom saisi
entreprise.nom = entered;
```

Vous allez maintenant sauver cette entreprise en ajoutant ces quelques lignes :

```
// sauvegarde NSError *error;
if (![[managedObjectContext save:&error]) {
    // Gérer l'erreur
}
```

Pour afficher la liste des entreprises dans la table view, vous avez besoin d'un tableau. Pour cela, déclarez dans TableRootViewController.h

```
NSMutableArray *entrepriseArray;
```

que vous initialiserez dans le viewDidLoad. Ensuite, de retour à la sauvegarde de la nouvelle entreprise, vous allez gérer l'ajout de la nouvelle entreprise tout en triant la liste par ordre alphabétique ! Ajoutez cette ligne :

```
// ajout d'une nouvelle entreprise dans le tableau  
[entrepriseArray addObject:entreprise];
```

Triez cette liste avec NSSort. On demande à trier par ordre alphabétique à partir de la clé nom.

```
// on trie  
NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"nom"  
➥ ascending:YES];  
NSArray *sortDescriptors = [NSArray arrayWithObjects:sortDescriptor, nil];  
[entrepriseArray sortUsingDescriptors:sortDescriptors];  
[sortDescriptor release];
```

Puis, pour rendre le tout joli et animé, ajoutez la nouvelle entreprise dans la liste avec une petite animation. Pour connaître son emplacement, il suffit de demander l'index de l'objet entreprise.

```
// on anime l'insertion d'une nouvelle ligne  
NSIndexPath *indexPath = [NSIndexPath  
➥ indexPathForRow:[entrepriseArray indexOfObject:entreprise] inSection:0];  
[self.tableView insertRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]  
➥ withRowAnimation:UITableViewRowAnimationFade];  
[self.tableView scrollToRowAtIndexPath:indexPath  
➥ atScrollPosition:UITableViewScrollPositionTop animated:YES];
```

Pour finir, modifiez ces trois méthodes afin d'afficher la liste des entreprises :

```
-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {  
    // Retourne le nombre de sections  
    return 1;  
}  
-(NSInteger)tableView:(UITableView *)tableView  
➥ numberOfRowsInSection:(NSInteger)section {  
    // Retourne le nombre de lignes dans la section  
    return [entrepriseArray 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];
```

```

    }
Entreprise *entreprise = (Entreprise*)[entrepreneurArray objectAtIndex:[indexPath row]];
cell.textLabel.text = entreprise.nom;
return cell;
}

```

Compilez, puis lancez votre application. Ajoutez des entreprises, elles doivent bien s'afficher !

Quittez puis relancez votre application. Attention à bien la terminer pour ne pas qu'elle reste en tâche de fond (double appui sur la touche Home, puis appui long sur votre application et appui sur le sens interdit). Zut, les données ne s'affichent pas ! Core Data ne marcherait pas ? J'ai fait tout ça pour rien ? Pas d'inquiétude, on va voir tout ça...

RÉCUPÉRER LA SAUVEGARDE DES DONNÉES

Nous avons vu comment modifier et ajouter une entreprise dans la base de données créée par Core Data. Ces objets entreprise existent bien, il faut simplement les récupérer au lancement de l'application. Ainsi dans le viewDidLoad, demandons à récupérer les entités Entreprise.

```

NSFetchRequest *request = [[NSFetchRequest alloc] init];
NSEntityDescription *entity = [NSEntityDescription entityForName:@"Entreprise"
    inManagedObjectContext:managedObjectContext];
[request setEntity:entity];

```

Ensuite, trions les données, toujours par ordre alphabétique :

```

NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"nom"
    ascending:YES];
NSArray *sortDescriptors = [NSArray arrayWithObjects:sortDescriptor, nil];
[request setSortDescriptors:sortDescriptors];
[sortDescriptor release];

```

Puis exécutons la requête pour récupérer le tableau :

```

NSError *error;
NSMutableArray *mutableFetchResults =
    [[[managedObjectContext executeFetchRequest:request error:&error] mutableCopy];
if (mutableFetchResults == nil) {
    // Gestion de l'erreur
}
self.entrepreneurArray = mutableFetchResults;
[mutableFetchResults release];
[request release];

```

Info

Vous remarquerez que l'on fait une `mutableCopy` du tableau retourné par `executeFetchRequest:error:`. En effet, cette méthode retourne un `NSArray`, alors que nous souhaitons un `NSMutableArray`...

Par ailleurs, n'oubliez pas de déclarer `entrepreneurArray` en `@property` et `@synthesize` !

Avant de continuer, faisons le point sur vos implémentations des méthodes dealloc. J'espère que vous ne m'avez pas attendu ! Voici les deux dealloc que vous devriez avoir :

```
CoreDataAppDelegate.m
-(void)dealloc {
    [navigationController release];
    [managedObjectContext release];
    [managedObjectModel release];
    [persistentStoreCoordinator release];
    [window release];
    [super dealloc];
}

TableRootViewController.m
-(void)dealloc {
    [addButton release];
    [managedObjectContext release];
    [entrepriseArray release];
    [super dealloc];
}
```

Avec l'arrivée du multitâche sous l'iOS 4, ce type d'application où l'on gère l'écriture des données est assez contraignant à déboguer. En effet, votre application est suspendue par défaut. Ce qui signifie que le système, dans la mesure du possible, garde l'état de l'application comme lorsque vous l'avez quitté. Il est donc impossible de savoir si la persistance des données vient du multitâche ou de votre sauvegarde. Pour la quitter réellement, il faut donc appuyer deux fois sur le bouton Home, puis la supprimer de l'Historique des ouvertures.

Pour éviter ceci, vous pouvez ajouter cette clé dans le CoreData-Info.plist :

Application does not run in background et cocher la case qui apparaît à droite.

Info

Rappelez-vous qu'*Apple ne garantit pas la sauvegarde complète de l'état courant de votre application. Il faut donc implémenter cette sauvegarde manuellement pour éviter les mauvaises surprises.*

Avant d'ajouter les employés dans les entreprises, nous allons gérer l'édition des entreprises, notamment leur suppression. Pour cela, implémenter ces deux méthodes. La première est appelée quand l'utilisateur glisse le doigt horizontalement sur la cellule puis appuie sur le bouton Supprimer. La seconde, quant à elle, est appelée lorsque l'utilisateur appuie sur le bouton Éditer :

```
#pragma mark
#pragma mark Edition
-(void)tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
    forRowAtIndexPath:(NSIndexPath *)indexPath {
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        // On supprime l'entreprise pointée par la cellule choisie par
        // l'utilisateur pour la suppression
```

```

NSManagedObject *entrepriseToDelete =
    [entrepriseArray objectAtIndex:indexPath.row];
[managedObjectContext deleteObject:entrepriseToDelete];
// On met à jour le tableau, ainsi que la table view.
[entrepriseArray removeObjectAtIndex:indexPath.row];
[tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
    withRowAnimation:UITableViewRowAnimationFade];
// On sauvegarde les changements
NSError *error;
if (![managedObjectContext save:&error])
{
    // Gestion de l'erreur
}
}
}

-(void)setEditing:(BOOL)editing animated:(BOOL)animated {
    [super setEditing:editing animated:animated];
    // On désactive le bouton d'ajout si on est en mode édition
    self.navigationItem.rightBarButtonItem.enabled = !editing;
}
}

```

Arrivé à ce stade, je vous propose d'aller boire un verre, vous aérer l'esprit. Nous allons en effet voir beaucoup de points. Le premier sera la gestion de l'édition (ajout/suppression de cellules) et le suivant, la création d'une table view pour permettre de créer de nouveaux employés.

AJOUTER DES EMPLOYÉS

Commençons par créer un nouveau table view controller que vous nommerez TableEmployesViewController. Ce table view controller présentera une liste des différents employés de l'entreprise sélectionnée. Il permettra l'ajout d'un nouvel employé ou la suppression d'un existant.

L'appel à ce contrôleur se fera après l'appui d'une ligne contenant le nom d'une entreprise. Ainsi, dans TableRootViewController.m, ajoutez cette méthode :

```

-(void)tableView:(UITableView *)tableView
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    TableEmployesViewController *viewController =[[TableEmployesViewController
        alloc] initWithStyle:UITableViewStylePlain];

    // on passe au contrôleur l'entreprise sélectionnée
    viewController.entreprise = [entrepriseArray objectAtIndex:indexPath.row];
    [self.navigationController pushViewController:viewController
        animated:YES];
    [viewController release];
}

```

Voici le .h de TableEmployesViewController :

```
#import <UIKit/UIKit.h>
@class Entreprise;
@interface TableEmployesViewController : UITableViewController {
    UIBarButtonItem *addButton;
    NSMutableArray *employesArray;
    Entreprise *entreprise;
}
@property (nonatomic, retain) NSMutableArray *employesArray;
@property (nonatomic, retain) Entreprise *entreprise;
@end
```

Modifions tout d'abord le viewDidLoad :

```
-(void)viewDidLoad {
    [super viewDidLoad];
    // on autorise la sélection des cells même pendant l'édition (dans le cas où
    // l'on autoriserait la modification d'un employé)
    //self.tableView.allowsSelectionDuringEditing = YES;
    // Mettre le titre
    self.title = entreprise.nom;
    // Mettre le bouton d'édition
    self.navigationItem.rightBarButtonItem = self.editButtonItem;
    // On initialise le tableau à partir des employés de l'entreprise
    self.employesArray = [[[NSMutableArray alloc]
    // initWithArray:[entreprise.Employes allObjects]] autorelease];
}
```

N'oubliez pas l'import de Entreprise.h ainsi que les @synthesize.

Info

Attention, employes est un objet de la classe NSSet. Pour récupérer un tableau des objets qu'il contient, il faut donc utiliser allObjects.

Notez que si l'on souhaitait récupérer tous les employés stockés dans la base de données (juste au-dessus, nous récupérons les employés de l'entreprise sélectionnée) il faudrait procéder comme suit :

```
NSManagedObjectContext *context = entreprise.managedObjectContext;
NSFetchRequest *request = [[NSFetchRequest alloc] init];
// Récupérer tous les employés
NSEntityDescription *entity = [NSEntityDescription entityForName:@"Employe"
inManagedObjectContext:context];
[request setEntity:entity];
// Les trier par leur nom
```

```

NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"nom"
➥ ascending:YES];
NSArray *sortDescriptors = [NSArray arrayWithObjects:sortDescriptor, nil];
[request setSortDescriptors:sortDescriptors];
[sortDescriptor release];
NSError *error;
NSMutableArray *mutableFetchResults = [[context executeFetchRequest:request
➥ error:&error] mutableCopy];
if (mutableFetchResults == nil) {
    // Gestion de l'erreur
}
self.employesArray = mutableFetchResults;
[mutableFetchResults release];
[request release];

```

Nous allons nous occuper de l'édition. L'idée est la suivante : en mode normal, nous affichons la liste des employés. En mode édition, nous affichons toujours cette liste, avec une ligne en plus qui nous permettra d'ajouter un nouvel employé.

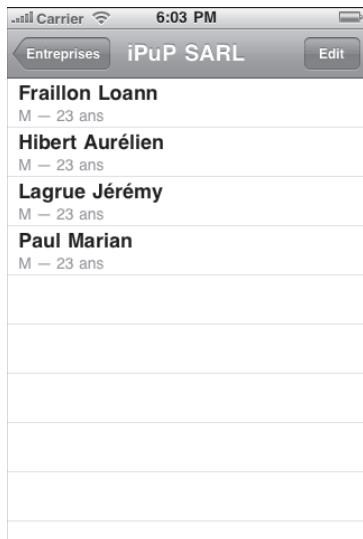


Figure 36.6 : La vue en mode normal.

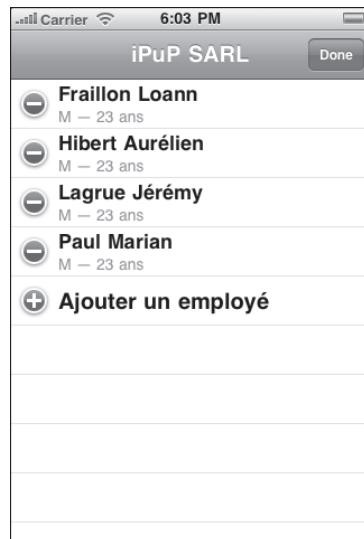


Figure 36.7 : La vue en mode édition.

Commençons par ajouter une ligne si l'on est en mode édition :

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    // Retourne le nombre de sections
    return 1;
}

-(NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section {
    // Retourne le nombre d'employés dans l'entreprise, + 1 si on est en mode
    // édition pour la ligne d'ajout d'un employé
    // Très important, sinon votre application plante en vous disant qu'il lui
    // manque des lignes ou qu'elle en a trop...
    NSUInteger count = [employeesArray count];
    if (self.editing) {
        count++;
    }
    return count;
}
```

Ensuite, il faut implémenter le comportement de l'application lorsque l'on appuie sur le bouton Edit. Cela va nous permettre d'ajouter ou enlever la ligne avec le signe + et de sauver les changements :

```
// gestion de l'état lorsque l'on touche le bouton éditer (Edit ou Done)
-(void)setEditing:(BOOL)editing animated:(BOOL)animated {
    [super setEditing:editing animated:animated];
    // On ne montre pas le bouton de retour lorsque l'on est en mode édition
    [self.navigationItem setHidesBackButton:editing animated:YES];
    // On indique que l'on va commencer les mises à jour (suppression /ajout)
    // de lignes (ici, une ligne à la fois)
    [self.tableView beginUpdates];
    // On récupère le nombre d'employés NSUInteger count = [employeesArray count];
    // On crée un tableau d'indexPath (ici un seul objet sera présent dans le
    // tableau)
    NSArray *employeeInsertIndexPath =
    // [NSArray arrayWithObject:[NSIndexPath indexPathForRow:count inSection:0]];
    // On ajoute ou enlève la ligne
    // En fait, cette ligne est celle qui contient le bouton '+' pour ajouter un employé !
    // On l'ajoute lorsqu'on commence l'édition, puis on l'enlève à l'appui du bouton Done
    UITableViewRowAnimation animationStyle = UITableViewRowAnimationNone;
    if (editing) {
        if (animated) {
            animationStyle = UITableViewRowAnimationFade;
        }
        [self.tableView insertRowsAtIndexPaths:employeeInsertIndexPath
withRowAnimation:animationStyle];
    }
}
```

```

    else {
        // on enlève toujours avec animation
        [self.tableView deleteRowsAtIndexPaths:employeIndexPath
withRowAnimation:UITableViewRowAnimationFade];
    }

    // les mises à jour sur la table view sont finies
    [self.tableView endUpdates];
    // Si on a fini d'éditer, alors on sauve le context de Core Data
    if (!editing) {
        NSManagedObjectContext *context = entreprise.managedObjectContext;
        NSError *error = nil;
        if (![context save:&error]) {
            // Gestion de l'erreur
        }
    }
}
}

```

Pour afficher un joli bouton "+" et un bouton "-", implémentez cette méthode :

```

-(UITableViewCellEditingStyle)tableView:(UITableView
➥*)tableView editingStyleForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // La Ligne pour ajouter reçoit un marqueur '+' l'autre un marqueur '-'
    if (indexPath.row == [employeesArray count])
    {
        return UITableViewCellEditingStyleInsert;
    }
    return UITableViewCellEditingStyleDelete;
}

```

Ensuite, implémentons la méthode appelée lorsque l'on clique sur Delete ou l'ajout d'un nouvel employé. La suppression est la même que celle vue dans TableRootViewController.

```

-(void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle)
➥editingStyle forRowAtIndexPath:(NSIndexPath *)indexPath {
    // On récupère le context
    NSManagedObjectContext *context = entreprise.managedObjectContext;
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        // On supprime la ligne du dataSource
        // On cherche l'employé à supprimer
        Employe *employe = [employeesArray objectAtIndex:indexPath.row];
        // On supprime l'employé du context
        [entreprise removeEmployesObject:employe];
        // Notez que si un même employé appartenait à plusieurs entreprises, et
        // que l'on souhaitait supprimer cet employé définitivement de toutes les
        // entreprises, on ferait
    }
}

```

```
/* [context deleteObject:employe]; */  
// On enlève l'employé du tableau, ainsi que la ligne qui lui correspond  
[employesArray removeObject:employe];  
[tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]  
    withRowAnimation:UITableViewRowAnimationFade];  
// On sauvegarde les changements  
NSError *error = nil;  
if (![context save:&error]) {  
    // Gestion de l'erreur  
}  
}  
else  
if (editingStyle == UITableViewCellEditingStyleInsert) {  
    // On crée une nouvelle instance de Employé, que l'on insert dans le  
    // tableau et la table view après création  
    [self insertEmployeAnimated:YES];  
}  
}  
}
```

Pour l'instant, laissons la méthode `insertEmployeAnimated:vide` :

```
-(void)insertEmployeAnimated:(BOOL)animated { }
```

Et n'oubliez pas de la déclarer dans le .h ! Affichons maintenant les employés, ainsi que le texte pour la ligne d'ajout :

```
-(UITableViewCell *)tableView:(UITableView *)tableView  
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {  
    NSUInteger row = indexPath.row;  
    // On utilise ici une cellule par défaut. On pourrait très bien utiliser  
    // une cellule par défaut pour la ligne d'ajout d'un Employé et une  
    // personnalisée pour l'affichage des informations de l'employé, si l'on  
    // souhaitait insérer une photo par exemple  
    if (row == [employesArray count]) {  
        // affichage de la ligne qui permet l'ajout d'un nouvel employé  
        static NSString *AddCellIdentifier = @"AddCell";  
        UITableViewCell *cell =  
            [tableView dequeueReusableCellWithIdentifier:AddCellIdentifier];  
        if (cell == nil) {  
            cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault  
                reuseIdentifier:AddCellIdentifier] autorelease];  
            cell.textLabel.text = @"Ajouter un employé";  
        }  
        return cell;  
    }  
    // affichage normal
```

```

static NSString *NormalCellIdentifier = @"NormalCell";
UITableViewCell *cell =
↳ [tableView dequeueReusableCellWithIdentifier:NormalCellIdentifier];
if (cell == nil) {
    cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle
        ↳ reuseIdentifier:NormalCellIdentifier] autorelease];
}
Employe *employe = (Employe*)[employesArray objectAtIndex:row];
cell.textLabel.text = [NSString stringWithFormat:@"%@ %@", 
    ↳ employe.nom, employe.prenom];
cell.detailTextLabel.text = [NSString stringWithFormat:@"%@ — %d ans",
    ↳ employe sexe, [employe.age intValue]];
return cell;
}

```

Lorsque l'on crée une nouvelle entreprise, elle n'a pas d'employé. On présente la vue pour l'ajouter automatiquement :

```

-(void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];
    if ([employesArray count] == 0) {
        // Il n'y a aucun employé dans l'entreprise, donc on souhaite en ajouter un
        [self setEditing:YES animated:NO];
        // on lance la vue pour l'ajout de
        ↳ l'employé
        [self insertEmployeAnimated:YES];
    }
}

```

Pour créer un nouvel employé, nous allons concevoir une vue spécialement pour l'occasion, rien que ça ! Elle ressemblera à la Figure 36.8. Créez un nouveau table view controller que vous nommerez TableEmployeDetailViewController.

Je ne détaillerai pas le code ici car ce n'est pas le propos. Il est en outre suffisamment commenté pour que vous le compreniez.

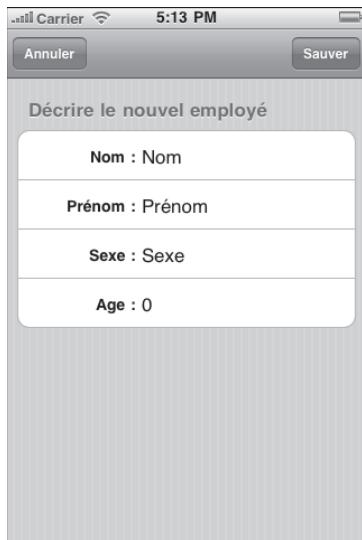


Figure 36.8 : La vue d'ajout d'un employé.

L'essentiel est d'utiliser un dictionnaire temporaire pour stocker les valeurs. Chaque cellule de notre table view correspond à un attribut de Employe. À chaque attribut correspond un numéro de ligne qui servira également comme clé dans le dictionnaire temporaire (0 correspond au nom, 1 au prénom...)

Par exemple :

```
tempValues : {  
    0 = Paul;  
    3 = 23;  
    1 = Marian;  
    2 = M;  
}
```

Ces valeurs sont définies dans le .h :

```
#import <UIKit/UIKit.h>  
@class Employe;  
#define kNumberOfEditableRows 4 // Nom, prénom, age et sexe  
#define kNameRowIndex 0  
#define kFirstNameRowIndex 1  
#define kSexRowIndex 2  
#define kAgeIndex 3  
#define kLabelTag 4096  
  
@interface TableEmployeDetailViewController :  
    ↪ UITableViewController<UITextFieldDelegate> {  
    Employe *employe;  
    NSArray *fieldLabels;  
    NSMutableDictionary *tempValues;  
    UITextField *textFieldBeingEdited;  
}  
@property (nonatomic, retain) Employe *employe;  
@property (nonatomic, retain) NSArray *fieldLabels;  
@property (nonatomic, retain) NSMutableDictionary *tempValues;  
@property (nonatomic, retain) UITextField *textFieldBeingEdited;  
  
-(void)cancel:(id)sender;  
-(void)save:(id)sender;  
-(void)textFieldDone:(id)sender;  
@end
```

N'oubliez pas dans le .m les import et @synthesize :

```
#import "TableEmployeDetailViewController.h"  
#import "TableEmployesViewController.h"  
#import "Employe.h"  
#import "Entreprise.h"  
@implementation TableEmployeDetailViewController
```

```

@synthesize employe;
@synthesize fieldLabels;
@synthesize tempValues;
@synthesize textFieldBeingEdited;
...
@end

```

Commençons par le viewDidLoad et le dealloc :

```

-(void)viewDidLoad {
    // On charge les valeurs des labels précisant les noms des champs
    NSArray *array = [[NSArray alloc] initWithObjects:@"Nom :", @"Prénom :",
        @"Sexe :", @"Age :", nil];
    self.fieldLabels = array;
    [array release];
    // bouton annuler
    UIBarButtonItem *cancelButton = [[UIBarButtonItem alloc]
        initWithTitle:@"Annuler" style:UIBarButtonItemStylePlain target:self
        action:@selector(cancel:)];
    self.navigationItem.leftBarButtonItem = cancelButton;
    [cancelButton release];
    // bouton sauvegarder
    UIBarButtonItem *saveButton = [[UIBarButtonItem alloc] initWithTitle:@"Sauver"
        style:UIBarButtonItemStyleDone target:self action:@selector(save:)];
    self.navigationItem.rightBarButtonItem = saveButton;
    [saveButton release];
    // déclaration du dictionnaire temporaire
    NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
    self.tempValues = dict;
    [dict release];
    [super viewDidLoad];
}

-(void)dealloc {
    [textFieldBeingEdited release];
    [tempValues release];
    [employe release];
    [fieldLabels release];
    [super dealloc];
}

```

Ensuite, les méthodes cancel: et save::

```

-(void)cancel:(id)sender{
    // Dans le cas où l'on annule, cela sous-entend que l'on ne souhaite pas
    // sauvegarder l'employé
    // On récupère tous les contrôleurs dans la navigation controller

```

```
// En 0 : TableRootViewController
// En 1 : TableEmployesViewController
// En 2 : TableEmployeDetailViewController
NSArray *allControllers = [self.navigationController viewControllers];
TableEmployesViewController *parent = (TableEmployesViewController*)
➥ [allControllers objectAtIndex:[allControllers count] - 2];
// avant dernier objet
Entreprise *entreprise = parent.entreprise;
// on enlève de l'entreprise
[entreprise removeEmployesObject:employe];
// et du tableau
[parent.employesArray removeObject:employe];
[self.navigationController popViewControllerAnimated:YES];
}

-(void)save:(id)sender {
    // Si on est en train d'éditer un text field (on n'a pas appuyé sur la touche
    ➥ retour), alors on prend la valeur en cours que l'on sauvegarde dans le
    ➥ dictionnaire temporaire
    if (textFieldBeingEdited != nil) {
        NSNumber *tagAsNum= [NSNumber numberWithInt:textFieldBeingEdited.tag];
        [tempValues setObject:textFieldBeingEdited.text forKey: tagAsNum];
        [tagAsNum release];
    }
    // On parcourt les valeurs du dictionnaire temporaire pour setter
    ➥ correctement l'employé

    for (NSNumber *key in [tempValues allKeys]) {
        switch ([key intValue]) {
            case kNameRowIndex:
                employe.nom = [tempValues objectForKey:key];
                break;
            case kFirstNameRowIndex:
                employe.prenom = [tempValues objectForKey:key];
                break;
            case kSexRowIndex:
                employe.sex = [tempValues objectForKey:key];
                break;
            case kAgeIndex:
                employe.age =
                ➥ [NSNumber numberWithInt:[[tempValues objectForKey:key] intValue]];
                break;
            default:
                break;
        }
    }
}
```

```

// On pourrait sauvegarder ici. On ne le fait pas, car par choix,
➥ TableEmployesViewController s'en charge en appuyant sur Done
// Notez que si vous ajoutez plusieurs employés sans appuyer sur Done, et
➥ que votre application crash, ces derniers ne seront pas sauvés. Pour
➥ éviter ceci, sauvez ici le context.
[self.navigationController popViewControllerAnimated:YES];
}

```

Ensuite, les méthodes qui servent à l'affichage de la table view :

```

// titre au dessus de la table view
-(NSString *)tableView:(UITableView *)tableView
➥ titleForHeaderInSection:(NSInteger)section {
    return @"Décrire le nouvel employé";
}

#pragma mark -
#pragma mark Table Data Source Methods
-(NSInteger)tableView:(UITableView *)tableView
➥ numberOfRowsInSection:(NSInteger)section {
    return kNumberOfEditableRows;
}

-(UITableViewCell *)tableView:(UITableView *)tableView
➥ cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *EmployeCellIdentifier = @"EmployeCellIdentifier";
    UITableViewCell *cell =
    ➤ [tableView dequeueReusableCellWithIdentifier:EmployeCellIdentifier];
    if (cell == nil) {
        // construction de la cellule
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
        ➤ reuseIdentifier:EmployeCellIdentifier] autorelease];
        UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(10, 10, 95, 25)];
        label.textAlignment = NSTextAlignmentRight;
        label.tag = kLabelTag;
        label.font = [UIFont boldSystemFontOfSize:14];
        [cell.contentView addSubview:label];
        [label release];
        UITextField *textField = [[UITextField alloc] initWithFrame:CGRectMake(110,
        ➤ 12, 180, 25)];
        textField.clearsOnBeginEditing = NO;
        [textField setDelegate:self];
        [textField addTarget:self
        ➤ action:@selector(textFieldDone:)forControlEvents:UIControlEventEditingDidEndOnExit];
        [cell.contentView addSubview:textField];
    }
}

```

```
NSUInteger row = [indexPath row];
UILabel *label = (UILabel *)[cell viewWithTag:kLabelTag];
label.text = [fieldLabels objectAtIndex:row];
// on cherche le text field parmi les sous vues de la cell (label,textField, ...)
UITextField *textField = nil;
for (UIView *oneView in cell.contentView.subviews) {
    if ([oneView isKindOfClass:[UITextField class]])
        textField = (UITextField *)oneView;
}
NSNumber *rowAsNum = [[NSNumber alloc] initWithInt:row];
switch (row) {
    case kNameRowIndex: // Si le nom a déjà été renseigné, alors on le place
        if ([[tempValues allKeys] containsObject:rowAsNum])
            textField.text = [tempValues objectForKey:rowAsNum];
        else // sinon, on choisit le nom par défaut de employé lors de sa création
            textField.text = employe.nom; // on choisit si le text field fait
            // appel au dictionnaire ainsi que le type du clavier
        [textField setAutocorrectionType:UITextAutocorrectionTypeNo];
        [textField setKeyboardType:UIKeyboardTypeDefault];
        break;
    case kFirstNameRowIndex:
        if ([[tempValues allKeys] containsObject:rowAsNum])
            textField.text = [tempValues objectForKey:rowAsNum];
        else
            textField.text = employe.prenom;
        [textField setAutocorrectionType:UITextAutocorrectionTypeNo];
        [textField setKeyboardType:UIKeyboardTypeDefault];
        break;
    case kAgeIndex:
        if ([[tempValues allKeys] containsObject:rowAsNum])
            textField.text = [tempValues objectForKey:rowAsNum];
        else // attention, employe.age est un NSNumber
            textField.text = [NSString stringWithFormat:@"%d",
employe.age intValue]];
        [textField setAutocorrectionType:UITextAutocorrectionTypeNo];
        [textField setKeyboardType:UIKeyboardTypeNumberPad];
        break;
    case kSexRowIndex:
        if ([[tempValues allKeys] containsObject:rowAsNum])
            textField.text = [tempValues objectForKey:rowAsNum];
        else
            textField.text = employe.sex;
        [textField setAutocorrectionType:UITextAutocorrectionTypeNo];
        [textField setKeyboardType:UIKeyboardTypeDefault];
```

```

        break;
    default:
        break;
    }

    // Par défaut, on a placé par exemple "Nom :" dans le label, et "Nom" par
    ➔ défaut pour le nom de l'employé, on regarde donc si le texte du label
    ➔ raccourci de 2 lettres en partant de la fin (" :") est égal au nom par défaut.
    // Si c'est le cas, c'est que l'on doit modifier le texte. Ainsi, on autorise
    ➔ le textField à effacer son contenu lorsqu'on le choisit
    if([[label.text substringToIndex:[label.text length]
    ➔ - 2] isEqualToString:textField.text] || [textField.text isEqualToString:@""])
        textField.clearsOnBeginEditing = YES;
    else
        textField.clearsOnBeginEditing = NO;
    if (textFieldBeingEdited == textField) textFieldBeingEdited = nil;
        textField.tag = row; [rowAsNum release];
    return cell;
}

#pragma mark -
#pragma mark Table Delegate Methods
-(NSIndexPath *)tableView:(UITableView *)tableView
    ➔ willSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    return nil; }

```

Et pour finir celles qui gèrent les text fields :

```

// Appelé lorsque l'on appuie sur la touche return du clavier
-(void)textFieldDone:(id)sender {
    // On cherche la cell qui contient le text field qui vient de finir d'être
    ➔ édité pour retrouver ensuite le numéro de la ligne correspondante
    // Note : sender est ce textField
    UITableViewCell *cell = (UITableViewCell *)[sender superview] superview;
    UITableView *table = (UITableView *)[cell superview];
    NSIndexPath *textFieldIndexPath = [table indexPathForCell:cell];
    NSUInteger row = [textFieldIndexPath row];
    row++;
    // si on est à la fin, on retourne au début ! un row%kNumberOfEditableRows ferait
    ➔ également l'affaire
    if (row >= kNumberOfEditableRows)
        row = 0;
    NSUInteger newIndex[] = {0, row};
    NSIndexPath *newPath = [[NSIndexPath alloc] initWithIndexes:newIndex length:2];
    // on cherche la cellule suivante ((row+1)%kNumberOfEditableRows)
    UITableViewCell *nextCell = [self.tableView cellForRowAtIndexPath:newPath];
    [newPath release];
}

```

```
UITextField *nextField = nil;
for (UIView *oneView in nextCell.contentView.subviews) {
    if ([oneView isKindOfClass:[UITextField class]])
        nextField = (UITextField *)oneView;
}
// on affiche le clavier pour le prochain text field
[nextField becomeFirstResponder];
}

#pragma mark -
Text Field Delegate Methods
-(void)textFieldDidBeginEditing:(UITextField *)textField {
    self.textFieldBeingEdited = textField;
}
-(void)textFieldDidEndEditing:(UITextField *)textField {
    // On pourrait tester ici si l'âge est réel (< 60 par exemple) // De plus, un
    ↪ test pourrait être fait sur le sexe qui devrait être M ou F
    NSNumber *tagAsNum = [[NSNumber alloc] initWithInt:textField.tag];
    [tempValues setObject:textField.text forKey>tagAsNum];
    [textField setClearsOnBeginEditing:NO];
    [tagAsNum release];
}
```

Ouf ! Une dernière petite chose à faire, et vous pourrez enfin ajouter des employés à votre entreprise ! En effet, il faut implémenter la méthode de TableEmployesViewController appelée lorsque l'on choisit d'ajouter un nouvel employé !

```
-(void)insertEmployeAnimated:(BOOL)animated {
    // on crée une nouvelle instance de Employe que l'on ajoute dans Core Data
    ↪ ainsi que dans le tableau
    Employe *employe = [NSEntityDescription insertNewObjectForEntityForName:@"Employe"
        ↪ inManagedObjectContext:[entreprise managedObjectContext]];
    // valeurs par défaut
    employe.nom = @"Nom";
    employe.prenom = @"Prénom";
    employe.sex = @"Sexe";
    employe.age = [NSNumber numberWithInt:0];
    // on ajoute l'employé à l'entreprise
    [entreprise addEmployesObject:employe];
    // Ajout dans le tableau et ajout d'une ligne dans la table view
    [employesArray addObject:employe];
    NSIndexPath *indexPath = [NSIndexPath indexPathForRow:[employesArray count]
        ↪ - 1 inSection:0];
    UITableViewRowAnimation animationStyle = UITableViewRowAnimationNone;

    if (animated) {
        animationStyle = UITableViewRowAnimationFade;
```

```
}

[self.tableView insertRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
    withRowAnimation:UITableViewRowAnimationFade];
// On push un nouveau table view controller pour éditer les informations
// de l'employé
TableEmployeDetailViewController *detailViewController =
    [[[TableEmployeDetailViewController
        alloc] initWithStyle:UITableViewStyleGrouped];
detailViewController.employe = employe;
[self.navigationController
    pushViewController:detailViewController animated:YES];
[detailViewController release]; }
```

N'oubliez pas cette ligne pour que tout fonctionne :

```
#import "TableEmployeDetailViewController.h"
```

Et le reload data pour afficher les bonnes données lorsque l'on revient de l'édition (avec le tri) :

```
-(void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    // on trie
    NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
initWithKey:@"nom" ascending:YES];
    NSArray *sortDescriptors = [NSArray arrayWithObjects:sortDescriptor, nil];
    [employeesArray sortUsingDescriptors:sortDescriptors];
    [sortDescriptor release];
    [self.tableView reloadData];
}
```

Jusqu'à maintenant, nous n'avons pas spécifiquement parlé d'iPad. Toutefois, vous devez savoir que l'iPad fonctionne également sous iOS.

Cet iOS est actuellement la version 3.2, mais la majorité de ce que nous avons appris jusqu'ici peut se transposer dans une application iPad. Regardons les principales différences.

CE QU'IL N'Y A PAS

- L'iPad n'est pas un téléphone : vous n'aurez donc pas accès aux SMS ni à la fonction téléphone.
- Actuellement, l'OS de l'iPad ne permet pas le multitâche. Cependant, vous pouvez, en attendant la mise à jour du système, utiliser le protocole NSCoder pour sauvegarder l'état courant de votre application.
- L'iPad ne possède pas d'appareil photo, ni à l'arrière, ni à l'avant comme l'iPhone 4.

LES PLUS

- En développant une application pour iPad, vous pourrez utiliser de nouvelles vues, parmi UISplitViewController (qui présente deux vues en même temps), UIPopoverController (qui présente une petite vue par-dessus les autres) et de nouvelles façons de présenter des vues modales.
- Avec l'iPad, votre application a la possibilité de partager des fichiers avec d'autres applications.
- L'écran est plus large (1024 × 768 pixels), plus de détails pourront s'afficher !

UN ÉCRAN PLUS LARGE

L'écran de l'iPad est plus large, il faudra donc adapter les graphismes en fonction. En effet, il ne suffit pas d'agrandir les graphismes d'une version iPhone, mais les repenser pour intégrer plus de détails. L'utilisateur de l'iPad, face à son grand écran, s'attendra à bénéficier d'une application en haute définition.

De plus, qui dit écran plus large dit possibilité de placer plus d'éléments sur une même vue. N'hésitez donc pas à proposer davantage de fonctionnalités !

Enfin, essayez de ne jamais changer toutes les vues de l'écran en même temps. En effet, l'utilisateur de l'iPhone a l'habitude d'un petit écran qui ne peut pas tout afficher au même endroit. Par contre, sur iPad, comme l'application Mail ou Bloc notes, vous pouvez placer une split view (UISplitViewController) qui présente, par exemple, à la fois une table view et son détail.

LA ROTATION

Il n'y a pas de sens pour regarder l'iPad, c'est même rappelé dans la publicité officielle... À ce titre, il est plus que préconisé de rendre votre application apte à présenter ses données en portrait comme en paysage. Pour cela, je vous conseille d'employer Interface Builder pour construire vos vues, car il sera plus facile d'observer les changements après vos réglages sans avoir à recompiler l'application chaque fois.

L'élément le plus important est la propriété `autoresizingMask` de la classe `UIView`. En changeant cette propriété, vous modifierez le comportement de la vue lorsque l'iPad tourne. Voici les différentes valeurs :

- `UIViewAutoresizingNone` La vue ne sera pas modifiée.
- `UIViewAutoresizingFlexibleLeftMargin` La vue se redimensionnera en s'étirant (ou se rétrécissant) vers la gauche.
- `UIViewAutoresizingFlexibleRightMargin` La vue se redimensionnera en s'étirant (ou se rétrécissant) vers la droite.
- `UIViewAutoresizingFlexibleTopMargin` La vue se redimensionnera en s'étirant (ou se rétrécissant) vers le haut.
- `UIViewAutoresizingFlexibleBottomMargin` La vue se redimensionnera en s'étirant (ou se rétrécissant) vers le bas.
- `UIViewAutoresizingFlexibleWidth` La vue se redimensionnera en étirant (ou rétrécissant) sa largeur.
- `UIViewAutoresizingFlexibleHeight` La vue se redimensionnera en étirant (ou rétrécissant) sa hauteur.

Sous Interface Builder, dans l'onglet Size d'une vue, vous pourrez choisir ces propriétés d'un simple clic en observant à travers l'animation le comportement. À la Figure 37.1, il est spécifié que la vue va être ancrée en haut et à gauche. Dans le code, cela correspondrait à `UIViewAutoresizingFlexibleBottomMargin` | `UIViewAutoresizingFlexibleRightMargin`.

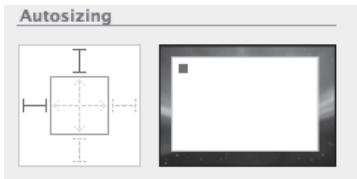


Figure 37.1 : Utilisez l'autoresizing.

Info

Les valeurs du type énumérées `UIViewAutoresizing` ne sont pas définies par hasard. En fait, la valeur est un entier, mais que l'on pourrait représenter en binaire :

```
enum {  
    UIViewAutoresizingNone          = 0,  
    UIViewAutoresizingFlexibleLeftMargin = 1 << 0,  
    UIViewAutoresizingFlexibleWidth     = 1 << 1,  
    UIViewAutoresizingFlexibleRightMargin = 1 << 2,  
    UIViewAutoresizingFlexibleTopMargin   = 1 << 3,  
    UIViewAutoresizingFlexibleHeight      = 1 << 4,  
    UIViewAutoresizingFlexibleBottomMargin = 1 << 5  
};  
typedef NSUInteger UIViewAutoresizing;
```

Ainsi, `UIViewAutoresizingNone` correspond à `00000000b`, `UIViewAutoresizingLeftMargin` à `00000001b`, `UIViewAutoresizingFlexibleWidth` à `00000010b`, ...

De ce fait, vous pouvez spécifier plusieurs paramètres en appliquant l'opérateur OU (|), obtenu avec le raccourci clavier `⌘+Alt+⇧+1`. Par exemple, en faisant une `view.autoresizingMask = UIViewAutoresizingFlexibleLeftMargin | UIViewAutoresizingFlexibleWidth;`, la valeur de `autoresizingMask` sera de `00000011b`, soit 3 en décimal.

Enfin, votre application iPad est susceptible d'être lancée en mode portrait ou paysage. Il est possible de définir la bonne image au chargement en spécifiant `Default-Portrait.png` et `Default-Landscape.png`. Figure 37.2, vous retrouvez une capture écran des images de chargement et l'icône comprenant les fichiers pour iPad, iPhone et iPhone 4. De plus, les images sont localisées (changent selon la langue de l'appareil).

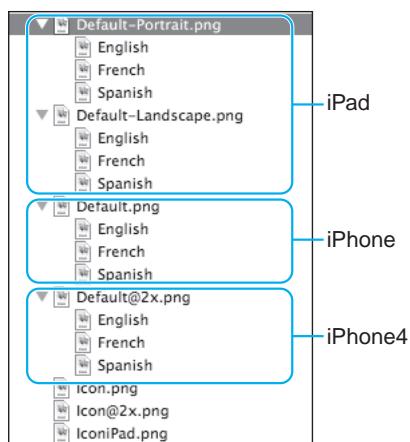


Figure 37.2 : Les images de chargement.

CRÉER UNE APPLICATION UNIVERSELLE

Maintenant, vous avez le choix :

- si vous souhaitez distribuer votre application seulement sur iPad, alors la question ne se pose pas ;
- si vous souhaitez distribuer votre application sur les deux plates-formes, vous pouvez créer un même projet pour une application dite universelle. Ainsi, l'utilisateur n'achètera qu'une version qu'il pourra lancer sur ces deux appareils s'il les possède ;
- vous décidez de créer deux applications. L'avantage est que l'utilisateur d'un appareil seul aura une application plus légère à télécharger et conserver. Par contre, s'il possède un iPhone et un iPad, il devra payer deux fois l'application, ce qui est très frustrant pour lui.

Finalement, c'est un dilemme : proposer une application universelle, pour satisfaire l'utilisateur, ou en vendre deux et espérer une rémunération plus juste face au travail fourni pour la deuxième version.

N'oubliez pas que le patron de conception MVC (*Model View Controller*) prend tout son sens ici. En effet, la vue pour l'iPad sera différente de celle pour l'iPhone, le contrôleur sera donc différent lui aussi car il ne gérera pas les mêmes éléments d'interfaces. Par contre, les données restent les mêmes !

Voici deux conseils pour gérer les différences dans une même classe :

- Pour charger les éléments d'interface selon l'appareil, utilisez `UI_USER_INTERFACE_IDIOM()`. Par exemple :

```
if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPhone)
{
    // on présente l'interface pour l'iPhone
}
else
{
    // dans le cas où il n'y a pas que l'iPad où l'iPhone (si dans un futur
    ➔ proche un nouvel appareil sort...)
    if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad)
    {
        // interface iPad
    }
}
```

- Pour utiliser une fonctionnalité non disponible sur les deux plates-formes, testez si la classe existe. Par exemple :

```
if(NSClassFromString(@"UIPopoverController"))
{
    // on peut afficher la popover
}
```

Nous allons voir quelques prémisses (pour comprendre le mécanisme d'une application universelle) en réalisant une application qui affichera une liste d'ingrédients. Pour plus de détails concernant les applications universelles, référez-vous à l'exemple TopPaid disponible dans la documentation.

Créez un nouveau projet de type Window-based Application, universelle (Figure 37.3) que vous nommerez UniversalApp.

À l'ouverture du projet, vous remarquerez que vous avez plusieurs application delegate et MainWindow.xib, un pour l'iPhone et un pour l'iPad (Figure 37.4).

Nous allons commencer par nous occuper de la partie iPad. Pour cela, ouvrez le fichier MainWindow_iPad.xib, puis ajoutez un nouveau contrôleur UISplitViewController (Figure 37.5).

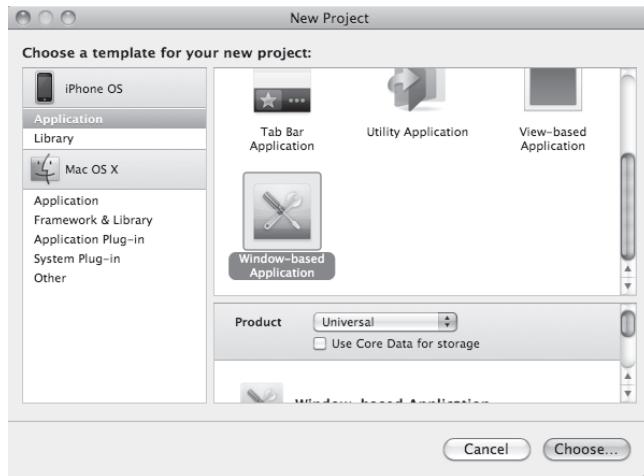


Figure 37.3 : Un nouveau projet universel.

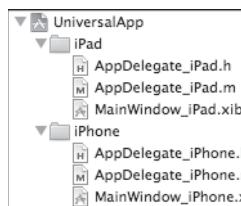


Figure 37.4 : Un fichier par plate-forme.

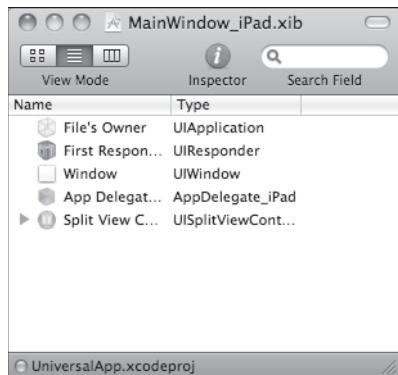


Figure 37.5 : Ajoutez un split view controller.

Ensuite, retournez sous Xcode puis dans AppDelegate_iPad.h et déclarez le contrôleur :

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

@interface AppDelegate_iPad : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    UISplitViewController *splitViewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet UISplitViewController *splitViewController;
@end
```

Dans le .m, ajoutez la vue à la fenêtre :

```
@synthesize window;
@synthesize splitViewController;

#pragma mark -
#pragma mark Application lifecycle

- (BOOL)application:(UIApplication
    *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    [window addSubview:splitViewController.view];
    [window makeKeyAndVisible];

    return YES;
}
```

Comme vous le voyez Figure 37.6, un split view controller affiche deux vues. Créons-les en ajoutant un nouveau fichier de type UITableViewController, nommé RootViewController comme à la Figure 37.7 (pour la vue de gauche) et un UIViewController nommé DetailViewController comme à la Figure 37.8.



Figure 37.6 : Un split view controller.

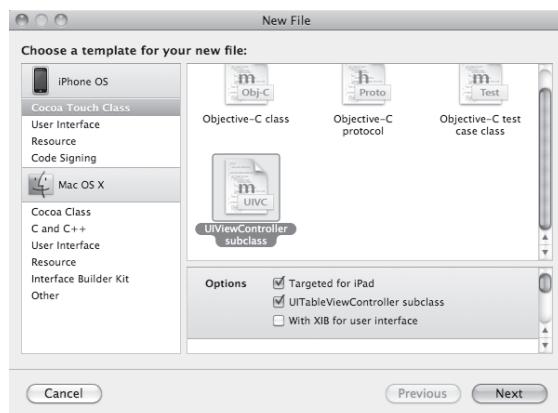


Figure 37.7 : Créez un fichier de type table view controller.



Figure 37.8 : Créez un nouveau contrôleur.

Ouvrez de nouveau le fichier MainWindow_iPad.xib, puis connectez le contrôleur comme à la Figure 37.9.

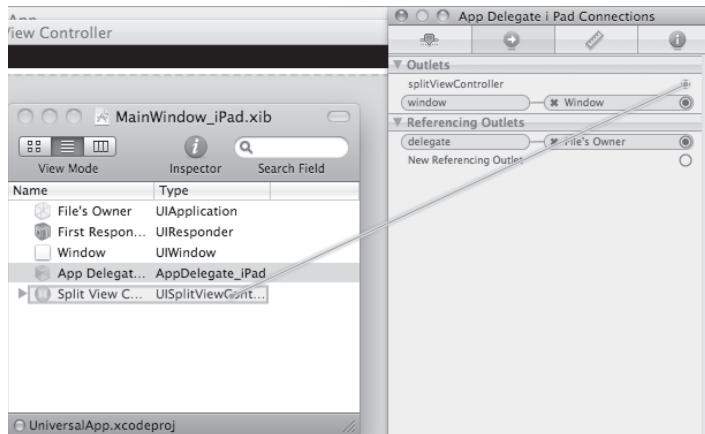


Figure 37.9 : Connectez le split view controller.

Ensuite, déroulez le split view controller dans la fenêtre MainWindow_iPad.xib, puis cliquez sur le view controller. Dans l'inspecteur, onglet Identity, spécifiez DetailViewController comme à la Figure 37.10. Puis, dans l'onglet Attributes, spécifiez le nom du fichier nib (Figure 37.11).

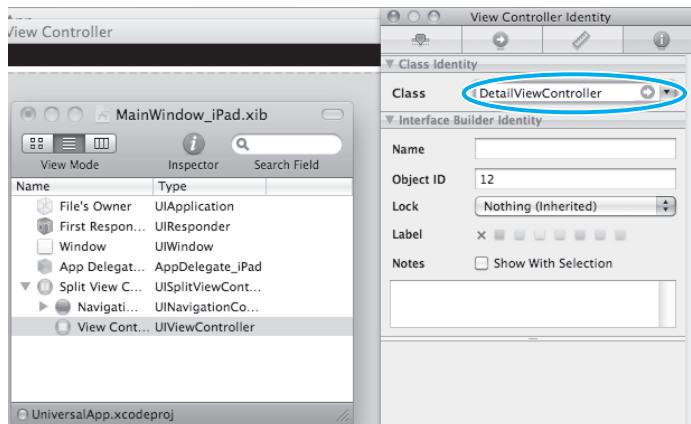


Figure 37.10 : Changez le nom de la classe pour DetailViewController.

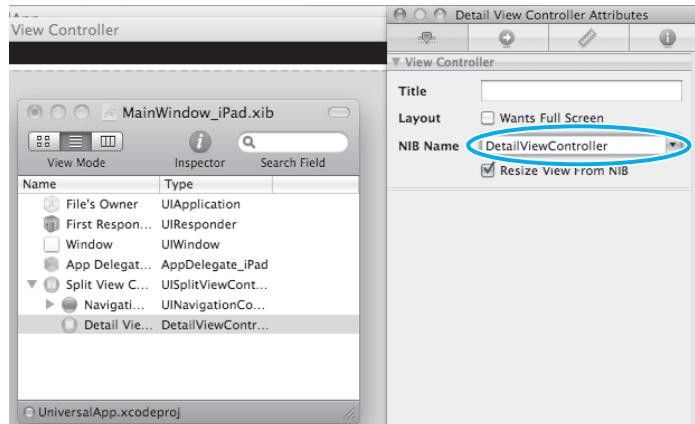


Figure 37.11 : Spécifiez le fichier NIB à charger.

Déroulez ensuite le navigation controller pour trouver le table view controller. Dans l'onglet Identity, spécifiez comme classe RootViewController (Figure 37.12).

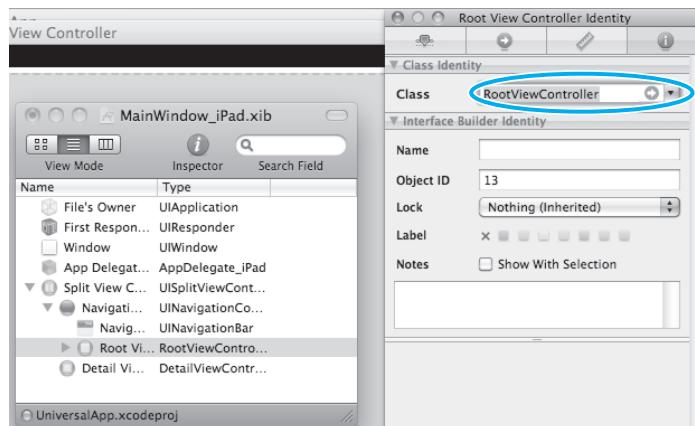


Figure 37.12 : Changez le nom de la classe pour RootViewController.

Sauvez vos modifications, puis retournez dans Xcode et dans RootViewController.h. Nous allons déclarer un tableau que nous initialiserons avec une liste de légumes à afficher.

```
#import <UIKit/UIKit.h>
@interface RootViewController : UITableViewController {
    NSMutableArray *arrayToDisplay;
}
@end
```

Dans le .m, initialisez le tableau :

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

    self.title = @"Ratatouille";

    arrayToDisplay = [[NSMutableArray alloc] init];
    [arrayToDisplay addObject:@"Tomate"];
    [arrayToDisplay addObject:@"Courgette"];
    [arrayToDisplay addObject:@"Aubergine"];
    [arrayToDisplay addObject:@"Poivron"];
}
```

Puis modifiez ces méthodes :

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section {
    return [arrayToDisplay 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];
    }
    cell.textLabel.text = [arrayToDisplay objectAtIndex:indexPath.row];
    return cell;
}
```

Pour finir avec la partie iPad, nous allons afficher la sélection sur la vue du contrôleur DetailViewController. Pour cela, dans DetailViewController.h, déclarez un label :

```
#import <UIKit/UIKit.h>
@interface DetailViewController : UIViewController {
    UILabel *detailLabel;
}
@property (nonatomic, retain) IBOutlet UILabel *detailLabel;
@end
```

Ajoutez dans le .m :

```
@implementation DetailViewController
@synthesize detailLabel;
```

Ensuite, rendez-vous dans DetailViewController.xib, puis ajoutez un label. Reliez-le comme à la Figure 37.13.

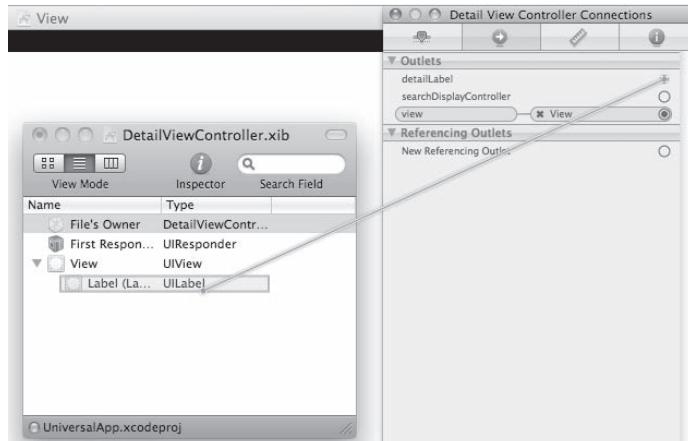


Figure 37.13 :
Reliez le label.

Sauvez et quittez, puis, dans RootViewController.h, nous allons déclarer un objet de la classe DetailViewController.

```
#import <UIKit/UIKit.h>
@class DetailViewController;
@interface RootViewController : UITableViewController {
    NSMutableArray *arrayToDisplay;
    IBOutlet DetailViewController *detailViewController;
}
@end
```

Dans le .m, affichez dans le label le légume sélectionné :

```
#import "RootViewController.h"
#import "DetailViewController.h"

@implementation RootViewController

// Laisser tel quel

#pragma mark -
#pragma mark Table view delegate

- (void)tableView:(UITableView *)tableView
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
```

```

detailViewController.detailLabel.text =
    [arrayToDisplay objectAtIndex:indexPath.row];
[detailViewController.detailLabel sizeToFit];
}

```

Ouvrez le fichier `MainWindow_iPad.xib`, puis cliquez sur le root ViewController et dans l'onglet Connections, reliez le `detailViewController` comme à la Figure 37.14.

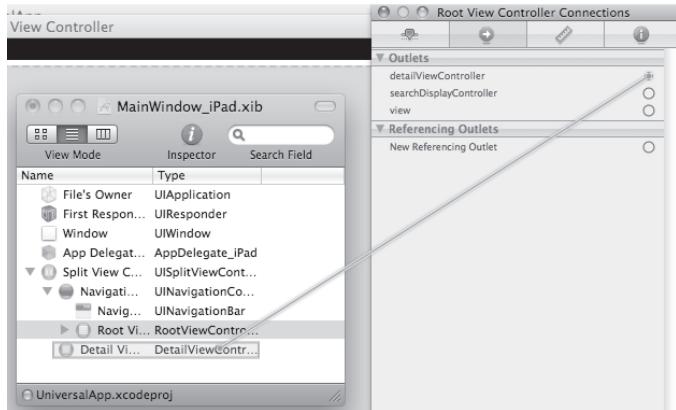


Figure 37.14 : Reliez le `detailViewController`.

Compilez, puis lancez votre application sous le simulateur iPad (voir Figure 37.15) pour observer le résultat !



Figure 37.15 : Réglez pour lancer le simulateur iPad.

Maintenant, pour que notre application fonctionne également avec l'iPhone, modifiez le fichier `AppDelegate_iPhone.h` pour déclarer un contrôleur de navigation :

```

#import <UIKit/UIKit.h>

@interface AppDelegate_iPhone : NSObject <UIApplicationDelegate> {

```

```
UIWindow *window;
UINavigationController *navigationController;
}

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

@end
```

Affichez-le dans le .m :

```
@implementation AppDelegate_iPhone

@synthesize window;
@synthesize navigationController;
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    [window addSubview:navigationController.view];
    [window makeKeyAndVisible];
    return YES;
}
```

Ouvrez ensuite le fichier MainWindow_iPhone.xib, puis ajoutez un contrôleur de navigation comme à la Figure 37.16

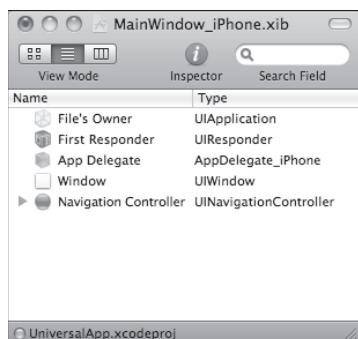


Figure 37.16 : Créez un contrôleur de navigation.

Changez la classe de son contrôleur (Figure 37.17). Ensuite, ajoutez un nouveau contrôleur de vue (Figure 37.18), modifiez sa classe pour DetailViewController (Figure 37.19) et changez le fichier nib (Figure 37.20). Enfin, reliez le detailViewController du RootViewController comme à la Figure 37.21.

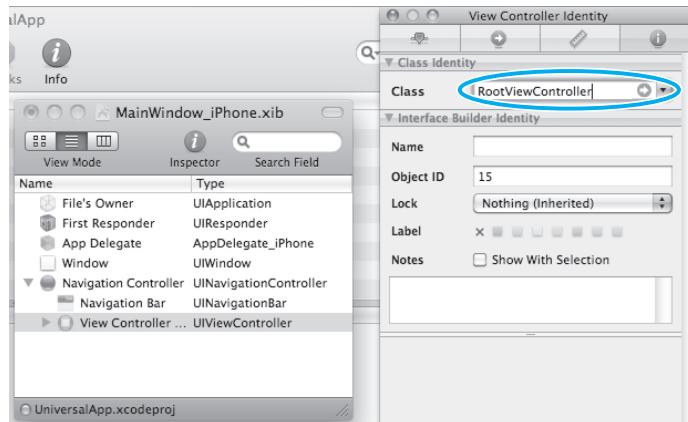


Figure 37.17 : Changez la classe pour RootViewController.

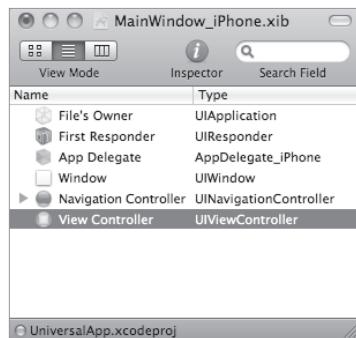


Figure 37.18 : Ajoutez un nouveau UIViewController.

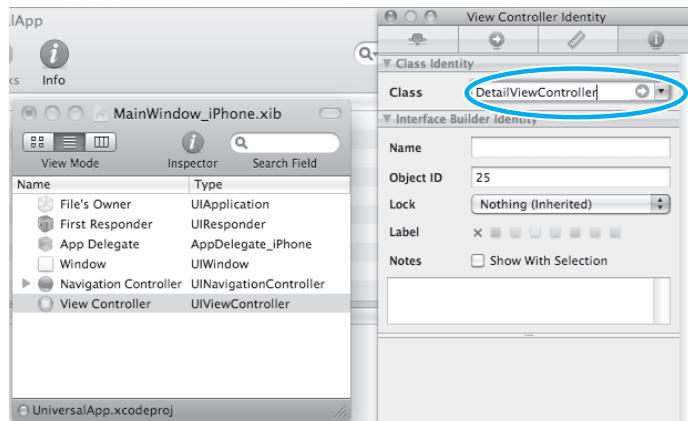


Figure 37.19 : Changez la classe pour DetailViewController.

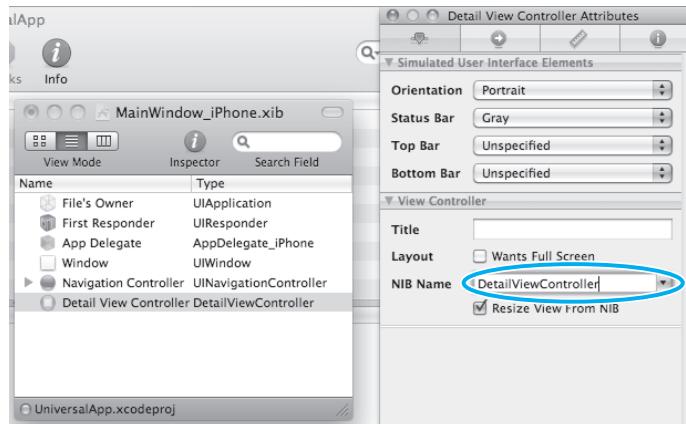


Figure 37.20 : Changez le fichier nib.

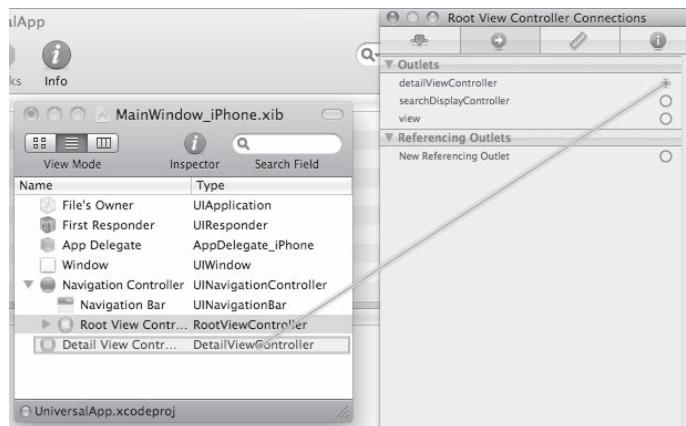


Figure 37.21 : Reliez le detailViewController.

Pour afficher le détail du légume sélectionné, il faut faire un push sur le detailViewController. Pour cela, modifiez le RootViewController.m :

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPhone) {
        // on présente l'interface pour l'iPhone
        [self.navigationController pushViewController:detailViewController
                                         animated:YES];
        detailViewController.detailLabel.text =
            [arrayToDisplay objectAtIndex:indexPath.row];
    }
}
```

```

    else
    {
        // interface iPad
        detailViewController.detailLabel.text =
            [arrayToDisplay objectAtIndex:indexPath.row];
    }
    [detailViewController.detailLabel sizeToFit];
}

```

Voilà, maintenant votre application se lance aussi bien sur iPhone que sur iPad !

Info

Pour faire tourner le simulateur, appuyez sur Cmd+flèche droite ou gauche.

UTILISER UN SPLIT VIEW CONTROLLER

Pour créer un split view controller, référez-vous à la section précédente. Vous pouvez aussi choisir de créer un nouveau projet de type Split View-based Application. Faites-le, et nommez-le PlayWithiPad. En lançant l'application, vous aurez directement un affichage de type split view. Vous remarquez qu'en mode portrait, la liste est disponible via le bouton nommé Root List. En appuyant sur ce bouton une popover s'affichera avec la liste.

CRÉER UNE POPOVER

Une popover est très pratique pour afficher des informations. Autant que faire se peut, essayez de ne pas inclure de bouton de type Done pour enlever la popover et enlevez-le après une action comme un clic sur une cellule.

L'affichage d'une popover se fait soit depuis un bouton d'une barre, soit depuis un rectangle spécifié. Pour l'afficher depuis un bouton dans une vue, utilisez cette méthode :

```

- (IBAction) displayPopoverWithList : (id) sender {
    UIButton *buttonSender = (UIButton*)sender;

    UIPopoverController *popover = [[UIPopoverController alloc]
        initWithContentViewController:listViewController];
    popover.delegate = self;

    // on fait un retain dessus pour pouvoir le réutiliser
    self.popoverController = popover;
    [popover release];

    [self.popoverController presentPopoverFromRect:buttonSender.frame inView:self.view
        permittedArrowDirections:UIPopoverArrowDirectionAny animated:YES];
}

```

Vous pouvez également choisir la direction d'affichage de la flèche de cette popover. Ici, nous laissons le choix au système avec UIPopoverArrowDirectionAny.

Pour afficher une action sheet dans une popover, comme à la Figure 37.22, il faut utiliser :

```
- (IBAction) displayPopoverWithActionSheet : (id) sender {  
    UIButton *buttonSender = (UIButton*)sender;  
  
    UIActionSheet *actionSheet = [[UIActionSheet alloc] initWithTitle:nil  
        delegate:self  
        cancelButtonTitle:nil  
        destructiveButtonTitle:nil  
        otherButtonTitles:@"Bouton 1", @"bouton 2", nil];  
  
    [actionSheet showFromRect:buttonSender.frame inView:self.view animated:YES];  
    [actionSheet release];  
}
```



Figure 37.22 : Une action sheet dans le popover !

AFFICHER UNE VUE MODALE

Vous pouvez changer la façon d'afficher une vue modale en changeant la propriété modalPresentationStyle d'un contrôleur. Vous aurez le choix entre présenter en plein écran (Figure 37.23), comme une page (Figure 37.24), ou comme une vue au-dessus de toutes les autres (Figure 37.25).

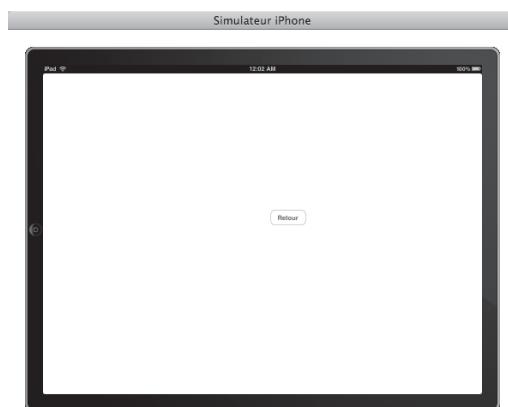


Figure 37.23 : Une vue modale en plein écran.

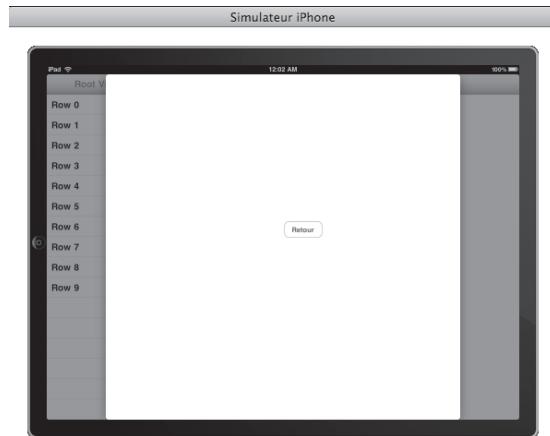


Figure 37.24 : Une vue modale affichée comme une page.

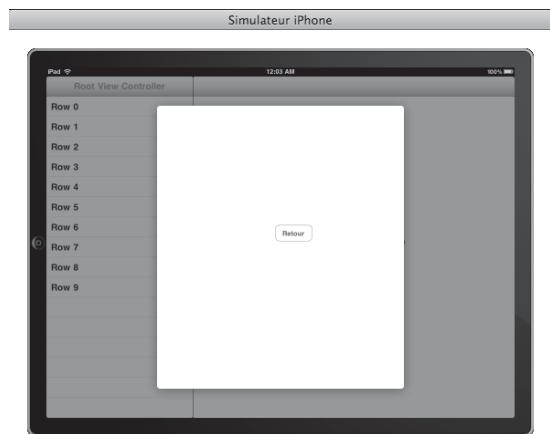


Figure 37.25 : Une vue modale au-dessus de la pile.

Voici le code pour y arriver :

```
- (void) loadModalViewController {  
    ModalViewController *controller = [[ModalViewController alloc]  
        initWithNibName:@"ModalViewController" bundle:nil];  
    self.modalVC = controller;  
    [controller release];  
}  
  
- (IBAction) displayModalViewFullScreen {  
    if (!modalVC) {  
        [self loadModalViewController];  
    }  
}
```

```
modalVC.modalPresentationStyle = UIModalPresentationFullScreen;
[self presentModalViewController:modalVC animated:YES];
}

- (IBAction) displayModalViewFormSheet {
    if (!modalVC) {
        [self loadModalViewController];
    }

    modalVC.modalPresentationStyle = UIModalPresentationFormSheet;
    [self presentModalViewController:modalVC animated:YES];
}

- (IBAction) displayModalViewPageSheet {
    if (!modalVC) {
        [self loadModalViewController];
    }

    modalVC.modalPresentationStyle = UIModalPresentationPageSheet;
    [self presentModalViewController:modalVC animated:YES];
}
```

LEXIQUE

Allocation/désallocation. L'allocation d'un objet consiste à demander au système de réserver un emplacement mémoire pour contenir les données de l'objet.

La désallocation est le processus inverse où l'on libère l'emplacement. Reportez-vous à la Fiche 32 pour vous familiariser avec l'utilisation de la mémoire.

Application delegate. L'application delegate (ou délégué de votre application) est une classe qui lance votre application et affiche la fenêtre principale

atomic/non atomic. Paramètres qui spécifient le comportement de l'objet dans le thread.

En spécifiant nonatomic pour un objet, vous précisez que votre objet sera susceptible d'être modifié par plusieurs threads en même temps (l'objet pourrait alors être corrompu en cas de mauvaise gestion de votre part).

À l'inverse, utiliser atomic permet de bloquer la ressource pour n'être utilisée que par le thread appelant. Utiliser atomic permet de conserver l'intégrité des données, mais ralentit l'accès à la ressource et est inutile si vous n'utilisez pas de threads.

Par ailleurs, vous pouvez utiliser nonatomic même en utilisant plusieurs threads lorsque votre code est écrit de manière à gérer les ressources correctement.

Voir section "Coder quelques éléments simples".

Autocomplétion. L'autocomplétion (ou remplissage automatique) est un outil intégré à Xcode qui permet de proposer la fin des mots que vous écrivez dans votre code. La gain de temps est très appréciable et permet de ne pas se tromper sur le nommage.

Bassin d'autorelease. Un bassin d'autorelease est un terme technique relatif à la gestion mémoire. Les objets placés dans ce bassin seront libérés à un instant t, au lieu d'être libérés immédiatement avec l'appel à release

Bundle/main bundle. Le main bundle est le dossier de votre application où sont placés l'exécutable et les ressources (images, fichiers...).

Dealloc. La méthode dealloc est relative à la gestion mémoire. Cette méthode est appelée lorsqu'un objet est libéré de la mémoire.

Device. Terme anglais qui signifie dispositif. Il désigne l'appareil : iPhone, iPod Touch ou iPad.

Getter/Setter. Get et Set sont respectivement traduits par récupérer et mettre. En programmation objet, le fait de setter un objet lui attribue une nouvelle valeur. À l'inverse, getter permet de récupérer sa valeur sans modifier l'objet.

Exemple : "on peut donc continuer notre initialisation en settant la propriété studentName." signifie que l'objet studentName va avoir une nouvelle valeur. Voir Fiche 32.

include. L'include est une directive pré-processeur qui permet d'indiquer au compilateur d'utiliser le fichier spécifié lors de la compilation. Elle est très peu utilisée en Objective-C et remplacée par import.

Release. La méthode release permet lors de son appel, de décrémenter le compteur de référence d'un objet. Une fois ce compteur de référence nul, l'emplacement mémoire de l'objet est libéré.

Exemple : "il faut release backgroundImage (décrémenter son retain count de 1)" signifie que vous allez diminuer (en quelque sorte) son empreinte mémoire.

Parser. Un parser, ou analyseur syntaxique, permet d'analyser un texte (en programmation, sous forme de XML ou JSON) et créer une liste de données à partir des balises trouvées.

Mode debug/release/distribution. Ce sont les trois modes de compilation utilisés par Xcode. Voir section “Introduction à Xcode”.

Provisioning. Un fichier, créé par Apple, nécessaire pour compiler sur un appareil. Ce fichier permet de signer l'application.

Push. Le Push est un service de notification mis en place par Apple qui permet à votre application de recevoir des applications d'un serveur distant à n'importe quel instant.

Il ne faut le confondre avec **pushViewController** : une méthode qui permet d'afficher un contrôleur de vue depuis un contrôleur de navigation. Typiquement, vous appellerez cette méthode lorsque l'utilisateur touchera une cellule.

Regex. Regex pour *Regular expression* est en informatique une chaîne de caractères qui décrit un ensemble de caractères possibles. Un test Regex peut par exemple permettre de vérifier la présence d'un caractère ou d'une expression dans une autre chaîne de caractères.

Exemple : “faire un regex sur le NSString” permet de vérifier que cet objet comporte bien certains caractères comme la présence de @ dans une adresse mail.

Retain count. Le retain count, pour compteur de référence, est une variable d'instance permettant de connaître le nombre de références mémoire d'un objet. Par exemple, lorsque vous faites un retain sur un objet, vous incrémentez son compteur de référence de 1. Voir Fiche 32.

Sender. Le sender est typiquement l'objet appelant d'une méthode relative à une action. Si vous affectez les actions de plusieurs boutons sur la même méthode, le sender va vous permettre de savoir quel bouton appelle la méthode. Une phrase comme “On fait une référence sur le sender, que l'on caste en UIButton”, signifie que vous pourrez utiliser le bouton à l'origine de l'appel.

Switch. Le Switch est un interrupteur (dans le cas des éléments d'interface UISwitch) et propose une interaction avec l'utilisateur en mode marche/arrêt (on/off).

Switch case. En programmation, le switch case permet de faire plusieurs tests de valeurs sur le contenu d'une même variable de type discrète (exemple : entier).

SDK. SDK pour Software Development Kit est un kit de développement logiciel. Il contient des outils qui permettent aux développeurs de programmer pour la plate-forme spécifiée.

Table view. Une table view est une liste de données couramment utilisée pour présenter des tableaux d'objet. Vous la trouverez par exemple dans l'iPod montrant la liste de vos morceaux.

Templates. Ce sont des modèles de projet qui constituent le point de départ de votre application.

Thread. Un thread est un sous-processus. Il est souvent traduit en français par *fil d'exécution*. Les threads permettent d'exécuter des fonctions en parallèle dans un même processus. Reportez-vous au document de la documentation Apple *Threading Programming Guide*.

Exemple : “animer l'Activity indicator view sur le thread principal” signifie que vous allez lancer l'animation de la roue depuis le thread principal, celui qui est utilisé par défaut.

Warnings de compilation. Dans Xcode, ils signalent en jaune une erreur de compilation non critique. Ces warnings n'empêcheront pas la compilation de votre exécutable, mais peuvent provoquer des plantages dès les premières secondes. Vérifiez-les toujours !

Index

A

ABPeoplePickerController 252
ABPeoplePickerControllerDelegate 256
Accéléromètre 112
Accessseurs 298
Achat 164
ADBannerViewDelegate 280
addSubview 195
Afficher
 carte 92
 publicité 281, 282
 tab bar 212
alert view 66
Alloc 297
Analyseur statique 305
Animation
 arrêter 229
 chemin 230
 Core Graphics 235
 couleurs 226
 courbe de vitesse 230
 enchaînement 230
 flip 21, 199
 groupe 238
 haut niveau 224
 lancer 227, 228
 répéter 227
 transparence 227
UIModalTransition 204
UIViewAnimations 225
vue 201
zoom 230

Annotations 95
API
 facebook 330
 météo 330
 twitter 330
Appareil 123
Apple ID 147
Application
 traduire 287
 universelle 376
App Store 3
 distribuer 4
 marketing 5
 vente 164
Assign 25, 64
Asynchrone
 GET 249
 méthode 250
 POST 250
 atomic 25, 64, 300
 autocorrection 40
Autorelease 297, 300
 bassin 301, 328
autoresizingMask 375

B

Barre de navigation 21, 184
Bassin d'autorelease 43, 301, 328
Blocks 117
Bluetooth 123
Boussole 100

Bouton 39
connexion/déconnexion 123
états 40
événement 50
nuage 81
playPauseButton 108
Start/Stop 220
trembler 229
UIButton 40
Build and run 22

C

Cacher, barre de statut 157

CALayer 229

Carnet d'adresses 252

contact 255

e-mail 265

liste 252

SMS 264

Carte 89

affichage 92

annotations 95

position 38

Catégories 60

Cellule

cliquer 73

personnalisée 75

Certificat

clé 149

création 10

développement 148

openssl 151

pem 151

production 148

serveur 149

CGRect 222

Clavier

rétracter 57

spécifier 40

Client 133, 134
CLLocationCoordinate2D 93, 96
CLLocationManager 100, 104, 116

Collision 220

Communiquer

appareils 123

serveur 117

sites web 188

Compilation

mode 22

simulateur 30

Compte de test 181

Compteur de référence 298

Connexion

deconnection 123

internet 125

locale 125

Contact

choisir 256, 257

propriétés 254

Contenu payant 164

Copy 25, 64, 297

Core Animation 199, 224

Core Data 348

Core Graphics 235

CoreLocation 100

Correction Voir Orthographe

@class 190

D

Dealloc 26

implémentation 298

Delegate 60, 216

méthodes 56, 85

naviguer 201

protocole 60

Délégué 63

méthode 246, 250

Documentation 14
aide contextuelle 14
autocomplétion 16
en ligne 14
forums 17
Objective-C 7

Données
archiver 343
base 310
lecture 313
désarchiver 344
échanger 128
liste 70
sauvegarder 340, 357
transmettre 127
trier 356

E

Écran
haute définition 374
iPad 374
plein 146
publicité 282
rotation 374
Effacer 162
EKCalendar 268
EKEvent 270
EKEventStore 269
EKEventViewController 277
E-mail 265
Événement
afficher 273
ajouter 274
bouton 50
contrôleur 268
delegate 60
envoyer 63
First responder 27
modifier 277

réagir 48
récupérer 268
secousse 162
supprimer 276
touchUpInside 85
EventKit 268

F

Fichier
implémentation 23
nib 22
plist 340
xib
File's owner 27
localiser 291
MainWindow 23
nib 22
view 191
File's owner 27
Filtre 114
First responder 27
Framework
CoreLocation 96, 100
json 331
Full screen 146

G

GameKit 123
Gateway 151
Geocoding 92, 94
GET 246
asynchrone 249
synchrone 248
getter 25, 64
GKPeerPickerController 123

GPS 89
adresse 92, 93
altitude 104
Map View 38
précision 103
vitesse 104
Grand Central Dispatch 117
Guide
marketing 5
Objective-C 13
programmation iPhone 13

iAd 278
activer 282
IBOutlet 25, 33, 298
id 63, 125
UDID 133

Image
afficher 157
ajouter 44
Interface Builder 46
album 109
modifier 159
par-dessus 160
sauvegarder 161
view 35
animation 36

Implémentation
fichiers 23
SQLManager 313
In App Purchase 164
Indicateur d'activité 42
Informer 147
init 297
insertSubview
atIndex 194
Interface Builder 27
Interface graphique 310

Interrupteur 53
iPad 374
iPod
bibliothèque 105
boussole 103
GPS 89
musique 105
iTunes Connect 9, 178

J

jouer
musique 105
vidéo 140
JSON 330

L

label multiligne 103
Langage
C 6
Objective-C 6
objet 6
transition
depuis C++ 7
depuis Java 7

Langue
client 287
traduire 291

Leaks 308
Licence
achat 8
App Store 4
frais 8
gratuite 8
programme
entreprise 8
standard 8

Liste
carnet d'adresses 252
circulaire 77
données 70
cliquer 73
style 71
JSON 336

Localisation
adresse 92, 93
altitude 92
GPS 89
latitude 93
longitude 93
MKUserLocation 89
précision 92, 94, 103
punaise 96

Localizable.strings 290

localizedDescription 93

MKReverseGeocoder 92

MKUserLocation 89

Mode
édition 359
paysage 376
portrait 376, 389

Mot-clé
@class 190
IBAction 52
IBOutlet 25, 33, 298
marketing 5
@optional 64
#pragma mark 66

Motion 163

Multitâche 115

Multitouch 160

Musique 105
information 109

lecteur 108

morceaux 108

MVC 348

M

MapKit 38
MapView 38

Marketing
description 5
guide 5
mot-clé 5

Mémoire 296
fuite 300
gestion 296

Méthode
appel régulier 221
asynchrone 250
classe 241
init 297
optionnelle 56, 60
requise 56, 60

MKAnnotation 95, 96

MKAnnotationView 95

MKCoordinateSpan 93

MKMapView 89, 92

N

Navigation
bare 207
contrôleur 201
pile 207
vue 201

nil 199

Nombre
ligne 72, 273
section 72

nonatomic 25, 64, 300

Notifications 115
locales 119
Push 147

NSArray 68, 194, 221

NSAutoreleasePool 301

NSCoding 122

NSData 127

NSDate 122
NSDictionary 340
NSError 93
NSIndexPath 72
NSInteger 68
NSLocale 294
NSLocalizedString 290, 291
NSLog 50
NSMutableArray 68, 317
NSMutableString 104
NSNumber 68, 96
NSObject 68, 167, 241, 313, 353
NSOperationQueue 117
NSPredicate 57
NSRunLoop 223
NSSet 272
NSSort 356
NSTimer 219
NSURL 329
NSURLConnection 247
 connectionDidFinishLoading 251
 didFailWithError 250
 didReceiveData 250
NSURLRequest 248
NSUserDefaults 343
NSValue 68
NSXMLParser 323
NSZombie 309

Objective-C 6
 multitâche 113
Objet 131
Onglet
 badge 218
 image 217
 ordre 217
 personnaliser 217
 sélectionner 214
 titre 217

OpenGL 21
Opensl 151
Ordinateur
 certificat 12
 génération 6
 simulateur 6, 303
Outil
 analyse statique 305
 Interface Builder 27
 Leaks 308
 Xcode 7, 21
@optional 64

P

Paiement 167
archivage 177
gérer 174
Parser
 JSON 323
 XML 323
Patron de conception 68, 167
 MVC 348
Photo 156
 afficher 156
 dessiner 156
 enregistrer 156
 sauvegarder 156
Pile de vue
 ajouter 194
 déplacer 194
popover 389
pop up 66
Position
 Afficher 89
 bannière 278
 carte 38
 CoreLocation 100
 Map View 38
 MKMapKit 89
 MKMapView 92
 utilisateur 89

POST 246
asynchrone 250
synchrone 249
Produit
 Consumable 179
 Non-consommable 179
 Subscription 179
Programme
 entreprise 8
 standard 8
Protocole
 ADBannerViewDelegate 280
 delegate 60
 délégué 63
 MKAnnotation 95
 utiliser 64
Provisioning
 créer 164
 gérer 12
 portail 147
Publicité 278
 afficher 281, 282
#pragma mark 66
@property 64, 190, 298, 300, 357



Quartz Core 229



Release 22, 26, 297
 auto 300
 bassin 43
reloadData 326
removeFromSuperview 198
Réseau 242

Ressources
 documentaires 13
 forums spécialisés 6, 17
 gestion 285
 localiser 294
 performance 4
 projet 23
 Retain 25, 45, 64, 297
 RootViewController 189
 @required 56, 64

S

Sandbox 153
Scroll view 81
 taille 84
Secousse, détecter 163
Sender 50
sendSynchronousRequest 248
Serveur 133
 certificat 149
 programmation 152
 store 164
 variables 241
setter 25, 64
Simulateur
 compiler 30
 GPS 89
 piège 6, 30, 303
 raccourci clavier 389
 zoomer 85
Singleton 112, 167
sizeToFit 103
SKPayment 174
SKPaymentTransactionObserver 174, 175
SKProductsRequestDelegate 173
Slider 41
SMS 264
Socket 152
Span 93

SQLite 310
 delete 321
 insert 310
sqlite3 316
Store 164
StoreKit 167
superview 198
Synchrone
 GET 248
 POST 249
@synchronized 113
@synthesize 64, 190, 298, 300, 357

T

tab bar 21, 212
Tâche de fond 116
 temps additionnel 118
Taille, paquets 128
Taps consécutifs 160
Texte
 centrer 28
 champ
 ajouter 55
 e-mail 261
 rétracer 56
 localiser 287
 sécurisé 40
 TextField 40
 Text view 39
Thread
 asynchrone 246
 atomic 25, 300
 bassin d'autorelease 43, 301, 328
 GCD 117
 multithreading 25, 113
 nonatomic 25, 300
 parser 328
 principal 42, 246
 UIKit 42

Timer 219
Token 152
Traduire
 application 287
 langues 291
Transition
 animer 199
 Objective-C 7
 vue 21

U

UIAccelerometer 112
UIActionSheet 257
UIActivityIndicatorView 42
UIButton 40, 50, 244
UIDatePicker 80, 121
UIKit 22, 33, 42
UIImagePickerController 157
UIImageView 157
UILabel 244
UIModalTransition 204
UINavigationController 311
UIPickerView 78
UIPickerViewDelegate 78
UIPopoverController 374
UISegmentedControl 209, 244
UISplitViewController 374
UITabBar 217
UITabBarController 212
UITabBarDelegate 216
UITabBarItem 217
UITableViewCell 312
UITableViewCellStyle 72, 75
UITableView 70, 181, 310, 332, 337, 379
UIView 161, 189
UIViewAnimationCurve 230
UIViewAnimations 225
UIViewController 189
UIWindow 34
URL 246

V

Vidéo 140
afficher 141
boucle 145
contrôleurs 141
écran 145
 plein 146
local 143
pause 144
play 144
streaming 140
taille 145
web 143
`viewWillAppear` 199
`viewWillDisappear` 199

Vue
 ajouter 144, 194
 animation 201
 contrôleurs 215
 déplacer 219
 échanger 192
 gestion 189, 191
 hiérarchie 201
 insérer 194
 modale 390
 navigation 201
 pile 190

X

Xcode 21
XML 323

Z

Zombies 309
Zoom
 animation 230
 simulateur 85
 valeur 93

PROGRAMMEZ POUR

iPhone, iPod touch, iPad

avec iOS 4

Le compagnon indispensable pour s'initier au développement iPhone/iPad !

37 fiches thématiques avec des **exemples concrets** de réalisation d'applications, pour appréhender les différentes fonctionnalités nécessaires à l'élaboration d'applications performantes, variées et ergonomiques pour les utilisateurs du monde entier.

Que vous soyez simple débutant ou déjà sensibilisé au développement d'applications iPhone/iPad, ce guide pratique, constitué de **tutoriels thématiques**, vous accompagnera tout au long de votre apprentissage.

- Découvrez les outils nécessaires pour démarrer et concevoir une première application élémentaire
- Manipulez des éléments simples d'interface et explorez le kit de développement iPhone (UIKit)
- Tirez parti des fonctionnalités natives comme l'accès à l'iPod, le bluetooth, l'accéléromètre, le push...
- Créez des animations, communiquez avec un site web, insérez de la publicité
- Traduisez vos applications, gérez la mémoire, sauvegardez des données

Retrouvez tous les codes sources de l'ouvrage et partagez votre expérience avec d'autres développeurs sur un espace dédié à l'échange autour de chaque projet de ce livre sur le forum communautaire de iPUP : <http://www.ipup.fr/forum>.



Niveau : Débutant / Intermédiaire

Catégorie : Développement mobile

La méthode iPUP

Après avoir écrit des tutoriels et aidé des centaines de membres à débuter, l'équipe iPUP vous propose avec ce livre de découvrir ou redécouvrir le développement iPhone/iPad à travers une méthode qui a fait ses preuves auprès de la communauté francophone.

PEARSON

Pearson Education France
47 bis rue des Vinaigriers
75010 Paris
Tél. : 01 72 74 90 00
Fax : 01 42 05 22 17
www.pearson.fr

ISBN : 978-2-7440-4168-6

