3

À la découverte du SDK de l'iPhone et de l'iPad

Au sommaire de ce chapitre

- Créer votre premier projet pour iPhone et iPad
- Maîtriser les outils de création d'interfaces
- Tester et déboguer votre premier projet

Après cette première découverte des éléments fondamentaux du développement pour l'iPhone et l'iPad, vous avez probablement très envie de passer à la phase pratique. Faites chauffer Xcode et Interface Builder, les deux outils indispensables du SDK : nous allons les parcourir sous toutes les coutures jusqu'à mettre en place notre première application iOS ! Comme vous le constaterez à travers ce premier exercice, ce sont les deux faces d'une même pièce ; vous effectuerez des allers-retours incessants entre ces deux applications de développement, en vous arrêtant régulièrement au Simulateur iOS afin de valider chaque étape. Sachez par ailleurs qu'il n'est pas nécessaire, à ce stade, de disposer d'un compte de développeur auprès d'Apple : vous pouvez parfaitement vous contenter de manipuler des objets, d'écrire du code source et de tester le résultat sous Mac OS X. Comme nous le verrons par la suite, le précieux sésame octroyé par Apple vous offre essentiellement une option supplémentaire : le dépôt immédiat de votre application compilée sur votre propre iPhone ou iPad depuis Xcode. Mais à ce stade,

il est encore inutile de manipuler votre premier projet dans des conditions réelles ; focalisez-vous essentiellement sur la découverte de l'interface de ces deux logiciels et sur les techniques générales de développement.

Au cours de ce chapitre, vous apprendrez à créer toutes sortes de projets pour iPhone et iPad sous Xcode et à gérer l'ensemble des fichiers fondamentaux qui composent une application. Grâce à Interface Builder, vous créerez la "vue" de votre application en profitant d'une bibliothèque riche de composants et vous mettrez en application tous les enseignements généraux à propos d'Objective-C et de Cocoa Touch jusqu'à manipuler votre premier projet sur le Simulateur. Vous profiterez enfin de la large bibliothèque d'applications gratuites disponibles sur le site d'Apple pour parfaire cette première approche et découvrir tous les outils de débogage intégrés à Xcode. À l'instar du système d'exploitation que se partagent l'iPhone et l'iPad, les techniques et les outils de développement sont également semblables en tout point ; nous parcourrons successivement ces deux plates-formes afin de mettre en relief les différences essentielles que vous devez retenir.

Et si le développement des applications les plus ergonomiques et fonctionnelles de l'App Store ne reposait que sur une judicieuse série de clics à travers Interface Builder ? Retroussez vos manches : nous allons le vérifier immédiatement !

Créer votre premier projet pour iOS

Au cours du premier chapitre, vous avez installé le SDK de l'iPhone et de l'iPad et vous disposez de tous les outils nécessaires au développement de votre première application. Commencez par lancer Xcode en ouvrant une fenêtre du Finder, puis en déroulant l'arborescence `Developer/Applications` et en effectuant un double-clic sur Xcode. C'est d'ailleurs dans ce répertoire que se trouvent les autres outils liés au SDK de l'iOS, en particulier Interface Builder et Instruments. En parallèle, profitez-en pour agencer les répertoires de votre disque dur et créer un dossier spécifiquement lié à votre premier projet. Son emplacement de stockage importe peu : vous aurez en revanche besoin de réunir tous les éléments constituant un même projet dans le dossier de votre choix.

Maîtriser Xcode

Xcode vous accueille par un écran de bienvenue qui vous offre un accès direct aux sections emblématiques du centre des développeurs d'Apple : création d'une application, construction de l'interface utilisateur, stockage des données, optimisation de votre code… Tous ces sujets sont traités à travers une série de vidéos, que vous avez tout intérêt à visionner si vous vous sentez perdus ! De très nombreux articles techniques issus de l'iOS Dev Center complètent également cette première approche. Pour cela, cliquez sur le lien Getting Started with Xcode. Si vous vous sentez en confiance, n'hésitez pas à décocher la case Show this window when Xcode launches pour ne plus être importuné par la suite. Cliquez alors sur le lien Create a new Xcode project.

Figure 3.1
L'écran de bienvenue de Xcode.

La liaison avec votre propre iPhone ou iPad

Si vous avez connecté votre iPhone ou votre iPad à votre Mac pendant le lancement de Xcode, vous êtes susceptible de rencontrer un message vous demandant si ce périphérique doit être utilisé dans le cadre du développement de votre application. Si vous avez souscrit au programme de développement d'Apple, reportez-vous plutôt à la procédure décrite après le paiement, sur le site des développeurs : elle détaille précisément l'activation de vos propres périphériques et vous bénéficierez ainsi d'une meilleure intégration avec les outils de développement. Pour l'heure, il est inutile de tester le résultat de notre application sur un véritable terminal : cliquez sur le bouton Ignore et fermez la fenêtre.

Déroulez le menu File et cliquez sur New Project. L'assistant de création de projet apparaît à l'écran. Comme nous l'avons vu précédemment, Xcode permet aussi bien de réaliser des applications Cocoa pour Mac OS X que pour iPhone, iPod Touch ou iPad: toutes ces plates-formes figurent ainsi dans le volet gauche de l'assistant. Cliquez sur l'intitulé Application, en dessous de la rubrique iOS, pour découvrir les sept modèles associés à ce type de développement. À l'instar des modèles de documents d'un traitement de texte par exemple, ils représentent sept projets-types préparés par Xcode. Leur utilisation vous fait ainsi gagner un temps précieux et vous bénéficiez directement de tous les éléments dont votre application a besoin. Voici à quoi ils correspondent (ils sont illustrés Figure 3.2) :

- **Navigation-Based Application.** On utilise ce modèle pour les applications s'articulant autour d'une série de listes et de contrôles de navigation, où les différentes vues s'imbriquent sous la forme d'arborescences. C'est le modèle général des options de réglage de l'iPhone ou de votre liste de contacts, par exemple. Ce type de modèle n'est disponible que pour les applications iPhone pures.
- **OpenGL ES Application.** Ce modèle prépare tous les éléments nécessaires aux applications graphiques, en particulier celles qui lancent des animations ou des effets visuels. Il se décline en deux variantes, prévues spécifiquement pour l'iPhone ou l'iPad.

- **Tab Bar Application.** Ce modèle crée rapidement une application dotée d'un système de navigation par onglets avec une vue prédéfinie pour le premier d'entre eux. Là encore, vous pouvez choisir un modèle préparé pour l'iPhone ou l'iPad.
- **Utility Application.** Comme son nom l'indique, ce modèle correspond en particulier aux utilitaires : il met automatiquement en place une vue avec un bouton "Info" permettant de configurer l'application et un système de navigation offrant un raccourci direct pour revenir à la vue principale du programme. Sans écrire la moindre ligne de code, vous disposez ainsi d'un canevas de base pour votre application : en cliquant sur ce bouton Info, un petit effet visuel à travers lequel l'affichage "se tourne" comme une feuille pour céder la place à un second écran est automatiquement ajouté à votre projet. Dans la mesure où les utilitaires constituent l'une des catégories-vedettes de l'iPhone, seul le mobile d'Apple est concerné par ce modèle.
- **View-based Application.** C'est le modèle le plus simple proposé par Xcode : il crée une vue de base sur laquelle vous déposerez tous les éléments depuis Interface Builder. Vous devez pleinement la personnaliser par la suite et vous avez donc la possibilité de choisir un modèle prévu pour l'iPhone ou l'iPad.
- **Window-based Application.** Il s'agit d'un modèle vierge, qui ne dispose que d'une instance de la fenêtre principale de l'iPhone ou de l'iPad. Privilégiez-le si vous souhaitez démarrer d'un canevas épuré, dans lequel il reste encore tout à faire ! Dans la mesure où il s'agit d'un modèle vierge, vous pouvez indifféremment choisir une version adaptée à l'iPhone, à l'iPad ou aux deux plates-formes (Universal).
- **Split View-based Application.** Spécifique à l'iPad, ce modèle démarre un projet avec une vue partagée : deux vues sont ainsi préparées et cohabitent sous forme de colonne, à l'image de l'application Mail incluse par défaut sur l'iPad. Nous reviendrons largement sur cette disposition spécifique à la tablette d'Apple par la suite ; les développeurs web de la première heure peuvent retenir que sa présentation s'apparente aux ancestrales frames du HTML 3.2.

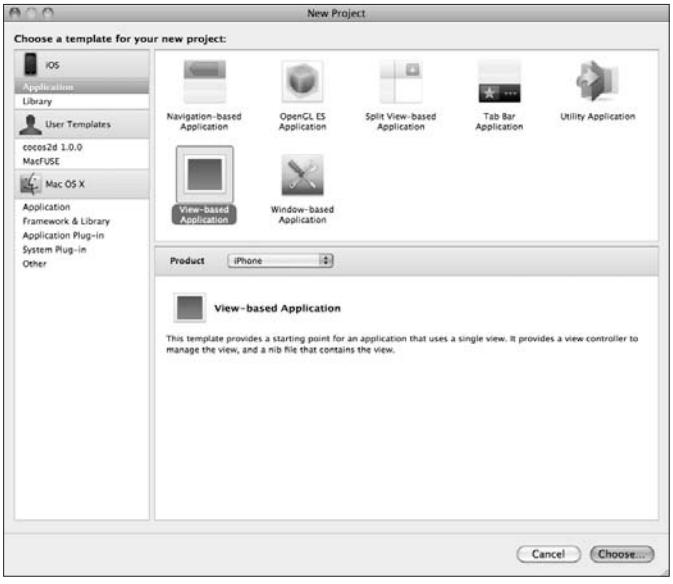


Figure 3.2.
Les différents modèles associés à la création d'un nouveau projet pour iOS sous Xcode.

Pour démarrer du bon pied, nous allons nous focaliser dans un premier temps sur une simple application iPhone, dans laquelle nous souhaitons afficher un texte à l'écran et déposer une série de contrôleurs. Cliquez sur le modèle `View-based Application`, choisissez iPhone dans le menu déroulant Product, puis cliquez sur le bouton Choose. Vous êtes ensuite invité à baptiser votre nouveau projet et à choisir un répertoire d'enregistrement. Le nom du projet a une influence majeure sur la création de tous les fichiers de base : choisissez-le avec soin et évitez surtout les doublons si vous enregistrez plusieurs projets dans le même répertoire. Dans notre exemple, nous l'appelons `Bienvenue` et nous le stockons dans le dossier `/Documents`. Cliquez ensuite sur le bouton Save pour valider votre saisie.

La fenêtre principale de Xcode apparaît à l'écran ; commencez par parcourir les fichiers automatiquement générés et associés à votre projet. Sur le volet gauche de la fenêtre figurent tous les groupes et les fichiers correspondant à votre application. Vous pouvez le confirmer en parcourant le dossier de stockage de votre projet dans le Finder : il s'agit ici d'une vue logique qui ne correspond pas à l'arborescence exacte des fichiers liés à votre application sur votre disque dur. À la manière d'une fenêtre du Finder, vous parcourez une série de groupes et de raccourcis que vous étendez à loisir et dont la racine correspond au nom de votre application – `Bienvenue` dans notre exemple. En cliquant sur l'un des fichiers, son contenu apparaît dans la fenêtre centrale de Xcode : c'est à cet emplacement que vous saisirez directement le code source de votre projet.

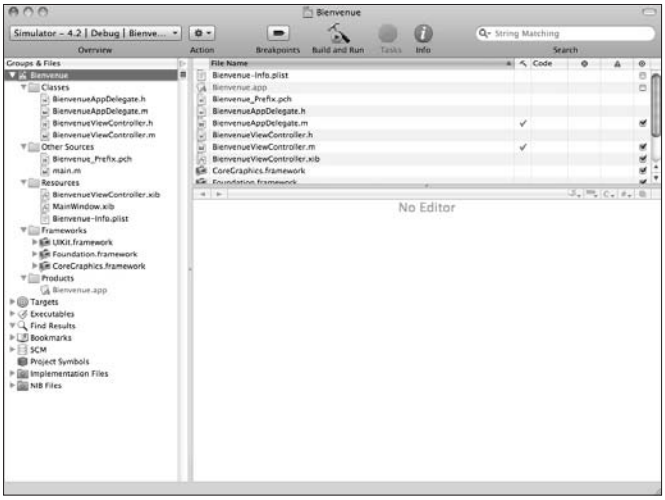


Figure 3.3
La fenêtre principale de Xcode après la création de notre premier projet.

Structure générale de tous les projets iOS

Cliquez sur le petit triangle précédant le nom de votre application, dans le volet gauche de Xcode : vous découvrez les cinq dossiers indispensables à votre projet. Passons leur rôle en revue.

- `Classes`. Il s'agit du groupe essentiel de votre application : il contient toutes les classes en Objective-C que vous avez développées. C'est d'ailleurs le seul groupe de Xcode qui correspond réellement à un dossier physique stocké sur votre disque dur, dans le répertoire associé à votre projet. Vous pouvez librement y créer des sous-dossiers afin d'agencer au mieux votre application et d'y voir plus clair au cours du développement et du débogage.

- `Other Sources`. Ce groupe réunit tous les scripts et codes sources qui ne correspondent pas à des classes en Objective-C. Par défaut, Xcode crée automatiquement deux fichiers indispensables :
  - `Bienvenue_Prefix.pch`. L'extension de ce fichier désigne un "en-tête précompilé" (precompiled-header, `.pch`), soit la liste exhaustive de toutes les bibliothèques et des frameworks sur lesquels s'appuie votre projet. En précompilant ces éléments immuables constituant votre application, Xcode vous fait gagner un temps précieux au moment de la compilation de votre projet : il ne se préoccupera plus que des classes que vous avez créées vous-mêmes. Par défaut, cet en-tête importe les deux frameworks essentiels du développement sous Cocoa Touch : *Foundation* et *UIKit*.
  - `main.m`. N'oubliez jamais qu'Objective-C est une surcouche du langage C ; ce fichier définit ainsi la méthode principale `main()` de votre application. Là encore, vous n'avez pas à intervenir sur ce fichier à moins de modifier les arguments de cette méthode. Nous ne nous y attèlerons que dans les projets de plus grande envergure, qui nécessitent un traitement particulier.
- `Resources`. Ce groupe contient tous les éléments indispensables à votre application qui ne correspondent pas à du code source. Vous y grefferez notamment les images, sons, vidéos, textes et fichiers de configuration associés à votre projet, à commencer par l'icône qui lui est propre. Là encore, vous pouvez librement agencer ce groupe comme bon vous semble et créer des sous-dossiers associés à chaque type de fichiers. D'ailleurs, si vous déplacez l'un de ces éléments dans la fenêtre de Xcode, le fichier correspondant stocké sur votre disque dur n'est pas modifié pour autant : Xcode dispose de son propre système de classement interne et vous évite de vous préoccuper de l'emplacement de stockage de toutes vos ressources. Trois éléments indispensables y figurent d'ores et déjà :
  - `BienvenueViewController.xib`. Il s'agit du fichier définissant l'interface graphique de votre application (sa "vue"), composé par Interface Builder. Il n'est pas possible de l'éditer manuellement dans Xcode mais vous pouvez en revanche l'ouvrir dans un éditeur de texte, comme TextEdit : vous découvrez alors qu'il s'agit d'un fichier XML décrivant l'interface à travers une série de balises propriétaires. Nous y reviendrons largement dans la suite de ce chapitre.
  - `MainWindow.xib`. Dans le même ordre d'idée, ce fichier décrit la vue de la fenêtre principale de votre application. Là encore, il s'agit d'un fichier lié à Interface Builder, que vous modifierez directement à travers cet utilitaire.
  - `Bienvenue-Info.plist`. Cette extension désigne une liste de propriétés (*Property List*, `.plist`) associée à votre projet. Vous pouvez la modifier manuellement à travers Xcode : elle regroupe tous les paramètres généraux de votre application, comme l'emplacement de son icône, son numéro de version ou sa langue par défaut. Nous personnaliserons ce fichier au cours de ce chapitre.
- `Frameworks`. Ce dossier regroupe tous les frameworks sur lesquels s'appuie votre application. Par défaut, trois frameworks sont chargés : UIKit, Foundation et CoreGraphics. Envisagez-les comme de grandes bibliothèques regroupant des éléments spécifiques à certaines facettes du développement (le traitement des effets visuels, les interactions avec l'utilisateur, etc.). Les grands studios créent généralement leurs propres frameworks afin d'optimiser et d'accélérer le développement de tous leurs projets, en reprenant régulièrement

des éléments déjà créés auparavant sans avoir à réinventer la roue. Comme nous le verrons par la suite, vous avez également la possibilité d'importer des frameworks libres de droits glanés sur le Web, qui seront ainsi liés à votre application. Dans tous les cas, cette architecture permet de n'importer que les frameworks dont votre application a besoin et ainsi de limiter les ressources qu'elle consomme.

- `Products`. Le dernier groupe réunit tous les fichiers créés au moment de la compilation, en particulier `Bienvenue.app` qui correspond à votre application compilée. Comme vous pouvez vous en rendre compte, ce fichier apparaît en rouge dans l'interface de Xcode ; c'est le signe qu'il manque encore à l'appel. Rien de plus normal : nous n'avons pas compilé le moindre code pour l'instant et Xcode s'est contenté de préparer le terrain

Comme nous l'avons vu, les fichiers indispensables à votre projet ont été créés automatiquement en reprenant son nom pour les identifier clairement. C'est là tout l'intérêt d'un tel environnement de développement, articulé autour du concept de modèles : vous vous épargnez les tâches rébarbatives et vous entamez directement la création de votre projet. Par ailleurs, ce système de classement propre à Xcode témoigne de la puissance d'une architecture où les vues, les contrôleurs et les modèles sont séparés. Vous pouvez ainsi vous préoccuper de la programmation de votre application tout en important une interface que vous avez déjà préparée, ou au contraire vous consacrer uniquement au design de votre application. Les premiers contours du développement d'une application universelle, ou d'une déclinaison pour l'iPad d'un utilitaire pour l'iPhone, s'esquissent : sans réinventer nécessairement la logique interne de votre application, vous serez ainsi en mesure de charger d'autres interfaces ou de pointer vers des classes spécifiques à chaque plate-forme. C'est l'une des forces du kit de développement pour iOS ; vous déclinez rapidement votre savoir-faire et votre travail à l'une ou l'autre de ces plates-formes. Nous allons d'ailleurs précisément nous en assurer : enfilez votre blouse de designer, nous allons réaliser l'interface de notre premier projet.

Maîtriser Interface Builder

Dans le volet gauche de Xcode, déroulez l'arborescence en face du groupe `Resources` et effectuez un double-clic sur le fichier `BienvenueViewController.xib`. Comme nous l'avons vu, il s'agit de la description de l'interface de votre application à travers Interface Builder ; le logiciel s'ouvre d'ailleurs automatiquement en arrière-plan.

Des fichiers nib aux fichiers xib : l'évolution du format d'Interface Builder

Nous l'avons vu au Chapitre 2 : la plupart des outils de développement de Mac OS X, en particulier l'environnement Cocoa, ont été mis en place à l'origine par les ingénieurs de NeXT. Dirigée par Steve Jobs avant son rachat par Apple, cette société a ainsi défini de nombreux standards qui ont persisté à travers les années. Parmi les produits phares de la firme, Interface Builder a été mis en place en 1988. Il a servi d'environnement de prédilection pour développer des applications pour des générations de système, dont NextSTEP, OpenSTEP, Mac OS X et aujourd'hui iOS.

*Ainsi, les éléments décrivant l'interface graphique d'une application Mac OS étaient stockés à l'origine dans un fichier `.nib` pour "NeXT Interface Builder". Dans la version la plus récente d'Interface Builder, cette extension a été revue et corrigée et on utilise désormais des fichiers `.xib` (*XML Interface Builder*). Sans entrer dans des détails*

trop techniques, sachez toutefois que ce nouveau format ne supplante pas totalement son auguste prédécesseur : il s'agit essentiellement d'un standard plus clair et plus facile à lire, qui est finalement recompilé en fichier `.nib` *lors de l'empaquetage de votre application. Mais sachez que par coutume et habitude, la plupart des développeurs continuent de parler de "fichiers nib" pour évoquer les éléments d'interfaces du SDK de l'iOS. Même la documentation officielle n'a pas envisagé de corriger ce réflexe et préserve l'appellation d'origine !*

Au lancement d'Interface Builder, trois fenêtres prennent place sur votre écran : `BienvenueViewController`, le panneau `Library` et le canevas `View` (voir Figure 3.4).

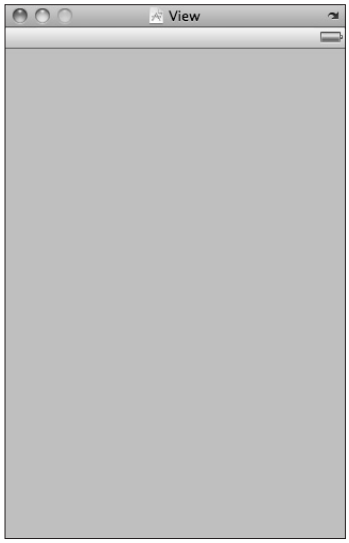


Figure 3.4.
L'interface `View` *de notre projet, à l'ouverture d'Interface Builder.*

La première fenêtre dresse l'inventaire de tous les éléments présents à l'affichage : chaque icône correspond ainsi à l'instance d'une classe Objective-C. Comme nous le verrons par la suite, toutes ces instances sont triées et laissent ainsi apparaître les dépendances entre les sous-classes. Vous manipulez cette fenêtre à la manière du Finder et vous "déroulez" l'arborescence d'une classe afin de découvrir les éventuelles sous-classes qu'elle héberge. La fenêtre `Library` contient précisément tous les éléments d'interfaces disponibles dans le SDK de l'iOS. Par un simple jeu de glisser-déposer vers la fenêtre `View`, vous créez ainsi des instances de chaque objet correspondant.

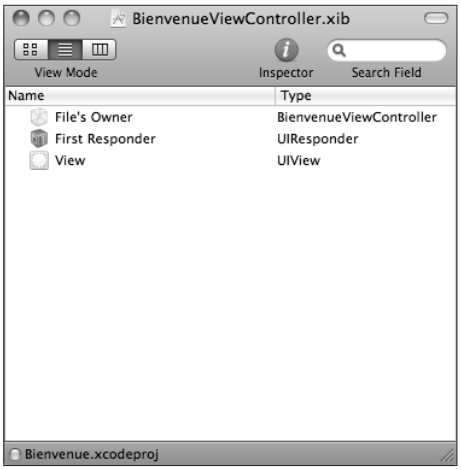
Par défaut, vous découvrez pour le moment trois icônes dans la fenêtre `BienvenueViewController : File's Owner`, `First Responder` et `View`. Les deux premiers éléments sont indispensables et ne peuvent d'ailleurs pas être supprimés (voir Figure 3.5).

Comme son nom l'indique, `File's Owner` correspond au "propriétaire" du fichier nib : en modifiant ses paramètres, vous pouvez ainsi lier plusieurs classes ou sous-classes entre elles.

Nous aurons tout le loisir de parcourir plus en détails ces notions de chargement et de liaisons de classes autour d'un exemple concret. Retenez pour l'instant que ce paramètre essentiel associe un fichier nib spécifique à l'instance d'une classe ou sous-classe. La sous-classe

`BienvenueViewController` a été créée automatiquement par Xcode en chargeant ce premier *template* ; son instance recherche immédiatement un fichier `BienvenueViewController.xib`, le charge en mémoire puis en devient naturellement le propriétaire. C'est précisément ce type d'associations que vous pourrez modifier par la suite, à travers l'icône `File's Owner`.

Figure 3.5
La fenêtre `BienvenueViewController` détaille le contenu du fichier `nib`.



L'icône `First Responder` correspond à l'objet d'interface avec lequel l'utilisateur interagit. Ce "répondeur" évolue sans cesse au cours de ses manipulations : il peut donc s'agir alternativement d'un champ de texte, d'un bouton, d'un onglet et ainsi de suite. L'intérêt d'un tel système est d'offrir un lien direct au programmeur pour communiquer avec cet objet et en récupérer les données, sans prévoir de multiples méthodes "surveillant" chaque contrôleur ou vue d'une application. Là encore, nous aurons largement l'occasion de détailler ce concept autour d'exemples concrets, dès le chapitre suivant.

Au-delà de ces deux icônes indispensables, notre interface ne comprend pour l'instant qu'une seule instance d'objet : `View`. Plus précisément, il s'agit d'une instance de la classe `UIView`, qui définit de manière globale une zone d'affichage à l'écran, avec laquelle l'utilisateur interagit. Vous comprenez ainsi mieux le nom du modèle de base de Xcode pour iOS (`View-based Application`). Tous les éléments d'interface, comme les boutons, les libellés, les réglettes ou les images correspondent à des instances de sous-classes d'`UIView`. Comme nous le verrons par la suite, il est naturellement possible de définir plusieurs vues au sein d'un même projet ; on parle alors "d'application multivues". C'est d'ailleurs le cas de l'immense majorité des applications pour iPhone, où l'on bascule entre plusieurs vues préparées avec Interface Builder, au gré des interactions. Un système de navigation par onglets permet précisément de passer d'une vue à l'autre, par exemple. Pour l'iPad, cette cohabitation entre une série de vues est encore plus explicite, comme nous le confirme le modèle `split-view` dont le principe est précisément de juxtaposer deux vues.

Ajouter des objets avec Interface Builder

Tout l'intérêt d'Interface Builder réside dans son mode de création intuitif et cohérent : vous visualisez simplement les instances de vos classes et vous manipulez une bibliothèque riche

d'objets. Nous l'avons vu : tous les éléments d'interface sont des instances de sous-classes d'`UIView` ; si vous avez naturellement la possibilité d'instancier de tels objets par des lignes de code, sous Xcode, il devient vite pénible de personnaliser leurs moindres attributs de cette manière. Imaginez un simple bouton à disposer sur la vue de votre application. En une ligne d'Objective-C, vous pouvez créer une instance de la classe `UIButton`, elle-même sous-classe d'`UIView`. Mais comment parviendrez-vous à placer précisément le bouton sur l'interface, au pixel près, et à régler ses dimensions à travers le code source, sans multiplier les essais successifs ?

La fenêtre `View` d'Interface Builder résout ce problème : elle correspond à la représentation graphique de l'instance `View` apparaissant dans le fichier `nib` (voir Figure 3.6). Par de simples jeux de glisser-déposer depuis la bibliothèque, vous créez ainsi des instances de tous les objets Cocoa Touch supportés par Interface Builder avant de les redimensionner ou de modifier leurs propriétés.

Figure 3.6
La fenêtre `View` correspond à la représentation visuelle de l'instance `View`. Vous y définissez directement l'interface de votre projet.

Plus précisément, les objets figurant dans la bibliothèque proviennent tous du framework *UIKit*, qui est associé à votre projet dès sa création (voir Figure 3.7). Rappelez-vous : ce framework est mentionné dans le volet gauche de Xcode, après la création de notre projet.

Mieux encore : Interface Builder gère de manière naturelle les imbrications et héritages de classes et d'objets (voir Figure 3.8). Dans de nombreux cas de figure, il est en effet impossible de créer l'instance d'une sous-classe si vous n'avez pas instancié la classe mère au préalable. Par exemple, vous ne pouvez pas créer un onglet spécifique au sein de votre application sans avoir déposé une "barre d'onglets" auparavant, prête à l'héberger. Puisque `View` correspond à la vue de notre interface utilisateur, les objets sous-jacents s'appellent des "sous-vues". À l'inverse de notre exemple, certains objets ne tolèrent pas de sous-vues. Dans tous les cas, il est donc impossible de commettre de telles erreurs avec Interface Builder ; l'application empêche tout simplement de glisser-déposer certains objets si l'opération est interdite. Interface Builder supervise automatiquement l'héritage entre les instances de classes et d'objets.

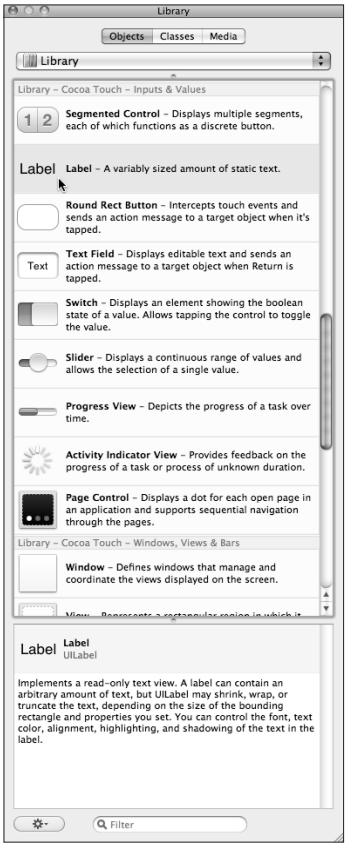


Figure 3.7
La bibliothèque d'objets du framework UIKit d'iOS.

Figure 3.8.
Tous les écrans de l'iPhone sont constitués d'éléments accessibles dans la bibliothèque d'objets d'Interface Builder, à l'image des Réglages généraux, qui réunissent des listes, des libellés et des icônes.

Pour comprendre le fonctionnement général d'Interface Builder, déposons un premier objet sur la fenêtre `View`. Parcourez la bibliothèque figurant sur la droite de l'écran et recherchez l'objet

`Label`. Il s'agit d'un simple "libellé", soit un texte statique apparaissant sur l'écran de l'iPhone ou de l'iPad et n'appelant pas la moindre interaction avec l'utilisateur (voir Figure 3.9).

Figure 3.9
On dépose directement notre libellé sur la vue principale de notre application, à travers Interface Builder.

Sélectionnez cet objet, maintenez le bouton gauche de la souris enfoncé et relâchez-le au-dessus de la fenêtre `View` : un simple champ encadré de poignées apparaît sur l'interface utilisateur. Effectuez un double-clic sur l'objet pour modifier le texte apparaissant par défaut et redimensionnez l'élément ou déplacez-le comme bon vous semble. Pour résumer, nous avons ainsi créé en quelques secondes une instance de la classe `UILabel` ; elle devient une sous-vue de `View` et vous apercevez son icône dans la fenêtre `BienvenueViewController.xib`, en déroulant l'arborescence `View`. Bravo : votre premier projet prend forme !

Tout l'intérêt de ce type de design consiste à simplifier au maximum les étapes nécessaires. Vous bénéficiez ainsi d'un éditeur wysiwyg[1] et vous visualisez directement l'interface utilisateur, telle qu'elle apparaîtra sur l'iPhone ou l'iPad de vos "clients".

Pour vous en persuader, déroulez le menu Tools et cliquez sur Attributes Inspector, après avoir sélectionné le libellé. Dans la fenêtre qui apparaît, vous reconnaissez les options traditionnelles d'un éditeur de texte (voir Figure 3.10).

Vous pouvez ainsi changer la couleur et la taille du texte, son alignement, la portée de son ombre ou son opacité par exemple. Multipliez les essais afin d'aboutir au résultat escompté. Vous pouvez invoquer l'outil Inspector pour chaque objet figurant sur l'interface utilisateur, y compris l'instance `View` en elle-même (voir Figure 3.11). Dans ce cas, vous avez essentiellement la possibilité de modifier la couleur d'arrière-plan.

1. Acronyme de *what you see is what you get* : ce que vous voyez est ce que vous obtiendrez.

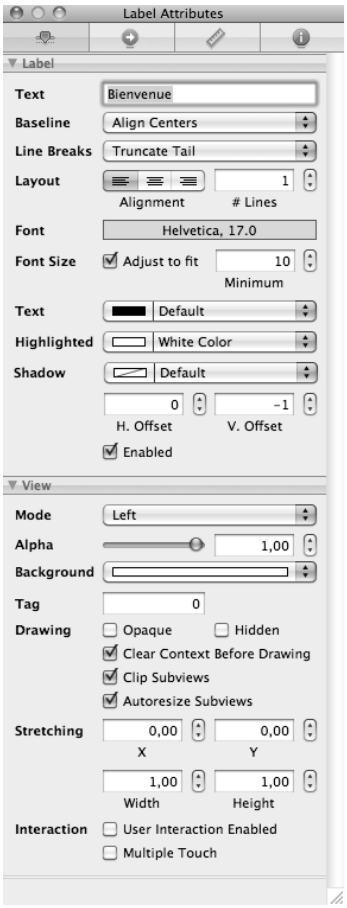


Figure 3.10
On définit simplement les propriétés de notre libellé, à l'aide de l'outil Inspector d'Interface Builder.

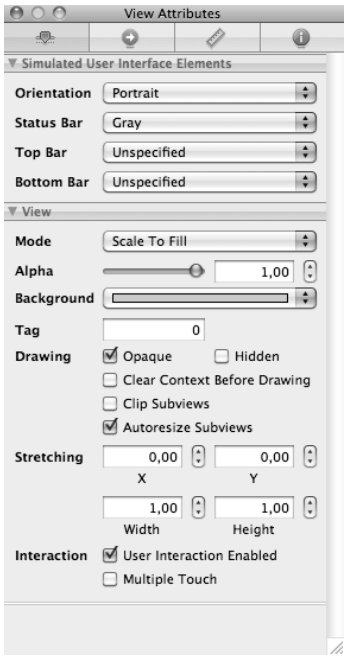


Figure 3.11
L'outil Inspector prend en charge tous les objets de l'interface utilisateur, même la vue principale.

Ce premier essai vous a convaincu de la puissance d'Interface Builder ? Parcourez tous les objets du framework *UIKit* figurant dans la bibliothèque et essayez de les déposer sur la vue principale de votre interface. Vous vérifiez ainsi la faisabilité de votre projet, tel que vous l'aviez esquissé sur papier au Chapitre 1. Selon toutes vraisemblances, de très nombreuses idées nouvelles jailliront spontanément au cours de l'expérience.

Les objets de la bibliothèque d'Interface Builder

On distingue essentiellement trois catégories d'objets UIKit, triées dans la bibliothèque d'Interface Builder : les conteneurs de données (*Data Views*), les objets de saisies (*Inputs & Values*) et les fenêtres et systèmes de navigation (*Windows, Views & Bars*). Voici les principales classes correspondantes :

Les conteneurs de données. Tableaux puisant leur contenu depuis une base de données, images, intégration du navigateur web ou vues défilant automatiquement : tous ces objets servent à agencer au mieux les données de votre application.

- `Table View`. La vue `Table` met en place un tableau visant à accueillir des informations, à la manière du carnet de contacts de l'iPhone par exemple ou de la liste des derniers messages de l'application Mail de l'iPad.
- `Table View Cell`. Après avoir créé un tableau, vous ajoutez des cellules individuelles afin de soutenir les données.
- `Image View`. Cet objet sert à charger des images, dans des formats standard. Si l'iPhone et l'iPad supportent de multiples formats, notamment le JPEG, privilégiez dans la mesure du possible les images au format PNG : Xcode parvient à les compresser au maximum au moment de la compilation.
- `Scroll View`. Cette vue crée un large conteneur qui supporte la navigation "au doigt" si son contenu dépasse la hauteur d'un écran. C'est notamment le cas des conditions générales auxquelles vous accédez dans les réglages de votre iPhone : le texte s'étend sur des centaines de lignes, que vous parcourez librement en mimant le comportement d'un ascenseur au doigt.
- `Web View`. Cette vue intègre un navigateur web au sein de votre interface utilisateur : il supporte même les boutons Précédent et Suivant.
- `Map View`. Cet objet ajoute intégralement une carte de type Google Maps à votre projet : l'utilisateur la manipule du bout des doigts et vous pouvez même choisir d'afficher une vue satellite ou une carte (ou les deux, à l'instar du service de cartographie de Google).
- `Text View`. Cette vue permet d'afficher un simple champ de texte multilignes.
- `Picker View`. Cet objet crée une "roulette" que vous complétez par des données qui vous sont propres. C'est notamment le cas de l'horloge ou du réveil de l'iPhone.
- `Date Picker`. Cet objet reprend la vue précédente en l'initialisant avec des dates et des calendriers. Idéal pour mettre rapidement en place une application d'agenda.
- `Ad BannerView`. Si vous envisagez d'ajouter des bannières publicitaires à votre application iOS, cet objet réserve une portion de l'écran à l'affichage de telles réclames interactives (par défaut, 50 pixels de hauteur). Son véritable intérêt réside dans la préparation de tous les mécanismes d'interaction : ainsi, si un message publicitaire prévoit l'affichage d'un site web ou d'une vidéo, l'iPhone ou l'iPad déclenchent les applications concernées, comme Safari par exemple, tout en notifiant votre propre application de la bascule. Nous revenons en détails sur ce système de revenus publicitaires au Chapitre 12.

La saisie de valeurs. Que vaut une application pour iPhone ou iPad si vous ne pouvez pas interagir avec elle ? Sur les deux plates-formes d'Apple, la saisie de données revêt des formes très diverses, du simple champ de texte faisant apparaître un clavier virtuel à l'originale "roulette" où des valeurs défilent et se sélectionnent du bout des doigts. En créant l'instance de telles classes, vous définissez des objets qui renvoient des données à l'application ; là encore, Interface Builder vous fait gagner un temps précieux en facilitant la configuration de tels mécanismes.

- `Segmented Controls`. Il s'agit d'un simple bouton permettant de basculer entre deux vues. Chaque "côté" du bouton peut contenir du texte ou des images, afin d'indiquer à l'utilisateur les options à sa disposition. En revanche, il n'est pas possible de superposer les deux types de données.
- `Label`. Cet objet est un simple champ de texte statique, généralement très court. Nous l'avons déjà utilisé dans notre premier projet.
- `Round Rect Button`. Cet objet crée un bouton rectangulaire que l'utilisateur peut presser au doigt.
- `TextField`. Cet objet correspond à un champ de texte saisissable par l'utilisateur : après l'avoir sélectionné, un clavier virtuel apparaît à l'écran.
- `Switch`. Cet objet crée un interrupteur visant à activer ou désactiver une option. On en retrouve de nombreuses occurrences dans les réglages de l'iPhone ou de la plupart des applications.
- `Slider`. Il s'agit d'une "réglette" permettant à l'utilisateur de choisir une valeur à travers une barre horizontale, qu'il manipule au doigt. Attention toutefois : n'oubliez pas que l'utilisateur ne peut pas forcément choisir une valeur très précise par son biais. En revanche, c'est un outil idéal pour moduler le niveau d'un paramètre.
- `Progress View`. Cet objet crée un indicateur de progression : il affiche l'état d'avancement d'une opération en cours, comme le chargement ou l'enregistrement de paramètres par exemple.
- `Activity Indicator View`. Dans le même ordre d'idée, cet "indicateur d'activité" renvoie à l'utilisateur un signal lui indiquant qu'une opération est en cours.
- `Page Control`. Cet objet ajoute une série de "points blancs", en bas de l'écran, permettant de basculer d'une "page" à l'autre. C'est l'élément qui figure en bas de l'écran de l'iPhone ou de l'iPad et qui vous permet de naviguer parmi les applications que vous avez installées.

Les systèmes de navigation. Sobre et épuré, l'écran de l'iPhone ou de l'iPad n'est pourtant jamais dépourvu d'un système de navigation, à commencer par la barre d'informations en haut de l'affichage, qui égrène des détails sur votre opérateur téléphonique, le niveau de batterie ou la puissance du signal WiFi. Barres d'outils, onglets et menus : vous avez la possibilité de reproduire un tel système pour soutenir votre application.

- `Window`. Il s'agit de l'élément de plus grand ensemble parmi les classes d'*UIKit*. Une telle "fenêtre" regroupe plusieurs vues et votre application peut elle-même compter plusieurs fenêtres.
- `View`. C'est l'élément créé par défaut à l'aide de notre *template* : vous déposez des objets sur une vue, qui deviennent alors des sous-vues. Les applications "multivues" basculent ainsi entre plusieurs vues préparées à l'avance sous Interface Builder.
- `Search Bar`. Spécifique à certaines applications, cette "barre de recherche" apparaît en haut à droite et permet à l'utilisateur d'exécuter des requêtes en faisant apparaître un clavier virtuel.
- `Navigation Bar`. Signifiant littéralement "barre de navigation", cet objet soutient plusieurs sous-éléments qui décrivent ainsi le système de navigation de votre application.