14

Rentabiliser son application grâce aux achats In App

Le SDK d'iOS a permis à de nombreux développeurs de gagner leur vie de manière indépendante, en faisant éclore d'innombrables *business models*. Le premier modèle est le plus simple à comprendre (et aussi l'un des plus rentables) : développer une application et la commercialiser à travers l'App Store. Une autre solution consiste à sortir deux versions d'une même application – une version gratuite expurgée de quelques fonctions et une version professionnelle (payante). Un autre modèle, probablement introduit par le Web 2.0, consiste à multiplier les annonces publicitaires. Les développeurs utilisent alors le SDK d'une régie publicitaire (ou iAds) pour afficher des publicités, en recevant une rétribution en fonction de leur nombre d'affichages ou de clics. Même si tous ces modèles complémentaires participent à élever les revenus du développeur, la vente directe d'applications sur l'App Store reste la solution la plus efficace. Les achats In App constituent en parallèle une chance supplémentaire de revendre des options ou des fonctionnalités supplémentaires à toutes vos applications iOS. Certaines applications embrassent d'ailleurs ce modèle au point de ne générer des profits qu'à travers les achats In App. On parle alors d'applications *freemium*, une technique de plus en plus en vogue et dont certains exemples figurent parmi les meilleures ventes de l'App Store.

Au cours de ce chapitre, vous découvrirez en détail le framework des achats In App d'Apple, `StoreKit.framework`, et apprendrez à utiliser un framework d'encapsulation, MKStoreKit. Vous découvrirez comment l'employer afin de déployer un mini-App Store au sein de votre application. Ce chapitre détaille également des solutions face aux problèmes les plus courants relatifs à l'intégration de StoreKit au sein d'une application.

Avant de commencer

Ce chapitre se divise en deux sections principales. La première moitié s'attarde sur la création et la personnalisation de produits à travers iTunes Connect et la seconde partie traite des aspects de programmation. Vous utiliserez MKStoreKit et parcourrez l'ensemble de ses fonctionnalités, en apprenant notamment à le personnaliser afin

qu'il corresponde à votre business model. À la fin de ce chapitre, nous étudierons les principaux problèmes auxquels les développeurs doivent faire face et les solutions à leur disposition. Veillez à suivre minutieusement les recommandations figurant dans les encadrés, qui vous délivrent de précieux conseils. Vous découvrirez notamment les risques que vous encourez en cas d'erreur. Commençons sans plus tarder notre tour d'horizon.

Les produits associés aux achats In App

Les produits susceptibles d'être commercialisés à travers les achats In App peuvent être classifiés en quatre catégories : le contenu, les fonctionnalités, les services et les abonnements. La dernière version du SDK les supporte toutes les quatre. Apple a introduit quatre types différents de produits, identifiés sous leur terminologie anglaise : *consumable* (consommable), *non-consumable* (non consommable), *auto-renewable subscriptions* (abonnements à renouvellement automatique) et *non-renewable subscriptions* (abonnements non renouvelables). De manière globale, un *consommable* est un produit qui s'épuise lorsqu'on l'utilise. À l'inverse, un élément *non consommable* correspond à un produit disponible de manière illimitée après son achat. Votre imprimante, par exemple, est un élément non consommable (ou "impérissable") alors que son encre est un consommable. Les abonnements, qu'ils soient automatiquement renouvelables ou non, correspondent au modèle des magazines papier. Vous pouvez commercialiser l'un des produits suivants, en fonction de votre business model :

- **Fonctionnalité.** Les produits considérés comme des "fonctionnalités" correspondent souvent à des possibilités verrouillées par défaut. Par exemple, un gestionnaire des tâches peut autoriser ses utilisateurs à consigner `n` opérations au maximum ; pour créer d'autres tâches par la suite, il doit débloquer cette fonction en s'acquittant de frais supplémentaires. Une fonctionnalité est quasiment toujours considérée comme non consommable.

- **Service.** Les produits considérés comme des "services" correspondent généralement à des fonctionnalités qui offrent une rétribution régulière au développeur. Les services s'apparentent ainsi à la catégorie des fonctionnalités, à ceci près qu'ils impliquent souvent de lourds calculs effectués sur un serveur distant. Les notifications push en constituent un exemple classique. Un client Twitter, par exemple, peut vendre un service de notifications sous forme de produits consommables, comme 1 000 notifications pour un euro.

- Les services peuvent également s'appuyer sur un système d'abonnement, si votre business model l'implique.

- **Abonnement.** Les produits considérés comme des "abonnements" correspondent essentiellement à du contenu ou à des services renouvelés sur une base régulière. Pour reprendre notre exemple précédent, un client Twitter peut offrir des notifications push illimitées pour un abonnement de 0,99 euro par trimestre.

- Le responsable d'une application décide lui-même de traiter son produit comme périssable, non consommable ou accessible *via* un système d'abonnement.

INFO

Les abonnements sont apparus avec iOS 3.0 mais se sont révélés très complexes d'utilisation, en laissant au développeur le soin de gérer leur renouvellement et/ou leur déploiement sur les autres appareils de l'utilisateur. Pour cette raison, ils ont été peu adoptés à leurs débuts. Avec iOS 4.3, Apple a introduit un nouveau type de produit baptisé *auto-renewable subscription* (abonnement autorenouvelable), dont il gère automatiquement le renouvellement. Aujourd'hui, vous avez tout intérêt à utiliser ce genre de produit. N'employez l'ancien type d'abonnement que si votre business dispose déjà de son propre système de traitement des abonnements, largement adopté par vos clients.

Éléments interdits

Nous venons de parcourir la liste des produits que vous êtes susceptible de vendre à travers les achats In App. Si Apple encourage la plupart des business models, on distingue toutefois une série de produits que vous ne pouvez pas commercialiser à travers ce système, à l'heure où nous mettons sous presse.

Le premier type d'élément (et probablement le plus important d'entre eux) que vous n'avez pas le droit de vendre à travers les achats In App concerne les biens ou les services physiques. Par exemple, si vous développez une application proposant des thèmes d'arrière-plan, vous avez le droit de vendre d'autres fichiers numériques mais pas des impressions papier des mêmes thèmes, à travers les achats In App. Dans le même ordre d'idée, si vous gérez un hôtel et que vous développiez une application pour gérer les réservations, vous n'avez pas le droit de percevoir des frais de réservation à travers ce système.

Le second type d'élément interdit concerne les échanges monétaires intermédiaires. Si vous créez une application qui propose des morceaux de musique, vous devez les vendre directement. Vous n'avez pas le droit d'inventer un système de "points" que l'utilisateur échange contre de la musique. Les cartes d'abonnement, les coupons d'achat ou tout autre système qui n'offre pas directement un produit au moment de la transaction sont interdits.

ATTENTION

Il est possible que certaines applications pratiquent ce type de techniques actuellement sur l'App Store. Vous ne devez pas les imiter. Vous courez le risque de vous faire rejeter par Apple. Si vous découvrez une application qui commercialise un élément interdit, il y a de grandes chances pour qu'elle soit passée entre les mailles du filet jusqu'à présent. Elle risque toutefois de se faire rejeter au cours de sa prochaine mise à jour.

Les loteries et les jeux de hasard sont tolérés dans une certaine mesure, notamment lorsque le développeur exerce son activité dans le cadre que prévoit la loi. Là encore, vous n'avez le droit de vendre de telles applications que dans les pays qui les autorisent. En France, la loi est très stricte à ce niveau et vous oblige à disposer d'une patente spéciale. Apple y veille scrupuleusement et vous êtes susceptible de lui adresser des documents complémentaires pour prouver le cadre de votre activité.

INFO

La documentation des développeurs d'Apple ne contient pas la moindre information sur les éléments autorisés ou interdits. Reportez-vous aux guides relatifs à la soumission d'application auprès de l'App Store, à l'adresse **developer.apple.com**, et à votre accord de licence de développeur iOS, pour de plus amples détails à ce sujet.

Repenser votre business model

Tous les éléments que vous comptez vendre à travers les achats In App (en particulier le contenu) doivent passer par le processus de validation d'Apple, ce qui prend généralement une à deux semaines. Intégrez ce délai à votre planning lorsqu'il constitue un aspect essentiel de votre business model.

Si vous développez une application qui propose de télécharger des thèmes d'arrière-plan, vous ne serez *a priori* pas en mesure de vendre un "thème du jour" à travers les achats In App – en tout cas, pas facilement. Vous devez alors envisager d'autres modèles économiques, comme la possibilité d'offrir un téléchargement lorsque l'utilisateur souscrit à un abonnement premium. Une autre solution consiste à soumettre d'emblée les "thèmes du jour" pour le mois à venir, afin de contrôler leur diffusion aux dates correctes. Assurez-vous de disposer d'une collection suffisamment riche pour ne pas connaître le moindre temps mort, en considérant les durées de validation les plus pessimistes possible.

Chaque produit que vous soumettez à l'App Store doit être configuré avec iTunes Connect. Cette étape prend de quelques minutes à une heure, en fonction de la complexité des captures d'écran que vous devez inclure. Si vous commercialisez une grande quantité de livres ou de contenu numériques, comme des thèmes d'arrière-plan, il n'est pas toujours possible de consacrer autant de temps à cette opération. Par ailleurs, la limite maximale du nombre d'unités que vous pouvez commercialiser est fixée à 5 000 exemplaires. Une solution alternative consiste alors à les considérer comme des produits consommables.

À ce stade, vous devez avoir choisi le type de produit que vous vendez à travers In App, qu'il s'agisse d'un élément consommable, impérissable ou proposé sous forme d'abonnement.

Configurer vos produits sur iTunes Connect

Intégrer les achats In App à votre application nécessite 20 % de configuration, 10 % de choix du bon business model et 70 % d'implémentation. Avec MKStoreKit, le développement pur avoisine les zéros. Toutefois, l'apparition de nouveaux types de produits a rendu l'étape de configuration plus complexe, et les changements de règles ainsi que l'absence de documentation claire sur les éléments autorisés ou non rendent plus difficile le choix d'un bon business model. La configuration reste ainsi l'aspect le plus difficile dans l'intégration de ce système.

Au cours de cette section, vous découvrirez comment définir correctement vos produits sur iTunes Connect. Nous partons ici du principe que vous avez déjà souscrit à une licence de développeur iOS et que vous disposez des autorisations nécessaires pour vous authentifier auprès des différents portails web, en particulier le programme des développeurs et iTunes Connect.

INFO

Vous devrez régulièrement vous connecter au portail du programme des développeurs iOS et à iTunes Connect au cours des pages qui suivent. Retenez leur adresse respective :

Portail du programme des développeurs iOS – **http://developer.apple.com/devcenter/ios/index.action**

iTunes Connect – **http://itunesconnect.apple.com**

Étape 1 : Créer un nouvel App ID pour votre application

Chaque application qui intègre les achats In App doit disposer d'un App ID unique. Cet ID ne peut pas comprendre de caractère joker (`*`). La convention établie consiste à adopter la notation en DNS inversé. En voici quelques exemples :

```
com.masociete.monapp.packniveau1
org.masociete.monapp.packniveau2
```

Pour créer un nouvel App ID, authentifiez-vous auprès du portail du programme des développeurs iOS et rendez-vous dans le portail de provisionnement iOS (*iOS provisioning portal*), comme le montre la Figure 14-1.

Cliquez sur le bouton `New App ID` et suivez les étapes de l'assistant afin de créer un App ID. Assurez-vous d'utiliser un identifiant pleinement qualifié, qui ne présente pas le moindre caractère joker.

ATTENTION

Si vous utilisez un caractère joker dans votre App ID, vous n'aurez pas la possibilité de commercialiser des achats In App par la suite. Si votre application figure déjà dans l'App Store et que vous désiriez intégrer les achats In App dans sa prochaine version, mais qu'elle n'utilise pas un App ID unique, tout fonctionnera correctement mais vos

utilisateurs ne seront pas automatiquement notifiés de la mise à jour à travers l'App Store : ils devront télécharger l'application à nouveau. Pour présenter les notifications de mise à jour, l'App Store implique qu'elles utilisent le même App ID, en incrémentant leur numéro de version. Même s'il est possible d'associer un nouvel App ID à un Bundle ID déjà existant (Apple Technical Note QA1680), nous vous conseillons d'employer un identifiant unique pour chaque application que vous développez. Avec iOS 5, d'autres fonctionnalités nécessitent d'utiliser un App ID unique, en particulier l'intégration de Game Center et d'iCloud. Si votre application risque d'utiliser ces types de fonctionnalités, notamment dans une future version, vous avez intérêt à leur associer un App ID unique dès le départ.

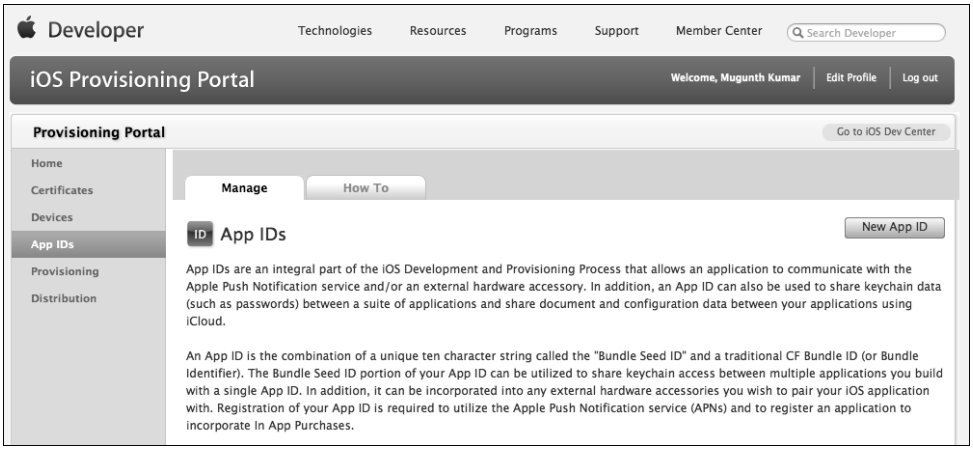

Figure 14-1
Le portail de provisionnement iOS.

Étape 2 : Générer des profils de provisionnement

Vous ne pouvez pas exécuter les achats In App à travers le simulateur ; vous devez disposer d'un appareil réel pour les tester. Vous avez donc intérêt à créer un profil de provisionnement afin d'exécuter votre application sur un vrai appareil. Il s'agit de la seconde étape : suivez le cinquième lien figurant dans le volet gauche du portail de provisionnement iOS. Rien d'inhabituel ici, effectuez la même opération qu'avec n'importe quelle application. N'oubliez pas de renseigner l'App ID créé au cours de la première étape.

Étape 3 : Ajouter votre application

Avant de créer des produits In App, vous devez disposer d'une application capable de les commercialiser. Commençons par créer une nouvelle application dans iTunes Connect. Accédez au site et cliquez sur *Manage Your Applications*. Il s'agit du premier lien sur la colonne de droite, comme le montre la Figure 14-2.

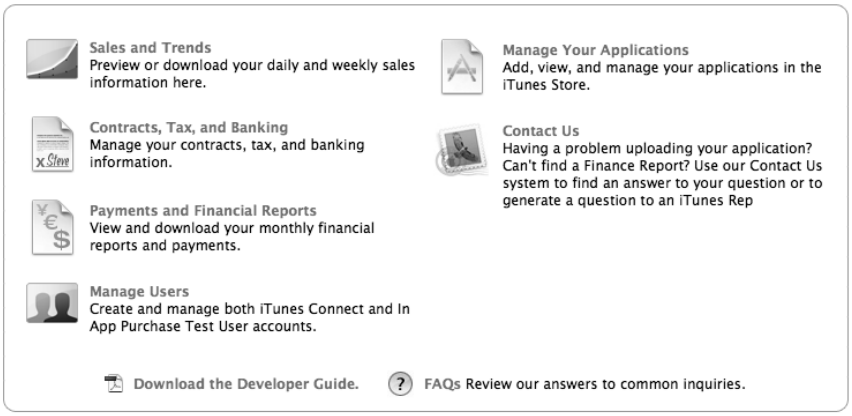

Figure 14-2
La page d'accueil d'iTunes Connect.

Créez une nouvelle application en suivant ce lien. Il est particulièrement important de choisir correctement le Bundle ID. Saisissez l'App ID que vous avez créé lors de la première étape en tant que Bundle ID, comme le montre la Figure 14-3.

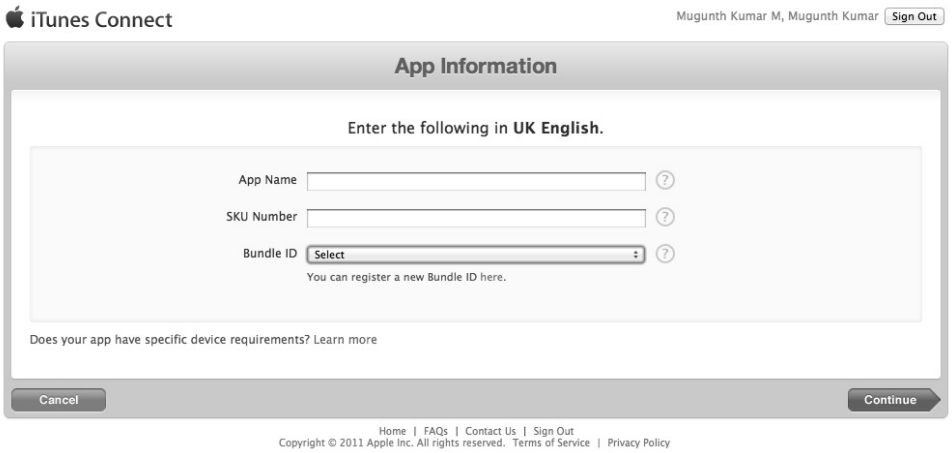

Figure 14-3
Le formulaire de création d'une nouvelle application.

INFO

Le Bundle ID et l'App ID correspondent à la même chose : Apple a choisi des noms différents.

Ultérieurement, vous interrogez l'App Store afin de récupérer des informations à propos des produits que vous commercialisez, notamment leur prix en fonction de la devise de l'utilisateur, leur nom ou leur description localisés. Si vous commettez une erreur à ce stade, vous ne serez pas capable de récupérer des informations détaillées à propos d'un élément In App par la suite. Vous recevrez des identifiants de produit invalides, comme nous allons le voir dans la suite de ce chapitre. Les produits que vous commercialisez dans les achats In App sont liés à une application à l'aide du Bundle ID (App ID).

INFO

Une application avec un Bundle ID spécifique ne peut pas commercialiser des produits In App prévus pour une autre application, qui s'appuie sur un Bundle ID différent.

Étape 4 : Créer les entrées des produits In App

Cliquez sur l'application afin d'accéder à la page présentée à la Figure 14-4.

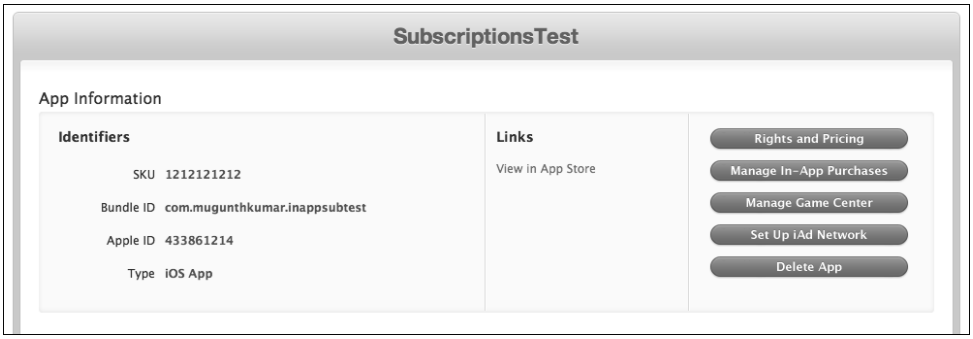

Figure 14-4
Les informations associées à une application.

Cliquez sur le bouton *Manage In-App Purchases* pour créer votre premier produit In App.

INFO

Si vous ne voyez pas l'option *Manage In-App Purchases* sur iTunes Connect, assurez-vous de disposer des privilèges d'administrateur dans votre compte. Ensuite, vérifiez que les détails de votre contrat de taxe et d'informations bancaires sont bien corrects. S'il s'agit de votre toute première application et que vous n'ayez pas encore soumis ce contrat, ce bouton n'apparaîtra pas sur la page. Dans ce cas, corrigez la situation et patientez : Apple prend généralement une semaine ou deux pour approuver vos documents, en fonction de votre localisation.

Sur la première page, vous parcourez les quatre types différents de produits que vous êtes en mesure de créer. Faites votre choix, en prenant en compte les explications figurant au début de ce chapitre.

INFO

Pour créer le Product ID des éléments que vous commercialisez à travers In App, nous vous conseillons d'ajouter un suffixe à votre Bundle ID. Par exemple, si votre Bundle ID est `com.masociete.monapp`, le Product ID des éléments In App pourrait être `com.masociete.monapp.inapp`. Vous vous assurez ainsi que les Product ID de vos autres applications n'entrent pas en conflit.

Consommables, non-consommables et abonnements non renouvelables

Pour tous les types de produits sauf les abonnements autorenouvelables, vous devez saisir un identifiant de produit et choisir un niveau de prix. Ajoutez une description qui apparaît lorsque l'utilisateur achète votre produit. Si votre produit est multilingue, ajoutez des descriptions dans toutes les langues supportées.

Vous devez à présent ajouter une capture d'écran du produit avant de soumettre le formulaire. À l'heure actuelle, vous pouvez envoyer une capture de 320 × 480 pixels. Cette capture n'est utilisée que par l'équipe de validation de l'App Store. Dans la plupart des cas, vous n'en avez besoin que pour les éléments de contenu. Pour les fonctionnalités ou les autres types de produits consommables, vous avez parfaitement la possibilité d'envoyer une capture présentant l'App Store.

INFO

Il n'est plus conseillé de proposer des abonnements non renouvelables depuis qu'Apple a lancé les abonnements autorenouvelables. D'un point de vue fonctionnel, les abonnements autorenouvelables offrent les mêmes services que les abonnements non renouvelables, tout en supportant le renouvellement automatique sans l'intervention de l'utilisateur et en déployant ses abonnements sur tous ses appareils.

Abonnements autorenouvelables

Les abonnements autorenouvelables sont légèrement différents. Vous créez une famille d'abonnements et vous indiquez leur durée. Vous avez ainsi la possibilité de décliner le même abonnement à un magazine à différentes durées. Par exemple, vous pouvez créer un abonnement hebdomadaire à cinq euros, mensuel à vingt euros et annuel à trois cents euros. Une famille d'abonnements dispose ensuite des mêmes options que les produits consommables.

Étape 5 : Générer le secret partagé

Pour les abonnements autorenouvelables, vous devez vous consacrer à une étape supplémentaire très importante : générer un secret partagé.

Dans la page d'information de l'application (présentée à la Figure 14-4), cliquez sur le lien *Manage In App Purchases*. Sur cette page, repérez le lien "View or generate a shared secret". Copiez le secret partagé : vous en aurez besoin dans le développement de votre code.

Étape 6 : Créer un compte d'utilisateur de test

La dernière étape consiste à créer des comptes d'utilisateur que vous emploierez pour tester vos achats In App après leur implémentation. Vous pouvez éventuellement vous y consacrer ultérieurement, mais prenez le temps de le faire dans la foulée afin d'achever la configuration des achats In App.

Pour créer des comptes d'utilisateur, ouvrez la page d'accueil iTunes Connect (reportez-vous à la Figure 14-2) et cliquez sur Manage Users. Deux liens apparaissent à l'écran : un premier pour créer un utilisateur iTunes Connect et un second pour créer un utilisateur de test. Suivez le second lien et créez un compte. L'opération ne devrait pas poser de problème.

Vous avez à présent terminé la configuration des achats In App. Si tout est correct, vous avez complété 30 % du processus. Il ne vous reste plus qu'à développer le code d'implémentation, une étape à laquelle nous allons maintenant nous consacrer.

Implémentation des achats In App

Si la configuration des achats In App n'est pas aisée, les étapes de programmation nécessaires à son implémentation le sont tout aussi peu et impliquent du code plutôt complexe. Voici une liste complète des tâches les plus importantes que vous devez effectuer pour intégrer les achats In App à votre application :

- Tout d'abord, sachez que votre application pourrait se fermer (notamment par un appel téléphonique) au moment de la transaction. Comme les transactions peuvent se poursuivre au-delà de votre application, vous devez disposer d'une classe Store Observer qui s'initialise au lancement de l'application et qui reçoit les achats effectués lorsqu'elle s'exécute à l'arrière-plan.
- Enregistrez le nombre d'éléments consommables achetés.
- Enregistrez l'achat d'un produit non consommable et permettez à l'utilisateur de le restaurer sur ses autres appareils.
- Vous devez enregistrer ces achats de manière sécurisée, à l'aide de la chaîne de clé iOS.
- Envoyez les abonnements non renouvelables à votre serveur et "stockez-les" dessus. Vous devez prévoir la possibilité de les restaurer sur n'importe quel appareil lorsque l'utilisateur s'authentifie auprès de votre application.

- Vous pouvez éventuellement distribuer numériquement le contenu périssable depuis votre serveur. Dans ce cas, votre serveur doit être capable de vérifier l'authenticité du reçu et de délivrer le contenu. Cela signifie également que votre application iOS doit envoyer des reçus d'achat à votre serveur, avant de télécharger le contenu.

- Vérifiez que les reçus des abonnements autorenouvelables sont toujours valides. Même si ce type d'abonnement se renouvelle automatiquement sans l'intervention de l'utilisateur, vous devez toujours le vérifier car un utilisateur a la possibilité d'annuler son abonnement. Celui-ci reste valide jusqu'à la fin de la période prévue. En tant que développeur, vous devez ainsi garder la trace des achats, de leur reçu et vérifier leur validité, *a priori* lors du lancement de l'application. Si les abonnements ne sont plus valides, cessez d'en afficher le contenu.

- Pour valider les abonnements autorenouvelables, envoyez votre reçu et le secret partagé que vous avez généré à l'étape 5 vers l'App Store, puis parcourez l'objet JSON (*JavaScript Object Notation*) retourné et récupérez la dernière date d'achat de l'abonnement.

- `StoreKit.framework` ne vous indique pas la période de fin d'un abonnement autorenouvelable. L'API vous retourne plutôt la date du dernier reçu. À partir de cette date, vous pouvez calculer la véritable date d'expiration.

- Affichez les prix et les descriptions localisés de vos produits, sur le contrôle de vue de votre Store.

Si toutes ces étapes vous paraissent complexes, sachez que MKStoreKit encapsule la plupart de ces fonctionnalités. Voyons à présent comment intégrer MKStoreKit au sein de votre application.

À la découverte de MKStoreKit

MKStoreKit est un framework open-source (MKBlog, *Tutorial des achats In App*) qui simplifie considérablement l'intégration des achats In App à votre application. À l'heure où nous mettons sous presse, la version 4.0 est la dernière en date et nous l'utiliserons à travers ce chapitre. Vous pouvez télécharger le code source sur le site officiel de Pearson ou sur Github (reportez-vous à la section "Pour aller plus loin", à la fin de ce chapitre).

Pourquoi MKStoreKit ?

MKStoreKit prend automatiquement en charge les opérations suivantes que nous avons étudiées jusqu'à présent :

- Lorsque votre application se ferme au cours d'une transaction, MKStoreKit le détecte automatiquement, continue d'observer les délégués de StoreKit et retient tous les achats effectués en dehors de votre application.
- Pour les produits consommables ou non consommables, MKStoreKit retient automatiquement les achats effectués dans la chaîne de clé iOS. Ces achats sont enregistrés avec les identifiants des produits en guise de clé, et il est donc inutile de personnaliser quoi que ce soit dans la plupart des cas.
- La classe `MKSKProduct` envoie les abonnements non renouvelables vers votre serveur. MKStoreKit s'accompagne d'un exemple de code PHP que vous pouvez déployer en l'état sur votre serveur web. Nous y reviendrons à la section "Personnaliser MKStoreKit", dans la suite de ce chapitre.
- Pour les consommables ou tout type de contenu que vous distribuez numériquement depuis votre serveur, vous devez poster le reçu de l'App Store vers votre serveur. Celui-ci doit alors le vérifier et le valider. S'il est bien valide, il distribue le contenu vers votre application en conséquence. La classe `MKSKProduct` prend en charge cette opération, en tandem avec du code côté serveur, sur lequel nous reviendrons ultérieurement.
- Pour les abonnements autorenouvelables, vous devez vérifier le dernier reçu au lancement de votre application et désactiver l'accès au contenu si l'utilisateur a annulé son abonnement. La classe `MKSKSubscriptionProduct` s'en occupe automatiquement, en se chargeant notamment de parcourir la réponse JSON de l'App Store et de vérifier que les abonnements sont toujours valides : elle vous le notifie à travers `NSNotificationCenter`. Vous devez toutefois indiquer à MKStoreKit la durée de validité d'un abonnement, en jours.
- MKStoreKit dispose de méthodes assistant qui retournent les descriptions et les prix de votre produit, formatés selon les préférences locales de l'utilisateur.

Vous pouvez ainsi personnaliser MKStoreKit afin de l'intégrer à votre application en moins de vingt-cinq lignes de code. Parfois, l'implémentation occupe même moins de dix lignes de code.

Mais, avant d'intégrer le framework, intéressez-vous à son fonctionnement interne. Nous allons à présent le détailler.

Fonctionnement de MKStoreKit

MKStoreKit utilise des blocs et non des délégués pour vous notifier les achats de produits. (Vous découvrirez les blocs plus en détail au Chapitre 15.) D'autres notifications, comme l'expiration des abonnements, sont envoyées vers `NSNotificationCenter`. Le framework s'articule autour d'une classe singleton principale, le `MKStoreManager`, et une série de classes complémentaires dont voici la liste :

- **MKStoreManager.h et MKStoreManager.m.** Il s'agit de la classe principale qui gère la majeure partie de l'implémentation. Vous devez initialiser ce singleton dans la méthode `applicationDidFinishLaunchingWithOptions:` de votre AppDelegate.
- **MKStoreObserver.h et MKStoreObserver.m.** Il s'agit de la classe qui implémente l'objet `SKPaymentTransactionObserver` de StoreKit.framework. Elle notifie l'achat de produits ou restaure les notifications vers le singleton MKStoreManager.
- **MKSKProduct.h et MKSKProduct.m.** Il s'agit d'une classe interne utilisée par MKStoreKit afin de valider les achats. `MKStoreManager` l'utilise pour communiquer avec votre serveur afin de vérifier si le reçu est valide et si le produit courant peut être téléchargé. On l'utilise uniquement dans le cas où un serveur vérifie les reçus et distribue numériquement le contenu.
- **MKSKSubscriptionProduct.h et MKSKSubscriptionProduct.m.** Il s'agit d'une autre classe interne utilisée par MKStoreKit pour valider les achats d'abonnement autorenouvelable. `MKStoreManager` utilise cette classe pour communiquer avec votre serveur afin de s'assurer que les reçus du dernier abonnement sont toujours valides. Si l'utilisateur a annulé son abonnement, cette classe notifie `MKStoreManager` qu'il n'est plus valide et `MKStoreManager` envoie une notification. Vous devez observer ces notifications (voir ci-dessous) dans votre contrôleur de vue et activer ou désactiver les boutons *S'abonner* en conséquence. Si votre application n'utilise pas d'abonnements autorenouvelables, vous n'avez pas besoin de le faire.

  ```
  kSubscriptionsPurchasedNotification
  kSubscriptionsInvalidNotification
  ```

- **MKStoreKitConfigs.h et MKStoreKitConfigs.plist.** Ces deux fichiers du framework vous permettent de le personnaliser en fonction de vos besoins. Nous y reviendrons dans la suite de ce chapitre.
- **Enregistrer les achats.** MKStoreKit utilise une chaîne de clé iOS pour conserver automatiquement un achat lorsqu'il est effectué.

Personnaliser MKStoreKit

Vous devez modifier deux fichiers importants de MKStoreKit :

```
MKStoreKitConfigs.h
MKStoreKitConfigs.plist
```

Le fichier plist contient la liste des produits que vous avez configurés sur iTunes Connect. Vous ajoutez vos produits sous les clés correspondantes du fichier plist, en fonction de leur type. Respectez la terminologie anglaise d'Apple : ajoutez vos consommables sous la clé `consumable`, les éléments impérissables sous

`non-consumable` et les abonnements sous `subscriptions`. Chaque type de clé dispose de ses propres sous-options. Vous apprendrez à les configurer dans la suite de ce chapitre.

Initialiser MKStoreKit

Avant de configurer MKStoreKit, initialisez-le dans la méthode `applicationDidFinishLaunchingWithOptions:` de votre AppDelegate. Vous vous assurez ainsi que les objets `StoreObservers` sont correctement initialisés afin de recevoir les transactions gérées en dehors de votre application. Vous initialisez le singleton en appelant le code suivant dans le délégué de votre application :

```
[MKStoreManager sharedManager];
```

Configurer les échanges avec un serveur

Lorsque vous vendez dans votre application du contenu distribué à partir de votre serveur, vous devez épouser un modèle de distribution spécifique que MKStoreKit intitule le *Server Product Model* (le modèle produit serveur). Dans ce modèle, l'application iOS gère l'achat et envoie le reçu de la transaction au serveur pour le vérifier. Le serveur vérifie alors le reçu auprès du serveur de vérification d'Apple et, s'il est bien valide, il redirige l'utilisateur vers le contenu désiré.

Configuration du serveur

MKStoreKit s'accompagne d'un code serveur développé en PHP afin de vérifier les reçus. Copiez le dossier Server Code et ouvrez son accès. Copiez ensuite l'URL publique de ce dossier. Imaginons qu'il est accessible à l'adresse :

```
http://api.exemple.com/servercode
```

Revenez ensuite au code source de votre application iOS, ouvrez le fichier `MKStoreKitConfigs.h` et repérez les lignes suivantes :

```
#define SERVER_PRODUCT_MODEL 0
#define OWN_SERVER nil
#define REVIEW_ALLOWED 1
```

Assignez `OWN_SERVER` à la valeur `@"http://api.exemple.com/servercode"`.

Et voilà. MKStoreKit communique avec le script featureCheck.php de ce dossier afin de vérifier les reçus et d'enregistrer les achats lorsque le serveur indique qu'ils sont valides. La validation d'un reçu s'effectue en le postant auprès de l'URL de validation d'Apple :

```
https://buy.itunes.apple.com/verifyReceipt
```

Pour tester ce fonctionnement au sein de la sandbox, vous devez utiliser

```
https://sandbox.itunes.apple.com/verifyReceipt
```

Le code du serveur bascule automatiquement vers cette adresse en fonction de la configuration que vous avez choisie. L'implémentation par défaut de featureCheck.php retourne des chaînes brutes – `YES` ou `NO` – en fonction de la validité des reçus. Vous pouvez éventuellement le modifier afin de retourner des objets JSON avec l'URL du contenu correspondant.

Configurer les achats de consommables

De manière générique, un consommable correspond à un produit qui s'épuise au fur et à mesure de son utilisation. L'encre d'une imprimante en constitue un exemple courant. Avec les achats In App, les consommables se comportent de la même manière : lorsqu'un utilisateur achète ce type de produit, il est stocké sur l'appareil et reste ainsi jusqu'à ce qu'on l'utilise. Vous n'êtes pas obligé de restaurer les consommables sur d'autres appareils. Lorsque l'on implémente des consommables, il arrive bien souvent qu'un même produit se décline en des quantités différentes, comme dans la vie réelle.

Par exemple, vous pouvez proposer deux produits au sein de votre jeu : une petite caisse de munitions comportant cent cartouches pour 0,99 euro et une grande caisse en contenant mille pour 5,99 euros. Au sein de votre jeu, les deux produits sont synonymes. En revanche, du côté du nombre d'unités, ils sont traités différemment et correspondent à des prix distincts. Pour implémenter ce modèle, MKStoreKit vous permet d'associer des noms différents à vos produits. De cette manière, vous pouvez indiquer à MKStoreKit de les traiter de manière séparée lors de l'achat, mais de les considérer de la même façon lorsqu'on les utilise. En clair, l'achat de l'un ou l'autre de ces produits incrémente le nombre d'éléments du même objet. Pour configurer MKStoreKit de cette manière, servez-vous des sous-options figurant sous la clé *Consumables*, dans MKStoreKitConfigs.plist, comme le montre la Figure 14-5.

Figure 14-5
Configuration des consommables dans MKStoreKit.

Vous ajoutez l'identifiant de produit de chaque consommable au sein du dictionnaire Consumables. Chaque produit s'identifie par son nom (`Name`) et son nombre d'unités (`Count`). En reprenant l'exemple précédent (la caisse de munitions), vous pouvez imaginer deux produits : `com.monapp.monjeu.petiteCaisse` et `com.monapp.monjeu.grosseCaisse`. La clé `Count` vous permet d'assigner le nombre d'unités associé à chaque version de ce produit. Vous définissez le nom du consommable dans la clé `Name`. Usuellement, MKStoreKit enregistre les achats en fonction de leur identifiant de produit. Mais, pour les consommables, il s'appuie sur la clé `Name` dans la mesure où un même produit peut se décliner en un nombre d'unités différentes au sein de votre application.

À présent, au cours de l'exécution de votre jeu/application, si l'utilisateur consomme votre produit, vous devez tout d'abord vérifier s'il est toujours disponible en appelant ces méthodes :

```
- (BOOL) canConsumeProduct:(NSString*) productName quantity:(int) quantity;
- (BOOL) consumeProduct:(NSString*) productName quantity:(int) quantity;
```

`consumeProduct` déduit automatiquement la quantité de produits consommée de la quantité achetée, en stockant le nombre d'unités restantes dans la chaîne de clé.

Configurer les abonnements autorenouvelables

La configuration des abonnements autorenouvelables s'apparente à celle des consommables. La première étape consiste à mentionner votre secret partagé. Copiez la valeur que vous avez générée précédemment dans ce chapitre et collez-la dans le fichier `MKStoreKitConfigs.h` :

```
#warning Shared Secret Missing Ignore this warning
# if you don't use autorenewable subscriptions
#define kSharedSecret @"<FILL IN YOUR SHARED SECRET HERE>"
```

Vous pouvez à présent supprimer la ligne `#warning`. Si vous n'utilisez pas d'abonnement autorenouvelable (et, donc, si vous n'avez pas de secret partagé), vous pouvez ignorer cet avertissement.

Ouvrez ensuite le fichier plist et, comme vous l'aviez fait avec les consommables, indiquez la durée de l'abonnement au lieu d'en préciser le nombre. Maintenant, sur vos contrôleurs de vue qui affichent la boutique, vous devez observer les notifications `kSubscriptionsPurchasedNotification` et/ou `kSubscriptionsInvalidNotification`, en activant ou désactivant vos boutons d'abonnement en conséquence.

Traiter l'achat

Maintenant que MKStoreKit est pleinement configuré, il est très simple de traiter un achat réel. Il vous suffit d'appeler une simple méthode, comme ici :

```
[[MKStoreManager sharedManager] buyFeature:@"com.monapp.mafonctionnalite"
   onComplete:^(NSString*) produitAchete
   {
      // On délivre le contenu pour le produit "produitAchete".
   }
   onCancelled:^
   {
      // on affiche éventuellement une erreur
}];
```

Après avoir configuré correctement MKStoreKit, il vous suffit donc d'un simple appel de méthode pour initialiser un achat. La sauvegarde de l'achat s'effectue

automatiquement à votre place. Pour vérifier si un achat a été effectué précédemment, vous appelez cette méthode :

```
[MKStoreManager isProductPurchased:@"com.monapp.fonctionnalite1"];
```

Cette méthode retourne un booléen qui indique si le produit a été acheté ou non. On restaure les achats en une seule ligne également. Parcourez le fichier `MKStoreManager.h` pour découvrir toutes les fonctionnalités de MKStoreKit.

En moins de dix lignes de code pour fonctionner, et avec une quinzaine de lignes de configuration, MKStoreKit gère toutes ces opérations très simplement.

Tester les achats In App

Maintenant que vous avez implémenté les achats In App au sein de votre application, vérifiez si tout fonctionne correctement. Vous avez besoin des identifiants de votre compte d'utilisateur de test que vous avez créé précédemment. Avant de commencer, accédez aux Réglages de l'appareil de test et sélectionnez le menu Store. Déconnectez-vous de l'App Store et assurez-vous qu'aucun utilisateur n'est identifié.

Exécutez l'application sur l'appareil et initiez un achat. On vous demande alors de saisir vos identifiants ou de créer un nouveau compte d'utilisateur. Choisissez la première option et entrez le nom d'utilisateur et le mot de passe définis précédemment. L'App Store vous demande de confirmer l'achat In App : validez. Vous vous assurez ainsi que tout fonctionne correctement et que l'intégration de ce système au sein de votre application est pleinement terminée.

Mais que se passe-t-il lorsque vous rencontrez un problème ? Passons à présent en revue les principaux écueils et leurs solutions.

Dépannage

Même après toutes les explications de ce chapitre, les achats In App demeurent l'un des frameworks les plus difficiles à dépanner.

Identifiants de produits invalides

L'erreur la plus courante se produit lorsque l'App Store considère que votre produit est invalide. Si vous avez minutieusement suivi les conseils prodigués au cours de ce chapitre, en lisant les astuces et les avertissements, une telle situation ne devrait pas arriver. Mais, si c'est le cas, vous rencontrez l'un des problèmes suivants :

- Le Bundle ID renseigné dans le fichier Info.plist ne correspond pas à l'App ID que vous avez créé.

- Vous n'avez pas encore soumis votre contrat sur les taxes et le paiement des applications iOS ou il n'a pas encore été accepté. Pour le corriger, rendez-vous sur iTunes Connect et cliquez sur le lien *Contracts, Tax and Banking*. Il est très important de vérifier que cet aspect est réglé, lorsque vous soumettez votre première application.
- Les appareils "jailbreakés" ne fonctionnent pas toujours correctement avec l'App Store. L'application AppSync diffusée à travers Cydia est souvent la cause de problèmes relatifs aux achats In App. Il vaut donc mieux les tester sur un appareil qui exécute une version non modifiée du système d'exploitation.

Parfois, même si vous vous êtes assuré qu'aucune de ces raisons n'est en cause, l'App Store persiste à vous indiquer que vos produits ne sont pas valides. D'après notre propre expérience, c'est souvent une désynchronisation des serveurs de distribution d'Apple qui est en cause. Apple s'appuie en effet sur des équipements très divers, disséminés à travers le monde : les miroirs locaux doivent refléter l'état des serveurs américains principaux. Patientez pendant plusieurs heures et testez à nouveau votre application.

Connexion impossible à iTunes Store

L'autre problème courant concerne le message "Connexion impossible à iTunes Store : Code: -1003". Une telle situation se produit lorsqu'un pare-feu bloque iTunes. Testez les achats In App en vous connectant à un autre réseau ou vérifiez votre connexion Internet.

Vous avez déjà acheté ce produit, mais il n'est pas téléchargé

Cette erreur se produit souvent avec les consommables, en particulier lorsque vous appuyez trop souvent sur le bouton Acheter. L'astuce consiste à désactiver ce bouton une fois l'achat effectué et à l'afficher à nouveau une fois que la transaction a abouti. Essayez de reproduire le processus de l'App Store au sein de votre application.

Si vous ne parvenez décidément pas à résoudre votre problème, essayez de supprimer l'application et de la réinstaller à nouveau.

En résumé

Les achats In App, bien qu'ils soient assez délicats à implémenter, constituent une solution innovante et originale pour monétiser vos applications. Choisir minutieusement votre business model et intégrer au mieux ce système peut largement contribuer à doper vos profits sur l'App Store. Il vous suffit d'observer le classement des applications les plus populaires du moment : près d'un quart d'entre elles épousent le modèle des *freemiums*, ces applications gratuites dont les fonctionnalités les plus avancées font l'objet d'un paiement. Ce modèle est donc particulièrement en

vogue et son succès ne devrait pas s'estomper dans les prochaines années. Puisque des frameworks comme MKStoreKit facilitent son déploiement, pourquoi ne pas l'essayer à votre tour ?

Pour aller plus loin

Documentation Apple

Les documents suivants sont disponibles dans la bibliothèque des développeurs iOS, à l'adresse **developer.apple.com**, ou à travers la documentation de Xcode et la table de référence de l'API.

- *Guide de soumission d'applications à l'App Store*
- *Guide de programmation des achats In App*
- *Récupérer les informations du Store*

Blogs

- iPhone Tutorial: In App Purchases | MKBlog.

  http://blog.mugunthkumar.com/coding/iphone-tutorial-–-in-app-purchases/

- MKStoreKit 4.0 – Supporting Auto Renewable Subscriptions | MKBlog.

  http://blog.mugunthkumar.com/coding/mkstorekit-4-0-supporting-autorenewable-subscriptions/

Autres ressources

- MKStoreKit sur Github.

  https://github.com/MugunthKumar/MKStoreKit

- Can't use in-app purchase: This is not a test user account / Boxcar support.

  http://help.boxcar.io/kb/general/cant-use-in-app-purchase-this-is-not-atest-user-account